Università degli Studi di Ferrara

Doctoral School of Graduate Studies
in Mathematic and Computer Sciences
– XXIII Edition –

Parallel Large-Scale Edge-Preserving Joint Inversion with PETSc and TAO

by Dr. Ambra Giovannini

Advisor: Prof. Gaetano Zanghirati

Doctoral Thesis

2011

Abstract

The focus of this thesis is on the study and the parallel implementation of a software package for the Tikhonov's approach to the joint inversion of multidimensional data, based on minimum support regularization and built on top of the well known and widely used high-performance parallel libraries PETSc and TAO.

Studying effective methods and implementing efficient codes for a truly joint inversion arouses great interest, because multiple types of observations of the same object can be used at once in a single procedure to recover an estimate of the object itself via non-invasive inspection. One of the main reasons for this interest is that jointly inverting different kind of data could allow to reduce both the ill-posedness of data reconstruction problem and the total number of data to be collected, while still preserving the accuracy of the results.

This is a relevant goal because the need to get more detailed information about the structure of the investigated physical systems, via non-invasive observations, is common to a large number of research and industrial fields such as Biology, Geophysics, Medicine and many many others; and the same applies to their large-scale scientific applications.

Original contributions are given to both the theoretical aspects and the numerical implementation of the joint inversion. First of all, we derive the analytical expressions of first- and second-order derivatives of the proposed joint inversion functional in its discretized version, to be used with first- and second-order optimization methods. Then we implemented a Matlab prototype and the HPC parallel code, named JoInv.

Another important part of the thesis is the design and the implementation of a PETSc-compliant recursive procedure for the structure prediction of sparse matrices products of every dimension, in both the sequential and the parallel cases.

Last, but not least, we provide the implementation of the well known Scaled Gradient Projection (SGP) method for simply constrained nonlinear programming problems as a new TAO solver.

We analyze the performances of all the developed code with respect to the most important metrics, *i.e.*, speedup, efficiency, Kuck's function, and DUSD model, by using simulated data.

A number of further research developments are outlined at the end, with the perspective in mind of using the code with large-scale real-world data that will be hopefully made available by interested people.

Contents

Introduction

Motivations

In Applied Sciences, the need for more detailed information about the structure of the investigated physical systems (no matter if they are the internal structure of the Earth, the Universe, the quarks distribution inside the hadrons or the human body) has forced the researcher to simultaneously collect various kind of *non-invasive* observational data. For instance, in Geophysics it is common to acquire magnetic, gravity or seismic data; in Medicine, MRI[1], PET[1] and CT[1] scans are extensively used in attempting more reliable diagnosis; in Astrophysics, acquisition of the same sky region at different wavelengths are tried to improve the accuracy of stars position reconstructions. As it is well-known, the process of recovering the system' structure from the observed data, called *data inversion*, is mathematically ill-posed. In general, all the multiple data sets acquired are aimed to provide extra information in order to reduce the ill-posedness of data reconstruction. However, until very recently, data inversions are performed separately and a matching of the results is attempted only afterwards, through a number of different techniques, that are most often strongly problem-dependent (*e.g.*, one can obtain the radar waves velocity distribution in a given volume and *then* compare it with the electrical conductivity distribution).

In the *joint inversion* a unique inversion procedure is adopted instead, which uses all the available data at once. Basically, it is possible to distinguish two main procedures to perform the joint inversion: the *non-structural* and the *structural* approaches. To sketch the ideas, let's consider the easiest case where only two properties of the volume of interest are measured and let's call *models* the unknown distributions of these properties. These models are thought as scalar functions of the spatial position $\boldsymbol{x} \in \mathbb{R}^n$, where usually $n = 3$. In some contexts (such as Geophysics) the image in $\mathcal{M} \subset \mathbb{R}$ of each of these functions is called *model space* and we sometimes use this name for easier reading. In the former approach the inversion is performed by using traditional methods, but with the following observation: if an explicit relationship φ is known to exist between the two properties, that is $m^{(2)} = \varphi(m^{(1)})$, then the model space $\mathcal{M}^{(1)}$ of $m^{(1)}$ is mapped onto the model space $\mathcal{M}^{(2)}$ of $m^{(2)}$. For instance, if $m^{(1)} = m^{(1)}(\boldsymbol{x})$ is the velocity distribution and $m^{(2)} = m^{(2)}(\boldsymbol{x})$ is the conductivity distribution of an electromagnetic wave in the soil, then such a φ exists. The key point here is that, if any known relationship exists, it is often empirical (*e.g.*, a strongly nonlinear φ is often roughly approximated with a polynomial as in [83]) and not always it is very reliable. On the other hand, the idea behind the structural approach introduced in [61] is the *geometrical assumption* that both the measured physical

[1]MRI: Magnetic Resonance Imaging. PET: Photon Emission Tomography. CT: Computerized Tomography.

properties tend to change in the same locations, often called *transition zones*. Even if this is not always the case, in many situations one can assume the spatial matching of these transition regions for the different properties, that is the matching of the boundaries separating the sub-volumes where each $m^{(j)}$ has different (almost) constant values. In other words, it is assumed that if any inhomogeneity is present in the investigated system, then it modifies the values of both the physical properties in the same sub-volume, even if not necessarily in the same way (amount and/or versus). When this happens, we say that $m^{(1)}$ and $m^{(2)}$ have the same *structure* and the data sets can be jointly inverted. Here we will follow the structural approach and we will implement a focusing technique for the joint inversion in the case of the reconstruction of blocky targets. The literature on edge-preserving inversion is really huge and successful approaches have been extensively studied and experimented, such as *total variation*, *Mamford-Shah* and *cross-gradient*, just to mention some of the most known. In the next subsections we briefly comment on these approaches and our choice, which is none of them.

Studying effective methods and implementing efficient codes for a *truly joint* inversion arouses great interest. One of the main reasons is that jointly inverting different kind of data could allow to reduce the total number of data to be collected, without losing the accuracy of the results. In fact, data of the same nature are often "dependent", that is to say they bring almost the same information. Thus, one can think to collect fewer data with a smaller burden of information, completing them with richer data of different nature and then invert them jointly. The development of such a technology would have several spin-offs. For instance, in Medical Imaging it would be possible to accurately reconstruct the size and the position of SPECT[2]-detected functional anomalies within anatomic structures, by compensating the spatial low-resolution limitations of SPECT data with the spatial high-resolution of CT data. A true joint inversion would increase the result's informative content with respect to that obtained by separate inversions, because it would provide correlations between anatomy and functionality by a truly integrated processing of the two data sets, in place of just overlapping the separated final images. Another example is in Geophysics, where jointly using the information from compressional-body-waves (P-waves) arrival and the information from surface (Rayleigh) waves allow to construct the most reliable near-surface seismic model to be used for the computation of statics correction, which is important for the seismic reflection analysis.

The main research goal of this thesis is the parallel implementation of a software package for the minimum support Tikhonov approach to joint inversion of 3D data, building it on top of the well known and widely used high-performance parallel libraries PETSc and TAO. This is a relevant goal and a meaningful contribution because these technologies are the today common core of a large number of large-scale scientific applications, challenging scientists in a wide variety of research and industrial fields such as Biology, Nanotechnology, Geophysics, Medicine and many many others. Moreover, as we just mentioned, a large part of these applications relies on the solution of difficult inverse problems. Nevertheless, at the time this thesis is written no such package is available yet in PETSc and TAO libraries, so we hope contributing to fill this gap with a flexible, extensible and well founded package. As it will be seen, both the theoretical formulation and the numerical implementation are quite difficult. Furthermore, the joint inversion we implemented involves eight parameters that have to be chosen by the user, accordingly

[2]SPECT: Single-Photon Emission Computerized Tomography.

with the particular application at hand and the *a priori* information he/she has about the problem solution. It is also well known that there is no single rule for parameter choice which is effective in all situations, even in the simplest case of standard inversion, but a number of methods are extensively used in practice such as F-test, L-curve, generalized cross validation (GCV), discrepancy principle, just to mention some. Beside this fact, the value of each one of these parameter matters a lot: sometimes, even small changes in their values make the difference between recovering a satisfactory model and getting an unsolvable problem. Validating an inversion method always involves reconstruction comparisons. However, this is a different job, which first needs the underlying computational support, and goes beyond the scope of this thesis. This thesis provides the previously unavailable tools to make that further investigation possible.

Each of the six main chapters contains some original contributions. In Chapter 1 we derive the analytical expressions of first- and second-order derivatives of the proposed joint inversion functional in its discretized version.

In Chapter 2 we deeply analyze the sparsity structure of all arrays involved in the actual computations, providing detailed description for both forward and central difference discretization schemes.

In Chapter 3 we design the high-performance computing (HPC) code structure for joint inversion, in both the sequential and the MPI-based parallel environments, based on the PETSc and TAO libraries. Here, detailed information is also given on what is missing there for our purposes and what solutions we found.

In Chapter 4 we design, analyze and implement a PETSc-compliant recursive procedure for the structure prediction of sparse matrices products. It is extremely effective in the sequential settings, but we are also able to maintain a correct recursion in the parallel settings, even if communications due to the PETSc data distribution slightly degrades the performances. To the best of our knowledge, no such code was previously available anywhere, neither in sequential nor in parallel versions.

In Chapter 5 we provide the implementation of the well known *Scaled Gradient Projection* (SGP) method for simply constrained nonlinear programming problems as a new TAO solver. This first-order solver has shown to be very effective in solving classical imaging inverse problems in many fields, where it often outperforms other standard iterative solvers. Its implementation thus widens the TAO set of first-order solvers by adding a new state-of-the-art gradient-based approach.

In Chapter 6 we analyze the performances of the implemented joint inversion code, named JoInv. The tests run on 3D synthetic data. Unfortunately, no sufficiently large real data sets are available to the author at the moment of writing this thesis. However, we hope that people dealing with large-scale real-world data sets will find this work interesting and try the code on their data soon after the publication of this material.

The original parts of this work are still unpublished, but they will hopefully appear soon in upcoming papers.

Background

As it is mentioned by Bertero and Boccacci [12], from the mathematical perspective there is a degree of ambiguity in the concept of inverse problems. According with J.B. Keller [74], we recognize two problems as *inverses* of one another if the mathematical formulation

of each one involves part or the whole solution of the other. Even if, mathematically speaking, the two problems could be generally seen at the same level because they are related by a sort of *duality*, in the sense that one problem can be derived from the other by exchanging the role of the unknowns and the data, from a physical viewpoint the situation is different. In fact, in Physics the *forward problem* (or *direct problem*) is identifying the problem expressing the cause-effect sequence, so that the corresponding *inverse problems* relates to the reversal process, that is finding the unknowns that generated the known consequences [125]. Hence, the physical laws governing the cause-effect sequence are assumed to be known or at least sufficiently well approximated. This work follows this point of view. Another key point in this context is that the physical forward problem generating the data is unavoidably directed towards a loss of information, due to the "transition" from one physical quantity with a certain information content to another physical quantity with a smaller content. Then, even an "exact" solution of the inverse problem can hardly provide the starting point (that is the right cause generating the data), because this would correspond to a *gain* of information. This is the source of the mathematical property called *ill-posedness*, affecting all real-world inverse problems.

We refer the reader for instance to [12, 106] for excellent introductions to the subject and for a list of illuminating examples.

There are more abstract problems that can be mathematically formulated as forward-inverse pairs, but they heavily involve advanced theoretical Mathematics and will not be considered in this work.

In what follows we only recall the basic formal definitions of the mathematical objects we will deal with in the rest of the work.

The forward problem

The forward problem describes how the data are generated by the "observed" object. In the physical context this always implies the use of an *inspection system* and of a corresponding *detection system*. For instance, in electro microscopy the inspection system is the *optical system*, while the detection system is a photomultiplier tube (PMT) or an avalanche photo-diode (APD) [11, 31, 32, 113]. The detection process introduces *sampling* and *noise*, which are the responsible for the loss of information.

The forward problem is generally modeled by a function or an *operator* (that is a mapping between vector spaces) applied to the object m:

$$\mathcal{A}(m) = d.$$

It can be a linear or a nonlinear mapping: if it is linear, then a number of theoretical results are known. If it is nonlinear, as it is often the case in real-world applications, then things are more complicated from both a theoretical and a numerical viewpoints.

The inverse problem

We now recall the mathematical definitions of the concepts briefly outlined before. For formal and much deeper discussions on the subject we refer the reader to classical and recent books such as [12, 35, 36, 56, 65, 76, 78, 98, 119, 124, 134].

Definition 1 *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be Hilbert spaces and let $\mathcal{A} : \mathcal{H}_1 \to \mathcal{H}_2$ be a linear or nonlinear operator. Given $d \in \mathcal{H}_2$, an* inverse problem *is to find $m \in \mathcal{H}_1$ such that $d = \mathcal{A}(m)$.*

The operator $\mathcal{A}$ describes the relationship between the *data* $d \in \mathcal{H}_2$ and the model parameters $m \in \mathcal{H}_1$, and it is a representation of the physical system generating the data.

One well known example of inverse problem is the Fredholm integral equation of the first kind, that in one dimension on the interval $]a, b[\subset \mathbb{R}$ can be written as

$$d(t) = \int_a^b k(t, s)m(s)\mathrm{d}s$$

The two-variable function $k(t, s)$, called the *kernel*, and the data $d(t)$ are given; the goal is to find the model function $m(s)$. Sometimes, the definition of the problem is helpful in defining suitable forms that have known solutions. For instance, in the case of space-invariant kernels (that is $k(t, s) = k(t - s)$) and infinite intervals, the previous equation becomes

$$d(t) = \int_{-\infty}^{+\infty} k(t - s)m(s)\mathrm{d}s = (k * m)(t)$$

which is the convolution product of k and m. In this case an analytic solution based on direct and inverse Fourier transforms of the data d and the kernel k is known. However, this is not always the case and most often one can only try to compute an approximated solution.

The concept of *ill-posedness*, as introduced by Hadamard [62, 63], can be formally stated as follows.

Definition 2 *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be Hilbert spaces and let $\mathcal{A} : \mathcal{H}_1 \to \mathcal{H}_2$ be a linear or nonlinear operator. The operator equation*

$$d = \mathcal{A}(m)$$

is said to be well-posed *provided that $\forall d \in \mathcal{H}_2$:*

1. *there exists $m \in \mathcal{H}_!$, called a solution, such that $d = \mathcal{A}(m)$;*
2. *the solution m is unique;*
3. *the solution m depends continuously on d, that is if $d = \mathcal{A}(m)$ and $d_* = \mathcal{A}(m_*)$, then $d \to d_* \;\Rightarrow\; m \to m_*$. One can also say that the solution m is* stable *with respect to perturbations on d.*

A problem that is not well-posed is said ill-posed.

An equivalent, more compact form of the previous definition is the following[3]:

Definition 3 *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be Hilbert spaces and let $\mathcal{A} : \mathcal{H}_1 \to \mathcal{H}_2$ be a linear or nonlinear operator. The operator equation*

$$d = \mathcal{A}(m)$$

is said to be well-posed *if $\mathcal{A}$ is bijective[4] (that is, if the inverse mapping $\mathcal{A}^{-1} : \mathcal{H}_2 \to \mathcal{H}_1$ exists) and the inverse operator $\mathcal{A}^{-1} : \mathcal{H}_2 \to \mathcal{H}_1$ is continuous.*

[3] Notice that the results are usually given for the more general class of *normed* spaces, However, one can always consider an Hilbert space as a normed space equipped with the norm induced by its inner product.

The nature of ill-posedness is then easily recognized:

- if $\mathcal{A}$ is not surjective, then the equation is not solvable for all $d \in \mathcal{H}_2$ (*nonexistence*);
- if $\mathcal{A}$ is not injective, then the equation may have more than one solution for the same $d \in \mathcal{H}_2$ (*nonuniqueness*);
- if $\mathcal{A}^{-1}$ exists, but it is not continuous, then the solution $m \in \mathcal{H}_1$ of the equation does not depend continuously on the data $d \in \mathcal{H}_2$ (*instability*).

From a mathematical viewpoint, the well-posedness of a problem is a property of the operator $\mathcal{A}$ *together with* the solution space $\mathcal{H}_1$ and the data space $\mathcal{H}_2$, *including* their norms [78].

Most inverse problems are ill-posed. It is worth emphasizing that the previous properties are stated in a continuous setting: in particular, the non-continuous dependence of the solution on the data is a property that clearly requires infinite-dimensional spaces. However, in practice the problems are discrete, obtained by discretization of the spaces and the operator equation. Thus, discrete problems have to be solved, but the very bad mathematical properties of the continuous problems they come from make them extremely *ill-conditioned*. More precisely, the fact that an operator $\mathcal{A}$ does not have a bounded inverse means that the condition numbers of its finite-dimensional approximations grow with the quality of the approximation. Increasing the degree of discretization, *i.e.*, increasing the accuracy of the approximation for the operator $\mathcal{A}$, will cause the approximate solution of the equation $\mathcal{A}(m) = d$ to become less and less reliable. This comes directly from the Picard's Theorem [100], as a consequence of the fact that the continuous operator has singular values decreasing to zero. From a strictly mathematical viewpoint, the discrete problems are well-posed (because whatever will be their size, their smallest singular value is strictly positive); however, due to their very large condition number, we will observe a lack of stability when we start resolving the finite-dimensional problem numerically [3, 35, 65, 78]. This fact in turn implies that standard linear or nonlinear solvers are unable to provide an acceptable answer, because they either do not compute a solution at all, or the solution they compute is meaningless (physically or in any other respect). In this situation, the way to face the problem's ill-posedness and ill-conditioning is to include in the problem formulation additional information, in the form of *constraints* that the desired solution has to satisfy [12]. These constraints come from the physics of the problem and are aimed to compensate for the loss of information that generated the bad mathematical properties. There are two main ways to introduce such an additional information: one is the class of *regularization methods*, where the constraints are explicitly included in the problem formulation, and the other is the class of *Bayesian methods*, where the additional information is of statistical nature. Both of them are extremely powerful tools and today's knowledge on their properties is noticeable with respect to both the theoretical and the computational sides. In this work, we will follow the former approach.

[4] We recall that a map $\mathcal{A} : \mathcal{X} \to \mathcal{Y}$ is said *bijective* if it is *injective and surjective*, that is if $\forall x_1, x_2 \in \mathcal{X}$, $x_1 \neq x_2 \Rightarrow \mathcal{A}(x_1) \neq \mathcal{A}(x_2)$ (injectivity) and $\forall y \in \mathcal{Y}\ \exists x \in \mathcal{X}$ such that $y = \mathcal{A}(x)$ (surjectivity). The two conditions can equivalently be expressed together as $\forall y \in \mathcal{Y}\ \exists! x \in \mathcal{X}$ such that $y = \mathcal{A}(x)$. The surjectivity is also stated as $\mathcal{A}(\mathcal{X}) = \mathcal{Y}$, that is by saying that the *range* (or *image*) of $\mathcal{A}$ is the whole codomain $\mathcal{Y}$.

Regularization

Regularization techniques are aimed to provide acceptable solutions to ill-posed (ill-conditioned) inverse problems, by transforming them into stable problems. Without regularization, the computed approximations are usually dominated by noise and are therefore meaningless or useless.

According with Tikhonov [12,35,65,78,124], the basic idea of regularization consists of considering a *family* of approximated solutions depending on a positive parameter, called the *regularization parameter*. The main property of this family is that, under suitable hypothesis, the approximations sequence converge to the inverse problem solution as the regularization parameter goes to zero, if the data are noise-free. Unfortunately, noise-free data are quite uncommon in practical applications. However, the power of these methods is that even when the data are noisy they can still provide an *optimal approximation* of the problem solution for a nonzero value of the regularization parameter. Most often inverse problems are *deconvolution problems*: for this class, Tikhonov-like regularization techniques are essentially *spectral filtering techniques*, for which we refer the reader to the wide literature available (see for instance [2,3,12,35,50,56,64–66,76,77,93,97,134], just to mention some of the most known references, in an absolutely non-exhaustive list).

From the mathematical viewpoint, given an *injective bounded linear* operator $\mathcal{A}$, a classical regularization method for the operator equation $\mathcal{A}(m) = d$ requires finding an approximation of the *unbounded inverse* operator $\mathcal{A}^{-1} : \mathcal{A}\big(\mathcal{H}_1\big) \to \mathcal{H}_1$ by a family $\big\{R_\lambda\big\}_{\lambda>0}$ of *bounded linear* operators $R_\lambda : \mathcal{H}_2 \to \mathcal{H}_1$ with the property of *pointwise convergence*

$$\lim_{\lambda\to 0} R_\lambda\big(\mathcal{A}\big(m\big)\big) = m \qquad \forall m \in \mathcal{H}_1$$

or, equivalently,

$$R_\lambda(d) \xrightarrow[\lambda\to 0]{} \mathcal{A}^{-1}(d) \qquad \forall d \in \mathcal{A}(\mathcal{H}_1)\,.$$

One needs to assume that $\mathcal{A}(\mathcal{H}_1)$ is dense in $\mathcal{H}_2$. For noisy data $d_\eta = d + \boldsymbol{\eta}$ under suitable conditions there exists an optimal value λ^* such that the corresponding computed solution m^* minimizes the regularization function, providing the best solution approximation which is compatible with both the data and the noise. How to find such a parameter value is a matter of fact and all available methods are essentially of heuristic nature, even if some of them are clever and based on theoretical justifications, such as for instance the Morozov's *discrepancy principle* [94,95], the Miller method [92], the generalized cross validation (GCV) and others (see also [12,35,65]). Another key point is how to include the additional information into the regularization method. In the Tikhonov approach, this is done by adding explicit constraints to the operator equation, in the form of a *penalization* term added to a data-fitting measure and/or of direct constraints on the required solution. Summarizing, the classical Tikhonov-like regularized problem has $R_\lambda(d_\eta; \mathcal{A}) = \mathcal{J}_1\big(\mathcal{A}(m), d_\eta\big) + \lambda\mathcal{J}_2(m)$ and can be expressed as a *minimization problem* of the form

$$\underset{m\in\Omega}{\text{minimize}}\ \ \mathcal{J}_1\big(\mathcal{A}(m), d_\eta\big) + \lambda\mathcal{J}_2(m)$$

where $\Omega \subseteq \mathcal{H}_1$, $\mathcal{J}_1$ is a measure of the *data fidelity* and $\mathcal{J}_2$ is the regularizing penalization term. The role of this last term is to impose to the solution some constraints on its size, or its smoothness, or its localization, or a mix of these. The actual numerical solution of such a problem for different regularization parameter values often shows what is known as *semiconvergence*, a behavior that allows to estimate the optimal value λ^* [35,65,134].

In this work, we deal with *non-standard* Tikhonov-like regularization, in the sense that the regularized functional involves more than one data set, coming from different operator equations applied *to the same object*, and also depends on a set of regularization parameters. As will be better explained later on, the idea is the same as that of classical regularization, but the deriving mathematical problem becomes more difficult. At the time this work is written, to the best of our knowledge there is not a well established theory for this approach, even if in the last 15 years some very interesting results have been given and the research on this topic is very appealing and continuously growing. We will not investigate on the convergence properties of the method, as well as on possible strategies for choosing the regularization parameters. Rather, we concentrate on the Computer-Science-oriented task of building a suitable software tool, that will allow the user to experiment this very promising technique and investigate its computational side.

Work structure

This thesis has 6 main chapters and two appendices, structured as follows.

In Chapter 1 we present the new mathematical formulation of a joint inversion problem and develop its derivatives. To simplify notations and reasoning, the two models case only is described and analyzed, but extension of all results to an arbitrary number of models is straightforward. Even if everything is first presented for the general case, the development after discretization is described for linear inverse problems only. The nonlinear case, however, differs only in the misfit parts.

Chapter 2 is a specific discussion about the structure of the matrices involved in JoInv computations; their patterns are important in order to correctly preallocate the memory, so that good performance can be reached.

In Chapter 3 there is a short review of the differences between the sequential and the parallel JoInv code and we highlight the strategies used for an efficient parallel implementation.

Chapter 4 is dedicated to a general discussion about sparse matrix allocation, sequential and parallel, when such a matrix is the result of a multiplication of sparse matrices. Standard graph theory is briefly described and than a new approach that does not need graph theory is presented. The last part of this chapter shows the analysis of the sequential and parallel performance of this new approach. This method enables an efficient memory preallocation, that is essential even when PETSc data structures are used.

Chapter 5 introduce the SGP (Scaled Gradient Projection) method and explains how it has been implemented as a standard TAO solver.

Chapter 6 focuses on the parallel performances evaluation of the JoInv code on a distributed-memory strictly coupled parallel machine. We remind here that we are interested in parallel performances only and completely disregard all reconstruction-related evaluations, which are strongly dependent upon parameter settings.

Finally, two appendices are provided, where relevant information are given to non-expert people. In Appendix A we recall the basic structure, the syntax and the use of MPI, PETSc, and TAO HPC libraries. These technologies are used all over the work. Appendix B gives a review of the most important performance analysis metrics, including some less known very interesting recent proposals.

Test machine system architecture

We tested all the developed software on an IBM-SP6 cluster hosted at CINECA Supercomputing Center (Bologna, Italy). Its main hardware and software features are the following:

- Model: IBM pSeries 575
- Architecture: IBM P6-575 Infiniband Cluster
- Processor Type: IBM Power6, 4.7 GHz
- Peak performance: over 100 Tflops
- Operating System: AIX 6.1 (IBM UNIX)
- Internal Network: Infiniband x4 DDR
- Computing Nodes: 168; 16 chips dual core / node = 32 cores/node = 5376 cores in total
- RAM: 1.2 TB (128 GB/node)
- Disk Space: 1.2 PB
- SMT: Hardware support for Simultaneous Multi-Threading; in some cases can double the number of processes per core (*i.e.* upto 64 virtual cpu/node).

For more detailed information about SP6 cluster see [118].

Notation

From a notational viewpoint, when not otherwise stated, bold lowercase Roman or Greek letters represent column vectors, while the non-bold ones are scalars, scalar functions and/or vector components depending on the context. Calligraphic uppercase letters are usually reserved to sets and operators. Roman uppercase letters are matrices or discrete nonlinear operators. If not otherwise stated, $\|\cdot\|$ is the Euclidean norm for both vectors and functions.

Chapter 1

Mathematical setting of edge-preserving joint inversion

1.1 The reasons for edge-preserving regularization

To simplify reasoning and notation, in this chapter we only consider the case where two data sets are available for the same domain and two models have to be recovered from these data.

In this thesis, we follow the classical Tikhonov's idea to face ill-posed problems by minimizing a regularized functional [12, 35, 76–78, 106, 119, 124, 134].

Suppose we want to investigate the hidden structure of an "object" and we have available two data sets $d^{(j)}$, $j = 1, 2$, obtained by some known and non-invasive observations of the object. We assume that a (possibly approximated) functional expression exists and is known of each transformation producing the observed data. We want to recover the object structure from the data or, better, we want to estimate a "model" of this structure, as accurate as possible.

In the joint inversion problem we use a functional that depends upon the two models, $m^{(1)}$ and $m^{(2)}$, whose minimization provides the approximate solution of the ill-posed problem. The idea we follow aims to write the required functional as the sum of the functionals used for the two separated inversions, plus one "joining" term involving both $m^{(1)}$ and $m^{(2)}$ simultaneously. Concerning the two functionals of the separated inversions we use the following:

$$\|\mathcal{A}(m) - d\| + \lambda S_{\mathrm{MS}}(m, m_{\mathrm{apr}}, \xi)\,. \tag{1.1}$$

Here, the first term is the *misfit* term (the Euclidean distance from the estimated and the observed data) where the *forward operator* $\mathcal{A}$ can be either a linear or a nonlinear operator expressing the process generating the observed data from the unknown object. The second term in (1.1) is the *regularization term* (sometimes called also *stabilization term*): here we choose the *Minimum Support Functional* (MSF) introduced and studied in [57, 82, 102, 144], which is defined as

$$MSF(m, m_{\mathrm{apr}}, \xi) = \int_\Omega \frac{(m - m_{\mathrm{apr}})^2}{(m - m_{\mathrm{apr}})^2 + \xi^2}\,\mathrm{d}V \tag{1.2}$$

where m_{apr} is a given *a priori estimate* of the expected model and ξ is a scalar called *focusing parameter*. The estimate m_{apr} is often set to a constant background value, but it

can possibly include a rough idea of the structure of the region being investigated. The focusing parameter has a twofold goal. First, since it's easily seen that $MSF(m, m_{\text{apr}}, \xi) \to \text{supp}(m - m_{\text{apr}})$ as $\xi \to 0$, then ξ enables the operator to asymptotically identify the support (that is the volume) where m differs from m_{apr}; hence, setting the latter to the background one would asymptotically reconstruct exactly the volume of hidden anomalies. Second, strict positiveness of ξ guarantees the existence of operator derivatives. The MSF operator is particularly suitable for structures with sharp boundaries and its properties are deeply studied and analyzed in [129, 144, 145].

Other largely known and studied choices are possible for the regularization functional, such as for instance the *Total Variation* (TV) regularization [12, 50, 93, 134], or the Mumford-Shah regularization [96, 97]. Alternatively, one could also consider the cross-gradient-constrained nonlinear generalized least-squares formulation of the joint inversion [43–49]. Each of these approaches has its own peculiarities (such as smoothing capabilities, or easier computations), but whether these becomes advantages or disadvantages usually depends on the application at hand. We stress here that we target the detection of blocky hidden structures and our goal is to emphasize the "volumetric relationship" we expect to hold between the collected data, within the well established Tikhonov setting.

The Lagrange multiplier λ in (1.1), multiplying either the misfit or the regularization term, represents the trade off between the minimization of one or another. How to choose the value of this parameter is a matter of fact [64] and a huge literature exists, which mainly provides more or less heuristic procedures. For instance, in denoising problems a quite well understood *discrepancy principle* can be used if a suitable estimate of the noise level is known (see for instance [12, 13, 129]). In general the parameter value is determined by requiring that the equation

$$\|\Phi(m, d)\| = \varepsilon \tag{1.3}$$

is satisfied, where $\Phi(m, d)$ is a given *discrepancy function* and ε is the error on the experimental measurements.

In the case of joint inversion, two Lagrange multipliers are usually needed because two such equations (1.3) have to be solved for the relative misfit of the two models, as it happens for the case of elastic and electromagnetic travel-time joint inversion (a method for a unique, less restrictive equation can be found in [61], but it needs strong monotonicity assumptions).

Let us motivate now the choice for the joint functional. One of the main issues in solving inverse problems is how to include into the formulation and/or the solution additional *a priori* information that can be available. In the present case of joint inversion, this additional *a priori* information that we want to use on the models is that their variations in the investigated volume occur in the same regions. For instance, an anomaly of the radar-waves slowness distribution with respect to the background corresponds to an anomaly of the elastic-waves slowness distribution and vice versa. Notice that this is the only information we impose: we don't know, in fact, if the variations of the two models are both positive or negative; moreover, also the relative variations with respect to the background can be different for the two models. To use the information, as proposed in [61], we first introduce a *structure operator* $\mathcal{S}(m)$. Qualifying features for a good structure operator are that it is independent on the variation magnitude of the properties characterizing the hidden structures (such as anomalies) and that it is independent on the versus of this change along the structure boundaries. For instance, in a Geophysical setting, it

is reasonable not to expect that if the propagation velocity of the electromagnetic waves in a ground anomaly becomes 10 times smaller than the one in the background, then also the electric conductivity change of the same amount, or even that it actually decreases. Hence, structure operators are designed to address at least these features. Another desirable property is that such an operator maps the models in the $[0, 1]$ interval. In [61], $\mathcal{S}$ depends on the modulus of the Laplacian of the models. Here, we use the following form for the structure operator:

$$\mathcal{S}(m^{(j)}, \xi_j) = \frac{\left|\nabla_{\boldsymbol{x}}\, m^{(j)}\right|^2}{\left|\nabla_{\boldsymbol{x}}\, m^{(j)}\right|^2 + \xi_j^2} \tag{1.4}$$

which operates exactly as the structure operator used in [61], but using the modulus of the gradient in place of the modulus of the Laplacian. It has been shown that this particular choice is good to reconstruct blocky targets in which the model changes discontinuously. The operator (1.4) is also known as *Minimum Gradient Support* (MGS) operator: its expression is similar to the gradient of the TV regularizer, but they are not the same. The operator (1.4) could also be used in the following form

$$\mathcal{S}(m^{(j)}, m^{(j)}_{\text{apr}}, \xi_j) = \frac{\left|\nabla_{\boldsymbol{x}}\, (m^{(j)} - m^{(j)}_{\text{apr}})\right|^2}{\left|\nabla_{\boldsymbol{x}}\, (m^{(j)} - m^{(j)}_{\text{apr}})\right|^2 + \xi_j^2} \tag{1.5}$$

that matches the previous one if $m^{(j)}_{\text{apr}}$ is constant, due to linearity of the gradient. From a computational viewpoint this form is easily implemented by just adding a vector difference, but it seems to provide no relevant benefits with respect to (1.4), so we retain that form. Notice also that other structure operators can be chosen from a theoretical viewpoint (provided that reasonable regularity conditions hold true): we could have chosen a polynomial as in [61] or a Fermi function, without affecting the reasoning. However, the choice can be of computational impact after problem discretization. A suitable choice of the *Joining Functional* (JF) comparing the structures of the two models can be the norm of the structures difference, as in [61], because the minimization of this functional gives models having "similar" structures. This choice have been demonstrated to be effective to reconstruct also smooth anomalies. In our case, since we are interested in anomalies having sharp boundaries, instead of minimizing the difference between the structure of the two models, we want to minimize *the volume* in which the two models have different structure. This, of course, is *not* the same goal: the models' structure difference can be small but spread over a large volume, while, on the contrary, it could be even not too small in value, but in a localized volume. For this reason, we use again the analytic form of MSF: in place of the difference between m and m_{apr}, we use this time the difference *between the structures* of the two models. Hence, the JF functional finally reads:

$$\begin{aligned} &JF(m^{(1)}, \xi_1, m^{(2)}, \xi_2, \xi_3) \\ &\qquad = \int_\Omega \frac{\left(S(m^{(1)}, \xi_1) - S(m^{(2)}, \xi_2)\right)^2}{\left(S(m^{(1)}, \xi_1) - S(m^{(2)}, \xi_2)\right)^2 + \xi_3^2}\, \mathrm{d}V \end{aligned} \tag{1.6}$$

where ξ_3 is again a focusing parameter.

In the continuous setting, the two-models joint inversion problem with minimum support edge-preserving regularization can be written as follows:

$$P\big(m^{(1)}, m^{(2)}, d^{(1)}, d^{(2)}, m^{(1)}_{\text{apr}}, m^{(2)}_{\text{apr}}, \boldsymbol{\lambda}, \boldsymbol{\xi}\big) =$$

$$\left\|\mathcal{A}_1(m^{(1)}) - d^{(1)}\right\|^2 + \left\|\mathcal{A}_2(m^{(2)}) - d^{(2)}\right\|^2 \tag{1.7a}$$

$$+ \lambda_1 \int \frac{\left(m^{(1)} - m^{(1)}_{\text{apr}}\right)^2}{\left(m^{(1)} - m^{(1)}_{\text{apr}}\right)^2 + \xi_1^2} \mathrm{d}V + \lambda_2 \int \frac{\left(m^{(2)} - m^{(2)}_{\text{apr}}\right)^2}{\left(m^{(2)} - m^{(2)}_{\text{apr}}\right)^2 + \xi_2^2} \mathrm{d}V \tag{1.7b}$$

$$+ \lambda_3 \int \frac{\left(\dfrac{\left(\nabla_{\boldsymbol{x}}\, m^{(1)}\right)^2}{\left(\nabla_{\boldsymbol{x}}\, m^{(1)}\right)^2 + \xi_3^2} - \dfrac{\left(\nabla_{\boldsymbol{x}}\, m^{(2)}\right)^2}{\left(\nabla_{\boldsymbol{x}}\, m^{(2)}\right)^2 + \xi_4^2}\right)^2}{\left(\dfrac{\left(\nabla_{\boldsymbol{x}}\, m^{(1)}\right)^2}{\left(\nabla_{\boldsymbol{x}}\, m^{(1)}\right)^2 + \xi_3^2} - \dfrac{\left(\nabla_{\boldsymbol{x}}\, m^{(2)}\right)^2}{\left(\nabla_{\boldsymbol{x}}\, m^{(2)}\right)^2 + \xi_4^2}\right)^2 + \xi_5^2} \mathrm{d}V \tag{1.7c}$$

Here one can easily detect the elements of the traditional inversion for the two separated models: the sum of the first terms in (1.7a) and (1.7b) is the objective function of the classical regularization problem for model $m^{(1)}$, while the sum of the rightmost terms in (1.7a) and (1.7b) is the objective function of the classical regularization problem for model $m^{(2)}$. On the contrary, the term (1.7c) represents the *joining term*, that is the additional regularization function: it *explicitly* binds the solution to meet the geometrical requirement of spatial matching between transition zones of the two models. The parameters to be *chosen* are $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \lambda_3)^T$ and $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_5)^T$.

Remark 1.1 *Actually, even if from a mathematical viewpoint the focusing parameters are aimed to allow differentiability, from a physical viewpoint they should rely on local characteristics of the model, such as its magnitude. Then, strictly speaking, each focusing parameter should be considered as a scalar function of the spatial position $\xi_j = \xi_j(\boldsymbol{x})$. However, the treatment would become overly complex and it is often so difficult to estimate good values even for the constant case, that it is common practice to consider focusing parameters just constants. In the rest of the work we will adhere to this practice.*

Note that most often the two *misfit* terms are quite different from each other and of different scales. As an example, $d^{(1)}$ could be potential difference measured in Volts (as in the case of electrical resistivity tomography), while $d^{(2)}$ a time in nanoseconds (as in the case of time propagation tomography). This is the reason why it is useful to keep distinct regularization parameters λ_1 and λ_2. An alternative could be to find a way to normalize the data. We briefly digress now on this idea. If some estimate δ of the absolute error in measurements (that is the *noise*) is available, then the constraint here can be the discrepancy $\|\mathcal{A}(m) - d\| = \delta$. To simplify the problem we assume that δ is the same for all kind of measurements, but in general we have an error vector $\boldsymbol{\delta}$ of the same size as the data, that is each measurement can have a different absolute error estimate. By scaling the operators and the data by the error we get:

$$\mathcal{B}_j(m^{(j)}) = \frac{\mathcal{A}_j(m^{(j)})}{\delta} \quad \text{and} \quad \widetilde{d}^{(j)} = \frac{d^{(j)}}{\delta} \qquad j = 1, 2$$

hence the misfit and the discrepancy constraint become

$$\left\|\mathcal{B}_j(m^{(j)}) - \widetilde{d}^{(j)}\right\|^2 \quad \text{and} \quad \left\|\mathcal{B}_j(m^{(j)}) - \widetilde{d}^{(j)}\right\|^2 = 1$$

respectively, $j = 1, 2$. Notice that these are dimensionless quantities. However, in general the noise depends on the single datum because one can have a different error estimate for each measurement. Thus the noise should be a function or, after discretization, a vector $\boldsymbol{\delta}$ the same length as $\boldsymbol{d}$. Then the misfit term after discretization reads as

$$\left\|\frac{A_j(\boldsymbol{m}^{(j)}) - \boldsymbol{d}^{(j)}}{\boldsymbol{\delta} j}\right\|^2 = \left\|\frac{A_j(\boldsymbol{m}^{(j)})}{\boldsymbol{\delta} j} - \frac{\boldsymbol{d}^{(i)}}{\boldsymbol{\delta} j}\right\|^2 = \left\|B_j(\boldsymbol{m}^{(j)}) - \widetilde{\boldsymbol{d}}^{(j)}\right\|^2$$

where vector quotients are intended as the vector of componentwise division. Unfortunately, the noise estimate is often not available and, moreover, while this normalization removes two terms from the overall joint inversion functional, it adds two extra nonlinear constraints to the problem, that in turn requires a more sophisticated solver for general equality constrained problems. Hence, in the following we will not follow this way.

The goal is thus to minimize P in (1.7) subject to *suitable* constraints. The constraints can be of different kind, but they are intended to provide the additional information to recover the model from the data. One kind of such an information is provided by the *support* of the gradient function of each model, that is the subdomain where each model changes its value. If we can assume that both the models change approximately in the same spatial positions (even if not necessarily of the same amount), then we can try to recover the anomaly domain by minimizing this support.

1.2 Discretization and linearity

We now leave the continuous setting and consider a discretized domain and a corresponding discrete problem. We further assume that the domain we investigate is a rectangle or a parallelepiped large enough to completely surround the hidden structures we would like to reconstruct.

For the space discretization we assume a rectangular grid (either 2D or 3D), with possibly different grid steps h_t in the different spatial dimensions x_t. The domain is then partitioned into a number L of cells with either a 2D rectangular or a 3D parallelepiped shape.

When dealing with derivatives discretization we will use forward or central finite differences computed at the cells center; inside each cell, the model is supposed to assume a constant value equal to the value at its center. The models now become vectors $\boldsymbol{m}^{(j)}$, $j = 1, 2$, by stacking in one column the cell values, following a natural (lexicographic) ordering, as it will be better described later.

Let's further assume from now on that the forward operators are *linear*: it means that $\mathcal{A}(m) = A\boldsymbol{m}$, where A is a matrix of suitable size. It is finally assumed that the data of both models are acquired with the same number N of samples.

Under these assumptions, one could notice that, by rearranging the operators, the models and the data in the form

$$A = \begin{pmatrix} A_1 & \mathbf{0} \\ \mathbf{0} & A_2 \end{pmatrix}, \quad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{m}^{(1)} \\ \boldsymbol{m}^{(2)} \end{pmatrix}, \quad \text{and} \quad \boldsymbol{Y} = \begin{pmatrix} \boldsymbol{d}^{(1)} \\ \boldsymbol{d}^{(2)} \end{pmatrix} \tag{1.8}$$

the misfit terms could be shortened to $\|A\boldsymbol{X} - \boldsymbol{Y}\|^2$. However, this representations does not provide any computational benefit and we no longer use it here.

The discretization of the minimum support functionals gives the following terms

$$S_{\mathrm{MS}}(\boldsymbol{m}^{(j)}, \xi_j) = \sum_{i=1}^{L} \frac{(m_i^{(j)} - m_{i,\mathrm{apr}}^{(j)})^2}{(m_i^{(j)} - m_{i,\mathrm{apr}}^{(j)})^2 + \xi_j^2} \qquad j = 1, 2.$$

In a similar way we can discretize the joining term in (1.7c). Then, the discretized joint inversion functional can be written as

$$P(\boldsymbol{m}^{(1)}, \boldsymbol{m}^{(2)}, \boldsymbol{d}^{(1)}, \boldsymbol{d}^{(2)}, \boldsymbol{m}_{\mathrm{apr}}^{(1)}, \boldsymbol{m}_{\mathrm{apr}}^{(2)}, \boldsymbol{\lambda}, \boldsymbol{\xi}) \tag{1.9a}$$

$$= \left\| A_1(\boldsymbol{m}^{(1)}) - \boldsymbol{d}^{(1)} \right\|^2 + \left\| A_2(\boldsymbol{m}^{(2)}) - \boldsymbol{d}^{(2)} \right\|^2 \tag{1.9b}$$

$$+ \lambda_1 \sum_{i=1}^{L} \frac{(m_i^{(1)} - m_{i,\mathrm{apr}}^{(1)})^2}{(m_i^{(1)} - m_{i,\mathrm{apr}}^{(1)})^2 + \xi_1^2} + \lambda_2 \sum_{i=1}^{L} \frac{(m_i^{(2)} - m_{i,\mathrm{apr}}^{(2)})^2}{(m_i^{(2)} - m_{i,\mathrm{apr}}^{(2)})^2 + \xi_2^2} \tag{1.9c}$$

$$+ \lambda_3 \sum_{i=1}^{L} \frac{\left(\dfrac{(\nabla_{\boldsymbol{x}}\, m_i^{(1)})^2}{(\nabla_{\boldsymbol{x}}\, m_i^{(1)})^2 + \xi_3^2} - \dfrac{(\nabla_{\boldsymbol{x}}\, m_i^{(2)})^2}{(\nabla_{\boldsymbol{x}}\, m_i^{(2)})^2 + \xi_4^2} \right)^2}{\left(\dfrac{(\nabla_{\boldsymbol{x}}\, m_i^{(1)})^2}{(\nabla_{\boldsymbol{x}}\, m_i^{(1)})^2 + \xi_3^2} - \dfrac{(\nabla_{\boldsymbol{x}}\, m_i^{(2)})^2}{(\nabla_{\boldsymbol{x}}\, m_i^{(2)})^2 + \xi_4^2} \right)^2 + \xi_5^2} \tag{1.9d}$$

In the next section we see how the function is used to seek the required models estimates.

1.3 Discrete derivatives computation

We plan to face the solution of the joint inversion problem, that is the minimization of the discrete joint functional (1.9), by first- or second-order iterative solvers. For this reason, we need to compute (or, better, approximate) functional derivatives with respect to the model and to the space. As it will be shown shortly, while discrete derivatives of the misfit terms and also of the regularization terms are not too difficult to compute, the first and second derivatives of the joining term (1.9d) involves more complicated computations.

Define the diagonal matrices

$$W_j = W_{\xi_j}(\boldsymbol{m}^{(j)}) = \mathrm{diag}\left(\left((m_i^{(j)} - m_{i,\mathrm{apr}}^{(j)})^2 + \xi_j^2 \right)_{i=1,\dots,M}^{-1/2} \right) \qquad j = 1, 2. \tag{1.10}$$

Then

$$S_{\mathrm{MS}}(\boldsymbol{m}^{(j)}) = \sum_{i=1}^{M} \frac{\left(m_i^{(j)} - m_{i,\mathrm{apr}}^{(j)}\right)^2}{\left(m_i^{(j)} - m_{i,\mathrm{apr}}^{(j)}\right)^2 + \xi_j^2} = \left\| W_j\left(\boldsymbol{m}^{(j)} - \boldsymbol{m}_{\mathrm{apr}}^{(j)}\right) \right\|_2^2 \qquad j = 1, 2. \tag{1.11}$$

Now, for easier notation we ignore in the next few equations the index j, since everything holds true for $j = 1, 2$. From the vector calculus and the chain rule we have

$$\nabla_{\boldsymbol{m}} \left(\|W(\boldsymbol{m} - \boldsymbol{m}_{\mathrm{apr}})\|_2^2 \right) = \nabla_{\boldsymbol{m}} \left(\left(W(\boldsymbol{m} - \boldsymbol{m}_{\mathrm{apr}}) \right)^T \left(W(\boldsymbol{m} - \boldsymbol{m}_{\mathrm{apr}}) \right) \right) \tag{1.12}$$

$$= 2 \left(\nabla_{\boldsymbol{m}} \left(W(\boldsymbol{m} - \boldsymbol{m}_{\mathrm{apr}}) \right) \right) W(\boldsymbol{m} - \boldsymbol{m}_{\mathrm{apr}}) \,. \tag{1.13}$$

Since

$$W(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})=\left(\frac{m_1-m_{1,\text{apr}}}{\sqrt{\left(m_1-m_{1,\text{apr}}\right)^2+\xi^2}},\dots,\frac{m_M-m_{M,\text{apr}}}{\sqrt{\left(m_M-m_{M,\text{apr}}\right)^2+\xi^2}}\right)^T \tag{1.14}$$

and given that

$$\begin{aligned}
&\frac{\partial}{\partial m_\ell}\left(\frac{m_i-m_{i,\text{apr}}}{\sqrt{\left(m_i-m_{i,\text{apr}}\right)^2+\xi^2}}\right)\\
&\quad=\delta_{\ell i}\frac{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{1/2}-(m_i-m_{i,\text{apr}})\left[\frac{1}{2}2(m_i-m_{i,\text{apr}})\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{-1/2}\right]}{(m_i-m_{i,\text{apr}})^2+\xi^2}\\
&\quad=\delta_{\ell i}\frac{(m_i-m_{i,\text{apr}})^2+\xi^2-(m_i-m_{i,\text{apr}})^2}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{3/2}}\\
&\quad=\delta_{\ell i}\frac{\xi^2}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{3/2}}\qquad \ell,i=1,\dots,M,
\end{aligned} \tag{1.15}$$

where $\delta_{\ell i}$ is the Kroneker's symbol[1], we have

$$\begin{aligned}
\nabla_{\boldsymbol{m}}\left(W(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})\right)&=\text{diag}\left(\left(\frac{\partial}{\partial m_i}\frac{m_i-m_{i,\text{apr}}}{\sqrt{\left(m_i-m_{i,\text{apr}}\right)^2+\xi^2}}\right)_{i=1,\dots,M}\right)\\
&=\text{diag}\left(\left(\frac{\xi^2}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{3/2}}\right)_{i=1,\dots,M}\right)
\end{aligned} \tag{1.16}$$

which is a $M\times M$ matrix. Getting back to (1.13), it follows that

$$\begin{aligned}
&2\left(\nabla_{\boldsymbol{m}}\left(W(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})\right)\right)W(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})=\\
&\quad=2\,\text{diag}\left(\left(\frac{\xi^2}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{3/2}}\right)_{i=1,\dots,M}\right)\cdot\\
&\qquad\qquad\cdot\,\text{diag}\left(\left(\frac{1}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{1/2}}\right)_{i=1,\dots,M}\right)(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})\\
&\quad=2\,\text{diag}\left(\left(\frac{\xi^2}{\left((m_i-m_{i,\text{apr}})^2+\xi^2\right)^{2}}\right)_{i=1,\dots,M}\right)(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})\\
&\quad=2\xi^2W^4(\boldsymbol{m}-\boldsymbol{m}_{\text{apr}})\\
&\quad=2\xi^2\left(\frac{m_1-m_{1,\text{apr}}}{\left((m_1-m_{1,\text{apr}})^2+\xi^2\right)^2},\dots,\frac{m_M-m_{M,\text{apr}}}{\left((m_M-m_{M,\text{apr}})^2+\xi^2\right)^2}\right)^T
\end{aligned} \tag{1.17}$$

[1] $\delta_{\ell i}=\begin{cases}1 & \text{if } \ell=i,\\ 0 & \text{otherwise.}\end{cases}$

which is a $M \times 1$ column vector. Summarizing:

$$\nabla_{\boldsymbol{m}^{(j)}} \left(\left\| W_j \big(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j)}_{\text{apr}} \big) \right\|_2^2 \right) = 2\xi_j^2 W_j^4 \big(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j)}_{\text{apr}} \big) \qquad j = 1, 2.$$

To compute also the second derivatives we observe that, by using (1.15), we have

$$\begin{aligned}
&\frac{\partial}{\partial m_\ell} \frac{m_i - m_{i,\text{apr}}}{\left((m_i - m_{i,\text{apr}})^2 + \xi^2 \right)^2} \\
&\quad = \delta_{\ell i} \frac{\left((m_i - m_{i,\text{apr}})^2 + \xi^2 \right)^2 - (m_i - m_{i,\text{apr}})\, 2 \left((m_i - m_{i,\text{apr}})^2 + \xi^2 \right) 2 (m_i - m_{i,\text{apr}})}{\left((m_i - m_{i,\text{apr}})^2 + \xi^4 \right)^4} \\
&\quad = \delta_{\ell i} \frac{(m_i - m_{i,\text{apr}})^2 + \xi^2 - 4(m_i - m_{i,\text{apr}})^2}{\left((m_i - m_{i,\text{apr}})^2 + \xi^4 \right)^3} \\
&\quad = \delta_{\ell i} \frac{\xi^2 - 3(m_i - m_{i,\text{apr}})^2}{\left((m_i - m_{i,\text{apr}})^2 + \xi^2 \right)^3} \qquad \ell, i = 1, \ldots, M.
\end{aligned} \tag{1.18}$$

Hence, for the Hessian we have ($j = 1, 2$):

$$\begin{aligned}
&\nabla^2_{\boldsymbol{m}^{(j)} \boldsymbol{m}^{(j)}} \left(\left\| W_j \big(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j)}_{\text{apr}} \big) \right\|_2^2 \right) \\
&\qquad = \nabla_{\boldsymbol{m}} \left(2\xi_j^2 W_j^4 \big(\boldsymbol{m}^{(j)} - \boldsymbol{m}^{(j)}_{\text{apr}} \big) \right) \\
&\qquad = 2\xi_j^2 \text{diag} \left(\left(\frac{\xi_j^2 - 3\big(m_i^{(j)} - m_{i,\text{apr}}^{(j)} \big)^2}{\left(\big(m_i^{(j)} - m_{i,\text{apr}}^{(j)} \big)^2 + \xi_j^2 \right)^3} \right)_{i=1,\ldots,M} \right) \qquad (1.19) \\
&\qquad = 2\xi_j^2 W_j^6 \text{diag} \left(\left(\xi_j^2 - 3\big(m_i^{(j)} - m_{i,\text{apr}}^{(j)} \big)^2 \right)_{i=1,\ldots,M} \right) \qquad (1.20)
\end{aligned}$$

which is a $M \times M$ matrix. Note that there are not mixed derivatives in the Hessian because the gradient of $\boldsymbol{m}^{(j)}$ depends on $\boldsymbol{m}^{(j)}$ only. Hence, we have the explicit expression of the first and second derivatives of the minimum support stabilizing term for both models: they will be needed in the application of gradient-like methods for the joint functional minimization.

Now we have to consider the *mixing* term. As we mentioned before, this functional is based on a *model structure* function $y(\boldsymbol{m})$ which depends on the spatial gradient of the model itself. In the following results we consider the case where the spatial derivative is approximated by a forward difference. The reasoning will be next generalized to other discretization schemes.

Lemma 1.1 *Suppose that the spatial derivative discretization steps h_{x_t}, $t = 1, 2, 3$, are strictly positive and small enough. Assume also that $\xi_\ell > 0$ $\forall \ell = 1, \ldots, 5$. Then the gradient with respect to the model of the joining functional can be written as*

$$\begin{aligned}
\nabla_{\boldsymbol{m}^{(j)}} MIX = \; & (-1)^{j-1} 4 \xi_{j+2}^2 \xi_5^2 \left[\left(\mathbf{1}_M^T \otimes I_M \right) D^T D \left(I_M \otimes \boldsymbol{m}^{(j)} \right) \right] \qquad (1.21) \\
& \cdot diag \left(\left(\left(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2 \right)^{-2} \right)_{i=1,\ldots,M} \right) W_5^4 (\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \,.
\end{aligned}$$

Proof. To compute the derivatives, we first re-formulate the expression in terms of its component functions and then we use the chain rule. The joining functional accounts for the support where the structure functions of the two models differ: that's why we apply the minimum support stabilizer to the difference of the two model structure functions. So, by defining

$$y_i^{(j)} = y\big(m_i^{(j)}\big) = \frac{\big\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\big\|_2^2}{\big\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\big\|_2^2 + \xi_{j+2}^2} \qquad j = 1, 2 \tag{1.22}$$

and $\boldsymbol{y}^{(j)} = \big(y_1^{(j)}, \ldots, y_M^{(j)}\big)^T$ we can write the mixing term as

$$MIX(\boldsymbol{m}) = \sum_{i=1}^{M} \frac{\big(y_i^{(1)} - y_i^{(2)}\big)^2}{\big(y_i^{(1)} - y_i^{(2)}\big)^2 + \xi_5^2} \qquad \text{with} \quad \boldsymbol{m} = \begin{pmatrix} \boldsymbol{m}^{(1)} \\ \boldsymbol{m}^{(2)} \end{pmatrix}. \tag{1.23}$$

Again, by introducing the matrices

$$W_{j+2} = W_{\xi_{j+2}}\big(\boldsymbol{m}^{(j)}\big) = \operatorname{diag}\left(\left(\frac{1}{\sqrt{\big\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\big\|_2^2 + \xi_{j+2}^2}}\right)_{i=1,\ldots,M}\right), \qquad j = 1, 2, \tag{1.24}$$

$$W_5 = W_{\xi_5}(\boldsymbol{m}) = \operatorname{diag}\left(\left(\frac{1}{\sqrt{\big(y_i^{(1)} - y_i^{(2)}\big)^2 + \xi_5^2}}\right)_{i=1,\ldots,M}\right), \tag{1.25}$$

we can further simplify the MIX term expression as

$$MIX(\boldsymbol{m}) = \Big\|W_5\big(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}\big)\Big\|_2^2. \tag{1.26}$$

To compute the first derivative of the joining term with respect to the j-th model, $j = 1, 2$, we can apply again the chain rule, having

$$\nabla_{\boldsymbol{m}^{(j)}}\, MIX = 2\Big(\nabla_{\boldsymbol{m}^{(j)}}\, \boldsymbol{y}^{(j)}\Big)\Big(\nabla_{\boldsymbol{y}^{(j)}}\, \big(W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)})\big)\Big) W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}), \tag{1.27}$$

because $\nabla_{\boldsymbol{m}^{(j)}}\, \boldsymbol{y}^{(k)} = \mathbf{0}$ when $k \neq j$. We can write

$$W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) = \left(\frac{y_1^{(1)} - y_1^{(2)}}{\sqrt{\Big(y_1^{(1)} - y_1^{(2)}\Big)^2 + \xi_5^2}}, \ldots, \frac{y_M^{(1)} - y_M^{(2)}}{\sqrt{\Big(y_M^{(1)} - y_M^{(2)}\Big)^2 + \xi_5^2}}\right)^T \tag{1.28}$$

and by proceeding in the same way as for (1.15) we have for $j = 1$

$$\frac{\partial}{\partial y_\ell^{(1)}}\left(\frac{y_i^{(1)} - y_i^{(2)}}{\sqrt{\Big(y_i^{(1)} - y_i^{(2)}\Big)^2 + \xi_5^2}}\right) = \delta_{\ell i}\frac{\xi_5^2}{\left(\Big(y_i^{(1)} - y_i^{(2)}\Big)^2 + \xi_5^2\right)^{3/2}} \qquad \ell, i = 1, \ldots, M. \tag{1.29}$$

For $j = 2$ we have instead

$$\begin{aligned}
&\frac{\partial}{\partial y_\ell^{(2)}}\left(\frac{y_i^{(1)}-y_i^{(2)}}{\sqrt{\left(y_i^{(1)}-y_i^{(2)}\right)^2+\xi_5^2}}\right)\\
&\quad=\delta_{\ell i}\frac{-\left((y_i^{(1)}-y_i^{(2)})^2+\xi_5^2\right)^{1/2}-(y_i^{(1)}-y_i^{(2)})\left[-(y_i^{(1)}-y_i^{(2)})\left((y_i^{(1)}-y_i^{(2)})^2+\xi_5^2\right)^{-1/2}\right]}{(y_i^{(1)}-y_i^{(2)})^2+\xi_5^2}\\
&\quad=\delta_{\ell i}\frac{-(y_i^{(1)}-y_i^{(2)})^2-\xi_5^2+(y_i^{(1)}-y_i^{(2)})^2}{\left((y_i^{(1)}-y_i^{(2)})^2+\xi_5^2\right)^{3/2}}\\
&\quad=\delta_{\ell i}\frac{-\xi_5^2}{\left((y_i^{(1)}-y_i^{(2)})^2+\xi_5^2\right)^{3/2}}\qquad \ell,i=1,\ldots,M.
\end{aligned}\tag{1.30}$$

Hence, in both cases

$$\frac{\partial}{\partial y_\ell^{(j)}}\left(\frac{y_i^{(1)}-y_i^{(2)}}{\sqrt{\left(y_i^{(1)}-y_i^{(2)}\right)^2+\xi_5^2}}\right)=\delta_{\ell i}(-1)^{j-1}\xi_5^2\left((y_i^{(1)}-y_i^{(2)})^2+\xi_5^2\right)^{-3/2}\qquad j=1,2,\tag{1.31}$$

where $\ell, i = 1, \ldots, M$, and $\delta_{\ell i}$ is again the Kroneker's symbol (related to the cell index, in this case). Thus, it follows that

$$\begin{aligned}
\nabla_{\boldsymbol{y}^{(j)}}\left(W_5\left(\boldsymbol{y}^{(1)}-\boldsymbol{y}^{(2)}\right)\right)&=(-1)^{j-1}\xi_5^2\,\mathrm{diag}\left(\left(\left(y_i^{(1)}-y_i^{(2)}\right)^2+\xi_5^2\right)_{i=1,\ldots,M}^{-3/2}\right)\\
&=(-1)^{j-1}\xi_5^2 W_5^3
\end{aligned}\tag{1.32}$$

which is a $M \times M$ diagonal matrix.

Now we have to compute the gradient of $\boldsymbol{y}^{(j)}$ with respect to $\boldsymbol{m}^{(j)}$, that is the matrix

$$\nabla_{\boldsymbol{m}^{(j)}}\,\boldsymbol{y}^{(j)}=\nabla_{\boldsymbol{m}^{(j)}}\,y(\boldsymbol{m}^{(j)})=\nabla_{\boldsymbol{m}^{(j)}}\left(y_1^{(j)},\ldots,y_M^{(j)}\right)^T\qquad j=1,2.\tag{1.33}$$

We forget once again for easier notation the superscript j (hence what follows applies to each model $\boldsymbol{m}^{(j)}$ separately) and we recall here that the spatial gradient $\nabla_{\boldsymbol{x}}\,\boldsymbol{m}$ is approximated by a (forward) finite difference operator $G = \left(G_{x_1}^T, G_{x_2}^T, G_{x_3}^T\right)^T$ such that

$$\nabla_{\boldsymbol{x}}\,\boldsymbol{m}=\begin{pmatrix}\frac{\partial}{\partial x_1}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_1}m_2(\boldsymbol{x}) & \ldots & \frac{\partial}{\partial x_1}m_M(\boldsymbol{x})\\ \frac{\partial}{\partial x_2}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_2}m_2(\boldsymbol{x}) & \ldots & \frac{\partial}{\partial x_2}m_M(\boldsymbol{x})\\ \frac{\partial}{\partial x_3}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_3}m_2(\boldsymbol{x}) & \ldots & \frac{\partial}{\partial x_3}m_M(\boldsymbol{x})\end{pmatrix}\approx\begin{pmatrix}\left(G_{x_1}\boldsymbol{m}(\boldsymbol{x})\right)^T\\ \left(G_{x_2}\boldsymbol{m}(\boldsymbol{x})\right)^T\\ \left(G_{x_3}\boldsymbol{m}(\boldsymbol{x})\right)^T\end{pmatrix}\tag{1.34}$$

where G_{x_t} is the $M \times M$ matrix of the forward finite difference approximation of the model derivative with respect to the spatial coordinate x_t, $t = 1, 2, 3$. The form of these discrete operators is well known and depends on the ordering chosen to number the spatial cells of the discretized volume: we return on the explicit form of G later in the section. Considering first the spatial gradient approximation of each i-th model component $m_i(\boldsymbol{x})$ we have

$$\nabla_{\boldsymbol{x}}\,m_i\approx\left((G_{x_1})_{i*}\boldsymbol{m},(G_{x_2})_{i*}\boldsymbol{m},(G_{x_3})_{i*}\boldsymbol{m}\right)^T=(S_iG)\boldsymbol{m}=\boldsymbol{g}_i\qquad i=1,\ldots,M,\tag{1.35}$$

where $(G_{x_t})_{i*}$ is the i-th row of the matrix G_{x_t}, $t = 1, 2, 3$, and S_i is the corresponding *selection matrix* sized $3 \times 3M$, that is

$$S_i = \begin{pmatrix} \mathbf{e}_i^T \\ \mathbf{e}_{M+i}^T \\ \mathbf{e}_{2M+i}^T \end{pmatrix} = \left(\begin{array}{c|c|c} 0 \ \dots 1 \dots \ 0 & 0 \ \dots\dots\dots \ 0 & 0 \ \dots\dots\dots \ 0 \\ 0 \ \dots\dots\dots \ 0 & 0 \ \dots 1 \dots \ 0 & 0 \ \dots\dots\dots \ 0 \\ 0 \ \dots\dots\dots \ 0 & 0 \ \dots\dots\dots \ 0 & 0 \ \dots 1 \dots \ 0 \end{array}\right), \qquad \begin{matrix} \uparrow \\ i \end{matrix} \quad \begin{matrix} \uparrow \\ M+i \end{matrix} \quad \begin{matrix} \uparrow \\ 2M+i \end{matrix} \tag{1.36}$$

so that $S_i G$ is a $3 \times M$ matrix and $\boldsymbol{g}_i$ is a 3×1 column vector, for all $i = 1, \dots M$. Computing the gradient with respect to the model we get

$$\nabla_{\boldsymbol{m}} \ (\nabla_{\boldsymbol{x}} \, m_i) \approx \nabla_{\boldsymbol{m}} \ \boldsymbol{g}_i = \nabla_{\boldsymbol{m}} \Big(S_i G \boldsymbol{m} \Big) = \Big(\nabla_{\boldsymbol{m}} \, \boldsymbol{m} \Big) \big(S_i G \big)^T = G^T S_i^T. \tag{1.37}$$

We can write

$$\begin{aligned} \nabla_{\boldsymbol{m}} \ \big(\|\nabla_{\boldsymbol{x}} \, m_i\|_2^2 \big) \approx \nabla_{\boldsymbol{m}} \Big(\boldsymbol{g}_i^T \boldsymbol{g}_i \Big) &= 2 \Big(\nabla_{\boldsymbol{m}} \ \boldsymbol{g}_i \Big) \boldsymbol{g}_i \\ &= 2 \big(S_i G \big)^T \boldsymbol{g}_i = 2 \big(S_i G \big)^T \big(S_i G \big) \boldsymbol{m} \end{aligned} \tag{1.38}$$

which is a $M \times 1$ column vector. Moreover, by letting

$$z(m_i) = \|\nabla_{\boldsymbol{x}} \, m_i\|_2^2 \quad \text{and} \quad y_i = z(m_i) / \big(z(m_i) + \xi^2 \big)$$

(where ξ is ξ_3 or ξ_4), from the chain rule it's easy to see as in (1.14) that

$$\frac{\partial y_i}{\partial m_\ell} = \frac{\partial}{\partial m_\ell} \left(\frac{z(m_i)}{z(m_i) + \xi^2} \right) = \delta_{\ell i} \frac{\xi^2}{\big(z(m_i) + \xi^2 \big)^2} \frac{\partial}{\partial m_\ell} \Big(z(m_i) \Big) \qquad \ell, i = 1, \dots, M. \tag{1.39}$$

Thus we obtain

$$\begin{aligned} \nabla_{\boldsymbol{m}} \, y_i(\boldsymbol{m}) &= \Big(\nabla_{\boldsymbol{m}} \, z(m_i) \Big) \left(\frac{\mathrm{d}}{\mathrm{d}z} \, y_i(z) \right) \\ &\approx 2 \Big(\xi^2 \big(z(m_i) + \xi^2 \big)^{-2} \Big) G^T S_i^T S_i G \boldsymbol{m} \qquad i = 1, \dots, M. \end{aligned} \tag{1.40}$$

By substituting (1.40) in (1.33) we get for $j = 1, 2$,

$$\begin{aligned} \nabla_{\boldsymbol{m}^{(j)}} \, \boldsymbol{y}^{(j)} \approx \big[G^T S_1^T S_1 G \boldsymbol{m}^{(j)} \quad \dots \quad G^T S_M^T S_M G \boldsymbol{m}^{(j)} \big] \\ \cdot \operatorname{diag} \left(\Big(2 \xi_{j+2}^2 \big(\|\nabla_{\boldsymbol{x}} \, m_i^{(j)}\|_2^2 + \xi_{j+2}^2 \big)^{-2} \Big)_{i=1,\dots,M} \right) \end{aligned} \tag{1.41}$$

which is a $M \times M$ matrix, indeed. We can re-write (1.41) in a slightly different way with the goal of isolating $\boldsymbol{m}^{(j)}$: we define

$$D = \begin{pmatrix} S_1 G & & \\ & \ddots & \\ & & S_M G \end{pmatrix} = \operatorname{diag} \Big(\big(S_i G \big)_{i=1,\dots,M} \Big) \tag{1.42}$$

which is a $3M \times M^2$ (very sparse) matrix, so that $D^TD = \mathrm{diag}\big((G^TS_i^TS_iG)_{i=1,\dots,M}\big)$ and

$$\begin{aligned}\Big[G^TS_1^TS_1G\boldsymbol{m}^{(j)} \quad \dots \quad G^TS_M^TS_MG\boldsymbol{m}^{(j)}\Big] = \\ = \begin{pmatrix} I_M & \dots & I_M \end{pmatrix} \begin{pmatrix} G^TS_1^TS_1G & & \\ & \ddots & \\ & & G^TS_M^TS_MG \end{pmatrix} \begin{pmatrix} \boldsymbol{m}^{(j)} & & \\ & \ddots & \\ & & \boldsymbol{m}^{(j)} \end{pmatrix} \\ = \Big(\mathbf{1}_M^T \otimes I_M\Big) D^TD \Big(I_M \otimes \boldsymbol{m}^{(j)}\Big), \end{aligned} \tag{1.43}$$

where $\mathbf{1}_M = (1,\dots,1)^T$ is sized $M \times 1$ and I_M is the $M \times M$ identity matrix.

Thus, by substituting (1.32), (1.41) and (1.43) into (1.27) we finally have the expression for the gradient of the joining term with respect to the j-th model:

$$\nabla_{\boldsymbol{m}^{(j)}} MIX = 2\Big(\nabla_{\boldsymbol{m}^{(j)}}\, \boldsymbol{y}^{(j)}\Big)\Big(\nabla_{\boldsymbol{y}^{(j)}}\, \big(W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)})\big)\Big) W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \tag{1.44}$$

$$\begin{aligned} &\approx 2\Big[\Big(\mathbf{1}_M^T \otimes I_M\Big) D^TD \Big(I_M \otimes \boldsymbol{m}^{(j)}\Big) \mathrm{diag}\left(\Big(2\xi_{j+2}^2\big(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\big)^{-2}\Big)_{i=1,\dots,M}\right)\Big] \\ &\quad \cdot \Big[(-1)^{j-1}\xi_5^2 W_5^3\Big] W_5(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \\ &= (-1)^{j-1} 4\xi_{j+2}^2\xi_5^2 \Big[G^TS_1^TS_1G\boldsymbol{m}^{(j)} \quad \dots \quad G^TS_M^TS_MG\boldsymbol{m}^{(j)}\Big] \\ &\quad \cdot \mathrm{diag}\left(\Big(\big(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\big)^{-2}\Big)_{i=1,\dots,M}\right) W_5^4(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \end{aligned} \tag{1.45}$$

$$\begin{aligned} &= (-1)^{j-1} 4\xi_{j+2}^2\xi_5^2 \left[\Big(\mathbf{1}_M^T \otimes I_M\Big) D^TD \Big(I_M \otimes \boldsymbol{m}^{(j)}\Big)\right] \\ &\quad \cdot \mathrm{diag}\left(\Big(\big(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\big)^{-2}\Big)_{i=1,\dots,M}\right) W_5^4(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \end{aligned} \tag{1.46}$$

which is a $M \times 1$ column vector for each $j = 1, 2$. ■

We now compute the Hessian matrix of the joining functional. Differently from the case of the two independent regularization terms, we will see that the Hessian of the joining term is not (block) diagonal, because the gradients with respect to $\boldsymbol{m}^{(1)}$ and $\boldsymbol{m}^{(2)}$ depend on both the models through the term $\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}$. Moreover, in what follows we will look for derivatives with respect to the model, so there will be no need for discrete second order spatial derivatives and operators.

Lemma 1.2 *In the same hypothesis of Lemma 1.1 the Hessian matrix with respect to the model of the joining functional can be written as*

$$\nabla^2_{\boldsymbol{mm}} MIX \approx \gamma \begin{pmatrix} H_1^{(1,1)} + H_2^{(1,1)} + H_3^{(1,1)} & H_3^{(1,2)} \\ H_3^{(2,1)} & H_1^{(2,2)} + H_2^{(2,2)} + H_3^{(2,2)} \end{pmatrix} \tag{1.47}$$

where $\gamma \neq 0$ is a constant.

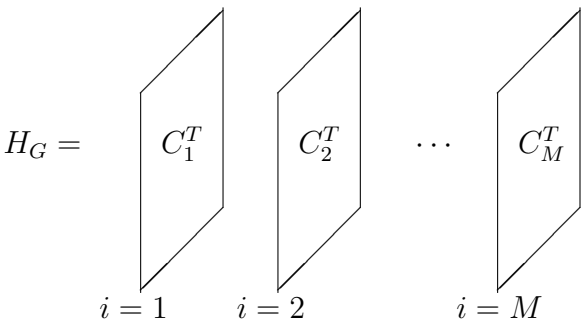


Figure 1.1: Pictorial representations of 3D arrays H_G.

Proof. Consider (1.45) and let $\gamma_j = (-1)^{j-1} 4 \xi_{j+2}^2 \xi_5^2$. By applying the chain rule we have:

$$\nabla^2_{\boldsymbol{m}^{(k)}\boldsymbol{m}^{(j)}} MIX = \nabla_{\boldsymbol{m}^{(k)}} \left(\nabla_{\boldsymbol{m}^{(j)}} MIX \right) \tag{1.48}$$

$$= \gamma_j \Bigg\{ \left(\nabla_{\boldsymbol{m}^{(k)}} \begin{bmatrix} G^T S_1^T S_1 G \boldsymbol{m}^{(j)} & \dots & G^T S_M^T S_M G \boldsymbol{m}^{(j)} \end{bmatrix} \right) \cdot \operatorname{diag} \left(\left(\left(\|\nabla_{\boldsymbol{x}} m_i^{(j)}\|_2^2 + \xi_{j+2}^2 \right)^{-2} \right)_{i=1,\dots,M} \right) W_5^4 (\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \tag{1.49}$$

$$+ \begin{bmatrix} G^T S_1^T S_1 G \boldsymbol{m}^{(j)} & \dots & G^T S_M^T S_M G \boldsymbol{m}^{(j)} \end{bmatrix} \cdot \left(\nabla_{\boldsymbol{m}^{(k)}} \operatorname{diag} \left(\left(\left(\|\nabla_{\boldsymbol{x}} m_i^{(j)}\|_2^2 + \xi_{j+2}^2 \right)^{-2} \right)_{i=1,\dots,M} \right) \right) \cdot W_5^4 (\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \tag{1.50}$$

$$+ \begin{bmatrix} G^T S_1^T S_1 G \boldsymbol{m}^{(j)} & \dots & G^T S_M^T S_M G \boldsymbol{m}^{(j)} \end{bmatrix} \cdot \operatorname{diag} \left(\left(\left(\|\nabla_{\boldsymbol{x}} m_i^{(j)}\|_2^2 + \xi_{j+2}^2 \right)^{-2} \right)_{i=1,\dots,M} \right) \cdot \nabla_{\boldsymbol{m}^{(k)}} \left(W_5^4 (\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \right) \Bigg\} \tag{1.51}$$

We begin by computing the derivatives in (1.49): here the gradient is applied to a matrix. From the tensor calculus we immediately have that the result is a three-dimensional array, but tensor calculus is *component-oriented* and it is sometimes not easy to understand the object structure. However, in this case it is not difficult to directly compute the derivatives, and the array products with the next matrices. To compute the whole gradient just consider that $\nabla_{\boldsymbol{m}^{(k)}}$ has to be applied to each separate column: it will then generate a matrix in the third array dimension. To simplify notation, we let $C_i = (S_i G)^T (S_i G)$, $i = 1, \dots, M$, and $H_G^{(k,j)} = \nabla_{\boldsymbol{m}^{(k)}} \begin{bmatrix} C_1 \boldsymbol{m}^{(j)} & \dots & C_M \boldsymbol{m}^{(j)} \end{bmatrix}$: the matrices C_i are very sparse matrices sized $M \times M$, whose elements come from the forward finite difference operators G_{x_t}, $t = 1, 2, 3$. We will look at their sparsity structure in Chapter 2.

Then, from $\nabla_{\boldsymbol{m}^{(k)}}\left(C_i\boldsymbol{m}^{(j)}\right) = \delta_{kj}C_i^T \ \forall i$ it follows

$$\left(H_G^{(k,j)}\right)_{*,i,*} = \delta_{kj}C_i^T$$

where $\left(H_G\right)_{*,i,*}$ is the i-th "layer" along the second direction of the three-dimensional array H_G, in Matlab-like notation (see Figure 1.1 for a pictorial representation). The matrix-vector product in (1.49) results in a column vector:

$$\boldsymbol{v}^{(j)} = \operatorname{diag}\left(\left(\left(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\right)^{-2}\right)_{i=1,\dots,M}\right) W_5^4(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)}) \qquad j = 1, 2,$$

with components

$$v_i^{(j)} = \frac{y_i^{(1)} - y_i^{(2)}}{\left(\left\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\right\|_2^2 + \xi_{j+2}^2\right)^2\left(\left(y_i^{(1)} - y_i^{(2)}\right)^2 + \xi_5^2\right)^2} \qquad i = 1, \dots, M.$$

Then, from the tensor calculus, the product of the 3D array H_G with the column vector $\boldsymbol{v}^{(j)}$ is well defined and is a $M \times M$ matrix

$$H_1^{(k,j)} = \delta_{kj}\left[\begin{array}{cccc} C_1^T\boldsymbol{v}^{(j)} & C_2^T\boldsymbol{v}^{(j)} & \dots & C_M^T\boldsymbol{v}^{(j)} \end{array}\right]. \tag{1.52}$$

Consider now the term in (1.50): once again, the gradient operator applied to the diagonal matrix results in a 3-dimensional array. Using the same reasoning and the same notation as before we can see that the operator $\nabla_{\boldsymbol{m}^{(k)}}$ applied to each column of the diagonal matrix generate a square $M \times M$ matrix in the third dimension of the 3D array. However, given that in the i-th column of the diagonal matrix only the i-th element is nonzero, in the generated matrix the only nonzero column is the i-th column. More precisely, using (1.38) we have, for all $i = 1, \dots, M$,

$$\begin{aligned} \nabla_{\boldsymbol{m}^{(k)}}\left(\left(\left\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\right\|_2^2 + \xi_{j+2}^2\right)^{-2}\right) &\approx -2\delta_{kj}\left(\left\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\right\|_2^2 + \xi_{j+2}^2\right)^{-3}\nabla_{\boldsymbol{m}^{(k)}}\left(\left\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\right\|_2^2\right) \\ &\approx -4\delta_{kj}\left(\left\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\right\|_2^2 + \xi_{j+2}^2\right)^{-3} C_i\boldsymbol{m}^{(j)} \\ &= \delta_{kj}\boldsymbol{w}^{(i,j)} \end{aligned} \tag{1.53}$$

which is an $M \times 1$ column vector. Once again, due to the sparsity of the matrices C_i, also the vectors $\boldsymbol{w}$, and hence the Hessian matrix, are very sparse: their sparsity structure is considered later in Chapter 2.

As before, by defining the three-dimensional array

$$H_D^{(k,j)} = \nabla_{\boldsymbol{m}^{(k)}}\left(\operatorname{diag}\left(\left(\left(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\right)^{-2}\right)_{i=1,\dots,M}\right)\right)$$

we have that its i-th layer along the second dimension is given by

$$\left(H_D^{(k,j)}\right)_{*,i,*} \approx \left[\begin{array}{ccccccc} \mathbf{0} & \dots & \mathbf{0} & \delta_{kj}\boldsymbol{w}^{(i,j)} & \mathbf{0} & \dots & \mathbf{0} \end{array}\right]$$

$$\uparrow$$

$$i\text{-th column}$$

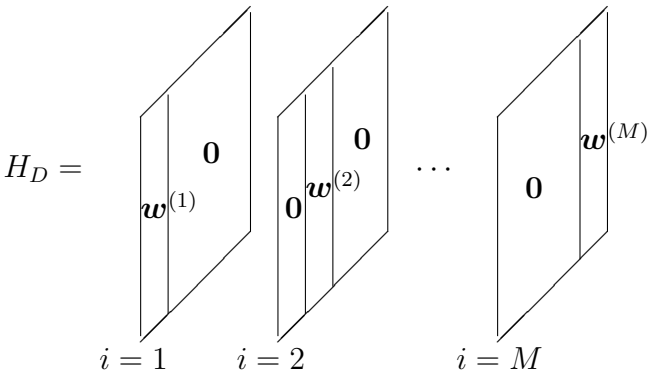


Figure 1.2: Pictorial representations of 3D arrays H_D.

which is a $M \times M$ matrix (see figure Figure 1.2 for a pictorial representation of H_D). Clearly, this matrix is identically zero for $k \neq j$.

Once again, we observe that $\boldsymbol{s} = W_5^4(\boldsymbol{y}^{(1)} - \boldsymbol{y}^{(2)})$ is a column vector with components

$$s_i = \frac{y_i^{(1)} - y_i^{(2)}}{\left(\left(y_i^{(1)} - y_i^{(2)}\right)^2 + \xi_5^2\right)^2} \qquad i = 1, \ldots, M.$$

By following the rules for tensor products we then have

$$H_D^{(k,j)}\boldsymbol{s} \approx \delta_{kj}\begin{bmatrix} s_1\boldsymbol{w}^{(1,j)} & s_2\boldsymbol{w}^{(2,j)} & \ldots & s_M\boldsymbol{w}^{(M,j)} \end{bmatrix}$$

which is a $M \times M$ matrix. It follows that (1.50) is well defined and is a $M \times M$ matrix:

$$\begin{aligned} H_2^{(k,j)} = \delta_{kj}&\begin{bmatrix} C_1\boldsymbol{m}^{(j)} & C_2\boldsymbol{m}^{(j)} & \ldots & C_M\boldsymbol{m}^{(j)} \end{bmatrix} \\ &\cdot \begin{bmatrix} s_1\boldsymbol{w}^{(1,j)} & s_2\boldsymbol{w}^{(2,j)} & \ldots & s_M\boldsymbol{w}^{(M,j)} \end{bmatrix}. \end{aligned} \tag{1.54}$$

Finally, consider the term in (1.51). By proceeding as in (1.29)–(1.31) we have

$$\begin{aligned} &\frac{\partial}{\partial y_\ell^{(j)}}\left(\frac{y_i^{(1)} - y_i^{(2)}}{\left(\left(y_i^{(1)} - y_i^{(2)}\right)^2 + \xi_5^2\right)^2}\right) \\ &\quad = \delta_{\ell i}\Big((y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2\Big)^{-4}\Big[(-1)^{j-1}\Big((y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2\Big)^2 \\ &\qquad\qquad - (y_i^{(1)} - y_i^{(2)})2\Big((y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2\Big)2\Big(y_i^{(1)} - y_i^{(2)}\Big)(-1)^{j-1}\Big] \\ &\quad = (-1)^{j-1}\delta_{\ell i}\Big((y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2\Big)^{-3}\Big[(y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2 - 4(y_i^{(1)} - y_i^{(2)})^2\Big] \\ &\quad = (-1)^{j-1}\delta_{\ell i}\frac{\xi_5^2 - 3\big(y_i^{(1)} - y_i^{(2)}\big)^2}{\Big((y_i^{(1)} - y_i^{(2)})^2 + \xi_5^2\Big)^3} \qquad j = 1, 2, \quad \ell, i = 1, \ldots, M. \end{aligned}$$

Hence

$$\nabla_{\boldsymbol{y}^{(j)}}\left(W_5^4\left(\boldsymbol{y}^{(1)}-\boldsymbol{y}^{(2)}\right)\right)=(-1)^{j-1}\mathrm{diag}\left(\left(\frac{\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2}{\big(\big(y_i^{(1)}-y_i^{(2)}\big)^2+\xi_5^2\big)^3}\right)_{i=1,\dots,M}\right)$$
$$=(-1)^{j-1}W_5^6\mathrm{diag}\left(\left(\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2\right)_{i=1,\dots,M}\right). \quad (1.55)$$

From (1.41) we also have

$$\nabla_{\boldsymbol{m}^{(k)}}\,\boldsymbol{y}^{(j)}\approx\delta_{kj}\left[C_1\boldsymbol{m}^{(j)}\quad\dots\quad C_M\boldsymbol{m}^{(j)}\right]\mathrm{diag}\left(\left(2\xi_{j+2}^2\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(j)}\|_2^2+\xi_{j+2}^2\big)^{-2}\right)_{i=1,\dots,M}\right). \quad (1.56)$$

Thus we get

$$\begin{aligned}\nabla_{\boldsymbol{m}^{(k)}}\left(W_5^4(\boldsymbol{y}^{(1)}-\boldsymbol{y}^{(2)})\right)\approx{}&\left[C_1\boldsymbol{m}^{(k)}\quad\dots\quad C_M\boldsymbol{m}^{(k)}\right]\\&\cdot\mathrm{diag}\left(\left(2\xi_{k+2}^2\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(k)}\|_2^2+\xi_{k+2}^2\big)^{-2}\right)_{i=1,\dots,M}\right)\\&\cdot(-1)^{k-1}W_5^6\mathrm{diag}\left(\left(\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2\right)_{i=1,\dots,M}\right)\\={}&(-1)^{k-1}2\xi_{k+2}^2\left[C_1\boldsymbol{m}^{(k)}\quad\dots\quad C_M\boldsymbol{m}^{(k)}\right]W_5^6\\&\cdot\mathrm{diag}\left(\left(\frac{\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2}{\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(k)}\|_2^2+\xi_{k+2}^2\big)^2}\right)_{i=1,\dots,M}\right)\end{aligned} \quad (1.57)$$

which is a $M\times M$ matrix. Using (1.57) in (1.51) gives

$$\begin{aligned}H_3^{(k,j)}={}&\left[C_1\boldsymbol{m}^{(j)}\quad\dots\quad C_M\boldsymbol{m}^{(j)}\right]\\&\cdot\mathrm{diag}\left(\left(\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(j)}\|_2^2+\xi_{j+2}^2\big)^{-2}\right)_{i=1,\dots,M}\right)\\&\cdot(-1)^{k-1}2\xi_{k+2}^2\left[C_1\boldsymbol{m}^{(k)}\quad\dots\quad C_M\boldsymbol{m}^{(k)}\right]W_5^6\\&\cdot\mathrm{diag}\left(\left(\frac{\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2}{\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(k)}\|_2^2+\xi_{j+2}^2\big)^2}\right)_{i=1,\dots,M}\right)\\={}&(-1)^{k-1}2\xi_{k+2}^2\\&\cdot\left[C_1\boldsymbol{m}^{(j)}\quad\dots\quad C_M\boldsymbol{m}^{(j)}\right]\mathrm{diag}\left(\left(\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(j)}\|_2^2+\xi_{j+2}^2\big)^{-2}\right)_{i=1,\dots,M}\right)\\&\cdot\left[C_1\boldsymbol{m}^{(k)}\quad\dots\quad C_M\boldsymbol{m}^{(k)}\right]\mathrm{diag}\left(\left(\big(\|\nabla_{\boldsymbol{x}}\,m_i^{(k)}\|_2^2+\xi_{k+2}^2\big)^{-2}\right)_{i=1,\dots,M}\right)\\&\cdot W_5^6\mathrm{diag}\left(\left(\xi_5^2-3\big(y_i^{(1)}-y_i^{(2)}\big)^2\right)_{i=1,\dots,M}\right)\end{aligned}$$

where

$$W_5^6 \mathrm{diag}\left(\xi_5^2 - 3\big(y_i^{(1)} - y_i^{(2)}\big)^2\right) = \mathrm{diag}\left(\left(\frac{\xi_5^2 - 3\big(y_i^{(1)} - y_i^{(2)}\big)^2}{\left(\big(y_i^{(1)} - y_i^{(2)}\big)^2 + \xi_5^2\right)^3}\right)_{i=1,\ldots,M}\right). \quad (1.58)$$

It follows that

$$H_3^{(j,j)} \approx (-1)^{j-1} 2\xi_{j+2}^2 \left(\left[C_1 \boldsymbol{m}^{(j)} \quad \ldots \quad C_M \boldsymbol{m}^{(j)}\right] \cdot \mathrm{diag}\left(\left(\|\nabla_{\boldsymbol{x}}\, m_i^{(j)}\|_2^2 + \xi_{j+2}^2\right)^{-2}\right)\right)^2$$
$$\cdot\, W_5^6 \mathrm{diag}\left(\left(\xi_5^2 - 3\big(y_i^{(1)} - y_i^{(2)}\big)^2\right)_{i=1,\ldots,M}\right). \quad (1.59)$$

Hence all $H_3^{(k,j)}$ are $M \times M$ square matrices and we have in general $H_3^{(k,j)} \neq \mathbf{0}$ also for $k \neq j$.

We then have the final expression of the Hessian of the mixing term with respect to the models:

$$\nabla^2_{\boldsymbol{mm}} MIX \approx \gamma \left\{ \begin{pmatrix} H_1^{(1,1)} & \mathbf{0} \\ \mathbf{0} & H_1^{(2,2)} \end{pmatrix} + \begin{pmatrix} H_2^{(1,1)} & \mathbf{0} \\ \mathbf{0} & H_2^{(2,2)} \end{pmatrix} + \begin{pmatrix} H_3^{(1,1)} & H_3^{(1,2)} \\ H_3^{(2,1)} & H_3^{(2,2)} \end{pmatrix} \right\} \quad (1.60)$$

$$= \gamma \begin{pmatrix} H_1^{(1,1)} + H_2^{(1,1)} + H_3^{(1,1)} & H_3^{(1,2)} \\ H_3^{(2,1)} & H_1^{(2,2)} + H_2^{(2,2)} + H_3^{(2,2)} \end{pmatrix} \quad (1.61)$$

which is a $2M \times 2M$ matrix with each block dependent on both $\boldsymbol{m}^{(1)}$ and $\boldsymbol{m}^{(2)}$, as expected. ■

Chapter 2

Sparsity structures of JoInv matrices

The dynamic process of allocating new memory and copying from the old storage area to the new one is intrinsically very expensive. Thus, to obtain good performance when assembling an AIJ matrix, it is crucial to preallocate the memory needed for the sparse matrix. In this context it is necessary to know and to understand the structure of the matrices we are going to use. Most of the JoInv matrices are very sparse because their structure comes from the stencil of finite difference problems, so a good preallocation can avoid the use of unnecessary extra memory and improve the performance. We can know in advance the sparsity structure by precomputing the information using a few lines of code; the overhead of determining the nonzero structure is quite small compared to the overhead of the inherently expensive `malloc` operations and data movements that are needed for dynamic allocation during matrix assembly.
We will return on structure prediction in Chapter 4.

To visualize and better understand the sparsity structure of the matrices that we use, we show and discuss here the pattern of some of them, for both those coming from forward finite differences and from central finite differences. All these figures are generated by supposing that the domain is rectangular and it is discretized in rectangular cells. Moreover, we suppose these cells are numbered in a *natural* way, that is first by x_1, than by x_2 and finally by x_3. The volume used to generate the plots has $5 \times 4 \times 3$ cells and it is shown in Figure 2.1.

2.1 Sparsity structures of JoInv matrices using Forward Differences

2.1.1 Sparsity structures of discrete operators

First of all we consider the form of the discrete spatial derivative operators G_{x_t}, $t = 1, 2, 3$. As we said before, we suppose that the domain is a parallelepiped, large enough to contain the whole volume of interest, and that it is discretized with parallelepiped cells. Moreover, we suppose these cells are numbered in a *natural* way, that is first by x_1, then by x_2 and finally by x_3. We additionally assume here that the discrete derivatives inside each cell are constant and that their values can be approximated by the values computed at the cells center.

Then, the forward finite difference operators in the three spatial directions are com-

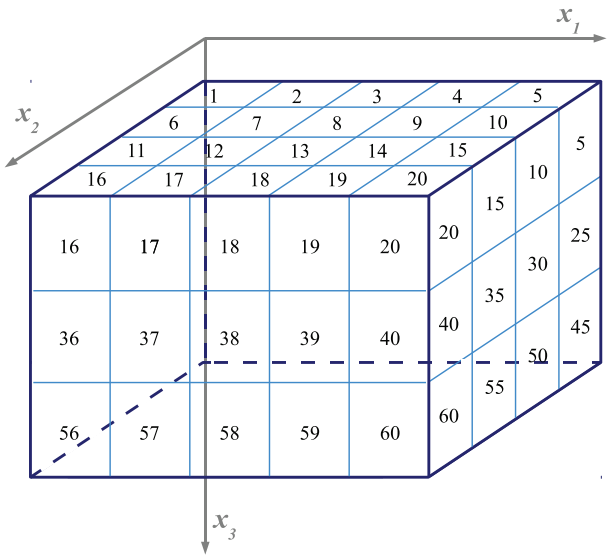


Figure 2.1: 3D volume used to generate the sparsity structures of the JoInv matrices shown in the next spy plots.

pletely defined for all the cells, but those located at the boundary positions of the domain: precisely, these cells are those belonging to the three "boundary faces" that do not contain the cell number 1. How to deal with these positions is a matter of boundary conditions. Among the possible choices, each of which affects some elements of the discrete derivatives operators, we choose here the *constant boundary conditions*: we thus assume that at the boundary faces the model does not change. It means that the spatial (discrete) gradient at these boundary cells has a zero component in the direction towards the exterior of the domain.

Under these hypothesis, we can easily derive the form of the discrete operator G. For simplicity's sake we consider a cell grid which is uniform in each spatial direction (even if the size can be different in the three directions): we call $h_t = h_{x_t}$, $t = 1, 2, 3$, the three constant cell sizes. Moreover, let's call $L_t = L_{x_t}$, $t = 1, 2, 3$, the number of cells in each spatial direction. Clearly, $L = L_1 L_2 L_3$ is the total number of cells in the domain. Consider now the first spatial direction, x_1: the L_1 adjacent cells in this directions have consecutive numbers in the chosen ordering, so the discrete derivative is given by

$$\frac{\partial m_i(\boldsymbol{x})}{\partial x_1} \approx \Delta_{x_1} m_i = \frac{m_{i+1} - m_i}{h_1}$$

for all the $L_1 - 1$ cells, but the last one, where it is zero. Hence the discrete partial derivative operator for these cells can be written as

$$B_{x_1} = \frac{1}{h_1} \underbrace{\begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \\ & & & & 0 \end{pmatrix}}_{L_1 \times L_1} \Bigg\} L_1 - 1 \text{ rows} \tag{2.1}$$

By repeating the reasoning for all the other adjacent cells along the first spatial direction, we have the discrete partial derivative operator for the whole domain as a block diagonal

matrix

$$G_{x_1} = \begin{pmatrix} B_{x_1} & & & \\ & B_{x_1} & & \\ & & \ddots & \\ & & & B_{x_1} \end{pmatrix}_{L \times L} \tag{2.2}$$

with $L_2 L_3$ blocks. It follows that $\Delta_{x_1} \boldsymbol{m}(\boldsymbol{x}) = G_{x_1} \boldsymbol{m}$.

Consider now the second spatial direction, x_2: here the difference between the indices of two adjacent cells is L_1, for all cells but those of the boundary face with largest x_2, where the partial discrete derivative will be zero. For the former cells we thus have

$$\frac{\partial\, m_i(\boldsymbol{x})}{\partial x_2} \approx \Delta_{x_2} m_i = \frac{m_{i+L_1} - m_i}{h_2}\,.$$

Thus, we can write the discrete partial derivative operator for all the cells in each "layer" of the cell grid parallel to the $x_1 x_3$ plane as

$$B_{x_2} = \frac{1}{h_2} \begin{pmatrix} -I_{L_1} & I_{L_1} & & & \\ & -I_{L_1} & I_{L_1} & & \\ & & \ddots & \ddots & \\ & & & -I_{L_1} & I_{L_1} \\ & & & & \mathbf{0}_{L_1} \end{pmatrix} \Bigg\} L_2 - 1 \text{ block rows} \tag{2.3}$$
$$(L_1 L_2) \times (L_1 L_2)$$

where I_{L_1} and $\mathbf{0}_{L_1}$ are the $L_1 \times L_1$ identity matrix and null matrix, respectively. By repeating the reasoning for all the other "layers" of the cell grid parallel to the $x_1 x_3$ plane one has the discrete partial derivative operator in the second direction for the whole domain, again as a block diagonal matrix

$$G_{x_2} = \begin{pmatrix} B_{x_2} & & & \\ & B_{x_2} & & \\ & & \ddots & \\ & & & B_{x_2} \end{pmatrix}_{L \times L} \tag{2.4}$$

with L_3 blocks. It follows that $\Delta_{x_2} \boldsymbol{m}(\boldsymbol{x}) = G_{x_2} \boldsymbol{m}$.

Last, consider the third spatial direction, x_3: the difference between the indices of adjacent cells in this direction is now $L_1 L_2$, for all cells but those of the boundary face with largest $|x_2|$, where the partial discrete derivative will be zero (using the modulus the discussion applies to both the upside and the downside ordering modes). For the former cells we have

$$\frac{\partial\, m_i(\boldsymbol{x})}{\partial x_3} \approx \Delta_{x_3} m_i = \frac{m_{i+L_1 L_2} - m_i}{h_3}\,.$$

The discrete partial derivative operator for all the cells in each "layer" of the cell grid parallel to the $x_1 x_2$ plain as

$$B_{x_3} = \frac{1}{h_3} \begin{pmatrix} -I_{L_1 L_2} & I_{L_1 L_2} \end{pmatrix}_{(L_1 L_2) \times 2(L_1 L_2)} \tag{2.5}$$

where $I_{L_1 L_2}$ is the identity matrix sized $(L_1 L_2) \times (L_1 L_2)$. By repeating the reasoning for all the other "layers" of the cell grid parallel to the $x_1 x_2$ plain we obtain the discrete

partial derivative operator in the third spatial direction for the whole domain, as a block upper bidiagonal matrix

$$
\begin{aligned}
G_{x_3} &= \begin{pmatrix} B_{x_3} & & & & \\ & B_{x_3} & & & \\ & & \ddots & & \\ & & & B_{x_3} & \\ & & & & \mathbf{0}_{L_1L_2} \end{pmatrix}_{L\times L} \\
&= \begin{pmatrix} -I & I & & & \\ & -I & I & & \\ & & \ddots & \ddots & \\ & & & -I & I \\ & & & & \mathbf{0} \end{pmatrix}_{L\times L} \Bigg\} L_3 - 1 \text{ block rows}
\end{aligned}
\tag{2.6}
$$

where $\mathbf{0}_{L_1L_2}$ is the $(L_1L_2)\times(L_1L_2)$ null matrix. It follows that $\Delta_{x_3}\boldsymbol{m}(\boldsymbol{x}) = G_{x_3}\boldsymbol{m}$.

Note that in case of nonuniform rectangular grids everything remains the same, except that the cell size h_t cannot be grouped anymore: instead, the nonzero elements $+1$ and -1 in the operators have to be substituted with the reciprocal of the distance between the centers of the corresponding adjacent cells in that direction. Moreover, note that usually in the actual computations only those cells which are effectively needed in the solution of the inverse problem are considered, all the other being neglected. The number M of these "selected" cells determines the model size in the minimization process. When computing the discrete spatial gradient, neighboring cells are involved to compute the discrete derivatives: hence, in this steps the full-length model vector is always needed.

Figure 2.2 shows the sparsity structures of the discrete spatial derivative operator G_{x_t}.

2.1.2 Sparsity structure of matrices C_i

We now describe the sparsity structure of matrices C_i, $i = 1, \dots, M$.

Once again, to help the understanding we sketch the structures on the toy example depicted in Figure 2.1.

According to (1.34), the spatial gradient $\nabla_{\boldsymbol{x}}\,\boldsymbol{m}$ becomes

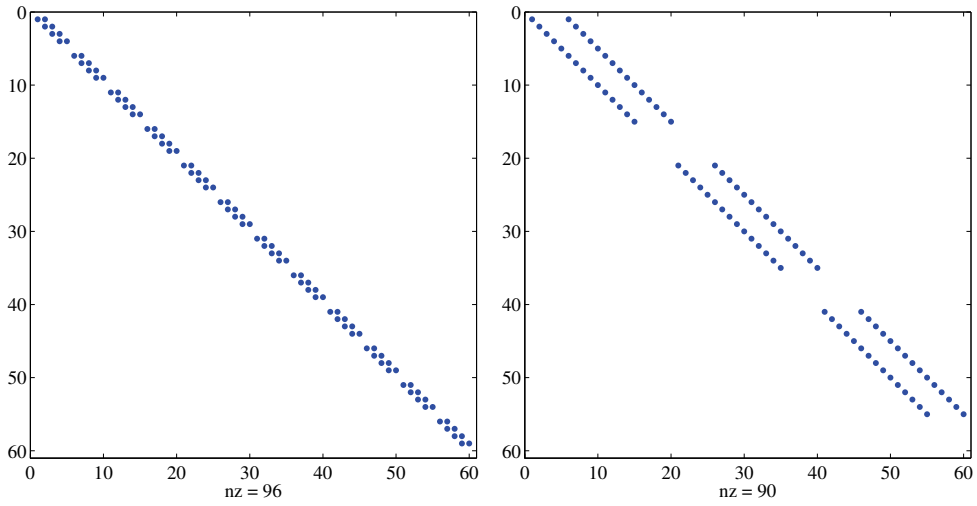
$$
\nabla_{\boldsymbol{x}}\,\boldsymbol{m} = \begin{pmatrix} \frac{\partial}{\partial x_1}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_1}m_2(\boldsymbol{x}) & \dots & \frac{\partial}{\partial x_1}m_{36}(\boldsymbol{x}) \\ \frac{\partial}{\partial x_2}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_2}m_2(\boldsymbol{x}) & \dots & \frac{\partial}{\partial x_2}m_{36}(\boldsymbol{x}) \\ \frac{\partial}{\partial x_3}m_1(\boldsymbol{x}) & \frac{\partial}{\partial x_3}m_2(\boldsymbol{x}) & \dots & \frac{\partial}{\partial x_3}m_{36}(\boldsymbol{x}) \end{pmatrix} \approx \begin{pmatrix} \big(G_{x_1}\boldsymbol{m}(\boldsymbol{x})\big)^T \\ \big(G_{x_2}\boldsymbol{m}(\boldsymbol{x})\big)^T \\ \big(G_{x_3}\boldsymbol{m}(\boldsymbol{x})\big)^T \end{pmatrix}
\tag{2.7}
$$

where G_{x_i} is the $M\times M$ (*i.e.* 60×60) matrix of the forward finite difference approximation of the model derivative with respect to the spatial coordinates x_t.

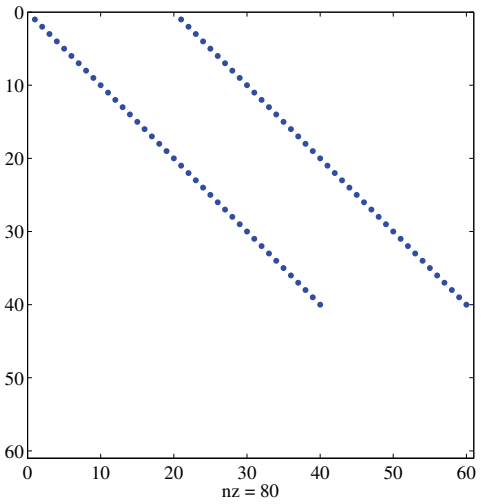
We also recall the selection matrix S_i in (1.35), sized $3\times 3M = 3\times 3\cdot 60$:

$$
S_i = \begin{pmatrix} \mathbf{e}_i^T \\ \mathbf{e}_{60+i}^T \\ \mathbf{e}_{2\cdot 60+i}^T \end{pmatrix} = \left(\begin{array}{c|c|c} 0\ \dots 1 \dots\ 0 & 0\ \dots\dots\ 0 & 0\ \dots\dots\ 0 \\ 0\ \dots\dots\ 0 & 0\ \dots 1 \dots\ 0 & 0\ \dots\dots\ 0 \\ 0\ \dots\dots\ 0 & 0\ \dots\dots\ 0 & 0\ \dots 1 \dots\ 0 \end{array}\right),
\tag{2.8}
$$

$$
\begin{array}{ccc} \uparrow & \uparrow & \uparrow \\ i & 60+i & 2\cdot 60+i \end{array}
$$



(a) Operator G_{x_1} (b) Operator G_{x_2}



(c) Operator G_{x_3}

Figure 2.2: Sparsity structures of the discrete spatial derivative operator G_{x_t} (forward difference).

so that S_iG is a $3 \times M$ (3×60) matrix and $\boldsymbol{g}_i$ is a 3×1 column vector, for all $i = 1, \ldots 60$.

Moreover, remember that we define $C_i = (S_iG)^T(S_iG)$, $i = 1, \ldots, 60$ and we call $L_1 = 5$, $L_2 = 4$, $L_3 = 3$, the number of cells in each spatial direction. Clearly, $L = L_1L_2L_3 = 60$ is the total number of cells in the domain.

The discrete partial derivative operator of our example with respect to x_1, for the first L_1 cells can be written as

$$B_{x_1} = \frac{1}{h_1} \underbrace{\begin{pmatrix} -1 & 1 & & & \\ & -1 & 1 & & \\ & & -1 & 1 & \\ & & & -1 & 1 \\ & & & & 0 \end{pmatrix}}_{5\times 5} \Bigg\} 5-1 \text{ rows} \tag{2.9}$$

By repeating the reasoning for all the other adjacent cells along the first spatial di-

rection, one has the discrete partial derivative operator for the whole domain as a block diagonal matrix

$$G_{x_1} = \begin{pmatrix} \begin{matrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} & & & \\ & \ddots & & \\ & & \begin{matrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \end{pmatrix}_{60\times 60} \tag{2.10}$$

By multiplying G_{x_1} and $\boldsymbol{m}$ we get a 40×1 column vector that in the i-th cell contains $\dfrac{m_{i+1} - m_i}{h_1}$.

In the same way, we can write the other matrices B_{x_2}, G_{x_2}, B_{x_3}, G_{x_3} as:

$$B_{x_2} = \frac{1}{h_2} \begin{pmatrix} \begin{matrix} -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \end{matrix} & & \\ & \ddots & \\ & & \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \end{pmatrix}_{20\times 20} \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} 4-1 \text{ block rows} \tag{2.11}$$

$$G_{x_2} = \begin{pmatrix} B_{x_2} & & & \\ & B_{x_2} & & \\ & & B_{x_2} & \\ & & & B_{x_2} \end{pmatrix}_{60\times 60} \tag{2.12}$$

$$B_{x_3} = \frac{1}{h_3} \begin{pmatrix} -1 & & & & 1 & & \\ & -1 & & & & 1 & \\ & & \ddots & & & & \ddots \\ & & & -1 & & & & 1 \end{pmatrix}_{20\times 40} \tag{2.13}$$

$$G_{x_3} = \begin{pmatrix} B_{x_3} & & \\ & B_{x_3} & \\ & & 0 \end{pmatrix}_{60\times 60} \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} 3-1 \text{ block rows} \tag{2.14}$$

Now, for each $i = 1 \dots M$, we can compute $S_i G$, that is a $(3 \times 108) \cdot (108 \times 36) = 3 \times 36$ sparse matrix.

Let's return to the explicit form of the C_i sparse matrices. For each i-th cell, we can calculate S_iG, that is a $(3 \times 3M) \cdot (3M \times M) = 3 \times M$ sparse matrix.

In the first row of S_iG, only the columns with indices i and $i+1$ have nonzero elements, namely -1 and 1, respectively, for $i \neq kL_1$, with $k = 1, \ldots, L_2L_3$. In other words, this is true for each cell which is not at the border on the right in the x_1 direction: we refer to this situation as *Property 1*. By the common Matlab-like notation, this row can also be written as

$$\big(S_i\big)_{1,*}G = \big(S_i\big)_{1,1:M}G_{x_1}$$

where $\big(S_i\big)_{1,*}$ is the first row of S_i and $\big(S_i\big)_{1,1:M}$ is its sub-vector having just the first M columns.

In the second row of S_iG the only nonzero columns are the i-th,and the $(i + L_1)$-th, containing -1 and 1, respectively, for $i \neq \{k(L_1L_2),\, k(L_1L_2) - 1,\, k(L_1L_2) - 2\}$, $k = 1, \ldots, L_3$. Again this sparsity pattern holds true for each cell that is not at the right border in the x_2 direction. We name this situation as *Property 2*.

Finally, the nonzero elements of the third row of S_iG are positioned in columns i and $i + L_1L_2$ with values -1 and 1, respectively, for $i \neq (L_1L_2L_3) - k$, $k = 0, \ldots, (L_1L_2) - 1$. We refer to this sparsity pattern as *Property 3*, which is common to all cells that are not at the right border in the x_3 direction.

Analogously to the first row, the second and third rows of S_iG can be written as

$$\big(S_i\big)_{2,*}G = \big(S_i\big)_{2,M+1:2M}G_{x_2} \quad \text{and} \quad \big(S_i\big)_{3,*}G = \big(S_i\big)_{3,2M+1:3M}G_{x_3}\,.$$

Now we can see how the $M \times M$ sparse matrices $C_i = (S_iG)^T(S_iG)$, $i = 1, \ldots, M$, look like. Let's consider a cell with index i which is not at the right border of any direction. The elements of the matrix C_i are the following:

$$\big(C_i\big)_{i,i} = 1/h_1^2 + 1/h_2^2 + 1/h_3^2 \tag{2.15a}$$
$$\big(C_i\big)_{k,k} = 1/h_1^2, \qquad k = i + 1, \tag{2.15b}$$
$$\big(C_i\big)_{k,k} = 1/h_2^2, \qquad k = i + L_1, \tag{2.15c}$$
$$\big(C_i\big)_{k,k} = 1/h_3^2, \qquad k = i + L_1L_2, \tag{2.15d}$$
$$\big(C_i\big)_{k,i} = \big(C_i\big)_{i,k} = -1/h_1^2, \qquad k = i + 1, \tag{2.15e}$$
$$\big(C_i\big)_{k,i} = \big(C_i\big)_{i,k} = -1/h_2^2, \qquad k = i + L_1, \tag{2.15f}$$
$$\big(C_i\big)_{k,i} = \big(C_i\big)_{i,k} = -1/h_3^2, \qquad k = i + L_1L_2. \tag{2.15g}$$

For instance, consider the case $i = 1$: the 60×60 matrix $C_1 = (S_1G)^T(S_1G)$ becomes

$$C_1 = \begin{array}{c} \begin{matrix} \quad (i) & \quad (i+1) & \quad (i+L_1) & \quad (i+L_1L_2) \\ \quad\downarrow & \quad\downarrow & \quad\downarrow & \quad\downarrow \end{matrix} \\ \begin{pmatrix} \vdots & \vdots & \vdots & \vdots \\ \ldots \;\; 1/h_1^2 + 1/h_2^2 + 1/h_3^2 & -1/h_1^2 \;\; \ldots & -1/h_2^2 \;\; \ldots & -1/h_3^2 \\ \ldots \;\; -1/h_1^2 & 1/h_1^2 & & \\ \vdots & & \ddots & \\ \ldots \;\; -1/h_2^2 & & 1/h_2^2 & \\ \vdots & & & \ddots \\ \ldots \;\; -1/h_3^2 & & & 1/h_3^2 \end{pmatrix} \end{array} \begin{array}{l} \\ \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} \tag{2.16}$$

where the only nonzero elements are explicitly reported (that is, all the other positions in the matrix contain zeroes, included those marked by dots). This sparsity pattern is shared by all domain cells such that no one of the Properties 1, 2 and 3 holds true, that we call *Group 1*.

Let's consider now the case of a cell i at the border only of the x_1 direction, that is Property 1 is true, while properties 2 and 3 are false. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 2*. For the sparse matrix C_i we can write:

$$\begin{aligned}
\left(C_i\right)_{i,i} &= 1/h_2^2 + 1/h_3^2 && && (2.17a)\\
\left(C_i\right)_{k,k} &= 1/h_2^2, && k = i + L_1, && (2.17b)\\
\left(C_i\right)_{k,k} &= 1/h_3^2, && k = i + L_1 L_2, && (2.17c)\\
\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} &= -1/h_2^2, && k = i + L_1, && (2.17d)\\
\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} &= -1/h_3^2, && k = i + L_1 L_2. && (2.17e)
\end{aligned}$$

For instance, for the cell $i = 10$ of the our example volume the 60×60 sparse matrix $C_3 = (S_3 G)^T (S_3 G)$ is as follows:

$$C_3 = \begin{array}{c}
\begin{array}{ccccccc}
\quad (i) & & (i+1) & & (i+L_1) & & (i+L_1L_2) \\
\quad \downarrow & & \downarrow & & \downarrow & & \downarrow \\
\quad \vdots & & \vdots & & \vdots & & \vdots
\end{array} \\
\begin{pmatrix}
\cdots & 1/h_2^2 + 1/h_3^2 & 0 & \cdots & -1/h_2^2 & \cdots & -1/h_3^2 \\
\cdots & 0 & 0 & & & & \\
 & \vdots & & \ddots & & & \\
\cdots & -1/h_2^2 & & & 1/h_2^2 & & \\
 & \vdots & & & & \ddots & \\
\cdots & -1/h_3^2 & & & & & 1/h_3^2
\end{pmatrix}
\begin{array}{l}
\leftarrow (i) \\
\leftarrow (i+1) \\
 \\
\leftarrow (i+L_1) \\
 \\
\leftarrow (i+L_1L_2)
\end{array}
\end{array} . \tag{2.18}$$

Being a border cell, there are additional zero elements with respect to C_1 in (2.16).

Consider now a cell i which is at the right border in the x_2 direction only, that is Property 1 and 3 are not satisfied, while Property 2 holds true. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 3*. The elements of the matrix C_i in this case can written as:

$$\begin{aligned}
\left(C_i\right)_{i,i} &= 1/h_1^2 + 1/h_3^2 && && (2.19a)\\
\left(C_i\right)_{k,k} &= 1/h_1^2, && k = i + 1, && (2.19b)\\
\left(C_i\right)_{k,k} &= 1/h_3^2, && k = i + L_1 L_2, && (2.19c)\\
\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} &= -1/h_1^2, && k = i + 1, && (2.19d)\\
\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} &= -1/h_3^2, && k = i + L_1 L_2. && (2.19e)
\end{aligned}$$

For instance, the cell with index $i = 36$ of the our example volume has the following

sparse matrix $C_7 = (S_7G)^T(S_7G)$:

$$C_7 = \begin{array}{c} \begin{array}{ccccccc} & (i) & (i+1) & & (i+L_1) & & (i+L_1L_2) \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \\ & \vdots & \vdots & & \vdots & & \vdots \end{array} \\ \begin{pmatrix} \dots & 1/h_1^2+1/h_3^2 & -1/h_1^2 & \dots & 0 & \dots & -1/h_3^2 \\ \dots & -1/h_1^2 & 1/h_1^2 & & & & \\ & \vdots & & \ddots & & & \\ \dots & 0 & & & 0 & & \\ & \vdots & & & & \ddots & \\ \dots & -1/h_3^2 & & & & & 1/h_3^2 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} . \tag{2.20}$$

Analogously, for a cell i which is at the right border in the x_3 direction only we have that Property 3 holds true and properties 1 and 2 are not satisfied. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 4*. The elements of the matrix C_i in this case can written as:

$$\left(C_i\right)_{i,i} = 1/h_1^2 + 1/h_2^2 \tag{2.21a}$$
$$\left(C_i\right)_{k,k} = 1/h_1^2, \qquad k = i+1, \tag{2.21b}$$
$$\left(C_i\right)_{k,k} = 1/h_2^2, \qquad k = i+L_1, \tag{2.21c}$$
$$\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} = -1/h_1^2, \qquad k = i+1, \tag{2.21d}$$
$$\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} = -1/h_2^2, \qquad k = i+L_1. \tag{2.21e}$$

For instance, the cell with index $i = 47$ in the toy example has the following sparse matrix $C_{47} = (S_{47}G)^T(S_{47}G)$:

$$C_{32} = \begin{array}{c} \begin{array}{ccccccc} & (i) & (i+1) & & (i+L_1) & & (i+L_1L_2) \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \\ & \vdots & \vdots & & \vdots & & \vdots \end{array} \\ \begin{pmatrix} \dots & 1/h_1^2+1/h_2^2 & -1/h_1^2 & \dots & -1/h_2^2 & \dots & 0 \\ \dots & -1/h_1^2 & 1/h_1^2 & & & & \\ & \vdots & & \ddots & & & \\ \dots & -1/h_2^2 & & & 1/h_2^2 & & \\ & \vdots & & & & \ddots & \\ \dots & 0 & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} . \tag{2.22}$$

There are now to consider the cases of cells that are simultaneously at the right border of multiple coordinate directions. We consider first those cells that are at the right border of x_1 and x_2 directions. In these cases the properties 1 and 2 are both satisfied, while Property 3 is not. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 5*. The nonzero elements in the corresponding matrices C_i are

the following:

$$\left(C_i\right)_{i,i} = 1/h_3^2 \tag{2.23a}$$

$$\left(C_i\right)_{k,k} = 1/h_3^2, \quad k = i + L_1L_2, \tag{2.23b}$$

$$\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} = -1/h_3^2, \quad k = i + L_1L_2. \tag{2.23c}$$

In our example, one of these cells has index $i = 20$, for which the sparse matrix $C_{20} = (S_{20}G)^T(S_{20}G)$ is:

$$C_9 = \begin{array}{c} \begin{array}{ccccccc} & (i) & (i+1) & & (i+L_1) & & (i+L_1L_2) \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \end{array} \\ \begin{pmatrix} & \vdots & \vdots & & \vdots & & \vdots \\ \dots & 1/h_3^2 & 0 & \dots & 0 & \dots & -1/h_3^2 \\ \dots & 0 & 0 & & & & \\ & \vdots & & \ddots & & & \\ \dots & 0 & & & 0 & & \\ & \vdots & & & & \ddots & \\ \dots & -1/h_3^2 & & & & & 1/h_3^2 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} . \tag{2.24}$$

We then consider the cells that are at the right border of x_1 and x_3 directions, where properties 1 and 3 are both satisfied, while Property 2 is not. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 6*. The nonzero elements in the corresponding matrices C_i are the following:

$$\left(C_i\right)_{i,i} = 1/h_2^2 \tag{2.25a}$$

$$\left(C_i\right)_{k,k} = 1/h_2^2, \quad k = i + L_1, \tag{2.25b}$$

$$\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} = -1/h_2^2, \quad k = i + L_1. \tag{2.25c}$$

In the example, one of these cells has index $i = 50$ and the corresponding sparse matrix $C_{50} = (S_{50}G)^T(S_{50}G)$ looks like

$$C_{33} = \begin{array}{c} \begin{array}{ccccccc} & (i) & (i+1) & & (i+L_1) & & (i+L_1L_2) \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \end{array} \\ \begin{pmatrix} & \vdots & \vdots & & \vdots & & \vdots \\ \dots & 1/h_2^2 & 0 & \dots & -1/h_2^2 & \dots & 0 \\ \dots & 0 & 0 & & & & \\ & \vdots & & \ddots & & & \\ \dots & -1/h_2^2 & & & 1/h_2^2 & & \\ & \vdots & & & & \ddots & \\ \dots & 0 & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} . \tag{2.26}$$

Considering now the cells that are at the right border of x_2 and x_3 directions, where properties 2 and 3 are both satisfied and Property 1 hold false, the nonzero elements in the corresponding matrices C_i are the following:

$$\left(C_i\right)_{i,i} = 1/h_1^2 \tag{2.27a}$$

$$\left(C_i\right)_{k,k} = 1/h_1^2, \quad k = i + 1, \tag{2.27b}$$

$$\left(C_i\right)_{k,i} = \left(C_i\right)_{i,k} = -1/h_1^2, \quad k = i + 1. \tag{2.27c}$$

We say that the matrices C_i resulted from a cell i with those properties belong to *Group 7*. In our example, the cell with index $i = 58$ is in such a position, thus its matrix $C_{58} = (S_{58}G)^T(S_{58}G)$ is:

$$C_{35} = \begin{array}{c} \begin{array}{ccccccc} & (i) & (i+1) & & (i+L_1) & & (i+L_1L_2) \\ & \downarrow & \downarrow & & \downarrow & & \downarrow \end{array} \\ \begin{pmatrix} & \vdots & \vdots & & \vdots & & \vdots \\ \dots & 1/h_1^2 & -1/h_1^2 & \dots & 0 & \dots & 0 \\ \dots & -1/h_1^2 & 1/h_1^2 & & & & \\ & \vdots & & \ddots & & & \\ \dots & 0 & & & 0 & & \\ & \vdots & & & & \ddots & \\ \dots & 0 & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \leftarrow (i) \\ \leftarrow (i+1) \\ \\ \leftarrow (i+L_1) \\ \\ \leftarrow (i+L_1L_2) \end{array} . \tag{2.28}$$

Finally, the last cell of the discretized domain is at the extreme corner, that is at the right border of all the three coordinate directions simultaneously. This cell does not have any successor to compute the forward difference approximation of the directional derivative we are considering in this paper. Therefore, the corresponding matrix C_L is the identically null matrix. For our example this means $C_{60} = \mathbf{0}$; this is the only matrix in *Group 8*.

Figure 2.3 shows the sparsity structures of the C_i matrices.

2.1.3 Sparsity structure of vectors $\boldsymbol{w}^{(i,j)}$

We can derive the explicit forms of the vectors $\boldsymbol{w}^{(i,j)}$, which are different depending on the position of the cell within the discretization grid, that is depending on whether its index i is at the border of some coordinate spatial direction.

We will develop first the case for the forward difference approximation scheme.

Let's consider a cell with index i which is not at the right border of any direction. The elements of the vector $\boldsymbol{w}^{(i,j)}$ are the following:

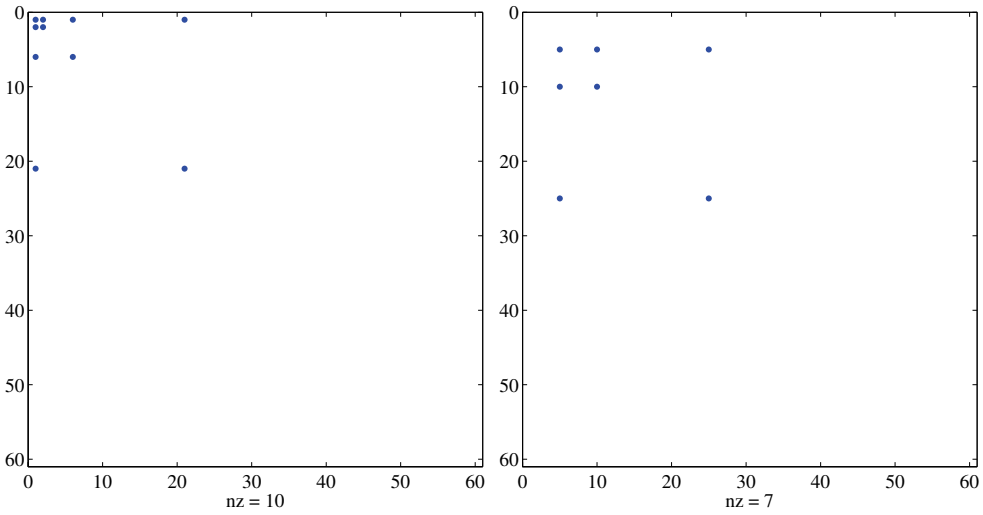
$$\boldsymbol{w}_i^{(i,j)} = \left(\frac{m_i^{(j)}}{h_1^2} + \frac{m_i^{(j)}}{h_2^2} + \frac{m_i^{(j)}}{h_3^2}\right) - \left(\frac{m_{i+1}^{(j)}}{h_1^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2}\right) \tag{2.29a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = -\frac{m_i^{(j)}}{h_1^2} + \frac{m_{i+1}^{(j)}}{h_1^2} \tag{2.29b}$$

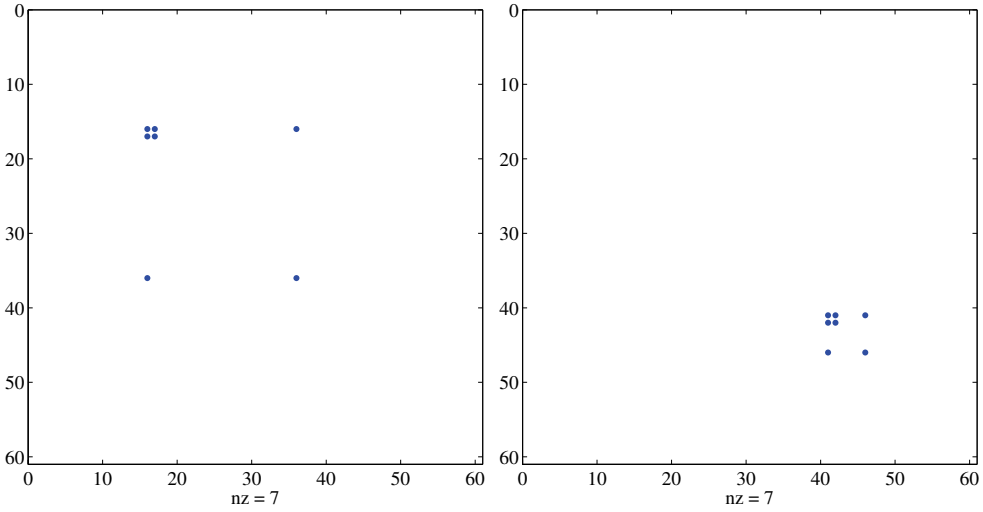
$$\boldsymbol{w}_{i+L_1}^{(i,j)} = -\frac{m_i^{(j)}}{h_2^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2} \tag{2.29c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = -\frac{m_i^{(j)}}{h_3^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2} \tag{2.29d}$$

Consider now the case of a cell i at the border only of the x_1 direction, that is Property



(a) C_i in group 1 (b) C_i in group 2



(c) C_i in group 3 (d) C_i in group 4

Figure 2.3: Sparsity structures of matrices C_i (forward difference) (cont.).

1 is true, while properties 2 and 3 are false. For the sparse vector $\boldsymbol{w}^{(i,j)}$ we can write:

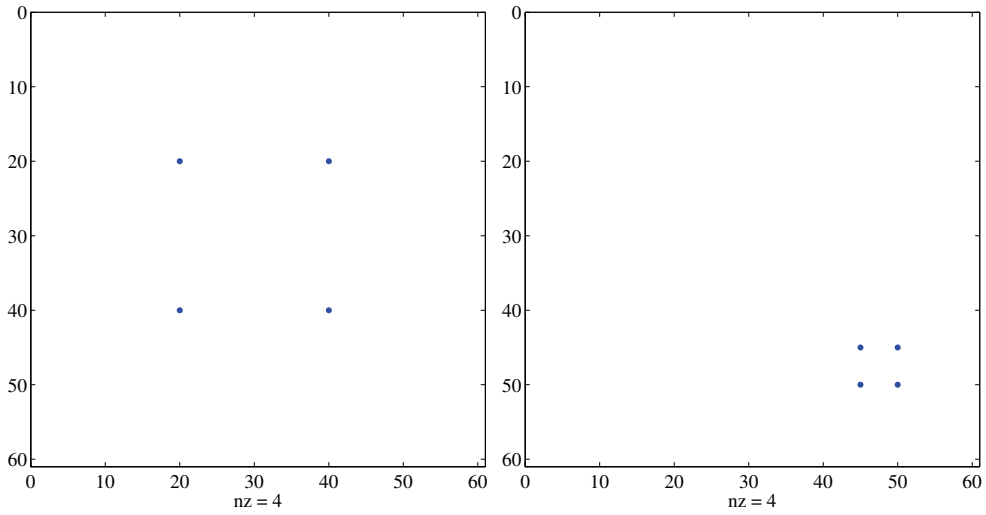
$$\boldsymbol{w}_i^{(i,j)} = \left(\frac{m_i^{(j)}}{h_2^2} + \frac{m_i^{(j)}}{h_3^2}\right) - \left(\frac{m_{i+L_1}^{(j)}}{h_2^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2}\right) \tag{2.30a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = 0 \tag{2.30b}$$

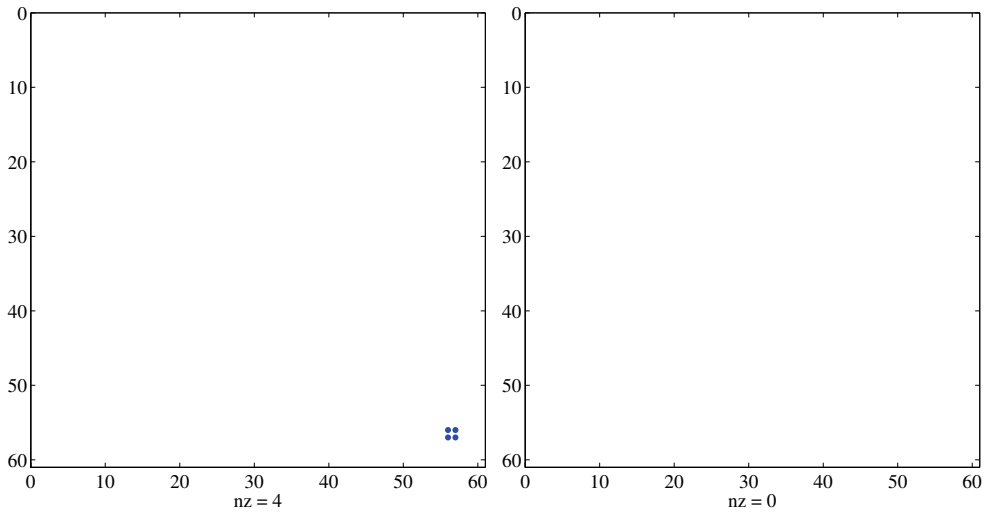
$$\boldsymbol{w}_{i+L_1}^{(i,j)} = -\frac{m_i^{(j)}}{h_2^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2} \tag{2.30c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = -\frac{m_i^{(j)}}{h_3^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2} \tag{2.30d}$$

Consider now a cell i which is at the right border in the x_2 direction only, that is Property 1 and 3 are not satisfied, while Property 2 holds true. The elements of the vector $\boldsymbol{w}^{(i,j)}$



(e) C_i in group 5 (f) C_i in group 6



(g) C_i in group 7 (h) C_i in group 8

Figure 2.3: Sparsity structures of matrices C_i (forward difference) (cont.).

in this case can written as:

$$\boldsymbol{w}_i^{(i,j)} = \left(\frac{m_i^{(j)}}{h_1^2} + \frac{m_i^{(j)}}{h_3^2}\right) - \left(\frac{m_{i+1}^{(j)}}{h_1^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2}\right) \tag{2.31a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = -\frac{m_i^{(j)}}{h_1^2} + \frac{m_{i+1}^{(j)}}{h_1^2} \tag{2.31b}$$

$$\boldsymbol{w}_{i+L_1}^{(i,j)} = 0 \tag{2.31c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = -\frac{m_i^{(j)}}{h_3^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2} \tag{2.31d}$$

Analogously, for a cell i which is at the right border in the x_3 direction only we have that Property 3 holds true and properties 1 and 2 are not satisfied. The elements of the vector

$\boldsymbol{w}^{(i,j)}$ in this case can written as:

$$\boldsymbol{w}_i^{(i,j)} = \left(\frac{m_i^{(j)}}{h_1^2} + \frac{m_i^{(j)}}{h_2^2}\right) - \left(\frac{m_{i+1}^{(j)}}{h_1^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2}\right) \tag{2.32a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = -\frac{m_i^{(j)}}{h_1^2} + \frac{m_{i+1}^{(j)}}{h_1^2} \tag{2.32b}$$

$$\boldsymbol{w}_{i+L_1}^{(i,j)} = -\frac{m_i^{(j)}}{h_2^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2} \tag{2.32c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = 0 \tag{2.32d}$$

There are now to consider the cases of cells that are simultaneously at the right border of multiple coordinate directions. We consider first those cells that are at the right border of x_1 and x_2 directions. In these cases the properties 1 and 2 are both satisfied, while Property 3 is not. The nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}_i^{(i,j)} = \frac{m_i^{(j)}}{h_3^2} - \frac{m_{i+L_1L_2}^{(j)}}{h_3^2} \tag{2.33a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = 0 \tag{2.33b}$$

$$\boldsymbol{w}_{i+L_1}^{(i,j)} = 0 \tag{2.33c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = -\frac{m_i^{(j)}}{h_3^2} + \frac{m_{i+L_1L_2}^{(j)}}{h_3^2} \tag{2.33d}$$

We then consider the cells that are at the right border of x_1 and x_3 directions, where properties 1 and 3 are both satisfied, while Property 2 is not. The nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}_i^{(i,j)} = \frac{m_i^{(j)}}{h_2^2} - \frac{m_{i+L_1}^{(j)}}{h_2^2} \tag{2.34a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = 0 \tag{2.34b}$$

$$\boldsymbol{w}_{i+L_1}^{(i,j)} = -\frac{m_i^{(j)}}{h_2^2} + \frac{m_{i+L_1}^{(j)}}{h_2^2} \tag{2.34c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = 0 \tag{2.34d}$$

Considering now the cells that are at the right border of x_2 and x_3 directions, where properties 2 and 3 are both satisfied and Property 1 hold false, the nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}_i^{(i,j)} = \frac{m_i^{(j)}}{h_1^2} - \frac{m_{i+1}^{(j)}}{h_1^2} \tag{2.35a}$$

$$\boldsymbol{w}_{i+1}^{(i,j)} = -\frac{m_i^{(j)}}{h_1^2} + \frac{m_{i+1}^{(j)}}{h_1^2} \tag{2.35b}$$

$$\boldsymbol{w}_{i+L_1}^{(i,j)} = 0 \tag{2.35c}$$

$$\boldsymbol{w}_{i+L_1L_2}^{(i,j)} = 0 \tag{2.35d}$$

Finally, the last cell of the discretized domain is at the extreme corner, that is at the right border of all the three coordinate directions simultaneously. This cell does not have any successor to compute the forward difference approximation of the directional derivative we are considering in this paper. Therefore, the corresponding vector $\boldsymbol{w}^{(i,j)}$ is the identically null vector.

2.1.4 Sparsity structure of matrices like $[C_1\boldsymbol{d} \ \ldots \ C_M\boldsymbol{d}]$

Some JoInv matrices derive from the multiplication of the C_i matrices by a dense vector $\boldsymbol{d}$: $\left[C_1\boldsymbol{d} \quad C_2\boldsymbol{d} \quad \ldots \quad C_M\boldsymbol{d}\right]$, *i.e.*(1.52), (1.53). The standard pattern of those kind of matrices is shown in Figure 2.4.

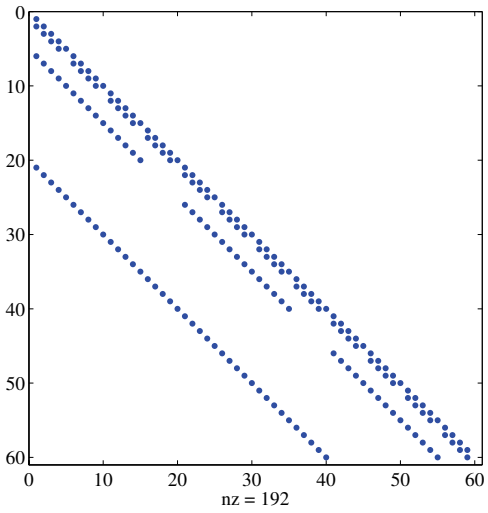


Figure 2.4: Pattern of the matrices $[C_1\boldsymbol{d}\, C_2\boldsymbol{d} \ \ldots \ C_M\boldsymbol{d}]$.

2.1.5 Hessian matrices

Misfit term Hessian

From the (formula ???) is easy to see that the Hessian of the misfit term depend only on the operator A_j, therefore we can't know in advance its structure.

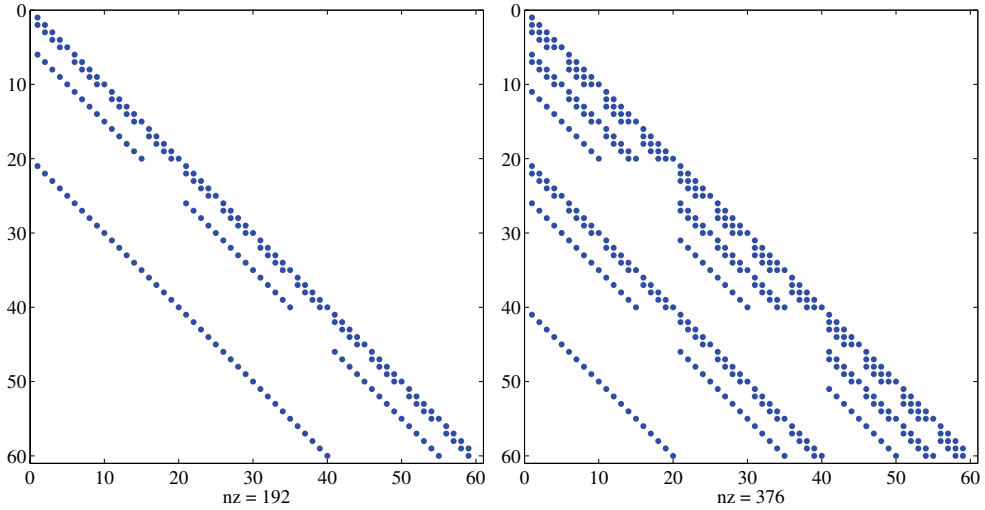
Regularization term Hessian

As you can see from (1.20), the regularization term is always a diagonal matrix, hence its implementation is trivial.

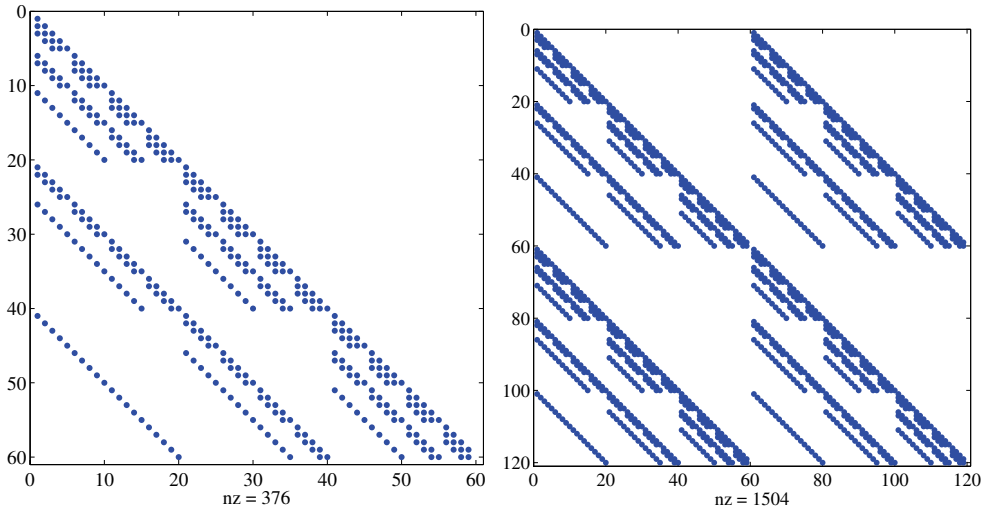
Coupling term Hessian

The structure of the coupling term Hessian is made up several pieces (1.47). The first one is the term in (1.52); once again, we have the same sparsity pattern of $\left[C_1\boldsymbol{d} \quad C_2\boldsymbol{d} \quad \ldots \quad C_M\boldsymbol{d}\right]$ (Figure 2.5(a)). Consider now the term in (1.54); its structure is also well known, because it is built multiplying two matrices with the same structure $C\boldsymbol{d}$ (Figure 2.5(b)). Finally, consider the term in (1.59): it has the same patter of the last Hessian matrix, only scaled

by a constant value and some diagonal matrices (Figure 2.5(c)). The final expression of the Hessian of the mixing term (1.61) is shown in Figure 2.5(d); since the nonzero structure of H_1 is a subset of H_2 and H_3 structure, in each of the four block we have same nonzero pattern.



(a) H_1 (b) H_2



(c) H_3 (d) $\nabla^2_{\boldsymbol{mm}} MIX$

Figure 2.5: Sparsity structures of $\nabla^2_{\boldsymbol{mm}} MIX$ and of its blocks.

2.2 Sparsity structures of JoInv matrices using Central Differences

In this section we consider how the sparsity structure of the matrices C_i and the vectors $\boldsymbol{w}^{(i,j)}$ change if the central finite difference discretization scheme is used in place of the forward finite difference.

2.2.1 Sparsity structures of discrete operators

Consider now the first spatial direction, x_1: the L_1 adjacent cells in this directions have consecutive numbers in the chosen ordering, so the discrete derivative is given by

$$\frac{\partial\, m_i(\boldsymbol{x})}{\partial x_1} \approx \Delta_{x_1} m_i = \frac{m_{i+h} - m_{i-h}}{2h_1}$$

for all the $L_1 - 2$ cells, but the first and the last one, where it is zero. Hence the discrete partial derivative operator for these cells can be written as

$$B_{x_1} = \frac{1}{2h_1} \underbrace{\begin{pmatrix} 0 & & & & & \\ -1 & 0 & 1 & & & \\ & -1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 0 & 1 \\ & & & & & 0 \end{pmatrix}}_{L_1 \times L_1} \Bigg\} L_1 - 2 \text{ rows} \tag{2.36}$$

By repeating the reasoning for all the other adjacent cells along the first spatial direction, one has the discrete partial derivative operator for the whole domain as a block diagonal matrix

$$G_{x_1} = \underbrace{\begin{pmatrix} B_{x_1} & & & \\ & B_{x_1} & & \\ & & \ddots & \\ & & & B_{x_1} \end{pmatrix}}_{L \times L} \tag{2.37}$$

with $L_2 L_3$ blocks. It follows that $\Delta_{x_1} \boldsymbol{m}(\boldsymbol{x}) = G_{x_1} \boldsymbol{m}$.

Consider now the second spatial direction, x_2: here the difference between the indices of two adjacent cells is L_1, for all cells but those of the boundary face, where the partial discrete derivative will be zero. For the former cells we thus have

$$\frac{\partial\, m_i(\boldsymbol{x})}{\partial x_2} \approx \Delta_{x_2} m_i = \frac{m_{i+L_1} - m_{i-L_1}}{2h_2}\,.$$

Thus, we can write the discrete partial derivative operator for all the cells in each "layer" of the cell grid parallel to the $x_1 x_3$ plain as

$$B_{x_2} = \frac{1}{2h_2} \underbrace{\begin{pmatrix} \mathbf{0}_{L_1} & & & & & \\ -I_{L_1} & \mathbf{0}_{L_1} & I_{L_1} & & & \\ & -I_{L_1} & \mathbf{0}_{L_1} & I_{L_1} & & \\ & & \ddots & \ddots & \ddots & \\ & & & -I_{L_1} & \mathbf{0}_{L_1} & I_{L_1} \\ & & & & & \mathbf{0}_{L_1} \end{pmatrix}}_{(L_1 L_2) \times (L_1 L_2)} \Bigg\} L_2 - 2 \text{ block rows} \tag{2.38}$$

where I_{L_1} and $\mathbf{0}_{L_1}$ are the $L_1 \times L_1$ identity matrix and null matrix, respectively. By repeating the reasoning for all the other "layers" of the cell grid parallel to the x_1x_3 plain one has the discrete partial derivative operator in the second direction for the whole domain, again as a block diagonal matrix

$$G_{x_2} = \begin{pmatrix} B_{x_2} & & & \\ & B_{x_2} & & \\ & & \ddots & \\ & & & B_{x_2} \end{pmatrix}_{L\times L} \tag{2.39}$$

with L_3 blocks. It follows that $\Delta_{x_2}\boldsymbol{m}(\boldsymbol{x}) = G_{x_2}\boldsymbol{m}$.

Last, consider the third spatial direction, x_3: the difference between the indices of adjacent cells in this direction is now L_1L_2, for all cells but those of the boundary face, where the partial discrete derivative will be zero. For the former cells we have

$$\frac{\partial\, m_i(\boldsymbol{x})}{\partial x_3} \approx \Delta_{x_3} m_i = \frac{m_{i+L_1L_2} - m_{i-L_1L_2}}{2h_3}\,.$$

The discrete partial derivative operator for all the cells in each "layer" of the cell grid parallel to the x_1x_2 plain as

$$B_{x_3} = \frac{1}{2h_3}\begin{pmatrix} -I_{L_1L_2} & \mathbf{0}_{L_1L_2} & I_{L_1L_2} \end{pmatrix}_{(L_1L_2)\times 3(L_1L_2)} \tag{2.40}$$

where $I_{L_1L_2}$ is the identity matrix sized $(L_1L_2) \times (L_1L_2)$. By repeating the reasoning for all the other "layers" of the cell grid parallel to the x_1x_2 plain we obtain the discrete partial derivative operator in the third spatial direction for the whole domain

$$G_{x_3} = \left.\begin{pmatrix} \mathbf{0}_{L_1L_2} & & & & \\ & B_{x_3} & & & \\ & & B_{x_3} & & \\ & & & \ddots & \\ & & & B_{x_3} & \\ & & & & \mathbf{0}_{L_1L_2} \end{pmatrix}\right\} L_3 - 2 \text{ block rows}_{L\times L} \tag{2.41}$$

where $\mathbf{0}_{L_1L_2}$ is the $(L_1L_2) \times (L_1L_2)$ null matrix. It follows that $\Delta_{x_3}\boldsymbol{m}(\boldsymbol{x}) = G_{x_3}\boldsymbol{m}$.

Figure 2.6 shows the sparsity structures of the discrete spatial derivative operator G_{x_t}.

2.2.2 Sparsity structure of matrices C_i

Now we can see how the $M \times M$ sparse matrices $C_i = (S_iG)^T(S_iG)$, $i = 1, \ldots, M$, look like. Let's consider a cell with index i which is not at the border of any direction, that is a cell in *Group 1*. The elements of the matrix C_i are the following:

$$\left(C_i\right)_{k,k} = 1/(2h_3)^2, \qquad k = i - L_1L_2, i + L_1L_2, \tag{2.42a}$$

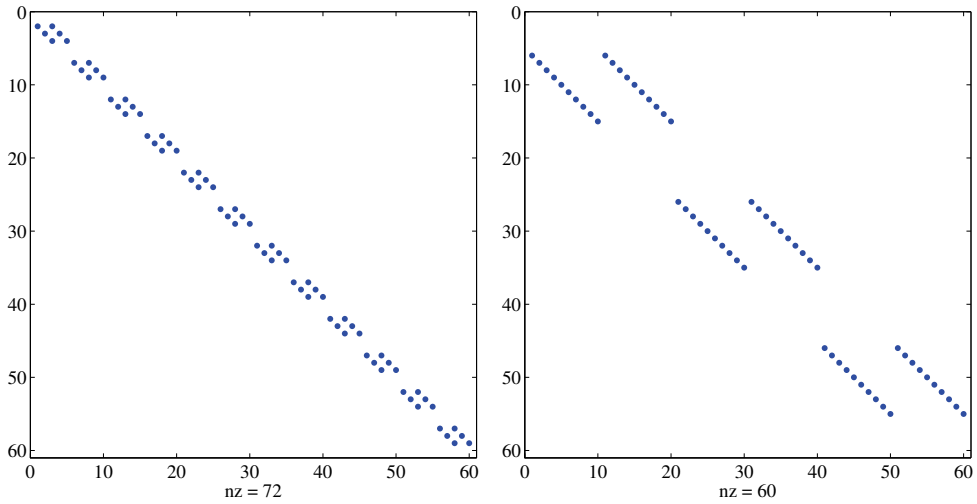
$$\left(C_i\right)_{k,k} = 1/(2h_2)^2, \qquad k = i - L_1, i + L_1, \tag{2.42b}$$

$$\left(C_i\right)_{k,k} = 1/(2h_1)^2, \qquad k = i - 1, i + 1, \tag{2.42c}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_3)^2, \quad k = i - L_1L_2;\ j = i + L_1L_2, \tag{2.42d}$$

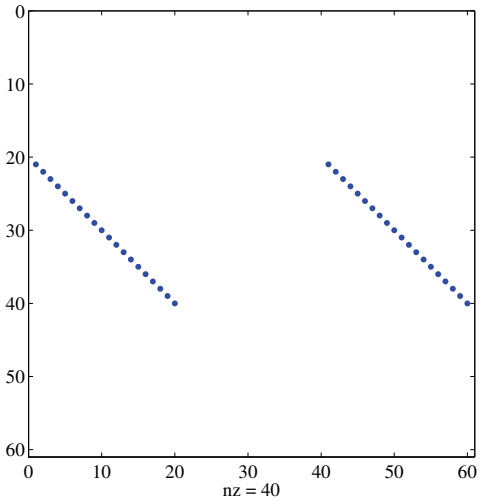
$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_2)^2, \quad k = i - L_1;\ j = i + L_1, \tag{2.42e}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_1)^2, \quad k = i - 1;\ j = i + 1. \tag{2.42f}$$



(a) Operator G_{x_1}

(b) Operator G_{x_2}



(c) Operator G_{x_3}

Figure 2.6: Sparsity structures of the discrete spatial derivative operator G_{x_t} (central difference).

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{matrix} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix} \\ \begin{pmatrix} h_3^{-2} & & & & & -h_3^{-2} \\ & h_2^{-2} & & & -h_2^{-2} & \\ & & h_1^{-2} & -h_1^{-2} & & \\ & & -h_1^{-2} & h_1^{-2} & & \\ & -h_2^{-2} & & & h_2^{-2} & \\ -h_3^{-2} & & & & & h_3^{-2} \end{pmatrix} \end{array} \begin{matrix} \\ \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{matrix} \tag{2.43}$$

where the only nonzero elements are explicitly reported.

Let's consider now the case of a cell i at the border only of the x_1 direction, that is Property 1 is true, while properties 2 and 3 are false. We say that the matrices C_i resulted from a cell i with those properties belong to *Group 2*. For the sparse matrix C_i we can write:

$$\left(C_i\right)_{k,k} = 1/(2h_3)^2, \qquad k = i - L_1L_2, i + L_1L_2, \tag{2.44a}$$

$$\left(C_i\right)_{k,k} = 1/(2h_2)^2, \qquad k = i - L_1, i + L_1, \tag{2.44b}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_3)^2, \quad k = i - L_1L_2;\ j = i + L_1L_2, \tag{2.44c}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_2)^2, \quad k = i - L_1;\ j = i + L_1. \tag{2.44d}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{matrix} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix} \\ \begin{pmatrix} h_3^{-2} & & & & & -h_3^{-2} \\ & h_2^{-2} & & & -h_2^{-2} & \\ & & 0 & 0 & & \\ & & 0 & 0 & & \\ & -h_2^{-2} & & & h_2^{-2} & \\ -h_3^{-2} & & & & & h_3^{-2} \end{pmatrix} \end{array} \begin{matrix} \\ \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{matrix} \tag{2.45}$$

again, only nonzero elements are explicitly reported.

Consider now a cell i which is at the border in the x_2 direction only, that is Property 1 and 3 are not satisfied, while Property 2 holds true (*Group 3*). The elements of the matrix C_i in this case can written as:

$$\left(C_i\right)_{k,k} = 1/(2h_3)^2, \qquad k = i - L_1L_2, i + L_1L_2, \tag{2.46a}$$

$$\left(C_i\right)_{k,k} = 1/(2h_1)^2, \qquad k = i - 1, i + 1, \tag{2.46b}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_3)^2, \quad k = i - L_1L_2;\ j = i + L_1L_2, \tag{2.46c}$$

$$\left(C_i\right)_{k,j} = \left(C_i\right)_{j,k} = -1/(2h_1)^2, \quad k = i - 1;\ j = i + 1. \tag{2.46d}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{matrix} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix} \\ \begin{pmatrix} h_3^{-2} & & & & & -h_3^{-2} \\ & 0 & & & 0 & \\ & & h_1^{-2} & -h_1^{-2} & & \\ & & -h_1^{-2} & h_1^{-2} & & \\ & 0 & & & 0 & \\ -h_3^{-2} & & & & & h_3^{-2} \end{pmatrix} \end{array} \begin{matrix} \\ \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{matrix} \tag{2.47}$$

Analogously, for a cell i which is at the right border in the x_3 direction only we have that Property 3 holds true and properties 1 and 2 are not satisfied (*Group 4*). The elements of the matrix C_i in this case can written as:

$$\big(C_i\big)_{k,k} = 1/(2h_2)^2, \qquad k = i-L_1, i+L_1, \tag{2.48a}$$
$$\big(C_i\big)_{k,k} = 1/(2h_1)^2, \qquad k = i-1, i+1, \tag{2.48b}$$
$$\big(C_i\big)_{k,j} = \big(C_i\big)_{j,k} = -1/(2h_2)^2, \quad k = i-L_1;\ j = i+L_1, \tag{2.48c}$$
$$\big(C_i\big)_{k,j} = \big(C_i\big)_{j,k} = -1/(2h_1)^2, \quad k = i-1;\ j = i+1. \tag{2.48d}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{matrix} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{matrix} \\ \begin{pmatrix} 0 & & & & & 0 \\ & h_2^{-2} & & & -h_2^{-2} & \\ & & h_1^{-2} & -h_1^{-2} & & \\ & & -h_1^{-2} & h_1^{-2} & & \\ & -h_2^{-2} & & & h_2^{-2} & \\ 0 & & & & & 0 \end{pmatrix} \end{array} \begin{matrix} \\ \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{matrix} \tag{2.49}$$

There are now to consider the cases of cells that are simultaneously at the border of multiple coordinate directions. We consider first those cells that are at the border of x_1 and x_2 directions. In these cases the properties 1 and 2 are both satisfied, while Property 3 is not (*Group 5*). The nonzero elements in the corresponding matrices C_i are the following:

$$\big(C_i\big)_{k,k} = 1/(2h_3)^2, \qquad k = i-L_1L_2, i+L_1L_2, \tag{2.50a}$$
$$\big(C_i\big)_{k,j} = \big(C_i\big)_{j,k} = -1/(2h_3)^2, \quad k = i-L_1L_2;\ j = i+L_1L_2. \tag{2.50b}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{array}{cccccc} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{pmatrix} h_3^{-2} & & & & & -h_3^{-2} \\ & 0 & & & 0 & \\ & & 0 & 0 & & \\ & & 0 & 0 & & \\ & 0 & & & 0 & \\ -h_3^{-2} & & & & & h_3^{-2} \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{array} \tag{2.51}$$

We then consider the cells that are at the border of x_1 and x_3 directions, where properties 1 and 3 are both satisfied, while Property 2 is not (*Group 6*). The nonzero elements in the corresponding matrices C_i are the following:

$$\big(C_i\big)_{k,k} = 1/(2h_2)^2, \qquad k = i-L_1, i+L_1, \tag{2.52a}$$

$$\big(C_i\big)_{k,j} = \big(C_i\big)_{j,k} = -1/(2h_2)^2, \quad k = i-L_1;\ j = i+L_1. \tag{2.52b}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{array}{cccccc} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{pmatrix} 0 & & & & & 0 \\ & h_2^{-2} & & & -h_2^{-2} & \\ & & 0 & 0 & & \\ & & 0 & 0 & & \\ & -h_2^{-2} & & & h_2^{-2} & \\ 0 & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{array} \tag{2.53}$$

Considering now the cells that are at the border of x_2 and x_3 directions, where properties 2 and 3 are both satisfied and Property 1 hold false (*Group 7*), the nonzero elements in the corresponding matrices C_i are the following:

$$\big(C_i\big)_{k,k} = 1/(2h_1)^2, \qquad k = i-1, i+1, \tag{2.54a}$$

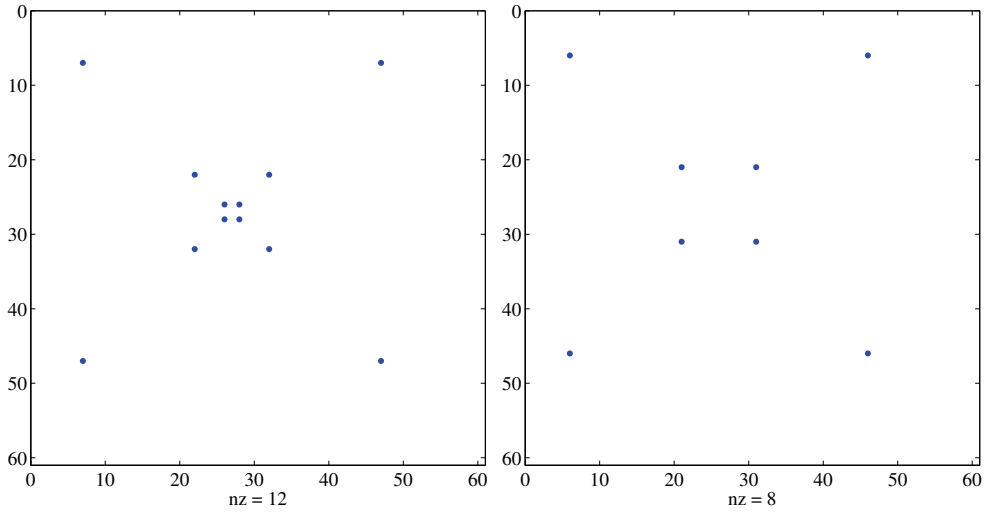
$$\big(C_i\big)_{k,j} = \big(C_i\big)_{j,k} = -1/(2h_1)^2, \quad k = i-1;\ j = i+1. \tag{2.54b}$$

In this case the matrix C_i becomes

$$C_i = \frac{1}{4}\begin{array}{c} \begin{array}{cccccc} (i-L_1L_2) & (i-L_1) & (i-1) & (i+1) & (i+L_1) & (i+L_1L_2) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{pmatrix} 0 & & & & & 0 \\ & 0 & & & 0 & \\ & & h_1^{-2} & -h_1^{-2} & & \\ & & -h_1^{-2} & h_1^{-2} & & \\ & 0 & & & 0 & \\ 0 & & & & & 0 \end{pmatrix} \end{array} \begin{array}{l} \leftarrow (i-L_1L_2) \\ \leftarrow (i-L_1) \\ \leftarrow (i-1) \\ \leftarrow (i+1) \\ \leftarrow (i+L_1) \\ \leftarrow (i+L_1L_2) \end{array} \tag{2.55}$$

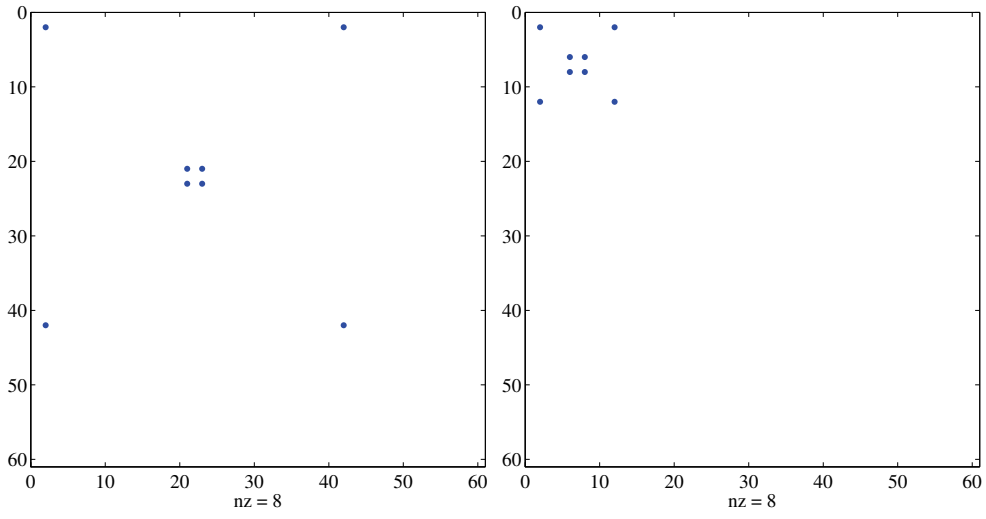
Finally, the cells of the discretized domain that are at the border of all the three coordinate directions simultaneously (*Group 8*). The corresponding matrix C_i is the identically null matrix.

Figure 2.7 shows the sparsity structures of the C_i matrices.



(a) C_i in group 1

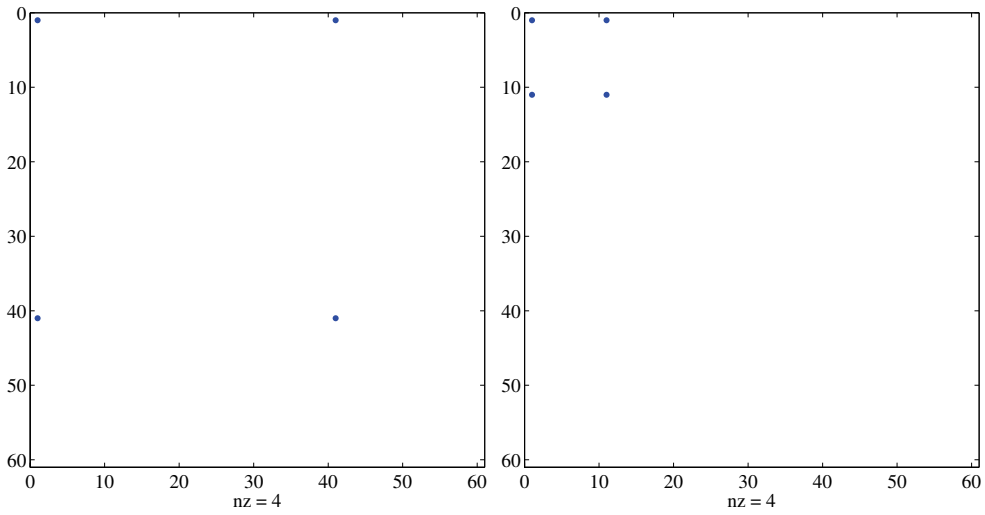
(b) C_i in group 2



(c) C_i in group 3

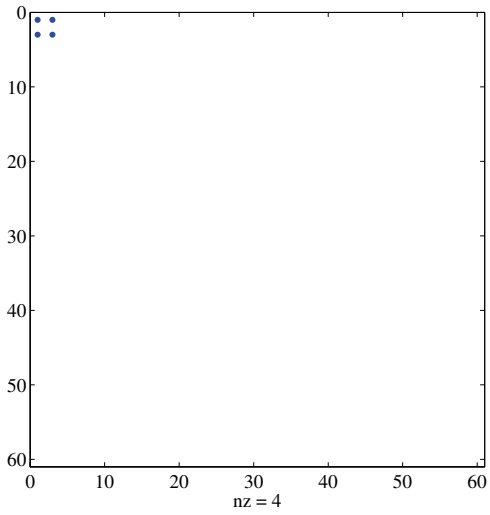
(d) C_i in group 4

Figure 2.7: Sparsity structures of matrices C_i (central difference) (cont.).

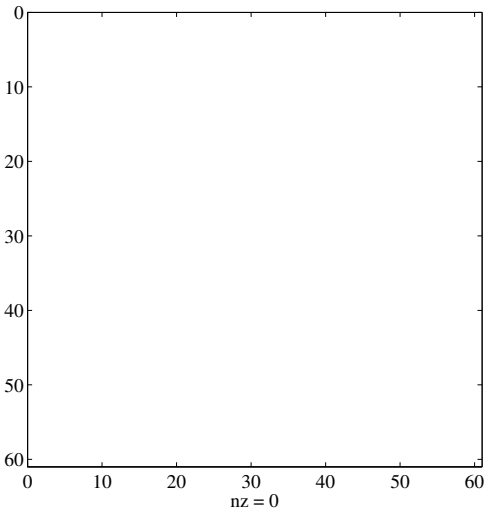


(e) C_i in group 5

(f) C_i in group 6



(g) C_i in group 7



(h) C_i in group 8

Figure 2.7: Sparsity structures of matrices C_i (central difference) (cont.).

2.2.3 Sparsity structure of vectors $\boldsymbol{w}^{(i,j)}$

We can derive once again the explicit form of the vectors $\boldsymbol{w}^{(i,j)}$, which look different depending on whether the cell is at the border of some coordinate direction.

Let's consider a cell with index i which is not at the border of any direction; the elements of the vector $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = \frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} - \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.56a}$$

$$\boldsymbol{w}^{(i,j)}_{i-L_1} = \frac{m^{(j)}_{i-L_1}}{4h_2^2} - \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.56b}$$

$$\boldsymbol{w}^{(i,j)}_{i-1} = \frac{m^{(j)}_{i-1}}{4h_1^2} - \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.56c}$$

$$\boldsymbol{w}^{(i,j)}_{i+1} = -\frac{m^{(j)}_{i-1}}{4h_1^2} + \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.56d}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1} = -\frac{m^{(j)}_{i-L_1}}{4h_2^2} + \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.56e}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = -\frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} + \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.56f}$$

Let's consider now the case of a cell i at the border only of the x_1 direction, that is Property 1 holds true for it, while properties 2 and 3 are false. For the sparse vector $\boldsymbol{w}^{(i,j)}$ we can write:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = \frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} - \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.57a}$$

$$\boldsymbol{w}^{(i,j)}_{i-L_1} = \frac{m^{(j)}_{i-L_1}}{4h_2^2} - \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.57b}$$

$$\boldsymbol{w}^{(i,j)}_{i-1} = 0 \tag{2.57c}$$

$$\boldsymbol{w}^{(i,j)}_{i+1} = 0 \tag{2.57d}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1} = -\frac{m^{(j)}_{i-L_1}}{4h_2^2} + \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.57e}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = -\frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} + \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.57f}$$

Consider now a cell i which is at the border in the x_2 direction only, that is Property 1 and 3 are not satisfied, while Property 2 holds true. The elements of the vector $\boldsymbol{w}^{(i,j)}$

in this case can written as:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = \frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} - \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.58a}$$

$$\boldsymbol{w}^{(i,j)}_{i-L_1} = 0 \tag{2.58b}$$

$$\boldsymbol{w}^{(i,j)}_{i-1} = \frac{m^{(j)}_{i-1}}{4h_1^2} - \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.58c}$$

$$\boldsymbol{w}^{(i,j)}_{i+1} = -\frac{m^{(j)}_{i-1}}{4h_1^2} + \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.58d}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1} = 0 \tag{2.58e}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = -\frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} + \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.58f}$$

Analogously, for a cell i which is at the border in the x_3 direction only we have that Property 3 holds true and properties 1 and 2 are not satisfied. The elements of the vector $\boldsymbol{w}^{(i,j)}$ in this case can written as:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = 0 \tag{2.59a}$$

$$\boldsymbol{w}^{(i,j)}_{i-L_1} = \frac{m^{(j)}_{i-L_1}}{4h_2^2} - \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.59b}$$

$$\boldsymbol{w}^{(i,j)}_{i-1} = \frac{m^{(j)}_{i-1}}{4h_1^2} - \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.59c}$$

$$\boldsymbol{w}^{(i,j)}_{i+1} = -\frac{m^{(j)}_{i-1}}{4h_1^2} + \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.59d}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1} = -\frac{m^{(j)}_{i-L_1}}{4h_2^2} + \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.59e}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = 0 \tag{2.59f}$$

There are now to consider the cases of cells that are simultaneously at the border of multiple coordinate directions. We consider first those cells that are at the border of x_1 and x_2 directions. In these cases the properties 1 and 2 are both satisfied, while Property 3 is not. The nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = \frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} - \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.60a}$$

$$\boldsymbol{w}^{(i,j)}_{i-L_1} = 0 \tag{2.60b}$$

$$\boldsymbol{w}^{(i,j)}_{i-1} = 0 \tag{2.60c}$$

$$\boldsymbol{w}^{(i,j)}_{i+1} = 0 \tag{2.60d}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1} = 0 \tag{2.60e}$$

$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = -\frac{m^{(j)}_{i-L_1L_2}}{4h_3^2} + \frac{m^{(j)}_{i+L_1L_2}}{4h_3^2} \tag{2.60f}$$

We then consider the cells that are at the border of x_1 and x_3 directions, where properties 1 and 3 are both satisfied, while Property 2 is not. The nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = 0 \tag{2.61a}$$
$$\boldsymbol{w}^{(i,j)}_{i-L_1} = \frac{m^{(j)}_{i-L_1}}{4h_2^2} - \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.61b}$$
$$\boldsymbol{w}^{(i,j)}_{i-1} = 0 \tag{2.61c}$$
$$\boldsymbol{w}^{(i,j)}_{i+1} = 0 \tag{2.61d}$$
$$\boldsymbol{w}^{(i,j)}_{i+L_1} = -\frac{m^{(j)}_{i-L_1}}{4h_2^2} + \frac{m^{(j)}_{i+L_1}}{4h_2^2} \tag{2.61e}$$
$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = 0 \tag{2.61f}$$

Considering now the cells that are at the border of x_2 and x_3 directions, where properties 2 and 3 are both satisfied and Property 1 holds false, the nonzero elements in the corresponding vectors $\boldsymbol{w}^{(i,j)}$ are the following:

$$\boldsymbol{w}^{(i,j)}_{i-L_1L_2} = 0 \tag{2.62a}$$
$$\boldsymbol{w}^{(i,j)}_{i-L_1} = 0 \tag{2.62b}$$
$$\boldsymbol{w}^{(i,j)}_{i-1} = \frac{m^{(j)}_{i-1}}{4h_1^2} - \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.62c}$$
$$\boldsymbol{w}^{(i,j)}_{i+1} = -\frac{m^{(j)}_{i-1}}{4h_1^2} + \frac{m^{(j)}_{i+1}}{4h_1^2} \tag{2.62d}$$
$$\boldsymbol{w}^{(i,j)}_{i+L_1} = 0 \tag{2.62e}$$
$$\boldsymbol{w}^{(i,j)}_{i+L_1L_2} = 0 \tag{2.62f}$$

Finally, the last cell of the discretized domain is at the extreme corner, that is at the border of all the three coordinate directions simultaneously. Therefore, the corresponding vector $\boldsymbol{w}^{(i,j)}$ is the identically null vector.

2.2.4 Sparsity structure of matrices like $[C_1\boldsymbol{d} \ \dots \ C_M\boldsymbol{d}]$

The structure of the JoInv matrices obtained by the multiplication of the sparse C_i matrices by a dense vector $\boldsymbol{d}$, $\begin{bmatrix} C_1\boldsymbol{d} & C_2\boldsymbol{d} & \dots & C_M\boldsymbol{d} \end{bmatrix}$, is shown in Figure 2.8.

2.2.5 Hessian matrices

Misfit term Hessian

Since the structure of the Hessian of the misfit term depends only on the operator A_j, it doesn't change switching from a forward difference scheme to the central difference scheme.

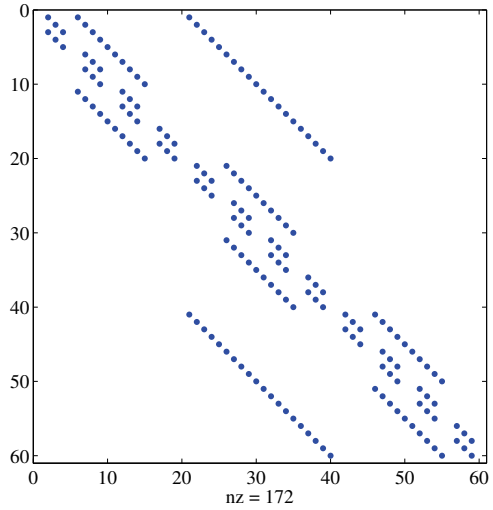


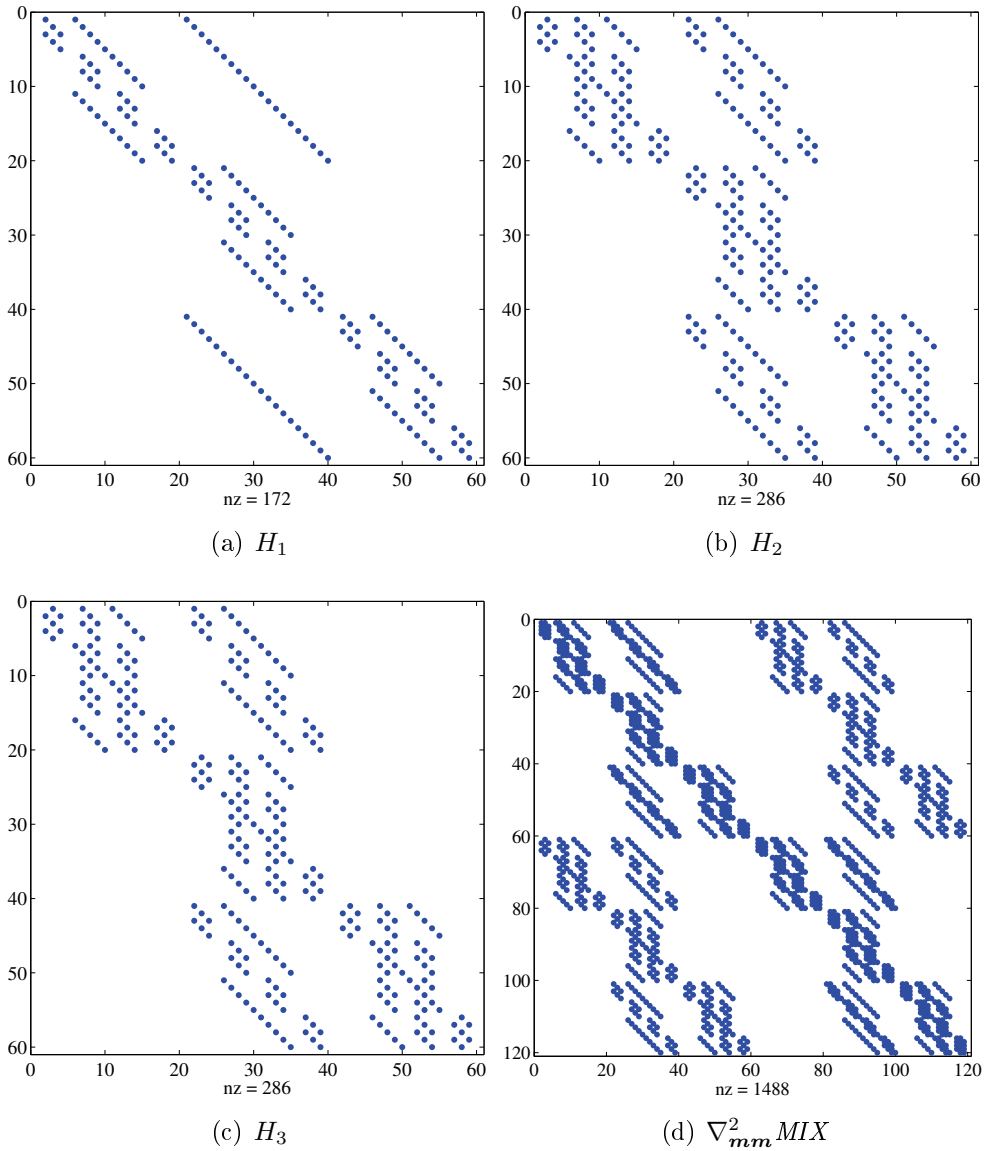
Figure 2.8: Pattern of the matrices $[C_1\boldsymbol{d}\ C_2\boldsymbol{d}\ \ldots\ C_M\boldsymbol{d}]$.

Regularization term Hessian

As it can be seen from (1.20), the regularization term is again a diagonal matrix, hence its implementation is trivial.

Coupling term Hessian

The structure of the coupling term Hessian is composed by several pieces (1.47). The first one is the term in (1.52): once again, we have the same sparsity pattern of $[\ C_1\boldsymbol{d}\ C_2\boldsymbol{d}\ \ldots\ C_M\boldsymbol{d}\ ]$ (Figure 2.9(a)). Consider now the term in (1.54): its structure is also well known, because it is built by multiplying two matrices with the same structure $C\boldsymbol{d}$ (Figure 2.9(b)). Finally, consider the term in (1.61): it has the same pattern of the last Hessian matrix, only scaled by a constant value and some diagonal matrices (Figure 2.9(c)). The final structure of the Hessian of the mixing term (1.61) is shown in Figure 2.9(d); since the nonzero structure of H_1 is a subset of H_2 and H_3 structures, in each of the four block we have the same nonzero pattern.



(a) H_1 (b) H_2

(c) H_3 (d) $\nabla^2_{mm} MIX$

Figure 2.9: Sparsity structures of $\nabla^2_{mm} MIX$ and of its blocks.

2.3 Comments on alternative representations of discrete derivative operators

It is well known that there exist alternative ways to represent the discrete derivative operators related to forward and central difference, both for 2D and 3D models. For instance, [66, chap. 7] analyze various ways of representing discrete partial derivatives operators for the regularization term in Tikhonov functionals as tensor products of simple matrices like identities, diagonals, bi-diagonals, tri-diagonals. For example, the Euclidean regularizer with forward finite difference for a 2D image $\mathbf{Y}$ sized $m \times n$ can be easily represented as $\|D\boldsymbol{y}\|_2^2$, where $\boldsymbol{y} = \mathrm{vec}(\mathbf{Y})$ is the "vectorized" 1D representation of the image and the derivative operator D is given by

$$D = \begin{pmatrix} I_n \otimes D_{1,m} \\ D_{1,n} \otimes I_m \end{pmatrix}$$

where I_n, I_m are identity matrices and the 1D finite difference operators $D_{1,m}$ and $D_{1,m}$ are $n \times n$ and $m \times m$ bi-diagonal matrices, respectively, with the same form of B_{x_1} in (2.1), but the last line. Generalization of these representations to 3D data can be found for example in [17, 18]. However, there are some comments:

- the form of the joining term (1.7c) is much more complicated than classical 2-norm or Total Variation regularizers and it is far quite whether a similar easy tensor-product representation could be useful or not, or if it even exists;
- tensor-product operators are known to be quite useful to express the direct solution of important simple partial difference equations coming from PDEs discretization, but computations become harder for increasingly difficult operators (the literature is huge and dates back to '60s and even earlier: see, *e.g.*, [20, 69, 87, 88, 90] and more recently [33, 42]);
- in the present work we are also strongly interested in the most compact and effective implementation suitable to PETSc's sparse matrices storing scheme, so the actual positions of nonzero elements matter the most.

For these reasons, we do not investigate further about different expressions for the discrete objective function gradient and Hessian.

Chapter 3

HPC implementation

3.1 JoInv Matlab prototype

First of all, we coded a Matlab prototype; as it is common knowledge, Matlab code is not efficient, but coding with it is quite simple and fast. A Matlab prototype is convenient because writing an HPC code starting from an outline code is easier and because we can check the implementation correctness. For the JoInv project, Figure 3.1 shows the streamlined Matlab flowchart.

3.2 What libraries?

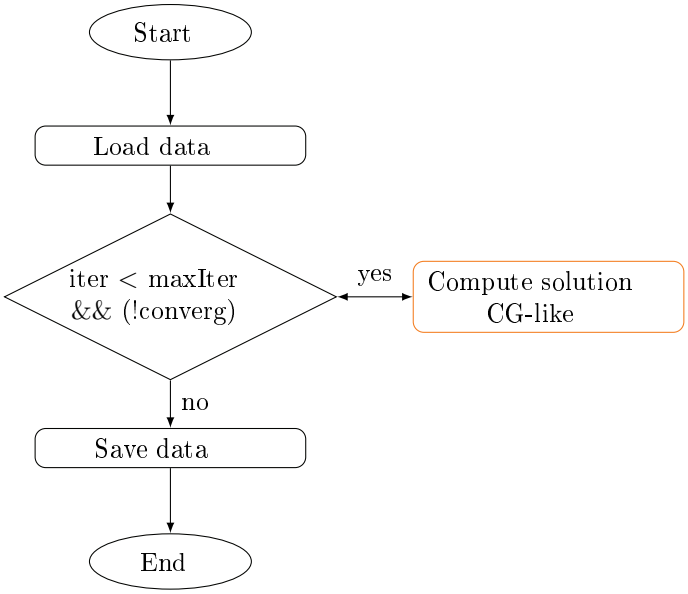
The first step of the implementation part was the study of the scientific libraries available in order to choose the best ones for JoInv. We were looking for a library with the following features:

- up-to-date;
- `C` or `C++` support;
- support for parallel operations using MPI;
- portable;
- vector and matrix sequential and parallel operations, such as multiplication;
- linear and nonlinear system solvers.

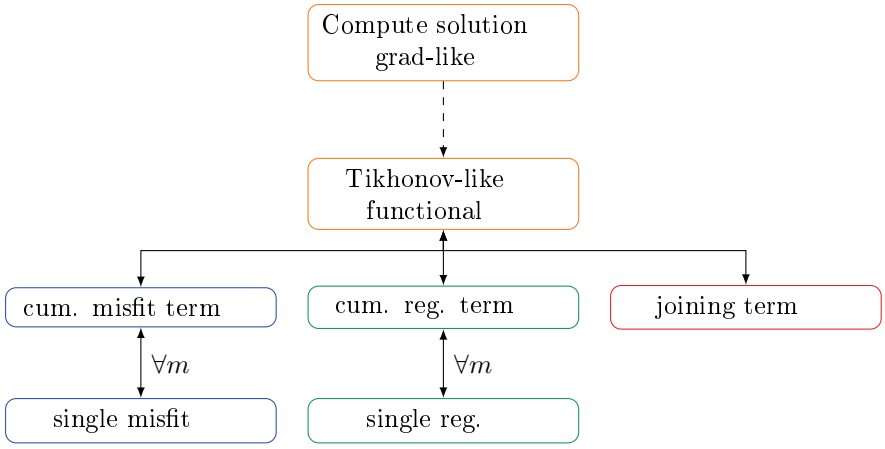
All the features we were interested in are listed in Table 3.1: PETSc and TAO libraries are the best choice. See Section A.2 and Section A.3 for more details about these libraries.

In future development we will improve JoInv performances by adding architecture-specific libraries, such as MKL and ACML for Intel and AMD, respectively.

How these libraries are related to each other is shown in Figure 3.2.



(a) The streamlined Matlab flowchart. We load the data and set the parameters; then, we compute the solution of our inverse problem using a gradient-like method. At the end, we save the result to check if the solution found is comparable to that found using a standard approach and if it matches with the solution found using the JoInv HPC implementation.



(b) Compute-solution block: we minimize the objective functional following the Tikhonov approach; therefore, we compute the misfit and the regularization terms for all models and then the joining term.

Figure 3.1: The streamlined Matlab flowchart.

	BLAS PBLAS	PETSc	TAO	LAPACK Sca-LAPACK	ARPACK PARPACK ARPACK++	ESSL PESSL	GSL	MKL	ACML
	Basic Linear Algebra Subprograms	Portable, Extensible Toolkit for Scientific Computation	Toolkit for Advanced Optimization	Linear Algebra PACKage, Scalable LAPACK	ARnoldi PACKage, Parallel ARPACK	(Parallel) Engineering and Scientific Subroutine library	GNU Scientific Library	Intel Math Kernel Library	AMD Core Math Library
Update	2005	2010	2010	2010	2000	2010	2010	2009	2010
$x^T y$	y (level 1)	BLAS LAPACK	PETSc	BLAS PBLAS	BLAS LAPACK	BLAS LAPACK (Optim.)	BLAS (Optim.)	BLAS LAPACK	BLAS (Optim.)
Ax	y (level 2)	BLAS LAPACK	PETSc	BLAS PBLAS	BLAS LAPACK	BLAS (Optim.)	BLAS (Optim.)	BLAS LAPACK	BLAS (Optim.)
$A \times B$	y (level 3)	BLAS LAPACK	PETSc	BLAS PBLAS	BLAS LAPACK	BLAS (Optim.)	BLAS (Optim.)	BLAS LAPACK	BLAS (Optim.)
C++	C	y	y	LAPACK++	ARPACK++	y, C	y	y (partial)	Fortran (C interface)
Eigenvalues	n	y (SLEPc)	n	y	y	y	y	y	y, LAPACK (Optim.)
Direct solver bend linear system	n	y	y	y	y	y	y	y	y, LAPACK (Optim.)
Iterative linear system solvers (*CG)	n	y	y	n	n	n	n	y	
MPI	n	y	y	y	y			y	
Note						IBM		Intel	AMD

Table 3.1: Common scientific libraries available and their features.

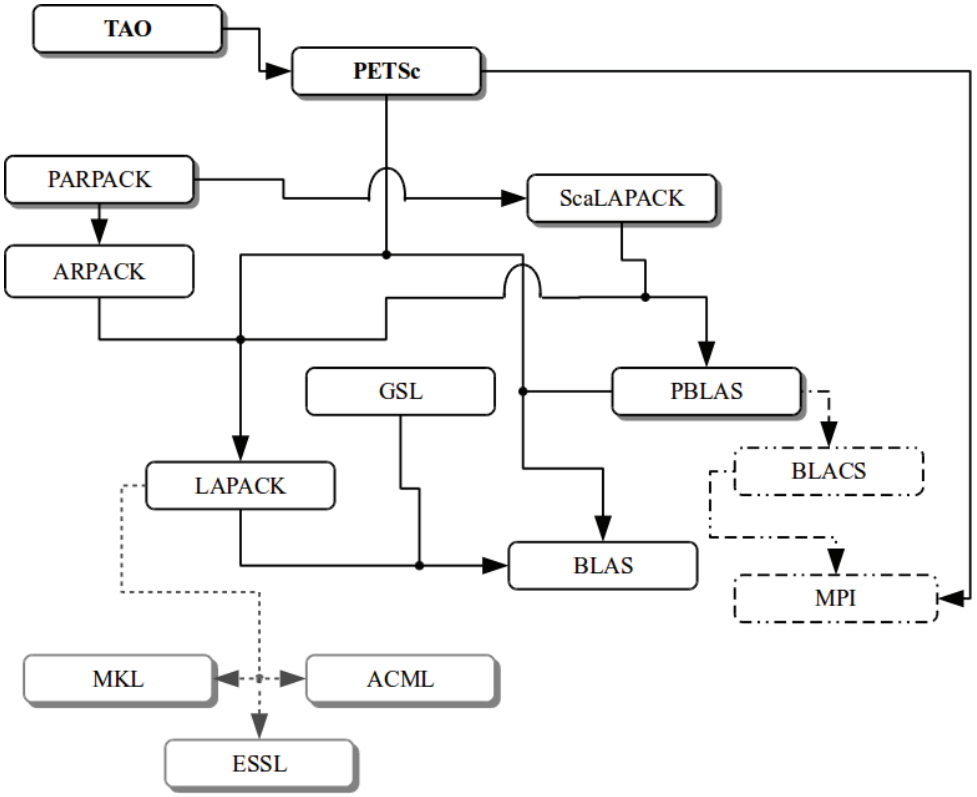


Figure 3.2: Relationship between some well known scientific libraries. Solid black arrow means “is dependent on”, dotted gray arrow means “contributes to”.

3.3 JoInv implementation choices

The Matlab code is only a useful prototype. For the actual HPC version, we made the following choices:

Programming language: `C` . The main advantages are:

- speed of the resulting application. `C` is a compiled language and `C` source code can be optimized much more than higher-level languages because the language set is relatively small and very efficient;
- simple interaction with PETSc and TAO libraries.

Libraries: MPI, PETSc and TAO

- MPI (Message Passing Interface) is a de facto portable, efficient, and flexible standard. See Section A.1 for more details about this library.
- PETSc (Portable, Extensible Toolkit for Scientific Computation) features include: parallel vector and matrices (several sparse storage formats and easy, efficient assembly), parallel linear, nonlinear equation solvers, and ODE integrators. See Section A.2 for more details about this library.
- TAO (Toolkit for Advanced Optimization) is a software for large-scale optimization problems, such as JoInv, and the current version has algorithms for unconstrained and bound-constrained optimization. See Section A.3 for more details about this library.

Documentation: Doxygen

- generates on-line and off-line documentation (LaTeX) directly from the source code;
- generates graph and diagram directly from the source code.

3.4 JoInv structure

In Figure 3.3 the streamlined JoInv call graph is shown. Violet blocks are the only functions called by a JoInv end user; we will came back to a basic JoInv usage in the following Section 3.6. The black blocks in the top of the graph are the initialization functions, where the data and the parameters are loaded and set. The application core starts from the `JoinvApplicationSolve` violet block; form here we evaluate the Tikohonov-like objective functional (orange blocks) by computing the misfit term (blue blocks), the regularization term (green blocks) and the joining term (red block). At the bottom of the graph there are the finalization functions, that check for options to be called at the conclusion of the program and print the requested data and parameters.

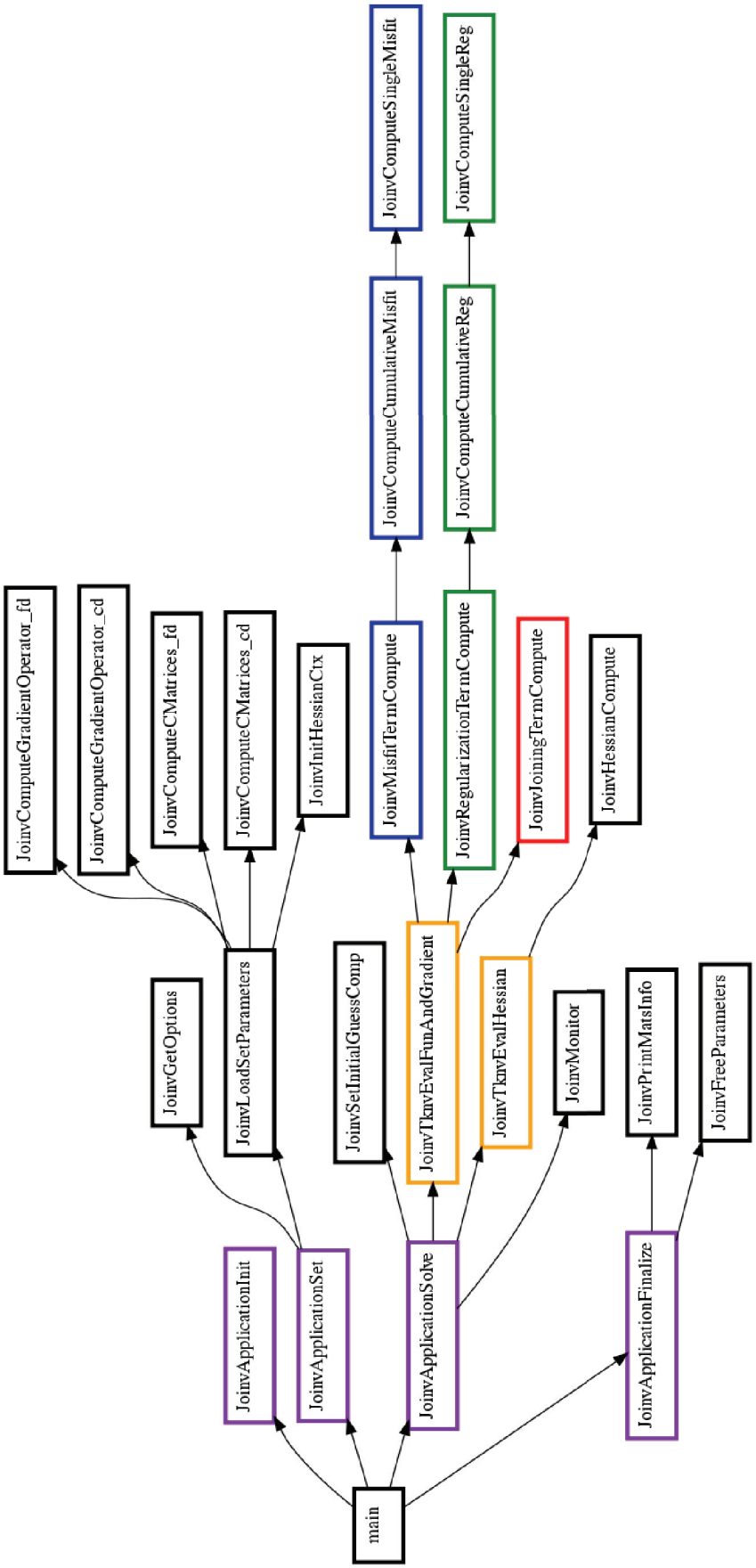


Figure 3.3: Streamlined JoInv call graph.

3.5 From sequential to parallel code

Since JoInv is written using PETSc and TAO libraries, that are already designed for message-passing parallel applications, changing the code from the sequential to the parallel version was not too difficult: most details of message-passing are hidden within parallel objects, such as vectors, matrices and solvers. Nevertheless, a number of important changes have been implemented to optimize parallel computations and minimize process communications.

3.5.1 Input and output

In the sequential code all the print statements, both on standard output and file, are done by calling

```
PetscPrintf(MPI_COMM_SELF, ...)
```

In the parallel version, when only the first process in the communicator has to print a message, the call becomes

```
PetscPrintf(MPI_COMM_WORLD, ...)
```

Instead, when we need synchronized output from several processors, such as for sorted outputs, the call becomes

```
PetscSynchronizedPrintf(MPI_COMM_WORLD, ...)
PetscSynchronizedFlush(MPI_COMM_WORLD)
```

`PetscSynchronizedFlush()` flushes the output from all processors involved in previous `PetscSynchronizedPrintf()` calls to the screen.

3.5.2 Parallel vectors and matrices

A^TB multiplication

In the sequential version the $C = A^TB$ matrix-matrix multiplication is performed by

```
MatMatMultTranspose(Mat A, Mat B, MatReuse scall, PetscReal fill,
                    Mat *C)
```

This routine is currently only implemented for pairs of `SeqAIJ` matrices, pairs of `SeqDense` matrices and classes which inherit from `SeqAIJ`; therefore we replaced that call with the following couple of calls, that work on `ParAIJ` matrices too:

```
MatTranspose(Mat A, MatReuse reuse, Mat *At)
MatMatMult(Mat At, Mat B, MatReuse scall, PetscReal fill, Mat *C)
```

We recall here that these two functions are very expensive in parallel, because they require a large number of data movement and communications, and that PETSc developers recommend to avoid them whenever is possible.

Local rows

When parallel vectors and matrices are created, they are distributed over all processes in the MPI communicator; the number of components that will be stored on each process are decided by PETSc.

For parallel vectors, that are distributed across the processes by ranges, it is possible to determine a process local range with the routine

```
VecGetOwnershipRange(Vec vec, int *low, int *high)
```

The argument `low` indicates the first component owned by the local process, while `high` specifies one more than the last owned by the local process. This command is useful, for instance, in assembling parallel vectors or for getting some values from local components, *e.g.*:

```
VecGetArray(Vec x, PetscScalar *a[])
VecGetValues(Vec x, PetscInt ni, const PetscInt ix[], PetscScalar y[])
```

For parallel matrices, that are partitioned by default with contiguous rows owned by the same process, the routine

```
MatGetOwnershipRange(Mat A, int *firstRow, int *lastRow);
```

informs the user that all rows from `firstRow` to `lastRow-1` will be stored on the local process (since the value returned in `lastRow` is one more than the global index of the last local row). This function is useful because, even though one may insert values into PETSc matrices with no regard to which process eventually stores them, for efficiency reasons it is recommended to generate most of the entries locally to the process that will own them, so that communications are minimized.

Getting the structure of a matrix row

In the sequential code, to get the compressed row storage indices i and j of sequential matrices, we call

```
MatGetRowIJ(Mat mat, PetscInt shift, PetscTruth symmetric,
            PetscTruth inodecompressed,
            PetscInt *n, PetscInt *ia[], PetscInt* ja[],
            PetscTruth *done)
```

This routine is currently implemented only for sequential matrices, therefore we replaced that call by

```
MatGetRow(Mat mat, PetscInt row, PetscInt *ncols,
          const PetscInt *cols[], const PetscScalar *vals[])
```

that gets a row of a parallel matrix.

3.6 JoInv basic usage

This chapter discusses the most relevant routines that a JoInv end user should call. They are: `JoinvApplicationInit()`, `JoinvApplicationFinalize()`, `JoinvApplicationSet()`, and `JoinvApplicationSolve()`.

In Figure 3.4 violet blocks show the four functions that a user must call to minimize a functional using JoInv. The black blocks show where the TAO functions are called. Finally, the blue blocks indicates JoInv's core routines.

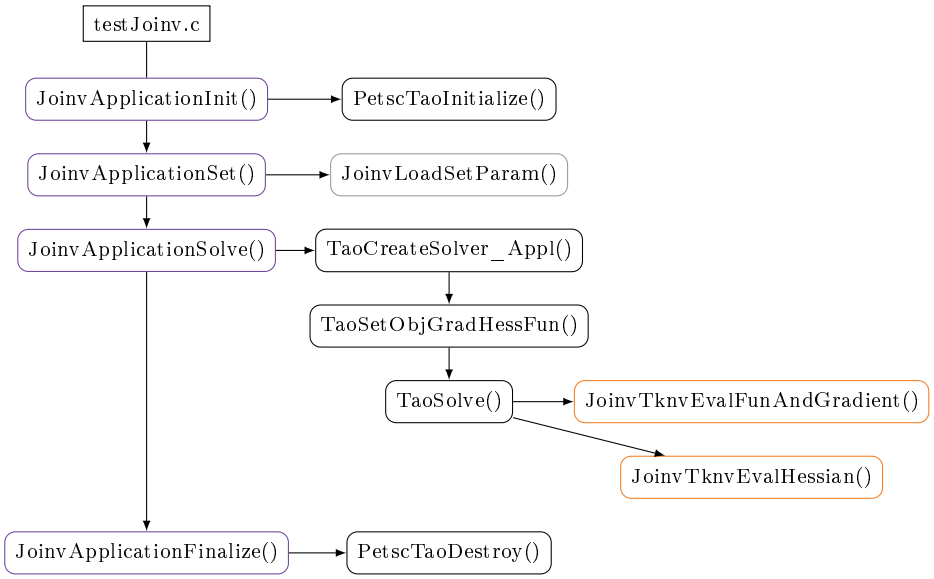


Figure 3.4: JoInv functions calls from the end user point of view.

A simple example of JoInv usage is provided in Listing 3.1; more details about the called routines can be found in the next sections.

```
1  /**
2   * testJoinv.c
3   *
4   * Test program for the Joinv pkg.
5   *
6   * Petsc my_config:
7   *      petsc_scalar: --with-scalar-type=real --with-precision=double
8   *
9   * Program usage:
10  *      make testJoinv
11  *      mpirun -n N ./testJoinv.x [-help] [all options]
12  */
13
14 #include <stdlib.h>
15 #include <stdio.h>
16
17 #include "JoinvApplication.h"
18
19 #undef __FUNCT__
20 #define __FUNCT__ "main"
21
22 int main(int argc, char *argv[])
23 {
24   JoinvErrorCode errInfo;
25   JoinvAppCtx    joinvCtx;
26
27   /* JoInv options file name */
28   char joinvOptFile[] = "joinvOptions";
```

```
29
30   /* Initialize JoInv, TAO, PETSc, and MPI */
31   errInfo = JoinvApplicationInit(&argc, &argv, joinvOptFile, &joinvCtx);
32
33   /* Set and load data and parameters */
34   errInfo = JoinvApplicationSet(&joinvCtx);
35
36   /* Solve the minimization problem */
37   errInfo = JoinvApplicationSolve(&joinvCtx);
38
39   /* Finalize JoInv, TAO, PETSc, and MPI */
40   errInfo = JoinvApplicationFinalize(&joinvCtx);
41
42   return errInfo;
43 }
```

Listing 3.1: Example of JoInv application code. After any call you should check the value of `errInfo` because the functions return a nonzero error code when they fail; here is not done for the sake of simplicity.

3.6.1 Initialization and finalization

The first JoInv routine in any application should be `JoinvApplicationInit()`. Most JoInv programs begin with a call to

```
JoinvAppCtx joinvCtx;

info = JoinvApplicationInit(
           int *argc, char ***argv,
           char *optFile, JoinvAppCtx *joinvCtx);
```

This command initializes JoInv, TAO as well as MPI, PETSc and other packages to which TAO applications may link (if these have not yet been initialized elsewhere). In particular, the arguments `argc` and `argv` are the command line arguments delivered in all `C` and `C++` programs; these arguments initialize the options database.

The argument `optFile` optionally indicates an alternative name for an options file, which by default is called `.petscrc` and is placed in the user's home directory.

The argument `joinvCtx` is a `JoinvAppCtx`: it is a `struct` including all the JoInv parameters.

One of the last routines that all JoInv programs should call is

```
info = JoinvApplicationFinalize(JoinvAppCtx *joinvCtx);
```

This routine finalizes JoInv and any other libraries that may have been initialized during the `JoinvApplicationInit()` phase.

3.6.2 Setting

The routine

```
info = JoinvApplicationSet(JoinvAppCtx *joinvCtx);
```

sets many JoInv parameters from the command line arguments or the options file. It must be called after `JoinvApplicationInit()` and before `JoinvApplicationSolve()`.

3.6.3 Solve

The routine

```
info = JoinvApplicationSolve(JoinvAppCtx *joinvCtx);
```

will apply the JoInv algorithm to the application that has been created by the user.

3.6.4 JoInv options

A complete list of JoInv options can be found in Table 3.2, Table 3.3 and Table 3.4. A sample JoInv options file is the following:

```
# FILE: joinvOptions
#
# ---------------
# MISFIT term opt
# ---------------
#
-joinv_misfit_type norm2
-joinv_cum_misfit_type weightedSum
-joinv_m1_operator_type 0
-joinv_m2_operator_type 0
-joinv_misfit_weights 1.0,1.0
#
# ------------------------
# REGULARIZATION term opt
# ------------------------
#
-joinv_reg_type minimumSupport
-joinv_cum_reg_type weightedSum
-joinv_lambda_1 2.0892
-joinv_lambda_2 0.5755
-joinv_xi_reg 5.0128,0.5012,0.3
-joinv_reg_weights 1.0,1.0
#
# ------------------------
# JOINT term opt
# ------------------------
#
-joinv_joint_type joinvMinimumSupport
-joinv_lambda_jf 1.2
-joinv_xi_jf 0.3,0.3,0.3
#
# ------------------------
# TAO opt
# ------------------------
#
-tao_method tao_nls
```

```
-tao_nls_ksp_type stcg
-joinv_tao_maxIter 1
-joinv_tao_fatol 1.0e-6
-joinv_tao_frtol 1.0e-6
-joinv_tao_catol 0.0
-joinv_tao_crtol 0.0
#
# ------------------------
# OTHER opt
# ------------------------
#
-joinv_fd_type forward
-joinv_verbosity 1
-joinv_iter_info 1
-joinv_mat_info 0
-joinv_view_tao_solver 1
#
# ------------------------
# IN - OUT FILE opt
# ------------------------
#
-joinv_in_A_1 toyModel/m1/A_matrix_petsc
-joinv_in_A_2 toyModel/m2/A_matrix_petsc
-joinv_in_d_1 toyModel/m1/Traveltimes_petsc
-joinv_in_d_2 toyModel/m2/Traveltimes_petsc
-joinv_in_celNum toyModel/cellNumXYZ.txt
-joinv_in_celDim toyModel/cellDimXYZ.txt
-joinv_in_bckgrndVel_apr toyModel/bckgrndVelApr.txt
#
-joinv_out_params 1
-joinv_out_params_fileName parametersLoaded.txt
-joinv_out_params_path dataOut/
-joinv_out_hessians 1
-joinv_out_gradients 1
#
# ------------------------
# MONITOR opt
# ------------------------
#
-joinv_monitor 1
-joinv_monitor_solVec monitor-sol_iter-
-joinv_monitor_misfit monitor-misfitVals
-joinv_monitor_reg monitor-regVals
-joinv_monitor_mix monitor-mixVals
-joinv_monitor_tknv monitor-tknvVals
```

Name	Values	Default	Description
-joinv_misfit_type	norm2	norm2	Misfit term type.
-joinv_cum_misfit_type	weightedSum	weightedSum	Cumulative misfit term type
-joinv_m1_operator_type	0 | 1	0	Choice between linear (0) or nonlinear operator (1), misfit term of model_1.
-joinv_m2_operator_type	0 | 1	0	Choice between linear (0) or nonlinear operator (1), misfit term of model_2.
-joinv_misfit_weights	double,double	1.0,1.0	Array of two real values that must be separated with commas with no intervening spaces.
-joinv_reg_type	minimumSupport	minimumSupport	Regularization term type.
-joinv_cum_reg_type	weightedSum	weightedSum	Cumulative regularization term type.
-joinv_lambda_1	double	-	λ_1
-joinv_lambda_2	double	-	λ_2
-joinv_xi_reg	double,double,double	-	ξ_1 and ξ_2. Array of values that must be separated with commas with no intervening spaces.
-joinv_reg_weights	double,double	1.0,1.0	Array of two real values that must be separated with commas with no intervening spaces.
-joinv_joint_type	joinvMinimumSupport	joinvMinimumSupport	Joint functional type.
-joinv_lambda_jf	double	-	λ_3 (joint functional lambda)
-joinv_xi_jf	double,double,double	-	ξ_3, ξ_4, ξ_5. Array of values that must be separated with commas with no intervening spaces.
-tao_method	tao_*	-	TAO method See TAO manual [8] for more details.
-joinv_tao_maxIter	int	100	Maximum number of iterations. See TAO manual [8] for more details.
-joinv_tao_fatol	double	1.0e-6	Absolute convergence tolerance. See TAO manual [8] for more details.
-joinv_tao_frtol	double	1.0e-6	Relative convergence tolerance. See TAO manual [8] for more details.
-joinv_tao_catol	double	0.0	Allowable error in constraints. See TAO manual [8] for more details.
-joinv_tao_crtol	double	0.0	Allowable relative error in constraints. See TAO manual [8] for more details.

Table 3.2: JoInv misfit, regularization,joining term, and TAO options.

Name	Values	Default	Description
`-joinv_in_A_1`	path/fileName	-	Binary input file (PETSc binary matrix format) for A_1 matrix (linear operator of model_1).
`-joinv_in_A_2`	path/fileName	-	Binary input file (PETSc binary matrix format) for A_2 matrix (linear operator of model_2).
`-joinv_in_d_1`	path/fileName	-	Binary input file (PETSc binary vector format) for d_1 vector (observed data of model_1).
`-joinv_in_d_2`	path/fileName	-	Binary input file (PETSc binary vector format) for d_2 vector (observed data of model_2).
`-joinv_in_celNum`	path/fileName	-	Text input file for number of cells in x, y, z directions.
`-joinv_in_celDim`	path/fileName	-	Text input file for cells dimension in x, y, z directions.
`-joinv_in_bckgrndVel_apr`	path/fileName	-	Text input file that contains an array of two values that must be separated with commas with no intervening spaces.
`-joinv_out_params`	0 | 1	0	Flag that indicates if you want the output text and binary files with the loaded options and parameters.
`-joinv_out_params_fileName`	path/fileName	-	Optional path of the output binary files with the loaded matrix and vectors.
`-joinv_out_params_path`	path/	-	Optional output text file with the loaded options and parameters. (A dimensions, d dimension, cell number, cell dimension, cell per layer, number of layers, misfit term type, regularization type, joint type, finite difference type, misfit weights, regularization parameters, focal parameters).
`-joinv_out_gradients`	0 | 1	0	Flag indicating if you want the output binary files with the computed gradients.
`-joinv_out_hessians`	0 | 1	0	Flag indicating if you want the output binary files with the computed Hessians.

Table 3.3: JoInv, input and output options.

Name	Values	Default	Description
`-joinv_fd_type`	forward | central	forward	Finite difference type used to compute gradient operators.
`-joinv_verbosity`	0 | 1	1	Enable or disable the display of some information during the iterations (*i.e.*what routine is being executed).
`-joinv_iter_info`	0 | 1	0	Enable or disable the display of information about the iterations.
`-joinv_mat_info`	0 | 1	0	Enable or disable the display of information about matrix storage.
`-joinv_view_tao_solver`	0 | 1	0	Enable or disable the display of the `TAO_SOLVER` data structure.
`-joinv_monitor`	0 | 1 | 2	0	Flag indicating how much output information about each iteration do you want (*i.e.*solution vector, misfit term values, regularization term values, joining term values, Tikhonov functional values). 0 - No informations; 1 - Informations printed only on standard output (monitor); 2 - Informations printed on standard output (monitor) and binary files.
`-joinv_monitor_solVec`	fileName	-	File name of the file that contains the solution vector computed at each iteration.
`-joinv_monitor_misfit`	fileName	-	File name of the file that contains the misfit term values computed at each iteration.
`-joinv_monitor_reg`	fileName	-	File name of the file that contains the regularization term values computed at each iteration.
`-joinv_monitor_mix`	fileName	-	File name of the file that contains the joining term values computed at each iteration.
`-joinv_monitor_tknv`	fileName	-	File name of the file that contains the Tikhonov functional values computed at each iteration.

Table 3.4: JoInv monitor options.

Chapter 4

Structure prediction of sparse matrix products

We consider the problem of predicting the nonzero structure of a product of two or more sparse matrices. The intention is to predict that structure using a low cost procedure before performing the computation-intensive matrix multiplication. The nonzero structure can serve as a guide for efficient memory allocation for the output matrix when performing the actual multiplication. Indeed, extensive literature exists on more efficient storage schemes and algorithms for matrix operations on sparse matrices (see, *e.g.*, [23, 52–54], just to mention some).

Throughout this chapter we assume that no cancellation occurs. That is, the inner product of two vectors with overlapping nonzero entries never cancels to zero. This is a very common assumption in the mathematical programming literature. The justification is that cancellations are unlikely when computing over real numbers. Furthermore, even when, due to a singular structure, cancellation occurs, rounding errors may prevent the resulting computed value from being zero.

We consider the goal of determining the full nonzero structure (that is, the exact locations of all nonzero entries) of a matrix products in a time proportional to the number of nonzero elements.

We experimentally tested our method on products of four randomly-generated large sparse matrices and on products of the matrices used by JoInv.

4.1 Structure prediction using graph theory

We mention that extensive literature exists on storage schemes and matrix operations on sparse matrices, and many sparse matrices algorithms have a phase that predicts the nonzero structure of the solution from the nonzero structure of the problem.

Graph theory is a useful language in which to state and prove structure prediction results. One reason for this is that the structural effect of a matrix computation often depends on path structure, which is easier to describe in term of graphs than in terms of matrices.

One common method used to predict nonzero structure is to represent the matrix A of dimension $n \times m$ by a bipartite graph G_A, with n nodes $\{v_1, \ldots, v_n\}$ at the lower level and m nodes $\{w_1, \ldots, w_m\}$ at the upper level.

In this way, a product $A = A_1 \dots A_L$ can be represented as a layered graph with $(L+1)$ layers as follows. Suppose A_i is of dimensions $n_i \times n_j$. For $1 \leq i \leq L$, the ith layer has n_i nodes and the edges between the ith and the jth layers are as is the bipartite graph G_{A_i}.

For $1 \leq r < r' \leq L$, consider the sub-product $B = A_r A_{r+1} \dots A_{r'}$. The following prepositions are immediate:

- Assuming no cancellation, the number of nonzero entries in the ith row of B equals the number of nodes in layer $r'+1$ that the ith node in layer r reaches via directed paths.
- The number of nonzero entries in the ith column of B is equal to the number of nodes in layer r that can reach the ith node in layer $r'+1$.

For example, consider the following nonzero structures (x denotes a nonzero entry):

$$A = \begin{pmatrix} x & 0 & 0 & 0 \\ 0 & 0 & x & x \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 0 & x & 0 & 0 & x \\ x & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & 0 & x \end{pmatrix} \quad C = \begin{pmatrix} 0 & 0 & 0 \\ x & 0 & 0 \\ 0 & 0 & x \\ 0 & x & 0 \\ x & 0 & 0 \end{pmatrix}$$

The goal is to compute the product $R = ABC$, so that

$$R = \begin{pmatrix} x & 0 & 0 \\ x & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Using the graph representation of the product of Figure 4.1, it can be easily recognized that from $A_{1,1}$ there are direct paths only to $C_{2,1}$, via $B_{1,2}$ or $B_{1,5}$. This means that $R_{1,1}$ is a nonzero entry, where the R indices are the A row index and the C column index, respectively. You can also see that the other nonzero entry is $R_{2,1}$, because there is a direct path from $A_{2,4}$ to $C_{5,1}$, via $B_{4,5}$.

4.2 Structure prediction without graph

From the graph representation one can figure out that the column indices of A are also the row indices of B and that the column indices of B are also the row indices of C. We started from this observation to write an algorithm that finds the nonzero structure of a matrix products without using the graph theory. The algorithm is shown in Algorithm 4.1 and Algorithm 4.2.

In Figure 4.2 we show how this algorithm works on the matrices A, B, C of the previous example.

Using this algorithm we can reduce the allocated memory required for the computation, without the need to create and save the structure of the graph that represents the product of the matrices, but only a couple of array for storing row and column indices of nonzero elements.

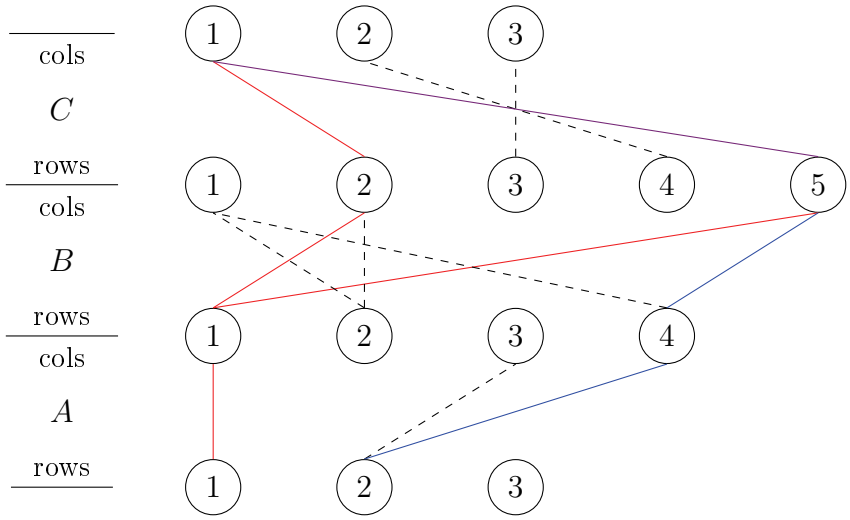


Figure 4.1: The layered graph of the product ABC. From node 1 of a row in A one can reach, via direct path, the node 1 of a column of C; this means that the resulting matrix R has a nonzero element in $R_{1,1}$. It can also be seen that from node 2 of A's rows one can reach, via direct path, node 1 of C's columns; this means that the resulting matrix R has another nonzero element in $R_{2,1}$. In other words, if there is a direct path from the ith row (node) of the first matrix and the jth column (node) of the last matrix, the resulting matrix has a nonzero element in position (i, j).

Algorithm 4.1 Find nonzero structure of matrix products (pseudo-code) – Part 1

```
1: for all rowA1 rows of the first matrix A_1 do
2:   m ← 1
3:   indsCol ← column indices of A_1 nonzero elements in row rowA1
4:   for all i in indsCol do
5:     indRowNextMat ← indsCol[i]
6:     call findNnzRec(m+1, indRowNextMat, outNzColInd)
7:   end for
8:   save Nonzero elements in (rowA1, outNzColInd)
9: end for
```

Time complexity analysis

To understand that the time complexity of Algorithm 4.1, doesn't depend on input data dimension but on the number of nonzero elements involved in the computation, consider the product of three sparse matrices $A_{m\times n}$, $B_{n\times p}$ and $C_{p\times q}$.

Let's define $\mathcal{R}_A$ as the set of all rows in A with at least one nonzero element and $\mathcal{C}_A$ as the set of all columns in A with at least one nonzero element:

$$\mathcal{R}_A = \{i \in \{1, \ldots, m\} \,|\, \exists\, a_{ij} \neq 0, 1 \leq j \leq n\}$$

$$\mathcal{C}_A = \{j \in \{1, \ldots, n\} \,|\, \exists\, a_{ij} \neq 0, 1 \leq i \leq m\}$$

We call the number of rows and columns with at least one nonzero element

$$\nu_A = \#\mathcal{R}_A \text{ and } \mu_A = \#\mathcal{C}_A,$$

Algorithm 4.2 Find nonzero structure of matrix products (pseudo-code) – Part 2

```
1: function findNnzRec(m, indRow, outNzColInd, lastMat)
2: indsCol ← column indices of A_m nonzero elements in row indRow
3: if m == lastMat then
4:   return outNzColInd ← indsCol
5: end if
6: for all i in indsCol do
7:   indRowNextMat ← indsCol[i]
8:   call findNnzRec(m+1, indRowNextMat, outNzColInd)
9: end for
```

respectively.

We define $\mathcal{R}_B$, $\mathcal{C}_B$, ν_B, μ_B, $\mathcal{R}_C$, $\mathcal{C}_C$, ν_C, μ_C in the same way.

When we compute AB we are interested in the set of elements' indices that are nonzero both in the i-th row of A and in the j-th column of B, that are

$$\mathcal{N}_{AB}(i,j) = \{k \in \{1,\dots,n\} \,|\, a_{ik}b_{kj} \neq 0, i \in \mathcal{R}_A, j \in \mathcal{C}_B, k \in \mathcal{C}_A \cap \mathcal{R}_B\};$$

its cardinality is:

$$\eta_{AB}(i,j) = \#\mathcal{N}_{AB}(i,j) \,\forall\, i \in \mathcal{R}_A, j \in \mathcal{C}_B.$$

In other words, $\eta_{AB}(i,j)$ is the maximum number of elements that are nonzero in both the i-th row of A and the j-th column of B, that is the maximum number of nonzero elements in the matrix AB.

Let's now introduce the sets of indices for each row and for each column with nonzero elements as:

$$\mathcal{J}_A(i) = \{j \in \{1,\dots,n\} \,|\, a_{ij} \neq 0\}$$

and

$$\mathcal{I}_A(j) = \{i \in \{1,\dots,m\} \,|\, a_{ij} \neq 0\}.$$

Hence we can rewrite $\mathcal{R}_A$ and $\mathcal{C}_A$ as

$$\mathcal{R}_A = \bigcup_{i=1}^{m} \mathcal{J}_A(i),$$

$$\mathcal{C}_A = \bigcup_{j=1}^{n} \mathcal{I}_A(j).$$

We can also rewrite their cardinality as

$$\nu_{A,i} = \#\mathcal{J}_A(i)$$

and

$$\mu_{A,j} = \#\mathcal{I}_A(j).$$

Next, we define the maximum number κ_A of nonzero elements in one row and, analogously, the number τ_A of nonzero elements in one column:

$$\kappa_A = \max_{1 \leq i \leq m} \{\nu_{A,i}\}$$

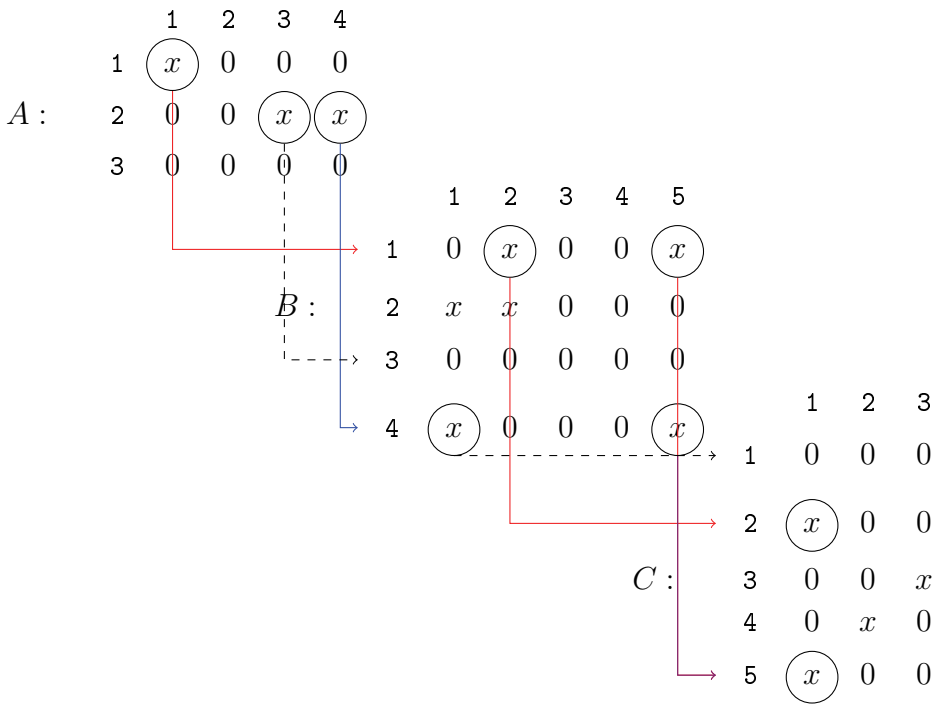


Figure 4.2: The sparse matrix products ABC. As seen in Figure 4.1, from $A_{1,1}$ nonzero element it can be reached, using the column index of this entry in A as the row index of the next matrix B, the nonzero elements $B_{1,2}$ and $B_{1,5}$; from these one can find the nonzero entries $C_{2,1}$ and $C_{5,1}$. That means that the resulting matrix R has a nonzero element in $R_{1,1}$. In the same way, from element $A_{2,4}$ one can reach, via $B_{4,5}$, the $C_{5,1}$ nonzero entry. It follows that the resulting matrix R has another nonzero element in $R_{2,1}$. In other words, if there is a path between a nonzero entry in the ith row of the first matrix and almost one nonzero entry in the jth column of the last matrix, then the resulting matrix has a nonzero element in position (i, j).

and

$$\tau_A = \max_{1 \le j \le n} \{\mu_{A,j}\}.$$

Thus, we can rewrite $\mathcal{N}_{AB}(i, j)$ as

$$\mathcal{N}_{AB}(i,j) = \{k \in \{1,\dots,n\} \,|\, a_{ik}b_{kj} \neq 0, k \in \mathcal{J}_A(i) \cap \mathcal{I}_B(j), \forall i \in \mathcal{R}_A, \forall j \in \mathcal{C}_B\}\,.$$

Its cardinality, that is the maximum number of nonzero elements in the resulting matrix AB, have an upper bound:

$$\eta_{AB}(i,j) = \#\mathcal{N}_{AB}(i,j) \le \min_{i \in \mathcal{R}_A, j \in \mathcal{C}_B} \{\mathcal{J}_A(i), \mathcal{I}_B(j)\} \le \min\{\kappa_A, \tau_B\}.$$

Let's introduce analogous definitions for the matrix C:

$$\nu_C = \#\{i \in \{1,\dots,p\} \,|\, \exists\, c_{ij} \neq 0, 1 \le j \le q\}$$

$$\mu_C = \#\{j \in \{1,\dots,q\} \,|\, \exists\, c_{ij} \neq 0, 1 \le i \le p\}$$

It's now easy to estimate the number of nonzero elements of the final matrix ABC. Observe that

$$\max(\text{nnz}(AB)) = \nu_A \mu_B;$$

we can write

$$\nu_{AB} \leq \nu_A\,, \quad \mu_{AB} \leq \mu_B$$

and

$$\nu_{ABC} \leq \nu_{AB}\,, \quad \mu_{ABC} \leq \mu_C$$

therefore

$$\max(\text{nnz}(ABC)) = \nu_{AB}\mu_C \leq \nu_A \mu_C.$$

We can finally write the time complexity of our algorithm for the matrix products ABC:

$$\max(\#\text{flops}) \leq \nu_A \mu_B + \nu_{AB}\mu_C$$

4.3 Sequential implementation

In Listing 4.1 and Listing 4.2 we report an outline sequential version of the Algorithm 4.1 implemented using PETSc data type and functions.

The time complexity of this implementation is reported in Table 4.1.

```
/**
 * [in]      arrayMat    - array of the matrices.
 * [in]      numMatMax   - number of matrices involved in the product.
 * [in]      nnz         - number of nonzero elements.
 * [out]     nz_row_ind  - row indices of nonzero elements.
 * [out]     nz_col_ind  - column indices of nonzero elements.
 * [return] Error code
*/
int findNnz(Mat* arrayMat, int numMatMax, int* nnz,
            PetscInt* nz_row_ind[], PetscInt* nz_col_ind[]){

  // [...]

  matIndex = 0;
  MatGetRowIJ(arrayMat[matIndex], 0, PETSC_FALSE, PETSC_FALSE,
              &matZero_nrow, &ia, &ja, &done);

  // [...]

  for(i=0; i < matZero_nrow ; i++){ // for each row in arrayMat[0]

    j_start       = ia[i];
    j_end         = ia[i+1];
    tmp_iter_nnz  = 0;

    for(j=j_start ; j<j_end ; j++){ // for each nonzero element

      indRowNextMat = ja[j];

      err = findNnzRec(arrayMat, numMatMax, matIndex+1, indRowNextMat, i,
            &tmp_iter_nnz, tmp_iter_nz_row_ind, tmp_iter_nz_col_ind);
    }
  }

  qsort(/*...*/);

  unique(/*...*/);

  //[...]

```

```
}
```

Listing 4.1: Find nonzero structure of a matrix product.
(`C` -function `findNnz`)

```
/**
 * [in]      arrayMat     - array of the matrices.
 * [in]      numMatMax    - number of matrices involved in the product.
 * [in]      matIndex     - index of matrix in which to look for nz elements.
 * [in]      indRow       - the row in matIndex you are looking for nz elem.
 * [in]      rowFirstMat  - index of the row of the first matrix considered.
 * [in,out] nz           - number of nonzero elements.
 * [in,out] nz_row_ind   - row indices of nonzero elements.
 * [in,out] nz_col_ind   - column indices of nonzero elements.
 * [return] Error code
*/
int findNnzRec(Mat* arrayMat, int numMatMax, int matIndex, int indRow,
    int rowFirstMat, int* tmp_nnz, PetscInt* tmp_nz_row_ind,
    PetscInt* tmp_nz_col_ind){

  //[...]

  MatGetRowIJ(arrayMat[matIndex], 0, PETSC_FALSE, PETSC_FALSE,
              &mat_nrow, &ia, &ja, &done);

  //[...]

  j_start = ia[indRow];
  j_end   = ia[indRow+1];

  for(j=j_start ; j<j_end ; j++){ // for each nz element in indRow

    if (matIndex == (numMatMax-1)){
      tmp_nz_row_ind[*tmp_nnz] = rowFirstMat;
      tmp_nz_col_ind[*tmp_nnz] = ja[j];
      *tmp_nnz = *tmp_nnz + 1;
      continue;
    }

    indRowNextMat = ja[j];

    err = findNnzRec(arrayMat, numMatMax, matIndex+1, indRowNextMat,
              rowFirstMat, tmp_nnz, tmp_nz_row_ind, tmp_nz_col_ind);

  }

  //[...]

}
```

Listing 4.2: Find nonzero structure of a matrix product.
(`C` -function `findNnzRec`)

4.4 Parallel implementation

We also implemented the MPI parallel version of our algorithm that finds the nonzero elements position in a matrix that is the result of a product of any number of sparse matrices.

As we shown before, that algorithm is split into two functions: the first one, `findNnz_mpi()`, works on all the rows of the first matrix; the second one, `findNnzRec_mpi()`, is the recursive function that works on all the other matrices.

Listing	Line	Cost	Times
Listing 4.1	14-16	c_1	1
Listing 4.1	20	c_2	$\mathcal{R}+1$
Listing 4.1	22-24	c_3	$\mathcal{R}$
Listing 4.1	28-31	c_4	$\mathcal{R} * \mathcal{Z}^{0r}$
Listing 4.1	35	$\mathcal{O}(\mathcal{Z} \log \mathcal{Z})$	1
Listing 4.1	37	$\mathcal{O}(\mathcal{Z})$	1
Listing 4.2	18-24	c_5	1
Listing 4.2	26	c_6	$\mathcal{Z}^{nr}+1$
Listing 4.2	28-35	c_7	$\mathcal{Z}^{nr}$
Listing 4.2	37-38	$c_8 + kT(\mathcal{Z}^{mj})$	$\mathcal{Z}^{nr}$

Table 4.1: Time complexity of `findNnz` and `findNnzRec` functions that find nonzero elements of sparse matrix product. $\mathcal{R}$ is the number of rows of the first matrix; $\mathcal{Z}^{0r}$ is the number of nonzero elements of the first matrix in row r; $\mathcal{Z}^{nr}$ is the number of nonzero elements of matrix n in row r; $\mathcal{Z}^{mj}$ is the number of nonzero elements of matrix $m = n+1$ in row j; k is the number of the next matrices in the product; $\mathcal{Z}$ is the number of nonzero elements in the result matrix.
Note: to achieve an optimal complexity you must use a function that returns the nonzero elements of a matrix row in constant time like `MatGetRowIJ` of PETSc, that store the matrices in the CSR format (Figure A.3).

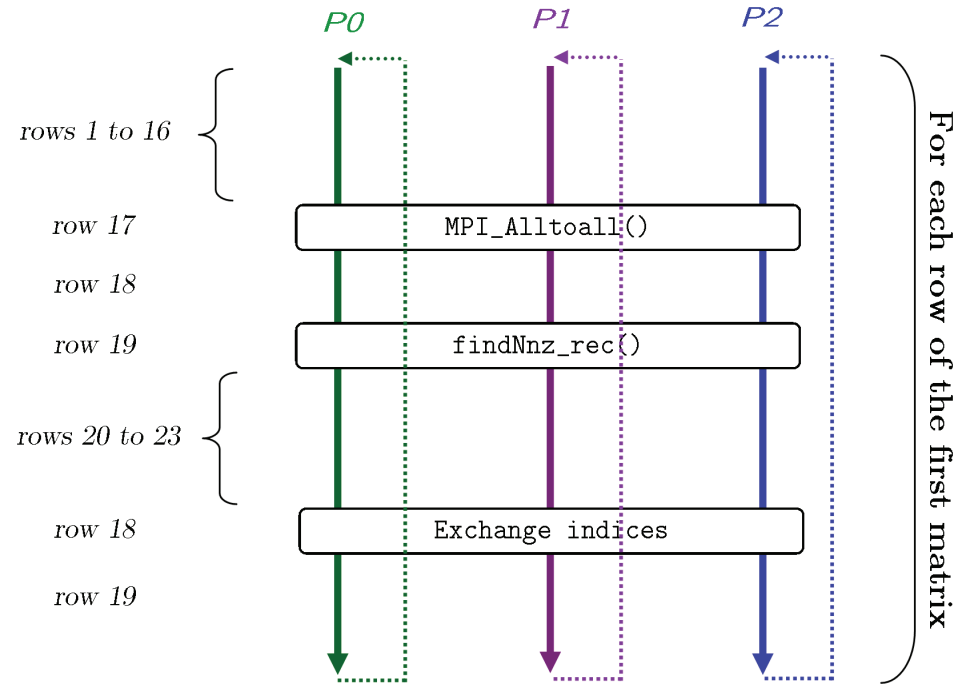
When we work with parallel sparse matrices using the PETSc `MATMPIAIJ` matrix format, each process owns a contiguous portion of the matrix rows (that is a "block distribution" is used); for instance, a 8×4 matrix on 3 processes will be partitioned by default with three rows on the first process, three on the second and two on the last process. Due to that matrix partitioning across the processes, the parallel version needs communication between processes in the following situation: suppose there are L factors $A_1, \ldots, A_L$ in the product and the factor A_ℓ, $1 \leq \ell < L$, is currently under investigation; process k, say, finds a nonzero element in column j and, at the same time, it does not own the row j of the next matrix $A_{\ell+1}$. The diagram in Figure 4.3 summarize the MPI communications between 3 processes when they run `findNnz_mpi()` and `findNnzRec_mpi()`; a detailed explanation of such communications is in Subsection 4.4.1.

4.4.1 Parallel algorithm description

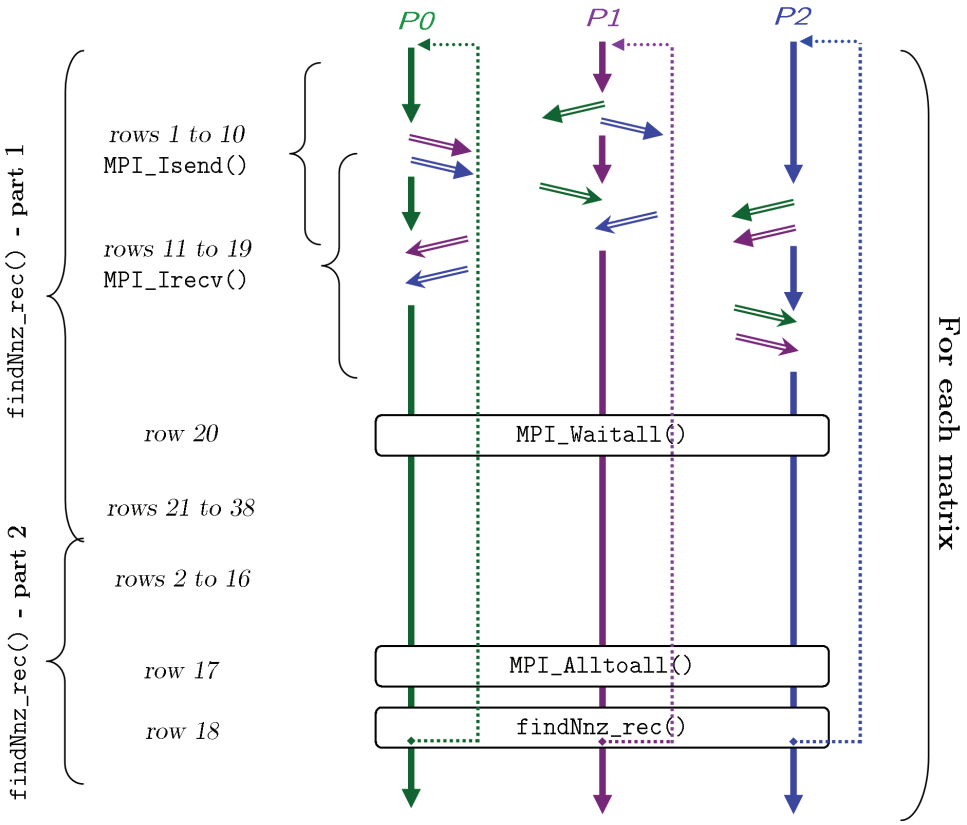
Let's explain in detail the parallel algorithm shown in Algorithm 4.3, Algorithm 4.4 and Algorithm 4.5 using the example in Figure 4.4.

We start from Algorithm 4.3. In what follows we describe the steps of the parallel function `findNnz_mpi()`, that analyzes the rows of the first matrix and, for each such row, calls the recursive function.

1. During the first step of the algorithm, each process analyzes one row of the first matrix. To obtain the indices i and j of the compressed row storage for sequential matrices, we use the PETSc function
   `MatGetRowIJ()`; that function does not work on the `MATMPIAIJ` matrix type, so in



(a) Algorithm 4.3, `findNnz_mpi()` MPI communications



(b) Algorithm 4.4 and Algorithm 4.5, `findNnzRec_mpi()` MPI communications

Figure 4.3: MPI communications between three processes $(P0, P1, P2)$ when running `findNnz_mpi()` and `findNnzRec_mpi()` functions. Rectangle shapes indicate MPI blocking calls; split arrows indicate nonblocking send and receives calls.

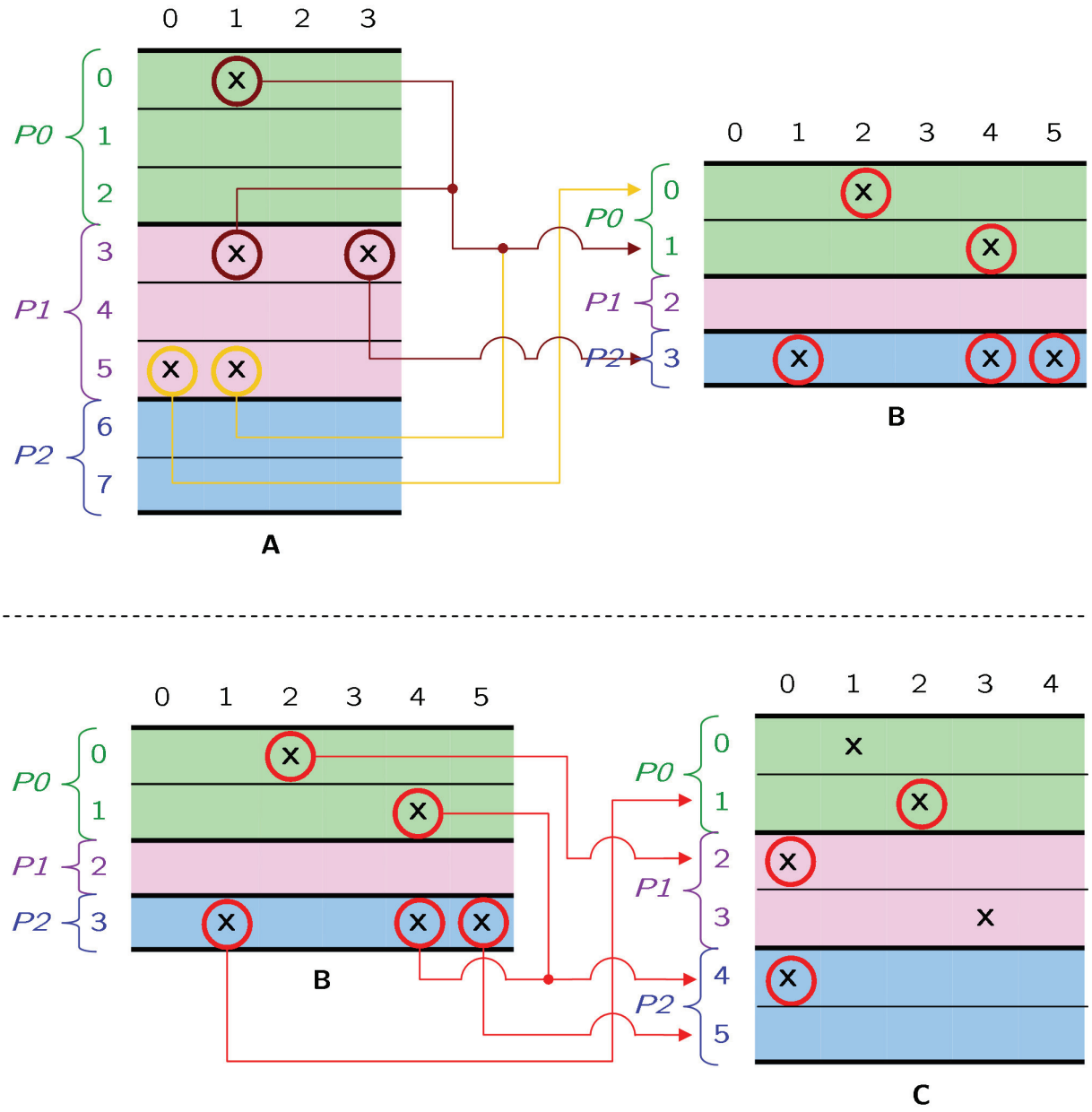


Figure 4.4: The parallel sparse matrix product ABC. Each matrix is split into group of contiguous rows that are owned by a single process; in this example $P0$ has got the first three rows of A, $P1$ has the second group of three rows, and $P2$ owns the last two rows; B and C are spread among processes in the same way. As it was seen in Section 4.2, starting from every nonzero element in position $A_{i,j}$, we can find what will be the coordinates of nonzero elements in the resulting matrix R of a product of matrices. All the rows of the first matrix A are parsed by Algorithm 4.3, one row at a time. For each nonzero value with coordinate $A_{i,j}$, we use the column index j as row index of the next matrix B; then Algorithm 4.4 and Algorithm 4.5 recursively parse the next matrices: for each nonzero value $B_{j,k}$, we use the column index k as row index of the next matrix C. Since C is the last matrix in the product $R = ABC$, if there are nonzero values in $C_{k,w}$ there will be nonzero entries in $R_{i,w}$. The crucial point of the parallel version is that, from the second matrix to the last, one process needs to know from the other processes if there are rows it must analyze and what they are: this involve MPI nonblocking communication between processes in a recursive function.

the parallel version we use `MatGetRow()`, that returns the structure of a single local row. In our example, $P0$ finds one nonzero element in column 1 and $P1$ finds two nonzero elements, one in column 1 and one in column 3, and two nonzero entries on row 5 in columns 0 and 1 (following iterations). **Note.** Let $\overline{m}_1$ the maximum number of local rows of the first matrix in all processes: then, each process that owns less than $\overline{m}_1$ local rows of the first matrix (because of the block distribution), at the first step must perform in any case this iterative scheme exactly $\overline{m}_1$ times, because the subsequent `MPI_Alltoall()` communication needs contribution from all the processes. Pseudo-code lines: from 1 to 9.

2. After that call, each process needs to know, for the column index j of each nonzero element, what process owns the j-th row (in global indexing) in the next matrix. The column indices found are stored in one send buffer, named `sendBuf[k]`, different for each process k that will receive those indices. In the send buffer, after each column index, also the index of the row in the first matrix from which the elements come is stored. Pseudo-code lines: from 10 to 15. In this way, $P0$ puts 1 (column index) in `sendBuf[0][0]` and 0 (row index of the first matrix) in his `sendBuf[0][1]`. Similarly, $P1$ puts 1 (column index) and 3 (row index of first matrix) respectively in his `sendBuf[0][0]` and `sendBuf[0][1]` locations and 3 (column index) and 3 (row index of first matrix) in his `sendBuf[2][0]` `sendBuf[2][1]` locations, respectively. Each process also stores another array, `howManyNzRow`, sized K, where it saves the number of nonzero indices that is going to send to each other process.

3. At this point there is the first `MPI_Alltoall()` communication, in which are sent, from each process to each other, the number of indices that are going to be exchanged. After this call each process know exactly how many requests it is going to receive from each other process; these values are stored in the array `howManyNzRecv`. Pseudo-code line: 17. In the example, $P0$ knows that it must receive one message from itself, $P1$ does not have request and $P2$ must receive messages from $P0$ and $P1$.

4. Now each process is ready to call the recursive function on the next matrix, that will return the number of nonzeros and their indices for one row of the first matrix. Pseudo-code line: 19.

5. When the recursive function `findNnzRec_mpi()` returns, each process adds the new nonzero indices to the previous ones. Pseudo-code lines: from 20 to 22.

6. When all the local rows of the first matrix have been analyzed, each process has the list of nonzero elements that it found in the last matrix. In our example, the list on $P0$ is $\{(3,2)\}$, the list on $P1$ is $\{(5,0)\}$ and the list on $P2$ is $\{(0,0), (3,0), (5,0)\}$. It's easily seen that some processes lists row indices that do not match with its local row indices. So, in these cases other communications are needed to reallocate the position of each nonzero element to the right process. After those MPI non-blocking calls, the algorithm sort the nonzero indices found and stops. At the end, the list of nonzero indices belonging to $P0$ is $\{(0,0)\}$, the one belonging to $P1$ is $\{(3,0), (3,2), (5,0)\}$ and the one on $P2$ is empty. Pseudo-code lines: from 24 to 25.

Algorithm 4.3 Calculate nonzero structure of matrix product (parallel pseudo-code); function `findNnz_mpi()`.

```
1: call MatGetOwnershipRange(): mats[ind] → myRowStart, myRowEnd
2: r ← myRowEnd − myRowStart
3: call MatGetOwnershipRanges(): mats[ind + 1] → rangesFirstMat
4: Compute R // maximum number of rows owned by one proc
5: i ← myRowStart
6: for ii = 0 to ii < R do
7:    howManyNzRow[] ← 0
8:    if ii < r then
9:       call MatGetRow(): i → nColsInRow, ja[]
10:      for j = 0 to j < nColsInRow do
11:         find what process k owns the ja[j]-th row in the next matrix
12:         sendBuf[k][howManyNzRow[k]] ← ja[j]
13:         sendBuf[k][howManyNzRow[k] + 1] ← i
14:         howManyNzRow[k] ← howManyNzRow[k] + 2
15:      end for
16:   end if
17:   call MPI_Alltoall(): howManyNzRow[] → howManyNzRecv[]
18:   iterNnz ← 0
19:   call findNnzRec_mpi() mats[ind + 1] → iterNnz, iterNzRow[], iterNzCol[] .
20:   tmpNnz ← tmpNnz + iterNnz
21:   tmpNzRow[] ← tmpNzRow[] + iterNzRow[]
22:   tmpNzCol[] ← tmpNzCol[] + iterNzCol[]
23: end for
24: Exchange the nonzero indices found between processes
25: Sort and unique on the nonzero indices found
```

We now describe the steps of the parallel recursive function `findNnzRec_mpi()`, shown in Algorithm 4.4 and Algorithm 4.5, that analyzes the matrices from the second to the last and returns the number of nonzeros and their row and column indices. Other inputs and outputs of this function are: the next matrix index, the maximum number of matrices involved in the product, the arrays `howManyNzRow`, `sendBuf` and `howManyNzRecv`.

1. During the first step of the recursive parallel function there are other MPI communications to exchange the indices that are needed to analyze the current matrix:
   - each process $P\ell$, $0 \leq \ell \leq K-1$ checks in `howManyNzRow[k]`, for each $0 \leq k \leq K-1$, if it must send the data in `sendBuf[k]` to process k. In the case, it calls the nonblocking function `MPI_Isend()`. Obviously, for $k = \ell$ it simply copies `sendBuf[k]` in `recvBuf[k]`. Pseudo-code in Algorithm 4.4, lines from 1 to 10.
   - Each process $P\ell$ then checks in `howManyNzRecv[k]` if it must receive something in `recvBuf[k]` form process k, $0 \leq k \leq K-1$: if `howManyNzRecv[k]` $\neq 0$, it calls the nonblocking function `MPI_Irecv()`. Pseudo-code in Algorithm 4.4, lines from 11 to 19.
   - At the end, before continuing, every process must waits for all given communications to complete, by calling `MPI_Waitall()`. Pseudo-code in Algorithm 4.4, line 20.
2. If the process is working on the last matrix involved in the product, A_L, for each element in `recvBuf` it must call `MatGetRow()`, set the output array of rows and columns indices of nonzero elements found and their total number. This completes the inspection and stops the recursion. Pseudo-code in Algorithm 4.4, lines from 21 to 38.
3. If the process is not at the last matrix, for each row index it repeats the steps 1-4 shown in the description of the function `findNnz_mpi()`, Algorithm 4.3. The only difference from those steps is that it has to analyze only the rows whose indices have just been received in its `recvBuf` array, in place of analyzing all rows as it is the case when the first matrix is inspected. Pseudo-code in Algorithm 4.5, lines from 2 to 18.

Remark 4.1 *We underline that the recursive structure of the code allows to predict the sparsity structure of the product of any number of matrices, of any size, dense and sparse, local or distributed, with the only limitation given by stack memory.*

The recursion is guided and synchronized by the factor matrices involved: this ensures the correctness of the recursion, that is the guarantee that it will eventually stops. While this is easily understood if the number of factors is finite, a formal proof can be given and will be addressed in an upcoming work.

This is a relevant contribution to large-scale computation because there is nothing similar in the literature and no code available at all, at the best of our knowledge at the time this work is written.

Algorithm 4.4 Recursively compute the structure of matrix product (parallel pseudo-code); function `findNnzRec_mpi()` - part 1.

```
1: for k = 0 to nProcs do
2:   if k == myRank then
3:     recvBuf[k] ← sendBuf[k]
4:     continue
5:   end if
6:   nSend ← howManyNzRow[k]
7:   if nSend > 0 then
8:     call MPI_Isend(): send sendBuf[k] to process k
9:   end if
10: end for
11: for k = 0 to nProcs do
12:   if k == myRank then
13:     continue
14:   end if
15:   nRecv ← howManyNzRecv[k]
16:   if nRecv > 0 then
17:     call MPI_Irecv(): receive in recvBuf[k] from process k
18:   end if
19: end for
20: call MPI_Waitall()
21: if is the last matrix then
22:   tmpNnz ← 0
23:   for k = 0 to nProcs do
24:     nRecv ← howManyNzRecv[k]
25:     for i = 0 to nRecv do
26:       indNextRow ← recvBuf[k][i]
27:       call MatGetRow(): indNextRow → nColsInRow, ja[]
28:       jj ← 0
29:       for j = tmpNnz to tmpNnz + nColsInRow do
30:         iterNzCol[j] ← ja[jj]
31:         iterNzRow[j] ← recvBuf[k][i + 1]
32:         jj + +
33:       end for
34:       j = tmpNnz ← tmpNnz + nColsInRow
35:     end for
36:   end for
37:   iterNnz ← tmpNnz
38: end if
39: // see Algorithm 4.5 for the second part of this function
```

Algorithm 4.5 Recursively compute the structure of matrix product (parallel pseudo-code); function, `findNnzRec_mpi()` - part 2.

```
1: // see Algorithm 4.4 for the first part of this function
2: call MatGetOwnershipRanges(): mats[ind + 1] → rangesFirstMat
3: howManyNzRow[] ← 0
4: for kk = 0 to nProcs do
5:    nRecv ← howManyNzRecv[kk]
6:    for i = 0 to nRecv do
7:       indNextRow ← recvBuf[kk][i]
8:       call MatGetRow(): indNextRow → nColsInRow, ja[]
9:       for j = 0 to j < nColsInRow do
10:         find what process k owns the ja[j]-th row in the next matrix
11:         sendBuf[k][howManyNzRow[k]] ← ja[j]
12:         sendBuf[k][howManyNzRow[k] + 1] ← recvBuf[kk][i + 1]
13:         howManyNzRow[k] ← howManyNzRow[k] + 2
14:      end for
15:   end for
16: end for
17: call MPI_Alltoall(): howManyNzRow[] → howManyNzRecv[]
18: call findNnzRec_mpi(): mats[ind + 1] → iterNnz, iterNzRow[], iterNzCol[]
19: return
```

4.4.2 Performance analysis

We already had expected beforehand that the parallel implementation of the program that determines the sparsity of the matrix products would be slower than in the sequential one: the computational content of the two functions is almost negligent, which means that there will hardly be any gain in subdividing the work over many cores. Furthermore, the amount of processing per process is decreasing linearly with p, the number of processes, while in `findNnz_mpi()` the communication alone already grows with p^2. Nevertheless, we tested the performance for the sake of completeness.

We tested both the sequential and the parallel version of this code on an IBM-SP6, that is a cluster of 168 Power6 575 nodes that has a peak performance of just over 100 Tflops and an Infiniband 4x DDR network; SP6 is hosted at CINECA Supercomputing Center. For more specific information about SP6 cluster see [118] and the Introduction.

The five matrices used for the experiments are generated randomly by a Matlab code; their dimensions, nonzero elements density, and averaged execution times are reported in Table 4.2.

N. procs	Execution time		
	Test 1	Test 2	Test 3
1	0.001206	0.001068	0.001160
2	0.160565	0.096530	104.348488
4	0.128353	0.074954	33.962100
8	0.093475	0.059371	9.037792
16	0.073490	0.052483	2.543933
32	0.119439	0.141998	0.943525
64	0.363801	0.405729	0.792933
128	1.883175	1.958538	1.582866
256	3.789214	4.289629	4.920911

Table 4.2: Execution time of the parallel Algorithm 4.4 and Algorithm 4.5 tested on five sparse random matrices with dimensions: $A = 5635 \times 5721$, $B = 5721 \times 6648$, $C = 6648 \times 7513$, $D = 7513 \times 6619$, $E = 6619 \times 4257$. For test 1, density of A is 0.001 and density of B, C, D, E is 0.0001; for test 2 the density of all the matrices is 0.0001. For test 3 the matrices dimensions are: $A = 25635 \times 18721$, $B = 18721 \times 16648$, $C = 16648 \times 17513$, $D = 17513 \times 16619$, $E = 16619 \times 24257$; their density is 0.0001.

As it can be seen in Table 4.2, the sequential code is more efficient than the parallel one: this behavior is due to the communication overhead and the use of a different PETSc call in the two versions.

During the analysis of the first matrix, in the sequential code the search of rows with nonzero elements is performed by

```
MatGetRowIJ(Mat mat, PetscInt shift, PetscTruth symmetric,
            PetscTruth inodecompressed,
            PetscInt *n, PetscInt *ia[], PetscInt* ja[],
            PetscTruth *done)
```

that returns the compressed row storage `ia` and `ja` indices for sequential matrices. In this way we can iterate only on the nonzero rows returned in `ia`.

Unfortunately, that function works for sequential matrices only; so in the parallel version, during the analysis of the first matrix, we must iterate over all rows and call the following function, to know if and where there are nonzero elements in one row:

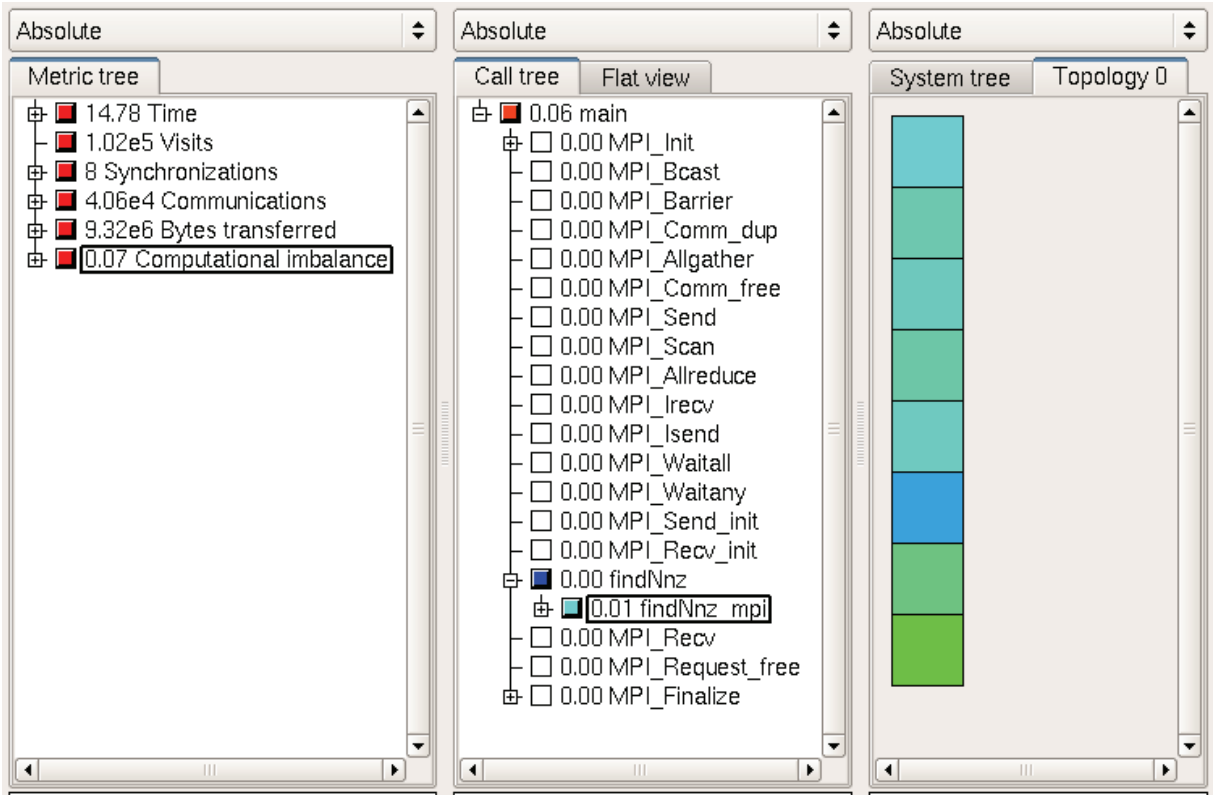
```
MatGetRow(Mat mat, PetscInt row,
          PetscInt *ncols, const PetscInt *cols[],
          const PetscScalar *vals[])
```

That variation brings on some extra computation, especially on the first matrix involved in the product.

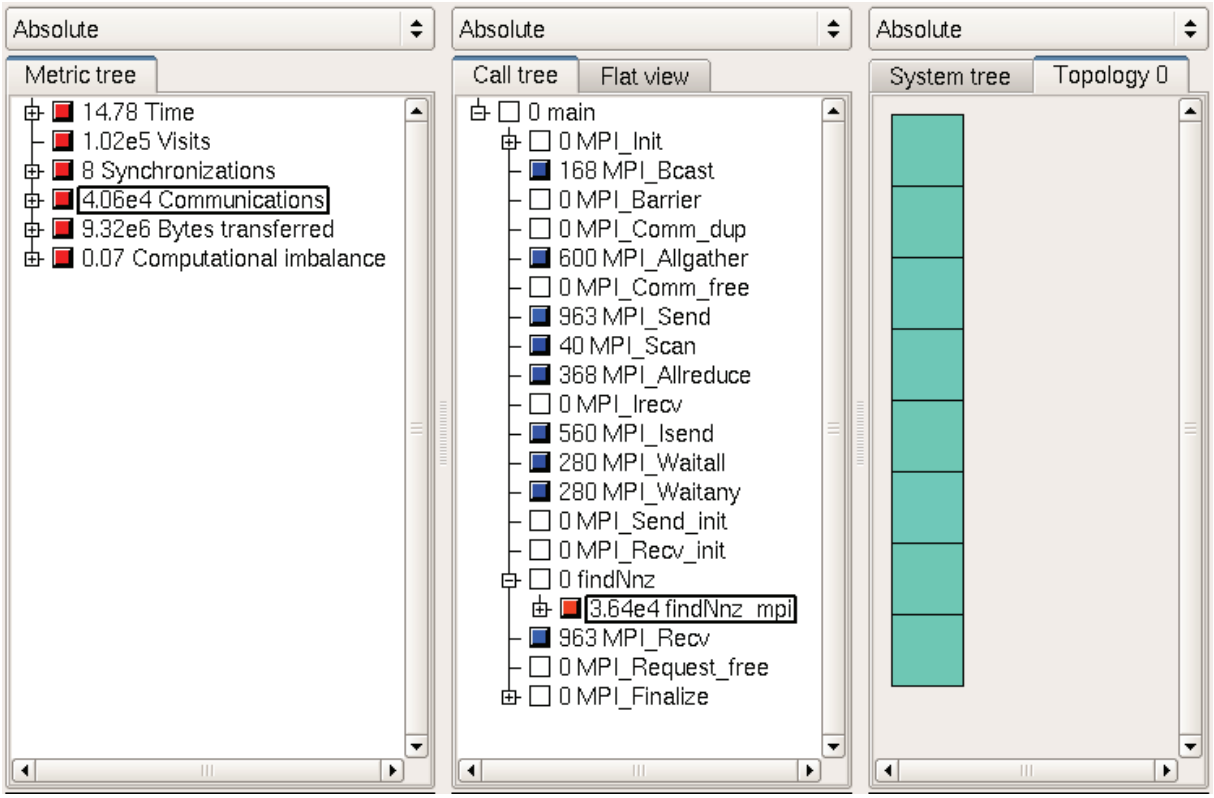
All the other extra-time is spent in communication between processes. The first heavy block of communication is performed for exchanging the column indices that will be the row indices in the next matrix. The second block of communication is performed when the processes have found the final nonzero elements positions. As explained before, some processes have row indices of nonzero elements that do not match any local row index of the result matrix. So the algorithm requires additional communications to reallocate each nonzero element position to the right process. Moreover, the `MPI_Alltoall()` call should be avoided because it is not scalable, and it should be replaced by `MPI_Isend()` and `MPI_Irecv()`.

The Scalasca software tool [122] confirms this performance analysis. Figure 4.5 shows two screenshots of the Scalasca report browser CUBE3 [121] with the summary analysis report of computation and communication. The examination of the computation performance summary reveals that almost all the computation is done in the `main()` function (red square), when the matrices are loaded and spread over the processors; whereas the function `findnnz_mpi()` does not give a significant contribute to the computational elapsed time (blue square). On the other hand, looking at the communication performance summary we see that almost all the communications are performed in `findnnz_mpi()` (red square): therefore, these communications are the bottleneck of this algorithm, but, as one can see in Algorithm 4.4 and Algorithm 4.5, they are strictly necessary.

The algorithm's communication part dominates the computation part: hence, it's hard to get good performance when it runs in parallel. However, since it takes advantage of the PETSc matrix formats, it's very portable and it can be useful when the dimension of the problem becomes too large for a sequential system. At last, the benefits given by a good and exact memory preallocation justify a little extra computation; here it's also important to emphasize that usually this kind of analysis is performed only once at the initialization step, thus the overhead due to the computation of the nonzero structure does not affect the performance of the master code and is usually completely negligible when problem sizes become larger and larger.



(a) Computation performance summary.



(b) Communication performance summary.

Figure 4.5: Scalasca screenshot of the runtime behavior of Algorithm 4.4 and Algorithm 4.5.

Chapter 5

SGP implemented as a new TAO solver

5.1 Scaled Gradient Projection method

The image formation can be modeled as a Fredholm integral equation of the first kind. After discretization, we can focus the attention on a system of linear equations

$$A\boldsymbol{m} = \boldsymbol{b}, \tag{5.1}$$

where a two-dimensional image $\mathcal{X} \in \mathbb{R}^{n\times n}$ is represented by the vector $\boldsymbol{m} = (m_i, \ldots, m_N) \in \mathbb{R}^N$, $N = n^2$, as a result of stacking the entries of $\mathcal{X}$ column by column. The matrix $A \in \mathbb{R}^{N\times N}$ represents the physical effects of the imaging system. The resulting image $\boldsymbol{b}$ is the sum of two terms: $\boldsymbol{b} = \boldsymbol{g} + \boldsymbol{\eta}$, $\boldsymbol{g} \in \mathbb{R}^N$ being the blurred image that would have been recorded in absence of noise and $\boldsymbol{\eta} \in \mathbb{R}^N$ denoting the noise affecting the image acquisition. The image restoration problem is then to obtain an approximation of $\boldsymbol{m}$, knowing A and $\boldsymbol{b}$.

Since the system (5.1) is given by the discretization of an ill-posed problem, the matrix A could be very ill-conditioned and a trivial approach that looks for the solution of (5.1) is in general not successful.

An alternative strategy is the recent Scaled Gradient Projection (SGP) method [19] suited for the constrained minimization problems like the following:

$$\begin{aligned} \min \quad & J(\boldsymbol{m}) \\ \text{sub. to} \quad & \boldsymbol{m} \geq 0, \end{aligned} \tag{5.2}$$

or

$$\begin{aligned} \min \quad & J(\boldsymbol{m}) \\ \text{sub. to} \quad & \boldsymbol{m} \geq 0 \\ & \textstyle\sum_{i=1}^{N} m_i = c, \end{aligned} \tag{5.3}$$

or

$$\begin{aligned} \min \quad & J(\boldsymbol{m}) \\ \text{sub. to} \quad & \boldsymbol{m} \in \Omega, \end{aligned} \tag{5.4}$$

where $J(\boldsymbol{m})$ is a continuously differentiable convex function measuring the difference between reconstructed image and measured data and, possibly, containing a penalty term expressing additional information on the solution, while the constraint force the non-negativity of the solution and, in case of problem (5.3), the so-called flux conservation property. In case of problem (5.4), $\Omega \subset \mathbb{R}^N$ is a closed convex set. We are interested in the case where the feasible region Ω is described by simple constraints.

Gradient projection type method are appealing approaches for these problems for two main reasons. Firstly, the special structure of the constraints makes the projection of a vector on the feasible region a non too expensive operation. Secondly, the recent advances on step length selection in gradient methods allow to improve the convergence rate of these scheme.

The main feature of SGP consist in the combination of non-expensive diagonally scaled gradient directions with steplength selection rules specially designed for these directions. Moreover, global convergence properties are ensured by exploiting a non-monotone line search strategy along the feasible direction.

5.1.1 Basic properties

Throughout the description, the 2-norm of vectors and matrices is denoted by $\|\cdot\|$ while $\|\cdot\|_D$ indicates the vector norm associated to a symmetric positive definite matrix D: $\|\boldsymbol{m}\|_D = \sqrt{\boldsymbol{m}^T D \boldsymbol{m}}$.

Let $\Omega \subset \mathbb{R}^N$ be a closed convex set and D be a symmetric positive definite $N \times N$ matrix, we define the projection operator $\mathbb{P}_{\Omega,D} : \mathbb{R}^N \to \Omega$ as

$$\mathbb{P}_{\Omega,D}(\boldsymbol{m}) \equiv \arg\min_{\boldsymbol{y}\in\Omega} \|\boldsymbol{y}-\boldsymbol{m}\|_D = \arg\min_{\boldsymbol{y}\in\Omega} \left(\phi(\boldsymbol{y}) \equiv \frac{1}{2}\boldsymbol{y}^T D \boldsymbol{y} - \boldsymbol{y}^T D \boldsymbol{m}\right). \tag{5.5}$$

We observe that, given the set Ω and the point $\boldsymbol{m}$, the operator $\mathbb{P}_{\Omega,D}(\boldsymbol{m})$ is a continuous function with respect to the elements of the matrix D. From the definition of stationary point and the strict convexity of the function ϕ introduced in (5.5), we have that $\mathbb{P}_{\Omega,D}(\boldsymbol{m})$ is defined also by

$$\left(\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \boldsymbol{m}\right)^T D \left(\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \boldsymbol{y}\right) \leq 0, \qquad \forall\ \boldsymbol{y} \in \Omega. \tag{5.6}$$

This can be proved by evaluating the gradient of the function ϕ in the point $\mathbb{P}_{\Omega,D}(\boldsymbol{m})$

$$\nabla\phi\left(\mathbb{P}_{\Omega,D}(\boldsymbol{m})\right) = D\left(\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \boldsymbol{m}\right)$$

Since from definition $\mathbb{P}_{\Omega,D}(\boldsymbol{m})$ is a constrained stationary point for the problem (5.5), we have:

$$-\nabla\phi(\mathbb{P}_{\Omega,D}(\boldsymbol{m}))^T(\boldsymbol{y} - \mathbb{P}_{\Omega,D}(\boldsymbol{m})) \leq 0, \qquad \forall\ \boldsymbol{y} \in \Omega\ ,$$

and from the symmetry of D and the previously evaluated gradient:

$$(\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \boldsymbol{m})^T D(\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \boldsymbol{y}) \leq 0, \qquad \forall\ \boldsymbol{y} \in \Omega\ .$$

Let $\mathcal{D}_L \subset \mathbb{R}^{N\times N}$ be the compact set of the symmetric positive definite $N \times N$ matrices such that $\|D\| \leq L$ and $\|D^{-1}\| \leq L$, for a given threshold $L > 1$. The next two lemmas state some properties of the projection operator defined in (5.5).

Lemma 5.1 *If $D \in \mathcal{D}_L$, then*

$$\|\mathbb{P}_{\Omega,D}(\boldsymbol{m}) - \mathbb{P}_{\Omega,D}(\boldsymbol{z})\| \leq L^2\|\boldsymbol{m}-\boldsymbol{z}\| \tag{5.7}$$

for any $\boldsymbol{m}, \boldsymbol{z} \in \mathbb{R}^N$.

Lemma 5.2 *A vector $\boldsymbol{m}_* \in \Omega$ is a stationary point of the problem (5.4) if and only if $\boldsymbol{m}_* = \mathbb{P}_{\Omega,D^{-1}}(\boldsymbol{m}_* - \alpha D\nabla J(\boldsymbol{m}_*))$ for any positive scalar α and for any symmetric positive definite matrix D.*

More details about Lemma 5.1 and Lemma 5.2 can be found in [19] and [139].

5.1.2 The SGP algorithm

The Lemma 5.2 shows the effect of the projection operator $\mathbb{P}_{\Omega,D^{-1}}$ on the points $(\boldsymbol{m}_* - \alpha D\nabla J(\boldsymbol{m}_*))$, $\alpha > 0$, when $\boldsymbol{m}_*$ is a stationary point of (5.4). In the case $\bar{\boldsymbol{m}} \in \Omega$ is a non-stationary point, $\mathbb{P}_{\Omega,D^{-1}}(\bar{\boldsymbol{m}} - \alpha D\nabla J(\bar{\boldsymbol{m}}))$ can be exploited to generate a descent direction for the function J in $\bar{\boldsymbol{m}}$. This idea serves as the basis for the method described in Algorithm 5.1. In particular, given $D_k \in \mathcal{D}_L$ and $\alpha_k \in [\alpha_{min}, \alpha_{max}]$, the SGP algorithm makes use of the following direction:

$$\boldsymbol{d}^{(k)} = \mathbb{P}_{\Omega,D_k^{-1}}(\boldsymbol{m}^{(k)} - \alpha_k D_k \nabla J(\boldsymbol{m}^{(k)})) - \boldsymbol{m}^{(k)} \ . \tag{5.8}$$

The properties of this direction are proved in Lemma 5.3, while the behavior of the sequence $\{\boldsymbol{d}^{(k)}\}$ is inspected in Lemma 5.4.

Algorithm 5.1 SGP (Scaled Gradient Projection Method Algorithm)

Choose the starting point $\boldsymbol{m}^{(0)} \in \Omega$
Set the parameters $\beta, \theta \in (0,1)$, $0 < \alpha_{min} < \alpha_{max}$
Fix a positive integer M.
for all $k = 1, 2, 3, \ldots$ **do**
  STEP 1 Choose $\alpha_k \in [\alpha_{min}, \alpha_{max}]$ and the scaling matrix $D_k \in \mathcal{D}_L$;
  STEP 2 Projection: $\boldsymbol{y}^{(k)} = \mathbb{P}_{\Omega,D_k^{-1}}(\boldsymbol{m}^{(k)} - \alpha_k D_k \nabla J(\boldsymbol{m}^{(k)}))$;
  If $\left\|\boldsymbol{y}^{(k)} - \boldsymbol{m}^{(k)}\right\| <$ **Tol** then stop; $\boldsymbol{m}^{(k)}$ is a stationary point;
  STEP 3 Descent direction: $\boldsymbol{d}^{(k)} = \boldsymbol{y}^{(k)} - \boldsymbol{m}^{(k)}$;
  STEP 4 Set $\lambda_k = 1$ and $J_{max} = \max_{0 \le j \le \min(k, M-1)} J(\boldsymbol{m}^{(k-j)})$;
  STEP 5 // Backtracking loop:
  if $J(\boldsymbol{m}^{(k)} + \lambda_k \boldsymbol{d}^{(k)}) \le J_{max} + \beta\lambda_k \nabla J(\boldsymbol{m}^{(k)})^T \boldsymbol{d}^{(k)}$ **then**
    go to Step 6;
  else
    set $\lambda_k = \theta\lambda_k$ and go to Step 5;
  end if
  STEP 6 Set $\boldsymbol{m}^{(k+1)} = \boldsymbol{m}^{(k)} + \lambda_k \boldsymbol{d}^{(k)}$.
end for

Looking at the general form of SGP, it is worth stressing that any choice of the steplength α_k in a closed interval and of the scaling matrix D_k in the compact set $\mathcal{D}_L$ is permitted. This is very important from a practical point of view since it allows one to make the updating rules of α_k and D_k problem related and oriented at optimizing the performance. For what concerns the steplength selection, we refer to Section 5.1.2, where we recall the strategy adopted by SGP.

The choice of the scaling matrix takes into account the effective form of the function J we are minimizing, as well as some additional properties of the optimization problem that has to be solved; some hints about the choice of the matrix D_k are exposed in Section 5.1.2.

Before to discuss the convergence properties of the method, some considerations about its main steps can be useful.

If the projection performed in step 2 returns a vector $\boldsymbol{y}^{(k)}$ equal to $\boldsymbol{m}^{(k)}$, then Lemma 5.2 implies that $\boldsymbol{m}^{(k)}$ is a stationary point and the algorithm stops. When $\boldsymbol{y}^{(k)} \neq \boldsymbol{m}^{(k)}$, it

is possible to prove that $\boldsymbol{d}_k$ defined in (5.8) is a descent direction for J in $\boldsymbol{m}^{(k)}$ (see Lemma 5.3) and the backtracking loop in step 5 terminates with a finite number of runs; thus the algorithm is well defined.

The nonmonotone line search strategy implemented in step 5 ensures that $J(\boldsymbol{m}^{(k+1)})$ is lower than the maximum of the objective function on the last M iterations [55]; of course, if $M = 1$ then the strategy reduces to the standard monotone Armijo rule [15].

SGP convergence

In this section we will focus on the case in which the algorithm generates an infinite sequence of iterates, denoted by $\{\boldsymbol{m}^{(k)}\}$. The main SGP convergence result is stated in Theorem 5.1, whose proof is based on some crucial properties reported in the next lemmas.

The first two lemmas are concerned with the descent condition and the boundedness of the directions $\boldsymbol{d}^{(k)}$, respectively.

Lemma 5.3 *Assume that $\boldsymbol{d}^{(k)} \neq \boldsymbol{0}$. Then, $\boldsymbol{d}^{(k)}$ is a descent direction for the function J at $\boldsymbol{m}^{(k)}$, that is, $\nabla J(\boldsymbol{m}^{(k)})^T \boldsymbol{d}^{(k)} < 0$.*

Lemma 5.4 *If the sequence $\{\boldsymbol{m}^{(k)}\}$ is bounded, then also the sequence $\{\boldsymbol{d}^{(k)}\}$ is bounded.*

In the next lemmas some properties of the accumulation points of the sequence $\{\boldsymbol{m}^{(k)}\}$ generated by SGP are recalled.

Lemma 5.5 *Assume that the subsequence $\{\boldsymbol{m}^{(k)}\}_{k\in K}$, $K \subset \mathbb{N}$, is converging to a point $\boldsymbol{m}_* \in \Omega$. Then, $\boldsymbol{m}_*$ is a stationary point of (5.4) if and only if*

$$\lim_{k\in K} \nabla J(\boldsymbol{m}^{(k)})^T \boldsymbol{d}^{(k)} = 0.$$

Lemma 5.6 *Let $\boldsymbol{m}_* \in \Omega$ be an accumulation point of the sequence $\{\boldsymbol{m}^{(k)}\}$ such that $\lim_{k\in K} \boldsymbol{m}^{(k)} = \boldsymbol{m}_*$, for some $K \subset \mathbb{N}$. If $\boldsymbol{m}_*$ is a stationary point of (5.4), then $\boldsymbol{m}_*$ is an accumulation point also for the sequence $\{\boldsymbol{m}^{(k+r)}\}_{k\in K}$ for any $r \in \mathbb{N}$. Furthermore,*

$$\lim_{k\in K} \|\boldsymbol{d}^{(k+r)}\| = 0, \qquad \forall\, r \in \mathbb{N}.$$

A convergence result for SGP is the following Theorem 5.1:

Theorem 5.1 *Assume that the level set $\Omega_0 = \{\boldsymbol{m} \in \Omega : J(\boldsymbol{m}) \leq J(\boldsymbol{m}^{(0)})\}$ is bounded. Every accumulation point of the sequence $\{\boldsymbol{m}^{(k)}\}$ generated by the SGP algorithm is a stationary point of (5.4).*

Update the scaling matrix

The choice of the scaling matrix D_k in SGP must be a non-excessively expensive task and should improve convergence rate. A diagonal scaling allows one to make the projection in step 2 of Algorithm 5.1 a non-significant computational cost; thus, we will concentrate on such kind of scaling matrices. A classical choice is to use a scaling matrix $D_k = \text{diag}\left(d_1^{(k)}, d_2^{(k)}, \ldots, d_N^{(k)}\right)$ that approximates the inverse of the Hessian matrix $\nabla^2 J(\boldsymbol{x}; \boldsymbol{y})$, for example by requiring

$$d_i^{(k)} \approx \left(\frac{\partial^2 J(\boldsymbol{x}^{(k)})}{(\partial x_i)^2} \right)^{-1}, \qquad i = 1, \ldots, N.$$

In this case an updating rule for the entries of D_k could be

$$d_i^{(k)} = \min\left\{ L, \max\left\{ \frac{1}{L}, \left(\frac{\partial^2 J(\boldsymbol{x}^{(k)})}{(\partial x_i)^2} \right)^{-1} \right\} \right\}, \qquad i = 1, \ldots, N, \tag{5.9}$$

where L is an appropriate threshold ensuring that $D_k \in \mathcal{D}_L$.

The EM method [112], also known as Richardson-Lucy method [86, 107] suggests a definition of the scaling matrix; by following this idea we may introduce the updating rule

$$d_i^{(k)} = \min\left\{ L, \max\left\{ \frac{1}{L}, x_i^{(k)} \right\} \right\}, \qquad i = 1, \ldots, N. \tag{5.10}$$

From a computational viewpoint, the updating rule (5.9) is more expensive than (5.10), due to the computation of the diagonal entries of the Hessian.

Update the steplength

Steplength selection rules in gradient methods have received an increasing interest in the last years from both the theoretical and the practical point of view. Following the original ideas of Barzilai and Borwein (BB) [7], we can regard the matrix $B(\alpha_k)$ as an approximation of the Hessian $\nabla^2 J(\boldsymbol{x}^{(k)})$ and derive two updating rules for α_k by forcing quasi-Newton properties on $B(\alpha_k)$:

$$\alpha_k^{\text{BB1}} = \underset{\alpha_k \in \mathbb{R}}{\text{argmin}} \, \| B(\alpha_k) \boldsymbol{s}^{(k-1)} - \boldsymbol{z}^{(k-1)} \| \tag{5.11}$$

and

$$\alpha_k^{\text{BB2}} = \underset{\alpha_k \in \mathbb{R}}{\text{argmin}} \, \| \boldsymbol{s}^{(k-1)} - B(\alpha_k)^{-1} \boldsymbol{z}^{(k-1)} \| \, , \tag{5.12}$$

where $\boldsymbol{s}^{(k-1)} = \left(\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)} \right)$ and $\boldsymbol{z}^{(k-1)} = \left(\nabla J(\boldsymbol{x}^{(k)}) - \nabla J(\boldsymbol{x}^{(k-1)}) \right)$.
Using $B(\alpha_k) = (\alpha_k D_k)^{-1}$ in (5.11) and (5.12), the steplengths

$$\alpha_k^{BB1} = \frac{{\boldsymbol{s}^{(k-1)}}^T D_k^{-1} D_k^{-1} \boldsymbol{s}^{(k-1)}}{{\boldsymbol{s}^{(k-1)}}^T D_k^{-1} \boldsymbol{z}^{(k-1)}} \tag{5.13}$$

and

$$\alpha_k^{BB2} = \frac{{\boldsymbol{s}^{(k-1)}}^T D_k \boldsymbol{z}^{(k-1)}}{{\boldsymbol{z}^{(k-1)}}^T D_k D_k \boldsymbol{z}^{(k-1)}} \tag{5.14}$$

Algorithm 5.2 SGP Steplength Selection

IF $k = 0$ THEN

set $\alpha_0 \in [\alpha_{min}, \alpha_{max}]$, $\tau_1 \in (0,1)$ and a nonnegative integer M_α;

ELSE

IF $\boldsymbol{s}^{(k-1)^T} D_k^{-1} \boldsymbol{z}^{(k-1)} \leq 0$ THEN

$\alpha_k^{BB1} = \alpha_{max}$;

ELSE

$\alpha_k^{BB1} = \max\left\{\alpha_{min}, \min\left\{\frac{\boldsymbol{s}^{(k-1)^T} D_k^{-1} D_k^{-1} \boldsymbol{s}^{(k-1)}}{\boldsymbol{s}^{(k-1)^T} D_k^{-1} \boldsymbol{z}^{(k-1)}}, \alpha_{max}\right\}\right\}$;

ENDIF

IF $\boldsymbol{s}^{(k-1)^T} D_k \boldsymbol{z}^{(k-1)} \leq 0$ THEN

$\alpha_k^{BB2} = \alpha_{max}$;

ELSE

$\alpha_k^{BB2} = \max\left\{\alpha_{min}, \min\left\{\frac{\boldsymbol{s}^{(k-1)^T} D_k \boldsymbol{z}^{(k-1)}}{\boldsymbol{z}^{(k-1)^T} D_k D_k \boldsymbol{z}^{(k-1)}}, \alpha_{max}\right\}\right\}$;

ENDIF

IF $\alpha_k^{BB2} / \alpha_k^{BB1} \leq \tau_k$ THEN

$\alpha_k = \min\left\{\alpha_j^{BB2},\ j = \max\{1, k - M_\alpha\}, \ldots, k\right\}$; $\quad \tau_{k+1} = \tau_k * 0.9$;

ELSE

$\alpha_k = \alpha_k^{BB1}$; $\quad \tau_{k+1} = \tau_k * 1.1$;

ENDIF

ENDIF

are obtained; that reduce to the standard BB rules in case of non-scaled gradient methods, that is when D_k is equal to the identity matrix for all k:

$$\alpha_k^{\mathrm{BB1}} = \frac{\boldsymbol{s}^{(k-1)^T} \boldsymbol{s}^{(k-1)}}{\boldsymbol{s}^{(k-1)^T} \boldsymbol{z}^{(k-1)}}, \qquad \alpha_k^{\mathrm{BB2}} = \frac{\boldsymbol{s}^{(k-1)^T} \boldsymbol{z}^{(k-1)}}{\boldsymbol{z}^{(k-1)^T} \boldsymbol{z}^{(k-1)}}. \tag{5.15}$$

Several steplength updating strategies have been devised to accelerate the slow convergence exhibited in most cases by standard gradient methods, and a lot of effort has been put into explaining the effects of these strategies [27, 28, 37, 40, 41, 146]. On the other hand, numerical experiments on randomly generated, library and real-life test problems have confirmed the remarkable convergence rate improvements involved by some BB-like steplength selections [27, 28, 40, 111, 138, 141, 146].

At this point, inspired by the steplength alternations implemented in the framework of non-scaled gradient methods, an updating rule for SGP is proposed, which adaptively alternates the values provided by (5.13) and (5.14). The details of the SGP steplength selection are given in Algorithm 5.2. This rule decides the alternation between two different selection strategies by means of the variable threshold τ_k instead of a constant parameter as done in [40] and [146]. This trick makes the choice of τ_0 less important for the SGP performance and seems able to avoid the drawbacks due to the use of the same steplength rule in too many consecutive iterations.

5.2 SGP implemented as a TAO solver

New optimization solvers can be added to TAO, that provides tools for facilitate the implementation of a solver. The advantages of implementing a new solver using TAO are several.

1. TAO includes many optimization solvers with an identical interface, so we may conveniently switch solvers to compare their effectiveness in JoInv applications.

2. TAO provides line searches, convergence tests, monitoring routines and other tools which are helpful within an optimization algorithm.

3. TAO offers vectors, matrices, index sets and linear solvers that can be used by the solver. These objects are standard mathematical constructions that have many different implementations. The objects may be distributed over multiple processors, restricted to a single processor, have a dense representation, use a sparse data structure or vary in many other ways. The solvers apply the operations through an abstract interface that leaves the details to TAO and external libraries. This abstraction allows solvers to work seamlessly with a variety of data structures while allowing application developers to select data structures tailored for their purposes.

4. TAO supports an interface to PETSc and allows the integration of other libraries as well.

SGP implemented as TAO solver is written in `C++` and include several routines with a particular calling sequence, as explained in the TAO manual [8].

In each of these routines except the initialization routine, there are two arguments. The first argument is always the TAO structure. This structure may be useful to obtain the vectors used to store the variables and the function gradient, evaluate function and gradient, solve a set of linear equations, perform a line search, and apply a convergence test.
The second argument is specific to this solver. This pointer will be set in the initialization routine and cast to an appropriate type in the other routines. The following structure Listing 5.1 is used by SGP algorithm:

```
typedef struct {

  TaoVec *g;  /// Gradient vector
  TaoVec *d;  /// Search direction or diag. of the scaling matrix
  TaoVec *s;  /// (Current - prev) solution estimate
  TaoVec *y;  /// (Current - prev) gradient estimate
  TaoVec *w;  /// Work vector
  TaoVec *c;  /// Linear constrain

  double lambda;  /// Nonmonotone line search parameter

  double xBoundLower;       /// Lower bound on solution vector
  double xBoundUpper;       /// Upper bound on solution vector
  TaoVec *xVecBoundLower;   /// Lower bound on solution vector
  TaoVec *xVecBoundUpper;   /// Upper bound on solution vector

  void *steplengthctx;      /// Steplength context
  void *scalingmatrixctx;   /// Scaling matrix context

} TAO_SGP;
```

Listing 5.1: Context for Scaled Gradient Projection method.
(Actual source code `sgp.h`)

Solver Routine

All TAO solvers have a routine that accepts a TAO structure and computes a solution. TAO will call this routine when the application program uses the routine `TaoSolve()` and pass to the solver information about the objective function and constraints, pointers to the variable vector and gradient vector, and support for line searches, linear solvers, and convergence monitoring. The following code Listing 5.2 solves a minimization problem using the SGP method explained in Section 5.1.

```
#undef __FUNCT__
#define __FUNCT__ "TaoSolve_SGP"

static int TaoSolve_SGP(TAO_SOLVER tao, void *solver)
{

  TAO_SGP *sgp  = (TAO_SGP *)solver;

  TAO_STEPLENGTH_BB* sl = (TAO_STEPLENGTH_BB*) sgp->steplengthctx;

  TaoVec *x;
  TaoVec *x_old;
  TaoVec *g     = sgp->g;
  TaoVec *d     = sgp->d;
  TaoVec *s     = sgp->s;
  TaoVec *y     = sgp->y;
  TaoVec *w     = sgp->w;


  TaoTerminateReason reason;

  double alpha = sl->initAlpha;

  double f,
         f_full,
         gnorm,
         cnorm,       // the infeasibility of the current solution
                      //        with regard to the box constraints.
         step = 1.0;

  int info      = 0;
  TaoInt status = 0;
  TaoInt iter   = 0;

  double gdx;

  TaoFunctionBegin;

  // Get vectors we will need
  info = TaoGetSolution(tao, &x); CHKERRQ(info);

  // Check convergence criteria
  info = TaoComputeFunctionGradient(tao, x, &f, g); CHKERRQ(info);

  info = g->Norm2(&gnorm); CHKERRQ(info);
  if (TaoInfOrNaN(f) || TaoInfOrNaN(gnorm)) {
     SETERRQ(1, "User provided compute function generated Inf or NaN");
  }

  // Check box feasibility of the initial point
  info = x->Clone(&x_old);    CHKERRQ(info);
  info = sgpProject(sgp, x);
```

```
53    info = x_old->Axpy(-1.0, x); CHKERRQ(info);
54    info = x_old->Norm2(&cnorm); CHKERRQ(info);
55
56    info = TaoMonitor(tao, iter, f, gnorm, cnorm, step, &reason);
57    if (reason != TAO_CONTINUE_ITERATING) {
58      TaoFunctionReturn(0);
59    }
60
61    // Scaling matrix
62    info = TaoApply_ScalingMatrix(tao, x, d, (void*) sgp->scalingmatrixctx);
63
64    while(1){
65
66      // in: d <- scaling matrix
67      info = d->PointwiseMultiply(d, g); CHKERRQ(info);
68
69      // d = x - alpha * d
70      info = d->Aypx( -alpha, x ); CHKERRQ(info);
71      info = sgpProject(sgp, d); CHKERRQ(info);
72      // out: d <- search dir
73
74      // in: d <- search dir
75      // d = d - x
76      info = d->Axpy( -1.0, x ); CHKERRQ(info);
77
78      // check if d == 0 (i.e. projection(x)==x) then stop
79      d->Norm2(&step); CHKERRQ(info);
80      if(step<TAO_ZER_SAFEGUARD){
81        step = 0.0;    // the next call to TaoMonitor will stop the algorithm
82      }
83
84      info = TaoMonitor(tao, iter, f, gnorm, 0.0, step, &reason);
85      if (reason != TAO_CONTINUE_ITERATING) {
86        TaoFunctionReturn(0);
87      }
88
89      info = y->CopyFrom(g); CHKERRQ(info);
90
91      // in: d <- search dir ; x = x_k ; g = g_k ; d = d_k
92
93      // check if the input parameter gdx is negative;
94      info = g->Dot(d, &gdx);
95      if(gdx>=0){
96        PetscPrintf(PETSC_COMM_WORLD, "WARNING: d is not a descent direction,
97                                       using the gradient \n");
98        info = d->CopyFrom(g); CHKERRQ(info);
99      }
100
101     step = 1.0;
102     info = TaoLineSearchApply(tao, x, g, d, w, &f, &f_full, &step, &status);
103           CHKERRQ(info);
104     // output: x = x_{k+1} ; g = g_{k+1} ; d = d_k
105
106     if(status){
107       PetscPrintf(PETSC_COMM_WORLD, "WARNING: Linesearch failed \n");
108     }
109
110     info = g->Norm2(&gnorm); CHKERRQ(info);
111     if (TaoInfOrNaN(f) || TaoInfOrNaN(gnorm)) {
112       SETERRQ(1, "User provided compute function generated Inf or NaN");
113     }
114
115     info = TaoMonitor(tao, ++iter, f, gnorm, 0.0, step, &reason);
116     if (reason != TAO_CONTINUE_ITERATING) {
117       TaoFunctionReturn(0);
118     }
119
120     // s_k = step * d_k = x_{k+1} - x_k
121     // in: d <- search dir
122     info = s->CopyFrom(d); CHKERRQ(info);
123     info = s->Scale( step ); CHKERRQ(info);
```

```
124
125     // y_k = g_{k+1} - g_k
126     // y   = g - y
127     info = y->Aypx( -1.0, g ); CHKERRQ(info);
128
129     // Scaling matrix
130     // in: d <- search dir
131     info = TaoApply_ScalingMatrix(tao, x, d, (void*) sgp->scalingmatrixctx);
132     // out: d <- scaling matrix
133
134     // BB step
135     // d <- scaling matrix
136     info = TaoApply_StepLength(tao, s, y, d, &alpha,
137                                (void*) sgp->steplengthctx );
138
139   }
140
141   sl->alpha = alpha;
142
143   info = TaoVecDestroy(x_old); CHKERRQ(info);
144
145   TaoFunctionReturn(0);
146 }
```

Listing 5.2: SGP solver routine.
(Actual source code `sgp.c`)

In lines 9, 62, 131 and 136 we use a steplength and scaling matrix explained in Section 5.1.2 and Section 5.1.2. More details on steplength and scaling matrix implementations are in Section 5.3.

Creation Routine

The TAO solver is initialized for a particular algorithm in a separate routine. The routine that creates the SGP algorithm shown above is implemented as follows in Listing 5.3.

```
1  EXTERN_C_BEGIN
2
3  #undef __FUNCT__
4  #define __FUNCT__ "TaoCreate_SGP"
5
6  int TaoCreate_SGP(TAO_SOLVER tao)
7  {
8    TAO_SGP *sgp;
9    int info;
10
11   TaoFunctionBegin;
12
13   info = TaoNew(TAO_SGP, &sgp); CHKERRQ(info);
14
15   // routines
16   info=TaoSetTaoSolveRoutine(tao, TaoSolve_SGP, (void *)sgp);
17        CHKERRQ(info);
18   info=TaoSetTaoSetUpDownRoutines(tao, TaoSetUp_SGP, TaoDestroy_SGP);
19        CHKERRQ(info);
20   info=TaoSetTaoOptionsRoutine(tao, TaoSetOptions_SGP);
21        CHKERRQ(info);
22   info=TaoSetTaoViewRoutine(tao, TaoView_SGP);
23        CHKERRQ(info);
24
25   // options
26   info = TaoSetMaximumIterates(tao, 100);              CHKERRQ(info);
27   info = TaoSetMaximumFunctionEvaluations(tao, 4000); CHKERRQ(info);
28   info = TaoSetTolerances(tao, 1e-4, 1e-4, 0, 0);      CHKERRQ(info);
29
30   // line search
31   info = TaoCreateProjectedArmijoLineSearch(tao); CHKERRQ(info);
```

```
32
33   // steplength and scaling matrix
34   info = TaoNew(TAO_STEPLENGTH_BB, &(sgp->steplengthctx));
35          CHKERRQ(info);
36   info = PetscLogObjectMemory(tao, sizeof(TAO_STEPLENGTH_BB));
37          CHKERRQ(info);
38   info = TaoCreateStepLengthBB(    tao, (void*) sgp->steplengthctx);
39
40   info = TaoNew(TAO_SCALINGMATRIX_BZZ, &(sgp->scalingmatrixctx));
41          CHKERRQ(info);
42   info = PetscLogObjectMemory(tao, sizeof(TAO_SCALINGMATRIX_BZZ));
43          CHKERRQ(info);
44   info = TaoCreateScalingMatrixBZZ(tao, (void*) sgp->scalingmatrixctx);
45
46   // default values
47   sgp->xBoundLower = 1e-10;
48   sgp->xBoundUpper = 1e+10;
49   sgp->linconstr   = PETSC_FALSE;
50
51   TaoFunctionReturn(0);
52 }
53
54 EXTERN_C_END
```

Listing 5.3: SGP creation routine.
(Actual source code `sgp.c`)

This routine sets the pointers to the actual SGP functions, sets default values and convergence tolerances, creates a line search supported by TAO and creates a steplength and a scaling matrix. More details on steplength and scaling matrix can be found in Section 5.3.

SetUp Routine

Since this routine has been set by the initialization routine, TAO will call it during the `TaoSetApplication()`. SGP setup routine has the following form (Listing 5.4):

```
1  #undef __FUNCT__
2  #define __FUNCT__ "TaoSetUp_SGP"
3
4  static int TaoSetUp_SGP(TAO_SOLVER tao, void *solver)
5  {
6    TAO_SGP *sgp = (TAO_SGP *)solver;
7    TaoVec *x;
8    int info;
9
10   TaoFunctionBegin;
11
12   info = TaoGetSolution(tao, &x); CHKERRQ(info);
13
14   info = x->Clone(&sgp->g);      CHKERRQ(info);
15   info = x->Clone(&sgp->d);      CHKERRQ(info);
16   info = x->Clone(&sgp->s);      CHKERRQ(info);
17   info = x->Clone(&sgp->y);      CHKERRQ(info);
18   info = x->Clone(&sgp->w);      CHKERRQ(info);
19   if(sgp->linconstr == PETSC_TRUE){
20     info = x->Clone(&sgp->c);    CHKERRQ(info);
21   }
22
23   info = x->Clone(&sgp->xVecBoundLower); CHKERRQ(info);
24   info = x->Clone(&sgp->xVecBoundUpper); CHKERRQ(info);
25
26   sgp->xVecBoundLower->SetToConstant(sgp->xBoundLower); CHKERRQ(info);
27   sgp->xVecBoundUpper->SetToConstant(sgp->xBoundUpper); CHKERRQ(info);
28
29   info = TaoSetVariableBounds(tao, sgp->xVecBoundLower, sgp->xVecBoundUpper);
30          CHKERRQ(info);
```

```
31
32    info = TaoSetLagrangianGradientVector(tao, sgp->g); CHKERRQ(info);
33    info = TaoSetStepDirectionVector(tao, sgp->d); CHKERRQ(info);
34
35    info = TaoCheckFG(tao); CHKERRQ(info);
36
37    TaoFunctionReturn(0);
38  }
```

Listing 5.4: SGP setup routine.
(Actual source code `sgp.c`)

This routine is used to allocate the work vectors, the gradient vector and the step direction vector. In order to use `tao_sgp` as any other TAO solver, the calls to the functions in lines 32, 33 are mandatory, even if they are not in the TAO manual example [8].

Destroy Routine

For the SGP method, the following routine (Listing 5.5) destroys the data structures created by earlier routines.

```
1  #undef __FUNCT__
2  #define __FUNCT__ "TaoDestroy_SGP"
3
4  static int TaoDestroy_SGP(TAO_SOLVER tao, void *solver)
5  {
6    TAO_SGP *sgp = (TAO_SGP *)solver;
7    int info;
8
9    TaoFunctionBegin;
10
11   info = TaoVecDestroy(sgp->g);               CHKERRQ(info);
12   info = TaoVecDestroy(sgp->d);               CHKERRQ(info);
13   info = TaoVecDestroy(sgp->s);               CHKERRQ(info);
14   info = TaoVecDestroy(sgp->y);               CHKERRQ(info);
15   info = TaoVecDestroy(sgp->w);               CHKERRQ(info);
16   if(sgp->linconstr == PETSC_TRUE){
17     info = TaoVecDestroy(sgp->c);             CHKERRQ(info);
18   }
19   info = TaoVecDestroy(sgp->xVecBoundLower); CHKERRQ(info);
20   info = TaoVecDestroy(sgp->xVecBoundUpper); CHKERRQ(info);
21
22   // steplength and scaling matrix
23   info = TaoDestroy_StepLength(   tao, (void*) sgp->steplengthctx);
24   info = TaoDestroy_ScalingMatrix(tao, (void*) sgp->scalingmatrixctx);
25
26   TaoFunctionReturn(0);
27 }
```

Listing 5.5: SGP destroy routine.
(Actual source code `sgp.c`)

Set Options Routine

The routine that sets solver options, such as the bounds or the parameters, is not mandatory, but it is very useful; SGP sets option functions as follows (Listing 5.6).

```
1  #undef __FUNCT__
2  #define __FUNCT__ "TaoSetOptions_SGP"
3
4  static int TaoSetOptions_SGP(TAO_SOLVER tao, void *solver)
5  {
```

```
6
7    TAO_SGP *sgp = (TAO_SGP *)solver;
8    int info;
9
10   TaoFunctionBegin;
11   info = TaoOptionsHead("SGP method"); CHKERRQ(info);
12
13   info = TaoOptionDouble("-tao_sgp_lambda", "nonmonotone line search param.",
14                          "", sgp->lambda, &sgp->lambda, 0);
15           CHKERRQ(info);
16   info = TaoOptionDouble("-tao_sgp_xBoundLower", "minimum value for scaling",
17                          "", sgp->xBoundLower, &sgp->xBoundLower, 0);
18           CHKERRQ(info);
19   info = TaoOptionDouble("-tao_sgp_xBoundUpper", "maximum value for scaling",
20                          "", sgp->xBoundUpper, &sgp->xBoundUpper, 0);
21           CHKERRQ(info);
22
23   // steplength e scaling matrix
24   info = TaoSetOptions_StepLength(   tao, (void*) sgp->steplengthctx);
25   info = TaoSetOptions_ScalingMatrix(tao, (void*) sgp->scalingmatrixctx);
26
27   // linesearch
28   info = TaoLineSearchSetFromOptions(tao); CHKERRQ(info);
29
30   TaoFunctionReturn(0);
31 }
```

Listing 5.6: SGP set options routine.
(Actual source code `sgp.c`)

View Routine

Another useful routine is the view routine (Listing 5.7) , that views informations about the SGP solver steps.

```
1  #undef __FUNCT__
2  #define __FUNCT__ "TaoView_SGP"
3
4
5  static int TaoView_SGP(TAO_SOLVER tao, void *solver)
6  {
7    int      info;
8    TAO_SGP *sgp = (TAO_SGP *)solver;
9
10   info = TaoView_StepLength(   tao, (void*) sgp->steplengthctx);
11   info = TaoView_ScalingMatrix(tao, (void*) sgp->scalingmatrixctx);
12
13   info = TaoLineSearchView(tao); CHKERRQ(info);
14
15   TaoFunctionReturn(0);
16 }
```

Listing 5.7: SGP view routine.
(Actual source code `sgp.c`)

5.3 Steplength and scaling matrix

As seen in Section 5.1, SGP algorithm needs a steplength and a scaling matrix. Since other solvers may also need them, like many solvers need a line search, we tried to implement the steplength and the scaling matrix following the structure of the TAO line searches.

Studying the TAO line search implementations written in the TAO source files /src/interface/line.c.html,

/src/linesearch/impls/morethuente/morethuente.c.html and /src/linesearch/impls/morethuente/morethuente.h.html, we find out that all the line searches share the same interface written in `line.c`. The routines declared there are wrappers of the implemented line search routines, for example, the Morè-Thuente source code is in `morethuente.c` and its structure is declared in `morethuente.h`.

The TAO source code /src/tao_impl.h contains the declaration of the `struct _p_TAO_SOLVER`, that is the main TAO solver context; Listing 5.8 shows a part of this declaration.

```
1  struct _p_TAO_SOLVER {
2
3    [...]
4
5    /*  --- User-provided Info ---*/
6
7    /* --- Routines and data that are unique to each particular solver --- */
8    void *data;     /* algorithm implementation-specific data */
9
10   int (*setup)(TAO_SOLVER,void*);    /* set up the nonlinear solver */
11   int (*solve)(TAO_SOLVER,void*);    /* a nonlinear optimization solver */
12   int (*setdown)(TAO_SOLVER,void*); /* destroys solver */
13   int (*setfromoptions)(TAO_SOLVER,void*); /* sets options from database */
14   int (*view)(TAO_SOLVER,void*);     /* views solver info */
15
16   TaoTruth setupcalled;        /* true if setup has been called */
17   TaoTruth set_method_called; /*flag indicating set_method has been called*/
18
19   [...]
20
21   /* --- Line Search Context ---*/
22
23   /* Line Search termination code and function pointers */
24   void          *linectx;
25
26   TaoInt lsflag;  /* Line search termination code (set line=1 on success) */
27
28   int     (*LineSearchSetUp)(TAO_SOLVER,void*);
29   int     (*LineSearchSetFromOptions)(TAO_SOLVER,void*);
30   int     (*LineSearchApply)(TAO_SOLVER,TaoVec*,TaoVec*,TaoVec*,TaoVec*,
31                              double*,double*,double*,TaoInt*,void*);
32   int     (*LineSearchView)(TAO_SOLVER,void*);
33   int     (*LineSearchDestroy)(TAO_SOLVER,void*);
34
35   [...]
36
37 };
```

Listing 5.8: A piece of the `struct _p_TAO_SOLVER`.
(Actual source code `/src/tao_impl.h`)

As you can see, in line 24 a line search context is declared and in lines 28–33 there are the functions related to its context.

In order to implement the steplength and the scaling matrix following the schema just explained, one should write a steplength interface, a matrix scaling interface and than write at least one implementation for each of the new solver's objects.

We are not able to strictly follow that schema because in /src/tao_impl.h there is not a generic `void *context` that can be used by a user to add new objects, however Figure 5.1 and Figure 5.2 show how the steplength and the scaling matrix implementation should be.

We temporary implemented the steplength and the scaling matrix context without the interface middle step; however we wrote all the required functions:

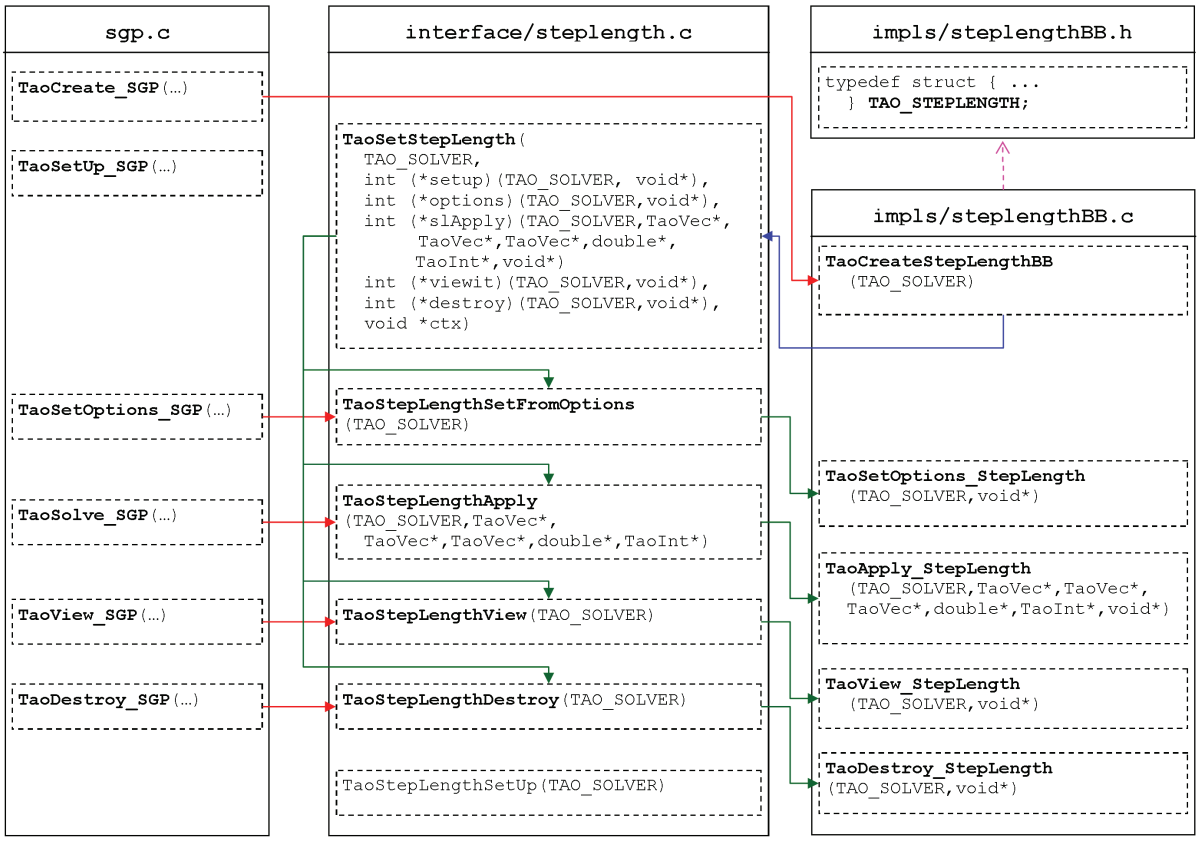


Figure 5.1: SGP solver calls the interface's routines implemented in `interface/steplength.c`, that are wrappers of the BB steplength routines, implemented in `impls/steplengthBB.c`. The `TAO_STEPLENGTH` data structure is declared in `impls/steplengthBB.h`.

```
int (*XXXSetUp)          (TAO_SOLVER, void*);
int (*XXXSetFromOptions)(TAO_SOLVER, void*);
int (*XXXApply)          (TAO_SOLVER, TaoVec*,
                          TaoVec*, TaoVec*, double*, void*);
int (*XXXView)           (TAO_SOLVER, void*);
int (*XXXDestroy)        (TAO_SOLVER, void*);
```

because in this way it will be quite simple and fast adding the missing interface step as soon as there will be the necessary data declaration in /src/tao_impl.h.

Figure 5.2: SGP solver calls the interface's routines implemented in `interface/scalingmatrix.c`, that are wrappers of the BZZ scaling matrix routines, implemented in `impls/scalingmatrixBZZ.c`. The `TAO_SCALINGMATRIX` data structure is declared in `impls/scalingmatrixBZZ.h`.

5.4 Troubleshooting

Even if the TAO manual section that explain how to add a solver is quite comprehensive, some implementation details are missing.

5.4.1 How-to: Register a new solver with TAO

You need to register the new solver with TAO; you can do this with

`TaoRegisterDynamic()`

If you have a new solver compiled into a dynamic library, then you need to give the path argument. If you are linking the new solver directly with your application, then you need to give the fourth argument (name of creation routine).
For the `tao_sgp` solver the following line

```
TaoRegisterDynamic("tao_sgp", 0, "TaoCreate_SGP", TaoCreate_SGP);
```

is added before

```
TaoCreate(PETSC_COMM_WORLD, "tao_sgp", &tao);
```

in the file that use the solver.
Then you will also have to declare this creation function as `extern "C"`:

```
extern "C" int TaoCreate_SGP(TAO_SOLVER);
```

See src/interface/tao_reg.c.html for an example.

5.4.2 How-to: Access to the underlying PETSc vector

Since when you write a new TAO solver you use `TaoVec` type instead of the `Vec` PETSc type, sometimes you need to access to the underlying PETSc vector. You should use the following call sequence:

```
TaoVec *tv;
   ...
TaoVecPetsc *tvp  = dynamic_cast<TaoVecPetsc*>(tv);
if (!tvp) {/* Send Error Message */}
Vec px = tvp->GetVec();
```

See src/unconstrained/impls/neldermead/neldermead.c for a complete example.
In `taopetsc.h` is declared the function `TaoVecGetPetscVec()`, but it is not implemented anywhere.

5.4.3 How-to: Set the gradient vector

In the solver's setup routine you should call

```
TaoSetLagrangianGradientVector(TAO_SOLVER solver, TaoVec* gg);
```

that sets a pointer to the address of a `TaoVec` that contains the gradient, otherwise you'll get a runtime null pointer error because the vector specified here is actually use by TAO as gradient vector and it's returned whenever `TaoGetGradient()` is called.

5.4.4 How-to: TAO_APPLICATION dynamic cast sequence

When you use TAO within PETSc, you usually set your TAO_APPLICATION as follows:

```
TAO_SOLVER tao;
TAO_APPLICATION taoapp;

[...]

info = TaoSetApplication(tao, taoapp); CHKERRQ(info);
```

Then, if you need to access your TAO_APPLICATION and you have only the TAO_SOLVER, you must write the following cast sequence:

```
TaoApplication      *tapp;
TaoPetscApplication *tpapp;
TAO_APPLICATION     app;

[...]

info = TaoGetApplication(tao, &tapp); CHKERRQ(info);
tpapp = dynamic_cast<TaoPetscApplication*>(tapp);
if (!tpapp) {
  SETERRQ(1,
      "Could not cast TaoApplication* to TaoPetscApplication*");
}
app = tpapp->papp;
```

See src/interface/fdtest.c for a complete example.

5.4.5 How-to: Add/Query an object to/from the Tao Application

You can add an object to your Tao Application calling:

```
TaoAppAddObject(TAO_APPLICATION taoapp,
                char *key, void *ctx, TaoInt *id)
```

and query the TAO Application for an object calling:

```
TaoAppQueryForObject(TAO_APPLICATION taoapp,
                     char *key, void **ctx)
```

Inside the query call, an object is identified by comparing the key values. Since this comparison is performed by

```
PetscStrncmp(const char a[], const char b[],
             size_t n, PetscTruth *t)
```

that only compare up to size_t n char and n is set to 10, the key strings must be shorter than 10 character, otherwise you will not able to identify the needed object.

5.5 SGP performance analysis

We tested SGP on an image deblurring problem; the original image is shown in Figure 5.3 and has dimensions 311×149; the other four images are generated duplicating the original image 2 (Figure 5.4), 8, 16, and 24 times in both directions; we refers to these images as $A_{1\times1}$, $A_{2\times2}$, $A_{8\times8}$, $A_{16\times16}$, $A_{24\times24}$ respectively.

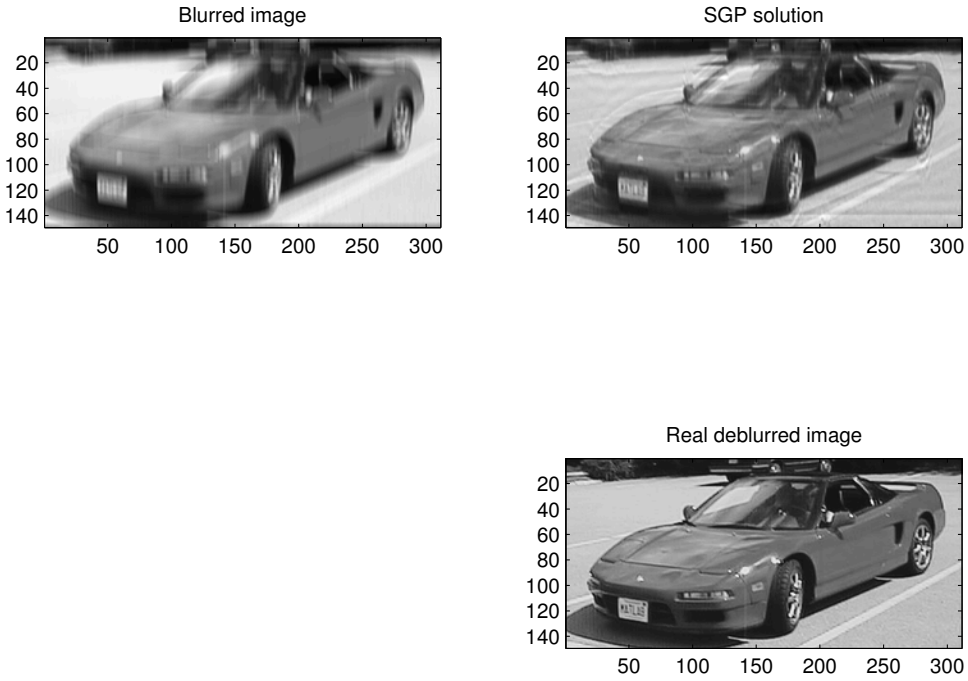


Figure 5.3: SGP test images. In the top left corner there is the blurred image; on the top right corner there is the image deblurred by SGP; on the bottom there is the real image that is been blurred.

In order to evaluate the SGP performance, we measure the execution times, the relative and scaled speedup (Section B.1), the efficiency (Section B.2), and the Kuck's function (Section B.3).

The speedup accounts for how much a parallel algorithm is faster than a sequential one; best performances (*i.e.* linear speedup) are obtained when $S(p) = p$, where p is the number of processors used for running the program.
The efficiency estimates how well the given processors are exploited in solving the problem, compared with the overhead due to communications and synchronizations; best performances are obtained when $E(p)$ is close to 1, superoptimal behavior when $E(p) > 1$.
Finally, the Kuck's function refers to how advantageous the parallel implementation remains as the number of processors increases. The maximizer of the Kuck's function is interpreted as the largest number of processors suitable for the parallel implementation to solve the given particular problem.

The main conclusion that can be drawn from Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9 is that SGP shows a very good scaling property: indeed the more the data grows, the more processors are efficiently used.

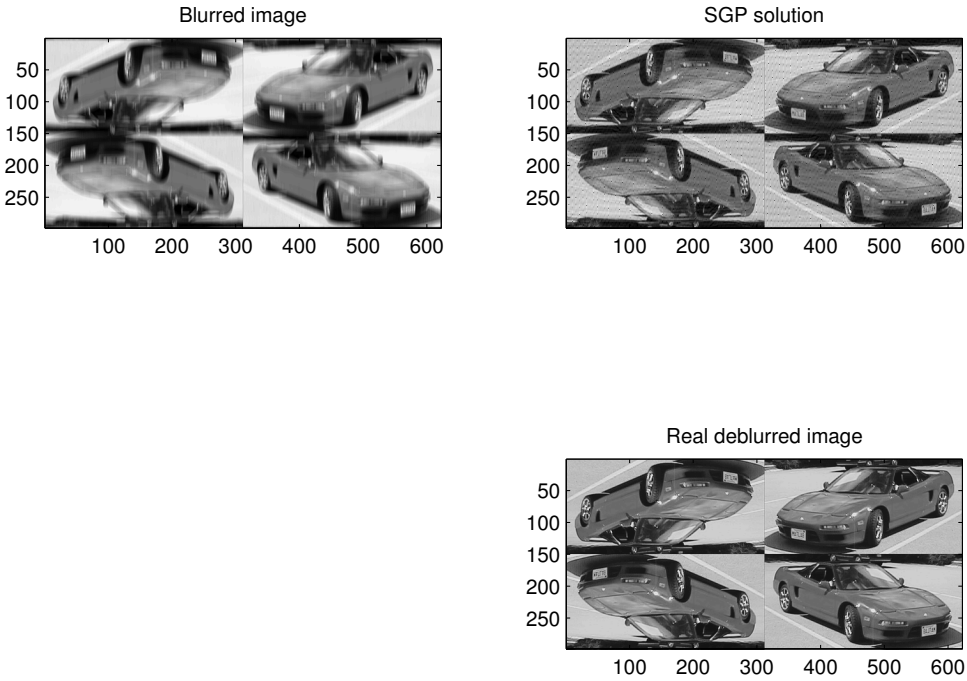


Figure 5.4: SGP duplicated test image. In the top left corner there is the blurred image; on the top right corner there is the image deblurred by using SGP; on the bottom there is the real image that is been blurred.

For example, if we analyze the execution times (Figure 5.5(a)), the relative speedup (Figure 5.6(a)), and the efficiency plots (Figure 5.7(a)) relative to $A_{1\times1}$ we can see a good behavior till 16 processors, that is in accord to the Kuck's function of Figure 5.8(a). If we now look at the same plots but relative to $A_{2\times2}$ dataset, (Figure 5.5(b) Figure 5.6(b) Figure 5.7(b)), that is four times bigger than $A_{1\times1}$ dataset, we can see a good behavior till 64 processors, that is in accord to the Kuck's function of Figure 5.8(b).

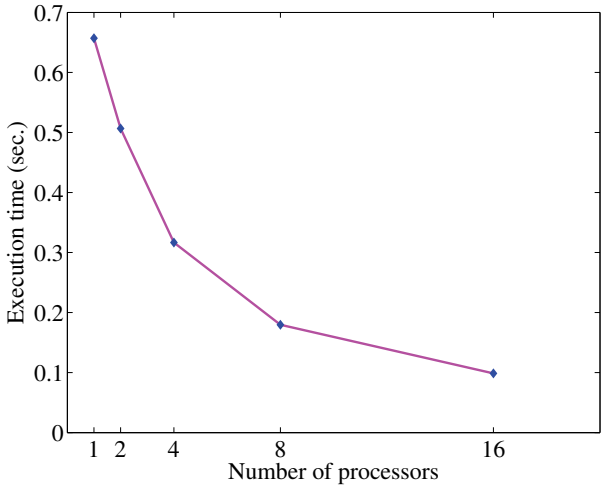
The speedup, efficiency, and Kuck's function charts relatives to $A_{16\times16}$ $A_{24\times24}$ are missing because, when run on one processor, they exceed the core's memory quota.

Scaled speedup recognizes that an increase in the number of processors brings with it an increase in the problem size. As seen in Figure 5.9, the scaled speedups for the SGP algorithm is observed to be nearly linear.

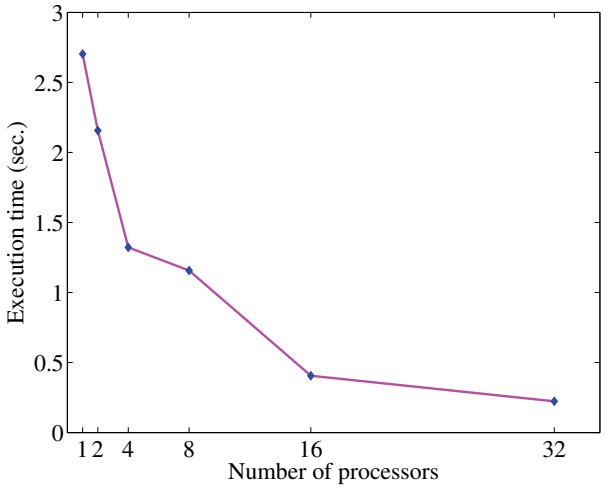
Moreover, in order to evaluate the correctness of the SGP implementation on more processors, we measure for each test the relative reconstruction error, defined as

$$||x_k - x_{\text{sol}}|| \,/\, ||x_{\text{sol}}|| \ ,$$

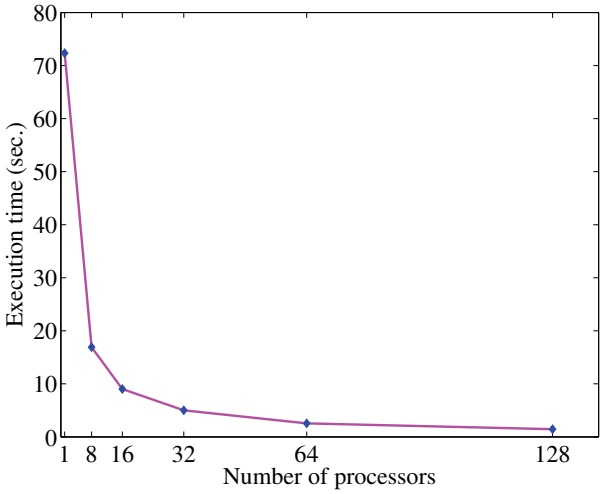
where x_{sol} is the image to be reconstructed, x_k is the reconstruction, and $||\cdot||$ is either $||\cdot||_2$ or $||\cdot||_\infty$. The measured reconstruction errors show that this SGP implementation has the same behavior both on single and on p processors.



(a) SGP execution times; $A_{1\times1}$

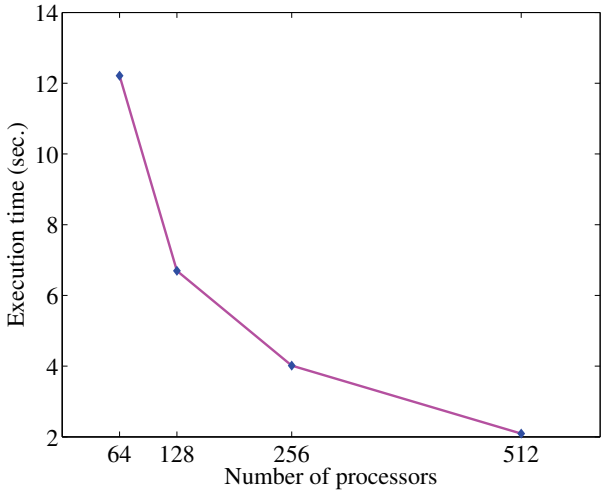


(b) SGP execution times; $A_{2\times2}$

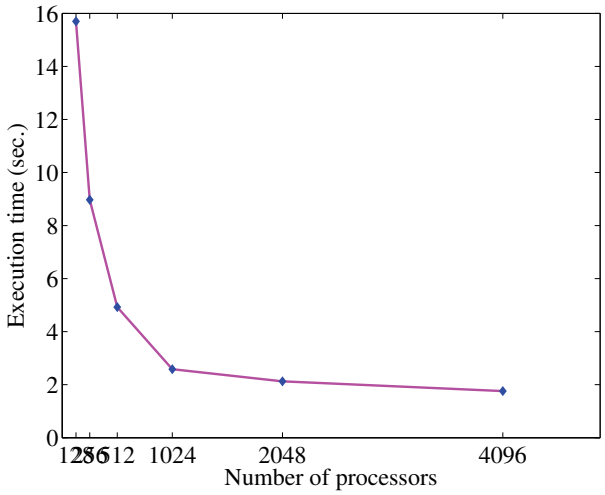


(c) SGP execution times; $A_{8\times8}$

Figure 5.5: SGP execution times on various dataset.

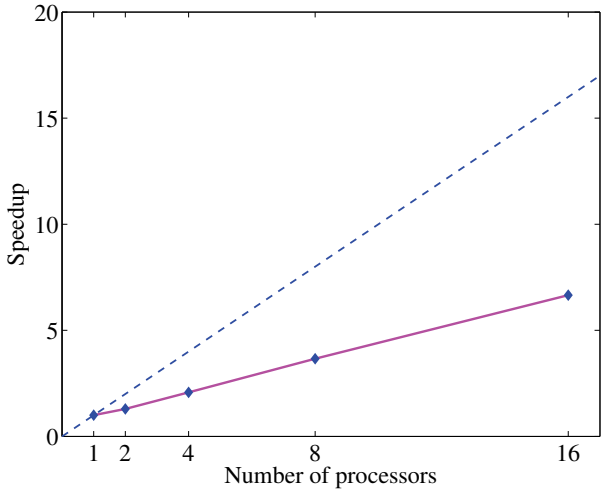


(d) SGP execution times; $A_{16\times16}$

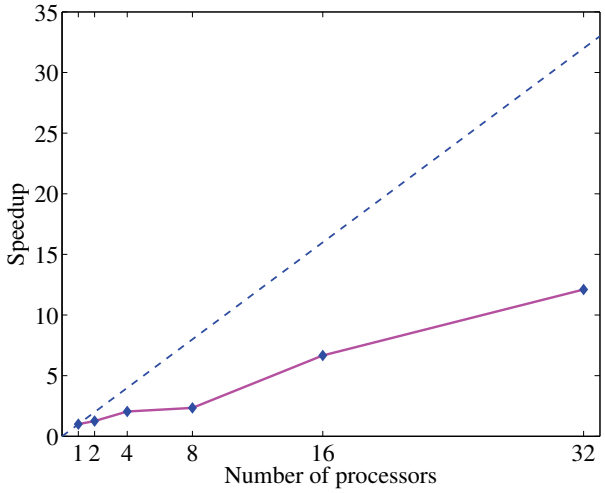


(e) SGP execution times; $A_{24\times24}$

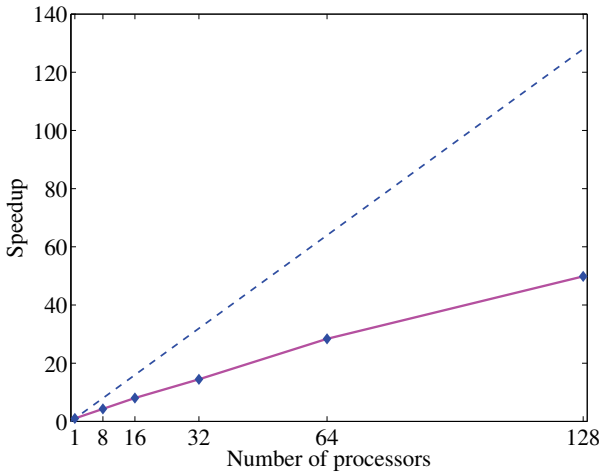
Figure 5.5: SGP execution times on various dataset (cont.).



(a) SGP relative speedup; $A_{1\times1}$

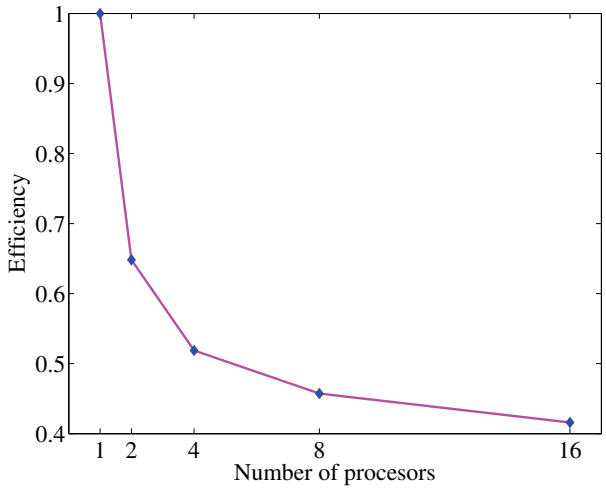


(b) SGP relative speedup; $A_{2\times2}$

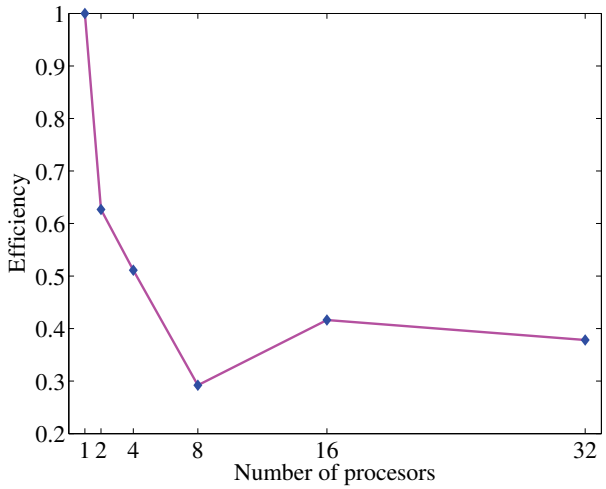


(c) SGP relative speedup; $A_{8\times8}$

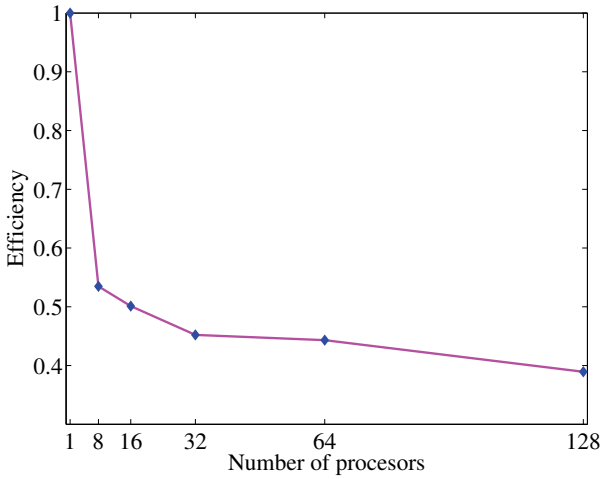
Figure 5.6: SGP relative speedup measured on various dataset.



(a) SGP efficiency; $A_{1\times1}$

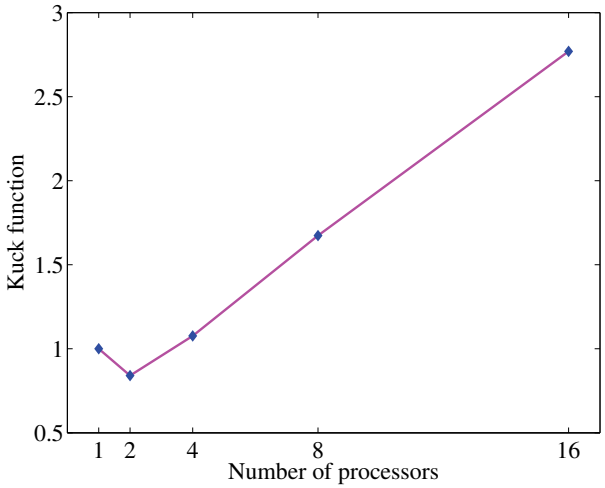


(b) SGP efficiency; $A_{2\times2}$

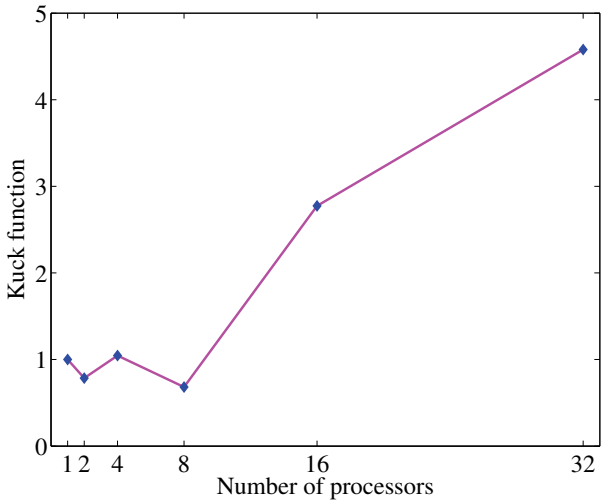


(c) SGP efficiency; $A_{8\times8}$

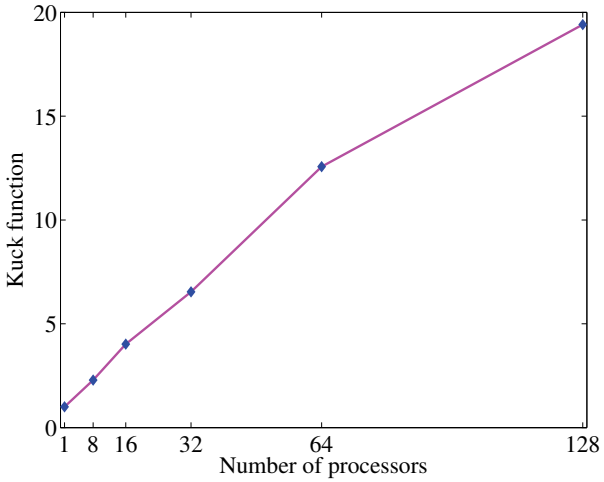
Figure 5.7: SGP efficiency measured on various dataset.



(a) SGP Kuck's function; $A_{1\times1}$



(b) SGP Kuck's function; $A_{2\times2}$



(c) SGP Kuck's function; $A_{8\times8}$

Figure 5.8: SGP Kuck's function measured on various dataset.

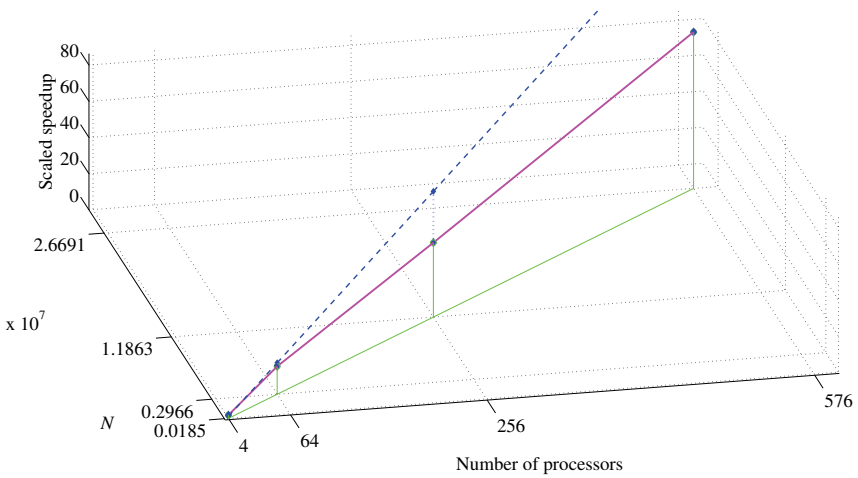


Figure 5.9: SGP scaled speedup, where $\mathcal{N}$ is the total number of elements in the test matrix A.

5.5.1 DUSD model on SGP

To asses the correctness of our experimental performance analysis, we compute here the theoretical optimal number $\tilde{p}$ of processors for the SGP algorithm on the input matrix $A_{1\times 1}$ using the DUSD model (Section B.7).

The IBM-SP6 hardware parameters that describe the computation rate r^s_∞ for floating-point operations, the asymptotic communication bandwidth r^c_∞, and the message latency t^c_0 are reported in Table 5.1.

What	Symbol	Value
Comput. rate, peak performance	r^s_∞	101 Tflop/s
Asymptotic communic. bandwidth	r^c_∞	16 Gb/s or 250 Mword/s (64-bit precision)
Message latency	t^c_0	2.103 μs

Table 5.1: IBM-SP6 hardware parameters.

We also estimate the following SGP code parameters:

$$s^s(N;p) = \frac{20N + 6N^2}{p},$$

$$s^c(N;p) = 15Np,$$

$$q^c(N;p) = 50 \cdot (3p),$$

where $s^s(N;p)$ is the number of floating point operations, $s^c(N;p)$ is the number of words being communicated, and $q^c(N;p)$ is the number of communications.
The related factorization are:

$$\begin{array}{cc} s^s_N(N) = 20N + 6N^2, & s^s_p(p) = \dfrac{1}{p} \\ s^c_N(N) = 15N, & s^c_p(p) = p \\ q^c_N(N) = 50, & q^c_p(p) = 3p \end{array}$$

Hence the resulting dimensionless execution time is:

$$\bar{T}(N;p) = \delta_3 \frac{1}{p} + \frac{\delta_3}{\delta_1} p + 3p, \tag{5.16}$$

with the value δ_1, δ_3 expressed as:

$$\delta_1 = \frac{20N + 6N^2}{15N} \frac{r^c_\infty}{r^s_\infty}, \quad \delta_3 = \frac{20N + 6N^2}{50} \frac{1}{r^s_\infty t^c_0}.$$

The optimal number $\tilde{p}$ of processors is derived by differentiating (5.16) with respect to p:

$$\tilde{p} = \sqrt{\frac{\delta_1 \delta_3}{3\delta_1 + \delta_3}},$$

A dimension	Dimensionless parameters		Optim. num. of proc. $\tilde{p}$
	δ_1	δ_3	
$A_{1\times1}$	41.1068	237.7548	5
$A_{2\times2}$	164.4328	3804.344	12
$A_{8\times8}$	657.7063	60864.94	25

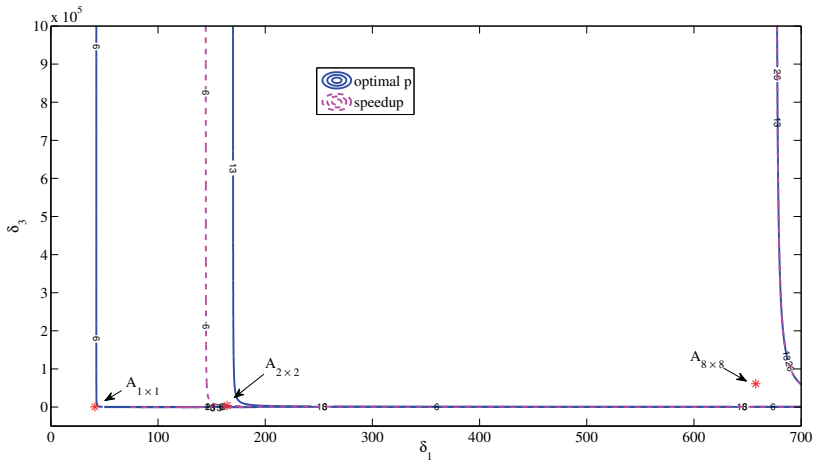
Table 5.2: DUSD parameters for SGP

and the speedup on this optimal number of processors is (B.5) :

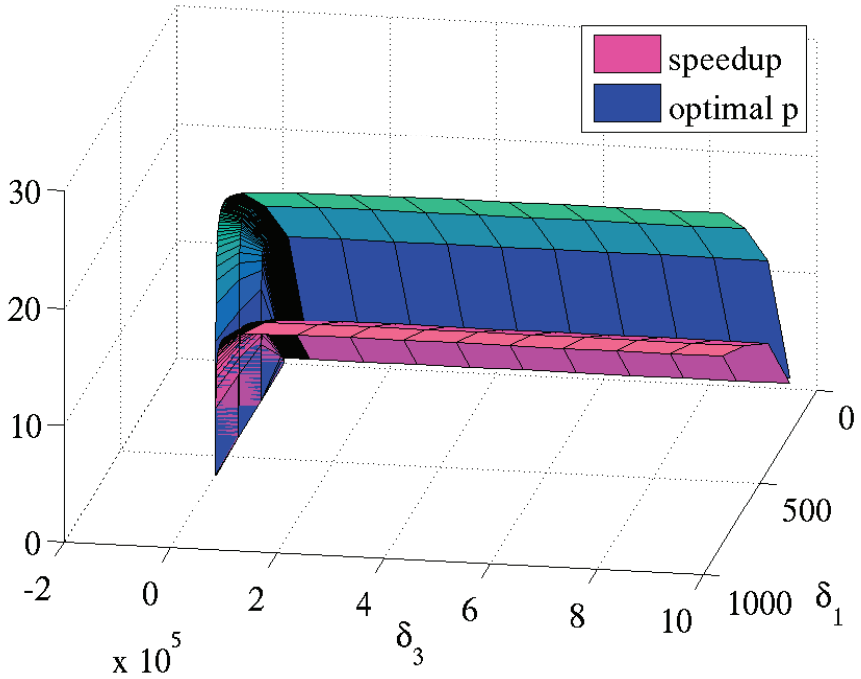
$$S_p(\delta_1, \delta_3, \tilde{p}) = \frac{1}{\tilde{p}} + \frac{\tilde{p}}{\delta_1} + \frac{3\tilde{p}^{-1}}{\delta_3}$$

The next step is to draw the contour plot, using the data in Table 5.1 and Table 5.2, as it is shown in Figure 5.10.

The optimal number of processors required by SGP for the data $A_{1\times1}$, $A_{2\times2}$, and $A_{8\times8}$ is 6, 13, and 26, respectively. This is consistent with the optimal number of processors that can be guessed observing the execution times and speedup plots in Figure 5.5 and Figure 5.6.



(a) The blue dotted lines represent the contour of the optimal number of processor; the magenta dashed lines represent the contour of the optimum speedup. The red stars are the optimal number of processors for $A_{1\times1}$, $A_{2\times2}$, and $A_{8\times8}$.



(b) The blue dotted lines of Figure 5.10(a) are the contour plot generated by the blue surface, that represents the optimum number of processors as δ_1 and δ_3 change; The magenta dotted lines of Figure 5.10(a) are the contour plot generated by the magenta surface that represents the speedup as δ_1 and δ_3 change.

Figure 5.10: DUSD model on SGP

Chapter 6

JoInv performance analysis

We tested JoInv's performance on a synthetic dataset (Figure 6.1) generated by the Matlab code SeismicTomo [129, 131, 145] written by Giulio Vignoli. The test set is characterized by a blocky structure embedded in a uniform background. A series of 12 transmitters (the circles) is vertically aligned on one side of the domain and another series of 22 receivers (the stars) is vertically aligned at the opposite side of the blocky structure. The simulated data are radar and elastic waves. The domain is discretized into $15 \times 16 \times 22$ parallelepiped cells. The blocky structure to be detected involves 64 cells and encloses a volume with different medium density such that the radar wave is decreased with respect to the background, while the elastic wave is increased with respect to the background. We tried to find a real geophysical dataset but we could not.

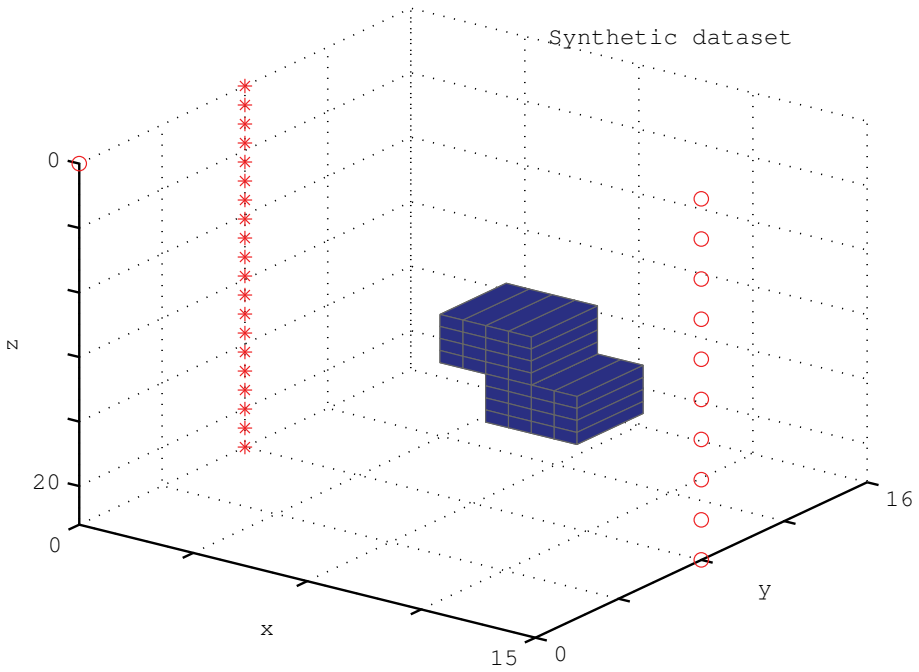


Figure 6.1: Synthetic dataset used for JoInv performance evaluation. The circles indicate the transmitters and the stars indicate the receivers.

Unfortunately, even if the Matlab code that build the synthetic dataset can manage

3D volumes much larger, we can only generate a 3D test with five thousands cells, that is not so much compared with a real dataset. Nevertheless it is enough for a basic performance evaluation because the elements of the matrices involved in the inverse problem are about 1.2 millions (the size depends both on the cells number and the number of the observed data). The results shown in the following plots refer to the JoInv performance when running over 150 iterations on the synthetic dataset previously described; performance results have been obtained on the IBM-SP6 cluster hosted at the CINECA Supercomputing center [118].

The software performances are evaluated by the following standard measures: the execution times, the speedup (Section B.1), the efficiency (Section B.2), and the Kuck's function (Section B.3), that are illustrated in Figure 6.2, Figure 6.3, Figure 6.4, and Figure 6.5, respectively.

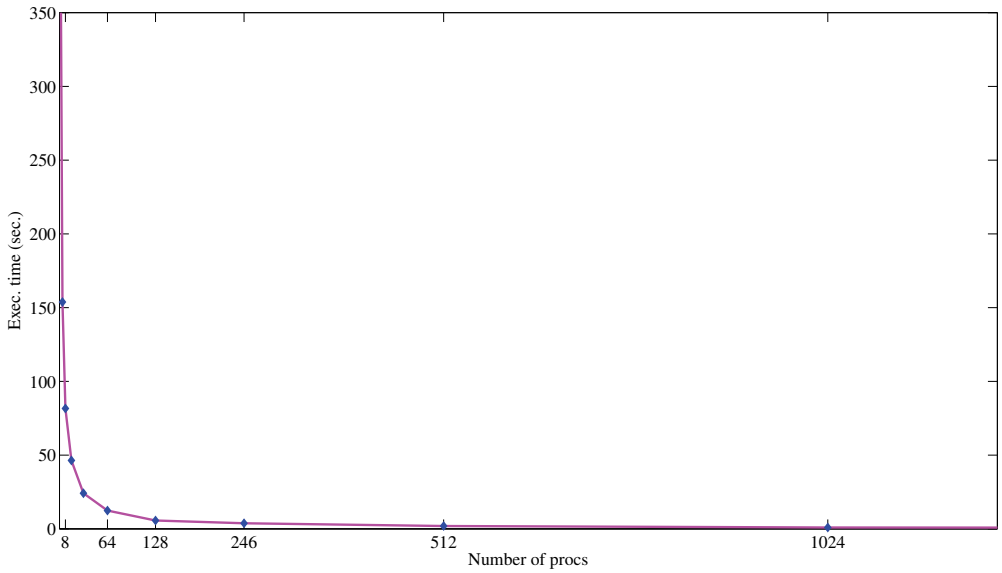


Figure 6.2: JoInv execution times.

The plots demonstrate good scaling property up to 1024 processors; over this threshold, the performance is limited by the startup of the program (data loading, data preallocation, computation of the data that do not change during the execution, sequential run-time portion of the code) and the communications between the processors.

The super-linear speedup and the very high efficiency on 2, 4, and 8 processors are due to the cache effects (memory latency and locality, both spatial and temporal), best load balancing between processes, and efficient communication.

Even though the parallel speedup, from 32 to 2048 processors, will never reach the 1:1 ratio of the ideal curve, the whole job is still being processed in a fraction of the time that would be required for a serial simulation, that often is not possible due to the memory requirements of large cell budgets.

In agreement to the speedup, a high degree of efficiency is obtained in parallel execution up to 1024 processors.

The Kuck's function clearly shows that up to 1024 processors can be effectively exploited by the parallel program on the given simulations. Little improvement is obtained

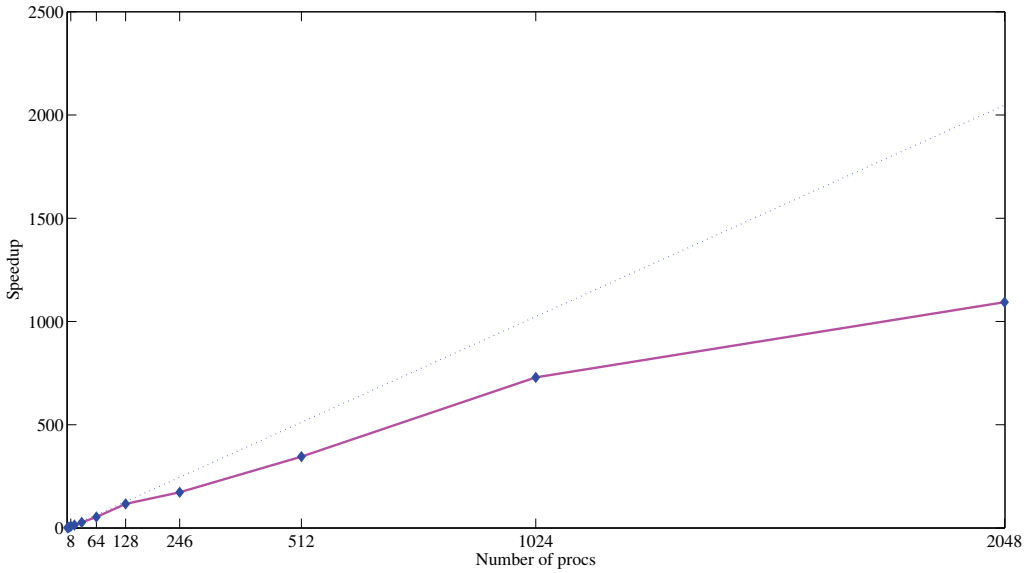


Figure 6.3: JoInv speedup.

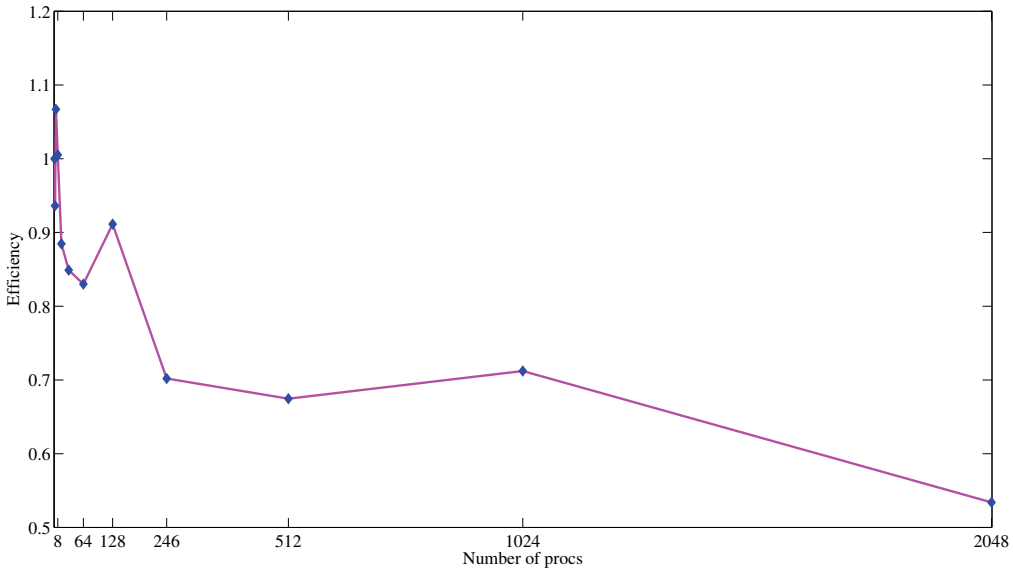


Figure 6.4: JoInv efficiency.

by providing additional processors, even though the performance degrades because the computational load on each processor becomes too small compared with the communication time.

However, as we have seen for SGP, this measure is less accurate than the DUSD estimate, so we feel that the optimal number of processors to be used with this test set should still be lower than that proposed by Kuck's function. In any case, providing accurate estimates of the DUSD parameters for the JoInv code is quite more difficult than for the SGP code, so we leave this activity as a future development.

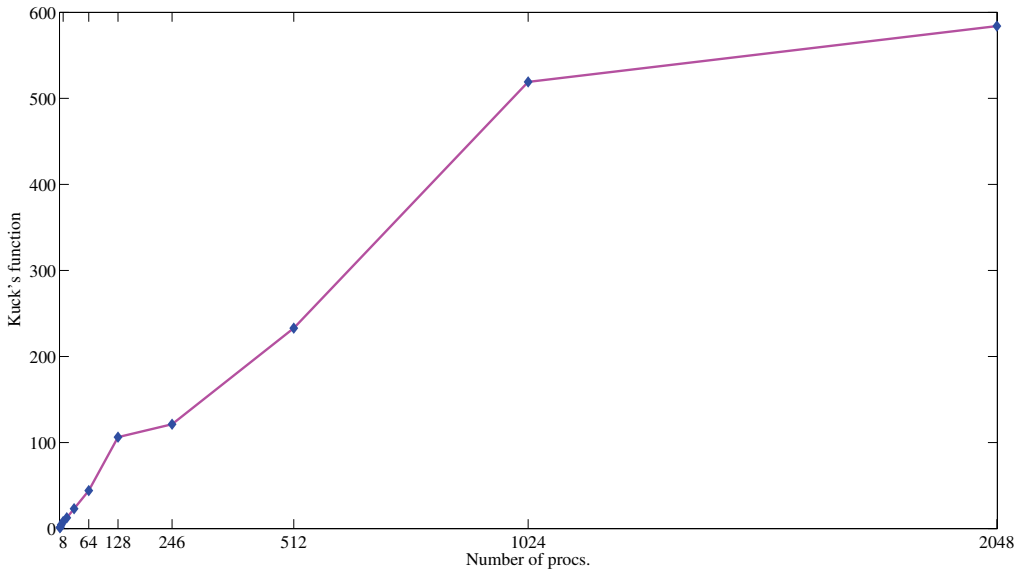


Figure 6.5: JoInv Kuck's function.

Conclusions

In this thesis the problem of jointly inverting multiple multidimensional data is considered from the high-performance computing viewpoint. The parallel implementation of a Tikhonov-like approach is proposed, built on top of the well know high-performance libraries PETSc and TAO. This choice allows the proposed software to be widely portable on different HPC architectures. Equally important, the large number of large-scale scientific application that have already been developed so far on top of the PETSc structure can easily integrate the proposed software, thus accessing a powerful tool to reconstruct hidden object from multiple sets of observations. Particular attention has been given to provide an effective implementation of the edge-preserving regularization based on the minimum-gradient-support functional, which is suitable to recognize blocky structures. Both first- and second-order derivatives have been analytically studied and implemented for the two-models case: this in turn allows the user to investigate a particular problem by using either the first-order or the second-order class of solvers available in the underlying libraries. The theoretical results as well as the practical implementation are easily extensible to the case of more than two models (and the corresponding data sets), provided that enough memory is available to store the additional arrays.

Even if we do not deal with the relevant tasks of how to select appropriate values for the regularization and the focalization parameters, our parallel implementation makes it possible to experiment whatever strategy the user would like to apply for this goal, thus providing a quantitative way to asses the merits and the demerits of classical as well as new criteria.

A large range of application fields can benefit from the study and the outcome of this thesis: from Geophysics to Medicine, from Chemistry to Astronomy, from Microscopy to Engineering. The proposed method and implementation makes it finally possible for the first time in an HPC environment to follow the approach of using the extra information provided by the multiple data sets acquired to perform the data inversion at once, as the cure for the problem ill-posedness. There are obvious large potential advantages in this approach: for example, jointly inverting different kind of data could allow to reduce the total number of data to be collected, without losing the accuracy of the results.

Our performance tests on the developed software show that the parallel implementation of the joint inversion is quite efficient and scalable, as long as the data size are large enough for the selected number of processing elements.

On the way of this main contribution, we have developed additional results which are of independent interest.

Concerning the sparse matrices preallocation problem, we studied and implemented a recursive method that allows to determine the full nonzero structure (that is, the exact locations of all nonzero entries) of a matrices product in a time proportional to the num-

ber of nonzero elements, without the use of graph theory and structures. That method, summarized in Chapter 4, does not show great performance on a parallel environment, because the algorithm's communication part dominates the computation part; nevertheless, since it takes advantage of the PETSc matrix formats, it's very portable and it can be useful when the dimension of the problem becomes too large for a sequential system. Moreover, when problem scale matters, the benefits given by a good and exact memory preallocation largely justifies little extra computations; here it's also important to emphasize that usually this kind of analysis is performed only once at the initialization step, thus the overhead due to the computation of the nonzero structure does not affect the performance of the master code and it is usually completely negligible when problem size becomes larger and larger. A particular aspect of this memory preallocation algorithm is that the strategy we developed preserves its recursive nature also in parallel: this is a quite unusual good feature for a situation where the tasks on different processors are not independent, as it is the case here.

Furthermore, we have implemented the effective Scaled Gradient Projection (SGP) method as a TAO solver for simply constrained smooth minimization problems. This contributes a new first-order iterative method to the TAO library, that can clearly be used independently of the joint inversion code. The implementation involves a number of classical features of descent methods, but it has been hard to code because the general structures available for first-order iterative methods in TAO do not explicitly support the use of more than one parameter-choice routine per method, while SGP needs at least two of them. We propose a flexible approach to overcome the problem, by introducing additional structures that are dynamically instantiated together with the solver when the problem object is initialized. This kind of implementation has several advantages: it preserves the code portability, it directly interfaces with PETSc, and it does not change the invocation syntax conventions of the TAO solvers, so that it's easy also for existing applications to test on it. The effectiveness of the provided SGP implementation has been demonstrated independently on the joint inversion code, on a set of artificially generated test problems of increasing size. The tests show good speedup and scalability properties, until the computational load of the single processor becomes too little to compensate for the communication time. It is worth noticing that we provide deep performance analysis of this solver by using the very accurate estimates provided by the *dimensionless universal scaling diagram* (DUSD). This model allows to carefully take all sequential and parallel aspects of the algorithm into consideration and can foresee its parallel potential better than other performance models. To do that, it requires an effort to get good estimates of some code measures that are uncommon to other models and sometimes hard to obtain. Nevertheless, we found it a useful tool, that surely deserves more attention than that it got.

It is our hope that the tools we developed in this thesis are helpful for both the practitioners and the theoretical scientists, by providing the formers with a more advanced tool for solving their inverse problems, while providing the others with a flexible test bench to study and experiment, for instance, new parameter-selection criteria suitable for the joint inversion. Hopefully, the developers of TAO and PETSc could consider to add the presented tools as official features of their effective libraries.

Future developments

The present work can be expanded in a number of directions. First of all, a fully nonlinear operator can be considered in the forward problem: this clearly impacts on both the derivatives computations and the code development. In fact, from an implementation viewpoint a nonlinear operator requires to be applied by means of a routine call in place of the matrix-vector multiplication we used for the linear case explicitly analyzed here. Moreover, the theory of nonlinear ill-posed problems is more complicated and less advanced than that available for linear ill-posed problems: as a consequence, few algorithms are available for their solution (see for instance [85, 103–105]). However, many real-world inverse problems are nonlinear in nature, so they call for suitable solvers, possibly implemented for HPC systems. The provided code can then be taken as an advanced starting point to build the required software.

Second, most of today's HPC architectures are multiprocessor systems equipped with multicore CPUs, generally grouped in nodes; each CPU has access to its own local memory, but often there is also shared memory available at a node level. This kind of hybrid systems have shown to be very effective in a number of respects and, most important, it seems reasonable to foresee that their potential for computational power will still grow quite a lot in the near future. To exploit this superior computing power one needs to use a *hybrid* coding paradigm, where both the inter-CPU and the inter-core parallelism are suitably used for the computations. Hence, another development direction of this project is surely the investigation of whether and how the code can be best adapted to the MPI/OpenMP hybrid programming paradigm.

Third, additional data distribution should be studied in cooperation with the PETSc developers team. The reason for that is twofold: on one hand, this could be necessary in order to be able to suitably implement effective parallel codes for the hybrid architectures we just mentioned here before and, on the other hand, it could allow the integration of other relevant features in the parallel library, such as fully parallel Fast Fourier Transform (FFT) [1] or wavelet transforms. These tools appear increasingly often in large scale inversion problems, because in certain conditions they allow to greatly lower the computational burden due to the application of the forward operator $\mathcal{A}$. For instance, it often happens that the data formation process can be modeled as a Fredholm integral equation of the first kind: after operator discretization, assuming periodic boundary conditions, the obtained matrix A is block-circulant with circulant blocks. Having a suitable data distribution, in many cases the matrix-vector products involving A can be performed in $\mathcal{O}(N \log N)$ time (N being the total number of pixels/voxels) by using the FFT [29], or a fast trigonometric transform such as the Discrete Cosine Transform (DCT) or the Discrete Sine Transform (DST) [115]. As it is well known, spatially-invariant point spread functions (PSF) generate structured (Toeplitz-like) matrices: this class of problems includes an important part of applications, so it would be a meaningful improvement to provide specialized tools, even if our choice for the present work has been to implement a pretty general approach, suitable also for spatially not-invariant cases, for which those structures no longer appear.

Fourth, additional *data-fidelity* functionals should be considered, other then the least squares loss, to provide the code better flexibility in facing a wider class of inversion problems. Moreover, also additional regularizing functional have to be implemented,

[1] PETSc team recently added support for parallel FFTW in real and complex precision

such as the pure *total variation* (TV) as well as its many variants [134]. Connected to this, some emerging, less classical regularization approaches could be considered for the solution of the joint inversion, such as the FDTR-TV method recently proposed (see for instance [79–81].

Fifth, other tensor-product-based representations can be considered when computing derivatives of discretized multidimensional domains. This could be useful for example when direct solvers can be applied in place of iterative solvers, when least-squares-like loss functions or constraints are considered in the regularized functional. Recent results make this a promising way to face the solution of huge-sized problems on very powerful architectures [42].

Sixth, the DUSD analysis of the SGP method could be extended to estimate the parallel performance of the full joint inversion algorithm. This is surely a difficult task, but it would provide the user with a meaningful tool to foresee what sort of computational benefit to expect by applying such a complex procedure to his/her data, using a given underlying HPC platform.

Seventh, a multithread version of the code could also be useful. Even far from large HPC systems such as the IBM SP6 or BlueGene, today's servers equipped with multi-CPU motherboards can reach considerable performances. They are often part of complex detection machines such as magnetic resonance imaging (MRI) systems, so a multithreaded implementation would allow the joint inversion approach to fully exploit the available hardware. Within this context, considering the increasing number of libraries and tools that have been developed worldwide in the last decade, even a porting to other emerging environments such as the Java world for scientific computing can be considered, along the lines of other similar efforts such as Parallel COLT [135–137].

Eighth, from the prototyping point of view, an interesting potential extension of the Matlab implementation is to integrate it with advanced packages dedicated to inverse problems, such as the Sparco project [10], the AIR Tools toolbox [110], the MOORe Tools toolbox [68, 71], the object oriented Restore Tools package [70], the mxTV package [26], the TVReg package [72] just to mention some of the most known.

Some of the highlighted development directions are easier than others and the list is certainly not exhaustive. However, they overall confirm how appealing, interesting and growing this fascinating research field is. We hope that the present work contributes to make the joint inversion approach more accessible for a wider set of applications, and helps for advances in both the computational and the theoretical sides.

Appendix A

MPI, PETSc and TAO libraries

A.1 MPI (Message Passing Interface)

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum [38], which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient and flexible standard for message passing that will be widely used for writing message passing programs. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact become the "industry standard" for writing message passing programs on HPC platforms [16].

MPI is an interface specification for the developers and users of message passing libraries. By itself, it is not a library – but rather the specification of what such a library should be. Simply stated, the goal of the Message Passing Interface is to provide a widely used standard for writing message passing programs. Interface specifications have been defined for `C` / `C++` and Fortran programs.

History and Evolution

- April 1992: Workshop on Standards for Message Passing in a Distributed Memory Environment, sponsored by the Center for Research on Parallel Computing, Williamsburg, Virginia. The basic features essential to a standard message passing interface were discussed and a working group established to continue the standardization process. Preliminary draft proposal developed subsequently.

- November 1992: Working group meets in Minneapolis. MPI draft proposal (MPI-1) from ORNL presented. Group adopts procedures and organization form the MPI Forum. MPIF eventually comprised of about 175 individuals from 40 organizations including parallel computer vendors, software writers, academia, and application scientists.

- November 1993: Supercomputing 93 conference. Draft MPI standard presented.

- May 1994: Final version of draft released.

- 1996: MPI-2 picked up where the first MPI specification left off and addressed topics which go beyond the first MPI specification.
- Today, MPI implementations are a combination of MPI-1 and MPI-2. A few implementations include the full functionality of both.

At present, the standard has several popular versions: version 1.2 (shortly called MPI-1), which emphasizes message passing and has a static runtime environment, and MPI 2.1 (MPI-2), which includes new features such as parallel I/O, dynamic process management, and remote memory operations. MPI-2 specifies over 500 functions and provides language bindings for ANSI `C` , ANSI Fortran (Fortran90), and ANSI `C++` . Object interoperability was also added to allow for easier mixed-language message passing programming. A side-effect of MPI-2 standardization (completed in 1996) was clarification of the MPI-1 standard, creating the MPI 1.2. Note that MPI-2 is mostly a superset of MPI-1, although some functions have been deprecated. MPI1.2 programs still work under MPI implementations compliant with the MPI-2 standard.

Reasons for using MPI

Standardization MPI is the only message passing library which can be considered a standard. It is supported on virtually all HPC platforms. Practically, it has replaced all previous message passing libraries.

Portability There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard.

Performance Opportunities Vendor implementations should be able to exploit native hardware features to optimize performance.

Functionality Over 115 routines are defined in MPI-1 alone.

Availability A variety of implementations are available, both vendor and public domain.

Programming model

- All parallelism is explicit: the programmer is responsible for correctly identifying parallelism and implementing parallel algorithms using MPI constructs.
- The number of tasks dedicated to run a parallel program is static. New tasks can not be dynamically spawned during run time. (MPI-2 addresses this issue).
- MPI uses objects called communicators and groups to define which collection of processes may communicate with each other. Most MPI routines require you to specify a communicator as an argument.
- Within a communicator, every process has its own unique, integer identifier assigned by the system when the process initializes. Ranks are contiguous and begin at zero. Used by the programmer to specify the source and destination of messages. Often used conditionally by the application to control program execution (if rank=0 do this / if rank=1 do that).

- MPI environment management routines are used for an assortment of purposes, such as initializing and terminating the MPI environment, querying the environment and identity, etc.
- MPI point-to-point operations typically involve message passing between two, and only two, different MPI tasks. One task is performing a send operation and the other task is performing a matching receive operation. There are different types of send and receive routines used for different purposes, *i.e.* synchronous send, blocking send/blocking receive, nonblocking send/nonblocking receive, buffered send, combined send/receive.
- Collective communication are used for different purpose, such as synchronization (processes wait until all members of the group have reached the synchronization point), data movement (broadcast, scatter/gather, all to all), collective computation (one member of the group collects data from the other members and performs an operation on that data). Collective operations are blocking.

MPICH2 as example of a widespread MPI implementation

MPICH2, developed by ANL (Argonne National Laboratory) [91], is a high-performance and widely portable implementation of the Message Passing Interface (MPI) standard (both MPI-1 and MPI-2). The goals of MPICH2 are:

1. to provide an MPI implementation that efficiently supports different computation and communication platforms including commodity clusters (desktop systems, shared-memory systems, multicore architectures), high-speed networks, and proprietary high-end computing systems;
2. to enable cutting-edge research in MPI through an easy-to-extend modular framework for other derived implementations.

MPICH2 is distributed as source (with an open-source, freely available license). It has been tested on several platforms, including Linux, Mac OS/X (PowerPC and Intel), Solaris (32- and 64-bit), and Windows.

We mention here MPICH2 as an example of a widespread MPI implementation. There are also other widespread implementation, *e.g.* OpenMPI, versions 1.4 and 1.5.1 [120], that nowadays are at least as widespread. In addition, most vendors have their own implementations, sometimes based on either of them but optimized for their particular node structure and network capabilities. In that respect MPICH2 has no special role.

A.2 PETSc (Portable, Extensible Toolkit for Scientific Computation)

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers; see, *i.e.*, [5] [4] [6] for a comprehensive discussion and references.

PETSc has successfully demonstrated that the use of modern programming paradigms can ease the development of large-scale scientific application codes in Fortran, `C` , and `C++` . Begun several years ago, the software has evolved into a powerful set of tools for the numerical solution of partial differential equations and related problems on high-performance computers. PETSc consists of a variety of libraries (similar to `C++` classes). Each library manipulates a particular family of objects (for instance, vectors) and the operations one would like to perform on the objects. Some of the PETSc modules deal with

- index sets, including permutations, for indexing into vectors, renumbering, etc;
- vectors;
- matrices (generally sparse);
- distributed arrays (useful for parallelizing regular grid-based problems);
- Krylov subspace methods;
- preconditioners, including multigrid and sparse direct solvers;
- nonlinear solvers;
- timesteppers for solving time-dependent (nonlinear) PDEs.

Each consists of an abstract interface (simply a set of calling sequences) and one or more implementations using particular data structures. Thus PETSc provides clean and effective codes for the various phases of solving PDEs, with a uniform approach for each class of problems. This design enables easy comparison and use of different algorithms (for example, to experiment with different Krylov subspace methods, preconditioners, or truncated Newton methods). Hence PETSc provides a rich environment for modeling scientific applications as well as for rapid algorithm design and prototyping. The libraries enable easy customization and extension of both algorithms and implementations. This approach promotes code reuse and flexibility, and separates the issues of parallelism from the choice of algorithms. The PETSc infrastructure creates a foundation for building large-scale applications. It is useful to consider the interrelationships among different pieces of PETSc. Figure A.1 is a diagram of some of these pieces; Figure A.2 presents several of the individual parts in more detail. These figures illustrate the library's hierarchical organization, which enables users to employ the level of abstraction that is most appropriate for a particular problem.

PETSc uses the MPI standard for all message-passing communication.

PETSc uses routines from

- BLAS ;
- LAPACK;
- LINPACK - dense matrix factorization and solve;
- MINPACK - sequential matrix coloring routines for finite difference Jacobian evaluations;

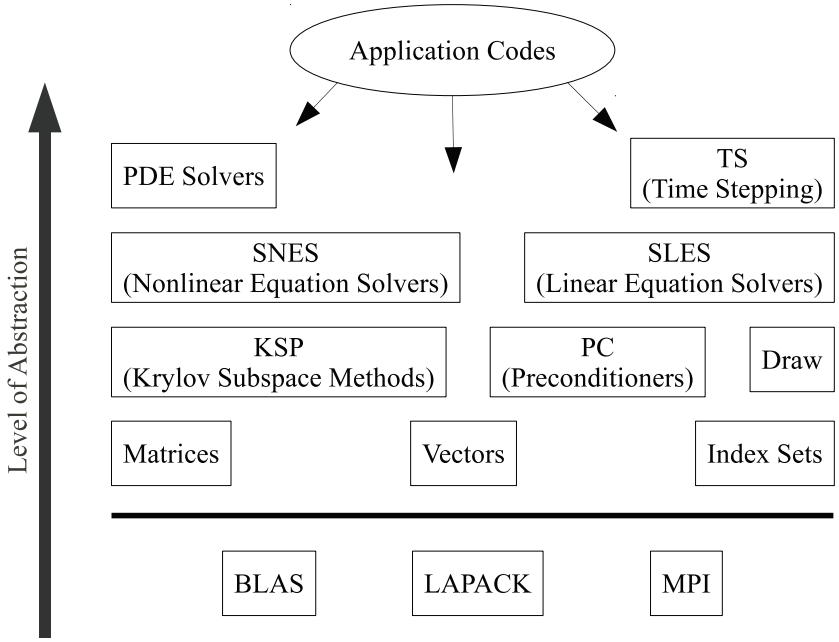


Figure A.1: Organization of the PETSc Libraries.

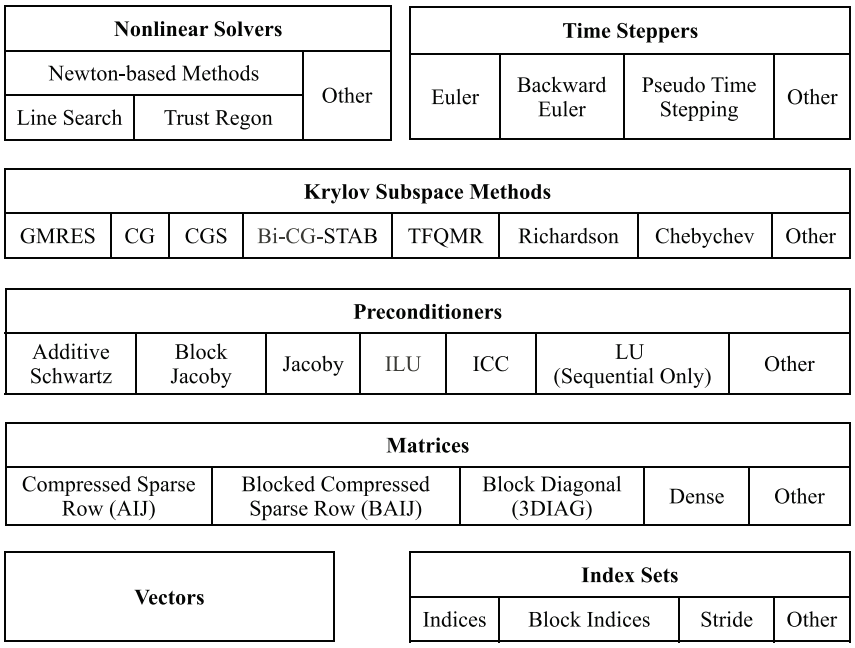


Figure A.2: Numerical libraries of PETSc.

- SPARSPAK - matrix reordering routines;
- libtfs - the efficient, parallel direct solver developed by Henry Tufo and Paul Fischer for the direct solution of a coarse grid problem (a linear system with very few degrees of freedom per processor).

PETSc provides a variety of matrix implementations because no single matrix format is appropriate for all problems. Currently it supports dense and sparse storage (both sequential and parallel versions), as well as several specialized formats. The default matrix

representation within PETSc is the general sparse AIJ format, also called the Yale sparse matrix format or Compressed Sparse Row format, CSR. In the PETSc AIJ matrix formats, the nonzero elements are stored by rows, along with an array of corresponding column numbers and an array of pointers to the beginning of each row (Figure A.3). Note that the diagonal matrix entries are stored with the rest of the nonzeros (not separately).

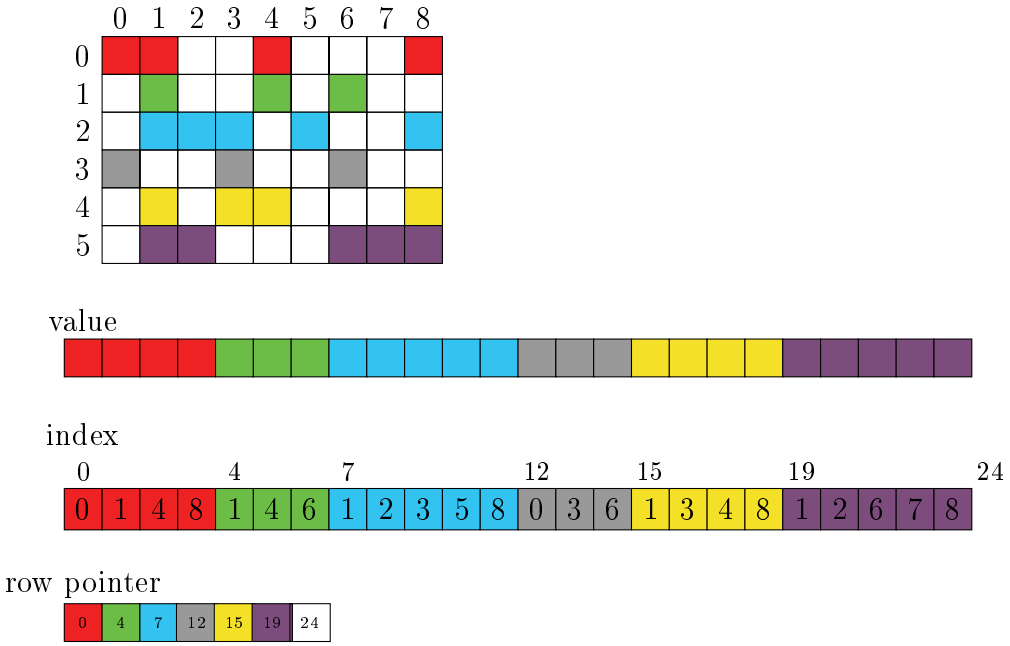


Figure A.3: Compressed Sparse Row format (CSR) or also called the Yale sparse matrix format or AIJ format.

A.3 TAO (Toolkit for Advanced Optimization)

The Toolkit for Advanced Optimization (TAO) focuses on the design and implementation of optimization software for the solution of large-scale optimization applications on high performance architectures; see [8] for a comprehensive discussion and references.

The TAO design philosophy places strong emphasis on the reuse of external tools where appropriate. This design enables bidirectional connection to lower-level linear algebra support (*e.g.* parallel sparse matrix data structures) provided in toolkits such as PETSc as well as higher-level application frameworks. That design decisions are strongly motivated by the challenges inherent in the use of large-scale distributed memory architectures and the reality of working with large and often poorly structured legacy codes for specific applications. Figure A.4 illustrates how the TAO software works with external libraries and application code.

The TAO solvers use four fundamental objects to define and solve optimization problems: vectors, index sets, matrices, and linear solvers. The concepts of vectors and matrices are standard, while an index set refers to a set of integers used to identify particular elements of vectors or matrices. An optimization algorithm is a sequence of well defined operations on these objects. These operations include vector sums, inner products, and matrix-vector multiplication. TAO makes no assumptions about the representation of

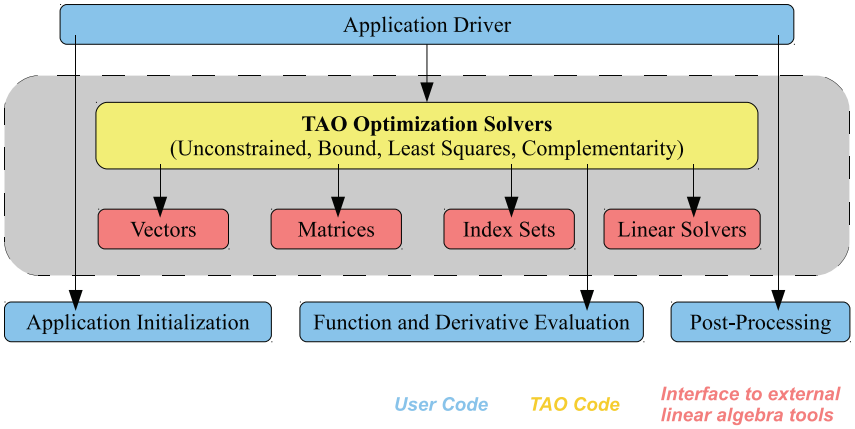


Figure A.4: TAO design.

these objects by passing pointers to data-structure-neutral objects for the execution of these numerical operations. With sufficiently flexible abstract interfaces, TAO can support a variety of implementations of data structures and algorithms. These abstractions allow the user to more easily experiment with a range of algorithmic and data structure options for realistic problems, such as within this case study. Such capabilities are critical for making high-performance optimization software adaptable to the continual evolution of parallel and distributed architectures and the research community's discovery of new algorithms that exploit their features. The current TAO implementation uses the parallel system infrastructure and linear algebra objects offered by PETSc, which uses MPI for all interprocessor communication. The TAO design philosophy eliminates some of the barriers in using independently developed software components by accepting data that is independent of representation and calling sequence written for particular data formats. The user can initialize an application with external frameworks, provide function information to a TAO solver, and call TAO to solve the application problem. The use of abstractions for matrices and vectors in TAO optimization software also enables the developers to leverage automatic differentiation technology to facilitate the parallel computation of gradients and Hessians needed within optimization algorithms.

Appendix B

Measuring parallel performance

There are various methods that are used to measure the performance of a certain parallel program. No single method is usually preferred over another since each of them, as will be seen later on, reflects certain properties of the parallel code. There is extensive literature regarding those methods, *e.g.* [117], [39], [126], [127], [128], [99], [59], [1], [67]; in this appendix are summarized some useful concepts related to the parallel performance measuring.

B.1 Speedup

The most obvious benefit of using a parallel computer is the reduction in the running time of the code. Therefore, when considering the performance of a parallel program on a given machine, it is usually compared to the performance of the same program on a single processor of that machine [99] [128] [126] [59]. A measure of the running time reduction is given by the ratio of the execution time on a single processor (the sequential version) to that on a multicomputer. This ratio is defined as the *speedup factor* and is given as

$$S(p) = \frac{t_s}{t_p}$$

where t_s is the execution time on a single processor and t_p is the execution time on a parallel computer using p processors. $S(p)$ therefore describes the scalability of the system as the number of processors is increased.

The ideal speedup is p when using p processors, i.e. when the computations can be divided into equal duration processes with each process running on one processor (with no communication overhead). This is called *embarrassingly parallel computing*.
A parallel program is considered to be quite good if its speedup is close to p.
In some cases, *super-linear speedup* ($S(p) > p$) may be encountered [59]. Usually this is caused by either using a suboptimal sequential algorithm or some unique specification of the hardware architecture that favors the parallel computation. For example, one common reason for super-linear speedup is that the sub-problem size to be handled by each processor core has become so small that it fits for a significant part into the cache of that core. As the cache memory is much faster and, more important, the memory latency is orders of magnitude lower, the waiting time for operands is much reduced, resulting in a higher performance per core.

There are also more specific definition of speedup. One definition focuses on how much faster a problem can be solved with p processors. Thus, it compares the best sequential algorithm with the parallel one under consideration. This definition is referred as *absolute speedup*:

$$S_a(p) = \frac{t_{\text{bestSeq}}}{t_p}$$

Another speedup, called *relative speedup*, deals with the inherent parallelism of the parallel algorithm under consideration. It is defined as the ratio of elapsed time of the parallel algorithm on one processor to elapsed time of the parallel algorithm on p processors:

$$S_r(p) = \frac{t_{(p=1)}}{t_p}$$

We can write the speedup in term of time required for the sequential and parallel operations of a program.
Let T_s represent the time required for the sequential operations of a parallel program and let T_p represent the amount of time required to complete all the operations that can be done in parallel, but as done in sequentially. Thus, the total time for the program to execute on a sequential machine should be computed as

$$T_1 = T_s + T_p.$$

Now if p is the number of processor that this program can use in parallel, then the total execution time in parallel can be computed as

$$T_{tot} = T_s + \frac{T_p}{p}.$$

Substituting the values for t_s and t_p into the previous equation for speedup yields

$$S(p) = \frac{T_s + T_p}{T_s + \frac{T_p}{p}}.$$

B.1.1 Fixed-size speedup

This viewpoint emphasizes shortening the time a problem takes to solve by parallel processing. With more and more computation power available, the problem can be solved in less and less time. Speedup formulation based on this philosophy is called *fixed-size speedup*. The fixed-size speedup is based on relative speedup. *Amdahl's Law* (see Subsection B.1.4) is an example of fixed-time speedup.

B.1.2 Fixed-time speedup

For some application we may have a time limitation, but we may not want to solve the problem as soon as possible. If we have more computation power, we may want to increase the problem size, do more operations, get a more accurate solution and keep the execution time unchanged. This viewpoint leads to the speedup model called *fixed-time speedup* [59].

One good example is weather forecasting. With more computation power, we may not want give the forecast earlier. Rather, we may wish to add more factors into the weather model giving a more precise forecast.

Gustafson's scaled speedup (see Subsection B.1.5) is a fixed-time speedup. It fixes the response time and is interested in how large a problem could be solved within this time.

Another common definition of fixed-time speedup is the *scaled speedup*:

$$S_\lambda(\lambda, p, \omega) = \lambda \frac{t_p(\omega)}{t_{\lambda p}(\lambda\omega)}, \qquad \omega > 0, \quad \lambda \geq 1$$

where $t_p(\omega)$ is the execution time on p processors for running an algorithm with ω data and $t_{\lambda p}(\lambda\omega)$ is the execution time on λp processors for running the same algorithm with $\lambda\omega$ data.
The scaled speedup curve is a function of how the size of the problem is allowed to grow. A good scaled speedup should be near to λ.

B.1.3 Memory-bounded speedup

When solving an application with one processor, the problem size is more often bounded by the memory limitation than by the execution time limitation. With more nodes available, instead of keeping the execution time fixed, we may want to meet the memory capacity and increase the execution time. In this case, when the problem size is scaled up with system size, we speak of *memory-bounded speedup*.

B.1.4 Amdahl's Law

Amdahl's law, also known as *Amdahl's argument*, is used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors [73], because the speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.

According to [1], *Amdahl's law* states that if α is the proportion that cannot be parallelized (*i.e.* cannot benefit from parallelization) and $(1-\alpha)$ is the proportion of a program that can be made parallel (*i.e.* benefit from parallelization), then the maximum speedup that can be achieved by using p processors is

$$\frac{1}{\alpha + \frac{(1-\alpha)}{p}}.$$

In the limit, as p tends to infinity, the maximum speedup tends to $1/\alpha$. As an example, if α is only 10%, the problem can be speed up by only a maximum of a factor of 10, no matter how large the value of p used.

$(1-\alpha)$ can be estimated by using the measured speedup $S(p)$ on a specific number of processors p using

$$P_{(1-\alpha)} = \frac{\frac{1}{S(p)} - 1}{\frac{1}{p} - 1}.$$

$P_{(1-\alpha)}$ can be used in Amdahl's law to predict speedup for a different number of processors.

B.1.5 Gustafson' speedup model

It is often the case that one would like to solve a large problem or to gain a better accuracy when mode processors are available instead of reducing the execution time. Based on this thought Gustafson introduced a speedup model that scales up the workload with the increasing number of processors in such a way to preserve the execution time.
According to [58] [73], the parallel fraction of the program $(1-\alpha)$ is problem dependent. Therefore, the normalized execution time on a single processor is expressed as:

$$t_s = \alpha + (1-\alpha)p$$

and on p processors:

$$t_p = \alpha + \frac{(1-\alpha)p}{p} = 1$$

Accordingly, the speed up is

$$S(p) = p - \alpha(p-1)$$

In this model, the scalability analysis can be used to determine how far the problem size can be scaled up with the increasing number of processors in order to preserve the execution time.

B.2 Efficiency

The *efficiency* of a parallel system describes the fraction of the time that is being used by the processors for a given computation, *i.e.*how well–utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization [126] [99]. It is defined as

$$E(p) = \frac{t_s}{p \times t_p}$$

where t_s is the execution time using one processor and t_p is the execution time using p processors. By substituting in this equations the previous speedup formulation we obtain

$$E(p) = \frac{S(p)}{p}.$$

An ideal value for $E(p)$ would be 1, meaning $S(p) = p$ and that the processors are 100% utilized throughout the execution of the program and there is not a parallelization overhead. An efficiency of less than p is expected because not all parts of a program can be run in parallel. For instance, input and output are often done by a single processor or many time is spent in communication.

There are also more specific definition of efficiency, *absolute efficiency* and *relative efficiency*, depending on which speedup, absolute or relative, we use.

B.3 Kuck's function

The Kuck's function refers to how advantageous the parallel implementation remains as the number of processors increases. It is defined as

$$K(p) = S(p) \times E(p) = \frac{S^2(p)}{p}$$

The maximizer of the Kuck's function is interpreted as the largest number of processors suitable for the parallel implementation to solve the given particular problem.

B.4 Cost

The *cost* of a computation [99] in a parallel environment is defined as the product of the number of processors used times the total execution time:

$$C(p) = p \times t_p$$

The above equation can be written as a function of the efficiency:

$$C(p) = \frac{t_s}{E(p)}.$$

B.5 Scaling efficiency

A common task in HPC [123] is *measuring the scalability* (also referred to as the *scaling efficiency*) of an application. This measurement indicates how efficient an application is when using increasing numbers of parallel processing elements (CPUs / cores / processes / threads / etc.).

There are two basic ways to measure the parallel performance of a given application, depending on whether or not one is cpu-bound or memory-bound. These are referred to as strong and weak scaling, respectively.

When scaling of parallel codes is discussed it is normally *strong scaling* that is being referred to, that is for a fixed system size how does the time to solution vary with the number of processors. *Weak scaling*, on the other hand, is how the time to solution varies with processor count with a fixed system size per processor. So in a weak scaling study when one doubles the number of processors one also doubles the system size.

B.5.1 Strong scaling

In this case the problem size stays fixed but the number of processing elements is increased. This is used as justification for programs that take a long time to run (something that is cpu-bound). The goal in this case is to find a "sweet spot" that allows the computation to complete in a reasonable amount of time, yet does not waste too many cycles due to parallel overhead.
In strong scaling, a program is considered to *scale linearly* if the speedup (in terms of work units completed per unit time) is equal to the number of processing elements used (p).
In general, it is harder to achieve good strong-scaling at larger process counts since the communication overhead for many/most algorithms increases in proportion to the number of processes used.

If the amount of time to complete a work unit with 1 processing element is t_s, and the amount of time to complete the same unit of work with p processing elements is t_p, the strong scaling efficiency (as a percentage of linear) is given as:

$$SSE(p) = p \times \frac{t_p}{t_s} \times 100\%.$$

B.5.2 Weak scaling

In this case the problem size (workload) assigned to each processing element stays constant and additional elements are used to solve a larger total problem (one that wouldn't fit in RAM on a single node, for example). Therefore, this type of measurement is justification for programs that take a lot of memory or other system resources (something that is memory-bound).
In the case of weak scaling, *linear scaling* is achieved if the run time stays constant while the workload is increased in direct proportion to the number of processors.
Most programs running in this mode should scale well to larger core counts as they typically employ nearest-neighbour communication patterns where the communication overhead is relatively constant regardless of the number of processes used; exceptions include algorithms that employ heavy use of global communication patterns, *e.g.*FFTs and transposes.

If the amount of time to complete a work unit with 1 processing element is t_s, and the amount of time to complete p of the same work units with p processing elements is t_p, the weak scaling efficiency (as a percentage of linear) is given as:

$$WSE(p) = \frac{t_p}{t_s} \times 100\%$$

.

B.6 Full timing model

A parallel system is defined as a combination of a parallel algorithm and parallel architecture on which the algorithm is implemented. Therefore, performance of the parallel system should consider the parallel architecture as well as the algorithm.

This is the underlying concept of DUSD model (see Section B.7). The execution time in DUSD model is based on three hardware and three software parameters, named as *3-parameter timing model* [67]. This execution time of a parallel program on N problem size using p processors is [67] [128] [73]:

$$T(N;p) = \frac{s^s(N;p)}{r^s_\infty} + \frac{s^c(N;p)}{r^c_\infty} + t^c_0 q^c(N;p)$$

where the hardware parameters are:

- r^s_∞ is the computation rate (flop/s);
- r^c_∞ is the asymptotic communication bandwidth (byte/s);
- t^c_0 is the message latency (s);

and the program parameters are:

- $s^s(N;p)$ is the number of floating point operations (flop);
- $s^c(N;p)$ is the the number of words being communicated (byte);
- $q^c(N;p)$ is the number of communications (1);

Now assume that the functions $s^s(N;p)$, $s^c(N;p)$ and $q^c(N;p)$ are separable in N and p. We can write $T(N;p)$ as

$$T(N;p) = \frac{s^s_N(N)s^s_p(p)}{r^s_\infty} + \frac{s^c_N(N)s^c_p(p)}{r^c_\infty} + t^c_0 q^c_N(N)q^c_p(p) \tag{B.1}$$

By differentiating (B.1) with respect to p and setting

$$\frac{\partial T(N;p)}{\partial p} = 0$$

we are able to find the optimum p^* and therefore an optimal temporal performance

$$R^*_T = \frac{1}{T^*(N;p^*(N))}$$

which is only dependent on N.

B.7 Dimensionless Universal Scaling Diagram (DUSD)

By making (B.1) dimensionless we can reduce the number of parameter in out timing model from three hardware parameters (r^s_∞ t^c_0) and three software parameters ($s^s(N;p)$ $s^c(N;p)$ $q^c(N;p)$) to only two dimensionless ratio [128] [73]. This is achieved by dividing (B.1) by $t^c_0 q^c_N(N)$; the resulting equation is termed as dimensionless execution time expressed as:

$$\bar{T}(N;p) = \delta_3 s^s_p(p) + \delta_2 s^c_p(p) + q^c_p(p) \tag{B.2}$$

with

$$\delta_2(N;t^c_0,r^c_\infty) = \frac{s^c_N(N)}{q^c_N(N)t^c_0 r^c_\infty}$$

$$\delta_3(N;t^c_0,r^s_\infty) = \frac{s^s_N(N)}{q^c_N(N)t^c_0 r^s_\infty}$$

where δ_2 represents the *dimensionless message length* and δ_3 is the *dimensionless work*.

Hockney defined a parameter that represents the ratio of δ_3 and δ_2 as *dimensionless computational intensity* δ_1 that is formulated as:

$$\delta_1(N;t^c_0,r^c_\infty) = \frac{s^s_N(N)}{s^c_N(N)}\left(\frac{r^c_\infty}{r^s_\infty}\right)$$

Substituting δ_1 into (B.2) yields the following expression:

$$\bar{T}(N;p) = \delta_3 s^s_p(p) + \frac{\delta_3}{\delta_1} s^c_p(p) + q^c_p(p) \tag{B.3}$$

The optimum number of processors in order to achieve a minimum execution time is computed by differentiating the dimensionless execution time in (B.3) with regards to the optimum number of processors.

Speedup of the parallel program is quantified by comparing execution time on a single processor to execution time on optimum number of processors. Hence, in terms of dimensionless time the speedup is defined as:

$$S_p(N;p) = \frac{\bar{T}(N;1)}{\bar{T}(N;p)} \tag{B.4}$$

with

$$\bar{T}(N;1) = \delta_3 s_p^s(1) \qquad \text{(there is no communication)}$$

$S_p(\delta_2, \delta_3, p)$ can be expressed in term of δ_2 δ_3 as follows:

$$S_p(\delta_2, \delta_3, p) = \frac{\delta_3 s_p^s(1)}{\delta_3 s_p^s(p) + \delta_2 s_p^c(p) + q_p^c(p)}$$

and for optimum $\tilde{p}$:

$$\tilde{S}_p(\delta_2, \delta_3, \tilde{p}(\delta_2, \delta_3)) = \frac{\delta_3 s_p^s(1)}{\delta_3 s_p^s(\tilde{p}) + \delta_2 s_p^c(\tilde{p}) + q_p^c(\tilde{p})}$$

We can also write the speedup formula for optimum number of processors in terms of δ_1 and δ_3 substituting (B.3) in (B.4):

$$S_p(\delta_1, \delta_3, \tilde{p}) = \frac{\frac{s_p^s(1)}{s_p^s(\tilde{p})}}{1 + \frac{1}{\delta_1}\frac{s_p^c(\tilde{p})}{s_p^s(\tilde{p})} + \frac{1}{\delta_3}\frac{q_p^c(\tilde{p})}{s_p^s(\tilde{p})}} \tag{B.5}$$

The representation of the this speedup can be conveniently done in a 2-D graph where the variables on the axes are $\delta_1 = \delta_3/\delta_2$ and δ_2. This graph is calls *Dimensionless Universal Scaling Diagram* (DUSD).

Any programs that posses the same ratios δ_2 and δ_3 (δ_1) will have the same optimum number of processors $\tilde{p}$ and the same optimum speedup $\tilde{S}_p$. Such programs all called *Computationally Similar*.

List of Figures

List of Tables

List of Algorithms

List of Listings

Bibliography

[1] Gene Myron Amdahl. Validity of single-processor approach to achieving large-scale computing capability. In *Proceedings of the AFIPS, Spring Joint Computer Conference*, pages 483–485, 1967.

[2] Richard Aster, Brian Borchers, and Clifford Thurber. *Parameter Estimation and Inverse Problems*. International Geophysics Series. Academic Press, Burlington, MA, USA, 2005.

[3] A. B. Bakushinsky and A. V. Goncharsky. *Ill-Posed Problems. Theory and Applications*, volume 301 of *Mathematics and Its Applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.

[4] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[5] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc web page, 2009. www.mcs.anl.gov/petsc.

[6] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[7] Jonathan Barzilai and Jonathan Michael Borwein. Two point step size gradient methods. *IMA J. of Numer. Anal.*, 8(1):141–148, 1988.

[8] Steve Benson, Lois Curfman McInnes, Jorge Moré, Todd Munson, and Jason Sarich. TAO user manual (revision 1.9). Technical Report ANL/MCS-TM-242, Mathematics and Computer Science Division, Argonne National Laboratory, 2007. www.mcs.anl.gov/tao.

[9] F. Benvenuto, Riccardo Zanella, Luca Zanni, and Mario Bertero. Nonnegative least-squares image deblurring: improved gradient projection approaches. *Inverse Problems*, 26(2):025004, February 2010.

[10] E. van den Berg, M. P. Friedlander, G. Hennenfent, F. Herrmann, R. Saab, and Ö. Yılmaz. Sparco: A testing framework for sparse reconstruction. Technical Report

TR-2007-20, Dept. Computer Science, University of British Columbia, Vancouver, October 2007. www.cs.ubc.ca/labs/scl/sparco/index.php/Main/HomePage.

[11] Mario Bertero, Patrizia Boccacci, , Gabriele Desiderà, and G. Vicidomini. Image deblurring with poisson data: from cells to galaxies. *Inverse Problems*, 25:123006, 2009.

[12] Mario Bertero and Patrizia Boccacci. *Introduction to Inverse Problems in Imaging*. Institute of Phisics Publ., Bristol, UK, 1998.

[13] Mario Bertero, Patrizia Boccacci, Giorgio Talenti, Riccardo Zanella, and Luca Zanni. A discrepancy principle for poisson data. *Inverse Problems*, 26(10):105004, October 2010.

[14] Mario Bertero, H. Lantiéri, and Luca Zanni. Iterative image reconstruction: a point of view. In *Proc. of "Mathematical Methods in Biomedical Imaging and Intensity-Modulated Radiation Therapy (IMRT)"*, Pisa, Italy, 2007. Pubbl. Centro De Giorgi.

[15] Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

[16] Lawrence Livermore National Laboratory Blaise Barney. Message Passing Interface (MPI), 2010. https://computing.llnl.gov/tutorials/mpi/.

[17] Silvia Bonettini and Valeria Ruggiero. A discrepancy principle for Poisson data: uniqueness of the solution for 2D and 3D data. Technical Report 195, Department of Mathematics, University of Ferrara, Italy, 2010.

[18] Silvia Bonettini and Valeria Ruggiero. On the uniqueness of the solution of image reconstruction problems with Poisson data. In *ICNAAM 2010: International Conference on Numerical Analysis and Applied Mathematics 2010*, volume 1281 of *AIP conference proceedings*, pages 1803–1806. Institute of Phisics Publ., 2010.

[19] Silvia Bonettini, Riccardo Zanella, and Luca Zanni. A scaled gradient projection method for constrained image deblurring. *Inverse Problems*, 25(1):015002, January 2009.

[20] Nicolas Bourbaki. *Algèbre, Chap. III, Algèbre Miltilinéaire*. Herman, Paris, France, 1948.

[21] Paola Brianzi, Fabio Di Benedetto, and Claudio Estatico. Improvement of space-invariant image deblurring by preconditioned landweber iterations. *SIAM J. Scientific Computing*, 30:1430–1458, 2008.

[22] Rogério Brito. *The official manual for the algorithms package*, 2009. www.ctan.org/get/macros/latex/contrib/algorithms/algorithms.pdf.

[23] Edith Cohen. Structure prediction and computation of sparse matrix products. *Journal of Combinatorial Optimization*, 2:307–332, 1999.

[24] D. Colombo and M. De Stefano. Geophysical modeling via simultaneous joint inversion of seismic, gravity, and electromagnetic data: Application to prestack depth imaging. *The Leading Edge*, 26(3):326–331, March 2007.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction To Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[26] J. Dahl, P.C. Hansen, S.H. Jensen, and T.L. Jensen. Algorithms and software for total variation image reconstruction via first-order methods. *Numerical Algorithms*, 53:67–92, 2010.

[27] Yu-Hong Dai and Roger Fletcher. New algorithms for singly linearly constrained quadratic programming problems subject to lower and upper bounds. *Math. Programming*, 106(3):403–421, 2006.

[28] Yu-Hong Dai, William W. Hager, Klaus Schittkowski, and Hongchao Zhang. The cyclic Barzilai–Borwein method for unconstrained optimization. *IMA J. Numer. Anal.*, 26:604–627, 2006.

[29] Philip J. Davis. *Circulant Matrices*. John Wiley & Sons, New York, NY, USA, 1979.

[30] JR De Mey, P Kessler, Julien Dompierre, FP Cordelires, Alain Dieterlen, JL Vonesch, and Jean-Baptiste Sibarita. Fast 4D microscopy. *Methods in Cell Biology*, 85:83–112, 2008.

[31] Nicolas Dey, Laure Blanc-Féraud, Christophe Zimmer, Zvi Kam, Pascal Roux, Jean-Christophe Olivo-Marin, and Josiane Zerubia. Richardson-Lucy algorithm with total variation regularization for 3D confocal microscope deconvolution. *Microscopy Research Technique*, 69:260–266, April 2006.

[32] Nicolas Dey, Laure Blanc-Féraud, Christophe Zimmer, Pascal Roux, Zvi Kam, Jean-Christophe Olivo-Marin, and Josiane Zerubia. 3D microscopy deconvolution using Richardson-Lucy algorithm with total variation regularization. Research Report RR-5272, INRIA, Sophia Antipolis Cedex, France, 2004.

[33] Fabio Di Benedetto. The mth difference operator applied to l^2 functions on a finite interval. *Linear Algebra and its Applications*, 366:173–198, 2003.

[34] Fabio Di Benedetto, Claudio Estatico, and Stefano Serra Capizzano. Superoptimal preconditioned conjugate gradient iteration for image deblurring. *SIAM J. Scientific Computing*, 26:1012–1035, 2005.

[35] Heinz W. Engl, Martin Hanke, and Andreas Neubauer. *Regularization of Inverse Problems*, volume 375 of *Mathematics and Its Applications*. Kluwer Academic Publ., Dordrecht, The Netherlands, 2000.

[36] Joel Feldman and Gunther Uhlmann. *Inverse problems*. http://www.math.ubc.ca/~feldman/ibook, 2005.

[37] Roger Fletcher. On the Barzilai-Borwein method. Technical Report NA/207, University of Dundee, 2001.

[38] MPI Forum. Message Passing Interface (MPI) forum home page. www.mpi-forum.org.

[39] Lloyd D. Fosdick, Carolyn J. C. Schauble, and Elizabeth R. Jessup. Computer performance: a tutorial, 1994.

[40] Giacomo Frassoldati, Gaetano Zanghirati, and Luca Zanni. New adaptive step-size selections in gradient methods. *J. of Indutrial and Management Optimization*, 4(2):299–312, 2008.

[41] Ana Friedlander, José Mario Martínez, Brigida Molina, and Marcus Raydan. Gradient method with retards and generalizations. *SIAM Journal on Numerical Analysis*, 36(1):275–289, 1998.

[42] Michael P. Friedlander and Kathrin Hatz. Computing non-negative tensor factorizations. *Optimization Methods & Software*, 23(4):631–647, August 2008.

[43] Luis Alonso Gallardo. *Joint two-dimensional inversion of geoelectromagnetic and seismic refraction data with cross-gradients constraint*. PhD thesis, Lancaster University, 2004.

[44] Luis Alonso Gallardo. Multiple cross-gradient joint inversion for geospectral imaging. *Geophys. Res. Lett.*, 34:L19301, 2007.

[45] Luis Alonso Gallardo and M. A. Meju. Characterization of heterogeneous near-surface materials by joint 2d inversion of dc resistivity and seismic data. *Geophys. Res. Lett.*, 30(13):1658–1–4, 2003.

[46] Luis Alonso Gallardo and M. A. Meju. Joint two-dimensional dc resistivity and seismic travel time inversion with cross-gradients constraints. *J. Geophys. Res.*, 109:03311, 2004.

[47] Luis Alonso Gallardo and M. A. Meju. Joint 2d cross-gradient imaging of magnetotelluric and seismic travel-time data for structural and lithological classification. *Geophys. J. Int.*, 169:1261–1272, 2007.

[48] Luis Alonso Gallardo, M. A. Meju, and Marco A. Perez-Flores. A quadratic programming approach for joint image reconstruction: mathematical and geophysical examples. *Inverse Problems*, 21:435–452, 2005.

[49] Luis Alonso Gallardo, Marco A. Perez-Flores, and Enrique Gomez-Treviño. A versatile algorithm for joint 3-d inversion of gravity and magnetic data. *Geophysics*, 68:949–959, 2003.

[50] Donald Geman and Chengda Yang. Nonlinear image recovery with half-quadratic regularization. *IEEE Trans. Image Proc.*, 4(7):932–946, July 1995.

[51] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[52] John R. Gilbert. Predicting structure in sparse matrix computations. *SIAM J. Matrix Anal. Appl*, 15:62–79, 1994.

[53] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Anal. Appl*, 13:333–356, 1992.

[54] John R. Gilbert, Esmond G. Ng, and G. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. In *Graph Theory and Sparse Matrix Computation*, pages 107–139. Springer-Verlag, 1992.

[55] Luigi Grippo, Francesco Lampariello, and Stefano Lucidi. A nonmonotone line–search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[56] Charles W. Groetsch. *Inverse Problems in the Mathematical Sciences*. Informatica International, Inc., Vieweg, Braunschweig, 1993.

[57] A. Guillem and V. Manichetti. Gravity and magnetic inversion with minimization of specific functional. *Geopysics*, 49(8):1354–1360, August 1984.

[58] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31:532–533, May 1988.

[59] John L. Gustafson. Fixed time, tiered memory, and superlinear speedup. In *Proceedings of the Fifth Distributed Memory Computing Conference (DMCC5)*, 1990.

[60] Eldad Haber, Uri M. Ascher, and Doug Oldenburg. 3-d electromagnetic inversion based on quasi-analytical approximation. *Inverse Problems*, 16(5):1297–1322, October 2000.

[61] Eldad Haber and Doug Oldenburg. Joint inversion: A structural approach. *Inverse Problems*, 13:63–77, 1997.

[62] Jaques Hadamard. Sur les problèmes aux dérivées et leur signification physique. *Bulletin of the Princeton University*, 13(1–20), 1902.

[63] Jaques Hadamard. *Lectures on Cauchy's Problem in Linear Partial Differential Equations*. Yale University Press, New Haven, CT, USA, 1923.

[64] Martin Hanke and Per Christian Hansen. Regularization methods for large-scale problems. *Surveys of Mathematics for Industry*, 3:253–315, 1993.

[65] Per Christian Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. Monographs on Mathematical Modeling and Computation. SIAM, Philadelphia, PA, USA, 1998.

[66] Per Christian Hansen, James G. Nagy, and Dianne P. O'Leary. *Deblurring Images. Matrices, Spectra and filtering*. SIAM, Philadelphia, PA, USA, 2006.

[67] Roger W. Hockney. *The Science of Computer Benchmarking*. SIAM, Philadelphia, PA, USA, 1996.

[68] M. Jacobsen. *Modular Regularization Algorithms*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2004. Supervised by Prof. Per Christian Hansen.

[69] Nathan Jacobson. *Lecturs in Abstract Algebra. Vol. 2. Linear Algebra*. Van Nostrand, Princeton, NJ, USA, 1953.

[70] Emory University James Nagy. RestoreTools - An Object Oriented Matlab Package for Image Restoration, 2007. www.mathcs.emory.edu/~nagy/RestoreTools/.

[71] T. K. Jensen. *Stabilization Algorithms for Large-Scale Problems*. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2006. Supervised by Per Christian Hansen, IMM.

[72] T. L. Jensen, J. H. Jorgensen, P. C. Hansen, and S. H. Jensen. Implementation of an optimal first-order method for strongly convex total variation regularization. In *submitted to BIT*, 2011.

[73] Maria A. Kartawidjaja. Analyzing scalability of parallel matrix multiplication using DUSD. *Asian Journal of Information Technology*, 9:78–84, 2010.

[74] Joseph B. Keller. Inverse problems. *American Mathematical Monthly*, 83:107–118, 1976.

[75] Samir Khuller and Uzi Vishkin. On the parallel complexity of digraph reachability, December 1994.

[76] Andreas Kirsch. *An introduction to the mathematical theory of inverse problems*, volume 120 of *Applied Mathematical Sciences*. Springer-Verlag, New York, NY, USA, 1996.

[77] Andreas Kirsch and Natalia Grinberg. *The Factorization Method for Inverse Problems*, volume 36 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, New York, NY, USA, 2008.

[78] Rainer Kress. *Linear Integral Equations*, volume 82 of *Applied Mathematical Sciences*. Springer-Verlag, New York, NY, USA, 2nd edition, 1999.

[79] Germana Landi and Elena Loli Piccolomini. A fast projected quasi-Newton method for nonnegative Tikhonov regularization. *International J. of Mathematics and Computer Science*, 3(3):199–213, 2008.

[80] Germana Landi and Elena Loli Piccolomini. An algorithm for image denoising with automatic noise estimate. *J. of Mathematical Imaging and Vision*, 34(1):98–106, 2009.

[81] Germana Landi, Elena Loli Piccolomini, and Fabiana Zama. A Total Variation-based reconstruction method for dynamic MRI. *Computational and Mathematical Methods in Medicine*, 9(1):69–80, 2008.

[82] B. J. Last and K. Kubik. Compact gravity inversion. *Geophysics*, 48(6):713–721, June 1983.

[83] Roy Lopamudra, Mrinal K. Sen, Kirk McIntosh, Paul L. Stoffa, and Yosio Nakamura. Joint inversion of first arrival seismic travel-time and gravity data. *J. Geophysics and Engineering*, 2(3):277–290, 2005.

[84] I. Loris, Mario Bertero, Christine de Mol, Riccardo Zanella, and Luca Zanni. Accelerating gradient projection methods for ℓ_1-constrained signal recovery by steplength selection rules. *Applied and Computational Harmonic Analysis*, 27:247–254, 2009.

[85] Shuai Lu, Sergei V. Pereverzev, and Ronny Ramlau. An analysis of Tikhonov regularization for nonlinear ill-posed problems under a general smoothness assumption. *Inverse Problems*, 23(1):217–230, February 2007.

[86] Leon B. Lucy. An iterative technique for the rectification of observed distributions. *Astronom. J.*, 79:745–754, 1974.

[87] Robert E. Lynch, John R. Rice, and Donald H. Thomas. Direct solution of partial difference equations by tensor product methods. *Numerische Mathematik*, 6:185–199, 1964.

[88] Robert E. Lynch, John R. Rice, and Donald H. Thomas. Tensor product analysis of alternating direction implicit methods. *SIAM J.*, 6:185–199, 1964.

[89] M. Maceira, C. A. Rowe, B. Borchers, and L. K. Steck. Simultaneous joint inversion of multiple geophysical data sets and 3d tomography. In *AGU Fall Meeting 2008*, pages S23A–1868. American Geophysical Union, 2008.

[90] Marvin Marcus. *Basic Theorems in Matrix Theory*, volume 57 of *Applied Mathematics Series*. National Bureau of Standards, Washington, D.C., USA, 1960.

[91] ANL Mathematics and Computer Science Division. The Message Passing Interface (MPI) standard, 2010. www.mcs.anl.gov/research/projects/mpi/.

[92] Keith Miller. Least squares methods for ill-posed problems with a prescribed bound. *SIAM J. on Mathematical Analysis*, 1(52–74), 1970.

[93] Jean-Michel Morel and Sergio Solimini. *Variational methods in image segmentation*, volume 14 of *Progress in Nonlinear Differential Equations and Their Applications*. Burkh auser, Boston, MA, USA, 1995.

[94] V. A. Morozov. On the solution of functional equations by the method of regularization. *Soviet Dokledy Mathematics*, 7(414–417), 1966.

[95] V. A. Morozov. *Regularization Methods for Ill-Posed Problems*. CRC Press, Boca Raton, Florida, USA, 1993.

[96] David Mumford and Jayant Shah. Boundary detection by minimizing functionals. In *Proc. IEEE Conference on Computational Vision and Pattern Recognition*, pages 22–26, 1985.

[97] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Comm. Pure Appl. Math.*, 42:577–685, 1989.

[98] Frank Natterer. *The mathematics of computerized tomography*, volume 32 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, USA, 2001.

[99] CFD online. Parallel computing. www.cfd-online.com/Wiki/Parallel_computing.

[100] Charles Émile Picard. Sur un théorème générale relatif aux équations intégrales de première espèce et sur quelques problèmes de physique mathématique. *Rendiconti del Circolo Matematico di Palermo*, 29(79–97), 1910.

[101] N. Polydorides. *Image Reconstruction Algorithms for Soft-Field Tomography*. PhD thesis, Manchester University, 2002.

[102] Oleg Portniaguine and Michael S. Zhdanov. Focusing geophysical inversion images. *Geophysics*, 64(3):874–887, 1999.

[103] Ronny Ramlau. TIGRA – an iterative algorithm for regularizing nonlinear ill-posed problems. *Inverse Problems*, 19(2):433–465, 2003.

[104] Ronny Ramlau. *Regularization of Nonlinear Ill-Posed Operator Equations: Methods and Applications*. PhD thesis, Center of Technomathematics, Bremen, Germany, January 2004.

[105] Ronny Ramlau and G. Teschke. Tikhonov replacement functionals for iteratively solving nonlinear operator equations. *Inverse Problems*, 21(5):1571–1592, 2005.

[106] Alexander G. Ramm. *Inverse Problems. Mathematical and Analytical Techniques with Applications to Engineering*. Springer Science + Business Media, Inc., Boston, MA, USA, 2005.

[107] William Hadley Richardson. Bayesian–based iterative method of image restoration. *J. Opt. Soc. Amer. A*, 62(1):55–59, 1972.

[108] Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. In *Foundations of Computer Science*, pages 679–688, 2002.

[109] Valeria Ruggiero, Thomas Serafini, Riccardo Zanella, and Luca Zanni. Iterative regularization algorithms for constrained image deblurring on graphics processors. *Journal of Global Optimization*, 48:145–157, 2010.

[110] Maria Saxild-Hansen. *AIR Tools - A matlab Package for Algebraic Iterative Reconstruction Techniques*. PhD thesis, Technical University of Denmark, Department of Informatics and Mathematical Modeling, Scientific Computing, 2010.

[111] Thomas Serafini, Gaetano Zanghirati, and luca Zanni. Gradient projection methods for quadratic programs and applications in training support vector machines. *Optimization Methods and Software*, 20(2–3):343–378, 2005.

[112] Larry A. Shepp and Yehuda Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Transaction on Medical Imaging*, 1(2):113–122, 1982.

[113] Jean-Baptiste Sibarita. Deconvolution microscopy. *Adv. Biochem. Eng./Biotechnol.*, 95:201–243, 2005.

[114] Derrick Stolee, Chris Bourke, and N. V. Vinodchandran. A log-space algorithm for reachability in planar acyclic digraphs with few sources. In *Proceedings of the 2010 IEEE 25th Annual Conference on Computational Complexity*, CCC '10, pages 131–138, Washington, DC, USA, 2010. IEEE Computer Society.

[115] Gilbert Strang. The discrete cosine transform. *SIAM Review*, 41(1):135–147, 1999.

[116] Wei-Jia Su and Adam M. Dziewonski. Simultaneous inversion for 3-d variations in shear and bulk velocity in the mantle. *Physics of The Earth and Planetary Interiors*, 100(1–4):135–156, March 1997.

[117] Xian-He Sun and Lionel M. Ni. Another view on parallel speedup. *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, pages 324–333, 1990.

[118] Cineca support. Cineca sp6 user guide. hpc.cineca.it/content/ibm-sp6-user-guide.

[119] Albert Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, Philadelphia, PA, USA, 2005.

[120] Open MPI Team. Open MPI: Open Source High Performance Computing, 2010. www.open-mpi.org/.

[121] Scalasca team. Cube, a performance report explorer for scalasca. /www.fz-juelich.de/jsc/datapool/scalasca/CubeGuide.pdf.

[122] Scalasca team. Scalasca, a software tool that supports the performance optimization of parallel programs by measuring and analyzing their runtime behavior. www.scalasca.org/start.html.

[123] SharcNet Team. Measuring parallel scaling performance. www.sharcnet.ca/help/index.php/Measuring_Parallel_Scaling_Performance.

[124] Andreĭ Nikolaevich Tikhonov and Vasilĭ ÍAk̀ovlevich Arsenin. *Solutions of ill-posed problems*. John Wiley & Sons, Winston, UK, 1977.

[125] V. F. Turchin, V. P. Kozlov, and M. S. Malkevich. The use of mathematical-statistics methods in the solution of incorrectly posed problems. *Soviet Physics Uspekhi*, 13(6):681–703, 1971.

[126] Aad J. van der Steen. Lecture 1: Metrics, methodology and presentation of results. /www.phys.uu.nl/~steen/vecp2k.html.

[127] Aad J. van der Steen. Lecture 2: Low-level performance parameters and benchmarks. /www.phys.uu.nl/~steen/vecp2k.html.

[128] Aad J. van der Steen. Lecture 5: Scaling and computational similarity. www.phys.uu.nl/~steen/vecp2k.html.

[129] Giulio Vignoli. *Focusing Inversion Techniques - Theory And Applications To Traveltime Tomography And Electrical Impedance Tomography*. PhD thesis, Ferrara University, 2005.

[130] Giulio Vignoli and Giuseppe Pagliara. Focusing inversion techniques applied to electrical resistance tomography in an experimental tank. *arXiv:physics*, 0606234:1–4, 2006.

[131] Giulio Vignoli and L. Zanzi. Focusing inversion technique applied to radar tomographic data. *arXiv:physics*, 0606243:1–4, 2006.

[132] Giulio Vignoli and Michael S. Zhdanov. Sharp boundary imaging in croswell seismic tomography. In *Proc. Ann. Mtg. Consortium for Electromagnetic Modeling and Inversion*, pages 155–172, 2004.

[133] Giulio Vignoli and Michael S. Zhdanov. Sharp boundary inversion in 3-d traveltime tomography. In *Proc. Ann. Mtg. Consortium for Electromagnetic Modeling and Inversion*, pages 229–243, 2005.

[134] Curtis R. Vogel. *Computational methods for inverse problems*, volume 10 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, PA, USA, 2002.

[135] Piotr Wendykier. *High Performance Java Software for Image Processing*. PhD thesis, James T. Laney School, 2009.

[136] Piotr Wendykier and James G. Nagy. Large-scale image deblurring in Java. In Marian Bubak, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *Proceedings of the 8th International Conference on Computational Science – ICCS 2008*, Kraków, Poland, 2008. Springer.

[137] Piotr Wendykier and James G. Nagy. Parallel COLT: A high-performance Java library for scientific computing and image processing. *ACM Transaction on Mathematical Software*, 37(3):1–22, September 2010.

[138] Stephen J. Wright, Robert D. Nowak, and Mário A. T. Figueiredo. Sparse reconstruction by separable approximation. *IEEE Trans. Signal Process.*, 57(7):2479–2493, 2009.

[139] Riccardo Zanella. *Scaled Gradient Projection methods for image and signal reconstruction*. PhD thesis, University of Modena and Reggio Emilia, 2009.

[140] Riccardo Zanella, Patrizia Boccacci, Luca Zanni, and Mario Bertero. Efficient gradient projection methods for edge-preserving removal of poisson noise. *Inverse Problems*, 25(4):045010, April 2009.

[141] Luca Zanni. An improved gradient projection-based decomposition technique for support vector machines. *Computational Management Science*, 3:131–145, 2006.

[142] Michael S. Zhdanov. *Geophysical Inverse Theory and Regularization Problems*. North-Holland/American Elsevier, Amsterdam, NL, 2002.

[143] Michael S. Zhdanov and Gabor Hursán. 3-d electromagnetic inversion based on quasi-analytical approximation. *Inverse Problems*, 16(5):1297–1322, October 2000.

[144] Michael S. Zhdanov and Ekaterina Tolstaya. Minimum support nonlinear parametrization in the solution of 3-d magnetotelluric inverse problems. *Inverse Problems*, 20:937–952, 2004.

[145] Michael S. Zhdanov, Giulio Vignoli, and T. Ueda. Sharp boundary inversion in crosswell travel-time tomography. *J. Geophysics and Engineering*, 3(2):122–134, 2006.

[146] Bin Zhou, Li Gao, and Yu-Hong Dai. Gradient methods with adaptive step-sizes. *Comput. Optim. Appl.*, 35(1):69–86, 2006.

Acknowledgements

My warmest thanks to the CINECA Supercomputing Center (Casalecchio di Reno, Italy), for having provided me with both the financial and the technical support during the three years of my Ph.D.: it has been an exciting experience, which would be wonderful to continue. But on top of this experience, my profound gratitude goes to Dr. Giovanni Erbacci, head of the CINECA's HPC Team: he trusted my research project and let my hope for study and research activities became the reality.

I'm extremely grateful also to Dr. Giulio Vignoli, currently assistant professor at the Earth Sciences Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, that involved me in his enthusiastic passion for the joint inversion idea when he was still a fellow of the "Mathematics for Technology" center of the University of Ferrara: the interesting discussion we had inspired my research work from the very beginning.

I would also like to thank Riccardo Zanella from University of Ferrara (Italy) who has been so helpful and willing to take some time to help me on SGP method, from the algorithm theory to the technical support during the implementation and the test stages.

I wish to thank the two referees, Prof. Hong Zhang and Prof. Aad van der Steen, for their many helpful suggestions and useful comments, that contributed to improve the quality of my work. A particular thank to Prof. Hong Zhang, for having given me the great opportunity to work with her for three months at the Argonne National Laboratory and also to cooperate with Lois Curfman McInnes. I really hope this collaboration can proceed in the future.

I would like to express my deep and sincere gratitude to my supervisor, Professor Gaetano Zanghirati, University of Ferrara (Italy). His wide knowledge and his logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for the present thesis.

All my colleagues deserve a loud mention: you have been great friends and psychologists for my academic (and not only) paranoia! I owe you many, many cakes!

My special gratitude is due to my parents Federico Giovannini and Nadia Robustini, to my brother Emanuele Giovannini, my fiancé Valerio Bellagamba, and their family for their loving support. Without their encouragement and understanding it would have been impossible for me to finish this work.