INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Algoritmos Distribuídos Para Problemas De Aproximação Esparsa

João Filipe de Castro Mota

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente: Prof. Carlos Jorge Ferreira Silvestre
Orientadores: Prof. João Manuel de Freitas Xavier
Prof. Pedro Manuel Quintas Aguiar
Vogal: Prof. Mário Alexandre Teles de Figueiredo

Setembro 2008

INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Distributed Algorithms For Sparse Approximation

João Filipe de Castro Mota

A Dissertation submitted in fulfillment of the requirements for the degree of Master of Science in:

Electrical and Computer Engineering

September 2008

Agradecimentos

Em primeiro lugar, gostaria de agradecer aos meus orientadores neste projecto, ao Prof. João Xavier e Prof. Pedro Aguiar, não só por me contagiarem com o entusiasmo que cada pequena descoberta científica desperta, mas também pelos valiosos comentários sobre questões técnicas fundamentais relacionadas com esta tese. Em particular, agradeço ao Prof. João Xavier as sucessivas e excelentes revisões da dissertação, bem como a disponibilidade para as fazer e tirar dúvidas. Também agradeço ao Prof. Mário Figueiredo os esclarecimentos prestados sobre o funcionamento do algoritmo GPSR. Gostaria de frisar que este projecto foi financiado em parte pela *Fundação para a Ciência Tecnologia*.

Ainda relacionado com a tese, gostaria de agradecer à Dragana por me ter dado a conhecer um artigo sobre o método de DQA, e que acabou por dar um rumo ao que haveria de ser esta tese. Ao colega e amigo Pedro Guerreiro gostaria também de dar uma palavra de agradecimento, não só pelo companheirismo, mas também pelas centenas, se não milhares, de conversas fascinantes, técnicas e não técnicas, que tivémos ao longo destes anos.

Aos amigos e familiares que sempre me apoiaram, mas cujo nome não mencionarei sob pena de me poder esquecer de alguém, deixo-lhes um obrigado muito sincero. Destaco a minha irmã Renata e o Tiago pelo apoio sempre presente.

Aos meus pais, pelas condições que sempre me proporcionaram ao longo da vida, dedico-lhes esta tese que, ainda que simbolicamente, representa o culminar de vários anos de estudo. Por isso, como sinal de gratidão, dedico-lhes este trabalho.

Finalmente, gostaria de agradecer à Filipa, não só o amor e carinho com que me prendou durante estes anos, mas também pela paciência e compreensão demonstradas aquando da realização desta dissertação, que muito tempo de atenção lhe roubou.

Resumo

Muitas aplicações requerem o conhecimento de uma combinação linear esparsa de sinais elementares que aproxima um dado sinal. Este problema é conhecido por "problema de aproximação esparsa" e surge em diversos contextos em ramos da Engenharia Electrotécnica e da Matemática Aplicada. A grande dificuldade em lidar com este tipo de problemas é a falta de convexidade ou a não–diferenciabilidade das medidas de esparsidade.

A grande contribuição desta tese é propôr, analisar e comparar algoritmos distribuídos que permitem resolver um problema de aproximação esparsa conhecido pelo nome de "basis pursuit". Há interesse considerável em resolver este problema de forma distribuída pela sua aplicabilidade em telecomunicações, computação e redes de sensores. Todos os algoritmos propostos baseiam–se na existência de uma partição da matriz que contém a descrição dos sinais elementares. Essa partição pode ser horizontal ou vertical e os respectivos blocos da matriz estão armazenados em diferentes processadores. Contudo, cada algoritmo, estando associado a uma determinada arquitectura de ligações entre os vários processadores disponíveis, opera sob essa arquitectura para resolver o "basis pursuit".

Também são apresentadas provas de convergência para todos os algoritmos propostos na tese. Com esta finalidade, as provas conhecidas de convergência para os métodos "Diagonal Quadratic Approximation" e "Nonlinear Gauss–Seidel" tiveram de ser generalizadas de modo a abranger uma nova classe de funções não–diferenciáveis. Essa nova classe é definida, e é-lhe dado o nome de "funcões rígidas".

Por último, são apresentados possíveis caminhos na futura investigação de algoritmos que resolvam o "basis pursuit" e uma sua versão mais robusta, "basis pursuit denoising", de forma centralizada, mas rápida.

Palavras Chave: Basis Pursuit, Algoritmos Distribuídos, Nonlinear Gauss–Seidel, Diagonal Quadratic Approximation, Método de Subgradiente, Funções Rígidas.

Abstract

Many applications require the knowledge of a sparse linear combination of elementary signals that can explain a given signal. This problem is known as the "sparse approximation problem" and arises in many fields of electrical engineering and applied mathematics. The great difficulty when dealing with sparse approximation problems is the lack of convexity or the non-differentiability inherent to the sparsity measures.

The main contribution of this thesis is the proposal, and in–depth analysis, of some distributed algorithms that solve a sparse convex approximation problem known as the "basis pursuit" problem. The interest in solving this problem distributedly concerns applications such as communications, computing and sensor networks. All the proposed algorithms assume that the matrix which contains the description of elementary signals is partitioned either horizontally or vertically among the several processors available. Nevertheless, each algorithm is based on a particular architecture for the links between the processors and must operate upon that architecture in order to solve the basis pursuit.

This thesis also provides proofs of convergence for all the proposed algorithms. This required extending the well-known results of convergence of the Diagonal Quadratic Approximation and the Nonlinear Gauss–Seidel methods to cover a new class of non-differentiable functions, which we call "rigid functions".

Finally, we also point out new possible directions in the research for centralized but fast algorithms to solve, not only the basis pursuit, but also the "basis pursuit denoising" problem.

Keywords: Basis Pursuit, Distributed Algorithms, Nonlinear Gauss–Seidel, Diagonal Quadratic Approximation, Subgradient Method, Rigid Functions.

Contents

1 Introduction **1**
1.1 Contributions . . . 2
1.2 Thesis Organization . . . 2
1.3 Preliminaries . . . 3
1.3.1 Why is sparsity usually associated with the ℓ_1-norm? . . . 3
1.3.2 BP as a Linear Program . . . 8
1.3.3 The Dual Of The BP . . . 8

2 Distributed BP With Horizontal Partition **11**
2.1 Subgradient Method . . . 12
2.1.1 Dual Approach For Solving The Bounded Basis Pursuit Problem . . . 13
2.1.2 Solving The Bounded BP Problem Through Its Dual . . . 17
2.1.3 The Resulting Algorithm . . . 22
2.2 Multiplier Methods . . . 29
2.2.1 Method of Multipliers As An External Loop . . . 29
2.2.2 Nonlinear Jacobi Approach . . . 31
2.2.3 Nonlinear Gauss–Seidel Approach . . . 39
2.3 Comparison Of The Methods . . . 45

3 Distributed BP With Vertical Partition **49**
3.1 Proposed Approach . . . 50
3.1.1 Application to the BPDN . . . 57

4 Fast Methods For BP and BPDN: Future Research Topics **59**
4.1 Ellipsoidal Approximation . . . 59
4.1.1 Ellipsoidal Approximation For The BP . . . 60
4.1.2 Ellipsoidal Approximation For The BPDN . . . 61
4.1.3 Generalization Of The BPDN . . . 64
4.1.4 The Quest For A “Perfect” Interior–Point Algorithm . . . 65
4.2 Ball Approximation . . . 66
4.3 Ellipsoidal And Ball Approximations In Generic ℓ_1-norm Problems . . . 68

5 Conclusions **70**

A Example of a non-rigid function **73**

B A Simple Subgradient Based Algorithm For Quadratic Programming **75**

C Quadratic Program Over An Ellipsoid **78**
C.1 Point Projection Over An Ellipsoid . . . 78
C.2 Strictly Convex Quadratic Program Over An Ellipsoid . . . 82

List of Figures

1.1 Graphics of the functions $|u|$, u^2 and $\mathrm{card}(u)$. 4
1.2 Alternative explanation of the sparsity inducing property of the ℓ_1-norm. 5
1.3 Graphical description of the constants C_0 and C_1, which appear in the theorems 1 and 2. 6
1.4 Plot of the functions $|x_i|$ and $a_i^T \lambda\, x_i$ for two situations. 9
1.5 Graphical depiction of the dual program (1.15). 10

2.1 Graphical interpretation of the solution of problem (2.10) 16
2.2 Architecture of the links between the processors for the subgradient method for bounded BP (algorithm 2). 23
2.3 Typical behavior of the inner product $a_i^T \lambda^k$ along the iterations of algorithm 2. 24
2.4 Filtering along the iterations of algorithm 2. 24
2.5 Evolution of the inner product $a_i^T \lambda^k$ along the iterations of the algorithm 2 when the correspondent component x_i^* is close to zero. 25
2.6 Example of a "false alarm" in the iterations of the algorithm 2. 25
2.7 Comparison between an exact solution of (2.1), x^*, and the solution returned by algorithm 2, $\hat{x}$. 26
2.8 Inner product $a_i^T \lambda^k$ along the iterations of the subgradient method, where a_i is the column that was wrongfully discarded in the experience of the Figure 2.7. 27
2.9 Evolution of the cost function $H(\lambda^k)$ along the iterations. 27
2.10 Same inner product $a_i^T \lambda^k$ of the Figure 2.8 (missing column), but where we used a value for the bound R of 1.48 instead of 3. 28
2.11 Required architecture for the implementation of the algorithm 7. 44

3.1 Graphical interpretation of the basis pursuit problem (3.1), where the matrix A is in $\mathbb{R}^{2\times 3}$. 49
3.2 Architecture needed for the implementation of algorithm 8; and illustration of the flow of information in its inner cycle (Nonlinear Gauss–Seidel step). 54
3.3 Histograms of the errors of each variable x_p, for $p = 1, \ldots, 5$, during the execution of algorithm 8. 57

4.1 Maximization of an inner product $(B^T b)^T x$ over the unit–radius sphere $\{x : \|x\| \leq 1\}$. . . 61
4.2 Difference between the solutions of the linear program (4.3) and its approximated problem (4.4). 62
4.3 Difference between the solutions of (4.9) and (4.10), *i.e.*, the projection of a point on a polyhedron and on the inscribed ellipsoid that best approximates that polyhedron. 63
4.4 Polyhedron $\mathcal{P}$, where the matrix $A \in \mathbb{R}^{2\times 50}$ is random. 67
4.5 Difference between the solutions of the approximated problem (4.22), represented by λ_a, and the initial problem (4.3). 68

A.1 Graphics with different shadings of the function $\max\left\{(x-1)^2+(y+1)^2, (x+1)^2+(y-1)^2\right\}$. 73

B.1 Comparison in terms of the relative error and time consumed of the solutions of the problem (B.2) given by the presented algorithm and by *Yalmip/Matlab*. 77

List of Tables

2.1 Different types of behavior of $|a_i^T \lambda^k|$ along the iterations of the subgradient method phase of algorithm 2. 25
2.2 Theoretical features of the algorithms presented in chapter 2. 46
2.3 Experimental results from the simulation of the SMBBP, the M/DQA and M/GS. 47

3.1 Illustration of the execution of 6 iterations of an inner cycle (Nonlinear Gauss–Seidel) of the algorithm 8, for $P = 4$. 56
3.2 Theoretical features of algorithm 8. 57

Acronyms

BP	Basis pursuit
BPDN	Basis pursuit denoising
RBP	Reduced basis pursuit
DQA	Diagonal quadratic approximation
SMBBP	Subgradient method for bounded basis pursuit
M/DQA	Multipliers/Diagonal quadratic approximation
M/GS	Multipliers/Gauss–Seidel

Chapter 1

Introduction

Over the last half century, society has benefited a lot from the significant advances in signal processing. Ranging from image and audio processing to genetics and bioengineering, its applications are numerous. So, it is no surprise that developments on these application areas can be observed almost everyday. Nevertheless, the field of signal processing itself is far from being static. The topic of sparse representation of signals is a quite recent and promising one [27, 28].

Sparse representation arises in so many fields that it is impossible to mention all here. Examples include signal reconstruction and restoration, denoising, regularization, estimation and fitting, interpolation, channel coding and compressed sensing. It is worthwhile to mention that compressed sensing provides a more efficient alternative to the Shannon-Nyquist sampling for a broad class of signals [16, 2, 12, 7]. However, most of the theoretical results found in the area of sparse representation are relatively recent and, consequently, advances in its related areas of application are expected in the near future.

At the same time, the computing architecture paradigm is changing towards more distributed environments, where computational resources are not just concentrated in a single place but over many. Sensor networks are just an example, having (real-time) applications such as security, surveillance, smart classroom, robotics, aerospace and medical monitoring [15, 19]. While the hardware for parallel computing has had great developments, there is still lot of space for advances in the parallelization of algorithms. A parallelizable algorithm has always a certain hardware architecture associated, and may not be suitable for other architectures. Therefore, it is of considerable interest to develop algorithms that can be used over several architectures.

The main difficulty encountered when dealing with sparse representation problems is the non-convexity or the non-differentiability of the sparsity measures. In this thesis, we are interested in two important problems that arise in the context of sparse representation:

$$\begin{array}{cc} \underset{\substack{Ax=b \\ \mathrm{var}\,:\, x\in\mathbb{R}^n}}{\min} & \|x\|_1 \end{array} \tag{BP}$$

and

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-b\|^2 + \beta\|x\|_1. \tag{BPDN}$$

Although both problems are convex, it will be shown in subsection 1.3.1 that (BP) is a convex relaxation of a combinatorial problem involving the ℓ_0-quasi norm; and (BPDN) is a more robust way of solving (BP).

Problem (BP) is known as the *basis pursuit* problem. It consists in finding a vector $x \in \mathbb{R}^n$ that has

the least ℓ_1-norm ($\|x\|_1 := \sum_i |x_i|$) and at the same time verifies the linear equation $Ax = b$. A is a real–valued matrix $m \times n$ and b is a vector in $\mathbb{R}^m$. We can assume, without loss of generality, that the rows of A are linearly independent. Thus, this problem only makes sense when $m < n$, *i.e.*, there are more unknowns than equations, being $Ax = b$ an underdetermined system. When $m = n$, the only feasible point is $x = A^{-1}b$.

A closely related problem is (BPDN), which is called *basis pursuit denoising* [13]. In fact, this is an heuristic to find a vector $x \in \mathbb{R}^n$ that is both sparse and close to verify equation $Ax = b$, where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In this thesis, when we write $\| \cdot \|$ we always refer to the ℓ_2-norm, unless otherwise stated. So, in (BPDN), we try to find a solution such that $Ax - b$ is small in the ℓ_2-norm sense, and x is small in the ℓ_1-norm sense. The number β is the trade-off parameter between the two objectives that we are trying to minimize. Unlike (BP), this problem makes sense for any relationship between the dimensions of A, m and n.

1.1 Contributions

Problems (BP) and (BPDN) belong to well–established classes of optimization problems. While (BP) can be written as a linear program, (BPDN) can be recast as a quadratic program. However, when one of the requirements is to solve these problems in a distributed/parallelized way, there are no references in the literature, to the best of our knowledge, of algorithms that fulfill this requirement.

In this thesis, we tackle the problem of solving the BP in a distributed environment and take for granted the fact that no single processor knows the entire matrix A. We consider both the cases where this matrix is partitioned horizontally and vertically. In the first case, we propose, analyze and compare three algorithms that differ, not only in the approaches, but also in the supporting architecture for the links between the processors. In the second case, we propose and analyze just one algorithm, although another one is possible but more inefficient, that relies on a link architecture different from those found in the algorithms that are based on a horizontal partition of A. Still in the latter case, we show how to easily adapt that algorithm to solve the BPDN.

Concerning the technical aspects of this work, we give for each algorithm a proof of convergence and define its conditions of applicability. The main difficulty in solving the BP and the BPDN in a distributed/parallelized way is the lack of guarantees of convergence, for non–differentiable functions, of well–known methods that induce parallelization. We extend the proofs of convergence of such methods (Diagonal Quadratic Approximation and Nonlinear Gauss–Seidel algorithms) to include a new kind of non–differentiable functions: the “rigid functions”. The concept of rigid functions is defined in the text as being part of the class of subdifferentiable functions that share a particular property with the differentiable functions.

Throughout this thesis, we also propose several possible topics of research in the area of algorithms that solve the BP and the BPDN. The case where centralized but fast algorithms are required is also addressed, bur we only emphasize details that can lead to new efficient algorithms.

1.2 Thesis Organization

In section 1.3, we present some useful material for the comprehension of this thesis, concerning not only recent theoretical results, but also some problem analysis.

In chapter 2, we tackle the problem of solving the BP with an horizontal partition of the matrix A. Three different algorithms are proposed, analyzed and compared.

Chapter 3 concerns the case when the matrix A is partitioned vertically for the resolution of both the BP and the BPDN.

Finally, in chapter 4, future research topics are proposed concerning the development of centralized fast algorithms to solve either the BP or the BPDN.

1.3 Preliminaries

This section provides some background, some analysis and some recent theoretical results on the basis pursuit (BP) and basis pursuit denoising (BPDN) problems. We will show that BP is a linear program as well as interpret its dual, thus collecting information for the analysis of the proposed algorithms. Furthermore, we will also explain the relationship between BP and BPDN, and the reason why these two problems are usually related to sparse approximation.

1.3.1 Why is sparsity usually associated with the ℓ_1-norm?

The basis pursuit problem

$$\begin{array}{cc} \min\limits_{\substack{Ax=b \\ \text{var}\,:\, x\in\mathbb{R}^n}} & \|x\|_1, \end{array} \tag{1.1}$$

where the data are $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, and the variable is $x \in \mathbb{R}^n$, is considered a *least–norm problem*

$$\begin{array}{cc} \min\limits_{\substack{Ax=b \\ \text{var}\,:\, x\in\mathbb{R}^n}} & \|x\|, \end{array} \tag{1.2}$$

where $\|x\|$ is any norm. Nevertheless, we will just consider three types of the well–known ℓ_p-norms ($p = 0$, $p = 1$ and $p = 2$). The ℓ_p-norm of a vector $x \in \mathbb{R}^n$ is defined by

$$\|x\|_p := \begin{cases} \left(\sum_{i=1}^n |x_i|^p\right)^{1/p} & ,\text{if} \quad 0 < p < +\infty \\ |\{i\,:\,x_i \neq 0\}| & ,\text{if} \quad p = 0 \\ \max_{i=1,\ldots,n} |x_i| & ,\text{if} \quad p = +\infty \end{cases},$$

where $|S|$ is the cardinality of a finite set S. Strictly speaking, $\|\cdot\|_p$ is a norm only for $p \geq 1$.

We can interpret problem (1.2) as a way of solving an ill-posed linear equation $Ax = b$, and finding its smallest solution (measured in the respective norm). When the chosen norm is the ℓ_2-norm, (1.2) has a unique solution called the *least–squares solution*, which can be easily obtained. However, the *least-squares solution* is rarely sparse[1]. On the other hand, if we use the ℓ_1-norm we will get sparse solutions very often.

Let's examine why this happens [6, page 296]. Interpreting the objective function $\|x\|$ on (1.2) as a penalizing term, we seek the least penalized solution. As problem (1.2) is equivalent to

$$\begin{array}{cc} \min\limits_{\substack{Ax=b \\ \text{var}\,:\, x\in\mathbb{R}^n}} & \|x\|^2, \end{array}$$

[1] By a sparse vector we mean a vector that has few non–zero entries.

making $x = (x_1, x_2, \ldots, x_n)$ and using the ℓ_2-norm we get

$$\begin{array}{ll} \min & x_1^2 + x_2^2 + \ldots + x_n^2. \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{1.3}$$

In its turn, the basis pursuit (1.1) can be written as

$$\begin{array}{ll} \min & |x_1| + |x_2| + \ldots + |x_n|. \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{1.4}$$

Therefore, we can restrict our analysis to $\mathbb{R}$, by comparing the functions $\phi_1, \phi_2 : \mathbb{R} \to \mathbb{R}$, $\phi_1(u) = |u|$ and $\phi_2(u) = u^2$. When $|u| = 1$, $\phi_1(u) = \phi_2(u) = 1$. When $|u| \gg 1$, we have $\phi_1(u) \ll \phi_2(u)$, meaning that the large values of each component of x are much more penalized using the ℓ_2-norm than using the ℓ_1-norm. Finally, when $|u| \ll 1$, we get $\phi_1(u) \gg \phi_2(u)$, so the ℓ_1-norm puts much more emphasis on small values of the components of x than the ℓ_2-norm (see Figure 1.1). As a consequence, the solution

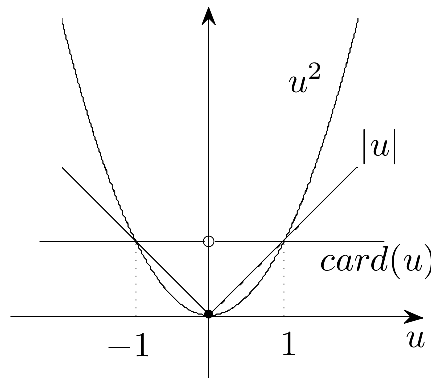


Figure 1.1: Graphics of the functions $|u|$, u^2 and $\operatorname{card}(u)$ (cardinality of the vector u, which is equivalent to the ℓ_0-quasi norm in $\mathbb{R}$) in order to illustrate the difference in the penalization of small and large components of u.

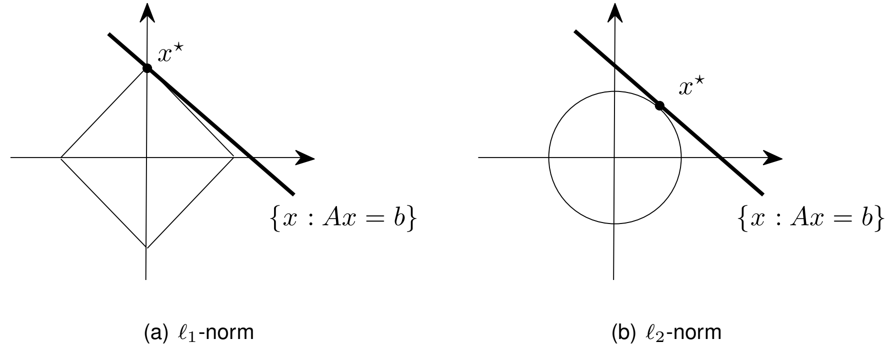
of (1.4) has more small or near zero components and perhaps a few large non–zero components, than the solution of problem (1.3).

Figure 1.2 gives another popular explanation (in the plane) why the ℓ_1-norm is preferred to the ℓ_2-norm in order to generate sparse vectors. The optimal solution in both cases is the first point of the corresponding norm ball to "touch" the set $\{x : Ax = b\}$, as if we were swelling up the ball. It happens that the ℓ_1-norm ball is larger on the coordinate axis directions than in the other directions, while the ℓ_2-norm ball is isotropic.

So if we seek a solution with lots of small (near–zero) components, the ℓ_1-norm is a good choice.

Actually, any (*quasi*) ℓ_p-norm with $0 \leq p \leq 1$, would lead to the same results. But from these (*quasi*) norms, the only convex one is the ℓ_1-norm ($p = 1$) — recall that non-convex problems are the hardest to solve. Nevertheless, as the cardinality of a vector is measured by the ℓ_0-quasi norm, the *sparsest* vector that solves the linear equation $Ax = b$ is the solution of (1.2) with the ℓ_0-quasi norm

$$\begin{array}{ll} \min & \|x\|_0. \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{1.5}$$



(a) ℓ_1-norm

(b) ℓ_2-norm

Figure 1.2: Alternative explanation of the sparsity inducing property of the ℓ_1-norm.

It is interesting to note that problem (1.1) is actually a convex relaxation of the problem (1.5). In Figure 1.1, we can see that the ℓ_p-norm that best approximates the ℓ_0-quasi norm near the origin, and is convex, is the ℓ_1-norm. This approximation can reveal very useful in the sense that we replace a combinatorial problem by a linear program (see subsection 1.3.2) — the simpler class of optimization problems.

Even more surprising is the fact that there are results that state that problems (1.1) and (1.5) are equivalent under certain conditions. To understand when this equivalence occurs, we need to use the concept of *restricted isometry constants*, introduced in [10] and refined in [11, 8].

Definition 1 (Restricted isometry constants)**.** *For each $s = 1, 2, \ldots, n$, the restricted isometry constant $\delta_s(A)$ of a matrix $A \in \mathbb{R}^{m\times n}$ is the smallest number such that*

$$(1 - \delta_s(A))\|x\|^2 \leq \|Ax\|^2 \leq (1 + \delta_s(A))\|x\|^2, \tag{1.6}$$

holds for all s-sparse vectors[2]. Similarly, the s, s'–restricted orthogonality constant $\theta_{s,s'}(A)$ is defined for $s + s' = 1, 2, \ldots, n$ as the smallest number such that

$$| < Ax, Ax' > | \leq \theta_{s,s'}(A) \cdot \|x\|\|x'\|, \tag{1.7}$$

holds for all s-sparse vector x and all s'-sparse vector x', where $< a, b >:= \sum_{i=1}^n a_i b_i$ for $a, b \in \mathbb{R}^n$.

The numbers $\delta_s(A)$ and $\theta_{s,s'}(A)$ measure how close any s-sparse collection of the columns of the matrix A behaves like an orthonormal system. In particular, the smaller these constants are, the closer this collection of columns is to an orthonormal system.

To see how important the restricted isometry constants are, we can see, for instance, that if $\delta_{2s}(A) < 1$, problem (1.5) has a unique s-sparse solution. Instead of proving this claim, let's see what happens when $\delta_{2s}(A) = 1$. In this case, there exists a vector $2s$-sparse h such that $Ah = 0$. We can then decompose h as $x - x'$, where both x and x' are both s-sparse, meaning that $Ax = Ax'$. This equation tells that if x is a solution of (1.2), so x' is.

Imagine a communication setting[3] in which the sender wishes to communicate a vector x to the receiver, but, instead of sending x (a large vector of size n), it just sends the vector b of size $m < n$, satisfying $Ax = b$. Matrix A is known by the receiver. Furthermore, the receiver knows that the vector x is s-sparse. In this case, if the matrix A has a restricted isometry constant $\delta_{2s}(A) < \sqrt{2} - 1$, the receiver can solve either problem (1.1) or problem (1.5), since both problems have the same solution. That is,

[2]A vector is s-sparse if it has at most s non–zero entries.

[3]It could be any situation that consisted in finding a sparse solution of an ill–posed linear system. For more situations see, for example, [12, 27, 2].

the convex relaxation of problem (1.5) (which is (1.1)) is exact. A more generic result, in which x is not necessarily s-sparse but completely generic, involves the *best sparse approximation* that one could obtain if one knew exactly the locations and the amplitudes of the s–largest absolute entries of x. This result is in [8] and is given by

Theorem 1. *Let x be an arbitrary vector in $\mathbb{R}^n$; A a real matrix m by n with a restricted isometry constant $\delta_{2s}(A) < \sqrt{2} - 1$; and b a vector in $\mathbb{R}^m$ such that $b = Ax$. Denote the vector which is equal to x only at the s–largest absolute entries and zero elsewhere by x_s. Then the solution $x^\star$ to* (1.1) *obeys*

$$\|x^\star - x\| \leq C_0 s^{-1/2} \|x - x_s\|_1,$$

where C_0 is a constant depending on $\delta_{2s}(A)$. In particular, if x is s-sparse, the error is zero.

We add that this theorem is only useful for values of $\delta_{2s}(A)$ smaller than about 0.4; and that the constant C_0 is rather small for $\delta_{2s}(A) < 0.3$ (see Figure 1.3). Similar results, with different conditions,

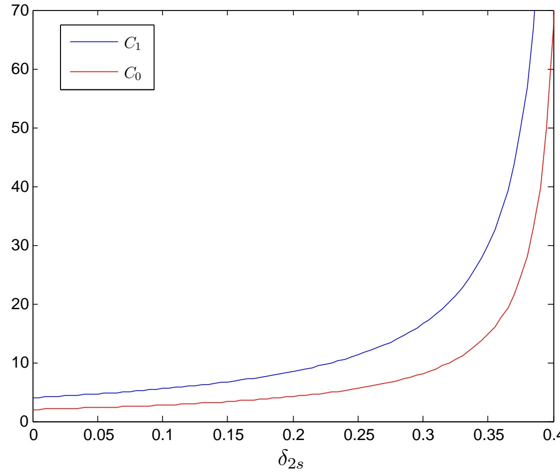


Figure 1.3: Graphical description of the constants C_0 and C_1, which appear in theorems 1 and 2, as a function of $\delta_{2s}(A)$. C_0 given by $C_0 = 2(1+\rho)/(1-\rho)$, where $\rho = \sqrt{2}\delta_{2s}(A)/(1-\delta_{2s}(A))$; and C_1 is given by $C_1 = 2\alpha/(1-\rho)$, where $\alpha = 2\sqrt{1+\delta_{2s}(A)}/(1-\delta_{2s}(A))$.

are proved in [11, 9], and also explained in [7]. The same articles also refer which kind of matrices have "good" restricted isometry constants, namely Gaussian random matrices with i.i.d. entries, Fourier ensembles and general orthogonal measurement ensembles. Still covering this issue, we mention [1]. This analysis would not be complete without mentioning the articles [27, 26], which provide a different analysis for the same problem.

Concerning the practical implications of this theory, [10, 2] state that we can code without loss (with an overwhelming probability) all the information of a s-sparse vector x of size n into a vector b of size m, with $m = \mathcal{O}(s \log(n/s))$, just by doing $b = Ax$, where A is a random matrix of the kind of the ones that were mentioned above.

The Basis Pursuit Denoising

Here, it will be explained the role of the basis pursuit denoising (BPDN) problem in the context of sparse approximation. Some recent theoretical results will also be presented.

Imagine again that we are in a communication setting, as explained in page 5, but now the signal b, known by the receiver, is contaminated with noise: $b = Ax + z$, where z is any perturbation for which we

know a magnitude upper–bound ϵ, *i.e.* $\|z\| \leq \epsilon$. If we know *a priori* that x is sparse, it makes sense to solve a slightly different version of the *basis pursuit*

$$\begin{array}{cc} \min & \|x\|_1. \\ \|Ax - b\| \leq \epsilon & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{1.8}$$

This problem, however, is not equivalent to a linear program anymore. It belongs to the class of the *second–order cone programs* [6, page 156]. The next theorem, proved in [8], shows that by solving problem (1.8) the receiver can reconstruct x in a stable way.

Theorem 2. *Let x be an arbitrary vector in $\mathbb{R}^n$; A a real matrix m by n with a restricted isometry constant $\delta_{2s}(A) < \sqrt{2} - 1$; and b a vector in $\mathbb{R}^m$ such that $b = Ax + z$, being z arbitrary noise, but bounded by ϵ,* i.e. *$\|z\| \leq \epsilon$. Denote the vector which is equal to x only at the s–largest absolute entries and zero elsewhere by x_s. Then the solution $x^\star$ to* (1.8) *obeys*

$$\|x^\star - x\| \leq C_0 s^{-1/2} \|x - x_s\|_1 + C_1 \epsilon,$$

where C_0 is the same constant of theorem 1 and C_1 is another constant that also only depends on $\delta_{2s}(A)$. In particular, if x is s-sparse, the error is only bounded by $C_1 \epsilon$.

The two constants C_0 and C_1 are plotted in Figure 1.3. Again, we note that theorems 1 and 2 only have practical interest for values of $\delta_{2s}(A)$ smaller than about 0.4 or less, depending on the desired accuracy.

We still need to know how to relate problem (1.8) to (BPDN). The KKT-system [6, page 243] for (1.8) will be used to prove that a solution of (1.8), which always exists according to Weierstrass's theorem, is either $x = 0$ or a solution of (BPDN), for some $\beta > 0$.

The constraint of (1.8) can be equivalently written as $1/2\|Ax - b\|^2 \leq 1/2\epsilon^2$. This way, the Lagrangian function $L : \mathbb{R}^n \times \mathbb{R}$ of (1.8) is

$$L(x, \mu) = \|x\|_1 + \frac{\mu}{2}\|Ax - b\|^2 - \frac{\mu}{2}\epsilon^2$$

and the corresponding KKT-system is

$$\left\{ \begin{array}{l} x^\star \in \arg\min_x (\mu^\star/2)\|Ax - b\|^2 + \|x\|_1 - (\mu^\star/2)\epsilon^2 \\ \|Ax^\star - b\| \leq \epsilon \\ \mu^\star \geq 0 \\ \mu^\star \left(\|Ax^\star - b\| - \epsilon \right) = 0 \end{array} \right. , \tag{1.9}$$

where $x^\star$ and $\mu^\star$ are the optimal primal and dual variables, respectively.

If $\mu^\star > 0$, the first equation of (1.9) is equivalent to

$$x^\star \in \arg\min_x \frac{1}{2}\|Ax - b\|^2 + \frac{1}{\mu^\star}\|x\|_1. \tag{1.10}$$

In this case, we have $\|Ax^\star - b\| = \epsilon$, from the last equation of (1.9). We also have that a solution of (1.8), provided that it exists and strong duality holds[4], is also a solution of (BPDN), for $\beta = 1/\mu^\star$.

On the other hand, if $\mu^\star = 0$, then the optimal primal variable minimizes $\|x\|_1$, due to the first equation of (1.9). So, the only solution is $x^\star = 0$.

[4]It suffices that there exists at least a vector $x \in \mathbb{R}^n$ such that $\|Ax - b\| < \epsilon$ (*Slater's condition*). In our case, as we are assuming that the rows of A are linearly independent, there exists always an $\tilde{x}$ such that $A\tilde{x} = b$, thus strong duality is always satisfied and (1.8) is always feasible.

As it is difficult to find the optimal dual variable $\mu^\star$ of (1.9), what is done in practice if one wants to solve (1.8) through the BPDN, is to solve (BPDN) for several values of β and choose the solution with the desired sparsity.

1.3.2 BP as a Linear Program

Using standard techniques in optimization [6], we can reformulate the basis pursuit (BP) as

$$\begin{array}{cc} \min\limits_{\substack{-t \le x \le t \\ Ax = b \\ \text{var:}(x,t) \in \mathbb{R}^n \times \mathbb{R}^n}} & 1_n^T t, \end{array} \tag{1.11}$$

which is clearly a linear program. 1_n stands for the column vector of size n with all its entries equal to one. Notice that the size of the variable is now $2n$.

1.3.3 The Dual Of The BP

In this subsection we will derive the dual of the basis pursuit problem

$$p^\star = \min_{\substack{Ax = b \\ \text{var}\,:\, x \in \mathbb{R}^n}} \|x\|_1. \tag{1.12}$$

The dual program of an optimization problem provides a lower bound $d^\star$ to the optimal value of the primal problem $p^\star$, *i.e*, $d^\star \le p^\star$. However, if the primal program is convex and verifies some constraint qualification, *e.g. Slater's condition* [6, page 226], strong duality holds, meaning that $d^\star = p^\star$. Moreover, when strong duality holds, by solving the KKT-system, one can find simultaneously the optimal primal and dual variables, thus the solution of the optimization problem.

It is easy to see that (1.12) is convex and satisfies *Slater's condition* (its constraints are affine), so solving its dual will reveal profitable if we want to get to its optimal solution. By representing the Lagrangian function by $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ (a notation used throughout the text), the steps to get to the dual program are:

1. $L(x, \lambda) = \|x\|_1 + \lambda^T b - \lambda^T A x$

2.

$$L(\lambda) = \inf_x \left[\|x\|_1 - \left(A^T \lambda\right)^T x \right] + \lambda^T b \tag{1.13}$$

$$= \sum_{i=1}^{n} \left[\inf_{x_i} |x_i| - a_i^T \lambda\, x_i \right] + \lambda^T b \tag{1.14}$$

We designated each column of the matrix A by a_i for $i = 1, \ldots, n$, *i.e.*,

$$A = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix},$$

thus, we can write

$$A^T\lambda = \begin{bmatrix} a_1^T\lambda \\ a_2^T\lambda \\ \vdots \\ a_n^T\lambda \end{bmatrix}.$$

This, together with the decomposition $x = (x_1, x_2, \ldots, x_n)$ justifies the passage from (1.13) to (1.14). Now, we have to evaluate the infimum of the function $|x_i| - a_i^T\lambda\, x_i$, for each $i = 1, \ldots, n$. Based on the plots of the functions $|x_i|$ and $a_i^T\lambda\, x_i$, which are in Figure 1.4, we can see that the infimum is only finite when $|a_i^T\lambda| \leq 1$, being zero in that case. Note that

$$|a_i^T\lambda| \leq 1, \quad \forall_{i=1,\ldots,n} \qquad \Longleftrightarrow \qquad \|A^T\lambda\|_\infty \leq 1.$$

Therefore, we can write

$$L(\lambda) = \begin{cases} \lambda^T b & , \quad \|A^T\lambda\|_\infty \leq 1 \\ -\infty & , \quad \text{otherwise} \end{cases}.$$

3. Finally, the dual program is

$$d^\star = p^\star = \max_\lambda \begin{cases} \lambda^T b & , \quad \|A^T\lambda\|_\infty \leq 1 \\ -\infty & , \quad \text{otherwise} \end{cases},$$

which is equivalent to

$$d^\star = \max_{\substack{\|A^T\lambda\|_\infty \leq 1 \\ \text{var}\,:\,\lambda \in \mathbb{R}^m}} \lambda^T b. \tag{1.15}$$

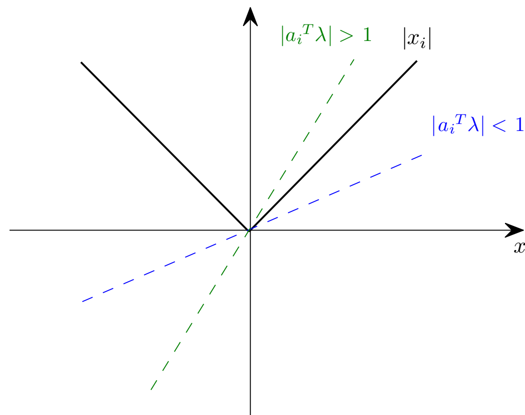


Figure 1.4: Plot of the functions $|x_i|$ and $a_i^T\lambda\, x_i$ for two situations: when the absolute value of the slope of the line $a_i^T\lambda\, x_i$ is greater than one, $|a_i^T\lambda| > 1$, the infimum of the function $|x_i| - a_i^T\lambda\, x_i$ is $-\infty$; when $|a_i^T\lambda| \leq 1$, the infimum is 0, being the origin a minimizer, *i.e.* $x_i^\star = 0$.

As expected, the dual of the BP (1.12) — which is itself equivalent to a linear program — is a linear program. Note that the corresponding linear programs are a little bit different in the size of the variables and in the number of constraints: while (1.11) has $2n + m$ linear constraints and the size of its variable is $2n$, (1.15) has $2n$ linear constraints and the size of its variable is only m.

In Figure 1.5, which represents graphically problem (1.15), we are considering that the polyhedron $\mathcal{P} = \{\lambda : \|A^T\lambda\|_\infty \leq 1\}$ is bounded. In fact, it can be shown that the linear independence of the rows

of A implies that $\mathcal{P}$ is bounded.

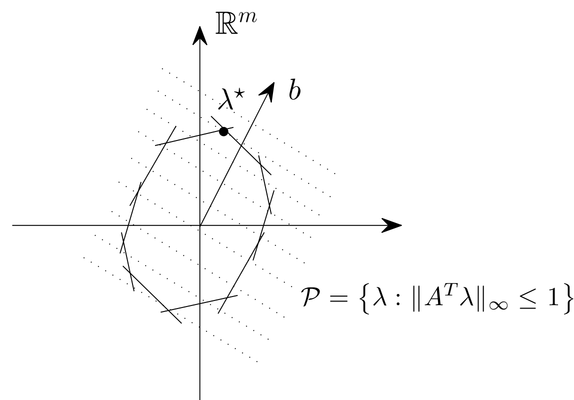


Figure 1.5: Graphical depiction of the linear program (1.15). The dashed lines represent the level curves of the objective function. One of the optimal points is always a vertex of the polyhedron.

This set $\mathcal{P}$ represents n inequalities of the form $|a_i^T\lambda| \leq 1$, for $i = 1, \ldots, n$, which are actually $2n$ linear inequalities. It is important to note that if we used any optimization method that minimizes a constrained optimization program based on projections on the constraining set (for instance, gradient [3, page 223] and subgradient [5] projection methods), it would be cumbersome to project a point on the set $\mathcal{P}$, just because this set is characterized by too many constraints $(2n)$ — recall that (BP) only becomes interesting for large values of n.

There are several methods that solve large-scale linear programs, such as simplex, interior-point or affine scaling methods. However, these methods are not readily adapted for a multi-processor/distributed environment.

Chapter 2

Distributed BP With Horizontal Partition

In this chapter, we propose, analyze and compare some algorithms for solving the basis pursuit (BP) problem

$$\begin{array}{ll} \min\limits_{\substack{Ax=b \\ \text{var}: x \in \mathbb{R}^n}} & \|x\|_1, \end{array} \tag{2.1}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $\|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|$ is the ℓ_1-norm. The variable is x, whereas A and b are given. We have already seen that (2.1) can be cast as a linear program in subsection 1.3.2. However, in this chapter, we consider the case where the matrix $A \in \mathbb{R}^{m \times n}$ is not concentrated in a single computer, thus standard methods for solving linear programs cannot be readily applied. This can happen if, for example, the dimensions of A are too large, or just because it's not practical to have A stored on a single computer.

In this chapter we assume that the matrix A is partitioned horizontally. Recall that A can be seen as an over–complete dictionary, perhaps integrating many families of functions. Each column of A, then, may represent a function. So, an horizontal partition makes sense when we want to operate each family of functions on different computers, for example to adapt each computer architecture to a particular family of functions.

We will then use several computers (or devices with some memory and computing power, which will be called processors, computers or nodes indistinctly) to try to solve problem (2.1).

Nowadays, distributed systems are becoming more and more important in applications such as communications, computing, or even sensor networks [4, 25, 19]. So, it shouldn't be too difficult to find an environment in applications where we have available several processors that can communicate with each other relatively fast.

To formalize, we assume that the matrix A is stored among the P computers available, being partitioned horizontally

$$A = \underbrace{\begin{bmatrix} A_1 & \cdots & A_p & \cdots & A_P \end{bmatrix}}_{n} \updownarrow m$$

into P blocks, each with n_p columns for $p = 1, \ldots, P$, throughout the rest of this chapter. As each block A_p is stored on a single processor, we are assuming that no processor knows the entire matrix A. Nevertheless, the processors have to cooperate to get to an optimal solution of (2.1).

We also assume that each block A_p contains adjacent columns of A, only for convenience of notation. In any case, it is always possible to re-order the columns so that this happens.

2.1 Subgradient Method

Here, we will try to solve (2.1) through its dual

$$\begin{array}{cl} \max & \lambda^T b, \\ \|A^T\lambda\|_\infty \leq 1 & \\ \text{var} : \lambda \in \mathbb{R}^m & \end{array} \tag{2.2}$$

nonetheless using a "trick" to get rid of the polyhedral constraint $\|A^T\lambda\|_\infty \leq 1$, which represents $2n$ linear inequalities. As we have already stated in subsection 1.3.3, the main difficulty here is the presence of the polyhedral constraints. The use of methods that project points on a polyhedron with many equations becomes inefficient, therefore it sounds a good idea to get rid of the constraint $\|A^T\lambda\|_\infty \leq 1$.

In fact, that constraint comes up when it is imposed that the dual function $L(\lambda)$ (on page 8, equation (1.14)) has a finite infimum. There are two ways of guaranteeing this without imposing restrictions on the variable λ:

- By transforming the dual Lagrangian function into a coercive function[1] with respect to the primal variable, for example, by adding a strictly convex quadratic term. Recall that a continuous coercive function has always a finite infimum (this method will be approached later);
- By guaranteeing that the primal constraint set is compact, and making use of Weierstrass's theorem.

This second way can be followed if a bound for the optimal solutions of (2.1) is known. Namely, we assume that there exists a sufficiently large R such that x^* belongs to the interior of $B_\infty(0, R)$, where $B_\infty(c, R) = \{x \,:\, \|x - c\|_\infty \leq R\}$ stands for the ℓ_∞ ball centered at c with radius R, and x^* denotes any solution of (2.1). Indeed, any norm for the ball would fit, but for our purposes the ℓ_∞ is the more adequate one, and simply means that the absolute value of any component of $x^\star$ cannot be larger than R.

In general, the introduction of the constraint $x^\star \in B_\infty(0, R)$ in (2.1) increases its optimal value but, provided that R is large enough, (2.1) is equivalent to the *bounded basis pursuit*

$$\begin{array}{cl} \min & \|x\|_1. \\ Ax = b & \\ x \in B_\infty(0, R) & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{2.3}$$

Note that one can easily find an R such that all the optimal solutions of (2.1) are in the interior of $B(0, R)$: take any point $\hat{x} \in \mathbb{R}^n$ that satisfies $A\hat{x} = b$; then, R can be equal to $n\|\hat{x}\|_\infty + \epsilon$, where $\epsilon > 0$ is some small number. This can be seen from the identity: $\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$, for any $x \in \mathbb{R}^n$. Indeed, if $\hat{x}$ is a feasible point of (2.1) and x^* any of its optimal solutions, we have

$$\begin{aligned} \|x^*\|_\infty &\leq \|x^*\|_1 \\ &\leq \|\hat{x}\|_1 \\ &\leq n\|\hat{x}\|_\infty. \end{aligned}$$

[1] We say that a function $f : \mathbb{R}^n \to \mathbb{R}$ is *coercive* if $\lim_{\|x\| \to +\infty} f(x) = +\infty$.

2.1.1 Dual Approach For Solving The Bounded Basis Pursuit Problem

By partitioning the variable x into P variables, $x = (x_1, x_2, \dots, x_P)$, and by using the convention for A on page 11, (2.3) is equivalent to

$$p^\star = \min_{\substack{A_1x_1 + A_2x_2 + \dots + A_Px_P = b \\ \|x_p\|_\infty \leq R, \qquad p = 1,2,\dots,P}} \|x_1\|_1 + \|x_2\|_1 + \dots + \|x_P\|_1. \tag{2.4}$$

Let's now derive the dual program of (2.4) by dualizing only the equality constraint:

1. The Lagrangian function $L : \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \dots \times \mathbb{R}^{n_P} \times \mathbb{R}^m \longrightarrow \mathbb{R}$ is given by

$$\begin{aligned} L(x_1, x_2, \dots, x_P, \lambda) &= \sum_{p=1}^{P} \|x_p\|_1 + \lambda^T b - \lambda^T \left(\sum_{p=1}^{P} A_p x_p \right) \\ &= \sum_{p=1}^{P} \left(\|x_p\|_1 - \lambda^T A_p x_p \right) + \lambda^T b \end{aligned}$$

2. The corresponding dual function is

$$\begin{aligned} L(\lambda) &= \inf_{\substack{\|x_p\|_\infty \leq R \\ p = 1,2,\dots,P}} L(x_1, x_2, \dots, x_P, \lambda) \\ &= \sum_{p=1}^{P} \left[\inf_{\|x_p\|_\infty \leq R} \left(\|x_p\|_1 - \lambda^T A_p x_p \right) \right] + \lambda^T b \end{aligned} \tag{2.5}$$

3. Note that, under the assumption that R is large enough and A is full–rank, *Slater's condition* holds for (2.3) for any vector b, hence also strong duality:

$$p^\star = d^\star = \max_{\lambda \in \mathbb{R}^m} L(\lambda) = - \min_{\lambda \in \mathbb{R}^m} -L(\lambda).$$

Let $H(\lambda)$ be equal to $-L(\lambda)$. Now, the goal is to solve

$$\min_{\lambda \in \mathbb{R}^m} H(\lambda), \tag{2.6}$$

where

$$H(\lambda) = -\lambda^T b - \sum_{p=1}^{P} \left[\inf_{\|x_p\|_\infty \leq R} \left(\|x_p\|_1 - \lambda^T A_p x_p \right) \right]. \tag{2.7}$$

Here, we note that $H(\lambda)$ can be written as a supremum of functions. Let be

$$r(\lambda, x) = -\lambda^T b + \lambda^T A x - \|x\|_1. \tag{2.8}$$

Then,

$$H(\lambda) = \sup_{\|x\|_\infty \leq R} r(\lambda, x).$$

In this equation, x must be seen as indexing the family of functions $r(\cdot, x)$. As $H(\lambda)$ is the pointwise supremum of convex differentiable functions it is convex but it may not be differentiable at every point. However, (2.6) can be solved using a non-descent algorithm called *subgradient method*[2] [5]. The only

[2]As the subgradient method is an iterative method, we should use, from now on, the notation λ^k to make clear that every

requirement for the function $H(\lambda)$ is to be subdifferentiable. It is known [23, Theorem 3.1.13] that if a function is closed and convex at an interior point of its domain, then it is subdifferentiable at that point. In fact, $H(\lambda)$ is closed on all $\mathbb{R}^m$, since it is a continuous and convex function. We can conclude, then, that $H(\lambda)$ is subdifferentiable.

Subgradient Method. We now present the canonical format of the subgradient method. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a subdifferentiable function over $\mathbb{R}^n$. Then, the subgradient method for the minimization of a function consists on:

Algorithm 1 (Subgradient Method)**.**

Initialization

- $x^0 \in \mathbb{R}^n$;
- *a sequence of step sizes,* $\{\alpha^k\}$;
- $k = 0$.

Step 1 *Compute a subgradient of* f *at the point* x^k: $g^k \in \partial f(x^k)$.

Step 2 $x^{k+1} = x^k - \alpha^k g^k$.

Step 3 $k \leftarrow k + 1$ *and return to Step* 1.

The notation $\partial f(x)$ is used to designate the subdifferential of the function f at the point x, *i.e.*, the set of all subgradients of f at that point.

There are many choices for the step sizes α^k. For some examples, we refer the reader to [5], where proofs of convergence are also provided.

Concerning the stopping criterion of the subgradient method, as [5] says, there is no "formal stopping criterion". However, there are good indicators that the convergence has already been attained. For example, if the subgradient g^k is too close to zero, that may mean that we are close to an optimal point. This follows from the fact that if a subdifferentiable convex function $f : \mathbb{R}^n \to \mathbb{R}$ has the zero vector in its subdifferential at a point x^*, then this is equivalent to saying that x^* is a global minimizer of f. Formally,

$$0 \in \partial f(x^*) \quad \Longleftrightarrow \quad x^* \text{ is a global minimizer of } f.$$

Another good evidence that might indicate that the algorithm has already converged is the fact that the cost function hasn't been decreasing for some iterations.

A subgradient of $H(\lambda)$. Returning to our problem, let X_λ be the index set

$$X_\lambda = \{\tilde{x} : \sup_{\|x\|_\infty \leq R} r(\lambda, x) = r(\lambda, \tilde{x})\},$$

i.e., the set of indices in x such that the supremum of $r(\lambda, x)$ subject to $\|x\|_\infty \leq R$ is attained for a given λ. The subgradient method only requires the availability of one subgradient of $H(\lambda)$ at each point. This way, as the subdifferential of a supremum of functions contains the convex hull of the union of the subdifferential of each function where the supremum is attained, *i.e.*,

$$\mathrm{co}\left(\bigcup_{x \in X_\lambda} \partial_\lambda r(\lambda, x)\right) \subset \partial_\lambda H(\lambda),$$

operation involving the variable λ is indexed to the iteration k of the subgradient method. However, we will not do so in this subsection, just for simplicity of notation. We will merely use the simple notation λ.

we have that, for $\tilde{x} \in X_\lambda$,

$$g \in \partial_\lambda r(\lambda, \tilde{x}) \Longrightarrow g \in \partial_\lambda H(\lambda).$$

In the above expressions, $\mathrm{co}(S)$ represents the convex hull of the set S and $\partial_\lambda f(\lambda)$ represents the subdifferential of the function $f(\lambda)$ with respect to the variable λ.

In fact, $r(\lambda, x)$ is differentiable with respect to λ, for a fixed x. So, $\partial_\lambda r(\lambda, x) = \{\nabla_\lambda r(\lambda, x)\}$, where $\nabla_\lambda r(\lambda, x)$ represents the gradient of $r(\lambda, x)$, with respect to λ. By (2.8), $\nabla_\lambda r(\lambda, \tilde{x}) = A\tilde{x} - b$, thus $A\tilde{x} - b \in \partial_\lambda H(\lambda)$, for $\tilde{x} \in X_\lambda$.

Finding $\tilde{x} \in X_\lambda$. However, we have not seen yet how to find an $\tilde{x}$ in X_λ. Since $\tilde{x}$ depends on λ, we will use the notation $\tilde{x}(\lambda)$ to make it clearer. This is where all the P processors come in.

From (2.7),

$$\tilde{x}_p(\lambda) \in \arg \min_{\|x_p\|_\infty \leq R} \|x_p\|_1 - \lambda^T A_p x_p, \qquad \text{for} \quad p = 1, 2, \ldots, P, \tag{2.9}$$

which can be computed in parallel by all the P processors. Then, a central node would collect all the $\tilde{x}_p(\lambda)$'s and would form $\tilde{x}(\lambda) = (\tilde{x}_1(\lambda), \tilde{x}_2(\lambda), \ldots, \tilde{x}_P(\lambda))$. Note that the existence of the solution of (2.9) is guaranteed by Weierstrass's theorem.

In its turn, every processor has to solve problem (2.9). Writing x_p as $\left(x_p^1, \ldots, x_p^{n_p}\right)$, for $p = 1, 2, \ldots, P$, and being a_p^j, $j = 1, \ldots, n_p$, each column of the submatrix

$$A_p = \begin{bmatrix} | & & | \\ a_p^1 & \cdots & a_p^{n_p} \\ | & & | \end{bmatrix},$$

we have the following equivalence

$$\min_{\|x_p\|_\infty \leq R} \|x_p\|_1 - \left(A_p^T \lambda\right)^T x_p \quad \Longleftrightarrow \quad \sum_{j=1}^{n_p} \min_{|x_p^j| \leq R} \left(|x_p^j| - \left({a_p^j}^T \lambda\right) x_p^j\right). \tag{2.10}$$

The solution for the jth component of x_p in (2.10) can be found in closed–form and is given by

$$\tilde{x}_p^j(\lambda) \in \left\{ \begin{array}{ccccc} \{0\} & \text{,if} & |{a_p^j}^T \lambda| & < & 1 \\ \left\{R \cdot \mathrm{sign}\left({a_p^j}^T \lambda\right)\right\} & \text{,if} & |{a_p^j}^T \lambda| & > & 1 \\ [-R, 0] & \text{,if} & {a_p^j}^T \lambda & = & -1 \\ [0, R] & \text{,if} & {a_p^j}^T \lambda & = & 1 \end{array} \right. , \quad j = 1, \ldots, n_p, \tag{2.11}$$

where the function $\mathrm{sign}(x)$ returns the sign of x. Figure 2.1 gives the graphical explanation of this solution. This way, we can find an $\tilde{x}(\lambda)$ in X_λ.

If we want to implement or simulate this method, we must decide which value to choose for $\tilde{x}_p^j(\lambda)$ on (2.11) when $|{a_p^j}^T \lambda| = 1$. We opt to choose the extremal ones, R or $-R$, for no particular reason. So, when ${a_p^j}^T \lambda = -1$, we make $\tilde{x}_p^j(\lambda) = -R$; when ${a_p^j}^T \lambda = 1$, we make $\tilde{x}_p^j(\lambda) = R$; and (2.11) becomes

$$\tilde{x}_p^j(\lambda) = \left\{ \begin{array}{ccccc} 0 & \text{,if} & |{a_p^j}^T \lambda| & < & 1 \\ R \cdot \mathrm{sign}\left({a_p^j}^T \lambda\right) & \text{,if} & |{a_p^j}^T \lambda| & \geq & 1 \end{array} \right. , \quad j = 1, \ldots, n_p. \tag{2.12}$$

In subsection 2.1.2 we will see that the optimal dual variable λ^* verifies $|{a_p^j}^T \lambda^*| \leq 1$ for all j and for all p. In other words, λ^* belongs to the set $\mathcal{P} = \left\{\lambda : \|A^T \lambda\|_\infty \leq 1\right\}$. However, along the iterations of the subgradient method, we don't have any guarantee that $\lambda \in \mathcal{P}$. This is the reason why (2.12) includes the possibility of $\lambda \notin \mathcal{P}$.

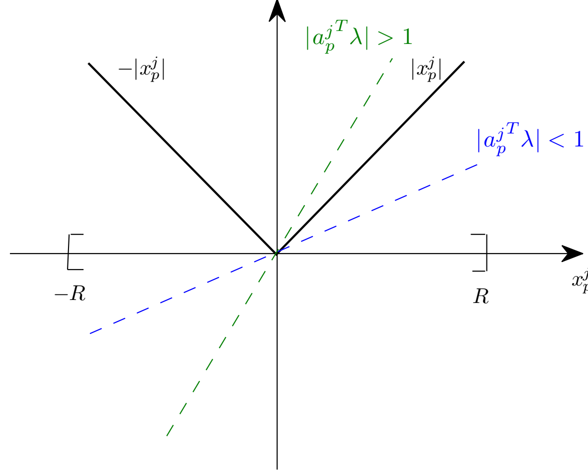


Figure 2.1: Graphical interpretation of the solution of problem (2.10), for each component of x_p. The solid line shows the function $|x_p^j|$, while the dashed lines show the linear function $-(a_p^{j\,T}\lambda)x_p^j$ for the two relevant cases: when the absolute value of the slope $-a_p^{j\,T}\lambda$ is smaller than one, the optimal solution is $\tilde{x}_p^j(\lambda) = 0$; otherwise, if that number is greater than one, $\tilde{x}_p^j(\lambda)$ lies on one of the extremal points, R or $-R$.

Cost function. We might be interested in using the cost function (2.7) in a stopping criterion of the subgradient method. Now, we will see how to evaluate it, assuming that at each iteration of the subgradient method the optimal point $\tilde{x}(\lambda)$, given by (2.12) for all j and for all p, is known. Replacing it in (2.7), we get

$$H(\lambda) = -\lambda^T b - \sum_{p=1}^{P} \left(\|\tilde{x}_p(\lambda)\|_1 - \lambda^T A_p \tilde{x}_p(\lambda) \right),$$

and taking into account that $\|\tilde{x}(\lambda)\|_1 = \|\tilde{x}_1(\lambda)\|_1 + \|\tilde{x}_2(\lambda)\|_1 + \ldots + \|\tilde{x}_P(\lambda)\|_1$, and

$$A_1\tilde{x}_1(\lambda) + A_2\tilde{x}_2(\lambda) + \ldots + A_P\tilde{x}_P(\lambda) = A\tilde{x}(\lambda),$$

where $\tilde{x}(\lambda) = (\tilde{x}_1(\lambda), \tilde{x}_2(\lambda), \ldots, \tilde{x}_P(\lambda))$ and $A = [A_1 A_2 \ldots A_P]$,

$$\begin{aligned} H(\lambda) &= -\lambda^T b + \lambda^T A\tilde{x}(\lambda) - \|\tilde{x}(\lambda)\|_1 \\ &= \lambda^T g^k - \underbrace{\|\tilde{x}(\lambda)\|_1}_{R|\tilde{\Omega}|}, \end{aligned}$$

being $\tilde{\Omega}$ the set of indices for which the components of $\tilde{x}(\lambda)$ are non-zero; $|\tilde{\Omega}|$ its cardinality; and $g^k = A\tilde{x}(\lambda) - b \in \partial_\lambda H(\lambda)$ a subgradient of $H(\lambda)$ at λ.

Transmission issue. From what we have seen until now, the only information that the processors have to transmit to the central processor in each step of the subgradient method is the product $A_p\tilde{x}_p(\lambda^k)$ to calculate a subgradient (and the number of the non-zero entries of $\tilde{x}_p(\lambda^k)$, if the central processor wants to evaluate the cost function).

2.1.2 Solving The Bounded BP Problem Through Its Dual

In this subsection, we will see how to find an optimal primal variable of the bounded BP (2.3); first by assuming that we know an optimal dual variable λ^*, and then adapting it to the practical case, where we only have available an approximation of λ^*. The notation x^* will be used to designate any solution of either (2.1) or (2.3). Indeed, both problems have the same solutions if the assumption taken at the beginning, that R is large enough, holds.

An optimal dual variable is known. We now assume that we know an optimal dual variable $\lambda^\star$, which solves (2.6).

Since strong duality holds, the optimal solution $x^\star$ can be obtained by solving the KKT-system for (2.3):

$$\begin{cases} x^\star \in \arg\min_{\|x\|_\infty \le R} \quad {\lambda^\star}^T b + \sum_{i=1}^n \left(|x_i| - a_i^T \lambda^\star x_i\right) \\ \qquad Ax^\star = b \end{cases} \tag{2.13}$$

Note that now we are not considering the partition of the variable x into P subvariables as before, but into n subvariables in $\mathbb{R}$ instead[3].

We now claim that λ^* must belong to the polyhedron $\mathcal{P} = \{\lambda \,:\, \|A^T\lambda\|_\infty \le 1\}$, *i.e.*, it satisfies $|a_i^T\lambda^*| \le 1$ for all $i = 1, \ldots, n$. Indeed, suppose that $|a_i^T\lambda^*| > 1$ for some i. Then, according to (2.13) we would have $|x_i^*| = R$ for any solution x^* of (2.3). But this contradicts our initial assumption that the solution set of (2.1) lies in the interior of $B_\infty(0, R)$ (we recall that the solution sets of (2.1) and (2.3) are the same).

Let's see what we can find about each component i of the solutions of (2.13).

- If $x_i^\star > 0$, then

$$\begin{aligned} \frac{d}{dx_i}\left(x_i - a_i^T\lambda^\star x_i\right)\Big|_{x_i^\star} = 0 &\Longleftrightarrow 1 - a_i^T\lambda^\star = 0 \\ &\Longleftrightarrow a_i^T\lambda^\star = 1 \end{aligned} \tag{2.14}$$

- If $x_i^\star < 0$, then

$$\begin{aligned} \frac{d}{dx_i}\left(-x_i - a_i^T\lambda^\star x_i\right)\Big|_{x_i^\star} = 0 &\Longleftrightarrow -1 - a_i^T\lambda^\star = 0 \\ &\Longleftrightarrow a_i^T\lambda^\star = -1 \end{aligned} \tag{2.15}$$

- Finally, when $x_i^* = 0$, we have $|a_i^T\lambda^*| \le 1$.

This implies that if $|a_i^T\lambda^*| < 1$, we have $x_i^* = 0$; if $a_i^T\lambda^* = 1$, then $x_i^* \ge 0$; and when $a_i^T\lambda^* = -1$, we have $x_i^* \le 0$.

From this, we can see that $x^\star$ cannot be found only from the knowledge of the optimal dual variable $\lambda^\star$ together with the first equation of (2.13). The only thing we can know with this information is the sign of each of its entries. Apparently this is not too valuable, but it allows us to discard some columns of the matrix A and solve a smaller problem, *i.e.*, if $|a_i^T\lambda^*| = 1$, we already know that the corresponding column of A, a_i, can be activated by x_i^*; if $|a_i^T\lambda^*| < 1$, it won't. So, the second equation of the KKT-system (2.13) can be solved much easily, in a lower dimension, using only the columns of A for which $|a_i^T\lambda^*| = 1$.

As we are expecting that the optimal solution is sparse, the reduction of the dimensions of the problem can be immense.

[3]The numbers of the processors don't matter here, so we change the index of a column and the index of a variable into the subscript notation throughout this subsection.

Only an approximation of λ^* is known. We now admit that only an approximation of λ^* is available. Let that approximation be $\hat{\lambda}$. This situation is the most common in practice and arises mainly because of the *bang–bang* solution we adopted for $\tilde{x}(\lambda^k)$, at the iteration k of the subgradient method, *i.e.*, the non-zero components of $\tilde{x}(\lambda^k)$ can only assume extremal values: R or $-R$. In fact, this originates a ripple in the inner products $a_i^T\lambda^k$, along the iterations of the subgradient method (see Figures 2.3-2.6 from the simulations). As a consequence, we need to establish a threshold $\xi > 0$ in order to tell, when the subgradient method has converged and we have access to $\hat{\lambda}$, whether each number $|a_i^T\hat{\lambda}|$ is smaller or equal to 1. This way, we have to adopt the following decision criterion

$$\begin{cases} \text{Column } a_i \text{ is discarded,} & \text{if} \quad |a_i^T\hat{\lambda}| \leq 1-\xi \\ \text{Column } a_i \text{ is kept,} & \text{if} \quad |a_i^T\hat{\lambda}| > 1-\xi \end{cases}. \tag{2.16}$$

Assuming that $\hat{\lambda}$ is accurate and that ξ is well adjusted, with the information from (2.16) the bounded BP (2.3) becomes

$$\min_{Mu=b} \|u\|_1, \tag{2.17}$$

a problem which we will call the *reduced basis pursuit* (RBP). Note that we dropped the constraint $\|u\|_\infty \leq R$, as we don't need it anymore. Indeed, we only used it as a "trick" to solve a dual problem and get an approximation of the dual variable. In problem (2.17), M is the matrix formed by the chosen columns of A, through the decision process (2.16). To formalize, we define the set

$$\Omega = \{i \,:\, |a_i^T\hat{\lambda}| > 1-\xi\},$$

and designate its cardinality by n'. This way, matrix M lies in $\mathbb{R}^{m\times n'}$ and is defined by

$$M = A|_\Omega.$$

For later use, we also define the set

$$\Omega^* = \{i \,:\, x_i^* \neq 0\},$$

where x^* is a solution of (2.1) (or (2.3)). Note that we have, in general, that $\Omega \neq \Omega^*$, for any optimal set Ω^*, not only due to the error in the dual variable, $\|\hat{\lambda} - \lambda^*\|$, but also because we must choose an $\xi > 0$. Actually, the incidence of these kinds of errors that lead to $\Omega \neq \Omega^*$ are the major drawback of this method.

Is RBP important? We can ask, however, if solving the RBP is really necessary. Actually, the answer can be yes and no. In the most interesting practical cases, when the optimal solution is quite sparse, the matrix A has good properties, the parameter ξ is well adjusted and the subgradient method finds an accurate dual variable $\hat{\lambda}$, we don't need to solve (2.17) since the linear system $Mu = b$ has a single solution; it just suffices to solve that linear system. The following lemma translates this into mathematical terms.

Lemma 1. *Let $\delta_s(A)$ designate the restricted isometry constant of the matrix A for s. Let also n' be the number of columns of the matrix M. If $\delta_{n'}(A) < 1$ and the linear system*

$$Mu = b \tag{2.18}$$

has a solution, then that solution is unique.

Proof. By assumption, (2.18) has at least one solution.

Now, admit that there exist two different solutions, *i.e.*, $\hat{u}$ and $\tilde{u}$ both in $\mathbb{R}^{n'}$ such that $\hat{u} \neq \tilde{u}$ and $M\hat{u} = M\tilde{u} = b$. This means that $M(\hat{u} - \tilde{u}) = 0$.

On the other hand, we have $\delta_{n'}(M) \leq \delta_{n'}(A) < 1$, which means that the columns of M are linearly independent. Therefore, it doesn't exist any $h \in \mathbb{R}^{n'} \backslash \{0\}$ such that $Mh = 0$. We reached a contradiction. □

The condition $\delta_{n'}(A) < 1$ usually holds if n' is small (that is, if the decision process (2.16) returns few columns), and if the matrix has practical interest (recall the motivations that make us try to solve (2.1) in subsection 1.3.1). For example, if A is a random matrix in $\mathbb{R}^m \times \mathbb{R}^n$, and $n' \leq m$, then this condition holds with high probability. Recall, however, that evaluating the restricted isometry constant of a matrix is impractical in most cases. Nevertheless, we will assume, from now on, that it is possible to know a certain integer L such that

$$n' \leq L \quad \Longrightarrow \quad \delta_{n'}(A) < 1. \tag{2.19}$$

In the worst case, when no information of the matrix A is available at all, $L = 1$ works. But this is an extreme case, with no practical interest. For example, for random matrices, an L equal to the number of rows, m, guarantees that (2.19) holds with high probability.

When has (2.18) **at least one solution?** The condition that the linear system (2.18) has at least one solution isn't easy to ensure in practice. It depends on a "good" stopping criterion of the subgradient method, as well as on a "good" adjustment of the parameter ξ (of course, if we take ξ close to 1, surely (2.18) has at least one solution, but we don't benefit from the small reduction of dimensions from A to M). It is intuitive that the existence of any optimal Ω^* such that $\Omega^* \subset \Omega$ is sufficient for (2.18) have at least one solution.

Lemma 2. *If it exists any Ω^* such that $\Omega^* \subset \Omega$, then the linear system* (2.18) *has at least one solution.*

Proof. Let x^* be an optimal solution of (2.1) (by the assumption on the rank of A, it exists). As x^* is a feasible point,

$$Ax^* = A|_{\Omega^*} u^* = b,$$

where $u^* = x^*|_{\Omega^*}$. Recall the definition of M: $M = A|_\Omega$. If $\Omega^* \subset \Omega$, then A_{Ω^*} is a submatrix of M, in the sense that all columns of A_{Ω^*} are contained in M. Therefore, the vector $\tilde{u}$ defined in $\mathbb{R}^{n'}$ $(n' = |\Omega|)$ that verifies $\tilde{u} = x^*|_{\Omega^*}$ and is zero elsewhere solves (2.18). □

Overcoming the infeasibility. We will see in the simulations that the subgradient method, together with (2.16), usually generates more columns than the needed ones, *i.e.*, $n' = |\Omega| > |\Omega^*|$, for any optimal set Ω^*. When the reason for choosing a non–optimal column is inherent to the subgradient method we say that a "*false alarm*" occurred (see Figure 2.6 for an illustration). Even though, in the case when we choose more columns than needed, we might have $\Omega^* \not\subset \Omega$, for all Ω^*. When this happens, we don't even have any guarantees that either the linear system (2.18) or the problem (2.17) are feasible. In this situation, perhaps it is more prudent to solve a problem like the BPDN instead:

$$\min_u \frac{1}{2}\|Mu - b\|^2 + \beta\|u\|_1, \tag{2.20}$$

for some value of β. The standard procedure would then be solving (2.20) for several values of β and then choosing the more appropriate solution, *i.e.*, the one we were expecting the most.

Though actually, when the number of columns n' of the matrix M is small enough, there is a value for β that yields good results: $\beta = 0$. Indeed, the solution is already sparse, so we don't need the term that induces sparsity, $\beta\|u\|_1$. Moreover, the resulting problem

$$\min_u \|Mu - b\|^2, \tag{2.21}$$

besides having a closed–form solution, has a single solution, with zero as an optimal value, if the conditions $\Omega^* \subset \Omega$ and $\delta_{n'}(A) < 1$ hold. This follows directly from lemmas 1 and 2. Also note that the computational complexity in solving either the linear system (2.18) or the least–squares (2.21) is similar.

On the other hand, when n' isn't small, for instance greater than m, now there is a great probability of the vector b being in the span set of M, making the RBP (2.17) have at least one feasible point. Besides that, if $\Omega^* \subset \Omega$, then the RBP returns an optimal solution u^*. We define an optimal solution u^* as being

$$u^* = x^*|_{\Omega^*},$$

that is, the vector that collects all the non–zero entries of a solution of (2.1), x^*.

A procedure to find $\hat{x}$. After solving the dual of the bounded BP, we must do our best in order to find a vector $\hat{x}$ that is close to an optimal solution x^* of the BP (2.1). Not even always we can guarantee that we find an x^*.

So, a possible procedure to try to find an x^* is the following:

Procedure 1 (Calculate $\hat{x}$)**.**

Input:

- *The set Ω and the matrix $M = A|_\Omega \in \mathbb{R}^{m\times n'}$;*
- *the vector b and the number L.*

Step 1 *Solve the least–squares problem*

$$q = \min_u \|Mu - b\|, \tag{2.22}$$

and designate the found solution by u'.

Step 2 *If $q > 0$ or $n' \leq L$, make $\hat{u} = u'$ and go to step 4.*

Step 3 *Else ($q = 0$), solve the RBP*

$$\min_{Mu=b} \|u\|_1, \tag{2.23}$$

and designate the found solution by $\hat{u}$.

Step 4 *Form $\hat{x} \in \mathbb{R}^n$ by making $\hat{x}|_\Omega = \hat{u}$ and equating all the other entries to zero.*

One positive aspect of this procedure is that it always evaluates the feasibility of (2.23) before trying to solve it. Moreover, the uniqueness of its feasible set is also checked by the condition $n' \leq L$ also before trying to solve it. If $q = 0$ and $n' \leq L$ we have already found the solution of (2.23) in step 1. On another hand, when it is found that the matrix M is incomplete in the sense that it can't span b, the vector that is closest to verify the linear system $Mu = b$ is returned.

The following theorem establishes an important result about procedure 1.

Theorem 3. *Procedure 1 returns $\hat{x}$ equal to an optimal solution x^* if and only if an optimal set Ω^* is contained in Ω,* i.e.

$$\exists_{\Omega^*} : \Omega^* \subset \Omega \qquad \Longleftrightarrow \qquad \hat{x} = x^*,$$

where Ω^* is any optimal set that doesn't necessarily correspond to x^*.

Proof. Admit that $\Omega^* \subset \Omega$. From lemma 2 the linear system $Mu = b$ has at least one solution. Hence, $q = 0$. Following the same steps of the proof of lemma 2, we will prove that the vector $\tilde{u}$ in $\mathbb{R}^{n'}$ such that $\tilde{u}|_{\Omega^*} = u^*$ and is zero elsewhere is a solution of the linear system $Mu = b$. Note that, by the definition of u^*, we have

$$Ax^* = A|_{\Omega^*} u^* = b.$$

This proves that $M\tilde{u} = b$, since M contains $A|_{\Omega^*}$ as its submatrix. There are two options now:

- If $n' \leq L$, then the vector $\tilde{u}$ is the only solution of the linear system $Mu = b$, as follows from lemma 1. Since we have $q = 0$, the vector u' found in step 1 must be equal to $\tilde{u}$. It follows that step 4 generates x^*, by the definition of $\tilde{u}$, $\hat{u}$ and u^*.
- Otherwise, if $n' > L$, the vector $\tilde{u}$ solves the RBP (2.23), because $\|\tilde{u}\|_1 = \|u^*\|_1 = \|x^*\|_1$. This way, step 4 generates an optimal solution x^*, even if there is another $u \neq \tilde{u}$ that solves (2.23).

To prove the converse implication, suppose that procedure 1 returned an x^* and that $\Omega^* \not\subset \Omega$ for all optimal sets Ω^*. Without loss of generality, pick one of those sets. If Ω^* is the empty set, then there is a contradiction since the empty set is contained in any set. Assume now that $\Omega^* \neq \emptyset$. As $\Omega^* \not\subset \Omega$, there is a $k \in \Omega^*$ such that $k \notin \Omega$. By the definition of Ω^*, we have $x_k^* \neq 0$. This contradicts the step 4 of the procedure.

□

At this point, we can say that for the case when no optimal set Ω^* such that $\Omega^* \subset \Omega$ exists and procedure 1 executes its 4th step, then this procedure returns not necessarily a vector close to an optimal solution, but the one that has the least ℓ_1-norm in the available linear space $\{u : Mu = b\}$.

Additional sign information From theorem 3 we see that the optimal solution can only be found running procedure 1 if there exists an Ω^* such that $\Omega^* \subset \Omega$. The information about the sign of the entries of the vector $\hat{x}$, which can be obtained from the dual variable $\hat{\lambda}$ using (2.14) and (2.15), isn't used in this procedure.

We can code it mathematically by the introduction of the restriction $\Phi u \geq 0$, where $\Phi = \mathrm{Diag}(r)$ and r is a vector in $\mathbb{R}^{n'}$ such that

$$r_i = \begin{cases} 1 & ,\text{if} \quad u_i > 0 \\ -1 & ,\text{if} \quad u_i < 0 \end{cases}, \qquad \text{for } i = 1, \ldots, n'. \tag{2.24}$$

The notation $\mathrm{Diag}(v)$ is used to designate the square matrix Q by Q (if $v \in \mathbb{R}^Q$), where its diagonal equals the entries of the vector v and is zero elsewhere, *i.e.*

$$\mathrm{Diag}(v) = \begin{bmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_Q \end{bmatrix}.$$

If we have $\Omega^* \subset \Omega$ for any optimal set Ω^*, then theorem 3 guarantees that the procedure 1 finds the optimal primal variable x^*. So, in this case, there is no benefit in using the inequality $\Phi u > 0$.

What about when it doesn't exist any Ω^* such that $\Omega^* \not\subset \Omega$? In this case, theorem 3 says that we can't have $\hat{x} = x^*$, for any optimal solution x^*. But does the inequality $\Phi u > 0$ help anyway?

In fact, it doesn't. If we add any restriction to (2.22) we might be “loosing feasibility” and even “gain infeasibility”, whereas if we add any restriction to (2.23) we might be “loosing optimality” (note that $\|\hat{x}\|_1 = \|\hat{u}\|_1$).

This way we conclude that it isn't worthy to use this information about the sign of the entries of $\hat{x}$, the variable returned by procedure 1. When $\hat{x} \neq x^*$, where x^* is any solution of (2.1), we don't have any guarantee that we will find a better solution than $\hat{x}$.

Transmission issue. After all the processors cooperate to get to a dual variable $\hat{\lambda}$ (subgradient method), from what we have seen in this subsection namely in procedure 1, it is straightforward to see that the only information that the processors $p = 1, \ldots, P$ need to transmit to the central processor is the set Ω and the matrix $M = A|_{\Omega}$.

2.1.3 The Resulting Algorithm

Gathering all that we have seen in the previous subsections, we finally can write the overall algorithm for solving the BP (2.1).

Algorithm 2 (Subgradient Method For Bounded BP)**.**

- Predefined Parameters/Initialization*:*
  - *A_p for each processor, $p = 1, \ldots, P$;*
  - *A bound R for $\|x^\star\|_\infty$; the maximum number of iterations, K; the parameter $0 < \zeta < 1$ for choosing the interesting columns: a_i is interesting if $|a_i^T \hat{\lambda}| \geq \zeta$ (note that $\zeta = 1 - \xi$); an integer L such that* (2.19) *is verified.*
  - *Choose λ^0.*
- Procedure (for central processor):
  - *Receive b (from outside);*
  - *for $k = 0$ until K [subgradient method cycle]*
    1. *§1 Send λ^k and R to each processor $p = 1, \ldots, P$;*
    2. *§1 Receive $A_p \tilde{x}_p(\lambda^k)$ (and possibly $|\Omega_p^k| = \|\tilde{x}_p(\lambda^k)\|_0$), from each processor;*
    3. *Compute $g^k = A_1 \tilde{x}_1(\lambda^k) + \ldots + A_P \tilde{x}_P(\lambda^k) - b$;*
    4. *Check stopping criterion (if based on the cost function, evaluate $H(\lambda^k) = {\lambda^k}^T g^k - R \sum_{p=1}^{P} |\Omega_p^k|$), and break the cycle if verified;*
    5. *Choose step size, α^k;*
    6. *$\lambda^{k+1} = \lambda^k - \alpha^k g^k$;*
  - *$\hat{\lambda} = \lambda^k$;*
  - *§2 Send $\hat{\lambda}$ and ζ to each processor $p = 1, \ldots, P$.*
  - *§2 Receive $\Omega_p = \{i \,:\, \hat{x}_i \neq 0,\ \textit{and } i \textit{ is in the index range of the processor } p\}$ and $A_p|_{\Omega_p}$ from each processor p.*
  - *Form the matrix $M = \left[A_1|_{\Omega_1} \cdots A_P|_{\Omega_P}\right]$.*
  - *Run **procedure 1**.*

- *Return $\hat{x}$.*

- Procedure (for each processor $p = 1, \ldots, P$)
  There are two modes (referenced above as §1 and §2):

  - *Mode 1:*
    1. *Receive λ^k and R;*
    2. *Compute $c = A_p^T \lambda^k$ (A_p is stored internally);*
    3. *Set $x^j = 0$ if $|c^j| < 1$, and $x^j = R \cdot \text{sign}(c^j)$ if $|c^j| \geq 1$, for $j = 1, \ldots, n_p$;*
    4. *Return $A_p \tilde{x}_p(\lambda^k)$, where $\tilde{x}_p(\lambda^k) = \left(x^1, \ldots, x^{n_p}\right)$ (and possibly $|\Omega_p^k| = \|\tilde{x}_p(\lambda^k)\|_0$).*
  - *Mode 2:*
    1. *Receive $\hat{\lambda}$ and ζ;*
    2. *Compute $c = A_p^T \hat{\lambda}$ (A_p is stored internally);*
    3. *Return the interesting columns of A_p, $A_p|_{\Omega_p}$, where $\Omega_p = \{i : \hat{x}_i \neq 0, \text{ and } i \text{ is in the index range of the processor } p\}$, and also return the set Ω_p.*

In Figure 2.2 there is a possible architecture for the implementation of algorithm 2. Note that the operations that each processor must be able to do are relatively simple: multiplications, additions and some binary operations. Moreover, there might be situations in which the products $A_p \tilde{x}_p(\lambda^k)$ and $A_p^T \lambda^k$ can be computed very fast, using the internal structure of the matrix A_p. Nonetheless, each processor must be able to store a matrix A_p. It operates in two modes: Mode 1 is run at most K times, receiving $m+1$ numbers and returning equally $m+1$ numbers, where m is the number of rows of A; mode 2 is run only once, receiving $m+1$ numbers, and returning a variable number of columns of A for further calculations at the central processor.

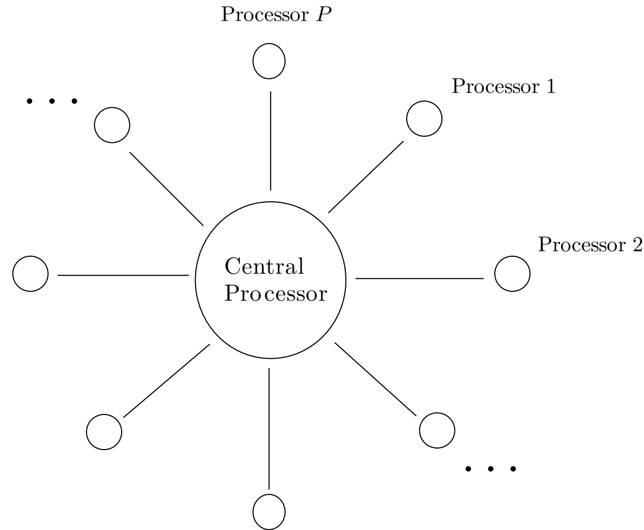
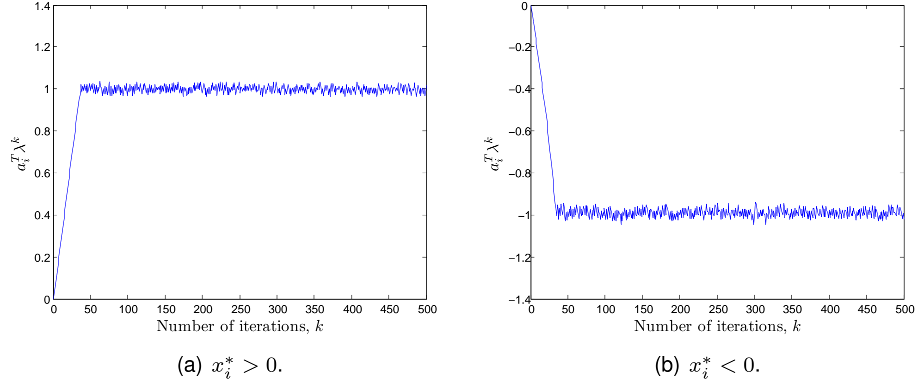


Figure 2.2: Required minimal architecture of the links between the processors for the implementation of the subgradient method for bounded BP (algorithm 2). There is a central node which doesn't know any column of A. In fact, A isn't stored in a single processor; its columns are stored throughout P processors, which solve distributedly a (dual) problem in order to determine which columns should be used for solving a reduced equivalent problem. This reduced problem is then solved at the central processor.



(a) $x_i^* > 0$.

(b) $x_i^* < 0$.

Figure 2.3: Typical behavior of the inner product $a_i^T\lambda^k$ along 500 iterations of algorithm 2, in which a_i is a column of A associated with a non-zero coefficient of x^*.

Ripple in the iterations. Variable ζ is really needed in algorithm 2 because of the ripple found in the inner product $a_i^T\lambda^k$ along the iterations of the subgradient method, for each column $i = 1, \dots, n$. This happens because we adopted a *bang–bang* solution for $\tilde{x}(\lambda^k)$, *i.e.*, its non-zero components can only assume extremal values: R or $-R$. In Figure 2.3, it is plotted the evolution of $a_i^T\lambda^k$ for columns a_i of A which are known to have a non-zero contribution for the exact solution, *i.e.*, $x_i^* \neq 0$, where x^* is the exact solution of (2.1), which can be obtained by software that solves its linear program formulation (1.11)[4]. Due to this behavior, if the parameter ζ is too close to one, we might be loosing interesting columns for the subsequent least–squares problem (2.22) or RBP (2.23) in procedure 1. So, not only a moderate value for ζ is advised, but also a filtering of the inner products $a_i^T\lambda^k$ over the iterations. Figure 2.4 shows an example. One can also consider the mean of the last T values.

The adaptation of the algorithm for this situation is trivial: each processor has to keep track of the last value of the filtered values, or of the last T values, of $a_i^T\lambda^k$ for each of its stored columns. When the processors are called in mode 2 they wouldn't use $c = A_p^T\lambda^k$, but the filtered products or the mean for the last T components instead.

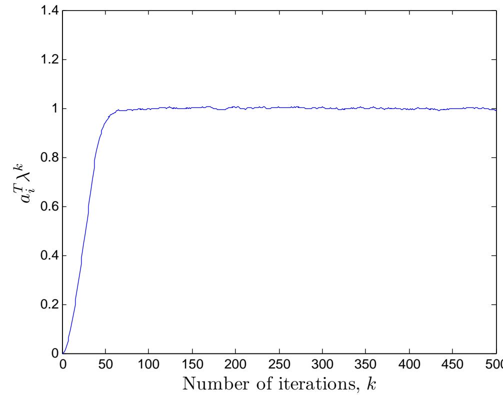


Figure 2.4: Same example as in Figure 2.3(a), but with filtering along the iterations. The used filter was an exponential weighted moving average one: $f^{k+1} = \gamma a_i^T\lambda^k + (1-\gamma)f^k$, with $f^0 = 0$, and $\gamma = 0.1$.

Figure 2.5 shows the typical behavior of $a_i^T\lambda^k$ for the situation where $x_i^* = 0$ in the optimal solution. Note that in Figs. 2.3 and 2.4 the number of iterations was deliberately reduced for the transient period be visible. Of course in both plots the behavior is maintained no matter how much we increase the

[4]Note that while we use x^* to designate the solution of (2.1), we use the notation $\hat{x}$ to designate the output of algorithm 2.

number of iterations.

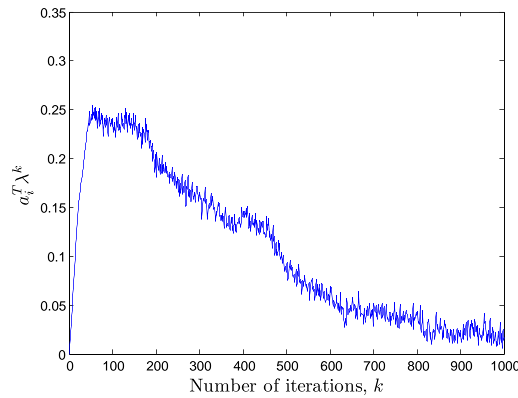
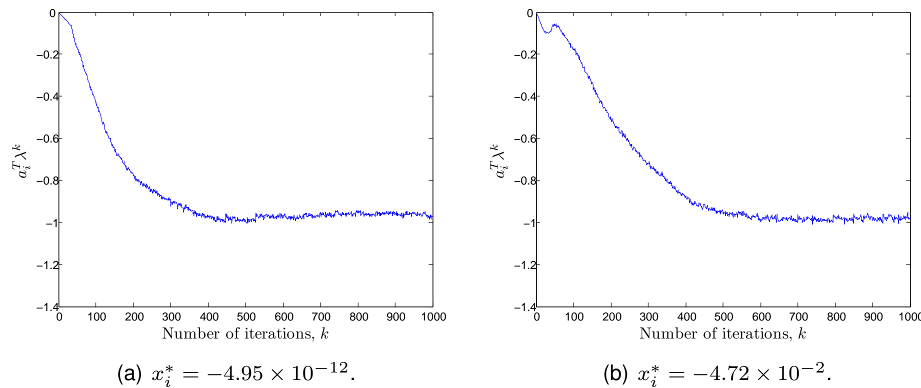


Figure 2.5: Evolution of the inner product $a_i^T \lambda^k$ along the iterations of the algorithm 2 when the correspondent component x_i^* is close to zero. Note that it doesn't approach 1 or -1 in general, it approaches an arbitrary number between them instead.

However, there are awkward situations in which columns with a practically null contribution to the optimal solution exhibit the same behavior as a column with a moderate contribution. That is, both inner products have similar behaviors over the iterations (see Figure 2.6). This means that there are some "false alarm" situations (we choose more columns than the ones we need) when the number of iterations is too high. This fact validates some of the options taken when we designed the procedure 1.



(a) $x_i^* = -4.95 \times 10^{-12}$.

(b) $x_i^* = -4.72 \times 10^{-2}$.

Figure 2.6: Inner product of a column of A with λ^k over 1000 iterations of the algorithm 2, for two different situations: in (a) the corresponding column has no contribution for the optimal solution; in (b) the corresponding column has a moderate contribution. Nevertheless, they exhibit the same behavior over the iterations. Note that $\max_i |x_i^*| = 1.49$ for this example.

So, we are able to distinguish four different situations, depicted on table 2.1. In general, there

Table 2.1: Different types of behavior of $|a_i^T \lambda^k|$ along the iterations of the subgradient method phase of algorithm 2.

$\lvert x_i^* \rvert$	Behavior of $\lvert a_i^T \lambda^k \rvert$
large	converges to 1 rapidly
moderate	converges slowly to 1
small	usually doesn't converge to 1
small	few situations when it converges to one, but slowly

are some "false alarms", but not even always the subgradient method together with the decision pro-

cess (2.16) “catches” all the necessary columns in order to find an optimal solution, *i.e.*, $\Omega^* \subset \Omega$. Figure 2.7 illustrates an example of this by showing the superposition of the plots of the exact solution of (2.1), x^*, and the result of the algorithm 2, $\hat{x}$. Note that, in the case of Figure 2.7, after the decision process, it didn't exist any Ω^* such that $\Omega^* \subset \Omega$, in spite of the cardinality of Ω being 16 (that is, the subgradient method returned 16 columns) and the cardinality of Ω^* for the x^* of the figure being only 10. Nevertheless, the relative error of the vector $\hat{x}$ returned by algorithm 2 in comparison with an optimal solution x^* was only 1.7%.

Actually, just one column indexed by Ω^* is missing in Ω. That missing column has a moderate contribution in x^* (-4.12×10^{-2}), thus the behavior of the corresponding inner product $a_i^T \lambda^k$ is similar to the one in Figure 2.6(b). This suggests that either we executed few iterations of the subgradient method, or we chose a too high value for ζ. Figure 2.8 shows the behavior of that missing column. There, it is

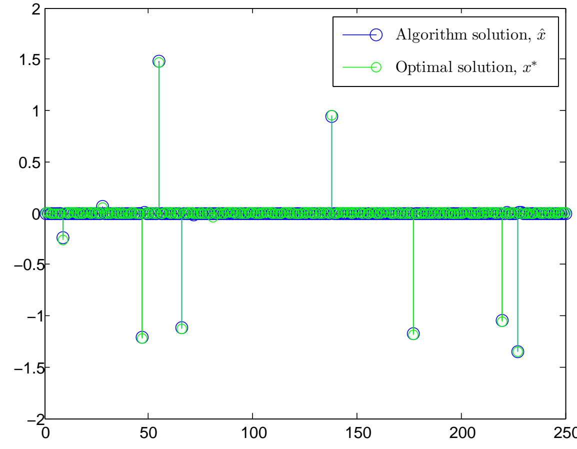


Figure 2.7: Comparison between an exact solution of (2.1), x^*, and the solution returned by algorithm 2, $\hat{x}$. The parameters of this experience were (the same for Figs. 2.3–2.9): $m = 50$ (number of rows of A), $n = 250$ (number of columns of A), $P = 10$ (number of processors), $R = 3$ (majorant of $\|x^*\|_\infty$), $K = 1000$ (number of iterations), $\gamma = 0.1$ (exponential moving average filter) and $\zeta = 0.9$ (decision process); and the step used in the subgradient method was $\alpha = 10^{-2}/\|g^k\|$; The column reduction resultant from the decision process was about 94%, having the matrix M just 16 columns. Notice that $\mathrm{card}(x^*) = 10$. The relative error of $\hat{x}$ was 1.7%, and we also had $\|A\hat{x} - b\| = 0.27$.

evident that just 1000 iterations of the subgradient method weren't enough for the inner product $a_i^T \lambda$ to get close to -1.

Guarantees of convergence. Concerning the guarantees of convergence of the subgradient method phase of algorithm 2 for the choices made in Figs. 2.3–2.9, we must say that this is dependent on the number of iterations, as well as on the spectral norm of the matrix A, on the vector b and even on R.

First, we note that [5] allows us to conclude that the step length chosen ($10^{-2}/\|g^k\|$) guarantees that $|H(\lambda^k) - H(\lambda^\star)| \downarrow 10^{-2}G/2$, as $k \to +\infty$. G is a bound for the norm of the subgradient $g^k = A\tilde{x}(\lambda^k) - b$, along all the iterations carried. We can estimate G by noticing that

$$
\begin{aligned}
\|g^k\| &= \|A\tilde{x}(\lambda^k) - b\| \\
&\leq \|A\tilde{x}(\lambda^k)\| + \|b\| \\
&\leq \sigma_{\max}(A)\|\tilde{x}(\lambda^k)\| + \|b\| && (2.25) \\
&\leq \sigma_{\max}(A)\sqrt{n}\|\tilde{x}(\lambda^k)\|_\infty + \|b\| && (2.26) \\
&\leq \sigma_{\max}(A)R\sqrt{n} + \|b\|, && (2.27)
\end{aligned}
$$

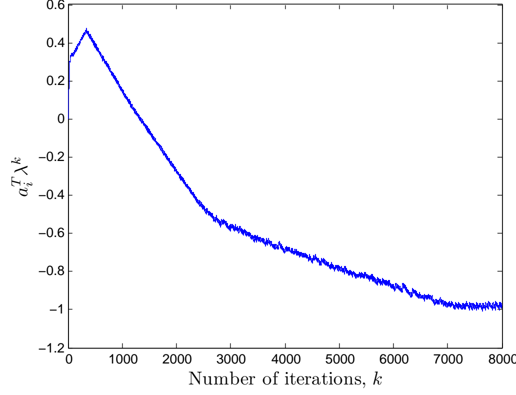


Figure 2.8: Inner product $a_i^T \lambda^k$ along the iterations of the subgradient method, where a_i is the column that was wrongfully discarded in the experience of the Figure 2.7. It is quite clear that 1000 iterations aren't enough for letting this inner product to get close to -1. On the other hand, if we had executed 8000 iterations of the subgradient method, the column reduction would only be 86% (35 columns returned by the subgradient method and by the decision process with $\zeta = 0.9$).

where $\sigma_{\max}(A)$ is the largest singular value of the matrix A. From (2.25) to (2.26) we used the fact that $\|x\| \leq \sqrt{n}\|x\|_\infty$ holds for any vector $x \in \mathbb{R}^n$. Inequality (2.27) follows obviously from $\|\tilde{x}(\lambda^k)\|_\infty \leq R$. Thus, we can make $G = \sigma_{\max}(A)R\sqrt{n} + \|b\|$. The matrix used in those graphics is a random matrix with $\sigma_{\max}(A) < 22.5$. Therefore, we can conclude that the subgradient method in algorithm 2 solves the dual problem (2.6) with an error on the cost function of 5.46 (the norm of b is less than 25.2 and we chose $R = 3$), as long as k is large enough.

As we can see, this bound on the dual cost function and the number of iterations of the subgradient method weren't enough to guarantee that exists an optimal index set of columns Ω^* such that $\Omega^* \subset \Omega$. If such set existed, theorem 3 would have guaranteed that we had found an optimal solution.

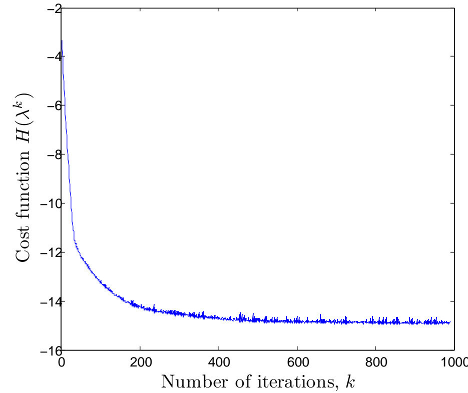


Figure 2.9: Evolution of the cost function $H(\lambda^k)$ along the iterations. The parameters are the ones described in Figure 2.7.

The cost function. Finally, the cost function $H(\lambda^k) = {\lambda^k}^T g^k - \|\tilde{x}(\lambda^k)\|_1$ along the iterations of the subgradient method is plotted in Figure 2.9. Note that it isn't always decreasing, since the subgradient method is a non–descent method. Indeed, it has the same rippling characteristic that the inner products $a_i^T \lambda^k$ have. Also note that the cost function only decreases slightly when some components $a_i^T \lambda^k$ are still converging to ± 1 (confront Figures 2.6 and 2.9). Therefore, care must be taken when choosing a stopping criterion for the subgradient method on algorithm 2 that depends on the cost function H.

A final remark should be done. Any stopping criterion based on the fact that the norm of the subgradient $g^k = A\tilde{x}(\lambda^k) - b$ is smaller than some threshold, wouldn't work very well in this case. This is so, because the values that the entries of $\tilde{x}$ take are extremal values, and don't give a good indication that we are close to find a solution of the linear system $Ax = b$.

Future research. We can pose the following question: is there a way of updating the bound R for the ℓ_∞-norm of the optimal solution x^*, during the subgradient method phase of algorithm 2? Apparently, the answer to this question seems simple: if during this phase of the algorithm we can calculate an $\hat{x}$ that satisfies $A\hat{x} = b$ and $\|\hat{x}\|_\infty < R$, then we can take a new R equal to $n\|\hat{x}\|_\infty + \epsilon$, where ϵ is a small positive number (of course, if the number $n\|\hat{x}\|_\infty + \epsilon$ is smaller than the previous value of R). However, the subgradient method phase of the algorithm 2 doesn't produce any feasible point (probably unless if it has already converged). An updating of R can increase the speed of convergence of the algorithm and perhaps reduce the rippling that we have talked about. Figure 2.10 shows the same inner product $a_i^T\lambda$ of the Figure 2.8, but where we reduced R from 3 to 1.48 (we already new that $\|x^*\|_\infty = 1.4725$). This shows that an updating of R can really decrease the number of needed iterations. Also, notice that a refinement of R also produces a refinement in the bound of the error of the cost function by reducing (2.27).

Another possible topic for future research is to study the behavior of the algorithm if we set a bound $R_i \in \mathbb{R}$ for the absolute value of each coordinate of x^*, instead of setting a global bound for all coordinates (that's what we do when we use the ℓ_∞-norm).

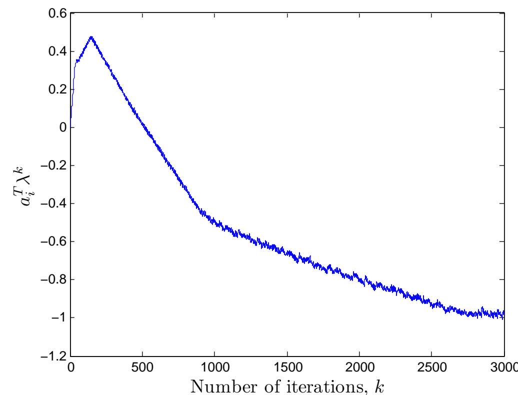


Figure 2.10: Same inner product $a_i^T\lambda^k$ of the Figure 2.8 (missing column), but where we used a value for the bound R of 1.48 instead of 3 (used in the experience that originated Figure 2.8). Note the significant reduction in the number of iterations.

2.2 Multiplier Methods

In this section we present two different algorithms for solving the basis pursuit problem

$$\begin{array}{cc} \min & \|x\|_1, \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{2.28}$$

when the matrix $A \in \mathbb{R}^{m \times n}$ is partitioned horizontally according to the convention used in page 11.

$$A = \underbrace{\begin{bmatrix} \boxed{A_1} & \cdots & \boxed{A_p} & \cdots & \boxed{A_P} \end{bmatrix}}_{n} \updownarrow m$$

Each block matrix A_p, $p = 1, 2, \ldots, P$, contains n_p consecutive columns of the matrix A. Obviously, the numbers n_p must verify $n = n_1 + n_2 + \ldots + n_P$.

2.2.1 Method of Multipliers As An External Loop

By partitioning the variable x into $(x_1, x_2, \ldots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, P$, we can write (2.28) as

$$\min_{A_1x_1 + A_2x_2 + \ldots + A_Px_P = b} \|x_1\|_1 + \|x_2\|_1 + \ldots + \|x_P\|_1. \tag{2.29}$$

The variable is now $(x_1, x_2, \ldots, x_P) \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \ldots \times \mathbb{R}^{n_P}$.

We will try to solve (2.28), with the format of (2.29), through its dual again. If we proceeded the calculation of the Lagrangian dual of (2.29), we would find a polyhedral constraint there. In order to avoid this constraint, we will not use the ordinary Lagrangian, but the *augmented Lagrangian* instead. Recall the discussion on page 12, where we stated that one way of avoiding this polyhedral constraint was to transform the dual Lagrangian function into a coercive one.

To see how we get the augmented Lagrangian, note that problem (2.28) is equivalent to

$$\min_{Ax=b} \|x\|_1 + \frac{\rho}{2}\|b - Ax\|^2, \tag{2.30}$$

where ρ is a positive scalar parameter. The augmented Lagrangian L_a of problem (2.28) is actually the ordinary Lagrangian of (2.30). Let's now calculate this augmented Lagrangian but with the variable decomposition of (2.29): $L_a : \mathbb{R}^{n_1} \times \ldots \mathbb{R}^{n_P} \times \mathbb{R}^m \longrightarrow \mathbb{R}$,

$$\begin{aligned} L_a(x_1, \ldots, x_P, \lambda) &= \sum_{p=1}^{P} \|x_p\|_1 + \lambda^T b - \lambda^T \left(\sum_{p=1}^{P} A_p x_p \right) + \frac{\rho}{2}\|b - \sum_{p=1}^{P} A_p x_p\|^2 \\ &= \lambda^T b + \sum_{p=1}^{P} (\|x_p\|_1 - \lambda^T A_p x_p) + \frac{\rho}{2}\|b - \sum_{p=1}^{P} A_p x_p\|^2. \end{aligned} \tag{2.31}$$

We must have always in mind that, considering $x = (x_1, x_2, \ldots, x_P)$, (2.31) is equivalent to

$$L'_a(x, \lambda) = \lambda^T b + \|x\|_1 - \lambda^T Ax + \frac{\rho}{2}\|b - Ax\|^2, \tag{2.32}$$

where $L'_a : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$.

Notice that (2.32) is coercive with respect to the primal variable x, *i.e.*, for a fixed λ. To see that, write $L'_a(x, \lambda)$ as

$$L'_a(x, \lambda) = \|x\|_1 + \phi(b - Ax),$$

where $\phi(z) = \lambda^T z + (\rho/2)\|z\|^2$. Since the hessian $\nabla^2 \phi(z) = \rho I_m$ is positive definite (I_m is the matrix identity in $\mathbb{R}^m$), there exists a finite m such that $\phi(z) \geq m$ for any $z \in \mathbb{R}^m$. Therefore,

$$\begin{aligned} L'_a(x, \lambda) &\geq \|x\|_1 + m \\ &\geq \|x\| + m, \end{aligned}$$

which tends to $+\infty$ as $\|x\| \to +\infty$, for a fixed λ. This proves the coercivity of $L'_a(x, \lambda)$ with respect to x.

As stated on [3, page 388], there are two mechanisms by which the minimization of (2.32) in the variable x can yield points close to the optimal point of (2.28), $x^\star$: one is by taking λ close to the optimal dual variable $\lambda^\star$; and the other is by taking a very large ρ.

Method of multipliers. However, there is an algorithm, known as the *method of multipliers* ([4, page 244] and [3, page 398]), that allows us to find simultaneously the optimal primal and dual variables, $x^\star$ and $\lambda^\star$, in an iterative way. In fact, this algorithm integrates both mechanisms.

Algorithm 3 (Method of Multipliers)**.**
Consider a convex and coercive function $f : \mathbb{R}^n \to \mathbb{R}$; a real–valued matrix $A \in \mathbb{R}^{m \times n}$; a vector $b \in \mathbb{R}^m$ and the optimization problem

$$\begin{array}{cc} \min & f(x). \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{P}$$

Let $L^f_a : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be the augmented Lagrangian of (P) *with parameter ρ. Then, the* method of multipliers *is well–defined and consists on:*

Initialization

- $\lambda^0 \in \mathbb{R}^m$*;*
- *two real numbers $\rho^0 > 0$ and $c \geq 1$;*
- $k = 0$.

Step 1 *Fix λ^k, and find*

$$x^k \in \arg\min_x L^f_a(x, \lambda^k). \tag{2.33}$$

Step 2 *Update λ^k and ρ^k,*

$$\lambda^{k+1} = \lambda^k + \rho^k \left(b - Ax^k\right), \qquad \rho^{k+1} = c\rho^k.$$

Step 3 $k \leftarrow k + 1$*; and return to Step 1.*

When we say that the method of multipliers is well–defined, we mean that there exists at least one solution of (2.33). This is due to the coercivity and convexity of the augmented Lagrangian $L^f_a(x, \lambda^k)$ in x for any value of λ^k. The coercivity of $L^f_a(x, \lambda^k)$ can be proved using similar arguments to those used for proving that $L'_a(x, \lambda)$ is also coercive. Note that the sum of coercive functions is also coercive. It can also be proved that a coercive and continuous[5] function has always a global minimizer.

[5]Note that a convex function finite everywhere is continuous everywhere.

In the algorithm, note that the parameter of the augmented Lagrangian is ρ^k, and is being updated in every iteration, although we don't refer it explicitly in the expression of $L_a^f(x, \lambda^k)$.

The convergence of the method of multipliers is established by

Theorem 4 (Convergence of the method of multipliers)**.**

- *Every accumulation point of the sequence* $\{x^k\}$ *generated by the* method of multipliers *solves* (P)*; and every accumulation point of the sequence* $\{\lambda^k\}$ *generated by the same algorithm converges to some optimal dual solution of* (P)*.*
- *Furthermore, the objective values of those sequences also converge to their optimum values.*

The proof of this theorem can be based on the fact that the method of multipliers is, in fact, equivalent to another method, called the *proximal method* [4, page 244], for which convergence can be proved [4, page 233].

Considering the stopping criterion of the method of multipliers, we can consider at least three:

- if at the iteration k we have $\|b - Ax^k\| \leq \epsilon^K$ for some predefined ϵ^K (*feasibility criterion*);
- if the dual variable didn't change more than a predefined ϵ from the previous iteration: $\|\lambda^k - \lambda^{k-1}\| \leq \epsilon$;
- or if the cost function L_a^f hasn't been decreasing too much (predefined value) for some iterations.

We will use preferably the feasibility criterion.

Application to our problem. So far, we have seen how to find the optimal primal and dual variables of the problem (2.29). Nonetheless, we transformed this problem into a sequence of successive minimizations of the augmented Lagrangian (2.31) in the variables $x_1, x_2, \ldots, x_P$. If the main problem of (2.29) was the existence of the coupling term $\sum_{p=1}^{P} A_p x_p = b$, we haven't solved this problem yet. In fact, (2.31) is not separable into P independent functions, as we wished. Any x_i is always coupled to x_j, for $j \neq i$, by the quadratic term $(\rho/2)\|b - \sum_{p=1}^{P} A_p x_p\|^2$. However, the positive aspect of the approach taken is that the domain of the augmented Lagrangian (2.32), for any fixed λ, is the full space. Recall that the domain of a function is the set of inputs which do not evaluate to infinity. This fact allowed us to avoid any kind of polyhedral constraints.

Now, we are going to see two well–known methods that can minimize the augmented Lagrangian (2.31) in a separable way, for each variable x_p, $p = 1, 2, \ldots, P$. These methods are known to converge for differentiable functions, thus those results don't apply to (2.31), as this function isn't differentiable. Even though, we will prove that they still converge in this case.

Throughout the next subsections we will always refer to the dual variable as λ^k. We do so to emphasize that the minimization of the augmented Lagrangian (2.31) is only a single step of the method of multipliers.

2.2.2 Nonlinear Jacobi Approach

In order to minimize (2.31) in each variable x_p, for $p = 1, 2, \ldots, P$, we define the function

$$
\begin{aligned}
L_a^p(x, y_p) &= L_a(x_1, \ldots, y_p, \ldots, x_P, \lambda^k) \\
&= \|y_p\|_1 - {\lambda^k}^T A_p y_p - {\lambda^k}^T \sum_{i \neq p} A_i x_i + \frac{\rho}{2}\|b - A_p y_p - \sum_{i \neq p} A_i x_i\|^2. \qquad (2.34)
\end{aligned}
$$

We consider this function only for notational purposes: when we say that we are minimizing $L_a^p(x, y_p)$ with respect to y_p, we are actually referring to the minimization of the augmented Lagrangian (2.31) with respect to x_p, while keeping the other block variables,

$$x_1, \dots, x_{p-1}, x_{p+1}, \dots, x_P,$$

and λ^k constant (with the values of the previous iteration).

Note that we dropped the dependence of λ^k (and ρ^k) in the definition; and we did that just for simplicity of notation.

The algorithm we propose for the minimization of (2.31), in the variables x_p for $p = 1, 2 \dots, P$, is a kind of a *Nonlinear Jacobi* algorithm [4, page 207] and is usually named *Diagonal Quadratic Approximation Method* [29, 24], or simply DQA. The DQA differs from the ordinary Nonlinear Jacobi in the updating of the variables (it is done in a filtering way, with a kind of an exponential moving average filter).

The DQA algorithm. To present the DQA algorithm, let $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ be a convex and coercive function. Decompose a variable $x \in \mathbb{R}^n$ into P subvariables in $\mathbb{R}^{n_p}$, *i.e.* $x = (x_1, x_2, \dots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \dots, P$. Using a notation similar to (2.34), define for each $p = 1, 2, \dots, P$

$$f_p(x, y_p) = f(x_1, \dots, x_{p-1}, y_p, x_{p+1}, \dots, x_P).$$

As the DQA is an iterative algorithm, we will index each variable involved in an iteration of this algorithm by the superscript t.

Algorithm 4 (DQA)**.**

Initialization

- $x^0 \in \mathbb{R}^n$;
- *Fix* τ*, a real number such that* $0 < \tau \leq 1/P$*, where* P *is the number of subvariables of* x*;*
- $t = 0$.

Step 1 *Find*

$$y_p^t \in \arg \min_{y_p \in \mathbb{R}^{n_p}} f_p(x^t, y_p), \tag{2.35}$$

for all $p = 1, \dots, P$.

Step 2 *Form the vector* $y^t = (y_1^t, y_2^t, \dots, y_P^t)$.

Step 3 *Update* x^t*,*

$$x^{t+1} = x^t + \tau(y^t - x^t). \tag{2.36}$$

Step 4 $t \leftarrow t + 1$*; and return to Step 1.*

Again, this algorithm is well–defined for coercive and convex functions, since (2.35) has always at least one solution in this case; thus, DQA is also well–defined if we apply it to $L_a^p(x, y_p)$, at each iteration of the method of multipliers.

A possible stopping criterion for this algorithm can be based on either the cost function (if it doesn't decrease after some iterations) or on the equation (2.36) (for example, $\|y^t - x^t\| \leq \epsilon^T$ for some small ϵ^T).

While it's proved to converge for differentiable functions [29], we don't have any guarantees of convergence of the DQA algorithm if we apply it to $L_a^p(x, y_p)$. It is easy to see that the term responsible for the non–differentiability of (2.34) is the ℓ_1-norm, which is, however, subdifferentiable. Nevertheless, we will prove the convergence of the DQA for a special kind of non–differentiable convex functions.

Convergence and rigid functions. The main reason why we can't apply the convergence results of DQA for differentiable convex functions to a subdifferentiable convex function $f : \mathbb{R}^n \to \mathbb{R}$ is the fact that, for the latter, a point x^* that minimizes f along all the coordinate directions[6] might not be a global minimizer of f. Formalizing, if d_i is a subgradient of f at a point x^* along the coordinate direction e_i, *i.e.*

$$f(x_1^\star, \ldots, u_i, \ldots, x_n^\star) \geq f(x_1^\star, \ldots, x_i^\star, \ldots, x_n^\star) + d_i^T(u_i - x_i^\star) \qquad \forall_{u_i \in \mathbb{R}},$$

then there is no guarantee that the vector $[d_1, \ldots, d_n]^T$, formed by the subgradients of f at $x^\star$ along all the coordinate directions, is a subgradient of f at that point.

Note that the same doesn't apply to the differentiable functions. If $f : \mathbb{R}^n \to \mathbb{R}$ is a differentiable convex function, and is minimized at the point $x^\star$, along all the coordinate directions, we have

$$\left.\frac{\partial f(x)}{\partial x_i}\right|_{x=x^\star} = 0, \qquad \text{for all } i = 1, 2, \ldots, n. \tag{2.37}$$

In fact, (2.37) is equivalent to $\nabla f(x^\star) = 0$, which means that $x^\star$ is a global minimizer of f.

We will designate the functions for which this happens as *rigid functions*[7].

Definition 2 (Rigid Function)**.** *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a subdifferentiable and continuous function over $\mathbb{R}^n$. We say that f is a* rigid function *if, for any point $x^\star \in \mathbb{R}^n$, we have*

$$0 \in \arg\min_h f(x^\star + he_i), \ \forall_{i=1,\ldots,n} \quad \Longrightarrow \quad 0 \in \partial f(x^\star), \tag{2.38}$$

where e_i is a canonical direction in $\mathbb{R}^n$.

If $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function, then there is an alternative way of writing that a point $x^\star$ minimizes f, *i.e.* $0 \in \partial f(x^\star)$. In particular, $x^\star$ is a global minimizer of f if and only if the directional derivative[8] of f at $x^\star$ is non–negative for all directions, *i.e.* if $f : \mathbb{R}^n \to \mathbb{R}$ is convex, then

$$x^\star \text{ is a global minimizer of f} \qquad \Longleftrightarrow \qquad f'(x^\star; v) \geq 0, \quad \forall_{v \in \mathbb{R}^n}. \tag{2.39}$$

Note that every directional derivative of a convex function over $\mathbb{R}^n$ is guaranteed to exist [3, page 709].

Although every convex and differentiable function is rigid, there are subdifferentiable convex functions that aren't rigid. In appendix A an example is provided. However, when the non-differentiability is only due to an adding term like $\|x\|_1$, there is always rigidity, as the following lemma proves.

Lemma 3. *Every function $f : \mathbb{R}^n \to \mathbb{R}$ of the type*

$$f(x) = g(x) + \|x\|_1,$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is a convex and differentiable function, is a rigid function.

Proof. f is obviously subdifferentiable and convex. Let $x^\star$ be a point for which $0 \in \arg\min_h f(x^\star + he_i)$, for $i = 1, 2, \ldots, n$, where e_i is a canonical direction in $\mathbb{R}^n$. This is equivalent to saying that, for all

[6] In an n-dimensional space, we designate a vector of the form $e_i = (0, \ldots, 1, \ldots, 0)$, where it is zero everywhere except at position i, where it takes the value 1, by a *canonical direction/vector* or *coordinate direction/vector*.

[7] The concept of a rigid function can also be developed for constrained minimization. As a consequence, all the results presented on this subsection can be easily generalized for that type of minimization.

[8] We define the *directional derivative* of a function $f : \mathbb{R}^n \to \mathbb{R}$ at the point x and along the direction v as

$$f'(x; v) = \lim_{h \downarrow 0} \frac{f(x + hv) - f(x)}{h},$$

whenever that limit exists.

$i = 1, 2, \ldots, n,$

$$f'(x^\star; he_i) \geq 0, \quad \forall_{h \in \mathbb{R}}. \tag{2.40}$$

Before proceeding, let's examine the directional derivative of the modulus function $n : \mathbb{R} \to \mathbb{R}$, given by $n(x) = |x|$, at an arbitrary point x.
When $x > 0$,

$$n'(x; v) = \lim_{h \downarrow 0} \frac{|x + hv| - |x|}{h} = \lim_{h \downarrow 0} \frac{x + hv - x}{h} = v;$$

when $x < 0$,

$$n'(x; v) = \lim_{h \downarrow 0} \frac{|x + hv| - |x|}{h} = \lim_{h \downarrow 0} \frac{-x - hv + x}{h} = -v.$$

So, $n'(x; v) = \frac{x}{|x|} v$, for $x \neq 0$. Finally, when $x = 0$,

$$n'(0; v) = \lim_{h \downarrow 0} \frac{|hv|}{h} = \lim_{h \downarrow 0} \frac{|h||v|}{h} = |v|.$$

Let $\mathcal{C}$ be the index set of the null coordinates of $x^\star$, *i.e.* $\mathcal{C} = \{i \,:\, x_i^\star = 0\}$. Let's evaluate the directional derivative of f at the point $x^\star$, along an arbitrary direction $v \in \mathbb{R}^n$.

$$\begin{aligned}
f'(x^\star; v) &= \lim_{h \downarrow 0} \frac{g(x^\star + hv) + \|x^\star + hv\|_1 - g(x^\star) - \|x^\star\|_1}{h} \\
&= \lim_{h \downarrow 0} \frac{g(x^\star + hv) - g(x^\star)}{h} + \lim_{h \downarrow 0} \frac{\|x^\star + hv\|_1 - \|x^\star\|_1}{h} \\
&= \nabla g(x^\star)^T v + \lim_{h \downarrow 0} \frac{\sum_{i=1}^n \left[|x_i^\star + hv_i| - |x_i^\star|\right]}{h} \\
&= \nabla g(x^\star)^T v + \sum_{i=1}^n n'(x_i^\star; v_i) \\
&= \nabla g(x^\star)^T v + \sum_{i \notin \mathcal{C}} \frac{x_i^\star}{|x_i^\star|} v_i + \sum_{i \in \mathcal{C}} |v_i| \\
&= \sum_{i \notin \mathcal{C}} v_i (\nabla g(x^\star)^T e_i + \frac{x_i^\star}{|x_i^\star|}) + \sum_{i \in \mathcal{C}} (v_i \nabla g(x^\star)^T e_i + |v_i|) \qquad (2.41) \\
&\geq 0. \qquad (2.42)
\end{aligned}$$

From (2.41) to (2.42), we used the fact that each term in both sums is non–negative. To see this, consider $i \notin \mathcal{C}$. Using (2.41), we get

$$\begin{aligned}
f'(x^\star; he_i) &= h(\nabla g(x^\star)^T e_i + \frac{x_i^\star}{|x_i^\star|}) \\
&\geq 0 \quad \forall_{h \in \mathbb{R}},
\end{aligned}$$

where the inequality follows from (2.40).

In its turn, if $i \in \mathcal{C}$ and we use the same equation, we get

$$\begin{aligned}
f'(x^\star; he_i) &= h \nabla g(x^\star)^T e_i + |h| \\
&\geq 0 \quad \forall_{h \in \mathbb{R}},
\end{aligned}$$

where the inequality is also due to (2.40).

Combining (2.39) with (2.42), we conclude that $x^\star$ is a minimizer of f, hence the implication (2.38) is valid for f. □

Now we are in conditions to prove the convergence of the DQA for a function that is convex, rigid and coercive[9].

Theorem 5 (Convergence of DQA)**.** *Let $f:\mathbb{R}^n \to \mathbb{R}$ be a convex, rigid and coercive function over $\mathbb{R}^n$. Admit the decomposition of a variable $x \in \mathbb{R}^n$ into $(x_1, x_2, \ldots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, P$, with $n = n_1 + n_2 + \ldots + n_P$. Then,*

1. *Every accumulation point of the sequence $\{x^t\}$ generated by the DQA algorithm solves $\min_{x\in\mathbb{R}^n} f(x)$;*
2. *The sequence of objective values converges to the optimum,* i.e., *$f(x^t) \downarrow f^* := \min_{x\in\mathbb{R}^n} f(x)$.*

Proof of part 1: This proof is, up to some simplifications which permit to discard the differentiability of f, the one contained in Proposition 9, page 46, in [29].

In the sequel, we shall use the notations $x^t = (x_1^t, x_2^t, \ldots, x_P^t)$ and $y^t = (y_1^t, y_2^t, \ldots, y_P^t)$.

First, note that y^t can be written as

$$y^t = \sum_{p=1}^{P} (x_1^t, x_2^t, \ldots, y_p^t, \ldots, x_P^t) - (P-1)x^t. \tag{2.43}$$

Inserting (2.43) into $x^{t+1} = x^t + \tau(y^t - x^t)$ yields

$$x^{t+1} = \tau \sum_{p=1}^{P} (x_1^t, x_2^t, \ldots, y_p^t, \ldots, x_P^t) + (1-\tau P)x^t. \tag{2.44}$$

It follows that

$$\begin{aligned} f(x^{t+1}) &= f\left(\tau \sum_{p=1}^{P} (x_1^t, x_2^t, \ldots, y_p^t, \ldots, x_P^t) + (1-\tau P)x^t\right) \\ &\le \tau \sum_{p=1}^{P} f(x_1^t, x_2^t, \ldots, y_p^t, \ldots, x_P^t) + (1-\tau P) f(x^t) &(2.45) \\ &= \tau\left(\sum_{p=1}^{P} f_p(x^t, y_p^t) - f(x^t)\right) + f(x^t) &(2.46) \\ &\le f(x^t). &(2.47) \end{aligned}$$

Inequality (2.45) is due to the convexity of f and $0 < \tau \le 1/P$. Inequality (2.47) follows from $f_p(x^t, y_p^t) \le f(x^t)$, due to the definition of y_p^t.

So far, we have shown that the sequence $\{f(x^t)\}$ is monotonically non–increasing.

Now, define the sublevel set $S_x = \{x \in \mathbb{R}^n \,:\, f(x) \le f(x^0)\}$. Due to the continuity (a convex function finite everywhere is continuous everywhere) and coercivity of f, this set is compact. Furthermore, as $\{f(x^t)\}$ is monotonically non–increasing, the whole sequence $\{x^t\}$ is contained in S_x.

From $x^{t+1} = x^t + \tau(y^t - x^t)$, we have $\|y^t - x^t\| = (1/\tau)\|x^{t+1} - x^t\|$. As the sequence $\{x^t\}$ is contained in S_x, the distance $\|x^{t+1} - x^t\|$ is bounded for any sequence $\{t\}$. Thus, because τ is a fixed parameter, $\|y^t - x^t\|$ is also bounded. This means that there exists a compact set S_y that contains the whole sequence $\{y^t\}$. Let $S = S_x \cup S_y$. S is also compact and contains both sequences $\{x^t\}$ and $\{y^t\}$.

[9] If we were using the concept of rigidity for constrained minimization in a compact set, the coercivity could be dropped.

Let x^* be an accumulation point of the sequence $\{x^t\}$. Also, let $\{t_j\}$ be a sequence of indices such that $x^{t_j} \to x^*$ and $y^{t_j} \to y^*$, where y^* is an accumulation point of $\{y^t\}$. The existence of such $\{t_j\}$ is guaranteed by the compactness of the set S.

Choose an arbitrary $z = (z_1, z_2, \ldots, z_P)$ in $\mathbb{R}^n$. There holds

$$f_p(x^{t_j}, y_p^{t_j}) \leq f_p(x^{t_j}, z_p) \tag{2.48}$$

because, by definition, $y_p^{t_j}$ solves

$$\min_{w_p \in \mathbb{R}^{n_p}} \; f_p(x^{t_j}, w_p).$$

Taking $j \to +\infty$ in (2.48) yields

$$f_p(x^*, y_p^*) \leq f_p(x^*, z_p). \tag{2.49}$$

Since $z_p \in \mathbb{R}^{n_p}$ can be arbitrary, inequality (2.49) shows that y_p^* solves

$$\min_{w_p \in \mathbb{R}^{n_p}} \; f_p(x^*, w_p). \tag{2.50}$$

In the remaining part of the proof, we will show that x_p^* also solves (2.50). For that, note that $t_{j+1} \geq t_j + 1$ and recall that $\{f(x^t)\}$ is monotonically non–increasing. Thus,

$$f(x^{t_{j+1}}) \leq f(x^{t_j+1}). \tag{2.51}$$

Concatenating (2.51) and (2.46) yields

$$f(x^{t_{j+1}}) \leq \tau \left(\sum_{p=1}^{P} f_p(x^{t_j}, y_p^{t_j}) - f(x^{t_j}) \right) + f(x^{t_j}). \tag{2.52}$$

Taking the limit $j \to +\infty$ in (2.52) leads to

$$f(x^*) \leq \tau \left(\sum_{p=1}^{P} f_p(x^*, y_p^*) - f(x^*) \right) + f(x^*),$$

or, equivalently,

$$\sum_{p=1}^{P} f_p(x^*, y_p^*) - f(x^*) \geq 0. \tag{2.53}$$

Joining $f_p(x^*, y_p^*) \leq f(x^*)$ with (2.53) yields $f_p(x^*, y_p^*) = f(x^*)$. That is, $f_p(x^*, y_p^*) = f_p(x^*, x_p^*)$, which proves that x_p^* solves (2.50).

Since f was assumed to be rigid and convex, it follows that x^* solves $\min_{x \in \mathbb{R}^n} \; f(x)$. □

Proof of part 2**:** We have seen in part 1 that the whole sequence $\{x^t\}$ is contained in a compact sublevel set $S_x = \{x \in \mathbb{R}^n \,:\, f(x) \leq f(x^0)\}$. Since f is continuous, it follows that $\{f(x^t)\}$ is bounded below. Also in part 1, it was proved that the sequence $\{f(x^t)\}$ is monotonically non–increasing. Thus, the sequence $\{f(x^t)\}$ converges, say to $\overline{f}$, *i.e.*, $f(x^t) \downarrow \overline{f}$ as $t \to +\infty$.

Now, since S_x is compact, the sequence $\{x^t\}$ has an accumulation point, say x^*. According to part 1, the point x^* is a global minimizer of f over $\mathbb{R}^n$, and obviously over S_x. Thus, $f(x^*) = f^* = \min_{x \in S_x} \; f(x)$. Let $\{x^{t_j}\}$ be a sub-sequence of $\{x^t\}$ converging to x^*. Because f is continuous, there holds $f(x^{t_j}) \to f(x^*)$ as $j \to +\infty$. But, since the whole sequence $\{f(x^t)\}$ is convergent to $\overline{f}$, we also have $f(x^{t_j}) \to f^*$. Thus, $\overline{f} = f^*$ and $f(x^t) \downarrow f^*$. □

Application to our problem. So, theorem 5, together with lemma 3, guarantees the convergence of the DQA algorithm when applied to the augmented Lagrangian $L_a(x_1,\ldots,x_P,\lambda^k)$, defined in (2.31), for an iteration k of the method of multipliers.

Note that the DQA is totally parallelizable, allowing each processor p to contribute to the calculation of the solution of (2.33), in the step 1 of the method of multipliers. This way, we can find the optimal primal and dual variables of the basis pursuit (2.28), in a parallelized way.

However, in each inner iteration t of the DQA, in step 1, each processor p has to solve

$$\min_{y_p} L_a^p(x_1^t,\ldots,y_p,\ldots,x_P^t), \tag{2.54}$$

where

$$L_a^p(x_1^t,\ldots,y_p,\ldots,x_P^t) = \|y_p\|_1 - {\lambda^k}^T A_p y_p - {\lambda^k}^T \sum_{i\neq p} A_i x_i^t + \frac{\rho^k}{2}\|b - A_p y_p - \sum_{i\neq p} A_i x_i^t\|^2,$$

for $p = 1,\ldots,P$.

It happens that (2.54) can be recast as basis pursuit denoising (BPDN) problem. Recently, efficient software that can solve a BPDN problem in a fast way has come up (see [16, 20] for example).

By doing $d_p^t = b - \sum_{j\neq p} A_j x_j^t$, (2.54) is equivalent to

$$\min_{y_p} \|y_p\|_1 - {\lambda^k}^T A_p y_p + \frac{\rho^k}{2}\|d_p^t - A_p y_p\|^2. \tag{2.55}$$

And if we develop the quadratic term,

$$\frac{\rho^k}{2}\|d_p^t - A_p y_p\|^2 = \frac{\rho^k}{2} y_p A_p^T A_p y_p - \rho^k {d_p^t}^T A_p y_p + \frac{\rho^k}{2}\|d_p^t\|^2,$$

we can see that (2.55) is equivalent to

$$\min_{y_p} \frac{1}{2}\|A_p y_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|y_p\|_1, \tag{2.56}$$

where $\gamma_p^{t,k} = d_p^t + \lambda^k/\rho^k$.

Multipliers/DQA Chaining the method of multipliers with DQA, and adapting them to solve (2.28), yields algorithm 5. Besides all the P processors, a central processor is needed. The architecture of the links is the same as in Figure 2.2. Actually, this architecture isn't the only possible one, but it seems that it is the one for which this algorithm runs faster. Note that the algorithm has two loops: the exterior one, indexed to k, performs the method of multipliers; and the interior one, indexed to t, executes the DQA algorithm.

The exterior loop usually takes a few iterations to converge. A good practice is to decrease the parameter ϵ_2, from the inner stopping criterion, in each external iteration, making it moderate at the beginning and small at the last iterations. In fact, we don't need an high accuracy at the first iterations, so decreasing ϵ_2 makes all the sense. Concerning the interior loop, this one takes usually more iterations to converge than the external loop. In most of the algorithms that solve a BPDN, we can usually benefit from using the solution found in the previous iteration as a starting point to the next one. To implement this in algorithm 5, it suffices that all the processors keep in memory the previous y_p. Note that all the complexity is at the P processors, being the algorithm for the central processor relatively simple. This is opposed to what happens in algorithm 2, subsection 2.1 (Subgradient method for bounded BP), which

uses the same architecture for the links. In fact, this unit only has to do sums, multiplications and some binary operations.

Algorithm 5 (Multipliers/DQA)**.**

- *Predefined Parameters/Initialization:*
  - A_p *for each processor,* $p = 1, \ldots, P$*;*
  - *Choose* $c \geq 1$ *(for actualizing* ρ^k*);*
  - *Choose* $0 < \tau \leq 1/P$*;*
  - *Choose* K *and* T *as the maximum number of iterations; and* ϵ^K*,* ϵ^T *for the stopping criteria;*
  - *Choose* $x^0 \in \mathbb{R}^n$*,* $\lambda^0 \in \mathbb{R}^m$ *and* $\rho^0 \in \mathbb{R}_+$*; note the decomposition* $x^t = (x_1^t, x_2^t, \ldots, x_P^t)$*;*
- *Procedure (for central processor):*
  - *Receive* b *(from outside);*
  - *For* $k = 0$ *until* K [Method of multipliers]*:*
    * *For* $t = 0$ *until* T [DQA]*:*
      1. §1 *Send* x_p^t *to each processor* $p = 1, \ldots, P$*;*
      2. §1 *Receive* $A_p x_p^t$ *from each processor;*
      3. *Compute* $d_p^t = b - \sum_{j \neq p} A_p x_p^t$*, for* $p = 1, \ldots, P$*.*
      4. §2 *Send* $\gamma_p^{t,k} = d_p^t + \lambda^k / \rho^k$ *to each processor* $p = 1, \ldots, P$*;*
      5. §2 *Collect* y_p *from each processor;*
      6. *Form* $y = (y_1, y_2, \ldots, y_P)$*;*
      7. *Stopping criterion: if* $\|x^t - y\| \leq \epsilon^T$*, break cycle;*
      8. $x^{t+1} = x^t + \tau\,(y - x^t)$*; and* $t \leftarrow t + 1$*.*
    * *Make* $x^k = x^t$ *for the last* t*;*
    * *Evaluate the sum* $Ax^k = \sum_{p=1}^{P} A_p x_p^k$ *(note that the vectors* $A_p x_p^k$ *have been previously calculated);*
    * *Stopping criterion: if* $\|b - Ax^k\| \leq \epsilon^K$*, stop the algorithm;*
    * $\lambda^{k+1} = \lambda^k + \rho^k \left(b - Ax^k\right)$*;* $\rho^{k+1} = c\rho^k$*; and* $k \leftarrow k + 1$*.*
- *Procedure (for each processor* $p = 1, \ldots, P$*): There are two modes (referenced above as §1 and §2):*
  - *Mode 1:*
    1. *Receive* x_p^t *and return the product* $A_p x_p^t$*.*
  - *Mode 2:*
    1. *Receive* $\gamma_p^{t,k}$*;*
    2. *Solve* $y_p = \arg\min_{w_p} (1/2)\|A_p w_p - \gamma_p^{t,k}\|^2 + (1/\rho^k)\|w_p\|_1$*, and return* y_p*.*

In each iteration, each processor exchanges with the central processor vectors of the size n in steps 1 and 5 (of the central processor algorithm) and of the size m in steps 2 and 4. Therefore, in each iteration $2(m+n)$ numbers are exchanged between any processor and the central node. Recall that in algorithm 2 (Subgradient method for bounded BP) only $2(m+1)$ numbers were exchanged in the iterative part of the algorithm. However, it happens that algorithm 2 requires usually much more iterations than algorithm 5 to converge.

The stopping criteria used in this algorithm serves merely as an example, and any other one that we have mentioned before for both the method of multipliers and the DQA could be easily implemented.

Note that theorems 4 and 5 and lemma 3 together guarantee that this algorithm converges to an optimal solution of (2.28).

2.2.3 Nonlinear Gauss–Seidel Approach

The DQA algorithm is a type of a Nonlinear Jacobi algorithm, where all the block variables are calculated at the same time. This means that, at each iteration, processor p finds the minimizer y_p of (2.34), for $p = 1, \ldots, P$. Consequently, no communications between the P processors are needed. Indeed, every processor has to communicate only with the central processor.

However, if the P processors are able to communicate with each other, an alternative algorithm, known as the *Nonlinear Gauss–Seidel* algorithm [4], can be used. This algorithm is known to be faster than the Jacobi algorithm, clearly benefiting from the communications between the processors. So, it is expected that it also performs faster than the DQA. Furthermore, with the Nonlinear Gauss–Seidel algorithm we can discard the central processor if we communicate the relevant information in a circular way. Nevertheless, the use of the Nonlinear Gauss–Seidel algorithm on non–differentiable functions is not always successful [4, page 209].

In this subsection, we will prove that the Nonlinear Gauss–Seidel algorithm converges for functions that are convex, rigid, coercive[10], and also have unique block coordinate minimizers; and use it to solve problem (2.33), *i.e.* the step 1 of the method of multipliers

$$x^k = \arg\min_{x\in X} L_a^f(x, \lambda^k), \tag{2.57}$$

for each iteration k. The original problem that we want to solve is the BP, so the augmented Lagrangian in (2.57) is

$$L_a(x, \lambda^k) = \|x\|_1 + {\lambda^k}^T b - {\lambda^k}^T Ax + \frac{\rho^k}{2}\|b - Ax\|^2. \tag{2.58}$$

Restrictions on the partition. This time, we have to make restrictions on the horizontal partition of the matrix $A \in \mathbb{R}^{m\times n}$ into P matrices A_p, with $A_p \in \mathbb{R}^{m\times n_p}$, for $p = 1, 2, \ldots, P$ and with $n = n_1 + n_2 + \ldots + n_P$. Namely, we assume that each block A_p has full column–rank. This imposes automatically that $n_p \leq m$.

In the context of the applications, this restriction is realistic either in cases where A is a random matrix (provided that n_p is smaller than m), or in cases where A is a dictionary of functions (here, it doesn't make much sense to have linear dependence inside a small set of a family of functions). In any case, if the number of available processors is less than the needed, it is always possible to simulate, say, two processors inside just one: if A_p can be partitioned into two full column–rank submatrices, then each time an operation for A_p is required, do the operations for one of those submatrices and then for the other.

The Nonlinear Gauss–Seidel algorithm. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex coercive function over $\mathbb{R}^n$. Consider the decomposition of a variable $x \in \mathbb{R}^n$ into $(x_1, x_2, \ldots, x_P)$, where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, P$, with $n = n_1 + n_2 + \ldots + n_P$. Then, the Nonlinear Gauss–Seidel algorithm consists on

Algorithm 6 (Nonlinear Gauss–Seidel)**.**

Initialization

[10] As the footnote on page 35 says, the same result would hold if we were dealing with a minimization over a compact set. In this case, the coercivity could be dropped.

- $x^0 \in \mathbb{R}^n$;
- $t = 0$.

Step 1 *For all* $p = 1, 2, \ldots, P$, *find*

$$x_p^{t+1} \in \arg \min_{x_p \in \mathbb{R}^{n_p}} f(x_1^{t+1}, \ldots, x_{p-1}^{t+1}, x_p, x_{p+1}^t, \ldots, x_P^t). \tag{2.59}$$

Step 2 $t \leftarrow t + 1$; *and return to Step 1.*

Notice that the successive minimizations (2.59) can be performed in an arbitrary order, although we will assume here, for simplicity of notation, that they follow the (circular) order $1, 2, \ldots, P, 1, \ldots$.

The Nonlinear Gauss–Seidel algorithm is well–defined for convex coercive functions in the sense that (2.59) is always solvable.

The stopping criterion of this algorithm can be based again, as the other methods that we have seen, or on the cost function f or on the fact that $\|x^{t+1} - x^t\| \leq \epsilon^T$, for some $\epsilon^T > 0$.

Theorem 6 (Convergence of the Nonlinear Gauss–Seidel Algorithm)**.** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a convex, rigid and coercive function over $\mathbb{R}^n$. Consider the decomposition of a variable $x \in \mathbb{R}^n$ into P blocks,* i.e. $x = (x_1, x_2, \ldots, x_P)$, *where $x_p \in \mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, n$, with $n = n_1 + n_2 + \ldots + n_P$. Further, assume that f has unique block coordinate minimizers, that is, for any $x = (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n$, the optimization problem*

$$\min_{w_p \in \mathbb{R}^{n_p}} f(x_1, \ldots, x_{p-1}, w_p, x_{p+1}, \ldots, x_P)$$

has a unique solution. Then, every limit point of the sequence $\{x^t\}$ generated by the Nonlinear Gauss–Seidel algorithm solves $\min_{x \in \mathbb{R}^n} f(x)$.

Proof. This is essentially the proof in [3] but with a major simplification. Also, the need for the differentiability of f is dropped. We will use the notation $x^t = (x_1^t, x_2^t, \ldots, x_P^t)$.

Define $y^t := x^{t+1} = (x_1^{t+1}, x_2^{t+1}, \ldots, x_P^{t+1})$. By the definition of the algorithm, the sequence $\{f(x^t)\}$ is non–increasing. So, if S is the sublevel set $\{x \in \mathbb{R}^n : f(x) \leq f(x^0)\}$, then both sequences $\{x^t\}$ and $\{y^t\}$ are all contained in S. This happens because f is coercive, which makes S a compact set.

Then, let x^* be a limit point of the sequence $\{x^t\}$, say $x^{t_j} \to x^*$ as $j \to +\infty$. Without loss of generality, we can assume that $\{y^{t_j}\}$ also converges (otherwise, pass to a convergent subsequence of $\{y^{t_j}\}$, which is possible due to the compactness of S). Let y^* be the limit of $\{y^{t_j}\}$, *i.e.*, $\{y^{t_j}\} \to y^*$ as $j \to +\infty$.

As $t_{j+1} \geq t_j + 1$ and $\{f(x^t)\}$ is non–increasing, we have

$$f(x^{t_{j+1}}) \leq f(x^{t_j+1}). \tag{2.60}$$

Also, by the definition of the algorithm, there holds

$$f(x^{t_j+1}) = f(x_1^{t_j+1}, x_2^{t_j+1}, \ldots, x_P^{t_j+1}) \leq f(x_1^{t_j+1}, x_2^{t_j}, \ldots, x_P^{t_j}), \tag{2.61}$$

and

$$f(x_1^{t_j+1}, x_2^{t_j}, \ldots, x_P^{t_j}) \leq f(x_1, x_2^{t_j}, \ldots, x_P^{t_j}), \tag{2.62}$$

for any $x_1 \in \mathbb{R}^{n_1}$. Concatenating (2.60), (2.61) and (2.62) gives

$$f(x^{t_{j+1}}) \leq f(x_1^{t_j+1}, x_2^{t_j}, \ldots, x_P^{t_j}) \leq f(x_1, x_2^{t_j}, \ldots, x_P^{t_j}), \tag{2.63}$$

for any $x_1 \in \mathbb{R}^{n_1}$. Taking limits in (2.63) yields

$$f(x_1^*, x_2^*, \ldots, x_P^*) \leq f(y_1^*, x_2^*, \ldots, x_P^*) \leq f(x_1, x_2^*, \ldots, x_P^*), \tag{2.64}$$

for any $x_1 \in \mathbb{R}^{n_1}$. Inequality (2.64) shows that both the points x_1^* and y_1^* solve the optimization problem

$$\min_{w_1 \in \mathbb{R}^{n_1}} f(w_1, x_2^*, \ldots, x_P^*).$$

By the assumption on the uniqueness of the block coordinate minimizers, we conclude that $x_1^* = y_1^*$. Thus, in particular, $x_1^{t_j+1} \to x_1^*$.

Reasoning as before, we also have

$$f(x^{t_{j+1}}) \leq f(x_1^{t_j+1}, x_2^{t_j+1}, x_3^{t_j} \ldots, x_P^{t_j}) \leq f(x_1^{t_j+1}, x_2, x_3^{t_j} \ldots, x_P^{t_j}), \tag{2.65}$$

for any $x_2 \in \mathbb{R}^{n_2}$. Taking limits in (2.65) yields

$$f(x_1^*, x_2^*, x_3^*, \ldots, x_P^*) \leq f(x_1^*, y_2^*, x_3^*, \ldots, x_P^*) \leq f(x_1^*, x_2, x_3^*, \ldots, x_P^*), \tag{2.66}$$

for any $x_2 \in \mathbb{R}^{n_2}$. Inequality (2.66) shows that both x_2^* and y_2^* solve

$$\min_{w_2 \in \mathbb{R}^{n_2}} f(x_1^*, w_2, x_3^*, \ldots, x_P).$$

Again, by the assumption on the uniqueness of the block coordinate minimizers, we conclude that $x_2^* = y_2^*$.

Proceeding similarly for $p = 3, \ldots, P$, we have that $x^* = y^*$ and x_p^* is the solution of

$$\min_{w_p \in \mathbb{R}^{n_p}} f(x_1^*, \ldots, x_{p-1}^*, w_p, x_{p+1}^*, \ldots, x_P^*).$$

Since f was assumed to be rigid we conclude that x^* solves $\min_{x \in \mathbb{R}^n} f(x)$. □

Convergence of the method for the augmented Lagrangian (2.58)**.** To make the theorem above applicable to (2.58), the assumption on the uniqueness of the block coordinate minimizers of (2.58) must be checked.

We claim that if each A_p has full column–rank, then (2.58) has unique block coordinate minimizers. To see that, fix $x = (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n$, where each block variable x_p belongs to $\mathbb{R}^{n_p}$, for $p = 1, 2, \ldots, P$; and define $L_a^p : \mathbb{R}^{n_p} \to \mathbb{R}$ as before, *i.e.* $L_a^p(y) = L_a(x_1, \ldots, x_{p-1}, y, x_{p+1}, \ldots, x_P, \lambda^k)$.

We must show that L_a^p has a unique minimizer over $\mathbb{R}^{n_p}$. We will establish this by showing that L_a^p is strictly convex. Indeed, we have

$$L_a^p(y) = \|(x_1, \ldots, y, \ldots, x_P)\|_1 + {\lambda^k}^T (d_p - A_p y) + \frac{\rho^k}{2} \|d_p - A_p y\|^2, \tag{2.67}$$

where we used the definition $d_p := b - \sum_{j \neq p} A_j x_j$. Equation (2.67) can be rewritten as $L_a^p(y) = h_p(y) + f_p(y)$, where

$$h_p(y) = \|(x_1, \ldots, y, \ldots, x_P)\|_1 + {\lambda^k}^T (d_p - A_p y) + \frac{\rho^k}{2} \left(\|d_p\|^2 - 2 d_p^T A_p y\right)$$

and

$$f_p(y) = \frac{\rho^k}{2} y^T A_p^T A_p y.$$

Since A_p has full column–rank, the matrix $A_p^T A_p$ is positive definite. Thus, f_p is a strictly convex function and, as a consequence, $L_a^p = h_p + f_p$ is also a strictly convex function (note that h_p is convex).

This way, with theorem 6 and lemma 3, we guarantee the convergence of the Nonlinear Gauss–Seidel algorithm for (2.58).

Work for each processor. In every iteration of the Nonlinear Gauss–Seidel algorithm, each processor has to solve (2.59), which is a BPDN problem in disguise. Indeed, it turns out that (2.59), when applied to the augmented Lagrangian (2.58), is equivalent to a problem like (2.56), but where the variable $\gamma_p^{t,k}$ is slightly different. To see this, first note that the step 1 of the Nonlinear Gauss–Seidel algorithm applied to (2.58) yields the problem

$$\min_{x_p \in \mathbb{R}^{n_p}} L_a(x_1^{t+1}, \ldots, x_{p-1}^{t+1}, x_p, x_{p+1}^t, \ldots, x_P^t, \lambda^k). \tag{2.68}$$

The objective function of (2.68) can be written as

$$\sum_{i<p} \|x_i^{t+1}\|_1 + \sum_{i>p} \|x_i^t\|_1 + \|x_p\|_1 + {\lambda^k}^T d_p^t - {\lambda^k}^T A_p x_p + \frac{\rho^k}{2}\|d_p^t - A_p x_p\|^2,$$

where $d_p^t = b - \sum_{i<p} A_i x_i^{t+1} - \sum_{i>p} A_i x_i^t$. Hence, (2.68) is equivalent to

$$\min_{x_p \in \mathbb{R}^{n_p}} \|x_p\|_1 - {\lambda^k}^T A_p x_p + \frac{\rho^k}{2}\|d_p^t - A_p x_p\|^2.$$

Now, taking the same steps used in passing from (2.55) to (2.56), we see that this problem is equivalent to

$$\min_{x_p \in \mathbb{R}^{n_p}} \frac{1}{2}\|A_p x_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|x_p\|_1, \tag{2.69}$$

where $\gamma_p^{t,k} = d_p^t + \lambda^k/\rho^k$.

This variable $\gamma_p^{t,k}$, being different from the one defined in subsection 2.2.2 page 37, can be updated from processor $p-1$ to processor p as follows:

$$\gamma_p^{t,k} = b + \frac{\lambda^k}{\rho^k} - \sum_{i<p} A_i x_i^{t+1} - \sum_{i>p} A_i x_i^t.$$

If the processor $p-1$ sends the vector

$$\phi_{p-1} = \gamma_{p-1}^{t,k} - A_{p-1} x_{p-1}^{t+1}$$

to the processor p, this processor can correctly find $\gamma_p^{t,k}$ by doing

$$\gamma_p^{t,k} = \phi_{p-1} + A_p x_p^t,$$

where x_p^t had been calculated in the previous iteration (note that it is x_p^{t+1} that we want to find and that's what problem (2.68) does).

This way, the only information that the processors have to exchange with each other is the vector ϕ_{p-1} (from the processor $p-1$ to the processor p) and the number ρ^k, in a circular way. This works if each processor p keeps in memory the vector x_p^t from the previous iteration. In fact, it can even use it to solve the current problem as a starting point, if the algorithm used to solve (2.69) requires it.

Note that there is no need of a central node or processor for this algorithm. The task of verifying if any of the algorithms has converged (the method of multipliers or the Nonlinear Gauss–Seidel), as well

as the task of updating the fraction λ^k/ρ^k in the vector $\gamma_p^{t,k}$, can be assigned to any of the P processors. The updating of this fraction in $\gamma_p^{t,k}$ can be easily done in any processor p: if it has in memory the previous value of the fraction, $\gamma_p^{t,k}$, it suffices to do

$$\gamma_p^{t,k+1} = \gamma_p^{t,k} - \frac{\lambda^k}{\rho^k} + \frac{\lambda^{k+1}}{\rho^{k+1}}.$$

We assume that both the tasks of checking the stopping criteria and updating the variables are all performed by the processor number 1.

The association of the method of multipliers with the Nonlinear Gauss–Seidel algorithm, both adapted to solve the BP (2.28), and with these details yields algorithm 7. In Figure 2.11 it is plotted the required architecture for the algorithm.

Algorithm 7 (Multipliers/Gauss–Seidel)**.**

- *Predefined Parameters/Initialization:*
  - A_p *for each processor,* $p = 1, \ldots, P$*;*
  - *Choose* $c \geq 1$ *(for actualizing* ρ^k*);*
  - *Choose* K *and* T *as the maximum number of iterations; and* ϵ^K*,* ϵ^T *for the stopping criteria;*
  - *Choose* $x^0 \in \mathbb{R}^n$*,* $\lambda^0 \in \mathbb{R}^m$ *and* $\rho^0 \in \mathbb{R}_+$*; note the decomposition* $x = (x_1, x_2, \ldots, x_P)$*;*
- *for* $k = 0$ *until* K [Method of Multipliers]*:*

  Procedure (for processor 1):

  - *If* $k = 0$*,*
    1. *Receive* b *(from exterior);*
    2. $t = 0$*;*
    3. *Initialize* $\gamma_1^{t,0}$ *(for example,* $\gamma_1^{t,0} = b + \frac{\lambda^0}{\rho^0}$*);*
    4. *Solve* $x_1^1 = \arg\min_{x_1} \frac{1}{2}\|A_1 x_1 - \gamma_1^{t,0}\|^2 + \frac{1}{\rho^k}\|x_1\|_1$*;*
    5. *Send* ρ^0 *and* $\phi_1 = \gamma_1^{t,0} - A_1 x_1^1$ *to processor* 2*.*
  - *If* $k > 0$*,*
    1. *Receive* ϕ_P *from processor* P *(processor* 1 *already knows* ρ*);*
    2. $t \leftarrow t + 1$*;*
    3. *Inner stopping criterion:*
       * *If* $t > T$ *or* $\|x^t - x^{t-1}\| \leq \epsilon^T$*,*
         check external stopping criterion:
         · *If* $k > K$ *or* $\|b - Ax^t\| \leq \epsilon^K$*,* stop the algorithm and return x.
         · *else, update* $\lambda^{k+1} = \lambda^k + \rho^k(b - Ax^t)$*,* $\rho^{k+1} = c\rho^k$*,* $\gamma_1^{t,k+1} = \gamma_1^{t,k} - \frac{\lambda^k}{\rho^k} + \frac{\lambda^{k+1}}{\rho^{k+1}}$*,* $k \leftarrow k + 1$ *and* $t = 0$*.*
       * *else,* $t \leftarrow t + 1$*.*
    4. *Set* $\gamma_1^{t,k} = \phi_P + A_1 x_1^t$*;*
    5. *Solve* $x_1^{t+1} = \arg\min_{x_1} \frac{1}{2}\|A_1 x_1 - \gamma_1^{t,k}\|^2 + \frac{1}{\rho^k}\|x_1\|_1$*;*
    6. *Keep* x_1^{t+1} *in memory;*
    7. *Send* ρ^k *and* $\phi_1 = \gamma_1^{t,k} - A_1 x_1^{t+1}$ *to processor* 2*.*

  Procedure (for processor $p = 2, \ldots, P$):

1. *Receive* ρ^k *and* ϕ_{p-1} *from processor* $p-1$*;*
2. *Set* $\gamma_p^{t,k} = \phi_{p-1} + A_p x_p^t$*;*
3. *Solve* $x_p^{t+1} = \arg\min_{x_p} \frac{1}{2}\|A_p x_p - \gamma_p^{t,k}\|^2 + \frac{1}{\rho^k}\|x_p\|_1$*;*
4. *Keep* x_p^{t+1} *in memory;*
5. *Send* ρ^k *and* $\phi_p = \gamma_p^{t,k} - A_p x_p^{t+1}$ *to processor* $p+1$.

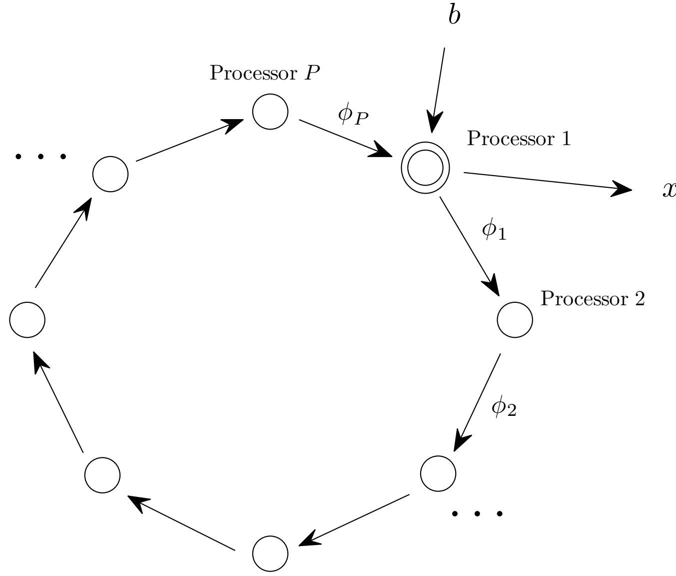


Figure 2.11: Required architecture for the implementation of the algorithm 7. Processor 1 is considered the starting/ending node which receives the vector b, updates the variables λ^k and ρ^k, and checks the stopping criteria. The output of the algorithm is the vector x.

Communications and stopping criteria. In comparison to what was seen for algorithms 2 (Sub-gradient for BP) and 5 (Multipliers/DQA), the communication requirements for this algorithm are less demanding. This is mainly due to the lack of a central processor. In the previous algorithms, in each iteration, all processors had to communicate with the central node. This algorithm, though, only requires a communication of $m+1$ numbers (for all processors) in each inner iteration.

We chose for this algorithm the possible stopping criteria: for the Gauss–Seidel algorithm (inner cycle), this one can stop if the maximum number of iterations is reached, or if the distance of two consecutive iterations, $\|x^t - x^{t-1}\|$, is smaller (or equal) than a constant ϵ^T; for the method of multipliers, it can also stop if the maximum number of iterations is reached or if $\|b - Ax^t\|$ is smaller (or equal) than the constant ϵ^K. Note that this implies that the processor 1 knows both the vectors x and Ax. This is possible if the processors communicate their own vectors x_p and $A_p x_p$ to each other[11], so that processor 1 can play his role. Actually, this can be done in the idle times of the links while they are not transmitting $\gamma_p^{t,k}$ nor ρ^k. In fact, the links are idle most of the time. If it is expensive to use the links, one can also opt for other criteria, such as imposing only the maximum number of iterations, or making circulate the global cost function (one number) and establishing a protocol saying that every processor stops one cycle if the global cost function doesn't decrease more than a predefined small value. Note that it is possible to make circulate the global cost function as long as each processor knows λ^k/ρ^k at each iteration of the method of multipliers.

[11] It is possible to know Ax from this because $Ax = \sum_{p=1}^{P} A_p x_p$, due to the horizontal partition of A.

Robustness to instantaneous link failures. As it is noted on the errata of [3], there is an alternative method to the fixed cyclic order of the Nonlinear Gauss–Seidel algorithm 6. Theorem 6 can be shown to converge for an arbitrary order of iteration, as long as there is an integer M such that at the end of every group of M contiguous iterations every processor has done at least one minimization (that is, found the corresponding x_p for $p = 1, \dots, P$). Thus, this algorithm is quite versatile in the sense that processors don't need to communicate with each other in a predeterminate order. So, if the processors communicate randomly with each other, for instance, this algorithm becomes robust to instantaneous link failures.

Future research. We leave the following topics for future research:

- Is it possible to make a fusion between the DQA and the Nonlinear Gauss–Seidel algorithms, if, for example, we had available a central processor and communications from all to all? What would be the properties of such algorithm? Would it be robust not only to instantaneous link failures, in the case of random communications, but also to permanent link failures?
- The adaptation of the algorithms 2, 5 and 7 to solve the BPDN or a greater class of rigid functions.

2.3 Comparison Of The Methods

All the algorithms that we proposed for solving the BP, with an horizontal partition of the matrix A, can solve it accurately, as long as the parameters of each of its "subalgorithms" are well adjusted. In this section, we compare these algorithms, taking into consideration not only theoretical aspects but also the computer simulations. The values of the parameters of the algorithms in the simulations were adjusted from experience with the hope that these simulations can replicate a realistic scenario.

We should also mention that, being all the algorithms based on several processors, it was only possible for the author just to simulate them in a single computer, being the simulations in a multi–processor environment beyond the scope of this thesis. The main focus here is the relative comparison of the algorithms in terms of their features, and not to present absolute standards for comparing distributed algorithms of this kind.

For a matter of simplicity, we will use the acronyms *SMBBP*, *M/DQA* and *M/GS* to designate respectively algorithms 2 (Subgradient method for bounded BP), 5 (Multipliers/DQA) and 7 (Multipliers/Gauss–Seidel).

The theoretical comparison of the algorithms is depicted in table 2.2, and the meaning of each row is explained next.

Degree of parallelism. We want to define a measure that translates somehow the gain in time that we would have in executing an algorithm if we increased the number of processors P. This measure must not depend on P and the higher it is, the higher the gain.

To define a possible such measure, let φ designate the number of distributed processors that execute the same task at the same time. Then, we can define the *degree of parallelism* of an algorithm as

$$\text{Degree of parallelism} := \lim_{P \to +\infty} \frac{\varphi}{P}.$$

In the definition, we assume that, for a given algorithm, all the distributed processors are equal and that no stopping criterion has been verified. As so, this definition must be applied only in a typical iteration. Further, it can only be applied to algorithms in which the distributed processors take no decisions. This

way, all the tasks have to be really performed at the same time (of course, only in theory) by all the distributed processors. The term "distributed" was used in the definition to exclude central processors, since we are not considering their replication. Note that all algorithms that we have seen, except the M/GS (algorithm 7), have equal distributed processors that take no decisions. Thus, this definition makes sense. It also makes sense for the M/GS, since in the typical iteration no stopping criterion is verified, and consequently processor 1 can be seen as equal to the other processors.

Then, the degree of parallelism of the SMBBP and the M/DQA is 1 (all the distributed processors work at the same time), whereas the degree of parallelism of the M/GS is 0 (only one processor can work at each iteration).

Complexity of the processors. We also want to define a measure that indicates the difficulty in implementing the tasks assigned to a processor. In the context of the algorithms presented in this thesis, we can distinguish between two kinds of processors: the ones that only perform simple operations, such as sums, multiplications and binary operations and have no interior loops; and the ones that, besides all these simple operations, also execute minimizations that imply interior loops. We represent the simple processors with symbol "$-$", and the complex processors with the symbol "$+$".

Communications requirements. When the algorithms were presented, we mentioned, for each one, the size of the vectors exchanged between the processors in each iteration. We will only consider here the communications involved in the iterative processes. For example, in the SMBBP only the communications performed in the subgradient method phase of this algorithm are considered.

Although the size of the vectors exchanged can give a good idea of the demanded speed for the links, this concept doesn't translate the total quantity of information exchanged during the whole algorithm, as it doesn't take into account the number of iterations needed for the algorithm to converge.

Table 2.2: Theoretical features of the algorithms presented in chapter 2.

	SMBBP	M/DQA	M/GS
Degree of parallelism	1	1	0
Complexity of distributed processors	$-$	$+$	$+$
Complexity of central processor	$+$	$-$	
Robustness to instantaneous link failures	No	No	Yes
Size of vectors exchanged by iteration	$2(m+1)$	$2(m+n)$	$m+1$

Results from simulations. Although the SMBBP, the M/DQA and the M/GS solve the BP problem or an heuristic to get to its solution, they use different architectures or very different approaches. While the first one uses the dual space to select some columns of matrix A and then solve a smaller BP, the M/DQA and M/GS algorithms solve directly the BP and both have a similar structure. So, there is no difficulty in comparing these two algorithms. The real problem is in comparing them with the SMBBP. As strong duality is verified in the bounded BP (2.3), we can use the dual cost function as a stopping criterion for the SMBBP, in order to compare it with the other algorithms: firstly, we run the M/DQA and the M/GS with exactly the same parameters and stopping criteria; then, we run the SMBBP and use as a stopping criterion, for the subgradient loop, the value of the cost function $H(\lambda)$ (2.7), that is, the subgradient method stops when $L(\lambda)(= -H(\lambda))$ is close to $p^\star$, where $p^\star$ is the optimal cost function for both the algorithms M/DQA and M/GS. In table 2.3, we considered that the subgradient method found an optimal dual solution when $L(\lambda) \geq 0.99p^\star$.

We must note that the results of the experiences depicted in table 2.3 didn't take into account any time spent in the communications.

Table 2.3: Experimental results from the simulation of the SMBBP, the M/DQA and M/GS. The numbers presented are the average of 10 random experiments in a simulation of a distributed environment with $P = 10$ processors, each with 25 columns of a matrix $A \in \mathbb{R}^{50\times 250}$. The errors were measured taking into account that the solution of the linear programming formulation of the BP, given by *Yalmip/Matlab*, is the correct one. The parameters for the M/DQA and M/GS were $\epsilon^T = 0.1$ and $\epsilon^K = 0.01$, while the stopping criterion for the SMBBP was $L(\lambda) \geq 0.99p^\star$, where $p^\star$ is the optimal cost given by the M/GS algorithm.

	SMBBP	M/DQA	M/GS
Error in x	3.11×10^{-2}	5.54×10^{-3}	8.58×10^{-3}
Error in $f(x)$	1.40×10^{-2}	1.72×10^{-3}	1.07×10^{-3}
Total Time [s]	2.28	1.03×10^{2}	6.58
Time in Parallel [s]	1.72	1.02×10^{2}	
Estimated Time [s]	7.32×10^{-1}	1.20×10^{1}	6.58
Inner Iterations		3.52×10^{2}	4.71×10^{2}
External Iterations	2.47×10^{3}	9.00	7.56
Percentage Of Used Columns	15.1		

The row "Error in x" represents the average of the absolute errors in the variable x, *i.e.*, $\|\hat{x} - x^*\|$, where $\hat{x}$ is the solution returned by any of the considered algorithms and x^* is the solution returned by the linear programming formulation (1.11) of the BP. Although the average of the error of the SMBBP was higher than the average error of the other methods, we must say that in 50% of the experiences the SMBBP returned the exact solution (errors of the order of 10^{-9}), that is, we had $\Omega^* \subset \Omega$ (recall the definitions of Ω^* and Ω in subsection 2.1.2). Neither the M/DQA nor the M/GS returned the exact solution once (with errors of magnitude 10^{-9}).

The row "Error in $f(x)$" represents the same as the first row, but applied to the cost function.

The "Total Time" and the "Time in Parallel" are the average of the total time taken by the whole algorithm and the respective estimated time spent in parallel tasks. This estimation, only for the SMBBP and M/DQA (since in the M/GS the processors work one at each time), is done by summing the time of all parallel tasks. With the total time and the parallel time, we can have an estimation of the time that the algorithms would take if they had really been executed in a multi–processor environment. Such estimation is given by

$$t_{\text{est}} = t_T + \frac{1-P}{P} t_p,$$

where t_{est} is the estimated time, t_T is the total time, t_P is the time spent in parallel and P is the number of processors. The row "Estimated Time" contains this estimation measured in seconds for all the algorithms.

The following rows contain the average of the internal iterations, if the algorithm has an inner loop, and the average of the external iterations.

The "Percentage Of Used Columns" row only applies to the SMBBP and is the average of the reduction of columns from the matrix A to the matrix M, according to the decision criterion (2.16).

We must say that, although the values in table 2.3 result from 10 random experiments, its average values, which are represented in the table, are representative of what happened in most of the experiments.

Overall comparison. In terms of overall comparisons of the methods, for an architecture that has a central processor (Figure 2.2), the SMBBP seems to be superior to the M/DQA in terms of time

consumed; concerning the "error in x", we can decrease the parameter ζ (increasing however the time estimated) and have errors much inferior to those of the M/DQA.

We can do exactly the same observations concerning the comparison of the SMBBP and the M/GS. However, the M/GS appears to be more competitive than the M/DQA, at least in the aspects featured in table 2.3. Also note that the M/GS has a completely different architecture for the links from the other algorithms (see Figure 2.11).

Estimated exchanges of information. Concerning the communications requirements, we can say that, in average, the processors in the SMBBP algorithm communicated 4.94×10^3 blocks of $m+1$ numbers, in the M/DQA communicated 6.97×10^4 blocks of m numbers (taking into account that $n = 10m$) and in the M/GS communicated 3.56×10^3 blocks of $m+1$ numbers. These numbers can be obtained from the information in tables 2.2 and 2.3.

Final conclusions. At this point, it is worth to say that the SMBBP is the more adequate to applications where accuracy is more important than the time spent in solving the BP. Though this is not evident from table 2.3, we know that if we increase the number of the iterations of the subgradient method in SMBBP or even just decrease the parameter ζ, we will have, with great probability, the condition $\Omega^* \subset \Omega$. This means that the SMBBP will return the exact solution of the BP (recall theorem 3).

On the other hand, the M/GS is the more adequate to applications where the time is more important than accuracy, being ϵ^T and ϵ^K, of the stopping criteria, the trade–off parameters.

Chapter 3

Distributed BP With Vertical Partition

In this chapter we present a distributed algorithm for solving the basis pursuit (BP) problem

$$\begin{array}{ll} \min & \|x\|_1, \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \qquad (3.1)$$

where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ are given and $x \in \mathbb{R}^n$ is the optimization variable.

However, opposed to what was assumed in chapter 2, here we will assume that the matrix A is partitioned vertically, where each of its blocks A_p is stored on a single processor. As before, no processor has full knowledge of this matrix, it just knows some (or even just one) rows of A instead.

There are situations for which this kind of partition of the matrix A makes all the sense. Imagine, for instance, a sensor network whose goal is to measure a phenomenon that can be represented in the vector form by $x \in \mathbb{R}^n$. Each sensor has its own sensing device, represented by a row $a_j \in \mathbb{R}^n$ of A, and can only measure a single projection $b_j \in \mathbb{R}$ of the vector x onto a_j, *i.e.*, $b_j = a_j^T x$ (here we have $P = m$ processors). If we know *a priori* that x is s-sparse and the matrix A has a restricted isometry constant $\delta_{2s}(A) < \sqrt{2} - 1$, then we can reconstruct x by solving (3.1), according to theorem 1 on page 6. Note that, as we are assuming that $m < n$, we are actually reconstructing an n-dimensional vector from m projections. (see Figure 3.1). Of course, the processors must be able to exchange information in order to get to a consensus about the variable x.

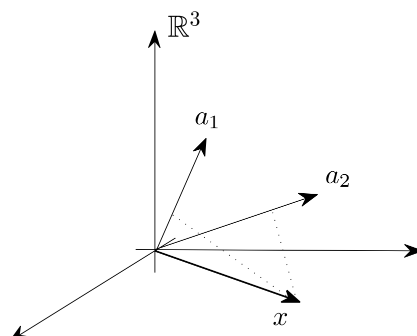


Figure 3.1: Graphical interpretation of the basis pursuit problem (3.1), where the matrix A is in $\mathbb{R}^{2\times 3}$. Knowing that x is a sparse vector, *i.e.* it lies on some coordinate plane, it is possible to reconstruct it only from the knowledge of the inner products $a_1^T x$ and $a_2^T x$, provided that the matrix $A = [a_1\, a_2]^T$, which has also to be known, satisfies some conditions.

The algorithm we present in this chapter, in spite of using many results of chapter 2, requires a

different communication link architecture from those found in chapter 2. We will also see how to easily adapt this algorithm to solve the BPDN, also in a distributed way.

3.1 Proposed Approach

To formalize, we assume that the matrix A has the following structure

$$A = \underbrace{\begin{bmatrix} \boxed{A_1} \\ \vdots \\ \boxed{A_p} \\ \vdots \\ \boxed{A_P} \end{bmatrix}}_{n} \Bigg\updownarrow m \quad ,$$

where each submatrix $A_p \in \mathbb{R}^{m_p \times n}$ is stored on processor p, for $p = 1, 2, \ldots, P$ and with $m = m_1 + m_2 + \ldots + m_P$. Also, we use the same partition for the vector $b \in \mathbb{R}^m$:

$$b = \begin{bmatrix} b_1 \\ \vdots \\ b_p \\ \vdots \\ b_P \end{bmatrix},$$

where $b_p \in \mathbb{R}^{m_p}$ for $p = 1, 2, \ldots, P$. Without loss of generality, we can assume that each submatrix A_p is full–rank, *i.e.*, its rows are linearly independent. We also assume, for convenience of notation, that each block A_p contains adjacent rows of A. Otherwise, a re–ordering of the rows of A and b can be carried out in order to verify this.

This way, we can write problem (3.1) as

$$\begin{array}{cl} \min & \frac{1}{P}\|x\|_1 + \frac{1}{P}\|x\|_1 + \ldots + \frac{1}{P}\|x\|_1, \\ A_1 x = b_1 & \\ A_2 x = b_2 & \\ \vdots & \\ A_P x = b_P & \\ \text{var} : x \in \mathbb{R}^n & \end{array}$$

and cloning the variable x into $x_1, x_2, \ldots, x_P$, this problem is equivalent to

$$\begin{array}{cc} \begin{array}{c} \min \\ A_1 x_1 = b_1 \\ A_2 x_2 = b_2 \\ \vdots \\ A_P x_P = b_P \\ x_1 = x_2 \\ x_2 = x_3 \\ \vdots \\ x_{P-1} = x_P \\ \text{var} : (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n \end{array} & \frac{1}{P}\|x_1\|_1 + \frac{1}{P}\|x_2\|_1 + \ldots + \frac{1}{P}\|x_P\|_1. \end{array} \tag{3.2}$$

We will solve problem (3.2) through its dual, by dualizing only the last $P-1$ equalities of its constraints.

Naive approach. If we used the ordinary Lagrangian, we would need additional constraints to guarantee that the minimizer of each of its subproblems doesn't lie at the infinity. To see this, note that the dual function $L : \underbrace{\mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n}_{P-1}$ of (3.2) (by dualizing only the last $P-1$ equalities) can be written as

$$L(\lambda_1, \lambda_2, \ldots, \lambda_{P-1}) = \sum_{p=1}^{P} \left[\begin{array}{cc} \inf & \left(\frac{1}{P}\|x_p\|_1 + (\lambda_p - \lambda_{p-1})^T x_p \right) \\ A_p x_p = b_p & \\ \text{var} : x_p \in \mathbb{R}^n & \end{array} \right], \tag{3.3}$$

where, by definition, we make $\lambda_0 := \lambda_P := 0_n$ (0_n is the zero vector in $\mathbb{R}^n$). It is clear that each subproblem in (3.3) can have an optimal value of $-\infty$. Imagine, for instance, that the vector $(\lambda_p - \lambda_{p-1})$ has an entry with a number that has an absolute value greater than $1/P$. Therefore, some kind of restrictions, for example polyhedral, must be added to the constraints of each subproblem of (3.3).

Augmented Lagrangian. As previously discussed, this kind of constraints in the dual problem can be avoided if we transform the dual Lagrangian function into a coercive function. We will do so by using the augmented Lagrangian and, subsequently, we will apply the method of multipliers (algorithm 3, on page 30) in order to minimize it.

The augmented Lagrangian $L_a : \underbrace{\mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n}_{P} \times \underbrace{\mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n}_{P-1} \to \mathbb{R}$ of (3.2) associated with the parameter ρ is

$$\begin{aligned} L_a(x_1, \ldots, x_p, \ldots, x_P, \lambda_1, \ldots, \lambda_{P-1}) = & \left[\frac{1}{P}\|x_1\|_1 + \lambda_1^T x_1 \right] + \ldots + \left[\frac{1}{P}\|x_p\|_1 + (\lambda_p - \lambda_{p-1})^T x_p \right] \\ & + \ldots + \left[\frac{1}{P}\|x_P\|_1 - \lambda_{P-1}^T x_P \right] + \rho\|x_1 - x_2\|^2 + \ldots + \rho\|x_{P-1} - x_P\|^2. \end{aligned} \tag{3.4}$$

From the coercivity and convexity of the ℓ_1-norm, it follows that the method of multipliers is well–defined for problem (3.2), and by theorem 4 its convergence is established.

The next step is then to see how we can solve the problem in the step 1 of the method of multipliers:

$$(x_1^k, \ldots, x_p^k, \ldots, x_P^k) \in \arg \min_{\substack{A_1 x_1 = b_1 \\ \vdots \\ A_p x_p = b_p \\ \vdots \\ A_P x_P = b_P}} L_a(x_1, \ldots, x_p, \ldots, x_P, \lambda_1^k, \ldots, \lambda_{P-1}^k), \tag{3.5}$$

where this problem is solved for fixed dual variables $\lambda_1^k, \ldots, \lambda_{P-1}^k$. Note that the updating of these variables is done by

$$\lambda_p^{k+1} = \lambda_p^k + \rho^k (x_p^k - x_{p+1}^k), \tag{3.6}$$

for $p = 1, 2, \ldots, P-1$.

Problem (3.5) is not separable into P independent subproblems in the variables $x_1, x_2, \ldots, x_P$, owing to the quadratical terms $\|x_{p-1} - x_p\|^2$, for $p = 1, 2, \ldots, P-1$. Recall that we had exactly the same problem in chapter 2 and we overcame it by using the DQA and the Nonlinear Gauss–Seidel algorithms. Fortunately, both algorithms can be applied to (3.5) since their convergence is ensured for this problem. This happens because the augmented Lagrangian (3.4) is convex, coercive and rigid on the primal variables $x_1, x_2, \ldots, x_P$, when the dual variables are fixed. Furthermore, (3.4) has unique block coordinate minimizers as it strictly convex on each block variable x_p, for $p = 1, 2, \ldots, P$.

To see that, note that the rigidity of (3.4) follows directly from lemma 3 on page 33. The strict convexity of each block coordinate x_p in (3.4) can be proved if we write (3.4) as

$$\sum_{p=1}^{P} \frac{1}{P} \|x_p\|_1 + \phi^{\lambda_1}(x_1 - x_2) + \phi^{\lambda_2}(x_2 - x_3) + \ldots + \phi^{\lambda_{P-1}}(x_{P-1} - x_P),$$

where $\phi^{\lambda_p}(z) = \lambda_p^T z + \rho \|z\|^2$, for $p = 1, 2, \ldots, P-1$. As each function ϕ^{λ_p} is strictly convex on the corresponding block variables, x_p and x_{p+1}, it results that (3.4) is also strictly convex on each block variable. This justifies the uniqueness of the block minimizers of the augmented Lagrangian (3.4). Concerning the coercivity of this function, we note that there exists an m such that

$$\phi^{\lambda_1}(x_1 - x_2) + \phi^{\lambda_2}(x_2 - x_3) + \ldots + \phi^{\lambda_{P-1}}(x_{P-1} - x_P) > m,$$

and, as a consequence,

$$L_a(x_1, \ldots, x_p, \ldots, x_P, \lambda_1, \ldots, \lambda_{P-1}) > \frac{1}{P} \sum_{i=1}^{P} (\|x_p\|_1 + m).$$

If we consider the variable $x := (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n$ and fix the dual variables $\lambda_1, \ldots, \lambda_{P-1}$, we have

$$\begin{aligned} L_a(x, \lambda_1, \ldots, \lambda_{P-1}) &> \frac{1}{P}(\|x\|_1 + m) \\ &\geq \frac{1}{P}(\|x\| + m), \end{aligned}$$

which tends to infinity as $\|x\| \to +\infty$. Thus, the augmented Lagrangian (3.4) is also coercive.

Consequently, theorems 5 and 6, that guarantee the convergence of the DQA and the Nonlinear

Gauss–Seidel algorithms, respectively, remain valid for the optimization problem (3.5).[1] However, we will just explore the Nonlinear Gauss–Seidel algorithm for two reasons:

1. it has better convergence properties, namely the speed of convergence is known to be higher than the speed of convergence of the DQA;
2. the minimum links between the processors that both algorithms require, unlike what happened in chapter 2, are too alike.

Nonlinear Gauss–Seidel Approach

The Nonlinear Gauss–Seidel algorithm (algorithm 6 on page 39), applied to the augmented Lagrangian (3.4), at the iteration k of the method of multipliers, yields

$$\min_{A_p y_p = b_p} L_a^p(y_p),$$

where

$$L_a^p(y_p) := \frac{1}{P}\|y_p\|_1 + (\lambda_p^k - \lambda_{p-1}^k)^T y_p + \rho^k \|x_{p-1}^{t+1} - y_p\|^2 + \rho^k \|y_p - x_{p+1}^t\|^2, \tag{3.7}$$

for $p = 1, 2, \ldots, P$. By definition, we make $\lambda_0 := \lambda_P := x_{P+1} := x_0 := 0_n$. It is implicit in (3.7) that we index each iteration of the Nonlinear Gauss–Seidel algorithm by the letter t.

Developing the quadratic terms, we can see that (3.7) is equivalent to

$$\min_{A_p y_p = b_p} \frac{1}{P}\|y_p\|_1 + \left[\lambda_p^k - \lambda_{p-1}^k - 2\rho^k (x_{p-1}^{t+1} + x_{p+1}^t)\right]^T y_p + 2\rho^k \|y_p\|^2. \tag{3.8}$$

In conclusion, the main task of each processor p is to solve problem (3.8). By using the epigraph technique, (3.8) can be recast as a quadratic program. Thus, there exist popular software packages that can solve this problem efficiently. In appendix B, an algorithm that solves directly problem (3.8) (in its non–differentiable form) is provided.

Inefficiency problem. We are expecting that the algorithm that we are developing will be too inefficient when compared to the algorithms that solve the original BP problem (3.1) in a centralized version, and even to the algorithms from chapter 2. Indeed, note that (3.8) is equivalent to a quadratic program with a variable of size $2n$ and $2n + m_p$ constraints, whereas the BP (3.1) belongs to a simpler class, it is equivalent to a linear program (see subsection 1.3.2), and has a $2n$-dimensional vector as a variable and $2n + m$ constraints. So, what we do here is to solve several times a harder problem, for each processor. However, the merit of this algorithm is that it allows to solve the BP with a vertical partition of the matrix A, and without the need of knowing the entire matrix.

A new architecture for the links. Note that, for a fixed k, from (3.8) we can see that the processor p at the calculation of x_p^{t+1} (*i.e.* the solution of (3.8)) only needs the vectors x_{p-1}^{t+1} and x_{p+1}^t from its neighbors. Because the extremal processors, 1 and P, only require the vectors x_2^t and x_{P-1}^{t+1}, respectively, the minimal architecture for the links is only a single line, as it is depicted on Figure 3.2.

Also note that, for solving (3.8), the processor p only needs to know the dual variables λ_{p-1}^k and λ_p^k. As it accesses the variables x_{p-1}^{t+1} and x_{p+1}^t (see Figure 3.2), the processor p can update the dual variables λ_{p-1}^k and λ_p^k by itself, that is, without exchanging any information with the neighbor processors (see (3.6)).

[1] Although both theorems are proved for unconstrained minimization, its generalization to the constrained case can be easily carried out, as long as we guarantee that the sequence of problems that we get has always a solution. This applies to our case as, by assumption, each submatrix A_p is full–rank.

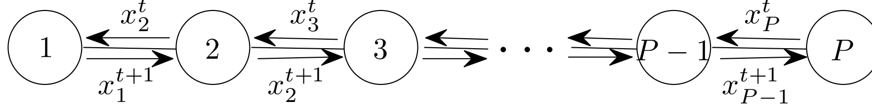


Figure 3.2: Architecture needed for the implementation of algorithm 8; and illustration of the flow of information in its inner cycle (Nonlinear Gauss–Seidel step).

Stopping criterion. As usual, the stopping criteria of either the method of multipliers or the Nonlinear Gauss–Seidel algorithm can be based on a maximum number of iterations, or on the fact that the variable calculated in the actual iteration doesn't differ too much from the one calculated at the previous iteration — for the Nonlinear Gauss–Seidel algorithm, the stopping criterion for the processor p can be $\|x_p^{t+1} - x_p^t\| < \epsilon^T$; and for the method of multipliers, it can be $\|x_p^k - x_{p+1}^k\| < \epsilon^K$, for the dual[2] variable λ_p (see (3.6)).

However, how can a processor know that the other processors have converged (for any of the algorithms)? The link architecture in Figure 3.2 doesn't allow any circular flow of information, so we can't assign the task of verifying the convergence of the algorithms to a particular processor, as we did in subsection 2.2.3. What we propose is a method that propagates the local information only when needed. That is, when a processor finds that a convergence criterion is satisfied, it transmits this information to its neighbors. However, the neighbors only transmit this new information to their other neighbors if they have also satisfied the same stopping criterion themselves. This way, we guarantee that when all the processors have converged in some algorithm, they all know that the others have also converged. All this information can be carried in a *convergence vector* $v^p \in \mathbb{R}^P$.[3] For instance, if each processor p has its own convergence vector v^p, then it can store all the information about the convergence of the other processors in the following way: if $v_l^p \neq 0$, this means that the processor p knows that processor l has already converged. Consequently, the processor p can update its convergence vector with the information from the convergence vectors of its neighbors by summing all them (whether it is a logical sum or not), *i.e.*

$$v^p \leftarrow v^p + v^{p-1} + v^{p+1}.$$

Concatenating the method of multipliers, the Nonlinear Gauss–Seidel algorithm and the details we have seen above we get the following algorithm:

Algorithm 8 (Multipliers/Gauss–Seidel for Vertical Partition)**.**

- *Predefined Parameters/Initialization:*
  - A_p *and* b_p *for each processor,* $p = 1, \ldots, P$*;*
  - *Choose* $c \geq 1$ *(for actualizing* ρ^k*);*
  - *Choose* K *and* T *as the maximum number of iterations; and* ϵ^K, ϵ^T *for the stopping criteria;*
  - *Choose* $x_p^0 \in \mathbb{R}^n$ *for* $p = 1, \ldots, P$*;* $\lambda_p^0 \in \mathbb{R}^n$*, for* $p = 1, \ldots, P-1$*;* $\rho^0 \in \mathbb{R}_+$*; and* $Q \in \mathbb{R}\backslash\{0\}$*;*
  - *Set the inner convergence vectors* $v^{p,T} \in \mathbb{R}^P$ *to zero, for* $p = 1, \ldots, P$*;*
  - *Set the external convergence vectors* $v^{p,K} \in \mathbb{R}^P$ *to zero, for* $p = 1, \ldots, P$*;*
  - $t_p = 0$*, for all* $p = 1, \ldots, P$ *(we refer to* t_p *instead of just* t *because these numbers are not synchronous, hence each processor has its own internal counter* t_p*).*

[2]Note that this stopping criterion implies that a consensus about the solution of (3.1) has been reached among all the processors. In fact, we can guarantee with this stopping criterion that the distance between any pair of x_p's is majored by $(P-1)\epsilon^T$.

[3]According to this notation, a superscript letter indicates the "owner" or the iteration, while a subscript letter indicates the number of the entry of a vector.

- *for* $k = 0$ *until* K [Method of Multipliers]*:*

  Procedure (for the processor $p = 1, \dots, P$**):**

  - *If it receives* $x_{p-1}^{t_p+1}$ *and* $x_{p+1}^{t_p}$, *and* $v_p^{p,T}$ *and* $v_p^{p,K}$ *are both different from zero,*
    1. *Form the vector* $\tau_p = \lambda_p^k - \lambda_{p-1}^k - 2\rho^k(x_{p-1}^{t_p+1} + x_{p+1}^{t_p})$*;*
    2. *Solve* $x_p^{t_p+1} = \arg\min_{A_p x_p = b_p} (1/P)\|x_p\|_1 + \tau_p^T x_p + 2\rho^k \|x_p\|^2$*;*
    3. *Send* $x_1^{t_p+1}$ *to processors* $p-1$ *(if* $v_{p-1}^{p,T} \neq 0$ *and* $v_{p-1}^{p,K} \neq 0$*) and* $p+1$ *(if* $v_{p+1}^{p,T} \neq 0$ *and* $v_{p+1}^{p,K} \neq 0$*);*
    4. *Check inner stopping criterion:*
       * *If* $t_p > T$ *or* $\|x_p^{t_p} - x_p^{t_p+1}\| < \epsilon^T$, *check external stopping criterion:*
         · *If* $k > K$ *or (*$\|x_p^{t_p+1} - x_{p-1}^{t_p+1}\| < \epsilon^K$ *and* $\|x_p^{t_p+1} - x_{p+1}^{t_p}\| < \epsilon^K$*), make* $v_p^{p,K} \leftarrow Q$, *and try to transmit the vector* $v^{p,K}$ *to the neighbor processors in the next idle times.*
         · *else, make* $v_p^{p,T} \leftarrow Q$, *and try to transmit the vector* $v^{p,T}$ *to the neighbor processors in the next idle times..*
    5. *Make* $t_p \leftarrow t_p + 1$.
  - *If the links to the processors* $p-1$ *and* $p+1$ *are idle,*
    1. *Transmit (if needed) or receive the external and the inner convergence vectors,* $v^{l,T}$ *and* $v^{l,K}$, *where* $l = p-1$ *or* $l = p+1$.
    2. *Update the convergence vectors:* $v^{p,T} \leftarrow v^{p,T} + v^{p-1,T} + v^{p+1,T}$*;* $v^{p,K} \leftarrow v^{p,K} + v^{p-1,K} + v^{p+1,K}$*;.*
    3. *If all entries of* $v^{p,K}$ *differ from zero,* stop the algorithm*.*
    4. *else,*
       * *If all entries of* $v^{p,T}$ *differ from zero,*
         · *Make* $x_p^k = x_p^{t_p}$ *(the last calculated value);*
         · *Make* $x_{p-1}^k = x_{p-1}^{t_p}$ *and* $x_{p+1}^k = x_{p+1}^{t_p-1}$ *(the last received values);*
         · *Update* $\lambda_p^{k+1} = \lambda_p^k + \rho^k(x_p^k - x_{p+1}^k)$ *and* $\lambda_{p-1}^{k+1} = \lambda_{p-1}^k + \rho^k(x_{p-1}^k - x_p^k)$*; and also update* $\rho^{k+1} = c\rho^k$.
         · *Make* $v^{p,T} = 0_P$.

Although algorithm 8 is written for a generic processor p, we must take into account that for the extremal processors, 1 and P, all kind of interaction of these processors with the processors $p-1$ and $p+1$, respectively, should be ignored.

Degree of parallelism. One might wonder how this algorithm behaves globally, that is, how does the fact that a processor is only capable of communicating with its neighbors affect the calculation of the new variables x_p^{t+1}? As any kind of information takes a few hops from the processor 1 to the processor P, we might think that the number of iterations between the calculation of x_p^t and x_p^{t+1} is about P iterations, for any $p = 1, \dots, P$. Fortunately, this is not true during the steady phase of the algorithm. In this case, only two iterations are needed. Table 3.1 shows the pattern of calculation of the new variables in the inner cycle of algorithm 8 during six iterations, *i.e.* the variables which result from solving problem (3.8) are shown with a circle around. From this table, we can observe not only that there exists a transient initial phase, where the calculation of the new variables is done in a sequential order, but also a steady phase where any variable is updated in two iterations (that is, two information exchange phases). See, for example, the processor 3 at the end of the third iteration of the algorithm, after the reception of the variable x_2^1 from processor 2 (row 6 and column 3). In this iteration, processor 3 has access to the

variables x_2^1 from processor 1, x_4^0 from processor 4 and its own x_3^0. Thus, it can now solve problem (3.8) and calculate the new variable x_3^1. Afterwards, it sends this new variable to the neighbors, processors 2 and 4. Then, after receiving their new variables, it can solve again (3.8). This means that processor 3 can calculate new variables in every two iterations. This same pattern is verified for all the other processors, during the steady phase of the algorithm.

Consequently, using the definition given in page 45 the degree of parallelism of the algorithm 8 is $1/2$. This is somewhat surprising because in algorithm 7, which is based on the same subalgorithms of algorithm 8, the degree of parallelism is zero (the processors only work one at each time).

The fact of the degree of parallelism of this algorithm being $1/2$, means that there are enough idle times (50%) to transmit the convergence vectors between the processors, whenever it is needed.

Also note that the size of the exchanged vectors is n in most of the times, and P (convergence vectors) in the last iterations of each cycle.

Table 3.1: Illustration of the execution of 6 iterations of an inner cycle (Nonlinear Gauss–Seidel) of the algorithm 8, for $P = 4$. Each column refers to a variable that a processor p has access (its own x_p^t, and the neighbor's variables, x_{p-1}^{t+1} and x_{p+1}^t). Note that it is only possible to calculate x_p^{t+1} when the processor p has access to x_{p-1}^{t+1} and x_{p+1}^t. When this happens, the new variable, x_p^{t+1}, appears circled after the internal calculations, and the old variables x_{p-1}^{t+1} and x_{p+1}^t are discarded. Note that after $P-1$ iterations the algorithm enters a steady state.

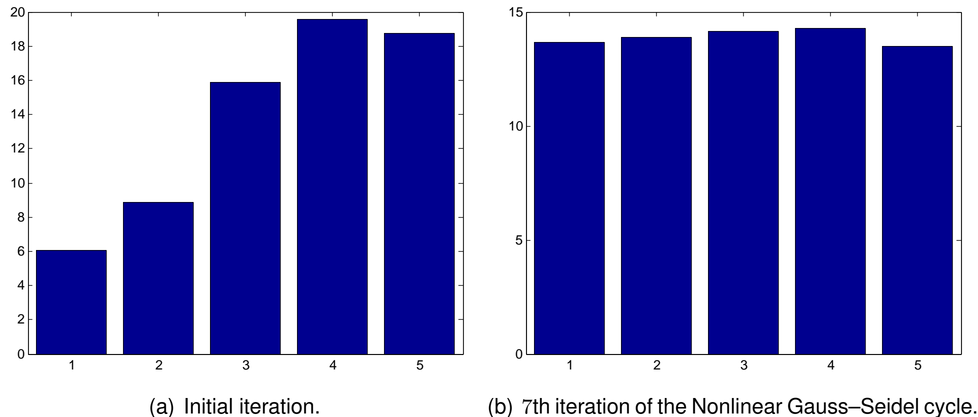
Processors									
①			②			③		④	
x_1^0			x_2^0			x_3^0			x_4^0
transmission/reception									
x_1^0	x_2^0	x_1^0	x_2^0	x_3^0	x_2^0	x_3^0	x_4^0	x_3^0	x_4^0
internal calculations									
$\textcircled{x_1^1}$		x_1^0	x_2^0	x_3^0	x_2^0	x_3^0	x_4^0	x_3^0	x_4^0
transmission/reception									
x_1^1		x_1^1	x_2^0	x_3^0	x_2^0	x_3^0	x_4^0	x_3^0	x_4^0
internal calculations									
x_1^1			$\textcircled{x_2^1}$		x_2^0	x_3^0	x_4^0	x_3^0	x_4^0
transmission/reception									
x_1^1	x_2^1		x_2^1		x_2^1	x_3^0	x_4^0	x_3^0	x_4^0
internal calculations									
$\textcircled{x_1^2}$			x_2^1			$\textcircled{x_3^1}$		x_3^0	x_4^0
transmission/reception									
x_1^2		x_1^2	x_2^1	x_3^1		x_3^1		x_3^1	x_4^0
internal calculations									
x_1^2			$\textcircled{x_2^2}$			x_3^1			$\textcircled{x_4^1}$
transmission/reception									
x_1^2	x_2^2		x_2^2		x_2^2	x_3^1	x_4^1		x_4^1
internal calculations									
$\textcircled{x_1^3}$			x_2^2			$\textcircled{x_3^2}$			x_4^1
transmission/reception									
x_1^3		x_1^3	x_2^2	x_3^2		x_3^2		x_3^2	x_4^1
internal calculations									
x_1^3			$\textcircled{x_2^3}$			x_3^2			$\textcircled{x_4^2}$

Characteristics of the algorithm. Using the concepts defined in section 2.3, we present the theoretical aspects of algorithm 8 in table 3.2. Note that, since the algorithms of chapter 2 are based on a different partition of the matrix A, we shouldn't compare the performance of those algorithms with the performance of algorithm 8. In fact, we have already seen that this algorithm is very inefficient, but, until so far, it was the only one that we could adapt to a vertical partition of A.

Table 3.2: Theoretical features of algorithm 8. All the used concepts are defined in section 2.3.

Degree of parallelism	$1/2$
Complexity of distributed processors	$+$
Robustness to instantaneous link failures	Yes
Size of vectors exchanged by iteration	$2n$

A curious aspect of this algorithm, though, is that before the processors update their dual variables λ_p^k, they need to get to a "small consensus" first, independently of the value of the variable ρ^k (for reasonable values of this variable, of course). This is evident from Figure 3.3, where the variables x_p^0, for five processors, were initialized with very different vectors (Figure 3.3(a)) and only after seven iterations of the inner cycle of algorithm 8 they were almost equal, although very different from the optimal solution (Figure 3.3(b)). We can see then, that the inner cycle is responsible for not keeping the variables very different from each other, and the external cycle is responsible for "pushing" the variables of each processor to near the optimal solution. In fact, when the variables are initialized with very different vectors, the algorithm firstly tries to bring them close to each other, spending some iterations of the inner cycle. And, after the variables being almost equal, the external loop brings them all in block close to the optimal solution. In this last phase of the algorithm, few inner loops are spent.



(a) Initial iteration.

(b) 7th iteration of the Nonlinear Gauss–Seidel cycle.

Figure 3.3: Histograms of the errors of each variable x_p, for $p = 1, \ldots, 5$, during the execution of algorithm 8. The variables were initialized in the following way: to x_1^0 was assigned a random vector of norm equal to 10; then, we made $x_5^0 = 2x_4^0 = 2x_3^0 = 2x_2^0 = 2x_1^0$. The histograms show the distance of each variable to the optimal one, x^*, at the initial iteration and at the 7th iteration of the Nonlinear Gauss–Seidel cycle, both at the first iteration of the method of multipliers (no updating of the dual variables was done).

3.1.1 Application to the BPDN

If, instead of the basis pursuit problem (3.1), we had the basis pursuit denoising problem

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|b - Ax\|^2 + \beta\|x\|_1, \tag{3.9}$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are given and $x \in \mathbb{R}^n$ is the optimization variable, we could also apply algorithm 8 with minor modifications to solve it. Using the same block partition of the matrix A and the vector b on page 50, we can write (3.9) as

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|b_1 - A_1 x\|^2 + \frac{1}{2}\|b_2 - A_2 x\|^2 + \ldots + \frac{1}{2}\|b_P - A_P x\|^2 + \frac{\beta}{P}\|x\|_1 + \frac{\beta}{P}\|x\|_1 + \ldots + \frac{\beta}{P}\|x\|_1.$$

Cloning the variable x into $x_1, x_2, \ldots, x_P$, we get problem (3.2) but for the BPDN:

$$\begin{array}{cc} \min & \displaystyle\sum_{p=1}^{P} \left(\frac{1}{2}\|b_p - A_p x_p\|^2 + \frac{\beta}{P}\|x_p\|_1 \right). \\ x_1 = x_2 & \\ x_2 = x_3 & \\ \vdots & \\ x_{P-1} = x_P & \\ \text{var} : (x_1, x_2, \ldots, x_P) \in \mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n & \end{array} \tag{3.10}$$

Since the augmented Lagrangian of (3.10) is convex, coercive and rigid, if we take the same steps used in pages 51-53 to get to (3.8), the result is the following optimization problem

$$\min_{y_p \in \mathbb{R}^n} \frac{1}{2}\|b_p - A_p y_p\|^2 + \frac{\beta}{P}\|y_p\|_1 + \left[\lambda_p^k - \lambda_{p-1}^k - 2\rho^k (x_{p-1}^{t+1} + x_{p+1}^t)\right]^T y_p + 2\rho^k \|y_p\|^2, \tag{3.11}$$

which is equivalent to a quadratic program. This way, algorithm 8 solves the BPDN, (3.9), if we replace problem (3.8) by problem (3.11) in this algorithm.

Chapter 4

Fast Methods For BP and BPDN: Future Research Topics

In this chapter we present possible new directions to solve the BP

$$\begin{array}{cc} \min & \|x\|_1 \\ Ax = b & \\ \text{var} : x \in \mathbb{R}^n & \end{array} \tag{4.1}$$

and the BPDN

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2 + \beta\|x\|_1 \tag{4.2}$$

problems in a fast way. The subjects approached in this chapter must not be seen as an extensive study of some proposed algorithms, as it was done in the previous chapters, but instead they must be seen as pointing to evidences that can lead to new efficient algorithms. To the best of our knowledge, such evidences were never emphasized in the literature.

Throughout this chapter we present possible techniques for solving both the BP and the BPDN with the underlying assumption that the matrix $A \in \mathbb{R}^{m \times n}$ is available beforehand with respect to the vector $b \in \mathbb{R}^m$. This assumption allows us to make some precalculations before the "arrival" of the vector b. Many applications, for example, where A is a known random matrix or some dictionary, give practical validity to this assumption.

4.1 Ellipsoidal Approximation

The approach of approximating a complicated set by a simpler set simplifies many problems, and is used in many contexts, such as optimization, system identification and control [14, 6]. In this chapter, we will do this by approximating a polyhedron by an ellipsoid in the most general case, or by a ball in a particular case.

This section, however, only concerns the more general case, in which the approximation is done by an ellipsoid.

Motivation of the approach. Recall that in subsection 1.3.3 we derived the dual program of the BP (4.1), which we replicate here:

$$\begin{array}{ll} \max & \lambda^T b. \\ \|A^T\lambda\|_\infty \leq 1 & \\ \text{var} : \lambda \in \mathbb{R}^m & \end{array} \tag{4.3}$$

We have also seen in subsection 2.1.2 that the knowledge of the optimal dual variable λ^*, that solves (4.3), can be used to discard the columns of the matrix A that aren't activated by the optimal primal solution x^*. As x^* is expected to be sparse, it activates few columns, thus this reduction of columns can be significant.

Being (4.3) a linear program, it would be interesting to see how we can obtain an approximate solution or a good warm–start (for an interior–point method) with practically no cost, provided that the matrix A is previously known, *i.e.*, much sooner (offline) than b (online).

4.1.1 Ellipsoidal Approximation For The BP

Recall that the shape of the set $\mathcal{P} = \left\{\lambda : \|A^T\lambda\|_\infty \leq 1\right\}$ (see Figure 1.5) resembles an ellipsoid, specially if there are many constraints, *i.e.*, if the matrix A has many columns. Besides that, $\mathcal{P}$ is always a convex set. So, this observation allows us to replace problem (4.3) by an approximated version

$$\max_{\lambda \in \mathcal{E}(B,d)} \lambda^T b, \tag{4.4}$$

where $\mathcal{E}(B, d) = \{Bx + d : \|x\| \leq 1\}$ represents the ellipsoid with the shape matrix B (square and non-singular) and center at d. The norm is the ℓ_2-norm. Another common representation of an ellipsoid is $\mathcal{E}_{\text{alt}}(B_{\text{alt}}, d) = \left\{x : (x - d)^T B_{\text{alt}}^{-1}(x - d) \leq 1\right\}$, where B_{alt} is symmetric and positive definite. We distinguish both representations by using the subscript "alt" on the latter, meaning alternative. One can easily check that both sets $\mathcal{E}(B, d)$ and $\mathcal{E}_{\text{alt}}(B_{\text{alt}}, d)$ are equal when $B = B_{\text{alt}}^{1/2}$.

However, the ellipsoid in (4.4) is not any ellipsoid, but the one that best approximates the set $\mathcal{P}$. We can use many criteria to define what "approximating" means in this case. Löwner-John ellipsoids, maximum volume inscribed ellipsoids [6, section 8.4], or even the linear combination of both are good choices, but here we choose the maximum volume inscribed ellipsoid, *i.e.*, the inner ellipsoid that best approximates the set $\mathcal{P}$. This choice has the advantage of every point of the ellipsoid is in the interior of the polyhedron $\mathcal{P}$. This is essential if we want to use an interior–point method to solve problem (4.4) exactly, using the solution of (4.4) as a starting point.

Note that the set $\mathcal{P}$ is symmetric about the origin, therefore also the maximum volume ellipsoid that best approximates it, meaning that $d = 0$. For this case, there is a bound for the error of approximation of a polyhedron by the maximum volume inscribed ellipsoid: if we expand the ellipsoid by $\sqrt{m}$, where m is the dimension of the ambient space, the expanded ellipsoid covers the whole polyhedron [6].

The problem of finding the maximum volume inscribed ellipsoid can be cast as a *second–order cone programming*, nonetheless this approach, for a polyhedron characterized by many constraints and in a moderate dimension ambient space, might not be useful even for an offline calculation. An iterative algorithm that works well for this situation is presented in [14].

We still need to see how to solve (4.4). In order for this approach to be advantageous, the calculation of the solution of (4.4) must have a low computational cost. But, as we will see, there is even a closed–

form expression for it:

$$\max_{\lambda \in \mathcal{E}(B,d)} \lambda^T b \iff \max_{\|x\| \le 1} b^T (Bx + d)$$
$$\iff \left[\max_{\|x\| \le 1} \left(B^T b\right)^T x \right] + b^T d$$

As we are maximizing an inner product in the unit–radius sphere, the optimal solution has the direction of vector $B^T b$ and norm equal to one: $x_a = \frac{B^T b}{\|B^T b\|}$ (see Figure 4.1). Therefore, the solution of (4.4) is:

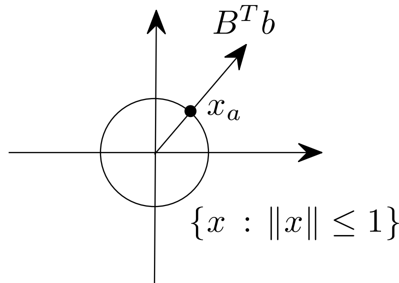


Figure 4.1: Maximization of an inner product $(B^T b)^T x$ over the unit–radius sphere $\{x \,:\, \|x\| \le 1\}$. The solution is $x_a = B^T b / \|B^T b\|$.

$$\begin{aligned} \lambda_a &= B x_a + d \\ &= B \frac{B^T b}{\|B^T b\|} + d. \end{aligned} \tag{4.5}$$

Figure 4.2 shows an example in two dimensions of the difference between the solutions of (4.3) and (4.4). Note that λ_a, given by (4.5), seems to be a good warm–start for an interior–point algorithm that solves the original linear program (4.3).

4.1.2 Ellipsoidal Approximation For The BPDN

Following the line of thought of subsection 4.1.1, we might wonder if a dual problem of the BPDN

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \|Ax - b\|^2 + \beta \|x\|_1 \tag{4.6}$$

can lead to an optimization problem over the set $\mathcal{P} = \left\{ \lambda \,:\, \|A^T \lambda\|_\infty \le 1 \right\}$. If so, we might benefit from a warm–start (due to an ellipsoidal approximation of this set) to solve exactly that dual of (4.6), whenever the matrix A is previously available. In fact, we will see that a dual program of (4.6) is equivalent to the projection of a point on the set $\mathcal{P}$. We will also see that the knowledge of the optimal dual variable λ^* allows us to discard the columns of the matrix A that aren't activated by the optimal solution x^*, just like what happens in the BP problem.

Point projection on $\mathcal{P}$. First of all, if we introduce a new variable $z \in \mathbb{R}^m$ (recall that $A \in \mathbb{R}^{m \times n}$), problem (4.6) is equivalent to

$$p^\star = \min_{\substack{Ax = z \\ \text{var} : (x,z) \in \mathbb{R}^n \times \mathbb{R}^m}} \frac{1}{2} \|z - b\|^2 + \beta \|x\|_1. \tag{4.7}$$

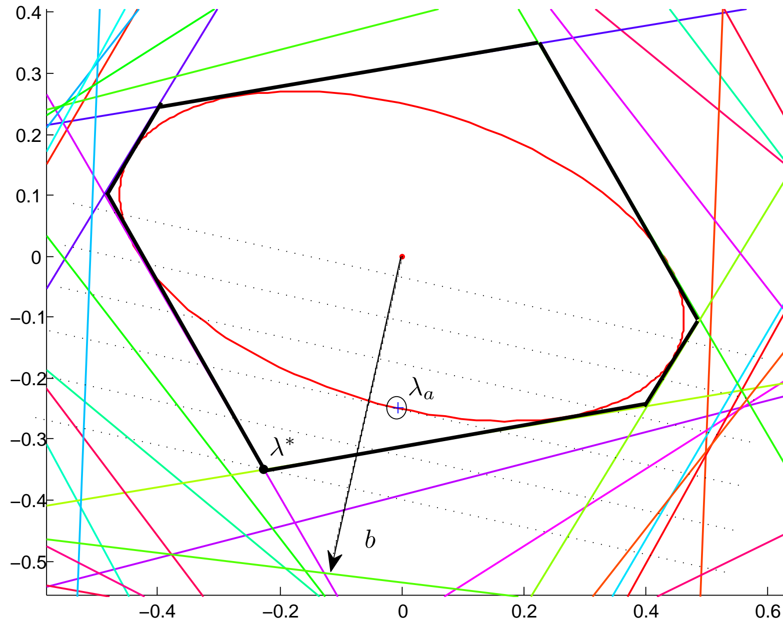


Figure 4.2: Difference between the solutions of the linear program (4.3) and its approximated problem (4.4). The solution of (4.3) is represented by λ^*, while the solution of (4.4) is represented by λ_a. The dimensions of the problem were: $A \in \mathbb{R}^{2\times 50}$, $b \in \mathbb{R}^2$. Note that, although there are 100 linear constraints, only 6 of them define the ellipsoid.

The dual function of (4.7) is

$$L(\lambda) = \inf_x \left[\left(A^T\lambda\right)^T x + \beta\|x\|_1 \right] + \inf_z \left[\frac{1}{2}\|z-b\|^2 - \lambda^T z \right]. \tag{4.8}$$

The infimum on x of the left–hand term of (4.8) is only finite when $\|A^T\lambda\|_\infty \leq \beta$, being zero in this case. The right–hand term is differentiable on z, so we get $z^\star = b+\lambda$, being its infimum equal to $-\frac{1}{2}\|\lambda\|^2 - b^T\lambda$. Therefore,

$$L(\lambda) = \begin{cases} -\frac{1}{2}\|\lambda\|^2 - b^T\lambda & , \quad \|A^T\lambda\|_\infty \leq \beta \\ -\infty & , \quad \|A^T\lambda\|_\infty > \beta \end{cases}.$$

So, the dual problem of (4.7) is

$$d^\star = \max_{\|A^T\lambda\|_\infty \leq \beta} -\frac{1}{2}\|\lambda\|^2 - b^T\lambda \iff \min_{\|A^T\lambda\|_\infty \leq \beta} \frac{1}{2}\|\lambda\|^2 + b^T\lambda,$$

and making the change of variable $\nu = -\frac{\lambda}{\beta}$,

$$\begin{aligned} &\iff \min_{\|A^T\nu\|_\infty \leq 1} \frac{\beta^2}{2}\|\nu\|^2 - \beta b^T\nu \\ &\iff \min_{\|A^T\nu\|_\infty \leq 1} \|\nu\|^2 - \frac{2}{\beta}b^T\nu \\ &\iff \min_{\|A^T\nu\|_\infty \leq 1} \|\nu\|^2 - \frac{2}{\beta}b^T\nu + \frac{1}{\beta^2}\|b\|^2 - \frac{1}{\beta^2}\|b\|^2 \\ &\iff \min_{\|A^T\nu\|_\infty \leq 1} \|\nu - \frac{1}{\beta}b\|^2, \end{aligned} \tag{4.9}$$

which is a problem of a point projection on a set. Namely, the solution of (4.9) is the point ν belonging to set $\mathcal{P} = \left\{\mu \, : \, \|A^T\mu\|_\infty \leq 1\right\}$ that is closest to b/β.

Again, we can solve the dual problem (4.9) by replacing the set $\mathcal{P}$ by its inner ellipsoidal approximation, $\mathcal{E}(B, d)$, and get an approximate solution. So, an approximation of the problem (4.9) is

$$\min_{\nu \in \mathcal{E}(B,d)} \|\nu - \frac{1}{\beta} b\|^2, \tag{4.10}$$

where $\mathcal{E}(B, d)$ is the maximum volume ellipsoid inscribed in the set $\mathcal{P}$. In appendix C.1, an algorithm that solves (4.10) is described and proved to converge. That algorithm has the property of its convergence being very fast.

Figure 4.3 shows a two–dimensional example of the difference between the solutions of (4.9) and (4.10), for the same polyhedron of Figure 4.2. Again, note that the solution ν_a of (4.10) is a good starting point for an interior–point algorithm that solves (4.9).

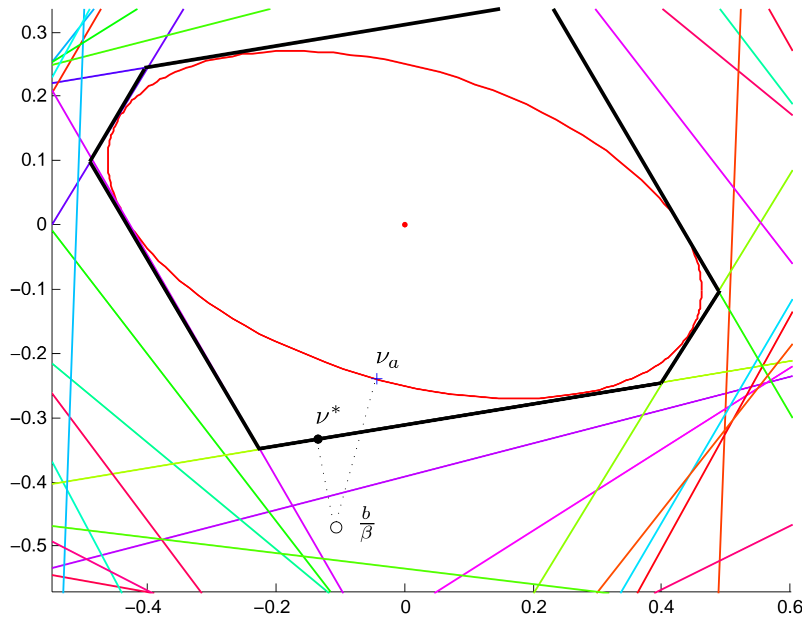


Figure 4.3: Difference between the solutions of (4.9), represented by ν^*, and (4.10), represented by ν_a. That is, the projection of a point on a polyhedron and on the inscribed ellipsoid that best approximates that polyhedron.

Finding the optimal primal variable. We still need to see how the knowledge of the dual optimal variable ν^* can help us to find the optimal primal variable x^*, which solves (4.6).

For an optimal dual variable $\lambda^* = -\beta\nu^*$, the infimum of the left–hand side term of (4.8):

$$x^* \in \inf_x \left[\left(A^T \lambda^*\right)^T x + \beta \|x\|_1 \right]$$

gives a corresponding optimal primal variable, x^*. Notice that the restriction set of (4.9) guarantees that such infimum is always finite. Let a_i be any column of the matrix A for $i = 1, \ldots, n$. Then, by analyzing the derivative of each function $a_i^T \lambda^* x_i + \beta |x_i|$ for the cases when x_i is positive, negative and zero, we come to the conclusion that:

$$\begin{cases} x_i^* > 0 & \Longrightarrow & a_i^T \lambda^* & = & -\beta \\ x_i^* < 0 & \Longrightarrow & a_i^T \lambda^* & = & \beta \\ x_i^* = 0 & \Longrightarrow & |a_i^T \lambda^*| & \leq & \beta \end{cases}.$$

Taking into account that we can't have $|a_i^T\lambda^*| > \beta$, we can conclude that

$$\begin{cases} a_i^T\lambda^* & = & -\beta & \implies & x_i^* \geq 0 \\ a_i^T\lambda^* & = & \beta & \implies & x_i^* \leq 0 \\ |a_i^T\lambda^*| & < & \beta & \implies & x_i^* = 0 \end{cases}.$$

This means that, provided that we know an optimal dual variable λ^*, we can discard the columns of A for which we already know that they aren't activated by the corresponding optimal primal solution x^*. Formally, if we define the set

$$\Omega^* = \{i \,:\, |a_i^T\lambda^*| = \beta\},$$

we can form the matrix $M = A|_{\Omega^*}$, and solve

$$\min_{u\in\mathbb{R}^{n'}} \frac{1}{2}\|Mu - b\|^2 + \beta\|u\|_1, \tag{4.11}$$

where the variable is $u \in \mathbb{R}^{n'}$, with $n' = |\Omega^*|$.

It is evident from the previous analysis that if u^* solves (4.11), then the vector x in $\mathbb{R}^n$ such that $x|_{\Omega^*} = u^*$ and is zero elsewhere solves (4.6).

So, if we solve the dual rapidly (note that finding an approximation of the optimal dual variable can be very quick), the solution of (4.6) can be obtained by solving (4.11). As the solution x^* is expected to be sparse, the reduction in the number of columns from matrix A to matrix M can be significant and, as a consequence, we solve a BPDN in a much lower dimension.

4.1.3 Generalization Of The BPDN

We seize this opportunity to introduce a new problem, which is a generalization of the basis pursuit denoising (BPDN):

$$\min_x \frac{1}{2}\|Ax - b\|_P^2 + \beta\|x\|_1, \tag{4.12}$$

where $\|y\|_P = \left(y^T P y\right)^{1/2}$ is the P–quadratic norm, being P a symmetric and positive definite matrix. We consider $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $\beta \in \mathbb{R}$.

The results from the previous subsection can be applied to solve (4.12) with a little additional effort.[1]

Using the previous approach (as for the ℓ_2-norm), (4.12) is equivalent to

$$p^\star = \min_{\substack{Ax = z \\ \text{var}\,:\,(x,z)\in\mathbb{R}^n\times\mathbb{R}^m}} \frac{1}{2}\|z - b\|_P^2 + \beta\|x\|_1. \tag{4.13}$$

The dual function of (4.13) is

$$L(\lambda) = \inf_x \left[\left(A^T\lambda\right)^T x + \beta\|x\|_1\right] + \inf_z \left[\frac{1}{2}z^T P z - (Pb)^T z - \lambda^T z\right] + \frac{1}{2}\|b\|_P^2.$$

Again, the left–hand term of the dual function is only finite (and equal to zero) when $\|A^T\lambda\|_\infty \leq \beta$. The minimizer of the right–hand term is $z^\star = P^{-1}\lambda + b$, leading to the infimum $-\frac{1}{2}\lambda^T P^{-1}\lambda - b^T\lambda - \frac{1}{2}\|b\|_P^2$. This way, the dual of (4.13) is

$$d^\star = \max_{\|A^T\lambda\|_\infty \leq \beta} -\frac{1}{2}\lambda^T P^{-1}\lambda - b^T\lambda,$$

[1] We also leave as a topic for future research finding applications where this problem can reveal important.

which is equivalent to

$$\begin{array}{cc} \min & \frac{1}{2}\lambda^T P^{-1}\lambda + b^T\lambda. \\ \|A^T\lambda\| \leq \beta & \\ \mathrm{var}: \lambda \in \mathbb{R}^m & \end{array} \tag{4.14}$$

The optimization problem (4.14) is a generic quadratic program over a polyhedron. Again, we can replace the set $\mathcal{P} = \{\lambda : \|A^T\lambda\|_\infty \leq 1\}$ by its maximum volume inner ellipsoid $\mathcal{E}(B,d)$, yielding

$$\min_{\lambda\in\mathcal{E}(B,d)} \frac{1}{2}\lambda^T P^{-1}\lambda + b^T\lambda. \tag{4.15}$$

We must notice that problem (4.15) is slightly different from problem (4.10). They are both quadratic problems over an ellipsoid, but the matrix that affects the quadratical term in (4.10) is the identity matrix. However, as P^{-1} is a symmetric and positive definite matrix, the solution of (4.15) can be easily computed. The respective algorithm is described in appendix C.2.

If there is an interior–point algorithm that solves (4.14), the solution of (4.15), which can be computed very quickly, can be used as a starting–point for that algorithm.

An optimal primal variable can be found using the method described in subsection 4.1.2.

4.1.4 The Quest For A "Perfect" Interior–Point Algorithm

In subsections 4.1.1, 4.1.2 and 4.1.3 we always relied on the existence of an interior–point method that can solve linear programs (first subsection) or quadratic programs (other subsections). In fact, such methods exist but become too slow for large scale problems. To exemplify, we will analyze an interior–point method called the *barrier method* [6].

The barrier method. The idea of a barrier method is to solve a problem (which, here, we particularize to a linear program)

$$\begin{array}{cc} \min & c^T x, \\ Ax \leq b & \\ \mathrm{var}: x \in \mathbb{R}^n & \end{array} \tag{4.16}$$

where $A \in \mathbb{R}^{m\times n}$, $x, c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, by transforming it into a sequence of them

$$\{\min_{x\in S} c^T x - \frac{1}{t_k}\sum_{i=1}^{m}\log(b_i - a_i^T x)\}, \tag{4.17}$$

indexed by k. Each vector a_i is a row of the matrix A, *i.e.*

$$A = \begin{bmatrix} a_1^T \\ \vdots \\ a_i^T \\ \vdots \\ a_m^T \end{bmatrix},$$

and $S = \{x : b_i - a_i^T x > 0\,,\, i = 1,\ldots,m\}$.

It seems that we replaced a simple problem by a sequence of complicated problems, but it turns out that the minimization of each problem in (4.17) can be carried out using a very fast method: *Newton's method* [6, page 487]. The reason why we solve a sequence of problems instead of just one is because Newton's method requires a starting point. This way, at the iteration $k+1$, we can use the solution

found at the iteration k as a starting point. Note that, as t_k approaches $+\infty$, the optimal solution and the optimal value of (4.17) approaches respectively the optimal solution and the optimal value of (4.16). We don't solve directly problem (4.17) for a large value of t_k because, without a good starting point, that problem becomes too hard to solve (the hessian of the objective becomes ill–conditioned).

Before proceeding, note that (4.17) is equivalent to

$$\{\min_{x\in S} t_k c^T x - \sum_{i=1}^{m} \log(b_i - a_i^T x)\}. \tag{4.18}$$

Newton's method requires the evaluation of the gradient and the hessian of the cost function of the problem that we are minimizing. Let $f_k(x)$ be the objective of a problem of (4.18). Then, it can be shown that

$$\nabla f_k(x) = t_k c + A^T d(x)$$

and

$$\nabla^2 f_k(x) = \sum_{i=1}^{m} \frac{1}{(b_i - a_i^T x)^2} a_i a_i^T, \tag{4.19}$$

where $d(x)$ is the vector in $\mathbb{R}^m$ such that $d_i(x) = 1/(b_i - a_i^T x)$, for $i = 1, \ldots, m$.

Computer simulations show that the construction of the hessian $\nabla^2 f_k(x)$, at each iteration k, is the weakest point of this method, for the cases where m and n have high values, since it consumes most of the time of the algorithm.

Thus, the research proposal that we make is either trying to overcome this hessian construction in the barrier method, or developing interior–point algorithms that don't require the evaluation of hessian matrices of the kind of (4.19), *i.e.*, that involve the sum of m matrices of size $n \times n$, with $m \gg n$. In a word, developing "fast " interior point methods for large scale problems.

4.2 Ball Approximation

Recall that in Figure 4.2 we mentioned the fact that, although we had 100 linear constraints defining the set $\mathcal{P} = \{\lambda : \|A^T\lambda\|_\infty \leq 1\}$, only 6 of them really defined the polyhedron. This happened because the columns of A had arbitrary norms.

However, if we restrict these columns to have unit–norm, the polyhedron $\mathcal{P}$ now resembles a ball (see Figure 4.4). In this case, each linear constraint has a (small) contribution to define the polyhedron. The assumption of unit–norm columns arises in many contexts and applications (see [18, 17, 6, 27]).

We will see that by replacing the set $\mathcal{P}$ by the unit–norm ball, $B(0,1) = \{\lambda : \|\lambda\| \leq 1\}$, in both problems (4.3) and (4.9), we get an approximate solution of the respective dual optimal variable.

So, in the case where each column of A has unit–norm, the assumption of the previous knowledge of the matrix A can be dropped, since the approximation can be done in real time. Indeed, we already know what the best approximation is, hence we don't need to do any precalculation before the "arrival" of the vector b.

Approximating the set $\mathcal{P}$. Figure 4.4 gives a good motivation for approximating the set $\mathcal{P}$ by a ball $B(0,R)$, centered at the origin and with radius R. Now, we will see that $R = 1$ is the best choice.

To find the maximum volume ball $B(0,R) = \{\lambda : \|\lambda\| \leq R\}$ that is inscribed inside the set $\mathcal{P}$, one can formulate the following optimization problem:

$$\max_{B(0,R)\subset\mathcal{P}} R. \tag{4.20}$$

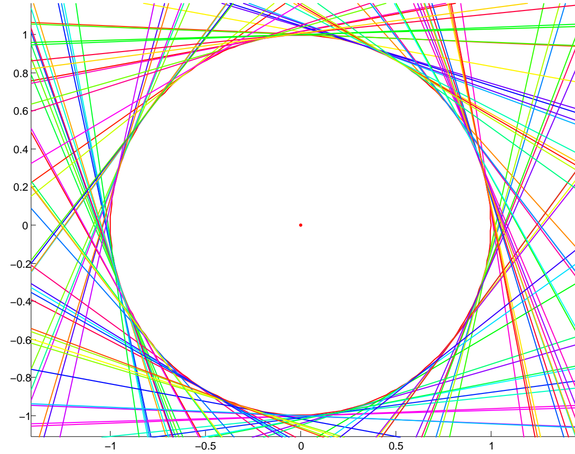


Figure 4.4: Polyhedron $\mathcal{P}$, where the matrix $A \in \mathbb{R}^{2\times 50}$ is random.

Recall the definition of the set $\mathcal{P}$: $\mathcal{P} = \{\lambda : \|A^T\lambda\|_\infty \leq 1\}$. Let a_i designate a column of the matrix A and let also $H^-_{a_i,1}$ designate the set $\{\lambda : a_i^T\lambda \leq 1\}$, for $i = 1,\ldots,n$. Then, the constraint of (4.20) can be written as

$$\begin{cases} B(0,R) & \subset & H^-_{a_i,1}\,, & \text{for} \quad i = 1,\ldots,n \\ B(0,R) & \subset & H^-_{-a_i,1}\,, & \text{for} \quad i = 1,\ldots,n \end{cases}.$$

Let now r_i designate either a_i or $-a_i$. The condition $B(0,R) \subset H^-_{r_i,1}$ can be written as

$$\left[\max_{\|\lambda\|\leq R} r_i^T\lambda\right] \leq 1. \tag{4.21}$$

The solution of the optimization problem in (4.21) is $\lambda_a = R(r_i/\|r_i\|)$ (in Figure 4.1 there is a graphical explanation of the solution of a similar problem). Replacing it in (4.21), we get $R\|r_i\| \leq 1$. Therefore, problem (4.20) is equivalent to

$$\begin{array}{ccc} & \max & R. \\ R & \leq & \frac{1}{\|a_1\|} \\ & \vdots & \\ R & \leq & \frac{1}{\|a_n\|} \\ R & \leq & \frac{1}{\|-a_1\|} \\ & \vdots & \\ R & \leq & \frac{1}{\|-a_n\|} \end{array}$$

Since $\|a_i\| = \|-a_i\| = 1$ for all $i = 1,\ldots,n$, we get

$$\max_{R\leq 1} R,$$

which has the obvious solution $R = 1$.

The resulting approximated problems. For the BP, if we replace the constraint $\|A^T\lambda\|_\infty \leq 1$ by $\lambda \in B(0,1)$ in problem (4.3), we get

$$\max_{\|\lambda\|\leq 1} b^T\lambda. \tag{4.22}$$

It is straightforward to see that the vector $\lambda_a = b/\|b\|$ solves it. Figure 4.5 shows the difference between the solution of this approximated problem and the exact solution λ^* of (4.3). Note that, unlike Figures 4.2

and 4.3, in Figure 4.5 all the columns of A contribute to define the polyhedron $\mathcal{P}$.

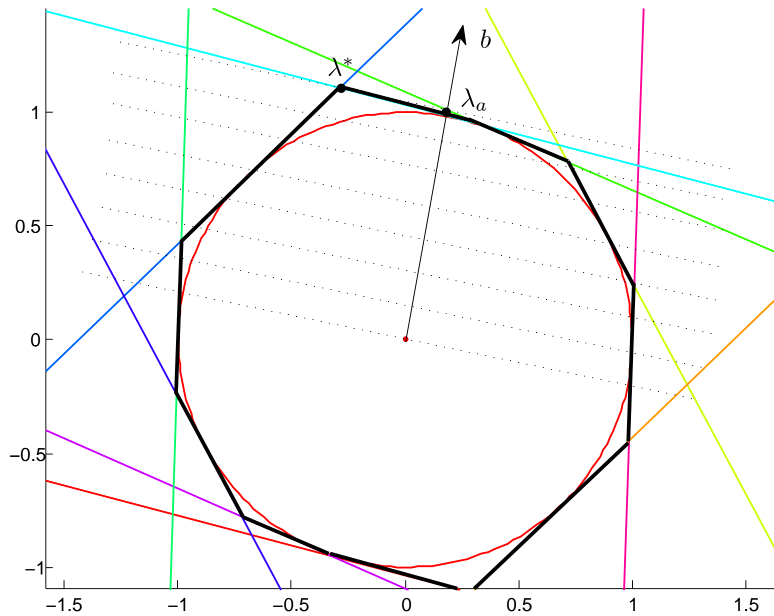


Figure 4.5: Difference between the solutions of the approximated problem (4.22), represented by λ_a, and the initial problem (4.3).

On the other hand, for the BPDN, if we replace the constraint $\|A^T\nu\|_\infty \leq 1$, in (4.9), by $\nu \in B(0,1)$, the resulting problem is

$$\min_{\|\nu\|\leq 1} \|\nu - \frac{1}{\beta}b\|^2. \tag{4.23}$$

The solution of (4.23) is the point b/β, when $\|b/\beta\| \leq 1$. When $\|b/\beta\| > 1$, the solution is $b/\|b\|$, *i.e.*,

$$\begin{cases} \nu_a = \frac{b}{\beta}, & \text{if } \|\frac{b}{\beta}\| \leq 1 \\ \nu_a = \frac{b}{\|b\|}, & \text{if } \|\frac{b}{\beta}\| > 1 \end{cases}.$$

4.3 Ellipsoidal And Ball Approximations In Generic ℓ_1-norm Problems

This section concerns a generalization of the techniques that we have seen in this chapter.

We are interested in problems where we want to minimize a generic function on the variable x plus a "sparsifying factor" $\|x\|_1$. Namely, problems of the kind

$$\min_{\substack{Ax=b \\ x\in \mathrm{dom} f}} \quad f(x) + \rho\|x\|_1, \tag{4.24}$$

where $A \in \mathbb{R}^{m\times n}$ with $m < n$ is a full–rank matrix, b is a given vector in $\mathbb{R}^m$, $f : \mathbb{R}^n \to \mathbb{R}$ is a generic convex function with domain $\mathrm{dom} f$ and $\rho \in \mathbb{R}$. The variable of the problem is $x \in \mathbb{R}^n$.

Let the matrix A be arbitrary, in the sense that its columns have arbitrary norms. The particularization for the case where these norms are unitary can be easily done, following the steps given in section 4.2.

Solving (4.24) First of all, note that (4.24) is equivalent to

$$\min_{\substack{x_1 \in \operatorname{dom} f \\ Ax_2 = b \\ x_1 = x_2}} \quad f(x_1) + \rho \|x_2\|_1. \tag{4.25}$$

The Lagrangian function $L : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ of (4.25), if we dualize only its last constraint, is

$$\begin{aligned} L(x_1, x_2, \lambda) &= f(x_1) + \rho\|x_2\|_1 - \lambda^T x_1 + \lambda^T x_2 \\ &= \left(f(x_1) - \lambda^T x_1\right) + \left(\rho\|x_2\|_1 + \lambda^T x_2\right). \end{aligned}$$

The corresponding dual function is

$$L(\lambda) = \inf_{x_1 \in \operatorname{dom} f} \left(f(x_1) - \lambda^T x_1\right) + \inf_{Ax_2 = b} \left(\rho\|x_2\|_1 + \lambda^T x_2\right). \tag{4.26}$$

Note that once the optimal dual variable λ^* is known, one can recover the corresponding optimal primal variable $x_1^* = x_2^* = x^*$ by solving either the left–hand term or the right–hand term of (4.26), with λ replaced by λ^*.

Also note that

$$\inf_{x_1 \in \operatorname{dom} f} \left(f(x_1) - \lambda^T x_1\right) = -f^*(\lambda),$$

where $f^* : \mathbb{R}^n \to \mathbb{R}$ is the *conjugate function* of f [6, page 91]. So, the approach that we are taking works well when the conjugate function of f is known and easy to evaluate at each point. Some examples functions and its conjugate functions are in [6].

The right–hand side term of (4.26) isn't always finite, so we must use a projection method to maximize $L(\lambda)$, for example the *projected subgradient method* [5].

What is interesting about this problem is that, for each λ, we can reduce the dimensions of

$$\inf_{Ax_2 = b} \left(\rho\|x_2\|_1 + \lambda^T x_2\right), \tag{4.27}$$

by solving its dual

$$\min_{\|A^T\mu + \lambda\|_\infty \le \rho} b^T \mu, \tag{4.28}$$

where the variable is $\mu \in \mathbb{R}^m$. Note that the constraint of (4.28) is a polyhedron $\{\mu : \|A^T\mu\|_\infty \le \rho\}$ centered at $-\lambda$. Therefore, if we have an approximation of this polyhedron beforehand, that approximation remains valid for any value of λ, provided that we also center it at $-\lambda$. The knowledge of an optimal dual variable μ^* of (4.28) allows us to discard most of the columns of the matrix A that aren't activated by the respective primal optimal solution.

We leave the analysis of this algorithm for future research, as well as finding a simple description of the condition that guarantees that (4.27) has a finite infimum.

Chapter 5

Conclusions

Throughout this dissertation we have seen algorithms that solve the basis pursuit (BP) distributedly. The BP is an important problem that arises when we want sparse representations of a phenomenon that can be explained as a linear combination of some causes. We took for granted that we always knew the matrix of causes A beforehand with respect to the phenomenon b. This fact, which makes sense in many practical applications, allows us to "distribute" blocks of the matrix A by several nodes/processors before the "arrival" of vector b. We considered two cases: an horizontal and a vertical partition of the matrix A.

For the horizontal partition, we saw three different algorithms: one that solves a dual problem in order to select the important columns of A and then solves a smaller version of the BP; and other two that use the method of multipliers to find the primal (and also the dual) variable. In each step of this method, there is the need of solving a non–separable optimization problem. To overcome this issue we used two well–known methods that can induce separability: the Diagonal Quadratic Approximation (DQA) and the Nonlinear Gauss–Seidel. Theoretically, these algorithms were known to converge only for differentiable cost functions and, in our problem, we had a non–differentiable one. So, we developed a new concept of functions to which we gave the name of *rigid functions* and proved that the DQA and the Nonlinear Gauss–Seidel are still guaranteed to converge for this kind of functions. As a consequence, we also proved convergence for our algorithms.

Concerning the vertical partition of the matrix A, we applied the previous results to ensure that the method of multipliers, concatenated with either the DQA or the Nonlinear Gauss–Seidel, converged when applied to a convex optimization problem with a rigid function as the objective. This way, we developed a distributed algorithm that solves the BP when the matrix A is partitioned into vertical blocks. We have seen that the inefficiency of this algorithm is due to the formulation of the distributed optimization problem: we transformed a problem with a "big size" variable into many of them. In terms of future research, this is perhaps the most important subject to explore, since there are no other algorithms to solve the BP with a vertical partition of A and the one that we have is too inefficient (slow). Also, this partition of matrix A arises naturally in a new area that is having an increasingly importance in applications: sensor networks.

Finally, the last part of this dissertation concerned the topic of centralized algorithms to solve the BP and the basis pursuit denoising (BPDN) fastly. We didn't present any new algorithm, instead we just pointed out some evidences that might lead to it. In fact, if we assume that matrix A is known beforehand with respect to b, this can provide us useful information or just prepare us better before the vector b arrives. This issue hasn't been properly explored in current literature, but we believe that it can lead to very efficient algorithms.

Bibliography

[1] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin. A simple proof of the restricted isometry property for random matrices. Technical report, Rice University, January 2007.

[2] R. Baraniuk, J. Romberg, and M. Wakin. Tutorial on compressive sampling. February 2008. Available at the Rice University Webpage.

[3] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2 edition, 1999.

[4] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific, 1997. Available at http://web.mit.edu/dimitrib/www/pdc.html.

[5] S. Boyd and A. Mutapic. Subgradient methods, notes for ee364b. Stanford University, available at http://www.stanford.edu/class/ee364b/notes/subgrad_method_notes.pdf, 23 January 2007.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. Also available at http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[7] E.J. Candès. Compressive sampling. European Mathematical Society, Proceedings of the International Congress of Mathematicians, Madrid, Spain, 2006.

[8] E.J. Candès. The restricted isometry property and its implications for compressed sensing. Preprint submitted to the Académie des sciences, February 2008.

[9] E.J. Candès, J. Romberg, and T. Tao. Stable signal recovery from incomplete and inaccurate measurements. *Comm. Pure Appl. Math.*, 59(8):1207–1223, August 2006.

[10] E.J. Candès and T. Tao. Near–optimal signal recovery from random projections and universal encoding strategies. *submitted to IEEE Trans. Inform. Theory*, November 2004. Available on the ArXiV preprint server: math.CA/0410542.

[11] E.J. Candès and T. Tao. Decoding by linear programming. *IEEE Trans. Inform. Theory*, 51(12):4203–4215, December 2005.

[12] E.J. Candès and M.B. Wakin. "people hearing without listening:" an introduction to compressive sampling. Applied and Computational Mathematics, California Institute of Technology.

[13] S.S. Chen, D.L. Donoho, and M.A. Saunders. Atomic decomposition by basis pursuit. *Society for Industrial and Applied Mathematics*, 20(1):33–61, 1998.

[14] F. Dabbene, P. Gay, and B.T. Polyak. Recursive algorithms for inner ellipsoidal approximation. *Elsevier*, 39(10):1773–1781, October 2003.

[15] D. Estrin, A. Sayeed, and M. Srivastava. Tutorial wireless sensor networks. http://nesl.ee.ucla.edu/tutorials/mobicom02/.

[16] M.A.T. Figueiredo, R.D. Nowak, and S.J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. To Appear in the IEEE Journal of Selected Topics in Signal Processing, 2007.

[17] P. Frossard, P. Vandergheynst, R.M.F. Ventura, and M. Kunt. A posteriori quantization of progressive matching pursuit streams. *IEEE Transactions on Signal Processing*, 52(2):525–535, February 2004.

[18] R. Gribonval and E. Bacry. Harmonic decomposition of audio signals with matching pursuit. *IEEE Transactions on Signal Processing*, 51(1):101–111, January 2003.

[19] S.S. Iyengar and R.R. Brooks. *Distributed Sensor Networks*. Chapman & Hall/CRC, 2004.

[20] S.J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large scale $l1$-regularized least squares. To Appear in IEEE Journal on Selected Topics in Signal Processing.

[21] E. Kreyszig. *Advanced Engineering Mathematics*. John Wiley and Sons, Inc., 8 edition, 1999.

[22] T.K. Moon and W.C. Stirling. *Mathematical Methods and Algorithms for Signal Processing*. Prentice Hall, 2000.

[23] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2003.

[24] A. Ruszczynski. Augmented lagrangian decomposition for sparse convex optimization. *International Institute for Applied Systems Analysis*, 1992.

[25] A.S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, 2006.

[26] J.A. Tropp. Just relax: Convex programming methods for subset selection and sparse approximation. Technical report, ICES Report 0404, The University of Texas at Austin, 2004.

[27] J.A. Tropp. Just relax: Convex programming methods for identifying sparse signals. *IEEE Transactions on information theory*, 51(3):1030–1051, March 2006.

[28] Rice university webpage. Compressive sensing resources. `http://www.dsp.ece.rice.edu/cs/`.

[29] A. Zakarian. *Nonlinear Jacobi and ϵ-relaxation methods for parallel network optimization*. PhD thesis, University of Wisconsin — Madison, 1995.

Appendix A

Example of a non-rigid function

In subsection 2.2.1, it is introduced the concept of a rigid function. It is easy to see that every convex and differential function is rigid. Here, though, it will be presented a convex subdifferentiable function that isn't rigid. In fact, the point that makes this function non–rigid is also a stationary point of the DQA algorithm presented in page 32. This means that, although this point minimizes the function along all the coordinate axis, it is not a minimizer, not even a local one. Note that being non–rigid, theorems 5 and 6 cannot be applied.

The function is

$$f(x,y) = \max\left\{(x-1)^2 + (y+1)^2, (x+1)^2 + (y-1)^2\right\}.$$

Owing to the fact that $f(x,y)$ is the maximum of two (strict) convex functions, $f(x,y)$ is also convex. Also, it is subdifferentiable over all $\mathbb{R}^2$. Its graph is plotted in Figure A.1.

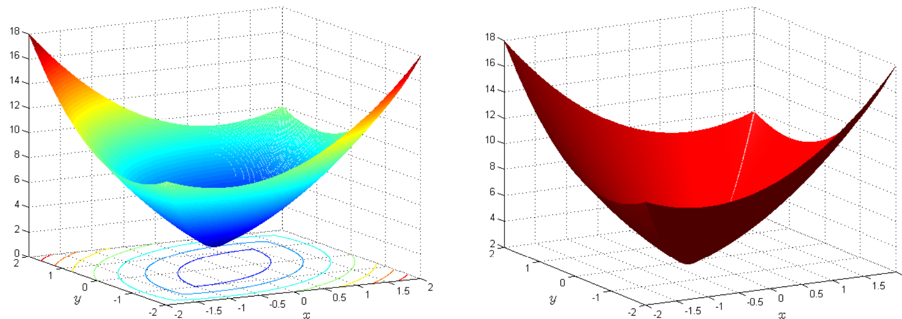


Figure A.1: Graphics with different shadings of the function $\max\left\{(x-1)^2 + (y+1)^2, (x+1)^2 + (y-1)^2\right\}$. Note that the function is non-differentiable along the line $x = y$. Though non-rigid, this function is convex and subdifferentiable on all $\mathbb{R}^2$.

Any point along the line $x = y$ could be used to prove that this function is non–rigid. We will use the point $(x,y) = (1,1)$.

First of all, note that the minimum of $f(x,y)$ is attained at $(0,0)$. This can be seen by noting that $f(x,y) = f(-x,-y)$ and

$$f(0,0) = 2 \leq \frac{f(x,y) + f(-x,-y)}{2} = f(x,y).$$

The above inequality follows from the fact that the minimum of

$$\begin{aligned} f(x,y)+f(-x,-y) &= \max\{(x-1)^2+(y+1)^2,(x+1)^2+(y-1)^2\} \\ &\quad + \max\{(x+1)^2+(y-1)^2,(x-1)^2+(y+1)^2\} \\ &= (x-1)^2+(x+1)^2+(y+1)^2+(y-1)^2 \end{aligned}$$

is 4 (it is achieved for $(x,y)=(0,0)$).

Let's now see that $(1,1)$ is a minimizer along both x and y. Indeed,

$$f(1,1)=4\le \max\left\{(x-1)^2+4,(x+1)^2\right\}=f(x,1),\quad \forall_{x\in\mathbb{R}},$$

and

$$f(1,1)=4\le \max\left\{(1+y)^2,4+(y-1)^2\right\}=f(1,y),\quad \forall_{y\in\mathbb{R}}.$$

On the other hand, the directional derivative along the vector $v=(-1,-1)$ is negative:

$$\begin{aligned} f'\left((1,1);v\right) &= \lim_{h\downarrow 0}\frac{f(1-h,1-h)-f(1,1)}{h} \\ &= \lim_{h\downarrow 0}\frac{\max\left\{h^2+(2-h)^2,(2-h)^2+h^2\right\}-4}{h} \\ &= \lim_{h\downarrow 0}\frac{h^2+4-2h+h^2-4}{h} \\ &= \lim_{h\downarrow 0}\quad 2h-2 \\ &= -2 \\ &< 0. \end{aligned}$$

The conclusion is that $f(x,y)$ is a non–rigid function.

Appendix B

A Simple Subgradient Based Algorithm For Quadratic Programming

In chapter 3 it is proposed an algorithm to solve the basis pursuit (BP) problem with a vertical partition of the matrix A (see equation (3.1)). There, the final algorithm relies on the fact that each processor involved in the optimization is able to solve problem (3.8), which we replicate here:

$$\min_{A_p y_p = b_p} \frac{1}{P}\|y_p\|_1 + \left[\lambda_p^k - \lambda_{p-1}^k - 2\rho^k(x_{p-1}^{t+1} + x_{p+1}^t)\right]^T y_p + 2\rho^k\|y_p\|^2. \tag{B.1}$$

Our purpose in this appendix is to present a very simple algorithm that can solve (B.1) and also that can be implemented in simple processors.

To do that, we change the notation for simplicity purposes, and write (B.1) as

$$p^* = \min_{Ax=b} \|x\|_1 + v^T x + c\|x\|^2, \tag{B.2}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $v \in \mathbb{R}^n$ and $c \in \mathbb{R}$ are constants, and $x \in \mathbb{R}^n$ is the variable of optimization.

The Lagrangian $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ of (B.2) is

$$\begin{aligned} L(x,\lambda) &= \|x\|_1 + v^T x + c\|x\|^2 + (A^T\lambda)^T x - b^T\lambda \\ &= \|x\|_1 + (v + A^T\lambda)^T x + c\|x\|^2 - b^T\lambda \\ &= \sum_{i=1}^n \left(|x_i| + \gamma_i(\lambda)x_i + cx_i^2\right) - b^T\lambda, \end{aligned}$$

where the vector $\gamma(\lambda) \in \mathbb{R}^n$ is, by definition, $\gamma(\lambda) := v + A^T\lambda$. Therefore, the dual Lagrangian function can be written as

$$L(\lambda) = \sum_{i=1}^n \left[\inf_{x_i}(|x_i| + \gamma_i(\lambda)x_i + cx_i^2)\right] - b^T\lambda. \tag{B.3}$$

The solution of each subproblem of (B.3) can be found in closed form, if we consider the following cases:

- If $x_i > 0$ and the calculate the derivative of $|x_i| + \gamma_i(\lambda)x_i + cx_i^2$, we get

$$1 + \gamma_i(\lambda) + 2cx_i = 0 \quad \Longleftrightarrow \quad x_i = -\frac{1+\gamma_i(\lambda)}{2c}. \tag{B.4}$$

  As we assumed that $x_i > 0$, we must have $\gamma_i(\lambda) < -1$.

- If $x_i < 0$ and we do the same procedure, we get

$$-1 + \gamma_i(\lambda) + 2cx_i = 0 \quad \Longleftrightarrow \quad x_i = -\frac{-1 + \gamma_i(\lambda)}{2c}, \tag{B.5}$$

  which implies that $\gamma_i(\lambda) > 1$.

- Finally, when $x_i = 0$, we have

$$0 \in [-1, 1] + \gamma_i(\lambda) \quad \Longleftrightarrow \quad \gamma_i(\lambda) \in [-1, 1]. \tag{B.6}$$

By assumption, the linear system $Ax = b$ is underdetermined and, also by assumption, the matrix A has full–rank (see the context of problem (B.1) in chapter 3). Thus, *Slater's condition* hold and, hence, also strong duality. This way, the dual problem is

$$d^* = p^* = \max_{\lambda} \sum_{i=1}^{n} f_i(\lambda) - b^T \lambda, \tag{B.7}$$

where each function $f_i : \mathbb{R}^n \to \mathbb{R}$ is given by

$$f_i(\lambda) = \inf_{x_i} (|x_i| + \gamma_i(\lambda) x_i + c x_i^2),$$

for $i = 1, \ldots, n$.

Note that (B.7) is equivalent to

$$\min_{\lambda} b^T \lambda - \sum_{i=1}^{n} f_i(\lambda). \tag{B.8}$$

The cost function of (B.8) can be written as the supremum of differentiable functions. As so, it is not differentiable itself, but is subdifferentiable. It is also convex on the variable λ. Thus, we are in conditions to apply the subgradient method (algorithm 1 on page 14). The only thing we have to know in each iteration is a subgradient of the objective of (B.8), for a fixed λ^k. In fact, it is straightforward to see that $g^k = b - Ax^*(\lambda^k)$ is a subgradient of the objective of (B.8), where each component of $x^*(\lambda^k)$ can be evaluated using equations (B.4), (B.5) and (B.6).

In Figure B.1 a comparison between the performances of this algorithm (subgradient method applied to (B.8)) and the algorithm used by *Yalmip/Matlab* is depicted. Note that, although the main focus here was to illustrate the simplicity of an algorithm for solving (B.2), competitive results were achieved.

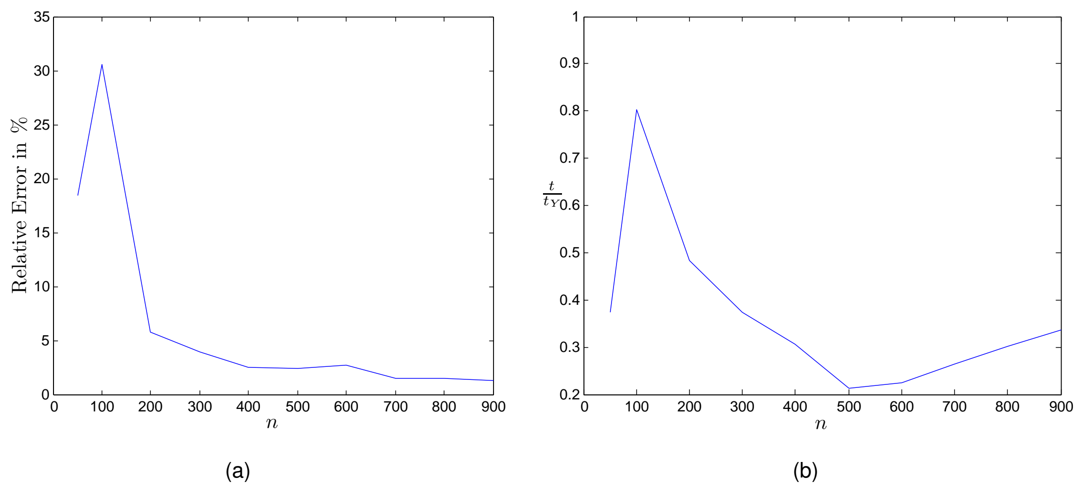


Figure B.1: Comparison in terms of the relative error and time consumed of the solutions of the problem (B.2) given by the presented algorithm and by *Yalmip/Matlab*. The plots are parameterized by the horizontal dimensions of the matrix A, n, although the vertical dimensions have also changed, accompanying n. In (b), it is represented the fraction t/t_Y, where t is the time consumed by the proposed algorithm, and t_Y is the time consumed by *Yalmip/Matlab*. Each point in both plots is the mean of 50 random experiences for the respective dimensions. The high error for small dimensions can only be explained by any outlier.

Appendix C

Quadratic Program Over An Ellipsoid

C.1 Point Projection Over An Ellipsoid

In this section we describe an algorithm to solve the problem of projecting a point $p \in \mathbb{R}^n$ on an ellipsoid. In the meanwhile, we also prove that it converges. There are several ways of representing mathematically an ellipsoid. Throughout chapter 4 we preferred to use the following:

$$\mathcal{E}(B, d) = \{Bx + d \,:\, \|x\| \leq 1\}, \tag{C.1}$$

where B is a non–singular and square matrix. However, the ellipsoid description that best fits our purposes here is

$$\mathcal{E}_{\text{alt}}(B_{\text{alt}}, d) = \left\{x \,:\, (x - d)^T B_{\text{alt}}^{-1} (x - d) \leq 1\right\}, \tag{C.2}$$

where B_{alt} is symmetric and positive definite. Note that (C.1) and (C.2) represent the same set whenever $B = B_{\text{alt}}^{1/2}$.

We can naturally consider that $p \notin \mathcal{E}_{\text{alt}}(B_{\text{alt}}, d)$, since the solution is trivial otherwise.

The point projection problem can be formulated as an optimization problem

$$\min_{(x-d)^T B_{\text{alt}}^{-1}(x-d) \leq 1} \frac{1}{2}\|x - p\|^2, \tag{C.3}$$

where the variable is x. We consider that $B_{\text{alt}} \in \mathbb{R}^{n \times n}$, as well as $x, d \in \mathbb{R}^n$. By multiplying the constraining equation of (C.3) by $1/2$, we get

$$\min_{\frac{1}{2}(x-d)^T B_{\text{alt}}^{-1}(x-d) \leq \frac{1}{2}} \frac{1}{2}\|x - p\|^2. \tag{C.4}$$

The KKT-system for (C.4) is

$$\begin{cases} x^\star \in \arg\min_x \frac{1}{2}\|x - p\|^2 + \frac{1}{2}\mu^\star (x - d)^T B_{\text{alt}}^{-1} (x - d) - \frac{1}{2}\mu^\star & \text{(stationarity)} \\ (x^\star - d)^T B_{\text{alt}}^{-1} (x^\star - d) \leq 1 & \text{(primal feasibility)} \\ \mu^\star \geq 0 & \text{(dual feasibility)} \\ \mu^\star \left[(x^\star - d)^T B_{\text{alt}}^{-1} (x^\star - d) - 1\right] = 0 & \text{(complementary slackness)} \end{cases}$$

We denote, respectively, by $x^\star$ and $\mu^\star$ the primal and the dual optimal variables (x^* and μ^* solve the KKT-system).

Since the objective function in the stationarity equation is differentiable, we can replace it by the

equation

$$\nabla_x \left(\frac{1}{2}\|x-p\|^2 + \frac{1}{2}\mu^\star (x-d)^T B_{\text{alt}}^{-1} (x-d) - \frac{1}{2}\mu^\star\right)\Bigg|_{x^\star} = 0 \iff x^\star - p + \mu^\star B_{\text{alt}}^{-1} (x^\star - d) \;=\; 0$$

$$\iff x^\star - d + \mu^\star B_{\text{alt}}^{-1} x^\star - \mu^\star B_{\text{alt}}^{-1} d \;=\; p - d$$

$$\iff \left(I + \mu^\star B_{\text{alt}}^{-1}\right)(x^\star - d) \;=\; p - d,$$

where I is the identity matrix of dimension n. As I and B_{alt} are both positive definite matrices (hence also B_{alt}^{-1}), so is $I + \mu^\star B_{\text{alt}}^{-1}$, also by the dual feasibility equation. Therefore, $(I + \mu^\star B_{\text{alt}}^{-1})^{-1}$ exists.

$$\iff x^\star = d + \left(I + \mu^\star B_{\text{alt}}^{-1}\right)^{-1} (p - d)\,. \tag{C.5}$$

We still need to figure out how to find the optimal dual variable $\mu^\star$. We made the assumption at the beginning that the point p belongs to the exterior of the ellipsoid. As a consequence, the optimal primal variable $x^\star$ must belong to the border of the ellipsoid. Then, $(x^\star - d)^T B_{\text{alt}}^{-1} (x^\star - d) = 1$. Replacing (C.5) in this equation, and noticing that the set of symmetric matrices is a subspace and that the inverse of a symmetric matrix is also symmetric, we have

$$(x^\star - d)^T B_{\text{alt}}^{-1} (x^\star - d) - 1 = 0 \iff \left[\left(I + \mu^\star B_{\text{alt}}^{-1}\right)^{-1} (p-d)\right]^T B_{\text{alt}}^{-1} \left[\left(I + \mu^\star B_{\text{alt}}\right)^{-1} (p-d)\right] - 1 = 0$$

$$\iff \underbrace{(p-d)^T \left(I + \mu^\star B_{\text{alt}}^{-1}\right)^{-1} B_{\text{alt}}^{-1} \left(I + \mu^\star B_{\text{alt}}^{-1}\right)^{-1} (p-d) - 1}_{\phi(\mu^\star)} = 0. \tag{C.6}$$

As B_{alt}^{-1} is symmetric and positive definite, it can be decomposed in the product $Q\Lambda Q^T$, where Q is an orthogonal matrix and

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \succ 0,$$

i.e., $\lambda_i > 0$, for $i = 1, \ldots, n$. Note that

$$\begin{aligned} \left(I + \mu^\star B_{\text{alt}}^{-1}\right)^{-1} &= \left[Q\left(I + \mu^\star \Lambda\right) Q^T\right]^{-1} \\ &= \left(Q^T\right)^{-1} \left(I + \mu^\star \Lambda\right)^{-1} (Q)^{-1} \\ &= Q\left(I + \mu^\star \Lambda\right)^{-1} Q^T. \end{aligned}$$

Using this fact on (C.6) yields

$$\begin{aligned} \phi(\mu^\star) &= (p-d)^T Q \left(I + \mu^\star \Lambda\right)^{-1} \underbrace{Q^T Q}_{I} \Lambda \underbrace{Q^T Q}_{I} \left(I + \mu^\star \Lambda\right)^{-1} Q^T (p-d) - 1 \\ &= (p-d)^T Q \left(I + \mu^\star \Lambda\right)^{-1} \Lambda \left(I + \mu^\star \Lambda\right)^{-1} Q^T (p-d) - 1. \end{aligned} \tag{C.7}$$

Due to its diagonal structure, the matrix $(I+\mu^\star\Lambda)^{-1}$ can be expressed as

$$(I+\mu^\star\Lambda)^{-1} = \begin{bmatrix} \frac{1}{1+\mu^\star\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{1+\mu^\star\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{1+\mu^\star\lambda_n} \end{bmatrix}.$$

And, consequently, the matrix product $(I+\mu^\star\Lambda)^{-1}\Lambda(I+\mu^\star\Lambda)^{-1}$ can be written as

$$\Psi = (I+\mu^\star\Lambda)^{-1}\Lambda(I+\mu^\star\Lambda)^{-1} = \begin{bmatrix} \frac{\lambda_1}{(1+\mu^\star\lambda_1)^2} & 0 & \cdots & 0 \\ 0 & \frac{\lambda_2}{(1+\mu^\star\lambda_2)^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\lambda_n}{(1+\mu^\star\lambda_n)^2} \end{bmatrix}.$$

Making the change of variable $y = Q^T(p-d)$, (C.7) equals

$$\phi(\mu^\star) = \underbrace{\left[(p-d)^T Q\right]}_{y^T}\Psi\underbrace{\left[Q^T(p-d)\right]}_{y} - 1 = y^T\Psi y - 1.$$

Finally, equation $\phi(\mu^\star) = 0$ is equivalent to

$$\sum_{i=1}^{n}\frac{\lambda_i}{(1+\mu^\star\lambda_i)^2}y_i^2 - 1 = 0. \tag{C.8}$$

Equation (C.8) defines implicitly the optimal dual variable $\mu^\star$: it is a zero of the function ϕ.

Before proceeding, let's see what the first and second order derivatives of ϕ look like:

$$\begin{aligned}
\dot{\phi}(\mu) &= \frac{d}{d\mu}\left[\sum_{i=1}^{n}\frac{\lambda_i}{(1+\mu\lambda_i)^2}y_i^2 - 1\right] \\
&= \sum_{i=1}^{n}\frac{d}{d\mu}\left(\frac{\lambda_i}{(1+\mu\lambda_i)^2}y_i^2\right) \\
&= -2\sum_{i=1}^{n}\frac{\lambda_i^2}{(1+\mu\lambda_i)^3}y_i^2; \qquad \text{(C.9)} \\
\ddot{\phi}(\mu) &= \frac{d}{d\mu}\left[-2\sum_{i=1}^{n}\frac{\lambda_i^2}{(1+\mu\lambda_i)^3}y_i^2\right] \\
&= -2\sum_{i=1}^{n}\frac{d}{d\mu}\left(\frac{1}{(1+\mu\lambda_i)^3}\right)\lambda_i^2 y_i^2 \\
&= 6\sum_{i=1}^{n}\frac{\lambda_i^3}{(1+\mu\lambda_i)^4}y_i^2. \qquad \text{(C.10)}
\end{aligned}$$

One of the properties of the orthogonal matrices is that they are full–rank, meaning that its null–space is $\{0\}$. As we are assuming that the point p lies outside the ellipsoid $\mathcal{E}_{\text{alt}}(B_{\text{alt}}, d)$, we have $p \neq d$, where d is the center of the ellipsoid. Thus, $y = Q^T(p-d) \neq 0$. Equivalently, we can say that the vector y has always a non–zero component, $y_i > 0$. Therefore, from equation (C.9), we can conclude that $\dot{\phi}(\mu) < 0$, for any $\mu \geq 0$. This means that the function ϕ is strictly decreasing for $\mu \geq 0$. Concerning the second order derivative, from the previous observation, we can conclude that $\ddot{\phi}(\mu) > 0$, for all $\mu \geq 0$, that is, ϕ is strictly convex over this domain.

The previous properties of the function ϕ allow us to use the *Newton–Raphson* method [21] for solving non–linear systems of equations.

Algorithm 9 (Newton–Raphson Method)**.** *Let $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ be a continuously differentiable function. Then, the Newton–Raphson method consists on:*

- *Choose $t_0 \in \mathbb{R}$;*
- *Iterate on k:*
  - *If $\dot{\phi}(t_k) = 0$, choose another starting point t_0, or report* failure*;*
  - *Otherwise, $t_{k+1} = t_k - \frac{\phi(t_k)}{\dot{\phi}(t_k)}$.*

The following lemma ensures that the *Newton–Raphson* method applied to the function ϕ, defined in (C.6), converges for any positive starting point t_0.

Lemma 4. *Let $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ be a twice continuously differentiable, strictly decreasing and strictly convex function. Assume that there exists $t^\star$ such that $\phi(t^\star) = 0$. Also assume that, for any $k \geq 0$, we have $t_k \neq t^\star$. Then, the Newton–Raphson method converges to $t^\star$.*

Proof. In the first place, let's see that the condition $\phi(t_k) > 0$, for any k, implies that $t_{k+1} > t_k$ and $\phi(t_{k+1}) > 0$. The first inequality holds because ϕ is strictly decreasing:

$$t_{k+1} = t_k - \frac{\overbrace{\phi(t_k)}^{>0}}{\underbrace{\dot{\phi}(t_k)}_{<0}} \Longrightarrow t_{k+1} > t_k.$$

By using the second order Taylor expansion of the function ϕ, and due to its strict convexity, we can confirm the second inequality:

$$\phi(t_{k+1}) = \underbrace{\phi(t_k) + \dot{\phi}(t_k)(t_{k+1} - t_k)}_{=0} + \frac{1}{2}\underbrace{(t_{k+1} - t_k)^2}_{\geq 0}\underbrace{\ddot{\phi}(t)}_{>0} > 0,$$

where $t \in [t_k, t_{k+1}]$. Furthermore, we can conclude that $\phi(t_{k+1}) > 0$ even if $\phi(t_k) < 0$. So,

$$\phi(t_k) > 0, \quad \text{for any } k \geq 1. \tag{C.11}$$

By the assumption of the lemma, $t^\star$ exists ($\phi(t^\star) = 0$). By (C.11), together with the fact that ϕ is strictly decreasing, we conclude that $t_k \leq t^\star$, for any $k \geq 1$. We also have that the sequence $\{t_k\}_{k=1}^{+\infty}$ is strictly increasing. Thus, the sequence $\{t_k\}_{k=1}^{+\infty}$ has a limit point, which we will designate by $\bar{t}$.

Let's now prove that $\phi(\bar{t}) = 0$, and therefore $\bar{t} = t^\star$, due to the injectivity of ϕ. Making the passage to the limit in

$$t_{k+1} = t_k - \frac{\phi(t_k)}{\dot{\phi}(t_k)},$$

we get

$$\bar{t} = \bar{t} - \frac{\phi(\bar{t})}{\dot{\phi}(\bar{t})},$$

which is equivalent to $\phi(\bar{t}) = 0$, since $\dot{\phi}(\bar{t}) < 0$ (strictly decreasing). □

Note that, owing to the fact that $\dot{\phi}(\mu) < 0$, for any $\mu \geq 0$, the Newton–Raphson method never reports failure. Algorithm 10 gathers all the steps that we have seen.

Algorithm 10 (Point Projection On A Ellipsoid)**.**

- *Predefined Parameters/Initialization:*
  - *Matrix B_{alt} and vector d that define the ellipsoid by*

$$\mathcal{E}_{alt}(B_{alt}, d) = \left\{x \,:\, (x-d)^T B_{alt}^{-1} (x-d) \leq 1\right\};$$

  - *The eigenvalue decomposition of B_{alt}^{-1}: $B_{alt}^{-1} = Q\Lambda Q^T$;*
  - *Tolerance ϵ for the stopping criterion of the Newton–Raphson method.*
- *Procedure:*
  - *Receive p, the point we want to project on the ellipsoid;*
  - *If $(p-d)^T B_{alt}^{-1} (p-d) \leq 1$,* return $x^\star = p$.
  - *Else*
    1. *Calculate $y = Q^T (p-d)$;*
    2. *Choose $\mu_0 \geq 0$; and set $k = 0$;*
    3. *Find the optimal dual variable $\mu^\star$ using Newton–Raphson method (iterate on k):*
       * *Evaluate*

$$\phi(\mu_k) = \sum_{i=1}^{n} \frac{\lambda_i}{(1+\mu_k\lambda_i)^2} y_i^2 - 1.$$

       * *If $|\phi(\mu_k)| < \epsilon$, break cycle;*
       * *Else, evaluate*

$$\dot{\phi}(\mu) = -2\sum_{i=1}^{n} \frac{\lambda_i^2}{(1+\mu\lambda_i)^3} y_i^2;$$

       and set

$$\mu_{k+1} = \mu_k - \frac{\phi(\mu_k)}{\dot{\phi}(\mu_k)}.$$

    4. *Set $\mu^\star = \mu_k$ and return*

$$x^\star = d + \left(I + \mu^\star B_{alt}^{-1}\right)^{-1} (p-d).$$

A few things must be said about algorithm 10: firstly, as it is well–known, Newton–Raphson method converges very fast, and so does algorithm 10. In a couple of iterations we get a very accurate solution.

Another aspect of this algorithm is that the evaluation of the functions ϕ and $\dot{\phi}$ can be made in parallel, if we have $P < n$ processors. The sum of the P numbers could be carried out in a central processor, or in a circular message–passing architecture.

Even though, without parallelization, the algorithm has a very low computational cost, mostly because of the fast speed of convergence of the Newton–Raphson method.

C.2 Strictly Convex Quadratic Program Over An Ellipsoid

In this section, we focus on solving a generic quadratic program over an ellipsoid. However, we consider that the matrix involved in the quadratic term is positive definite. So, the problem that we want to solve is

$$\min_{y \in \mathcal{E}_{\text{alt}}(B_{\text{alt}}, d)} y^T M y + v^T y, \tag{C.12}$$

where M is a square matrix of size n, with $M \succ 0$ and $M^T = M$; $\mathcal{E}_{\text{alt}}(B_{\text{alt}}, d)$ is a generic ellipsoid in $\mathbb{R}^n$ centered at d and with $B_{\text{alt}} \succ 0$. The variable is $y \in \mathbb{R}^n$.

We can transform problem (C.12) into a problem of the kind of (C.3) by a suitable change of variables. Namely, we introduce the variable x that satisfies $x = M^{1/2}y$. Note that this change of variables is well–defined as the matrix M is positive definite (hence also the matrix $M^{1/2}$).

Concerning the cost function of (C.12), it becomes

$$\begin{aligned} y^T M y + v^T y &= (\underbrace{(M^{1/2})^T}_{=M^{1/2}} y)^T (M^{1/2} y) + v^T y \\ &= x^T x + v^T (M^{-1/2}) x \\ &= x^T x + v^T (M^{-1/2}) x + \frac{1}{4}\|M^{-1/2} v\|^2 - \frac{1}{4}\|M^{-1/2} v\|^2 \\ &= \|x - (-\frac{1}{2} M^{-1/2} v)\|^2 - \frac{1}{4}\|M^{-1/2} v\|^2. \end{aligned}$$

The constraining equation of (C.12), on the other hand, becomes

$$\begin{aligned} (y-d)^T B_{\text{alt}}^{-1} (y-d) \le 1 &\iff (M^{-1/2}x - d)^T B_{\text{alt}}^{-1} (M^{-1/2}x - d) \le 1 \\ &\iff \left[M^{-1/2}(x - M^{1/2}d)\right]^T B_{\text{alt}}^{-1} \left[M^{-1/2}(x - M^{1/2}d)\right] \le 1, \end{aligned}$$

and as $(M^{-1/2})^T = M^{-1/2}$,

$$\begin{aligned} &\iff (x - M^{1/2}d)^T M^{-1/2} B_{\text{alt}}^{-1} M^{-1/2} (x - M^{1/2}d) \le 1 \\ &\iff x \in \mathcal{E}_{\text{alt}}\left(M^{1/2} B_{\text{alt}} M^{1/2}, M^{1/2} d\right). \end{aligned}$$

Therefore, to solve problem (C.12), we first find $x^\star$ by solving problem

$$\min_{x \in \mathcal{E}_{\text{alt}}\left(M^{1/2} B_{\text{alt}} M^{1/2}, M^{1/2} d\right)} \|x - (-\frac{1}{2} M^{-1/2} v)\|^2,$$

using the algorithm 10 in section C.1; and then we make $y^\star = M^{-1/2} x^\star$.