# Distributed convex optimization using ADMM


Dimitris Christodouleas

**Thesis Committee**
Professor Athanasios Liavas
Professor Minos Garofalakis
Professor Vassilis Digalakis

Chania, December 2014

# Abstract

ADMM (Alternating Direction Method of Multipliers) is a technique for the solution of optimization problems. It was developed and initially studied during the 1970's, but recently has attracted significant interest, mainly because it can be used to solve problems that handle large datasets in a distributed manner.

In this thesis, we describe the ADMM algorithm and some applications of it in machine learning and signal processing problems. We also give a distributed implementation of ADMM for a big least squares problem using MPI (Message Passing Interface).

# List of Figures

# Contents

# Chapter 1

# Introduction

Many problems in engineering, especially in the fields of machine learning and signal processing can be posed as convex optimization problems. These problems often deal with datasets that are extremely large, so it would be useful to develop algorithms that solve these problems in a distributed manner. In this thesis, we describe the ADMM algorithm [1], which is the basic tool we used to solve these problems in both serial and distributed manner.

In Chapter 2, we describe the ADMM algorithm, which derived in the 1970's as a combination of dual decomposition method and the method of multipliers.

In Chapter 3, we refer to some of the basic patterns we used to derive the algorithms for solving the problems with ADMM.

In Chapter 4, we develop an algorithm for solving quadratic problems.

In Chapter 5, we use ADMM to solve some machine learning and signal processing problems.These problems involve linear regression (Least squares, Lasso, Group Lasso), classification (logistic regression), compressed sensing and denoising (total variation denoising, trend filtering).

In Chapter 6, consensus and sharing techniques are introduced and are used to develop distributed algorithms for the problems described in the previous chapter.

In Chapter 7, we describe an implementation of a distributed ADMM algorithm for a big least squares problem on the grid computing system of Technical University of Crete, using MPI (Message Passing Interface) framework.

The proofs for the derivation of algorithms in chapter 5 can be found in part A of the appendix. In part B of the appendix, we describe how the basic functions of MPI work and we include the code for the distributed implementation.

# Chapter 2

# ADMM Description

## 2.1 Dual Ascent and Dual Decomposition

### 2.1.1 Dual Ascent Algorithm

Let us consider the following convex optimization problem

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{s.t.} \quad\ Ax = b, \end{aligned} \tag{2.1}$$

with $x \in \Re^n$, $A \in \Re^{m \times n}$ and $f : \Re^n \to \Re$ a convex function. Lagrangian for problem (2.1) is

$$L(x, y) = f(x) + y^T(Ax - b). \tag{2.2}$$

and the dual function is

$$g(y) = \inf_x L(x, y) = -f^*\big(-A^T y\big) - b^T y. \tag{2.3}$$

The dual problem is defined as

$$\text{maximize} \ \ g(y), \ \ y \in \Re^m. \tag{2.4}$$

Assuming strong duality holds, then optimal values of primal and dual problems are the same. We can recover a primal optimal point $x^*$ from a dual optimal $y^*$ as

$$x^* = \operatorname*{argmin}_x L(x, y^*).$$

In dual ascent we try to solve dual problem, using a gradient ascent method. $\nabla g(y)$ denotes the direction of movement with the largest increase of function $g(y)$, so we try to maximize $g(y)$ moving towards that direction. If the gradient exists

$$\nabla_y g(y) = \nabla_y\big(\inf_x L(x, y)\big) = \nabla_y\big(L(x^*, y)\big) = \nabla_y f(x^*) + \nabla_y y^T(Ax - b) = Ax - b.$$

The dual ascent algorithm consists of the following steps

$$\begin{aligned} x^{k+1} &= \operatorname*{argmin}_x L\big(x, y^k\big) \\ y^{k+1} &= y^k + a^k\big(Ax^{k+1} - b\big), \end{aligned} \tag{2.5}$$

where $a^k$ is a scalar stepsize and $k$ is the iteration counter. The first step of (2.5) is a $x$-minimization step and the second is a dual variable update. The dual ascent generally converges for a well-chosen $a^k$ and under strict assumptions. We will see later how we can make this method more robust.

### 2.1.2 Dual Decomposition

The major benefit of dual ascent method is that it can lead to a decentralized implementation at some cases. Suppose cost function is separable

$$f(x) = \sum_{i=1}^{N} f_i(x_i),$$

where $x$ has been split into subvectors, $x = [x_1, ..., x_N]$, where variables $x_i \in \Re^{n_i}$. Matrix A can be also decomposed as

$$A = [A_1 \cdots A_N].$$

So Lagrangian can be written as

$$L(x, y) = \sum_{i=1}^{N} L_i(x_i, y) = \sum_{i=1}^{N} \left( f_i(x_i) + y^T A_i x_i - \frac{1}{N} y^T b \right).$$

This shows that Lagrangian function is also separable in $x$. This means that the $x$-minimization step of (2.5) can be split into $N$ subproblems that can be solved in parallel. This leads to the equations below

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}}\, L_i\big(x_i, y^k\big) \\ y^{k+1} &= y^k + a^k\big(Ax^{k+1} - b\big) \end{aligned} \tag{2.6}$$

So the dual-variable update step is the same as in dual ascent, but $x$ minimization problem splits into $N$ subproblems that can be solved independently. The algorithm above is called dual decomposition. So in the first step each processor solves 1 of the $N$ subproblems and calculates the corresponding subvector $x_i^{k+1}$ of $x^{k+1}$. Then each processor broadcasts each subvector $x_1^{k+1}$ to every other node. So every processor gathers the vector $x^{k+1}$ and uses it to calculate $y^{k+1}$ using the second step of (2.6). A different approach would be to assign to a single *master* processor to compute $y$ at each step. So the master processor gathers local $x_i$ from the other nodes and uses them to compute $y$ and then broadcasts its value to every other node, in order to compute their local $x_i$s.

## 2.2 Method of Multipliers and ADMM

### 2.2.1 Method of Multipliers

In this subsection we will describe a method that was developed to robustify dual ascent. Let us start by defining the Augmented Lagrangian function

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + \frac{\rho}{2}\|Ax - b\|_2^2, \tag{2.7}$$

where $\rho$ is called the penalty parameter. $L_\rho$ is the Lagrangian function of the following problem

$$\begin{aligned} &\text{minimize} \quad f(x) + \frac{\rho}{2}\|Ax - b\|_2^2 \\ &\text{s.t.} \quad\quad Ax = b, \end{aligned} \tag{2.8}$$

This problem is equivalent to the initial problem described in (2.1), since the quadratic penalty term is 0, $\forall x$ that is in the feasible set of the problem. Applying dual ascent to the modified problem we get

$$\begin{aligned} x^{k+1} &= \underset{x}{\operatorname{argmin}}\, L_\rho\big(x, y^k\big) \\ y^{k+1} &= y^k + \rho\big(Ax^{k+1} - b\big). \end{aligned} \tag{2.9}$$

This is called the *method of multipliers*. We use the penalty parameter $\rho$ as scalar stepsize for dual variable update. By adding the quadratic penalty term the algorithm converges under far more relaxed conditions, compared to these of dual ascent. The main drawback of this method is that by introducing the quadratic penalty term we destroy the splitting of Lagrangian function and therefore we can't apply decomposition. In next subsection we will see how we can overcome this barrier, using ADMM.

### 2.2.2 ADMM

ADMM (Alternating Direction of Multipliers) is an algorithm that is intended to blend the decomposability of dual ascent with the superior convergence properties of the method of multipliers. This algorithm solves problems in the form

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c, \end{array} \tag{2.10}$$

with variables $x \in \Re^n$ and $z \in \Re^m$, where $A \in \Re^{p\times n}$, $B \in \Re^{p\times m}$, and $c \in \Re^p$. The basic difference between the above problem and the initial problem (2.1) is that variable called $x$ there is split into 2 parts ($x$ and $z$), with the objective function seperable across this splitting. We form the augmented Lagrangian function of this problem as

$$L_\rho(x, y, z) = f(x) + g(z) + y^T(Ax + Bz - c) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2.$$

ADMM consists of the iterations below

$$\begin{aligned} x^{k+1} &= \underset{x}{\operatorname{argmin}}\, L_\rho\big(x, z^k, y^k\big) \\ z^{k+1} &= \underset{z}{\operatorname{argmin}}\, L_\rho\big(x^{k+1}, z, y^k\big) \\ y^{k+1} &= y^k + \rho\big(Ax^{k+1} + Bz^{k+1} - c\big) \end{aligned} \tag{2.11}$$

ADMM is pretty similar in fashion to dual ascent. It consists of a $x$-minimization step, a $z$-minimization step and a dual variable update step. Step-size of dual variable update step is equal to augmented Lagrangian parameter $\rho$. Minimizing augmented Lagrangian function jointly over variables $x$ and $z$ and ADMM becomes the method of multipliers. Separating minimization over $x$ and $z$ into 2 steps is what allows decomposition, when $f$ or $g$ are separable.

### 2.2.3 Scaled Form of ADMM

We will now describe an alternating form of ADMM, which is often more convenient to use. Define $r = Ax + Bz - c$ and $u = \dfrac{1}{\rho}y$. Augmented Lagrangian now becomes

$$L_\rho(x, y, z) = f(x) + g(z) + \frac{\rho}{2}\|r\|_2^2 + y^T r = f(x) + g(z) + \frac{\rho}{2}\|r\|_2^2 + \rho u^T r. \tag{1}$$

$$\frac{\rho}{2}\|u + r\|_2^2 = \frac{\rho}{2}(u + r)^T(u + r) = \frac{\rho}{2}\big(\|u\|_2^2 + 2u^T r + \|r\|_2^2\big) = \frac{\rho}{2}\|u\|_2^2 + \rho u^T r + \frac{\rho}{2}\|r\|_2^2.$$

$$\text{So} \quad \frac{\rho}{2}\|u + r\|_2^2 - \frac{\rho}{2}\|u\|_2^2 = \rho u^T r + \frac{\rho}{2}\|r\|_2^2. \tag{2}$$

Using (1) and (2), we have

$$L_\rho(x, y, z) = f(x) + g(z) + \frac{\rho}{2}\|u + r\|_2^2 - \frac{\rho}{2}\|u\|_2^2.$$

So ADMM equations become

$$\begin{aligned} x^{k+1} &= \operatorname*{argmin}_x \left( f(x) + \frac{\rho}{2} \|Ax + Bz^k - c + u^k\|_2^2 \right) \\ z^{k+1} &= \operatorname*{argmin}_z \left( g(z) + \frac{\rho}{2} \|Ax^{k+1} + Bz - c + u^k\|_2^2 \right) \\ u^{k+1} &= u^k + Ax^{k+1} + Bz^{k+1} - c. \end{aligned} \tag{2.12}$$

This is called the scaled form of ADMM, since it is expressed in terms of a scaled version of dual variable. It is usually preferred over the unscaled form of ADMM, since it produces shorter formulas.

# Chapter 3

# General Patterns in ADMM

Structure in $f$, $g$, $A$ and $B$ can be exploited to carry out $x$ and $z$- updates more efficiently. The following analysis will be written for $x$- update. We express $x$- update as

$$x^+ = \underset{x}{\operatorname{argmin}} \left( f(x) + \frac{\rho}{2}\|Ax - v\|_2^2 \right), \tag{3.1}$$

where $v = -Bz + c - u$ is a known constant vector for purposes of $x$- update.

## 3.1 Proximity Operator

Consider the case where $A = I$ which appears very frequently in practice. Then (3.1) is written

$$x^+ = \underset{x}{\operatorname{argmin}} \left( f(x) + \frac{\rho}{2}\|x - v\|_2^2 \right). \tag{3.2}$$

As a function of $v$, the right-hand side is denoted as $prox_{f,\rho}(v)$ and is called proximity operator of f with penalty $\rho$. If $f$ is simple enough, the $x$- update can be evaluated analytically. If $f$ is the indicator function of a closed non-empty convex set $C$, then the $x$- update is

$$x^+ = \underset{x}{\operatorname{argmin}} \left( f(x) + \frac{\rho}{2}\|x - v\|_2^2 \right) = \Pi_C(v),$$

where $\Pi_C$ is the projection operator onto set $C$.

### 3.1.1 Soft thresholding

Let $f(x) = \lambda\|x\|_1$ and $A = I$. The minimization step of $x_i$-update becomes

$$x_i^* = \underset{x_i}{\operatorname{argmin}} \left( \lambda|x_i| + \frac{\rho}{2}(x_i - v_i)^2 \right).$$

The term $\|x\|_1$ is not differentiable at 0. However a closed-form solution to the problem can be found, using subgradients. Solution is

$$x_i^* = S_{\frac{\lambda}{\rho}}(v_i),$$

where the thresholding operator is defined as

$$S_k(a) = \begin{cases} a - k & a > k \\ 0 & |a| \le k \\ a + k & a < -k. \end{cases} \tag{3.3}$$

Soft thresholding operator is the proximity operator for $l_1$-norm.

### 3.1.2 Block soft thresholding

Let $f(x) = \lambda\|x\|_2$ and $A = I$. The minimization step of $x_i$-update becomes

$$x^* = \underset{x}{\operatorname{argmin}} \left( \lambda||x||_2 + \frac{\rho}{2}||x - v||_2^2 \right).$$

A closed-form solution can be found for this problem, and is called *block soft thresholding operator*.

$$x^* = (1 - (\lambda/\rho)/||v||_2)_+ v. \tag{3.4}$$

### 3.1.3 Singular value thresholding

Consider the singular value decomposition (SVD) of a matrix $X$ of rank $r$.

$$X = U\Sigma V^T, \quad \Sigma = diag(\{\sigma_i\}_{1 \le i \le r})$$

For each $\tau > 0$, we introduce the *singular value thresholding operator* defined as follows

$$D_\tau(X) = UD_\tau(\Sigma)V^T, \quad D_\tau(\Sigma) = diag(\{\sigma_i - \tau\}_+).$$

This operator simply applies a soft thresholding rule to the singular values of $X$, effectively shrinking them towards zero. For each $\tau > 0$ and $Y \in \Re^{m \times n}$, singular value thresholding operator obeys.

$$D_\tau(Y) = \underset{X}{\operatorname{argmin}} \left( \frac{1}{2}||X - Y||_F^2 + \tau||X||_* \right), \tag{3.5}$$

where $||\cdot||_*$ is the *nuclear norm*, and $||\cdot||_F$ is the *Frobenius norm*

## 3.2 Matrix inversion Lemma

A very useful matrix identity that we will use widely in this thesis is the matrix inversion lemma (or Sherman-Morrison-Woodbury formula),

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}. \tag{3.6}$$

A proof of this expression can be found in the appendix.

# Chapter 4

# ADMM for solving quadratic and linear programs

## 4.1 Using ADMM to solve QPs and LPs

The standard form of quadratic program is

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + q^T x \\ \text{subject to} & Ax = b,\ x \geq 0, \end{array} \tag{4.1}$$

where $x \in \Re^n$ and $P$ is positive definite. If $P = 0$, then the problem becomes a standard form linear program. ADMM form for this problem is

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array} \tag{4.2}$$

with $f(x) = \dfrac{1}{2}x^T Px + q^T x$ and $g(z)$ is the indicator function of the non-negative orthant $\Re^n_+$. The scaled form ADMM for this problem is

$$\begin{array}{l} x^{k+1} = \underset{x}{\text{argmin}} \left( f(x) + \dfrac{\rho}{2}\|x - z^k + u^k\|_2^2 \right) \\ z^{k+1} = \left(x^{k+1} + u^k\right)_+ \\ u^{k+1} = u^k + x^{k+1} - z^{k+1}. \end{array} \tag{4.3}$$

The $x$-minimization step is a problem with optimality conditions

$$\begin{bmatrix} P + \rho I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} -q + \rho(z - v) \\ b \end{bmatrix}.$$

An efficient computation of $x^{k+1}$ can be achieved by calculating the factorization of $\rho I + P$ at first iteration and then using it on following iterations.

# Chapter 5

# Solving $l_1$ problems with ADMM

ADMM explicitly targets problems that split into two parts, $f$ and $g$. So it is a natural fit for applications in signal processing and machine learning, where a lot of problems involve minimizing a cost function together with a regularization term. In this section, we will see a variety of problems that involve $l_1$ norms, which are very important across statistics, signal processing, and machine learning.

## 5.1 Least absolute deviations

In data fitting applications, when the data contains large outliers, instead of least squares fitting is used a technique that is called *Least Absolute Deviations*, which provides a more robust fit. We want to minimize $\|Ax - b\|_1$ instead of $\|Ax - b\|_2^2$. This problem is written in ADMM form as

$$\begin{aligned} &\text{minimize} \quad \|z\|_1 \\ &\text{subject to} \quad Ax - z = b, \end{aligned} \tag{5.1}$$

so $f = 0$ and $g = \|\cdot\|_1$. Assuming $A^T A$ is invertible, the scaled form ADMM is

$$\begin{aligned} x^{k+1} &= \left(A^T A\right)^{-1} A^T \left(b + z^k - u^k\right) \\ z^{k+1} &= S_{\frac{1}{\rho}}\left(Ax^{k+1} - b + u^k\right) \\ u^{k+1} &= u^k + Ax^{k+1} - z^{k+1} - b. \end{aligned} \tag{5.2}$$

## 5.2 Huber fitting

Another way of attaining a more robust fit in a data fitting application, is called *Huber function fitting*,

$$\text{minimize} \quad g^{hub}(Ax - b), \tag{5.3}$$

where *Huber penalty function* $g^{hub}$ is given by

$$g^{hub}(a) = \begin{cases} a^2/2 & |a| < 1 \\ |a| - (1/2) & |a| \geq 1, \end{cases} \tag{5.4}$$

where $a \in \Re$. Huber penalty function can be extended to vector arguments as the sum of the Huber function of the components. So Huber function is quadratic for small arguments and linear for bigger ones. So it is more robust compared to ordinary least squares (that assign quadratic penalty to big residuals), by giving much less weight to big residuals. Huber fitting can be put in ADMM form as

$$\begin{aligned} &\text{minimize} \quad f(x) + g(z) \\ &\text{subject to} \quad Ax - z = b, \end{aligned} \tag{5.5}$$

where $f(x) = 0$, and $g(z) = \sum_{i=1}^{m} g^{hub}(z_i)$. $X$ and $u$ updates of ADMM for this problem will be the same as in the LAD problem. Let $w = Ax + u - b$, then $z$ update for this problem will be

$$z_i^{k+1} = \begin{cases} w_i - \dfrac{1}{\rho}, & \text{if } w_i > \frac{(1+\rho)}{\rho} \\ \rho \dfrac{w_i}{(1+\rho)}, & \text{if } |w_i| \leq \frac{(1+\rho)}{\rho} \\ w_i + \dfrac{1}{\rho}, & \text{if } w_i < -\frac{(1+\rho)}{\rho}. \end{cases} \tag{5.6}$$

### 5.2.1 Numerical Example. Fitting data with outliers

Now we consider the case of fitting a model to data that contains outliers. We compare the performance of Ordinary Least Squares (OLS) estimator with that of Huber Fitting(HF) and Least Absolute Deviations estimator (LAD). This example will show how important is to use robust fitting in order to handle cases like this one. Let $y = ax + b$ the equation of a line in $\Re^2$. Now we generate 20 samples that lie on this line and we add white gaussian noise to them. In 2 of them we add relatively large noise(outliers). Then we will fit the noisy data to a line, using OLS, HF, and LAD estimator. The two figures below show the results we get from this experiment.

Figure 5.1: Least absolute deviations estimator versus least squares estimator in model fitting with outliers problem.

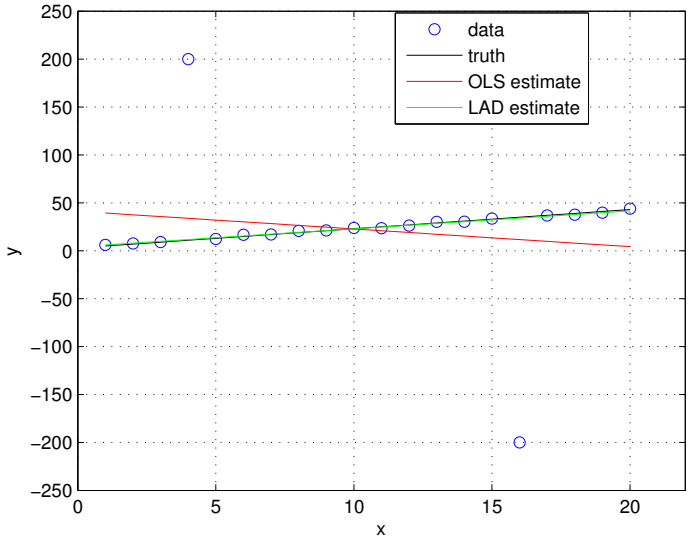
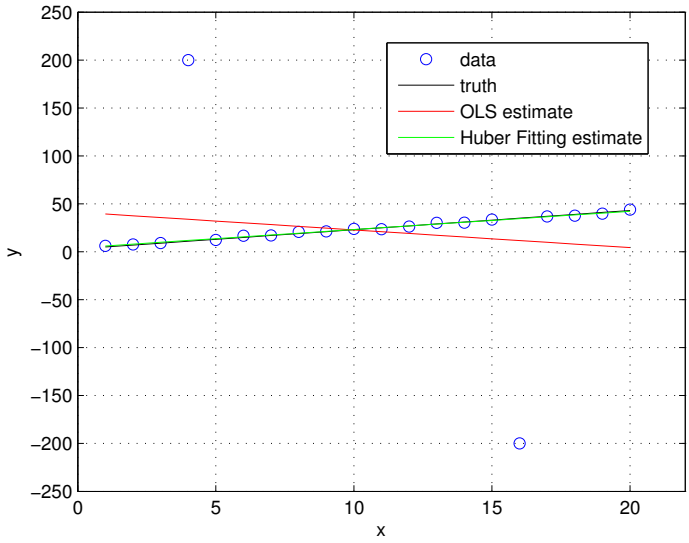


Figure 5.2: Huber Fitting estimator versus least squares estimator in model fitting with outliers problem.




In figure 5.1, we see that the line fitted by least squares estimator is rotated away from the main locus of points, but line from LAD estimator is close to the true line model. In figure 5.2, we see that the line fitted by Huber Fitting estimator is also close to the true line model.

## 5.3 Basis Pursuit

A very important problem that arises in signal processing, and specifically in applications like compressed sensing [2], is called *Basis Pursuit*. It is the optimization problem of the form

$$\begin{aligned} &\text{minimize} \quad \|x\|_1 \\ &\text{subject to} \quad Ax = b, \end{aligned} \tag{5.7}$$

where $x \in \Re^n$, data $A \in \Re^{m \times n}$, $b \in \Re^m$, with $m < n$. It is the most basic technique that is used to find a sparse solution of an underdetermined system of linear equations. Since it minimizes the $l_1$ norm of $x$, it usually finds solutions that are sparse. The ADMM form of this problem is

$$\begin{aligned} &\text{minimize} \quad f(x) + \|z\|_1 \\ &\text{subject to} \quad x - z = 0, \end{aligned} \tag{5.8}$$

where $f(x)$ is an indicator function of $\{x \in \Re^n | Ax = b\}$. ADMM algorithm is then

$$\begin{aligned} x^{k+1} &= \Pi\big(z^k - u^k\big) \\ z^{k+1} &= S_{\frac{1}{\rho}}\big(x^{k+1} + u^k\big) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1}, \end{aligned} \tag{5.9}$$

where $\Pi$ is the projection onto $\{x \in \Re^n | Ax = b\}$. Projection of the vector $(z - u)$ onto an affine set $Ax = b$ is given by

$$\Pi(z-u) = \big(I - A^T\big(AA^T\big)^{-1}A\big)(z-u) + A^T\big(AA^T\big)^{-1}b. \tag{5.10}$$

### 5.3.1 Numerical example. Compressed sensing using Basis Pursuit

Let $z \in \Re^n$ be an unknown signal. Suppose we now have $m$ linear measurements of signal $z$.

$$b_i = a_i^T z + v_i, \quad i = 1, \ldots, m,$$

where $v \in \Re^m$ is the noise, and $a_i \in \Re^n$ known signals. Standard reconstruction methods require at least $n$ samples. But if we have prior information about $z$, such as sparsity of the signal, we can reconstruct it using fewer than $n$ measurements. In this case, we can use Basis Pursuit as a heuristic method to solve this problem, finding a sparse solution to an under-determined system

$$\text{minimize } ||z||_1 \quad s.t.\ Az = b,$$

where $A \in \Re^{mxn}$ is the *compressed sensing matrix*. As an example, we consider a sparse signal reconstruction problem with a signal, $x \in \Re^{3000}$ which consists of 120 spikes with amplitude $\pm 1$. We then generate the compressed sensing matrix $A \in \Re^{1024 \times 3000}$. So we have

$$b = Ax + v,$$

where $v$ is drawn according to the Gaussian distribution $N(0, 0.01^2 I)$. We reconstructed the signal using Basis Pursuit, and Least Norm solution and we compare the results.

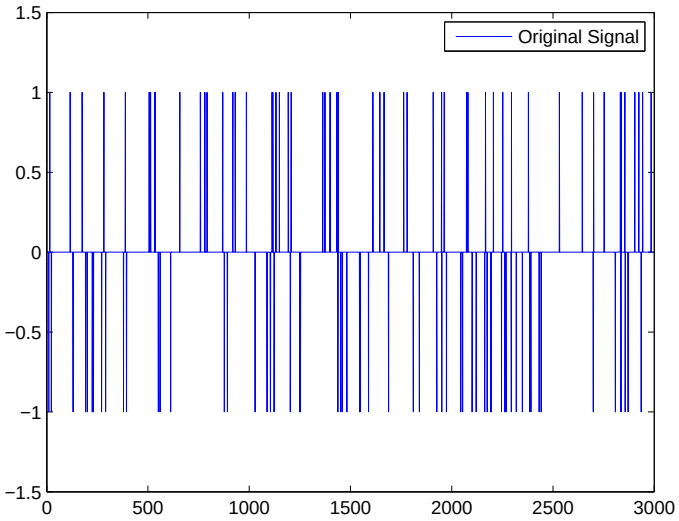



Figure 5.3: Original Signal

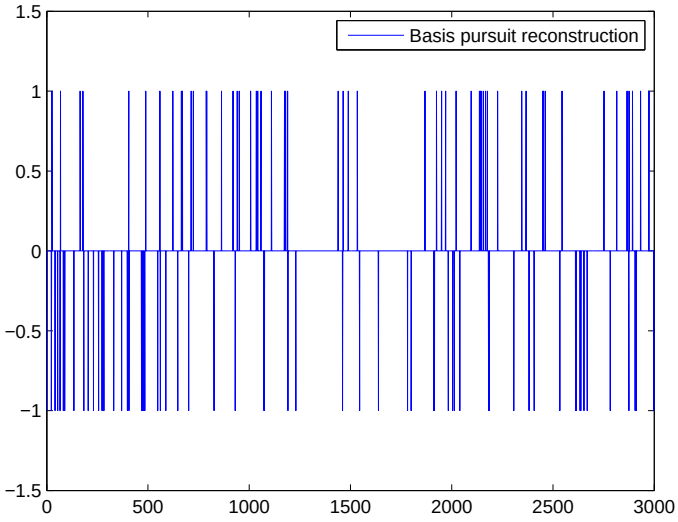



Figure 5.4: Reconstructed signal using Basis Pursuit

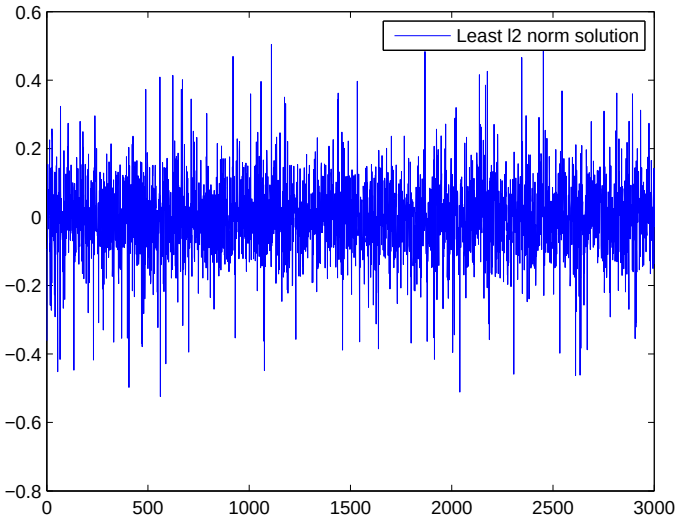



Figure 5.5: Reconstructed Signal using $l_2$ least norm solution

We clearly see that Basis pursuit (5.4) returns a sparse reconstructed signal that is pretty similar with the original. The $l_2$-least norm solution (5.5) on the other hand is not sparse and the spikes are far attenuated, and it is very difficult to detect the original signal. So we see that basis pursuit is very efficient for compressed sensing problems.

## 5.4 Lasso

Least squares is the basic tool for linear regression. In practice it is often used a technique called *regularization*, either because observation matrix can be ill-conditioned (i.e. very high correlation between features), or to impose constraints that come out from prior knowledge about parameter vector (i.e. sparsity). Regularization improves the conditioning of the problem. The most common regularizers used are the squared $l_2$ norm (*Tikhonov regularization*), and the $l_1$ norm. If the regularizer is $l_1$ norm, then the problem is called *Lasso Regression*.

$$\text{minimize} \quad \frac{1}{2}\|Ax-b\|_2^2 + \lambda\|x\|_1, \tag{5.11}$$

where $\lambda$ is a scalar regularization parameter. In ADMM form, the Lasso can be written as

$$\begin{aligned} &\text{minimize} \quad f(x)+g(z) \\ &\text{subject to} \quad x-z=0, \end{aligned} \tag{5.12}$$

where $f(x)=\frac{1}{2}\|Ax-b\|_2^2$ and $g(z)=\lambda\|z\|_1$. ADMM algorithm for Lasso is

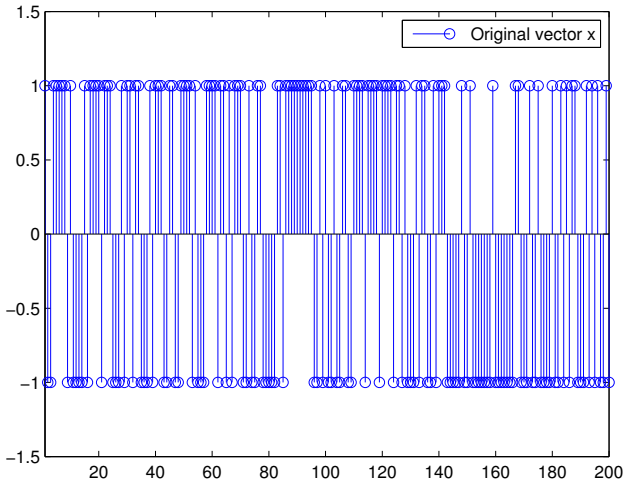
$$\begin{aligned} x^{k+1} &= \left(A^TA+\rho I\right)^{-1}\left(A^Tb+\rho(z^k-u^k)\right) \\ z^{k+1} &= S_{\frac{\lambda}{\rho}}\left(x^{k+1}+u^k\right) \\ u^{k+1} &= u^k + x^{k+1} - z^{k+1}. \end{aligned} \tag{5.13}$$

$A^TA+\rho I$ is always invertible since $\rho>0$, and by the addittion of the term $\rho I$ the problem is more well-conditioned than the unregularized least squares problem. Also, $||x||_1$ term leads parameter vector to be sparse, so the algorithm also does *feature selection*. We can compute at the first iteration a factorization of $A^TA+\rho I$ and use it in the following iterations.

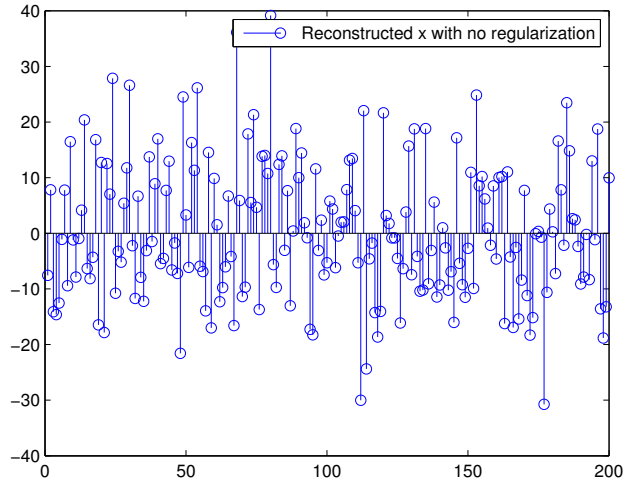
### 5.4.1 Numerical example. Tikhonov regularization for solving ill-conditioned linear systems

Let matrix $A \in \Re^{m\times n}$ a known matrix and $y \in \Re^m$ a known vector. We want to estimate a vector $x \in \Re^n$, where $y = Ax + v$, with $v$ white gaussian noise. Matrix $A$ is constructed to be *ill-conditioned*. We will reconstruct vector $x$ using ordinary least-squares, and then Tikhonov regularized Least Squares, with regularization parameter $\lambda = 10^{-3}$. In our simulations we will use noise with variance $10^{-2}$.

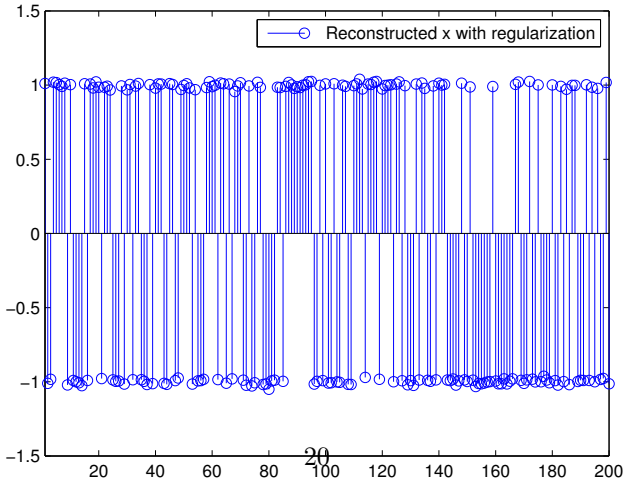
Figure 5.6: Ill-conditioned system solution




(a) Original Signal




(b) Reconstructed signal Ordinary Least squares

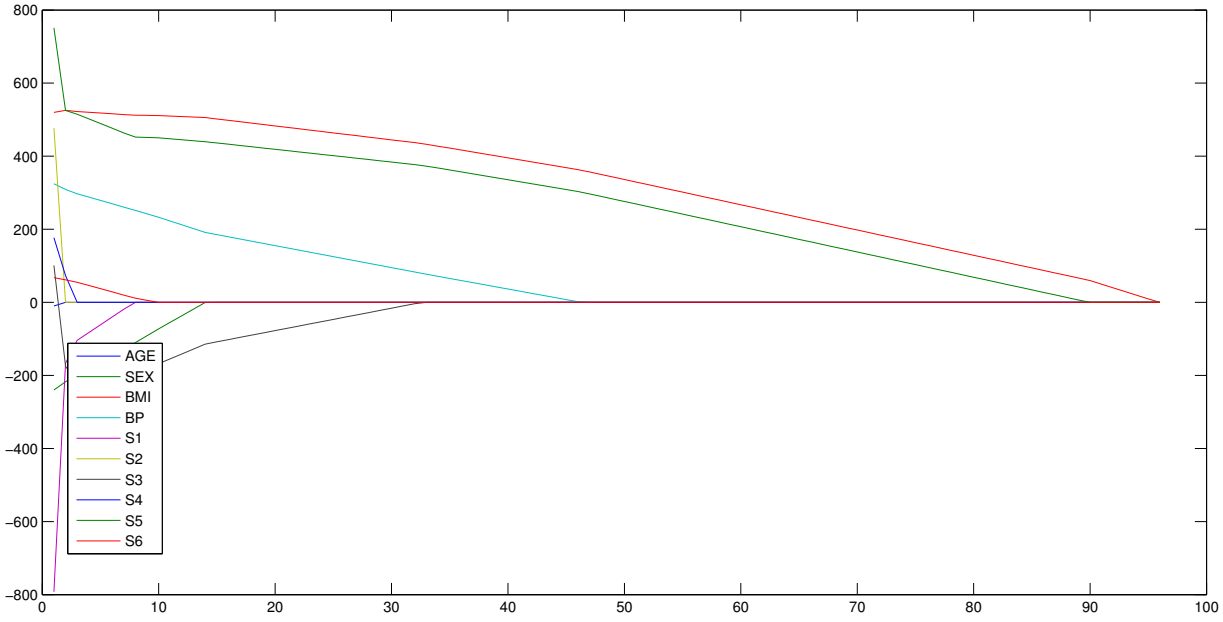


As we can see in the figure 5.6b , reconstructed $x$ with no regularization is nowhere near the original $x$, even if noise has very low density. On the other hand, in 5.6c we see that by adding the regularization term, and thus improving the conditioning of the problem, the reconstructed $x$ is pretty close to the original.

### 5.4.2 Numerical example. Feature selection using Lasso.

As we said above, LASSO returns a sparse feature vector. The bigger the value of $\lambda$ is, the more sparse feature vector will be. So we can use LASSO to select a subset of the features, in order to make the model simpler and more interpretable. In our experiment, we used the diabetes dataset (see [3] for more details). It describes a dependent variable as a linear function of 10 independent regressors. In the figure below we show the *regularization path* (as $\lambda$ increases the value of regressors goes towards zero) for this dataset.

Figure 5.7: Regularization path for diabetes dataset.




We see in figure 5.7 that, as $\lambda$ increases, regressors' values shrink towards zero. If we would like to select only 3 features to make this model simpler, we would choose (BMI, S5, BP).

## 5.5 Generalized Lasso

Lasso problem can be generalized to

$$\text{minimize} \quad \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|Fx\|_1, \tag{5.14}$$

where $F$ is an arbitary linear transformation. We will consider two special cases of this problem with applications in signal processing.

- Let $A = I$, and $F \in \Re^{(n-1)\times n}$ the first order difference matrix.

$$F_{ij} = \begin{cases} 1, & j = i+1, \\ -1, & j = i, \\ 0, & otherwise. \end{cases} \tag{5.15}$$

  This problem is called *total variation denoising*, and is used to remove noise from time series. Notice that term $||Fx||_1$ leads the first derivative of the signal to be sparse, so the reconstructed signal will generally be *piece-wise constant*.

- Let $A = I$, and $F \in \Re^{(n-2)\times n}$ the second order difference matrix.

$$F_{ij} = \begin{cases} 1, & j = i+1, \\ -2, & j = i, \\ 1, & j = i-1, \\ 0, & otherwise. \end{cases} \tag{5.16}$$

  This problem is called *trend filtering*, and is used to recover underlying trends from noisy time series. Notice that term $||Fx||_1$ leads the second derivative of the signal to be sparse, so the reconstructed signal will generally be *piece-wise linear*.

In ADMM form, problem (5.14) can be written as

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x - z = 0, \end{array} \tag{5.17}$$

So the corresponding scaled ADMM algorithm will be

$$\begin{aligned} x^{k+1} &= \left(I + \rho F^T F\right)^{-1}\left(b + \rho F^T (z^k - u^k)\right) \\ z^{k+1} &= S_{\frac{\lambda}{\rho}}\left(Fx^{k+1} + u^k\right) \\ u^{k+1} &= u^k + Fx^{k+1} - z^{k+1}. \end{aligned} \tag{5.18}$$

### 5.5.1 Numerical example. Total variation denoising

Let $x \in \Re^n$ a signal, where coefficients $x_i$ correspond to the value of some function of time, sampled at evenly spaced points. We assume that signal does not vary to rapidly, so $x_t \sim x_{t+1}$. The signal is corrupted by additive noise $v$, so we obtain the signal

$$x_{cor} = x + v.$$

We choose to model noise as a small and rapidly varying signal. Our goal now is to reconstruct original signal from the noisy samples. This problem is often called *de-noising*, or *smoothing*. This problem can be formulated as

$$\text{minimize } ||\hat{x} - x_{cor}||_2^2 + \phi(\hat{x}),$$

where $x$ is the variable and function $\phi$ is called *smoothing objective*. It is used to measure lack of smoothness of the signal. So we try to find a signal that is both close to $x_{cor}$ and also smooth. Now, let matrix $D \in \Re^{(n-1)\times n}$, be the first order difference matrix (5.15). The simplest de-noising method uses quadratic smoothing function

$$\phi_{quad}(\hat{x}) = \sum_{i=1}^{n-1} (\hat{x}_{i+1} - \hat{x}_i)^2 = ||D\hat{x}||_2^2.$$

This method smooths the signal putting penalty in the variations of signal. It puts big penalties on big variations and small penalties on small ones. So any big variation (i.e. edge) in the signal will be greatly attenuated too. The problem has closed form solution. Now let us consider the smoothing function

$$\phi_{tvd}(\hat{x}) = \sum_{i=1}^{n-1} |\hat{x}_{i+1} - \hat{x}_i| = ||D\hat{x}||_1.$$

This method performs smoothing, putting far less weight on big variations in signal, compared to quadratic smoother. So it can be used as an *edge-preserving* reconstruction method. Also it assigns bigger penalties on small variations pushing them to zero. So reconstructed signal $x$ will generally have sparse first order derivative, and it will be *piece-wise constant*. This technique is called *total variation de-noising*. In our experiment we created a piece-wise constant signal (with 2 amplitudes $\pm 2$ ) and we added noise $v$, which was drawn from distribution $N(0, 0.5^2 I)$. We used both Quadratic Smoothing, and Total Variation Denoising to smooth the time series.

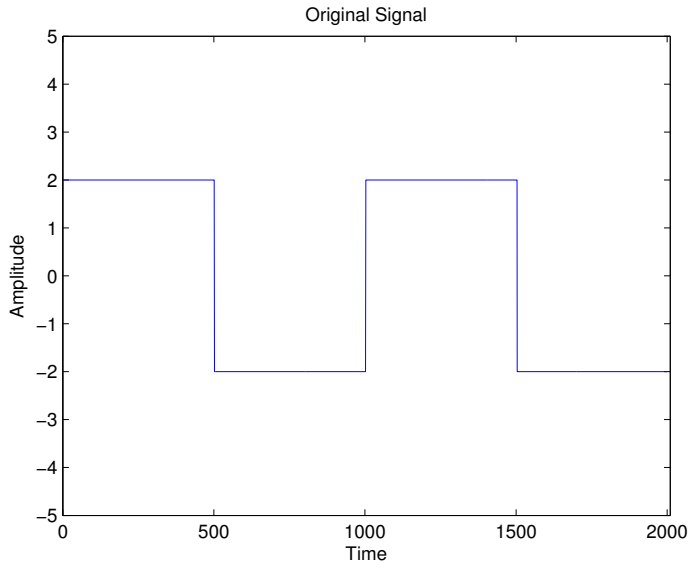



Figure 5.8: Original Signal

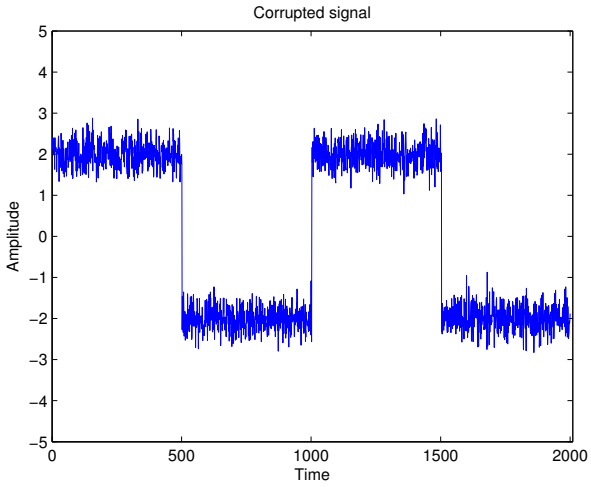



Figure 5.9: Signal corrupted with noise

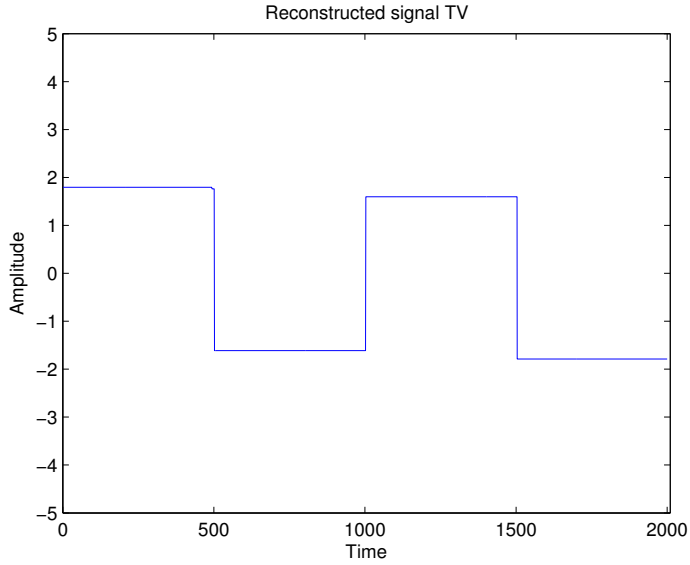



Figure 5.10: Reconstructed Signal using TVD

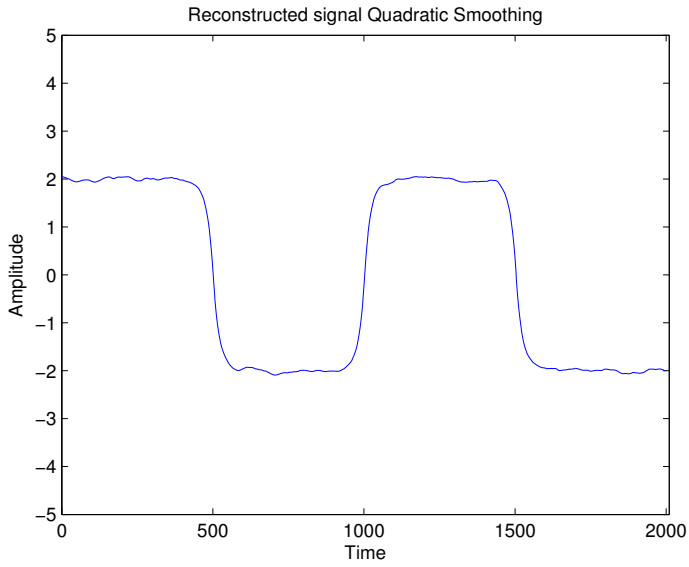



Figure 5.11: Reconstructed Signal using QS

We observe that quadratic smoothing 5.10 de-noises the signal pretty efficiently, but it also smooths out the edges of the signal. On the other hand, the edges of the signal are less attenuated with total variation denoising 5.11, and also the reconstructed signal is piece-wise constant

### 5.5.2 Numerical example. Trend Filtering

Now let us again consider the signal de-noising problem. Now we will try to smooth out the signal, putting penalties on the variations of signal's gradient (so we actually put penalty on the second order derivative of the signal). Now let matrix $D \in \Re^{(n-2)\times n}$ be the second order difference matrix (5.16). The simplest filter we can design for this purpose is called *Hodrick-Prescott filtering*, that is chosen to minimize the objective

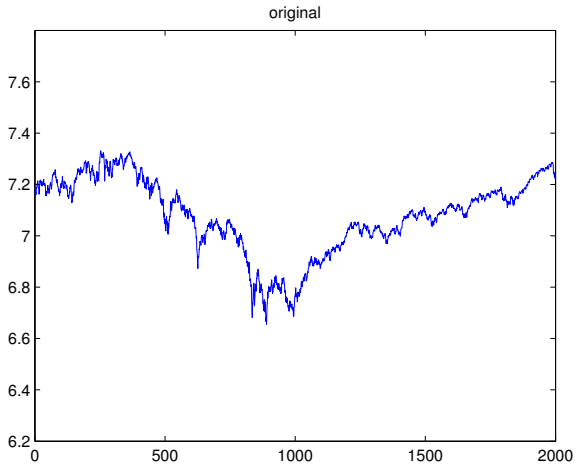
$$(1/2)||y - x||_2^2 + \lambda||Dx||_2^2.$$

This technique smooths signal putting very big penalties on big variations of signal's gradient. This problem has closed form solution. Now let us consider a technique called *trend filtering* that minimizes the objective

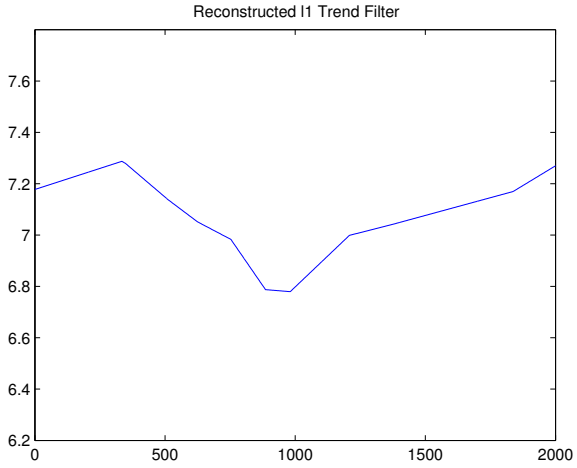
$$(1/2)||y - x||_2^2 + \lambda||Dx||_1.$$

This method puts much less weight compared to Hodrick-Prescott filter on big variations of signal's gradient. However, it puts more weight on small variations pushing them near zero. So the reconstructed signal will generally have sparse second order derivative, and it will be *piece-wise linear*. So $l_1$ trend filter is a technique well-suited to analyzing time-series with an underlying piece-wise linear trend. In our experiment we used S&P500 index for the period of March 25, 1999, to March 9, 2007 (see [4] for more details). We tried both Hodrick-Precott Filtering and $l_1$ Trend-Filtering, in order to find piece-wise linear trends in the time-series.

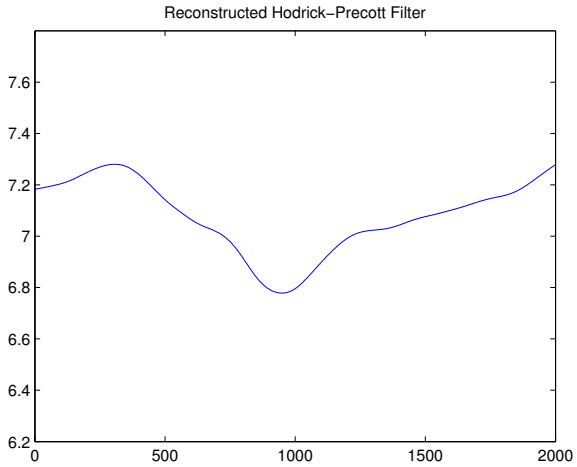
Figure 5.12: Trend Filtering




(a) Original Signal




(b) Reconstructed signal using $l_1$-Trend filtering




(c) Reconstructed Signal using Hodrick-Precott filtering

From figure 5.12b we observe that $l_1$ Trend Filter gives a piece-wise linear reconstruction of the signal. On the other hand, Hodrick-Prescott Filtering 5.12c smooths the signal, but reconstruction is not piece-wise linear.

## 5.6 Group Lasso

Now, let us consider the case when the $||x||_1$ regularizer is replaced by $\sum_{i=1}^{N} ||x_i||_2$, where $x = (x_1, \ldots, x_N)$, where $x_i \in \Re^{n_i}$. The regularizer here is seperable w.r.t. the partition $x_1, \ldots, x_N$, but not fully seperable. Group Lasso arises in applications (i.e. bioinformatics), where correlated features can be put into groups. Group lasso, instead of feature selection, performs group selection, shrinking all the features in a group towards 0.

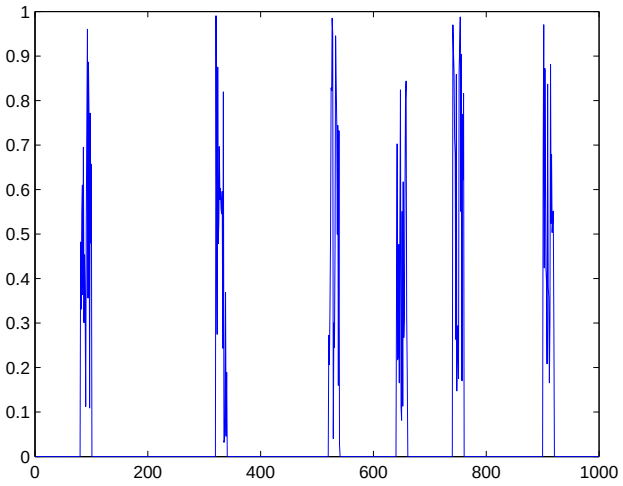
ADMM for this problem is the same as in Lasso, with the $z$-update replaced with block soft thresholding

$$z_i^{k+1} = \boldsymbol{S}_{\frac{\lambda}{\rho}}(x_i^{k+1} + u_i^k), \quad i = 1, \ldots, N. \tag{5.19}$$

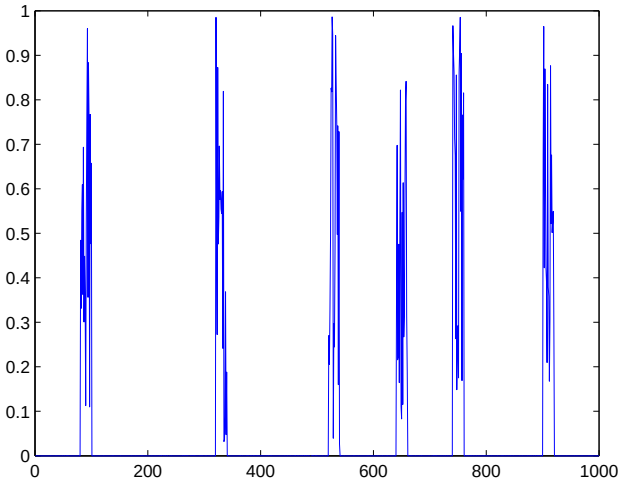
### 5.6.1 Numerical example. Group Lasso

For our experiment we have created a feature matrix of size $500 \times 1000$. The features are divided into 50 groups, and each group is chosen to be sparse with probability $p = 0.9$. The first figure shows the actual value of the features and the second one the value of the features after applying group lasso algorithm.

Figure 5.13: Group Lasso application




(a) Original feature vector




(b) Feature vector using Group Lasso algorithm

# Chapter 6

# Distributed ADMM and applications

In this chapter we will explain the problems of consensus and sharing, using ADMM and we 'll see expansions of them and applications to machine learning and signal processing.

## 6.1 Global variable Consensus

Let us first consider the case, where we have a single global variable with the objective split into $N$ parts

$$\text{minimize } f(x) = \sum_{i=1}^{N} f_i(x),$$

where $x \in \Re^n$ and $f_i : \Re^n \to \Re$. We refer to $f_i$ as the ith term in the objective. The goal is to solve this problem in such way that each element can be handled by its own processing element. This problem can be rewritten with local variables $x_i \in \Re^n$ and a common global variable $z$

$$\begin{aligned} &\text{minimize } \sum_{i=1}^{N} f_(x_i) \\ &\text{subject to } x_i - z = 0, \quad i = 1, \ldots, N. \end{aligned} \tag{6.1}$$

This is called as the *variable consensus problem*, since constraints are that all local variables should be equal. Consensus can be seen as a technique for splitting objectives $\sum_{i=1}^{N} f_i(x_i)$, which do not split, due to the variable being shared across terms, into separable objectives $\sum_{i=1}^{N} f_i(x_i)$, which split easily. Now we will show how to derive ADMM algorithm for this problem. At first, we form the Augmented Lagrangian for this problem.

$$L_\rho(x_1, \ldots, x_N, z, y) = \sum_{i=1}^{N} \left( f_i(x_i) + y_i^T(x_i - z) + (\rho/2)||x_i - z||_2^2 \right).$$

We can see that Lagrangian function is separable along each $x_i$. So ADMM update for $x_i$ will be

$$x_i^{k+1} = \underset{x_i}{\text{argmin}} \left( f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)||x_i - z^k||_2^2 \right).$$

Now we will find the solution for $z$-update.

$$z_{k+1} = \underset{z}{\text{argmin}} \left( -\sum_{i=1}^{N} y_i^{kT} z + (\rho/2) \sum_{i=1}^{N} ||x_i^{k+1} - z||_2^2 \right).$$

Using first optimality condition

$$-\sum_{i=1}^{N} \nabla_z y_i^{kT} z + (\rho/2) \sum_{i=1}^{N} \nabla_z ||x_i^{k+1} - z||_2^2 = 0$$

$$-\sum_{i=1}^{N} y_i^{kT} + \rho \sum_{i=1}^{N} (-x_i^{k+1} + z^{k+1}) = 0$$

Let us define $\bar{\alpha} = (1/N) \sum_{i=1}^{N} \alpha_i$. So the equation becomes

$$-N\bar{y}^k - \rho N \bar{x}^{k+1} + \rho N z^{k+1} = 0$$

$$z^{k+1} = (1/\rho)\bar{y}^k + \bar{x}^{k+1}. \tag{6.2}$$

$y$- update will be given by

$$y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - z^{k+1}).$$

Averaging over $y$-update we get

$$\bar{y}^{k+1} = \bar{y}^k + \rho \bar{x}^{k+1} - \bar{z}^{k+1}$$

Using (6.2), we have

$$\bar{y}^{k+1} = \bar{y}^k + \rho \bar{x}^{k+1} - \bar{y}^k - \rho \bar{x}^{k+1}$$

$$\bar{y}^{k+1} = 0.$$

This means that from (6.2)

$$z^{k+1} = \bar{x}^{k+1}.$$

Finally, ADMM algorithm for consensus problem will be

$$\begin{aligned} x_i^{k+1} &= \text{argmin}(f_i(x_i)) + y_i^{kT}(x_i - \bar{x}) + (\rho/2)||x_i - \bar{x}^k||_2^2) \\ y_i^{k+1} &= y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1}). \end{aligned} \tag{6.3}$$

Looking at the first part of (6.3), we can see that at each step of the algorithm each node minimizes its local cost function $f_i$, plus 2 terms that try to pull $x_i$ towards $\bar{x}$, and so drive the local variables into consensus.

### 6.1.1 Global variable consensus with regularization

In a simple variation on the global variable consensus problem an objective term $g$ is added, which usually represents regularization.

$$\sum_{i=1}^{N} f_i(x_i) + g(z) \quad \text{s.t. } x_i - z = 0, \ i = 1, \ldots, N.$$

The resulting ADMM algorithm is

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\text{argmin}} \left( f_i(x_i) + y_i^{kT}(x_i - z^k) + (\rho/2)||x_i - z^k||_2^2 \right) \\ z^{k+1} &= \underset{z}{\text{argmin}} \left( g(z) + \sum_{i=1}^{N} (-y_i^{kT} z + (\rho/2)||x_i^{k+1} - z||_2^2) \right) \\ y_i^{k+1} &= y_i^k + \rho(x_i^{k+1} - z^{k+1}). \end{aligned} \tag{6.4}$$

$z$- update can be put in simpler form as follows

$$\begin{aligned}
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left( g(z) - \sum_{i=1}^{N} (y_i^{kT} z) + (\rho/2) \sum_{i=1}^{N} (x_i^{k+1})^T (x_i^{k+1}) - \rho \sum_{i=1}^{N} (x_i^{k+1})^T z + (\rho/2) \sum_{i=1}^{N} z^T z \right) \\
&= \underset{z}{\operatorname{argmin}} \left( g(z) - \sum_{i=1}^{N} (y_i^{kT}) z - \rho \sum_{i=1}^{N} (x_i^{k+1})^T z + N(\rho/2) z^T z \right) \\
&= \underset{z}{\operatorname{argmin}} \left( g(z) - N(\bar{y}^k)^T z - N\rho(\bar{x}^{k+1})^T z + N(\rho/2) z^T z \right) \\
&= \underset{z}{\operatorname{argmin}} \left( g(z) + (N\rho/2) \big( (-2/\rho)(\bar{y}^k)^T z - 2(\bar{x}^{k+1})^T z + (\rho/2) N z^T z \big) \right) \\
&= \underset{z}{\operatorname{argmin}} \left( g(z) + (N\rho/2) \left( \frac{\bar{y}^{kT} \bar{y}^k}{\rho^2} + z^T z + (\bar{x}^{k+1})^T (\bar{x}^{k+1}) - 2(\bar{x}^{k+1})^T z + 2(\bar{x}^{k+1})^T y^k - \frac{2}{\rho} \bar{y}^{kT} z \right) \right) \\
&= \underset{z}{\operatorname{argmin}} \left( g(z) + (N\rho/2) ||z - \bar{x}^{k+1} - \bar{y}^k/\rho||_2^2 \right).
\end{aligned}$$

In the fifth equation we have added some additional terms to bring this problem in the final form. These terms do not contain $z$, so they do not affect the solution of the optimization problem at all. Now let us consider $x_i$ update

$$\begin{aligned}
x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + y_i^{kT} (x_i - \bar{z}^k) + (\rho/2) ||x_i - z^k||_2^2 \right) \\
&= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2) \left( \frac{2 y_i^{kT} x_i}{\rho} - \frac{2 y_i^{kT} \bar{z}^k}{\rho} + x_i^T x_i - 2 x_i^T z^k + z^{kT} z + y_i^{kT} y_i^k \right) \right) \\
&= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2) ||x_i - z^k + y^k/\rho||_2^2 \right).
\end{aligned}$$

In the second equation we added term $y_i^{kT} y_i^k$. Since it does not contain $x$, it does not affect the optimization problem. Now let us define scaled dual variable $u = (1/\rho)y$. Scaled ADMM for the global variable consensus with regularization problem will be

$$\begin{aligned}
x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2) ||x_i - z^k + u_i^k||_2^2 \right) \\
z^{k+1} &= \underset{z}{\operatorname{argmin}} \left( g(z) + (N\rho/2) ||z - \bar{x}^{k+1} - \bar{u}^{k+1}||_2^2 \right) \\
u_i^{k+1} &= u_i^k + x_i^{k+1} - z^{k+1}.
\end{aligned} \tag{6.5}$$

## 6.2 Sharing Problem

Another problem with many applications is the sharing problem

$$\text{minimize} \ \sum_{i=1}^{N} f_i(x_i) + g\left( \sum_{i=1}^{N} x_i \right), \tag{6.6}$$

with variables $x_i \in \Re^n$, $i = 1, \ldots, N$, where $f_i$ is a local cost function for subsystem $i$, and $g$ is the shared objective, which takes as argument the sum of the variables. Sharing can be written in ADMM form as

$$\begin{aligned}
&\text{minimize} \ \sum_{i=1}^{N} f_i(x_i) + g\left( \sum_{i=1}^{N} z_i \right) \\
&\text{subject to } x_i - z_i = 0, \ \ i = 1, \ldots, N,
\end{aligned} \tag{6.7}$$

with variables $x_i$, $z_i \in \Re^n$, $i = 1, \ldots, N$. The scaled form of ADMM for sharing is

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2)||x_i - z_i^k + u_i^k||_2^2 \right) \\ z_i^{k+1} &= \underset{z}{\operatorname{argmin}} \left( g(\sum_{i=1}^N z_i) + (\rho/2) \sum_{i=1}^N ||z_i - u_i^{k+1} + x_i^{k+1}||_2^2 \right) \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z_i^{k+1}. \end{aligned} \tag{6.8}$$

$z$- update requires solving a problem in $Nn$ variables, but we will show that it is possible to carry it out by solving a problem in just $n$ variables. Let $\alpha_i = u_i^k + x_i^{k+1}$. Then $z$-update can be written as

$$\begin{aligned} &\text{minimize } g(N\bar{z}) + (\rho/2) \sum_{i=1}^N ||z_i - \alpha_i||_2^2 \\ &\text{subject to } \bar{z} = (1/N) \sum_{i=1}^N z_i, \end{aligned}$$

with additional variable $\bar{z} \in \Re^n$. We will try to solve the problem above minimizing over $z_1, \ldots, z_N$ with $\bar{z}$ fixed. This has solution

$$z_i = \alpha_i + \bar{z} - \bar{\alpha}, \tag{6.9}$$

so the $z$- update can be computed by solving the unconstrained problem

$$\text{minimize } g(N\bar{z}) + (\rho/2) \sum_{i=1}^N ||\bar{z} - \bar{\alpha}||_2^2$$

for $\bar{z} \in \Re^n$ and then applying (6.9). Substituting (6.9) for $z_i^{k+1}$ in the $u$-update gives

$$u_i^{k+1} = \bar{u}^k + \bar{x}^{k+1} - \bar{z}^{k+1},$$

which shows that the variables $u_i^k$ are all equal and can be replaced with a dual variable $u \in \Re^m$. Substituting in the expression for $z_i^k$ in the $x$-update, the final algorithm becomes

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2)||x_i - x_i^k + \bar{x}^k - \bar{z}^k + u^k||_2^2 \right) \\ \bar{z}^{k+1} &= \underset{\bar{z}}{\operatorname{argmin}} \left( g(N\bar{z}) + (N\rho/2) \sum_{i=1}^N ||\bar{z} - u^k - \bar{x}^{k+1}||_2^2 \right) \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z_i^{k+1}. \end{aligned} \tag{6.10}$$

A more detailed proof for the derivation of ADMM algorithm for sharing problem can be found in the appendix.

### 6.2.1 Optimal Exchange

Here we describe a special case of sharing problem, which is called *optimal exchange*.

$$\begin{aligned} &\text{minimize } \sum_{i=1}^N f_i(x_i) \\ &\text{subject to } \sum_{i=1}^N x_i = 0, \end{aligned} \tag{6.11}$$

with variables $x_i \in \Re^n$, $i = 1, \ldots, N$, where $f_i$ represents the cost function for subsystem $i$. This is a sharing problem where the shared objective $g$ is the indicator function of the set $\{0\}$. ADMM algorithm for the exchange problem will be

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( f_i(x_i) + (\rho/2)||x_i - x_i^k + \bar{x}^k + u^k||_2^2 \right) \\ u^{k+1} &= u^k + \bar{x}^{k+1}. \end{aligned} \tag{6.12}$$

A more detailed proof on the derivation of this algorithm can be found on the appendix.

## 6.3 Applications on machine learning and signal processing

Now we will see how we can use the consensus and sharing techniques to solve large scale problems arising in model fitting (regression, classification) and signal processing (compressed sensing). Let us now make an introduction to model fitting A general convex model fitting can be written in form

$$\text{minimize } l(Ax - b) + r(x), \tag{6.13}$$

with parameters $x \in \Re^n$, where $A \in \Re^{m \times n}$ is the feature matrix, $b \in \Re^m$ is the output vector, $l : \Re^m \to \Re$ is a convex loss function, and $r$ is a convex regularization function. We assume that $l$ is additive, so

$$l(Ax - b) = \sum_{i=1}^{m} l_i({a_i}^T x - b_i),$$

where $l_i : \Re \to \Re$ is the loss function for the $i$th training example, $a_i \in \Re^n$ is the feature vector for example $i$, and $b_i$ is the response for example $i$. We also assume the regularization function $r$ is seperable. The most common examples are $r(x) = \lambda||x||_2^2$ (called *ridge penalty*), and $r(x) = \lambda||x||_1$ (called *lasso penalty*).

### 6.3.1 Regression

A special case of model fitting is Regression. Consider a linear model fitting problem, with measurements of the form

$$b_i = a_i^T x + v_i,$$

where $a_i$ is the $i$th feature vector, $b \in \Re$, and independent measurement noises$u_i$. $r$ term is used for regularization, or to impose constraints, coming from our prior knowledge about $x$. Our goal is to estimate the parameters of the model $x$ (based on measurements and prior knowledge about $x$).

### 6.3.2 Classification

Many classification problems can also be put in form of the general model fitting problem (6.13), with $A$, $b$, $l$, and $r$ appropriately chosen. Let $p_i \in \Re^{n-1}$, denote the feature vector of the $i$th example and let $q_i \in \{-1, 1\}$ denote the binary outcome or class label, for $i = 1, \ldots, m$. The goal is to find a weight vector $v\Re^{n-1}$ and offset $v \in \Re$, such that

$$sign(p_i^T w + v) = q_i$$

holds for many examples. Expression $p_i^T w + v$ is called the *discriminant function*. Expression $\mu_i = q_i(p_i^T w + v)$ is called the *margin* of the $i$th training example. Loss functions in the context of classification are written as functions of margin , so the loss for the $i$th training example is

$$l_i(\mu_i) = l_i\big(q_i(p_i^T w + v)\big).$$

A classification error is made if and only if the margin is negative, so $l_i$ should be positive and decreasing for negative arguments and zero or small for positive arguments. To find the parameters $w$ and $v$, we minimize the average loss plus a regularization term on weights. Some common loss functions are *hinge loss* $(1-\mu_i)_+$, *exponential loss* $\exp(-\mu_i)$, and *logistic loss* $\log(1+\exp(-\mu_i))$.

## 6.4 Splitting across examples

Now we will discuss how to solve model fitting problem (6.13) with a very large number of training examples and a modest number of features. Our goal is to solve the problem, such that a processor only handles a subset of the training data. This technique is useful, either when are too many training examples and it is inconvenient to process them on a single machine or when the data is collected or stored in a distributed fashion. First, we partition $A$ and $b$ by rows,

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_N \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix},$$

with $A_i \in \Re^{m_i \times n}$ and $b_i \in \Re^{m_i}$, where $\sum_{i=1}^N m_i = m$. Thus, $A_i$ and $b_i$ represent the $i$th block of data and will be handled by the $i$th processor. We first put the model fitting problem in the consensus form

$$\begin{array}{l} \text{minimize } \displaystyle\sum_{i=1}^N l_i(A_i x_i - b_i) + r(z) \\ \text{subject to } x_i - z = 0,\ i = 1, \ldots, N, \end{array} \tag{6.14}$$

where variables $x_i \in \Re^n$, and $z \in \Re^n$. Here, $l_i$ refers to the loss function for the $i$th block of data. The problem can now be solved by applying the global variable consensus ADMM with regularization algorithm

$$\begin{array}{l} x_i^{k+1} = \underset{x_i}{\text{argmin}} \left( f_i(x_i) + (\rho/2)||x_i - z^k + u_i^k||_2^2 \right) \\ z^{k+1} = \underset{z}{\text{argmin}} \left( r(z) + (N\rho/2)||z - \bar{u}^k - \bar{x}^{k+1}||_2^2 \right) \\ u_i^{k+1} = u_i^k + x_i^{k+1} - z^{k+1}. \end{array} \tag{6.15}$$

The first step, is a $l_2$ regularized model fitting problem, can be carried out in parallel for each data block. The second step requires gathering variables to compute the average and then perform the minimization step. Now let us consider some applications of it.

### 6.4.1 Least squares regression

Let us consider a "tall" ordinary least squares problem ($m \gg n$). It can be split in the way we describe in the beginning of this section.

$$\begin{array}{l} x_i^{k+1} = \underset{x_i}{\text{argmin}} \left( (1/2)||A_i x - b_i||_2^2 + y_i^{kT}(x_i - \bar{x}^k) + (\rho/2)||x_i - \bar{x}^k||_2^2 \right) \\ y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1}). \end{array} \tag{6.16}$$

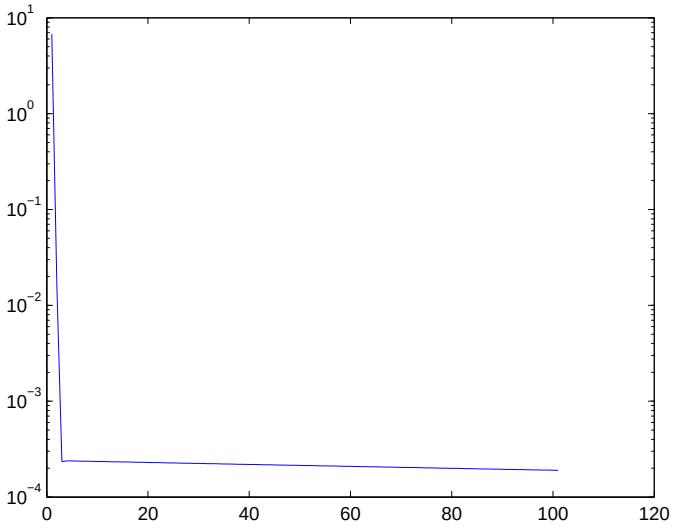
Solving minimization problem for $x_i$, ADMM algorithm becomes

$$\begin{array}{l} x_i^{k+1} = (A_i^T A_i + \rho I)^{-1}(A_i^T b_i - y_i + \rho \bar{x}) \\ y_i^{k+1} = y_i^k + \rho(x_i^{k+1} - \bar{x}^{k+1}). \end{array} \tag{6.17}$$

### 6.4.2 Numerical example. Consensus Least squares

For our experiment we have created a tall feature matrix of size $10000 \times 50$, and we have distributed the matrix into 20 blocks. We use consensus least squares algorithm to train our model and we calculate the residual (difference between actual and fitted model parameters).

Figure 6.1: Consensus Least Regression.




In figure we observe that the algorithm converges very fast in the first two iterations, but then the algorithm converges very slow.

### 6.4.3 Lasso

Now let us consider a "tall" lasso problem. Consensus ADMM algorithm for this problem comes straight from (6.15)

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( ||A_i x_i - b_i||_2^2 + (\rho/2)||x_i - z^k + u_i^k||_2^2 \right) \\ z^{k+1} &= \underset{z}{\operatorname{argmin}} \left( \lambda||z||_1 + (N\rho/2)||z - \bar{u}^k - \bar{x}^{k+1}||_2^2 \right) \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned} \tag{6.18}$$

$x_i$ update takes the form of a Tikhonov-regularized least squares problem, with solution

$$x_i^{k+1} = (A_i^T A_i + \rho I)^{-1}(A_i^T b_i + \rho(z^k - u_i^k)).$$

$z$ update will be given by the soft thresholding operator as

$$z^{k+1} = S_{\lambda/\rho N}(\bar{x}^{k+1} + \bar{u}^k).$$

### 6.4.4 Sparse Logistic Regression

The next problem to solve will be the classification problem, described in , using logistic loss function $l_i$ and $l_1$ regularization. ADMM algorithm for this problem will be

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( ||l_i(A_i x_i) - b_i||_2^2 + (\rho/2)||x_i - z^k + u_i^k||_2^2 \right) \\ z^{k+1} &= S_{\lambda/\rho N}(\bar{x}^{k+1} + \bar{u}^k) \\ u_i^{k+1} &= u_i^k + x_i^{k+1} - z^{k+1}. \end{aligned} \tag{6.19}$$

This is identical to distributed lasso, except for $x_i$ update, which here involves a $l_2$ regularized logistic regression problem that can be efficiently solved by Newton algorithm [5].

## 6.5 Splitting along features

Now we consider the model fitting problem with a modest number of examples and a large number of features. Problems like these arise in areas like natural language processing, where the observations might be a corpus of documents, and features could include all the words that appear in each document. The goal is to solve these problems, by letting each processor to handle only a subset of the features. In this section we will show how this can be done by formulating it as a sharing problem. We partition parameter vector $x$ as $x = (x_1, \ldots, x_N)$, with $x_i \in \Re^{n_i}$, where $\sum_{i=1}^N n_i = n$. Matrix $A$ can be partitioned as $A = \begin{bmatrix} A_1 & \ldots & A_N \end{bmatrix}$, with $A_i \in \Re^{m \times n_i}$, and the regularization function as $r(x) = \sum_{i=1}^N r_i(x_i)$. Model fitting problem becomes

$$\text{minimize } l\left( \sum_{i=1}^N A x_i - b \right) + \sum_{i=1}^N r_i(x_i).$$

We can put this problem in sharing ADMM form as follows

$$\begin{aligned} &\text{minimize } l\left( \sum_{i=1}^N z_i - b \right) + \sum_{i=1}^N r_i(x_i) \\ &\text{subject to } A_i x_i - z_i = 0, \ i = 1, \ldots, N, \end{aligned} \tag{6.20}$$

Scaled ADMM algorithm for this problem is

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( r_i(x_i) + (\rho/2)||A_i x_i - z_i^k + u_i^k||_2^2 \right) \\ z^{k+1} &= \underset{z}{\operatorname{argmin}} \left( l\left( \sum_{i=1}^N z_i - b \right) + \sum_{i=1}^N (\rho/2)||A_i x_i - z_i^k + u_i^k||_2^2 \right) \\ u_i^{k+1} &= u_i^k + A_i x_i^{k+1} - z^{k+1}, \end{aligned} \tag{6.21}$$

where $z_i \in \Re^m$, and $z \in \Re^{mN}$.

### 6.5.1 Lasso

Let us consider now the algorithm above for the Lasso problem

$$\begin{aligned} x_i^{k+1} &= \underset{x_i}{\operatorname{argmin}} \left( r_i(x_i) + (\rho/2)||A_i x_i - A_i x_i^k - \bar{z}^k + \overline{Ax}^k + u^k||_2^2 + \lambda||x||_1 \right) \\ \bar{z}^{k+1} &= \frac{1}{N+\rho} \left( b + \rho \overline{Ax}^{k+1} + \rho u^k \right) \\ u^{k+1} &= u^k + \overline{Ax}^{k+1} - \bar{z}^{k+1}. \end{aligned} \tag{6.22}$$

Each $x_i$-update is a lasso problem with $n_i$ variables, which can be solved by a serial lasso solver(i.e. CVX) .

### 6.5.2 Group Lasso

Let us now consider the group lasso with the feature groups coinciding with the blocks of features, and $l_2$ (not squared) regularization

$$\text{minimize } (1/2)||Ax - b||_2^2 - \lambda \sum_{i=1}^{N} ||x_i||_2.$$

The $z$-update and $u$-update are the same as in Lasso case but $x_i$- update is different

$$x_i^{k+1} = \underset{x_i}{\text{argmin}} \left( (\rho/2)||A_i x_i - A_i x_i^k - \bar{z}^k + \overline{Ax}^k + u^k||_2^2 + \lambda||x_i||_2 \right).$$

Set $v = A_i x_i^k + \bar{z}^k - \overline{Ax}^k - u^k$. So $x_i$ update involves minimizing function

$$(\rho/2)||A_i x_i - v||_2^2 + \lambda||x_i||_2.$$

Solution for this problem is, $x_i = 0$, if and only if $||A_i^T v||_2 \leq \lambda/\rho$. Otherwise, solution will be

$$x_i = (A_i^T A_i + wI)^{-1} A_i^T v,$$

for the value of $v > 0$ that gives $v||x_i||_2 = \lambda/\rho$. This value can be found using one parameter search (using bisection for example) over $v$.

# Chapter 7

# Implementation of distributed ADMM using MPI

## 7.1 MPI

We have implemented the ADMM algorithm for consensus least squares problem and test it on GRID computing system of Technical University of Crete [6]. We used the MPI (Message Passing Interface) framework of MATLAB and the programming was in SPMD [7] (single program, multiple data) style. Every node of the distributed computing system runs the same code, but has its own set of local variables. There can be communication between the processors, using the functions of MPI framework. In the next section we give a description of the algorithm we have implemented.

## 7.2 Algorithm Description

At first, we distribute the feature matrix along the processors, as well as the observation vector, so each node has only to handle its own part of the data.Then at each iteration of the algorithm we follow the steps below

1. Each processor computes its own $x$ according to the first part of equation (6.16) and sends that value to the central collector node.
2. Central collector then computes mean value $\bar{x}$ using the variables that received in the previous step and broadcasts it to every other processor.
3. Finally, each node update its local dual variable using the second part of equation (6.16).

# Appendix A

# ADMM

## A.1 Soft thresholding

We want to find

$$x_i^+ = \underset{x_i}{\operatorname{argmin}} \big(\lambda|x_i| + \rho/2(x_i - u_i)\big)^2.$$

Function $|x|$ is not differentiable at 0. However we can compute the subgradient of this function. Subderivative of this function is the interval $[-1, 1]$ at $x = 0$, 1 for $x > 0$ and $-1$ for $x < 0$. To minimize the function $\lambda|x_i| + \rho/2(x_i - u_i)$ , we have to set the gradient (or subgradient) of the function equal to 0. So:

$$0 = \nabla_{x_i}\big(\lambda|x_i^+| + (\rho/2)(x_i^+ - u_i)^2\big) = \lambda\nabla_{x_i}|x_i^+| + (\rho/2)\nabla_{x_i}(x_i^+ - u_i)^2 = \lambda\nabla_{x_i}|x_i^+| + \rho(x_i^+ - u_i).$$

We divide both sides by $\rho$. This leads to:

$$\frac{\lambda}{\rho}\nabla_{x_i}|x_i^+| + x_i^+ = u_i$$

We have 3 cases:

1. $x_i^+ > 0$, $\dfrac{\lambda}{\rho} + x_i^+ = u_i$. So $x_i+ = u_i - \dfrac{\lambda}{\rho}$. Since $x_i^+ > 0$: $u_i > \dfrac{\lambda}{\rho}$
2. $x_i^+ < 0$, $-\dfrac{\lambda}{\rho} + x_i^+ = u_i$. So $x_i+ = u_i + \dfrac{\lambda}{\rho}$. Since $x_i^+ < 0$: $u_i < -\dfrac{\lambda}{\rho}$
3. $x_i^+ = 0$, $\left[-\dfrac{\lambda}{\rho}, \dfrac{\lambda}{\rho}\right] = u_i \Rightarrow |u_i| < \dfrac{\lambda}{\rho}$

Following the above results, we define as *soft thresholding*, or shrinkage operator the below:

$$S_{\frac{\lambda}{\rho}}(u_i) = \begin{cases} u_i - \dfrac{\lambda}{\rho}, & u_i > \dfrac{\lambda}{\rho} \\ 0, & |u_i| \le \dfrac{\lambda}{\rho} \\ u_i + \dfrac{\lambda}{\rho}, & u_i < -\dfrac{\lambda}{\rho}. \end{cases} \tag{A.1}$$

## A.2 Quadratic function with equality constraints

We will show analytically here how we computed the optimality conditions for the problem of minimizing a quadratic function $(1/2)x^TPx + q^Tx$ over an affine set $\{x|Ax = b\}$ The $x$-minimization step of ADMM algorithm for this problem is

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left( \frac{1}{2} x^T P x + q^T x + \frac{\rho}{2} \|x - z^k + u^k\|_2^2 \right)$$

Primal optimality condition: $Ax = b$. The dual optimality condition for this problem will be

$$0 = \nabla_x \left( \frac{1}{2} x^T P x + q^T x + \frac{\rho}{2} (x - z^k + u^k)^T \big(x - z^k + u^k\big) + v^T (Ax - b) \right) = Px + q + \rho x - \rho z + \rho u^k + A^T v$$

This leads to

$$(P + \rho I)x + A^T v = -q + \rho(z - u). \tag{A.2}$$

The solution of the problem is given by the system of linear equations below

$$\begin{bmatrix} \rho I + P & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ v \end{bmatrix} = \begin{bmatrix} -q + \rho(z - u) \\ b \end{bmatrix}.$$

## A.3 Least Absolute Deviations

We will show analytically here how we derived the $x_{k+1}$ and $z_{k+1}$ for ADMM algorithm for Least Absolute Deviations.

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left( \frac{\rho}{2} \|Ax^k - z^k - b + u^k\|_2^2 \right).$$

This is a Least-Squares problem. So:

$$x^{k+1} = \big(A^T A\big)^{-1} A^T \big(b + z - u^k\big). \tag{A.3}$$

According to Scaled ADMM algorithm $z_{k+1}$ for least absolute deviations will be:

$$z^{k+1} = \underset{z}{\operatorname{argmin}} \left( \|z\|_1 + \frac{\rho}{2} \|Ax^{k+1} - z - b + u^k\|_2^2 \right)$$

We have that

$$\begin{aligned} 0 &= \nabla_z (\|z\|_1 + \frac{\rho}{2} (Ax^{k+1} - z - b + u^k)'(Ax^{k+1} - z - b + u^k)) \\ &= \nabla_z (\|z\|_1) + \frac{\rho}{2} \nabla_z \big(2x^{Tk+1} A^T z + z^T z + 2b^T z - 2u^{Tk} z\big) \\ &= \nabla_z (\|z\|_1) - \rho A x^{k+1} + \rho z + \rho b - \rho u^k = \nabla_z (\|z\|_1) - \rho A x^{k+1} + \rho z + \rho b - \rho u^k. \end{aligned}$$

This leads to

$$\begin{aligned} &\nabla_z (\|z\|_1) + \rho z = \rho A x^{k+1} - \rho b + \rho u^k \\ &\frac{1}{\rho} \nabla_z (\|z\|_1) + z = A x^{k+1} - b + u^k. \end{aligned}$$

We need to consider the following 3 cases:

1. If $z_i > 0$, $z_i = (Ax^{k+1})_i - b_i + u_i^k - \frac{1}{\rho}$. So $(Ax^{k+1})_i - b_i + u_i^k > \frac{1}{\rho}$
2. If $z_i < 0$, $z_i = (Ax^{k+1})_i - b_i + u_i^k + \frac{1}{\rho}$. So $(Ax^{k+1})_i - b_i + u_i^k < -\frac{1}{\rho}$
3. If $z_i = 0$, $[-\frac{1}{\rho}, \frac{1}{\rho}] = (Ax^{k+1})_i - b_i + u_i^k$. So $|(Ax^{k+1})_i - b_i + u_i^k| < \frac{1}{\rho}$

Following the above results we have

$$z^{k+1} = S_{\frac{1}{\rho}}(Ax^{k+1} - b + u^k). \tag{A.4}$$

## A.4 Huber Fitting

Now we will give a proof for $z$-update in ADMM in Huber Fitting problem. $z$-update is given by

$$z^{k+1} = \text{argmin}\left(\sum_{i=1}^{m} g^{hub}(z_i) + \frac{\rho}{2}||z - w||_2^2\right), \quad \text{where, } w = Ax^{k+1} + u^k - b.$$

This minimization can be split at component level and be performed individually for each $z_i$, $i = 1, \ldots, m$. We have to consider the 3 following cases

1. Suppose $|z_i| \leq 1$. Then $g^{hub}(z_i) = z_i^2/2$.

   $$z_i^{k+1} = \underset{z_i}{\text{argmin}}(z_i^2/2 + (\rho/2)(z_i - w_i)^2)$$
   $$z_i^{k+1} = \underset{z_i}{\text{argmin}}(z_i^2/2 + (\rho/2)z_i^2 + (\rho/2)w_i^2 - \rho w_i z_i).$$

   To derive the solution, we have to find where the gradient becomes zero.

   $$\nabla(z_i^2/2 + (\rho/2)z_i^2 + (\rho/2)w_i^2 - \rho z_i w_i) = 0.$$

   This leads to

   $$0 = z_i^{k+1} + \rho z_i - \rho w_i$$
   $$z_i^{k+1} = \rho\frac{w_i}{(1+\rho)}.$$

   Since we assumed $|z_i| \leq 1$, then $|w_i| \leq \frac{1+\rho}{\rho}$.

2. Now let $z_i > 1$ . Then $g^{hub}(z_i) = |z_i| = z_i$ This means that

   $$(z^{k+1})_i = \underset{z_i}{\text{argmin}}(z_i + (\rho/2)z_i^2 + (\rho/2)w_i^2 - 2\rho w_i z_i).$$

   Setting gradient to 0, we get

   $0 = 1 + \rho z_i^{k+1} - \rho w_i$. Finally, $z_i^{k+1} = w_i - \frac{1}{\rho}$.

   Since we assumed $|z_i| > 1$, then

   $$w_i - \frac{1}{\rho} > 0, \text{ and } w_i > \frac{1+\rho}{\rho}.$$

3. Now we assume $z_i < 1$. So $g^{hub}(z_i) = -z_i$. Setting gradient to 0, we get

   $0 = -1 + \rho z_i^{k+1} - \rho w_i$. Finally, $z_i^{k+1} = w_i + \frac{1}{\rho}$.

   Since we assumed $|z_i| < 1$, then

   $w_i + \frac{1}{\rho} < 0$, and $w_i < -\frac{1+\rho}{\rho}$.

So $z$-update is

$$z_i^{k+1} = \begin{cases} w_i - \frac{1}{\rho}, & w_i > \frac{1+\rho}{\rho}, \\ \rho \frac{w_i}{1+\rho}, & |w_i| \leq \frac{1+\rho}{\rho}, \\ w_i + \frac{1}{\rho}, & w_i < -\frac{1+\rho}{\rho}. \end{cases} \tag{A.5}$$

## A.5 Basis Pursuit

We will show how we derived $z_{k+1}$ of ADMM algorithm for Basis Pursuit problem.

$$z_{k+1} = \underset{z}{\operatorname{argmin}} \left( \|z\|_1 + \frac{\rho}{2} \|x^{k+1} - z + u^k\|_2^2 \right).$$

We have that:

$$\begin{aligned} 0 &= \nabla_z \|z\|_1 + \frac{\rho}{2} \nabla_z (x^{k+1} - z + u^k)^T (x^{k+1} - z + u^k) \\ &= \nabla_z \|z\|_1 + \frac{\rho}{2} \nabla_z (-2u^{(k+1)T} z - 2x^{(k+1)T} z + z^T z) \\ &= \nabla_z \|z\|_1 - \rho x^{k+1} - \rho u^k + \rho z. \end{aligned}$$

This leads to

$$\frac{1}{\rho} \nabla_z \|z\|_1 + z = x^{k+1} + u^k.$$

We have the 3 following cases

1. If $z_i > 0$, $z_i = x_i^{k+1} + u_i^k - \frac{1}{\rho}$. So $x_i^{k+1} + u_i^k > \frac{1}{\rho}$
2. If $z_i < 0$, $z_i = x_i^{k+1} + u_i^k + \frac{1}{\rho}$. So $x_i^{k+1} + u_i^k < -\frac{1}{\rho}$
3. If $z_i = 0$, $[-\frac{1}{\rho}, \frac{1}{\rho}] = x_i^{k+1} + u_i^k$. So $|x_i^{k+1} + u_i^k| < \frac{1}{\rho}$

Following the above results

$$z^{k+1} = S_{\frac{1}{\rho}}(x^{k+1} + u^k). \tag{A.6}$$

## A.6 Lasso

In this section we will show analytically how we derived the $x^{k+1}$, and the $z^{k+1}$ step of the ADMM algorithm for Lasso($l1$-regularized least-squares) problem.

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left( \frac{1}{2}\|Ax - b\|_2^2 + \frac{\rho}{2}\|x - z^k + u^k\|_2^2 \right)$$

We have that

$$\begin{aligned} 0 &= \frac{1}{2}\nabla_x (Ax-b)^T(Ax-b) + \frac{\rho}{2}\nabla_x (x - z^k + u^k)^T(x - z^k + u^k) \\ &= \frac{1}{2}\nabla_x (x^T A^T Ax - 2x^T A^T b) + \frac{\rho}{2}\nabla_x (x^T x - 2z^{kT}x + 2u^{kT}x) \\ &= A^T Ax - A^T b + \rho x - \rho z^k + \rho u^k. \end{aligned}$$

This leads to

$$(A^T A + \rho I)x = A^T b + \rho(z^k - u^k),$$

which is the solution of a linear system $Mx = c$, with $M = A^T A + \rho I$, and $c = A^T b + \rho(z^k - u^k)$.

$$x^{k+1} = (A^T A + \rho I)^{-1}(A^T b + \rho(z^k - u^k)). \tag{A.7}$$

For $z^{k+1}$ step we have

$$z_{k+1} = \underset{z}{\operatorname{argmin}}(\lambda\|z\|_1 + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2)$$

We have that

$$\begin{aligned} 0 &= \lambda\nabla_z\|z^k\|_1 + \frac{\rho}{2}\nabla_z (x^{k+1} - z + u^k)^T(x^{k+1} - z + u^k) \\ &= \lambda\nabla_z\|z\|_1 + \frac{\rho}{2}\nabla_z (z^T z - 2x^{(k+1)T}z - 2u^{kT}z) \\ &= \lambda\nabla_z\|z\|_1 - \rho x^{k+1} + \rho z - \rho u^k \\ &= \lambda\nabla_z\|z\|_1 - \rho x^{k+1} + \rho z - \rho u^k, \end{aligned}$$

We divide by $\rho$ each side to get

$$\frac{\lambda}{\rho}\nabla_z\|z\|_1 + z = x^{k+1} + u^k$$

We have the 3 following cases

1. If $z_i > 0$, $z_i = x_i^{k+1} + u_i^k - \frac{\lambda}{\rho}$. So $x_i^{k+1} + u_i^k > \frac{\lambda}{\rho}$
2. If $z_i < 0$, $z_i = x_i^{k+1} + u_i^k + \frac{\lambda}{\rho}$. So $x_i^{k+1} + u_i^k < -\frac{\lambda}{\rho}$
3. If $z_i = 0$, $[-\frac{\lambda}{\rho}, \frac{\lambda}{\rho}] = x_i^{k+1} + u_i^k$. So $|x_i^{k+1} + u_i^k| < \frac{\lambda}{\rho}$

Following the above results

$$z^{k+1} = S_{\frac{\lambda}{\rho}}(x^{k+1} + u^k). \tag{A.8}$$

## A.7 Generalized Lasso

In this section we will show analytically how we derived the $x^{k+1}$, and the $z^{k+1}$ step of the ADMM algorithm for generalized lasso problem.

$$x^{k+1} = \underset{x}{\operatorname{argmin}} \left( \frac{1}{2}\|x-b\|_2^2 + \frac{\rho}{2}\|Fx - z^k + u^k\|_2^2 \right)$$

We have that

$$\begin{aligned} 0 &= \frac{1}{2}\nabla_x (x-b)^T(x-b) + \frac{\rho}{2}\nabla_x (Fx - z^k + u^k)^T(Fx - z^k + u^k) \\ &= \frac{1}{2}\nabla_x (x^T x - 2x^T b) + \frac{\rho}{2}\nabla_x (x^T F^T Fx - 2z^{kT} Fx + 2u^{kT} Fx) \\ &= x - b + \rho F^T Fx - \rho F^T z^k + \rho F^T u^k. \end{aligned}$$

This leads to

$$(I + \rho F^T F)x = b + \rho F^T (z^k - u^k),$$

which is the solution of a linear system $Mx = c$, with $M = I + \rho F^T F$, and $c = b + \rho F^T (z^k - u^k)$.

$$x^{k+1} = (I + \rho F^T F)^{-1}(b + \rho(z^k - u^k)). \tag{A.9}$$

For $z^{k+1}$ step we have:

$$z_{k+1} = \underset{z}{\operatorname{argmin}} (\lambda\|z\|_1 + \frac{\rho}{2}\|Fx^{k+1} - z + u^k\|_2^2)$$

We have that

$$\begin{aligned} 0 &= \lambda\nabla_z \|z^k\|_1 + \frac{\rho}{2}\nabla_z (Fx^{k+1} - z + u^k)^T (Fx^{k+1} - z + u^k) \\ &= \lambda\nabla_z \|z\|_1 + \frac{\rho}{2}\nabla_z (z^T z - 2x^{(k+1)T} F^T z - 2u^{kT} z) \\ &= \lambda\nabla_z \|z\|_1 - \rho Fx^{k+1} + \rho z - \rho u^k. \end{aligned}$$

Dividing 2 sides of the equation by $\rho$

$$\frac{\lambda}{\rho}\nabla_z \|z\|_1 + z = Fx^{k+1} + u^k.$$

We have the 3 following cases

1. If $z > 0$, $z = Fx^{k+1} + u^k - \frac{\lambda}{\rho}$. So $Fx^{k+1} + u^k > \frac{\lambda}{\rho}$
2. If $z < 0$, $z = Fx^{k+1} + u^k + \frac{\lambda}{\rho}$. So $Fx^{k+1} + u^k < -\frac{\lambda}{\rho}$
3. If $z = 0$, $[-\frac{\lambda}{\rho}, \frac{\lambda}{\rho}] = Fx^{k+1} + u^k$. So $|Fx^{k+1} + u^k| < \frac{\lambda}{\rho}$

Following the above results

$$z^{k+1} = S_{\frac{\lambda}{\rho}}(Fx^{k+1} + u^k). \tag{A.10}$$

# Appendix B

# MPI

We will now describe the most basic functions of MPI in Matlab and their functionality.

- **labSend**: A node calls this function to send data to another node. It's a non-blocking function, which means that a processor after sending data can continue executing commands, even if the receiving lab hasn't received the data.
- **labReceive**: A node calls this function to receive data sent by another node (using labsend function). It is a blocking function, which means that a processor cannot continue executing commands until reception of the data is complete.
- **labBroadcast**: The labBroadcast() function is used to send data from one node to all other nodes. Each processor is blocked until it receives the data.
- **labBarrier**: This function that can be used to block execution of each node until all nodes reach the labBarrier() statement. It can be used for synchronization, but it is not often necessary (because as we said before labReceive and labBroadcast are blocking functions).
- **labindex**: Each node is assigned a unique ID number(from 1 to the total number of nodes). labindex() returns this number. It can be used to assign specific commands to a node.
- **numlabs**: This function returns the total number of nodes.

For source code examples and tutorial of how to run these examples on a distributed computing system a useful link is [8]

Here is the code for consensus LS-ADMM

```
%%%%%%%% M-file for parallel ADMM LASSO CONSENSUS problem. %%%%%%%

function e_time = p_ADMM(M,sparse_x)

p = 100;
rho = 1;                        % Penalty factor
n = 50;                         % Dimension of minimization variable x
N = M*p;                        % Number of equations of least squares problem
%l = 1;                         % Regularization parameter
A = randn(N/numlabs,n);         % Distributed matrix A

%tmp_x = 5 * randn(n,1); pos_x = [abs(tmp_x) > 6];
%sparse_x = tmp_x .* pos_x;   % sparse parameter vector

w = 2 * randn(N/numlabs,1);
b = A * sparse_x + w;

for ii=1:M/numlabs
    invA(:,:,ii) = ( A((ii - 1)*p + 1 : (ii - 1)*p + p,:)' * A((ii - 1)*p + 1 : (ii - 1)*p + p,:) + rho
    Ab(:,ii) = invA(:,:,ii) * A((ii - 1)*p + 1 : (ii - 1)*p + p,:)' * b((ii - 1)*p + 1 : (ii - 1)*p + p
end

iters = 1000;
x_bar = zeros(n,1);                     %Initial x_bar
y = rand(n,M/numlabs);                  %Initial vector y
x = zeros(n,M/numlabs);

tic;
labBarrier();                           %Block untill all labs reach that point.Synchronization point #1.

 iter = 1;
while(1)
  iter = iter + 1;

 for ii=1:M/numlabs
  x(:,ii) =  Ab(:,ii) + rho* invA(:,:,ii) *(x_bar - y(:,ii));  %Update local x
 end


 labBarrier();                          %Block untill all labs reach that point.Synchronization point #2.

 if(labindex ~= 1)                      %This block of code is executed by the nodes who update the local

  labSend(x,1);                         %Send value of x from each local node to node with index 1

  x_bar = labBroadcast(1);              %Receive global variable x_bar

 else                                   %This block of code is executed by the node who computes the glob
   nlsp =M/numlabs;
   x_b(:,1:nlsp) = x;


     for ii = 2:numlabs
   x_b(:,(ii-1)*nlsp + 1: ii*nlsp) =  labReceive(ii);                %Receive  value of x from each lo
     end

   x_bar = mean(x_b')';                 %Compute global variable x_bar
   x_bar = labBroadcast(1,x_bar);       %Broadcast global variable x_bar

 end


 for ii=1:M/numlabs
    y(:,ii) = y(:,ii) + rho*(x(:,ii) - x_bar);                      %Update local value of y.
 end

  labBarrier();                          %Block untill all labs reach that point.Synchronization point #3
```

```
  if(iter == iters)
  break;
  end

end

e_time = toc;
```

# Bibliography


[1] S. Boyd, N. Parikh, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, 2011.

[2] D. Donoho, "Compressed sensing," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 52, 2006.

[3] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society*, vol. 58, 1996.

[4] S.-J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, "$l_1$ trend filtering," *SIAM REVIEW*, 2009.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. 2004.

[6] *Homepage for grid computer of Technical University of Crete.*

[7] *SPMD description.*

[8] *Tutorial for using MPI in Matlab.*