Distributed BP For Vertical Partition: Problem For Each Node

João Mota*

June 30, 2009

Abstract

It is presented an efficient method to solve the problem that appears when we apply the method of multipliers together with the Nonlinear Gauss–Seidel to solve the basis pursuit (BP), *i.e.* minimizing $\|x\|_1 = |x_1| + \ldots + |x_n|$ subject to $Ax = b$ ($A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$), distributedly. We assume a vertical partition for the matrix A. We also make a detailed flop count analysis.

1 Introduction

In [2] we saw that the application of the method of multipliers (outer loop) together with the Nonlinear Gauss-Seidel (inner loop) yields, for each node, a problem with the following format

$$\begin{array}{ll} \min\limits_{\substack{Ax = b \\ \text{var}:\, x \in \mathbb{R}^n}} & \|x\|_1 + v^\top x + c\|x\|^2, \end{array} \tag{1}$$

where $v, x, a \in \mathbb{R}^n$ and $c \in \mathbb{R}_+$. In this paper we describe an efficient solver for this problem, which can be found in Matlab code in `http://www.isr.ist.utl.pt/~jmota/software.html`. First, we consider the case where A is just one row, *i.e.* the problem

$$\begin{array}{ll} \min\limits_{\substack{a^\top x = b \\ \text{var}:\, x \in \mathbb{R}^n}} & \|x\|_1 + v^\top x + c\|x\|^2. \end{array} \tag{2}$$

In what follows, we reformulate problem (2) and describe briefly our approach.

Using the epigraph technique, (2) is equivalent to

$$\begin{array}{ll} \min\limits_{\substack{x \leq t \\ -x \leq t \\ a^\top x = b \\ \text{var}:\, (x,t) \in \mathbb{R}^n \times \mathbb{R}^n}} & 1_n^\top t + v^\top x + c\|x\|^2, \end{array}$$

being $1_n = [1, 1, \ldots, 1]^\top \in \mathbb{R}^n$,

$$\Longleftrightarrow \quad \min\limits_{\substack{\begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} \leq 0_{2n} \\ \begin{bmatrix} a^\top & 0_n^\top \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} = b}} \quad \begin{bmatrix} v^\top & 1_n^\top \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix} + c \begin{bmatrix} x^\top & t^\top \end{bmatrix} \begin{bmatrix} I_n & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix} \begin{bmatrix} x \\ t \end{bmatrix},$$

*I would like to thank João Xavier and Markus Püschel for their insightful comments and suggestions about this work.

where I_n is the identity matrix in $\mathbb{R}^{n\times n}$, $0_n = [0,0,\dots,0]^\top \in \mathbb{R}^n$ and $0_{n\times n}$ is the zero matrix in $\mathbb{R}^{n\times n}$,

$$\iff \min_{\substack{Mu \le 0_{2n} \\ p^\top u = b \\ \text{var: } u \in \mathbb{R}^{2n}}} \quad s^\top u + cu^\top Ru, \tag{3}$$

where

$$p = \begin{bmatrix} a \\ 0_n \end{bmatrix} \qquad s = \begin{bmatrix} v \\ 1_n \end{bmatrix} \qquad u = \begin{bmatrix} x \\ t \end{bmatrix}$$

$$M = \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \qquad R = \begin{bmatrix} I_n & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix}.$$

Problem (3) is a quadratic program (QP). We will use an interior point method to solve it: barrier method [1, page 568]. Doing so, we transform (3) into a sequence of problems

$$\{ \min_{\substack{p^\top u = b \\ \text{var: } u \in \mathbb{R}^{2n}}} \quad \mu_k \left(s^\top u + cu^\top Ru\right) - \sum_{i=1}^{2n} \log(-m_i^\top u)\}_{k=0}^{+\infty}, \tag{4}$$

where

$$M = \begin{bmatrix} m_1^\top \\ m_2^\top \\ \vdots \\ m_{2n}^\top \end{bmatrix}.$$

Note that each subproblem in (4) has the implicit constraints $m_i^\top u < 0$, $i = 1,\dots,2n$.

Let $f(u) = \mu_k \left(s^\top u + cu^\top Ru\right) - \sum_{i=1}^{2n} \log(-m_i^\top u)$, then

$$\nabla f(u) = \mu_k s + 2\mu_k cRu + M^\top d(u)$$
$$\nabla^2 f(u) = 2\mu_k cR + M^\top \text{Diag}^2(d(u))M,$$

where

$$d(u) = \begin{bmatrix} \frac{1}{-m_1^\top u} \\ \frac{1}{-m_2^\top u} \\ \vdots \\ \frac{1}{-m_{2n}^\top u} \end{bmatrix} = \begin{bmatrix} -\frac{1}{x_1 - t_1} \\ \vdots \\ -\frac{1}{x_n - t_n} \\ ----- \\ \frac{1}{x_1 + t_1} \\ \vdots \\ \frac{1}{x_n + t_n} \end{bmatrix} = \begin{bmatrix} B_1 \\ ----- \\ B_2 \end{bmatrix}.$$

We use the following notation[1] (for $a \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$)

$$\text{Diag}(a) = \begin{bmatrix} a_1 & 0 & \cdots & 0 \\ 0 & a_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \end{bmatrix}, \quad \text{diag}(\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}) = \begin{bmatrix} a_{11} \\ a_{22} \\ \vdots \\ a_{nn} \end{bmatrix}.$$

[1] Note that in Matlab code both operations are performed using the same notation: diag. And the output is a vector (resp. matrix) if the input is a matrix (resp. vector).

Newton's Method. In each step of the barrier method, we have to solve (see (4))

$$\min_{p^\top u=b} f(u),$$

for a function $f(u)$ which is twice continuously differentiable (in its domain). Under this conditions, it is possible to use a fast converging line search algorithm: Newton's method. In [1, Chapter 10], Boyd and Vanderberghe present a way to implement a version of Newton's method capable to deal with equality constraints. In section 5, we will see the implementation of this algorithm in detail. At this point, it is important to note that at each iteration of Newton's method we have to solve the following linear system

$$\begin{bmatrix} \nabla^2 f(u) & p \\ p^\top & 0 \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(u) \\ p^\top u - b \end{bmatrix}. \tag{5}$$

Our goal is to solve this system efficiently, since it is usually the bottleneck of Newton's method. We will see that the structure of the matrix on the left–hand side of (5) will allow us to solve that linear system with $\mathcal{O}(n)$ flops, being far more efficient than general purpose algorithms that solve any kind of linear systems ($\mathcal{O}(n^3)$ flops).

Organization. In sections 2 and 3 we will explore the structure of $\nabla f(u)$ and $\nabla^2 f(u)$ respectively. This will provide useful information when we solve (5) in section 4. Section 5 concerns the implementation of Newton's method and section 6 considers the outer loop of the the barrier method. In section 7 we measure the performance of the algorithm through some simulations. After that, in section 8, we show the steps needed to adapt the algorithm for the case where each node stores more than one row of A.

Throughout this text we will provide Matlab code for the implementation of this algorithm and at the same time make a flop count. The assumption we make is that at each code portion we have access to the variables defined in previous code portions.

2 Evaluating $\nabla f(u) = \mu_k s + 2\mu_k c R u + M^\top d(u)$

- $M^\top d(u) = \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = \begin{bmatrix} B_1 - B_2 \\ -B_1 - B_2 \end{bmatrix}$
- $\mu_k s + 2\mu_k c R u = \mu_k \begin{bmatrix} v \\ 1_n \end{bmatrix} + 2\mu_k c \begin{bmatrix} x \\ 0_n \end{bmatrix}$

Therefore,

$$\nabla f(u) = \begin{bmatrix} \mu_k v + 2\mu_k c x + B_1 - B_2 \\ \mu_k 1_n - (B_1 + B_2) \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}.$$

Code 1 (Evaluating $\nabla f(u)$ — $11n + 2$ flops)**.**
Having has inputs a, b, v, c, μ_k and $u = [\ x \quad t\ ]^\top$,

- `n = length(x);`
- `onesv = ones(n,1);`
- `B1 = 1./(t - x);` *(2n flops)*
- `B2 = 1./(t + x);` *(2n flops)*
- `g1 = mu*v + (2*mu*c)*x + B1 - B2;` *(5n + 2 flops)*
- `g2 = mu*onesv - (B1 + B2);` *(2n flops)*

---

3 Analysis of $\nabla^2 f(u) = 2\mu_k cR + M^\top \mathbf{Diag}^2(d(u))M$

- $M^\top \mathrm{Diag}^2(d(u))M$

$$
= \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix}
\left[\begin{array}{ccc|ccc}
\frac{1}{(x_1-t_1)^2} & & 0 & & & \\
 & \ddots & & & & \\
0 & & \frac{1}{(x_n-t_n)^2} & & & \\
\hline
 & & & \frac{1}{(x_1+t_1)^2} & & 0 \\
 & & & & \ddots & \\
 & & & 0 & & \frac{1}{(x_n+t_n)^2}
\end{array}\right]
\begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix}
$$

$$
:= \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \begin{bmatrix} \tilde{D}_1 & \\ & \tilde{D}_2 \end{bmatrix} \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix}
$$

$$
= \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \begin{bmatrix} \tilde{D}_1 & -\tilde{D}_1 \\ -\tilde{D}_2 & -\tilde{D}_2 \end{bmatrix}
$$

$$
= \begin{bmatrix} \tilde{D}_1 + \tilde{D}_2 & -\tilde{D}_1 + \tilde{D}_2 \\ -\tilde{D}_1 + \tilde{D}_2 & \tilde{D}_1 + \tilde{D}_2 \end{bmatrix}
$$

- $2\mu_k cR = 2\mu_k c \begin{bmatrix} I_n & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix} = \begin{bmatrix} 2\mu_k c I_n & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix}$

Therefore,

$$
\boxed{\nabla^2 f(u) = \begin{bmatrix} \tilde{D}_1 + \tilde{D}_2 + 2\mu_k c I_n & -\tilde{D}_1 + \tilde{D}_2 \\ -\tilde{D}_1 + \tilde{D}_2 & \tilde{D}_1 + \tilde{D}_2 \end{bmatrix}.}
$$

Note that, although $\nabla^2 f(u)$ is a matrix, we will not need to build any matrix because

- it is formed by blocks of diagonal matrices;
- we will only need to solve a linear system, *i.e.*, for this purpose, there is no need to build a matrix explicitly provided it has some simple structure.

At this point, we also note that $\nabla^2 f(u)$ is positive definite ($\nabla^2 f(u) \succ 0$): $M^\top \mathrm{Diag}^2(d(u))M \succ 0$ because if $w \in \mathbb{R}^{2n} \setminus \{0\}$, we have $w^\top M^\top \mathrm{Diag}^2(d(u))Mw = (Mw)^\top \mathrm{Diag}^2(d(u))(Mw) = q^\top \mathrm{Diag}^2(d(u))q > 0$, where $q \in \mathbb{R}^{2n}$ is arbitrary (notice that M is full–rank). This is true because all elements of $d(u).^2$ are positive ($.^2$ denotes the pointwise power of 2). As we are summing a positive semi–definite matrix ($2\mu_k cR$) to a positive definite matrix ($M^\top \mathrm{Diag}^2(d(u))M$), the result is positive definite.

Also note that[2]

$$
\tilde{D}_1 := \mathrm{Diag}(D_1) = \mathrm{Diag}(B_1.^2), \qquad \tilde{D}_2 := \mathrm{Diag}(D_2) = \mathrm{Diag}(B_2.^2).
$$

4 Solving The Linear System

Here, we will provide an algorithm to solve (5):

$$
\begin{bmatrix} \nabla^2 f(u) & p \\ p^\top & 0 \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(u) \\ p^\top u - b \end{bmatrix}.
$$

[2]From now on, we will represent a matrix with a tilde above and the respective diagonal by the same letter but without the tilde.

Calculating $[\nabla^2 f(u)]^{-1}$. First, we will derive an expression for the inverse of the hessian matrix $\nabla^2 f(u)$. Note that

$$\tilde{H} := \nabla^2 f(u) = \begin{bmatrix} \tilde{A}_h & \tilde{B}_h \\ \tilde{B}_h & \tilde{C}_h \end{bmatrix},$$

where the block matrices

$$\begin{aligned} \tilde{A}_h &= \tilde{D}_1 + \tilde{D}_2 + 2\mu_k c I_n \succ 0 \\ \tilde{B}_h &= \tilde{D}_2 - \tilde{D}_1 \\ \tilde{C}_h &= \tilde{D}_1 + \tilde{D}_2 \succ 0 \end{aligned}$$

are all diagonal.

The Schur complement of $\tilde{A}_h$ in $\tilde{H}$ is $\tilde{S}_h = \tilde{C}_h - \tilde{B}_h \tilde{A}_h^{-1} \tilde{B}_h$. Since $\tilde{H}, \tilde{A}_h \succ 0$, we also have $\tilde{S}_h \succ 0$ ([1, page 651]). Note that it is easy to compute the inverse of $\tilde{S}_h$, since it is a diagonal matrix.

The inverse of $\tilde{H}$ is given by

$$\begin{aligned} \tilde{H}^{-1} &= \begin{bmatrix} \tilde{A}_h^{-1} + \tilde{A}_h^{-1} \tilde{B}_h \tilde{S}_h^{-1} \tilde{B}_h \tilde{A}_h^{-1} & -\tilde{A}_h^{-1} \tilde{B}_h \tilde{S}_h^{-1} \\ -\tilde{S}_h^{-1} \tilde{B}_h \tilde{A}_h^{-1} & \tilde{S}_h^{-1} \end{bmatrix} \\ &:= \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix}. \end{aligned} \tag{6}$$

Notice that $\tilde{A}_h^{-1} \tilde{B}_h \tilde{S}_h^{-1} = \tilde{S}_h^{-1} \tilde{B}_h \tilde{A}_h^{-1}$ because all the matrices are diagonal.

Code for calculating the elements of $\tilde{H}^{-1}$. Now, the Matlab code to calculate the elements of $\tilde{H}^{-1}$. Note that we do not need to build any vector of size greater than n: we will only work either with vectors of size n or with scalars.

Code 2 (Calculating the elements of $\tilde{H}^{-1}$ — $16n + 2$ flops).

Assumption: we have access to all variables defined in Code 1.

- `D1 = B1.^2;` *(n flops)*
- `D2 = B2.^2;` *(n flops)*
- `d_plus = D1 + D2;` *(n flops)*
- `d_minus = D2 - D1;` *(n flops)*
- `A_h_inv = 1./(d_plus + (2*mu*c)*onesv);` *($2n + 2$ flops)*
- `S_h_inv = 1./(d_plus - (d_minus.*A_h_inv).*d_minus);` *($4n$ flops)*
- `aux1 = (S_h_inv.*d_minus).*A_h_inv;` *($2n$ flops)*
- `Q12 = -aux1;` *(n flops)*
- `Q11 = A_h_inv + (A_h_inv.*d_minus).*aux1;` *($3n$ flops)*

Solution of the linear system. The linear system that we need to solve is

$$\begin{bmatrix} \tilde{H} & p \\ p^\top & 0 \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} = \begin{bmatrix} -g_1 \\ -g_2 \\ c_g \end{bmatrix},$$

with

$$\tilde{H} = \nabla^2 f(u), \qquad p = \begin{bmatrix} a \\ 0_n \end{bmatrix}, \qquad \nabla f(u) = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}, \qquad c_g = b - p^\top u.$$

Let

$$A := \begin{bmatrix} \tilde{H} & p \\ p^\top & 0 \end{bmatrix}.$$

From a previous discussion, $\tilde{H} = \nabla^2 f(u) \succ 0$. Consequently, the Schur complement $\tilde{S}$ of $\tilde{H}$ in A is well defined and given by

$$\begin{aligned} \tilde{S} &= -p^\top \tilde{H}^{-1} p \\ &= -\begin{bmatrix} a^\top & 0_n^\top \end{bmatrix} \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} a \\ 0_n \end{bmatrix} \\ &= -a^\top \tilde{Q}_{11} a \\ &= -Q_{11}^\top (a.^2) \qquad (\in \mathbb{R}), \end{aligned}$$

where $Q_{11} = \mathrm{diag}(\tilde{Q}_{11})$.

The inverse of A is given by

$$A^{-1} = \begin{bmatrix} \tilde{H}^{-1} + \tilde{H}^{-1} p \tilde{S}^{-1} p^\top \tilde{H}^{-1} & -\tilde{H}^{-1} p \tilde{S}^{-1} \\ -\tilde{S}^{-1} p^\top \tilde{H}^{-1} & \tilde{S}^{-1} \end{bmatrix}.$$

And the solution of the linear system is

$$\begin{bmatrix} d \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} \tilde{H}^{-1} + \tilde{H}^{-1} p \tilde{S}^{-1} p^\top \tilde{H}^{-1} & -\tilde{H}^{-1} p \tilde{S}^{-1} \\ -\tilde{S}^{-1} p^\top \tilde{H}^{-1} & \tilde{S}^{-1} \end{bmatrix} \begin{bmatrix} -g \\ c_g \end{bmatrix},$$

where $g := \nabla f(u)$,

$$= \begin{bmatrix} -\tilde{H}^{-1} g - \tilde{H}^{-1} p \tilde{S}^{-1} p^\top \tilde{H}^{-1} g - \tilde{H}^{-1} p \tilde{S}^{-1} c_g \\ ---------------------- \\ \tilde{S}^{-1} p^\top \tilde{H}^{-1} g + \tilde{S}^{-1} c_g \end{bmatrix}.$$

Before we write the code to evaluate these expressions, let us first analyze some of its terms:

- $\tilde{H}^{-1} g = \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} (Q_{11} \odot g_1) + (Q_{12} \odot g_2) \\ (Q_{12} \odot g_1) + (Q_{22} \odot g_2) \end{bmatrix} := \begin{bmatrix} Hg1 \\ Hg2 \end{bmatrix}$,
  where $a \odot b$ denotes the Hadamard product of two vectors a and b, and $Q_{ij} = \mathrm{diag}(\tilde{Q}_{ij})$.
- $p^\top \tilde{H}^{-1} g = \begin{bmatrix} a^\top & 0_n^\top \end{bmatrix} \begin{bmatrix} Hg1 \\ Hg2 \end{bmatrix} = a^\top Hg1 := w$
- $\tilde{H}^{-1} p = \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} a \\ 0_n \end{bmatrix} = \begin{bmatrix} (Q_{11} \odot a) \\ (Q_{12} \odot a) \end{bmatrix} := \begin{bmatrix} Qa1 \\ Qa2 \end{bmatrix}$

Simplifying the solution,

$$\begin{bmatrix} d_1 \\ d_2 \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} -(Hg1 + w \cdot \tilde{S}^{-1} \cdot Qa1 + c_g \tilde{S}^{-1} \cdot Qa1) \\ -(Hg2 + w \cdot \tilde{S}^{-1} \cdot Qa2 + c_g \tilde{S}^{-1} \cdot Qa2) \\ ------------------ \\ w\tilde{S}^{-1} + c_g \tilde{S}^{-1} \end{bmatrix} = \begin{bmatrix} -(Hg1 + (w \cdot \tilde{S}^{-1} + c_g \tilde{S}^{-1})Qa1) \\ -(Hg2 + (w \cdot \tilde{S}^{-1} + c_g \tilde{S}^{-1})Qa2) \\ ---------------- \\ (w + c_g)\tilde{S}^{-1} \end{bmatrix}$$

$$= \begin{bmatrix} -(Hg1 + \lambda \cdot Qa1) \\ -(Hg2 + \lambda \cdot Qa2) \\ --------- \\ (w + c_g)\tilde{S}^{-1} \end{bmatrix}$$

Code 3 (Solving the linear system — $21n + 4$ flops)**.**
Assumption: we have access to all variables defined in Codes 1 and 2.

- `S_inv = -1/(Q11'*(a.^2));` *($4n + 1$ flops)*
- `Hg1 = (Q11.*g1) + (Q12.*g2);` *($3n$ flops)*
- `Hg2 = (Q12.*g1) + (S_h_inv.*g2);` *($3n$ flops)*
- `Qa1 = Q11.*a;` *(n flops)*
- `Qa2 = Q12.*a;` *(n flops)*
- `lambda = (a'*(Hg1-x) + b)*S_inv;` *($3n + 3$ flops)*
- `d1 = -(Hg1 + lambda*Qa1);` *($3n$ flops)*
- `d2 = -(Hg2 + lambda*Qa2);` *($3n$ flops)*

With this, the total number of flops required to calculate $\nabla f(u)$ and to solve the linear system (5) is $(11n + 2) + (16n + 2) + (21n + 4) = \boxed{48n + 8 \text{ flops}}$.

5 Modified Newton's Method

Suppose that we want to find the minimizer of a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$. A general descent method or line search algorithm consists of the following steps:

Algorithm 1 (Line Search Algorithm)**.**

Initialization

Choose $x_0 \in \mathbb{R}^n$ *and* $\epsilon > 0$*; set* $k = 0$.

Loop

- *Calculate* $g_k = \nabla f(x_k)$*;*
- *If* $\|g_k\| < \epsilon$, stop*;*
- *Find a descend direction* d_k*;*
- *Choose a step* α_k*;*
- $\boxed{x_{k+1} = x_k + \alpha_k d_k}$*;*
- $k \leftarrow k + 1$.

We say that a direction d_k is a *descent direction* of a function f at a point x_k if $d_k^\top \nabla f(x_k) < 0$, *i.e.*, if the function f decreases (locally) along that direction starting from x_k. If the function is twice continuously differentiable and strictly convex (this is equivalent to saying that $\nabla^2 f(x) \succ 0$ for any x), then $d_k = -[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$ is a descent direction. Indeed, $d_k^\top \nabla f(x_k) = \nabla^\top f(x_k) d_k = -\nabla^\top f(x_k)[\nabla^2 f(x_k)]^{-1} \nabla f(x_k) < 0$. Such direction is called a Newton's direction.

The idea of a line search algorithm is that from iteration to iteration we decrease the function f. In order to do that, the step α_k that we choose must yield $f(x_k + \alpha_k d_k) < f(x_k)$. There are several choices for this step (exact line minimization, Armijo's rule, Wolfe's rule,...), but we will only be concerned in using Armijo's rule, since it can be proved that the line search algorithm with Newton's direction and Armijo's rule converges (quadratically!) for strictly convex self–concordant functions ([1, page 503]). We have shown that the objective of (4) for each subproblem is strictly convex, since $\nabla^2 f(u) \succ 0$ (see page 3). We still need to show that

$$f(u) = \mu_k \left(s^\top u + c u^\top R u\right) - \sum_{i=1}^{2n} \log(-m_i^\top u)$$

is self–concordant. A function $f : \mathbb{R}^n \to \mathbb{R}$ is self–concordant if the function $\phi(\alpha) = f(u + \alpha d)$ verifies $|\frac{d^3}{d\alpha^3}\phi(\alpha)| \leq k[\frac{d^2}{d\alpha^2}\phi(\alpha)]^{3/2}$ for some positive k and for all u and v in $\mathbb{R}^n$. In [1, page 499] it is proved that the sum of self–concordant functions is also self–concordant, and that $-\sum_{i=1}^m \log(b_i - a_i^\top u)$ is self–concordant. Summing this function with a quadratic convex function yields a self–concordant function, since the third derivative of a convex quadratic function is zero and the second derivative is positive.

Newton's method with equality constraints has the same properties as the unconstrained Newton's method, namely quadratic convergence. The difference is that Newton's direction is obtained by solving a linear system of the type of (5). In fact, if we start from a feasible point ($p^\top u = b$ in our case) that system simplifies to

$$\begin{bmatrix} \nabla^2 f(u) & p \\ p^\top & 0 \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla f(u) \\ 0 \end{bmatrix}.$$

We will always assume this, as it makes sense in our context.

Also, the stopping criterion can no longer be $\|g_k\| < \epsilon$. The quantity $\sqrt{-\nabla^\top f(u) d}$ gives an estimate of the error on the cost function at the actual iteration, thus it will be used in the stopping criterion. This expression applies both to the constrained and the unconstrained cases.

Armijo's rule. The Armijo's rule or backtracking line search consists of the following. Consider the function $\phi : \mathbb{R} \to \mathbb{R}$ defined by $\phi(\alpha) = f(x_k + \alpha d_k)$, where d_k is a descent direction of f at the point x_k.

Algorithm 2 (Armijo's rule)**.**

Initialization

Choose $\alpha_0 > 0$*,* $0 < \beta < 1$ *and* $0 < c_1 < 1$ *(usually* $\alpha_0 = 1$ *and* $c_1 = 10^{-2} \sim 10^{-4}$*).*

Loop

- *If* $\phi(\alpha) \leq \phi(0) + c_1 \alpha \dot{\phi}(0)$*,* stop.
- $\alpha_{k+1} = \beta \alpha_k$*;*
- $k \leftarrow k + 1$.

Application to our problem. Given a direction d_j (solution of the linear system (5)), we need to compute $\phi(\alpha) = f(u_j + \alpha d_j)$, $\phi(0)$ and $\dot{\phi}(0)$.

- $$\begin{aligned}\phi(\alpha) &= \mu_k \left(s^\top u_j + c u_j^\top R u_j\right) + \mu_k \left(\alpha s^\top d_j + 2\alpha c u_j^\top R d_j + \alpha^2 c d_j^\top R d_j\right) \\ &\quad - \sum_{i=1}^{2n} \log(-m_i^\top u_j - \alpha m_i^\top d_j) \\ &= \underbrace{\mu_k \left[v^\top x_j + 1_n^\top t_j + c x_j^\top x_j\right]}_{\text{does not depend on } \alpha} + \mu_k \left[\alpha(v^\top d_1 + 1_n^\top d_2 + 2c x_j^\top d_1) + \alpha^2 c d_1^\top d_1\right] \\ &\quad - \sum_{i=1}^{2n} \log(-m_i^\top u_j - \alpha m_i^\top d_j)\end{aligned}$$

  Note that

  $$\begin{aligned}\sum_{i=1}^{2n} \log(-m_i^\top u_j - \alpha m_i^\top d_j) &= \sum_{i=1}^{n} \log(-m_i^\top u_j - \alpha m_i^\top d_j) + \sum_{i=n+1}^{2n} \log(-m_i^\top u_j - \alpha m_i^\top d_j) \\ &= \sum_{i=1}^{n} \log(t_j(i) - x_j(i) + \alpha(d_2(i) - d_1(i))) + \sum_{i=1}^{n} \log(t_j(i) + x_j(i) + \alpha(d_2(i) + d_1(i))) \\ &= 1_n^\top \log.(t_j - x_j + \alpha(d_2 - d_1)) + 1_n^\top \log.(t_j + x_j + \alpha(d_2 + d_1))\end{aligned}$$

  where log. is the pointwise logarithm of a vector. Therefore,

  $$\begin{aligned}\phi(\alpha) &= \mu_k \left[v^\top x_j + 1_n^\top t_j + c x_j^\top x_j\right] + \mu_k \left[\alpha(v^\top d_1 + 1_n^\top d_2 + 2c x_j^\top d_1) + \alpha^2 c d_1^\top d_1\right] \\ &\quad - 1_n^\top \log.(t_j - x_j + \alpha(d_2 - d_1)) - 1_n^\top \log.(t_j + x_j + \alpha(d_2 + d_1)) \qquad (7)\end{aligned}$$

  Designating the terms $t_j - x_j + \alpha(d_2 - d_1)$ and $t_j + x_j + \alpha(d_2 + d_1)$ respectively by $\arg\log_1$ and $\arg\log_2$, we can write

  $$\begin{aligned}1_n^\top \log.(t_j - x_j + \alpha(d_2 - d_1)) + 1_n^\top \log.(t_j + x_j + \alpha(d_2 + d_1)) &= \log[\arg\log(1) \times \arg\log(2) \times \\ &\quad \ldots \times \arg\log(n)], \qquad (8)\end{aligned}$$

  where $\arg\log := \arg\log_1 \odot \arg\log_2$, whenever the bit–precision of the processor is accurate enough[3]. Hence, (7) is written as

  $$\begin{aligned}\phi(\alpha) &= \mu_k \left[v^\top x_j + 1_n^\top t_j + c x_j^\top x_j\right] + \mu_k \left[\alpha(v^\top d_1 + 1_n^\top d_2 + 2c x_j^\top d_1) + \alpha^2 c d_1^\top d_1\right] \\ &\quad - \log[\arg\log(1) \times \arg\log(2) \times \ldots \times \arg\log(n)].\end{aligned}$$

[3]In the experiments we did not have any kind of errors due to this calculations.

- Making $\alpha = 0$ in the previous expression,

$$\phi(0) = \mu_k \left[v^\top x_j + 1_n^\top t_j + c x_j^\top x_j \right] - \log[\arg\log(1) \times \arg\log(2) \times \ldots \times \arg\log(n)].$$

- $\dot{\phi}(0) = \nabla f(u_j)^\top d_j = g_1^\top d_1 + g_2^\top d_2$

Now, we will consider the code (and count the number of flops) to calculate $\phi(\alpha)$, $\phi(0)$ and $\dot{\phi}(0)$. We will take into account that the last two can be precomputed (outside) the Armijo's rule loop and have factors used in the calculation of $\phi(\alpha)$.

Code 4 (Calculating $\phi(\alpha)$, $\phi(0)$ and $\dot{\phi}(0)$)**.**
Assumption: we have access to all variables defined in Codes 1, 2 and 3.

- `var1 = mu*(v'*x + onesv'*t + c*x'*x);` *($5n + 3$ flops)*
- `var2 = t-x;` *(n flops)*
- `var3 = t+x;` *(n flops)*

These last two variables have already been calculated, but we calculate them again just for simplicity.

- `arglog_phi0 = var2.*var3;` *(n flops)*
- `phi_0 = var1 - log(arglog_phi0(1)* ... *arglog_phi0(n)));` *($n + 1$ flops + 1 log)*[4]
- `phi_deriv_0 = g1'*d1 + g2'*d2;` *($4n + 1$ flops)*
- `var4 = mu*(v'*d1 + onesv'*d2 + (2*c)*x'*d1);` *($6n + 4$ flops)*
- `var5 = (mu*c)*(d1'*d1);` *($3n + 1$ flops)*
- `diff_d = d2 - d1;` *(n flops)*
- `sum_d = d2 + d1;` *(n flops)*

And inside the Armijo's loop,

- `arglog1 = var2 + alpha*diff_d;` *($2n$ flops)*
- `arglog2 = var3 + alpha*sum_d;` *($2n$ flops)*
- `arglog = arglog1.*arglog2;` *(n flops)*
-
```
phi_alpha = var1 + alpha*var4 + alpha^2*var5
            - log(arglog(1)*...*arglog(n)));
```

($n + 3$ flops + 1 log)

Note that while we assume that u_j is a feasible (interior) point, we must certify that $u_j + \alpha d_j$ is also feasible, otherwise we would get complex numbers when calculating the logs.

With all this into account, the line search algorithm (algorithm 1) is translated into (pseudo) Matlab code as follows.

[4] Note that we only need to calculate $\phi(0)$ in the first iteration. In the other ones, we make $\phi(0) = \phi(\alpha)$.

Code 5 (Line search algorithm).

- *Choose* x_0, t_0, $\epsilon^2 > 0$.
- *Loop: for* $j = 1 : N$
  - *Evaluate* g_1 *and* g_2 *(Code 1)* *(11n + 2 flops)*
  - *Solve the linear system, finding* d_1 *and* d_2 *(Code 2 and 3)* *(37n + 6 flops)*
  - `if -(g1'*d1 + g2'*d2) < epsilon, break;` *(4n + 2 flops)*
  - *Calculate* `var1, var2, var3, phi_deriv_0, var4, var5, diff_d, sum_d` *and* `phi_0` *only if* $j == 1$*; otherwise set* `phi_0 = phi_alpha` *(Code 4)* *(20n + 9 flops + 1 log)*
  - `alpha = 1; beta = 0.5; c1 = 10^(-2);`
  - *Loop: for* $l = 1 : L$ *(Armijo's rule)*
    - *Calculate* `arglog1` *and* `arglog2` *(Code 4)* *(4n flops)*
    - `if min(arglog1) > 0 && min(arglog2) > 0`
      - *Calculate* `phi_alpha` *(Code 4)* *(2n + 3 flops + 1 logs)*
      - `if phi_alpha <= phi_0 + c1*alpha*phi_deriv_0, break;` *(3 flops)*
    - `alpha = beta*alpha;` *(1 flop)*
  - `x = x + alpha*d1 ; t = t + alpha*d2;` *(4n flops)*

The total number of flops is roughly

$$(76n + 19)N + (6n + 7)NL \quad \text{flops}$$

and

$$NL + 1 \quad \text{logs}$$

where

- n: size of each row of the matrix A, usually $n > 10^4$
- N: number of Newton steps, usually $N \leq 30$ (if no warm–start, about 20 in the first iteration and then no more than 10 in the following)
- L: number of steps in the Armijo's rule, usually $L \leq 10$ (no more than 8 in the experiments)

6 The Barrier Method

At the beginning we transformed a problem

$$\min_{\substack{Mu \leq r \\ p^\top u = b}} \quad s^\top u + cu^\top Ru \tag{9}$$

into several

$$\{\min_{p^\top u = b} \mu_k \left(s^\top u + cu^\top Ru\right) - \sum_{i=1}^{2n} \log(-m_i^\top u)\}_{k=0}^{+\infty}. \tag{10}$$

In fact, (10) is equivalent to (9) if μ_k is very high. However, solving each subproblem (10) with a very high μ_k is very difficult. So, the idea of the barrier method is to solve sequentially each subproblem in (10) starting from a small of μ_k and increasing it from iteration to iteration. The key for this approach to be successful is to use the optimal point found in one iteration as a starting point for the next iteration. By doing so, we can decrease a lot the number of iterations to solve each subproblem in (10).

In the experiments, we started with a $\mu = 5 \times 10^{-2}$ and from iteration to iteration we multiplied μ by 50. After 8 iterations of the barrier method, we had found the exact x that solves (2).

If we designate the number of iterations of the barrier method by K, the total number of flops and logs necessary to solve (2) is

$$\begin{aligned} & K[(76n + 19)N + (6n + 7)NL] \\ = & \boxed{6KnNL + \mathcal{O}(KnN) + \mathcal{O}(KNL)} \quad \text{flops} \end{aligned}$$

and

$$\begin{aligned} & K(NL + 1) \\ = & \boxed{KNL + \mathcal{O}(K)} \quad \text{logs.} \end{aligned}$$

7 Experiments

Figure 1 shows the number of millions of floating point operations (MFlops) per second of the implementation of the above algorithms, as a function of the dimensions of the problem n, *i.e.* the dimensions of the variable x. Each point is an average out of 5 experiences, where a, b, v and $c > 0$ are randomly generated in each experiment.

As the algorithm involves calculations of logarithms, we needed to account those calculations into the experience. We did that by previously testing how many flops a log operation takes (about 25 flops).

There was no warm–start for any algorithm, being the initialization of the "first" interior point random. The parameters of the algorithms were:

- `epsilon`$= 10^{-3}$ (stopping criterion of Newton's method);
- `N`$= 20$ (maximum number of Newton's iterations);
- `L`$= 50$ (maximum number of Armijo's iterations);
- `K`$= 8$ (number of iterations of the barrier method);
- $\mu_0 = 0.05$ (initialization of the barrier parameter);
- $\beta = 50$ ($\mu_{k+1} = \beta\mu_k$).

Specifications of the computer where this experience was executed:

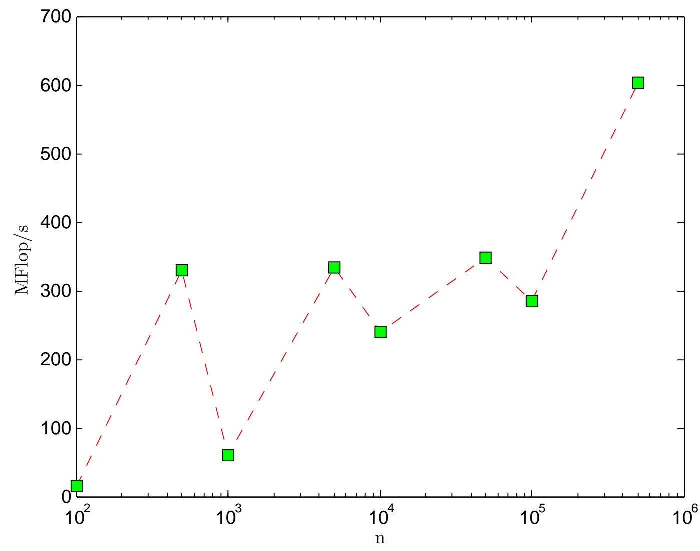

Figure 1: **Update** Mflops per second of the Matlab implementation of the algorithm that solves (2).

Processor Intel(R) Core 2 Quad 2.66 GHz(GenuineIntel)

Cpu 2000 MHz

Cache Size 3072 KB

8 Generalization to the case where each node has more than one row of A

Suppose that instead of just storing just one column of the (measurements) matrix A, each node stores p_i columns. For simplicity, we will designate this number just by p. Then, instead of (2), we have

$$\begin{array}{cc} \min & \|x\|_1 + v^\top x + c\|x\|^2, \\ Ax = b & \\ \text{var: } x \in \mathbb{R}^n & \end{array} \tag{11}$$

where $v, x \in \mathbb{R}^n$, $c \in \mathbb{R}_+$ and $A \in \mathbb{R}^{p\times n}$. By applying the same steps that drove us from (2) to (3), we get

$$\begin{array}{cc} \min & s^\top u + cu^\top Ru, \\ Mu \leq r & \\ Pu = b & \\ \text{var: } u \in \mathbb{R}^{2n} & \end{array} \tag{12}$$

where

$$P = \begin{bmatrix} A & 0_{p\times n} \end{bmatrix} \in \mathbb{R}^{p\times 2n} \qquad s = \begin{bmatrix} v \\ 1_n \end{bmatrix} \qquad u = \begin{bmatrix} x \\ t \end{bmatrix} \qquad r = 0_{2n}$$

$$M = \begin{bmatrix} I_n & -I_n \\ -I_n & -I_n \end{bmatrix} \qquad R = \begin{bmatrix} I_n & 0_{n\times n} \\ 0_{n\times n} & 0_{n\times n} \end{bmatrix},$$

i.e., all the matrices/vectors have the same meaning as in (3), except the matrix P.

The barrier method yields the following sequence of problems

$$\{ \begin{array}{cc} \min & \mu_k \left(s^\top u + cu^\top Ru\right) - \sum_{i=1}^{2n} \log(-m_i^\top u)\}_{k=0}^{+\infty}, \\ Pu = b & \\ \text{var: } u \in \mathbb{R}^{2n} & \end{array} \tag{13}$$

being

$$M = \begin{bmatrix} m_1^\top \\ m_2^\top \\ \vdots \\ m_{2n}^\top \end{bmatrix}.$$

Note that the difference between (4) and (13) is the matrix P. It is straightforward to see that the only modification we have to make in the previous analysis is in the solution of the linear system (5). That linear system becomes, in our case,

$$\begin{bmatrix} \nabla^2 f(u) & P^\top \\ P & 0_{p\times p} \end{bmatrix} \begin{bmatrix} d \\ \lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(u) \\ Pu - b \end{bmatrix}, \tag{14}$$

where λ is now a vector in $\mathbb{R}^p$. Even in solving this linear system, the inversion of the matrix $\tilde{H} := \nabla^2 f(u)$ is still as we did in codes 1 and 2. What basically changes is the Schur complement $\tilde{S}$ of $\tilde{H}$:

$$\begin{aligned} \tilde{S} &= -P\tilde{H}^{-1}P^\top \\ &= -\begin{bmatrix} A & 0_{p\times n} \end{bmatrix} \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} A^\top \\ 0_{n\times p} \end{bmatrix} \\ &= -A\tilde{Q}_{11}A^\top \in \mathbb{R}^{p\times p}. \end{aligned} \tag{15}$$

Since (15) has no particular structure, the computation of its inverse takes $(7/3)p^4$ flops. This is so if we use a Cholesky factorization (see [1, page 670]). In fact, we can decompose $-\tilde{S}$ into a Cholesky product because $-\tilde{S}$ is positive definite.

With this information, the solution of the linear system (14) is now

$$\begin{bmatrix} d \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} \tilde{H}^{-1} + \tilde{H}^{-1}P^\top \tilde{S}^{-1} P \tilde{H}^{-1} & -\tilde{H}^{-1}P^\top \tilde{S}^{-1} \\ -\tilde{S}^{-1}P\tilde{H}^{-1} & \tilde{S}^{-1} \end{bmatrix} \begin{bmatrix} -g \\ c_g \end{bmatrix},$$

where $g := \nabla f(u)$ and $c_g = b - Pu$,

$$= \begin{bmatrix} -\tilde{H}^{-1}g - \tilde{H}^{-1}P^\top \tilde{S}^{-1}P\tilde{H}^{-1}g - \tilde{H}^{-1}P^\top \tilde{S}^{-1}c_g \\ ---------------------- \\ \tilde{S}^{-1}P\tilde{H}^{-1}g + \tilde{S}^{-1}c_g \end{bmatrix}.$$

As we did in page 6, we can analyze each of the individual terms of the solution and seek some simplifications:

- $\tilde{H}^{-1}g = \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} (Q_{11} \odot g_1) + (Q_{12} \odot g_2) \\ (Q_{12} \odot g_1) + (Q_{22} \odot g_2) \end{bmatrix} := \begin{bmatrix} Hg1 \\ Hg2 \end{bmatrix}$,
  where $a \odot b$ is the Hadamard product of two vectors a and b, and $Q_{ij} = \text{diag}(\tilde{Q}_{ij})$.
- $P\tilde{H}^{-1}g = \begin{bmatrix} A & 0_{p\times n} \end{bmatrix} \begin{bmatrix} Hg1 \\ Hg2 \end{bmatrix} = AHg1 := w$
- $\tilde{H}^{-1}P^\top = \begin{bmatrix} \tilde{Q}_{11} & \tilde{Q}_{12} \\ \tilde{Q}_{12} & \tilde{Q}_{22} \end{bmatrix} \begin{bmatrix} A^\top \\ 0_{n\times p} \end{bmatrix} = \begin{bmatrix} \tilde{Q}_{11}A^\top \\ \tilde{Q}_{12}A^\top \end{bmatrix} := \begin{bmatrix} QA1 \\ QA2 \end{bmatrix}$

Simplifying the solution,

$$\begin{bmatrix} d_1 \\ d_2 \\ --- \\ \lambda \end{bmatrix} = \begin{bmatrix} -(Hg1 + QA1 \times \tilde{S}^{-1} \times w + QA1 \times \tilde{S}^{-1} \times c_g) \\ -(Hg2 + QA2 \times \tilde{S}^{-1} \times w + QA2 \times \tilde{S}^{-1} \times c_g) \\ -------------------- \\ \tilde{S}^{-1} \times (w + c_g) \end{bmatrix} = \begin{bmatrix} -(Hg1 + QA1 \times \tilde{S}^{-1} \times (w + c_g)) \\ -(Hg2 + QA2 \times \tilde{S}^{-1} \times (w + c_g)) \\ ----------------- \\ \tilde{S}^{-1} \times (w + c_g) \end{bmatrix}$$

$$= \begin{bmatrix} -(Hg1 + QA1 \times \lambda) \\ -(Hg2 + QA2 \times \lambda) \\ ---------- \\ \tilde{S}^{-1} \times (w + c_g) \end{bmatrix}$$

Code 6 (Solving the linear system (general case) — $(2p^2 + 8p + 10)n + 4p^2 + 3p + (7/3)p^4$ flops)**.**

Assumption: we have access to all variables defined in Codes 1 and 2.

- `S = -A*[Q11.*a1 Q11.*a2 ... Q11.*ap];` *($A = [a_1, \dots, a_p]^\top$)* *($2(n+1)p^2 + np$ flops)*
- `S_inv = inv(S);` *(using Cholesky)* *($(7/3)p^4$ flops)*
- `Hg1 = (Q11.*g1) + (Q12.*g2);` *($3n$ flops)*
- `Hg2 = (Q12.*g1) + (S_h_inv.*g2);` *($3n$ flops)*
- `QA1 = [Q11.*a1 Q11.*a2 ... Q11.*ap];` *(previously calculated)* *(0 flops)*
- `QA2 = [Q12.*a1 Q12.*a2 ... Q12.*ap];` *(np flops)*
- `lambda = S_inv*(A*(Hg1-x) + b);` *($2(n+1)p + p + 2p^2$ flops)*
- `d1 = -(Hg1 + QA1*lambda);` *($2(p+1)n$ flops)*

- `d2 = -(Hg2 + QA2*lambda);` *($2(p+1)n$ flops)*

With this, the total number of flops required to calculate $\nabla f(u)$ and to solve the linear system (14) is

$$
\begin{aligned}
&(11n+2) + (16n+2) + [(2p^2+8p+10)n + 4p^2 + 3p + (7/3)p^4] \\
&\qquad = (2p^2+8p+37)n + 4p^2 + 3p + 4 + (7/3)p^4 \quad \text{flops.}
\end{aligned}
$$

Therefore, the total number of flops needed to solve problem (11) is

$$
\begin{aligned}
&K\{[(2p^2+8p+76)n + 4p^2 + 3p + 23 + (7/3)p^4]N + (6n+7)NL\} \\
&\qquad = \boxed{2Kp^2nN + \tfrac{7}{3}Kp^4N + 6nNLK + \mathcal{O}(Kp^2N)} \quad \text{flops}
\end{aligned}
$$

and

$$
\begin{aligned}
&K(NL+1) \\
&= \boxed{KNL + \mathcal{O}(K)} \quad \text{logs,}
\end{aligned}
$$

where the meaning of the variables is explained in page 11. However, here we have an additional variable, p, representing the number of rows of A in (11).

References

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. Also available at `http://www.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`.

[2] J. Mota, J. Xavier, P. Aguiar, and M. Püschel. Distributed algorithms for basis pursuit. In *2nd International Workshop on Signal Processing with Adaptive Sparse Structured Representations*, Saint-Malo, France, April 2009.