Preconditioning via Diagonal Scaling

Reza Takapoui Hamid Javadi

June 4, 2014

1 Introduction

Interior point methods solve small to medium sized problems to high accuracy in a reasonable amount of time. However, for larger problems as well as stochastic problems, one needs to use first-order methods such as stochastic gradient descent (SGD), the alternating direction method of multipliers (ADMM), and conjugate gradient (CG) in order to attain a modest accuracy in a reasonable number of iterations.

The *condition number* of a matrix A, denoted by $\kappa(A)$, is defined as the ratio of its maximum singular value to its minimum singular value. Both theoretical analysis and practical evidence suggest that the precision and convergence rate of the first-order methods can depend significantly on the condition number of the matrices involved in the problems. As an example, the CG algorithm for solving the linear system $Ax = b$ achieves a faster rate of convergence when the condition number of A is smaller. Hence, it is desirable to decrease the condition number of matrix A by applying a transformation to it; this process is called *preconditioning*.

A special case of preconditioning is called *diagonal scaling*. Here, we are interested in finding diagonal matrices D and E to minimize the condition number of the matrix $A' = DAE$, in order to accelerate first-order methods. For example, in applying CG to solve $Ax = b$, we can solve $A'\tilde{x} = Db$ instead, and recover $x = E^{-1}\tilde{x}$, while taking advantage of the small condition number of A'.

In our numerical experiments and from theoretical analysis, we have concluded that preconditioning can improve the performance of the first-order methods in two different ways. First, it can significantly accelerate the linear algebra operations. For example in [OCPB13], each step of the algorithm involves running CG which can be done remarkably faster if the appropriate preconditioning is applied. The second effect of the preconditioning, which should be distinguished from the first one, is decreasing the number of iterations for achieving desired accuracy by following different intermediate points in ADMM.

In this report, we first discuss heuristics for diagonal scaling. Next, we motivate preconditioning by an example, and then we study preconditioning for a specific splitting form in ADMM called *graph projection splitting*. Finally we examine the performance of our methods by some numerical examples.

2 Matrix equilibration

Let $A \in \mathbf{R}^{m\times n}$ be given. *Matrix equilibration* is a heuristic method to find diagonal matrices $D \in \mathbf{R}^{m\times m}$ and $E \in \mathbf{R}^{n\times n}$ to decrease the condition number of DAE. The basic idea in equilibration is to find D and E such that all columns of DAE have equal ℓ_p norms and all rows of DAE have equal ℓ_p norms. (The parameter $p \geq 1$ can be selected in our algorithm.) We show that this can be formulated as a convex optimization problem. Consider the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i,j} |A_{ij}|^p e^{x_i} e^{y_j} \\ \text{subject to} & \sum_i x_i = 0 \\ & \sum_j y_j = 0, \end{array}$$

with variables $x \in \mathbf{R}^m$ and $y \in \mathbf{R}^n$. The optimality conditions for this problem can be shown to be equivalent to equilibration of DAE for $D = \mathbf{diag}(e^x)$ and $E = \mathbf{diag}(e^y)$. Although matrix equilibration is a convex problem, it is computationally favorable to solve this problem with heuristic iterative methods, rather than using interior point methods. There are several methods for matrix equilibration, here we mention a few famous ones. We refer interested readers to [Bra10] for a thorough discussion on matrix equilibration.

- *Sinkhorn-Knopp equilibration algorithm* was originally designed to convert matrices with nonnegative entries to doubly stochastic matrices [Sin64]. However, with a slight modification, it can be used to perform equilibration in any ℓ_p norm.
- *Ruiz equilibration algorithm* was originally proposed to equilibrate square matrices [Rui01]. Here, we made a small modification to it so that A can be rectangular.
- *Matrix free algorithms* are the methods that obtain information about a matrix only through matrix-vector products. Matrix-free methods are useful when a matrix is represented as an operator with no access to the matrix entries [Bra10].

Algorithm 1 Sinkhorn-Knopp

Initialize $d_1 = \mathbf{1}_m, d_2 = \mathbf{1}_n$.
while $r_1 > \epsilon_1$ or $r_2 > \epsilon_2$ **do**
  $(d_1)_i := (Ad_2)_i^{-1}$.
  $(d_2)_j := (A^T d_1)_j^{-1}$.
  $B := \mathbf{diag}(d_1) A \mathbf{diag}(d_2)$.
  $r_1 = \frac{\max_i \|B_{i:}\|}{\min_i \|B_{i:}\|}, r_2 = \frac{\max_i \|B_{:j}\|}{\min_i \|B_{:j}\|}$.
end while
return $D = \mathbf{diag}(d_1), E = \mathbf{diag}(d_2)$.

Algorithm 2 Ruiz

Initialize $d_1 = \mathbf{1}_m, d_2 = \mathbf{1}_n, B = A$.
while $r_1 > \epsilon_1$ or $r_2 > \epsilon_2$ **do**
  $(d_1)_i := (d_1)_i (\|B_{i:}\|_p)^{-1/2}$.
  $(d_2)_j := (d_2)_j (m/n)^{1/2p} (\|B_{:j}\|_p)^{-1/2}$.
  $B := \mathbf{diag}(d_1) A \mathbf{diag}(d_2)$.
  $r_1 = \frac{\max_i \|B_{i:}\|}{\min_i \|B_{i:}\|}, r_2 = \frac{\max_i \|B_{:j}\|}{\min_i \|B_{:j}\|}$.
end while
return $D = \mathbf{diag}(d_1), E = \mathbf{diag}(d_2)$.

Although these algorithms work extremely well in practice, there is no theoretical guarantee that they decrease the condition number. In fact, in our numerical experiments, we observed that in some cases they might slightly increase the condition number.

3 Example

In this section, we study one simple example to motivate diagonal scaling for ADMM. Consider the consensus problem

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & x = z, \end{array} \tag{1}$$

with variables $x, z \in \mathbf{R}^n$, where $f(x) = (1/2)\|Ax - b\|_2^2$, with $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$, and $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is a convex, closed, and proper function. This includes several popular problems such as lasso, support vector machine, and non-negative least squares. Instead of solving (1) directly, we can use ADMM to solve the equivalent problem

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Fx = Fz, \end{array} \tag{2}$$

where $F \in \mathbf{R}^{n\times n}$ is invertible. The augmented Lagrangian for this problem will be

$$L_F(x, z, y) = f(x) + g(z) + (Fy)^T(x - z) + (1/2)\|x - z\|_{F^2}^2. \tag{3}$$

Defining Fy as the new dual variable, we see that (3) is identical to the augmented Lagrangian for (1) except that a different norm is used to augment the Lagrangian. Notice that taking $F = \sqrt{\rho}\mathbf{I}$ is equivalent to scaling the step size in ADMM by ρ. Hence, choosing the best matrix F (that results in the fastest convergence) is at least as difficult as finding the optimal step size ρ in [BPC+10]. Now, a question of our interest here is the following: by expanding the class of matrices F from multiples of identity to diagonal matrices, how much can we accelerate the ADMM? To answer this question we note that it can be shown that ADMM achieves linear convergence in this case and (an upper bound for) the rate of convergence is $\kappa(F(A^TA)^{-1}F^T)$. Hence choosing F such that $\kappa(F(A^TA)^{-1}F^T)$ is smaller can hopefully result in faster convergence for ADMM.

To illustrate this effect, we used ADMM to solve the lasso problem with and without presence of matrix F. We generated a random matrix A with size $m = 7500$ and $n = 2500$, and used ADMM to solve (2) in three different cases with the stopping tolerance set to 10^{-4}. First, we performed ADMM directly on (1) without presence of F. It took 7416 iterations for the algorithm to converge. For the second case, we took $D = \sqrt{\rho^*}\mathbf{I}$, where $\rho^* = \sqrt{\lambda_{\min}(A)\lambda_{\max}(A)}$ which maximizes the upper bound on the convergence rate of the algorithm ($\rho^* = 7.2$ in this example). In this case, it took 1019 iterations until the algorithm converged. Finally we used D which equilibrates $(A^TA)^{-1}$. In the last case it took only 8 iterations for the algorithm to converge. This shows the important effect of preconditioning on the convergence rate of the ADMM for solving the lasso problem.

4 Graph projection splitting

Consider the following problem which is in canonical *graph form* [PB13]

$$\begin{array}{ll} \text{minimize} & f(y) + g(x) \\ \text{subject to} & Ax = y. \end{array} \tag{4}$$

Graph projection splitting [PB13] is a form of ADMM to solve this problem serially. The essential idea is to define $z = (x, y)$ and $\phi(z) = f(y) + g(x)$, and solve

$$\begin{array}{ll} \text{minimize} & \phi(z) + I_{Ax'=b'}(z') \\ \text{subject to} & z = z' \end{array}$$

instead. Here we notice that (4) is equivalent to

$$\begin{array}{ll} \text{minimize} & f(D^{-1}\tilde{y}) + g(E\tilde{x}) \\ \text{subject to} & \tilde{y} = DAE\tilde{x}, \end{array}$$

where $\tilde{x} = E^{-1}x$ and $\tilde{y} = Dy$ for diagonal $D \in \mathbf{R}^{m\times m}$ and $E \in \mathbf{R}^{n\times n}$ with positive diagonal entires. Using graph projection splitting for this problem is equivalent to running ADMM on

$$\begin{array}{ll} \text{minimize} & \phi(z) + I_{Ax'=y'}(z') \\ \text{subject to} & \left[\begin{array}{cc} E^{-1} & 0 \\ 0 & D \end{array}\right] z = \left[\begin{array}{cc} E^{-1} & 0 \\ 0 & D \end{array}\right] z'. \end{array}$$

The second step of ADMM is to project onto the *graph subspace* $G = \{(\tilde{x}, \tilde{y}) | DAE\tilde{x} = \tilde{y}\}$ which is a linear transformation defined by

$$\Pi_G(\tilde{x}, \tilde{y}) = \left[\begin{array}{cc} I & (DAE)^T \\ DAE & -I \end{array}\right]^{-1} \left[\begin{array}{cc} I & (DAE)^T \\ 0 & 0 \end{array}\right] \left[\begin{array}{c} \tilde{x} \\ \tilde{y} \end{array}\right].$$

In the next section, we will discuss how we can choose D and E to accelerate ADMM in graph projection splitting form. Notice that for fixed DAE (even when D and E are changing in each step), the projection matrix remains the same, and hence no additional factorization caching is required in evaluating the projection operator. We will return to this fact in the next section.

5 Parameter selection

In this section we propose methods for choosing matrices D, E to speed up the graph projection splitting algorithm. We argue that following the steps below can help us to achieve a desired accuracy in smaller number of iterations.

- Using the algorithms discussed in §2, we choose $\hat{D}$, $\hat{E}$ such that $\hat{D}A\hat{E}$ is equilibrated. This will usually result in reducing the condition number of A. As a result, the singular values of $\hat{D}A\hat{E}$ will be closer to each other and intuitively matrix A becomes more isotropic. We notice that defining $D = \alpha\hat{D}$ and $E = \beta\hat{E}$ for any $\alpha, \beta \in \mathbf{R}_{++}$, the matrix DAE is equilibrated. We have two degrees of freedom left: $\alpha\beta$ can be chosen to scale DAE and β/α can play the role of step size in ADMM.

- The product $\alpha\beta$ is chosen to set $\|DAE\|$. Let $\gamma = \|\hat{D}A\hat{E}\|$, then we will have $\|DAE\| = \alpha\beta\gamma$. Since DAE is likely to have a small condition number, this means that the norms of y and x are related with a factor close to $\|DAE\|$. This should be chosen appropriately for every problem. In most examples, setting $\|DAE\| = 1$ gives us reasonably good results. Of course, this depends on the functions f and g. We have inspected this effect in the next section.

- Note that we can change β/α at each step while DAE is constant. Doing this procedure can help us to change ρ adaptively to balance the norms of dual and primal residuals. More importantly, this procedure is costless and does not require additional factorization caching in direct methods, also in indirect methods warm start can be used.

6 Numerical results

In this section we inspect the effect of parameters explained in §5 on the speed of graph projection splitting algorithm for two different problems.

6.1 Lasso

Consider the lasso problem

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \lambda\|x\|_1,$$

where $A \in \mathbf{R}^{m\times n}$ and $b \in \mathbf{R}^m$ and $\lambda \in \mathbf{R}$ are the problem data and $x \in \mathbf{R}^n$ is the decision variable. We can write this problem in form (4) with $f(y) = \|y - b\|_2^2$ and $g(x) = \lambda\|x\|_1$. We generate instances of the lasso problem with $m = 750$ and $n = 250$. We plot the average number of iterations required versus values of $\beta\alpha$ (scaling) and β/α (step size ρ) in Figure 1. The relative tolerance was set to 10^{-4} as the stopping criterion.

6.2 Linear program (LP)

Consider the problem

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \preceq b \end{array}$$

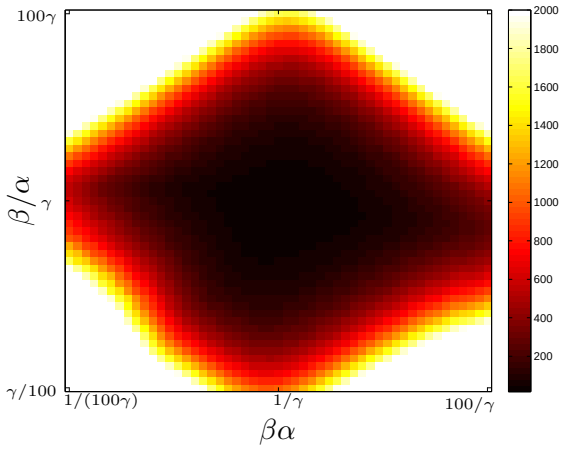
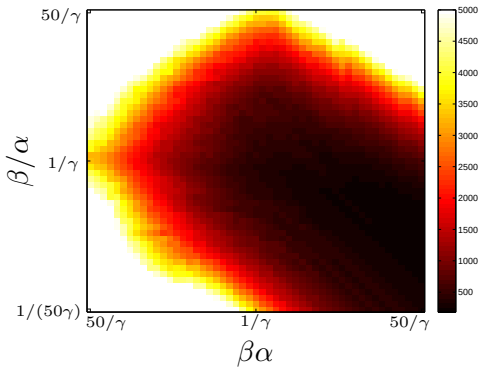
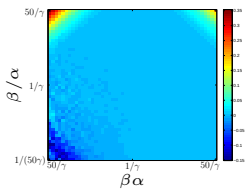


Figure 1: Average number of iterations for graph projection splitting on the lasso problem

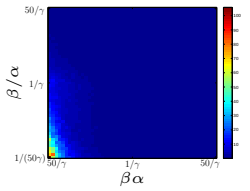
where $c \in \mathbf{R}^n$ and $b \in \mathbf{R}^m$ and $A \in \mathbf{R}^{m \times n}$ are the problem data and $x \in \mathbf{R}^n$ is the decision variable. This can be written in form (4) by $f(y) = I_{\{y \leq b\}}(y)$ and $g(x) = c^T x$. We generate several instances of the problem with $m = 750$ and $n = 250$. We plot the number of iterations required, relative objective suboptimality and relative constraint error versus values of $\beta\alpha$ (scaling) and β/α (step size ρ). The relative tolerance was set to 10^{-4} as the stopping criterion.



(a) Average number of iterations



(b) Relative suboptimality



(c) Relative constraint error

Figure 2: Graph projection splitting on LP with inequality constraint

We see that after matrix equilibration, the remaining two degrees of freedom need to be set appropriately to achieve a desired accuracy in smaller number of iterations. Setting α and β such that $\|DAE\| = 1$ and choosing appropriate step size afterwards usually results in a fast convergence.

References

[BPC+10] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 2010.

[Bra10] A. M. Bradley. Algorithms for the equilibration of matrices and their application to limited-memory quasi-newton methods. *Ph.D. thesis, Institute for Computational and Mathematical Engineering, Stanford University*, 2010.

[OCPB13] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Operator splitting for conic optimization via homogeneous self-dual embedding. *arXiv preprint arXiv:1312.3039*, 2013.

[PB13] N. Parikh and S. Boyd. Graph projection block splitting for distributed optimization. *Mathematical Programming Computation*, 2013.

[Rui01] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. *Rutherford Appleton Lab., Oxon, UK, Tech. Rep. RAL-TR-2001-034*, 2001.

[Sin64] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.