Multi-block ADMM Methods for Linear & Semidefinite Programming

Nico Chaves, Mike Schoenhals, Junjie (Jason) Zhu

March 16, 2017

1 Introduction

The alternating direction method of multipliers (ADMM) is a first-order optimization method. It divides the original problem into smaller pieces, resulting in an easier update procedure. In this project, we applied variants of ADMM to the following optimization problems.
Linear Programming Primal:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{LP}$$

Linear Programming Dual:

$$\begin{aligned} \max_{\mathbf{y},\mathbf{s}} \quad & \mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & A^T\mathbf{y} + \mathbf{s} = \mathbf{c}, \\ & \mathbf{s} \geq \mathbf{0}. \end{aligned} \tag{LD}$$

SDP Primal:

$$\begin{aligned} \min_{X} \quad & C \bullet X \\ \text{s.t.} \quad & \mathcal{A}X = \mathbf{b} \\ & X \succeq \mathbf{0} \end{aligned} \tag{SDP}$$

SDP Dual:

$$\begin{aligned} \max_{\mathbf{y},s} \quad & \mathbf{b}^T\mathbf{y} \\ \text{s.t.} \quad & \mathcal{A}^T\mathbf{y} + S = C \\ & S \succeq \mathbf{0} \end{aligned} \tag{SDD}$$

In this report, we first re-formulate the problems to derive closed form updates. For (**??**) and (**??**), we also derive their updates when applying block splitting and the interior-point approach. Furthermore, we examine the effect of preconditioning on solving (**??**) and (**??**). We implemented all of the algorithms in MATLAB and ran experiments to test how these methods perform on different simulated problems.

Preconditioning

In this report, we demonstrate that preconditioning can help ADMM solve linear programs faster. After preconditioning the system, we solve the modified system

$$A'x = \mathbf{b}' \tag{1}$$

where

Standard Preconditioning

To apply "standard" preconditioning to a linear programming problem, one first computes

$$A' = (AA^T)^{-1/2}A, \qquad \mathbf{b}' = (AA^T)^{-1/2}\mathbf{b}, \tag{2}$$

and then substitutes A' and $\mathbf{b}'$ for A and $\mathbf{b}$ in the original problems (**??**) and (**??**). Clearly, this re-formulation does not change the feasible regions for (**??**) or (**??**).

Cholesky Preconditioning

Block-Splitting ADMM

As we will show in Section 2, ADMM requires computation of a potentially large matrix inverse. Since matrix inversion is approximately cubic in complexity, this computation may become the bottleneck for sufficiently large problems. By splitting the decision variables into blocks and updating the blocks sequentially, one can reduce the runtime by computing multiple small matrix inverses instead of a single large matrix inverse.

In this report, we analyze the effect of block splitting on ADMM. With two specific examples, we demonstrate that block splitting with 3 or more blocks may not converge to the optimal solution. However, we demonstrate through experiments that randomly permuting the update order of the blocks at each iteration resolves this issue.

We also demonstrate that block splitting and preconditioning work well when used together. However, the preconditioning approach we consider in this report requires a matrix inverse computation. Therefore, one still may suffer from a slow matrix inversion if one uses preconditioning. In our future work, we plan to study inverse-free preconditioning methods such as Cholesky preconditioning.

2 Basic ADMM for LP

In this section, we provide re-formulations to enable us to easily solve the LP for the primal and dual problems. We derive the basic algorithms which include update steps and convergence criteria. (For all the algorithms we introduce, we use random feasible initialization for the variables, but better initialization techniques could be explored if one has more information about the structure of the problems.) Furthermore, the majority of the basic update equations derived here will be extended to the multi-block ADMM updates in Section 4.

Primal Re-formulation

We re-formulate the primal LP problem as:

$$\begin{array}{ll} \text{minimize}_{\mathbf{x}_1,\mathbf{x}_2} & \mathbf{c}^T\mathbf{x}_1 \\ \text{subject to} & A\mathbf{x}_1 = \mathbf{b} \\ & \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0} \\ & \mathbf{x}_2 \geq \mathbf{0} \end{array} \tag{OPT3}$$

and consider the split Lagrangian function:

$$L^P(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{s}) = \mathbf{c}^T\mathbf{x}_1 - \mathbf{y}^T\left(A\mathbf{x}_1 - \mathbf{b}\right) - \mathbf{s}^T\left(\mathbf{x}_1 - \mathbf{x}_2\right) + \frac{\beta}{2}\left(\|A\mathbf{x}_1 - \mathbf{b}\|^2 + \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right).$$

Primal Update

To obtain the updates for the primal variables in ADMM, we must compute the gradient of these variables and set the gradient to zero. Here we update $\mathbf{x}_1$ and $\mathbf{x}_2$ sequentially using the update function for each respectively:

$$\mathbf{x}_1^{k+1} = \arg\min_{\mathbf{x}_1^k} L^P(\mathbf{x}_1^k, \mathbf{x}_2^k, \mathbf{y}^k, \mathbf{s}^k), \tag{3}$$

$$\mathbf{x}_2^{k+1} = \arg\min_{\mathbf{x}_2^k \geq 0} L^P(\mathbf{x}_1^{k+1}, \mathbf{x}_2^k, \mathbf{y}^k, \mathbf{s}^k). \tag{4}$$

To solve for $\mathbf{x}_1$, we set the gradient to zero:

$$\nabla_{\mathbf{x}_1} L^P(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{s}) = \mathbf{c} - A^T\mathbf{y} - \mathbf{s} + \beta\left(A^T\left(A\mathbf{x}_1 - \mathbf{b}\right) + \left(\mathbf{x}_1 - \mathbf{x}_2\right)\right) = \mathbf{0}$$

and we obtain the update step for $\mathbf{x}_1$:

$$\mathbf{x}_1^{k+1} = \left(A^TA + I\right)^{-1}\left(\frac{1}{\beta}A^T\mathbf{y}^k + \frac{1}{\beta}\mathbf{s}^k - \frac{1}{\beta}\mathbf{c} + A^T\mathbf{b}^k + \mathbf{x}_2^k\right) \tag{5}$$

Similarly, we solve

$$\nabla_{\mathbf{x}_2} L^P = \mathbf{s} + \beta\left(\mathbf{x}_2 - \mathbf{x}_1\right) = \mathbf{0}$$

to obtain the update for $\mathbf{x}_2$. Since the elements of $\mathbf{x}_2$ are separable, we can update each one independently as follows:

$$x_{2,j}^{k+1} = \max\left\{x_{1,j}^k - \frac{1}{\beta}s_j^k, 0\right\} \text{ for } j = 1, .., n. \tag{6}$$

Next, we update the multipliers $\mathbf{y}$ and $\mathbf{s}$ using

$$\mathbf{y}^{k+1} = \mathbf{y}^k + \beta(A\mathbf{x}_1^{k+1} - \mathbf{b}) \tag{7}$$

$$\mathbf{s}^{k+1} = \mathbf{s}^k - \beta(\mathbf{x}_1^{k+1} - \mathbf{x}_2^{k+1}) \tag{8}$$

These two updates guarantee that $\left(\mathbf{x}_1^{k+1}, \mathbf{x}_2^{k+1}, \mathbf{y}^{k+1}, \mathbf{s}^{k+1}\right)$ is dual feasible, i.e.,

$$\nabla_{\mathbf{x}_1, \mathbf{x}_2} L^P(\mathbf{x}_1^k, \mathbf{x}_2^k, \mathbf{y}^k, \mathbf{s}^k) = \mathbf{0}.$$

We terminate when the solution is primal feasible, i.e., $A\mathbf{x}_1^{k+1} - \mathbf{b} \approx \mathbf{0}$ and $\mathbf{x}_1^{k+1} - \mathbf{x}_2^{k+1} \approx \mathbf{0}$.

Dual Re-formulation

We convert the problem of the dual to a minimization problem to apply ADMM:

$$\begin{aligned} \text{minimize}_{\mathbf{y},\mathbf{s}} \quad & -\mathbf{b}^T\mathbf{y} \\ \text{subject to} \quad & A^T\mathbf{y} + \mathbf{s} = \mathbf{c}, \\ & \mathbf{s} \geq \mathbf{0}. \end{aligned} \tag{OPT4}$$

The dual LP problem does not require additional re-formulation because the non-zero variable $\mathbf{s}$ is already separable, which can be observed via the dual Lagrangian function:

$$L^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{b}^T\mathbf{y} - \mathbf{x}^T\left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) + \frac{\beta}{2}\left\|A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right\|^2.$$

Note that $\mathbf{x}$ would be non-positive since we changed maximization to minimization and we would need to take the negative of $\mathbf{x}$ of the final solution as the solution to the original LP.

Dual Update

Similar to the primal updates, we update $\mathbf{y}$ and $\mathbf{s}$ according to

$$\mathbf{y}^{k+1} = \arg\min_{\mathbf{y}} L^d(\mathbf{y}^k, \mathbf{s}^k, \mathbf{x}^k), \tag{9}$$

$$\mathbf{s}^{k+1} = \arg\min_{\mathbf{s}\geq 0} L^d(\mathbf{y}^{k+1}, \mathbf{s}^k, \mathbf{x}^k). \tag{10}$$

Setting the gradients with respect to $\mathbf{y}$ and $\mathbf{s}$ to zero, i.e.,

$$\nabla_{\mathbf{y}} L^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{b} - A\mathbf{x} + \beta A\left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) = \mathbf{0}, \tag{11}$$

$$\nabla_{\mathbf{s}} L^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{x} + \beta\left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) = \mathbf{0}, \tag{12}$$

we have the updates for $\mathbf{y}$ and $\mathbf{s}$:

$$\mathbf{y}^{k+1} = \left(AA^T\right)^{-1}\left(\frac{1}{\beta}\left(A\mathbf{x}^k + \mathbf{b}\right) - A\mathbf{s}^k + A\mathbf{c}\right), \tag{13}$$

$$s_j^{k+1} = \max\left\{\frac{1}{\beta}x_j^k - (A^T\mathbf{y}^{k+1})_j + c_j, 0\right\} \text{ for } j = 1, .., n. \tag{14}$$

We update $\mathbf{x}$ using

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \beta\left(A^T\mathbf{y}^{k+1} + \mathbf{s}^{k+1} - \mathbf{c}\right). \tag{15}$$

We terminate and declare optimality when $A\mathbf{x}^{k+1} + \mathbf{b} \approx \mathbf{0}$.

3 Interior-Point ADMM for LP

The interior-point method is alternative way to handle the non-negative constraints in LP problems by including the log-barrier function in the objective function. In this section, we derive the update formulas, whereas the remaining details of the algorithms follow from the basic ADMM in Section 2. We include the interior-point method here as the derivation from the re-formulation is straightforward. However, we did not extensively test the convergence behavior of the interior-point methods because the focus of the project is multi-block ADMM and we found empirically that interior-point ADMM yielded highly similar results to the basic ADMM.

Primal LP Updates

Here we modify the formulation in (**??**) to include the log barrier function:

$$\begin{aligned} \text{minimize}_{\mathbf{x}_1,\mathbf{x}_2} \quad & \mathbf{c}^T\mathbf{x}_1 + \mu\sum_j \ln(x_{2,j}) \\ \text{subject to} \quad & A\mathbf{x}_1 = \mathbf{b}, \\ & \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0}, \\ & \mathbf{x}_2 > \mathbf{0}, \end{aligned} \tag{OPT5}$$

where the Lagrangian function is given by

$$L_\mu^p(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}, \mathbf{s}) = \mathbf{c}^T\mathbf{x}_1 - \mu\sum_j \ln(x_{2,j}) - \mathbf{y}^T(A\mathbf{x}_1 - \mathbf{b}) - \mathbf{s}^T(\mathbf{x}_1 - \mathbf{x}_2) + \frac{\beta}{2}\left(\|A\mathbf{x}_1 - \mathbf{b}\|^2 + \|\mathbf{x}_1 - \mathbf{x}_2\|^2\right).$$

Setting the gradient of this to zero yields the same update for $\mathbf{x}_1$ as in (**??**). However, we need to modify the update for $\mathbf{x}_2$ as now we have:

$$s_j - \frac{\mu}{x_{2,j}} + \beta(x_{2,j} - x_{1,j}) = 0 \tag{16}$$

which yields

$$x_{2,j} = \frac{1}{2\beta}\left(\beta x_{1,j} - s_j + \sqrt{\beta^2 x_{1,j}^2 - 2\beta s_j x_{1,j} + s_j^2 + 4\beta\mu}\right), \text{ for } j = 1, .., n \tag{17}$$

The update equations for $\mathbf{y}$ and $\mathbf{s}$ remain the same as in (**??**) and (**??**). The update equation for $\mathbf{x}$ remains the same as in (**??**). At the end of each iteration, we update $\mu^{k+1} = \gamma\mu^k$ for some constant $\gamma < 1$, and the termination condition remains the same as the non-interior-point method.

Dual LP Updates

We can use the previous formulation in (**??**) using the barrier function:

$$\begin{aligned} \text{minmize}_{\mathbf{y},\mathbf{s}} \quad & -\mathbf{b}^T\mathbf{y} + \mu\sum_j \ln(s_j) \\ \text{subject to} \quad & A^T\mathbf{y} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} > \mathbf{0}. \end{aligned} \tag{OPT6}$$

where the Lagrangian function is given by

$$L_\mu^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{b}^T\mathbf{y} - \mu\sum_j \ln(s_j) - \mathbf{x}^T\left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) + \frac{\beta}{2}\left\|A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right\|^2.$$

Setting the gradient of this to zero yields the same update for $\mathbf{y}$ as in (**??**) However, we need to modify the update for $\mathbf{s}$ as now we have:

$$-\frac{\mu}{s_j} + x_j + \beta\left(s_j - (\mathbf{c} - (A^T\mathbf{y}))_j\right) = 0 \tag{18}$$

which yields

$$s_j = \frac{1}{2\beta}\left(\beta(\mathbf{c} - (A^T\mathbf{y}))_j + x_j + \sqrt{\beta^2(\mathbf{c} - (A^T\mathbf{y}))_j^2 - 2\beta(\mathbf{c} - (A^T\mathbf{y}))_j x_j + x_j^2 + 4\beta\mu}\right), \text{ for } j = 1, .., n \tag{19}$$

The update equation for $\mathbf{x}$ remains the same as in (**??**). As in the primal interior-point algorithm, we update $\mu^{k+1} = \gamma\mu^k$ for some constant $\gamma < 1$ at the end of each iteration, and the termination condition remains the same as the non-interior-point method.

4 Multi-Block ADMM

As mentioned previously, splitting the problem into blocks may reduce computation time by avoiding a large matrix inversion. However, when the set of variables are separable, it may not be necessary to apply block updates on them because at the end of each iteration, these separable variables would take identical values no matter how they are grouped. For separable variables, the most efficient way would be updating each variable separately (or in parallel). Thus, we only need to consider the cases where we split $\mathbf{x_1}$ in the primal problems (i.e., (**??**) and (**??**)), or to split $\mathbf{y}$ in the dual problems (i.e., (**??**) and (**??**)).

Primal with Block Splitting (B=2 Blocks)

To help make our derivations concrete, we first consider splitting the problem into $B = 2$ blocks of equal size as follows:

$$\mathbf{x}_1 = \begin{bmatrix}\mathbf{x}_{1,1}\\ \mathbf{x}_{1,2}\end{bmatrix}, \quad A = \begin{bmatrix}A_1 & A_2\end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix}\mathbf{c}_1\\ \mathbf{c}_2\end{bmatrix}$$

Then the Lagrangian can be expressed as:

$$\begin{aligned} L^P(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}) &= \mathbf{c}_1^T\mathbf{x}_{1,1} + \mathbf{c}_2^T\mathbf{x}_{1,2} - \mathbf{y}^T\left(A_1\mathbf{x}_{1,1} + A_2\mathbf{x}_{1,2} - \mathbf{b}\right) - \mathbf{s}_1^T\left(\mathbf{x}_{1,1} - \mathbf{x}_{2,1}\right) - \mathbf{s}_2^T\left(\mathbf{x}_{1,2} - \mathbf{x}_{2,2}\right) \\ &\quad + \frac{\beta}{2}\left(\left\|A_1\mathbf{x}_{1,1} + A_2\mathbf{x}_{1,2} - \mathbf{b}\right\|^2 + \left\|\mathbf{x}_{1,1} - \mathbf{x}_{2,1}\right\|^2 + \left\|\mathbf{x}_{1,2} - \mathbf{x}_{2,2}\right\|^2\right) \end{aligned}$$

Taking the gradient with respect to the 1st block of $\mathbf{x}_1$:

$$\nabla_{\mathbf{x}_{1,1}} L^P(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}) = \mathbf{c}_1 - A_1^T\mathbf{y} - \mathbf{s}_1 + \beta\left(A_1^T\left(A_1\mathbf{x}_{1,1} + A_2\mathbf{x}_{1,2} - \mathbf{b}\right) + \left(\mathbf{x}_{1,1} - \mathbf{x}_{2,1}\right)\right)$$

Setting the gradient to 0, we obtain the update for $\mathbf{x}_{1,1}$:

$$\mathbf{x}_{1,1}^{k+1} = \left(A_1^TA_1 + I\right)^{-1}\left(\frac{1}{\beta}A_1^T\mathbf{y} + \frac{1}{\beta}\mathbf{s}_1 - \frac{1}{\beta}\mathbf{c}_1 + A_1^T\mathbf{b} + \mathbf{x}_{2,1}^k - A_1^TA_2\mathbf{x}_{1,2}^k\right)$$

By symmetry, the update step for the 2nd block of $\mathbf{x}_1$ is:

$$\mathbf{x}_{1,2}^{k+1} = \left(A_2^TA_2 + I\right)^{-1}\left(\frac{1}{\beta}A_2^T\mathbf{y} + \frac{1}{\beta}\mathbf{s}_2 - \frac{1}{\beta}\mathbf{c}_2 + A_2^T\mathbf{b} + \mathbf{x}_{2,2}^k - A_2^TA_1\mathbf{x}_{1,1}^{k+1}\right)$$

Note that here we use $\mathbf{x}_{1,1}^{k+1}$ to update $\mathbf{x}_{1,2}$. Our results show that when $B > 2$ blocks, one may need to randomly permute the update order. As a result, on some iterations, we may first update $\mathbf{x}_{1,2}$, then use the new value to update $\mathbf{x}_{1,1}$.

Primal Block Splitting (For a General Number of Blocks)

We can easily extend the above result to a general number of blocks B. Let U denote the set of blocks which have already been updated at the current iteration. To update block i of $\mathbf{x}_1$, we compute:

$$\mathbf{x}_{1,i}^{k+1} = \left(A_i^TA_i + I\right)^{-1}\left(\frac{1}{\beta}A_i^T\mathbf{y} + \frac{1}{\beta}\mathbf{s}_i - \frac{1}{\beta}\mathbf{c}_i + A_i^T\mathbf{b} + \mathbf{x}_{2,i}^k - \sum_{j\neq i, j\in U} A_i^TA_j\mathbf{x}_{1,j}^{k+1} - \sum_{j\neq i, j\notin U} A_i^TA_j\mathbf{x}_{1,j}^k\right)$$

where A_i refers to the i^{th} block of columns of A.

Dual ADMM With Block Splitting (B=2 Blocks)

Now, we derive the corresponding update for the dual ADMM with block splitting. Again, we first consider the $B = 2$ case and split the problem into 2 blocks of equal size:

$$\mathbf{y} = \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix}, \quad A^T = \begin{bmatrix}A_1^T & A_2^T\end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix}$$

Then the Lagrangian can be expressed as:

$$L^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{b}_1^T\mathbf{y}_1 - \mathbf{b}_2^T\mathbf{y}_2 - \mathbf{x}^T\left(A_1^T\mathbf{y}_1 + A_2^T\mathbf{y}_2 + \mathbf{s} - \mathbf{c}\right) + \frac{\beta}{2}\left\|A_1^T\mathbf{y}_1 + A_2^T\mathbf{y}_2 + \mathbf{s} - \mathbf{c}\right\|^2$$

Differentiating with respect to $\mathbf{y}_1$:

$$\nabla_{\mathbf{y}_1} L^d(\mathbf{y}, \mathbf{s}, \mathbf{x}) = -\mathbf{b}_1 - A_1\mathbf{x} + \beta A_1\left(A_1^T\mathbf{y}_1 + A_2^T\mathbf{y}_2 + \mathbf{s} - \mathbf{c}\right)$$

Setting the gradient to 0, we obtain the update for $\mathbf{y}_1$:

$$\begin{aligned} A_1\left(A_1^T\mathbf{y}_1^{k+1} + A_2^T\mathbf{y}_2^k + \mathbf{s} - \mathbf{c}\right) &= \frac{1}{\beta}\left(A_1\mathbf{x} + \mathbf{b}_1\right) \\ A_1A_1^T\mathbf{y}_1^{k+1} &= \frac{1}{\beta}\left(A_1\mathbf{x} + \mathbf{b}_1\right) - A_1\left(A_2^T\mathbf{y}_2^k + \mathbf{s} - \mathbf{c}\right) \\ \mathbf{y}_1^{k+1} &= \left(A_1A_1^T\right)^{-1}\left(\frac{1}{\beta}\left(A_1\mathbf{x} + \mathbf{b}_1\right) - A_1\left(A_2^T\mathbf{y}_2^k + \mathbf{s} - \mathbf{c}\right)\right) \end{aligned}$$

By symmetry, the update for $\mathbf{y}_2$ is:

$$\mathbf{y}_2^{k+1} = \left(A_2A_2^T\right)^{-1}\left(\frac{1}{\beta}\left(A_2\mathbf{x} + \mathbf{b}_2\right) - A_2\left(A_1^T\mathbf{y}_1^{k+1} + \mathbf{s} - \mathbf{c}\right)\right)$$

assuming that we update $\mathbf{y}_2$ after $\mathbf{y}_1$. As in the primal case, we may need to randomly permute the update order at each iteration.

Dual Block Splitting (For a General Number of Blocks)

We can easily extend the above result to a general number of blocks. Let U denote the set of blocks which have already been updated at the current iteration. To update block i of $\mathbf{y}$:

$$\mathbf{y}_i^{k+1} = \left(A_iA_i^T\right)^{-1}\left(\frac{1}{\beta}\left(A_i\mathbf{x} + \mathbf{b}_i\right) - A_i\left(\mathbf{s} - \mathbf{c}\right) - \sum_{j\neq i, j\in U} A_iA_j^T\mathbf{y}_j^{k+1} - \sum_{j\neq i, j\notin U} A_iA_j^T\mathbf{y}_j^k\right)$$

Using Preconditioning for Dual Block Splitting

Here, we show that block-splitting yields the same result as non-block-splitting if preconditioning is already applied to the dual problem.

A sufficient condition for the variables to be separable is when the coefficient of these variables in the gradient function is the identify matrix. Interestingly, we show that when preconditioning is applied to the dual LP problem (whether or not implemented with the interior point method), the variables $\mathbf{y}$ become separable. To see this, consider the update equation (**??**), where the coefficient of $\mathbf{y}$ is AA^T. When preconditioning is applied, we have

$$A'(A')^T = (AA^T)^{-1/2}AA^T(AA^T)^{-1/2} = (AA^T)^{-1/2}(AA^T)^{1/2}(AA^T)^{1/2}(AA^T)^{-1/2} = I.$$

Therefore, the variables in $\mathbf{y}$ become separable and block-splitting produces the same result as non-block-splitting at the end of each iteration when preconditioning is applied. We'll observe this effect in the Experiments section, which shows that the number of iterations required by the dual is independent of the number of blocks when preconditioning is applied.

5 Experiments and Results

We evaluated the performance of the different ADMM approaches by randomly simulating LP problems and recording the number of iterations needed to converge. In particular, we simulated LP problems given by (**??**) by randomly selecting $A \in \mathbb{R}^{m \times n}$ and $c \in \mathbb{R}^n$. We only considered feasible problems. To ensure feasibility, we randomly selected a feasible point $\mathbf{x}_{\text{feas}} \geq 0$ and let $\mathbf{b} = A\mathbf{x}_{\text{feas}}$. We considered a solution to be optimal when $|A\mathbf{x} - \mathbf{b}| < 10^{-3}$ (for the primal formulation) or $|A\mathbf{x} + \mathbf{b}| < 10^{-3}$ (for the dual formulation). For interior point algorithms, we use $\gamma = 0.99$, i.e. at each iteration, we update the parameter μ using $\mu^{k+1} = 0.99 \times \mu^k$. Each set of experiments is described in more detail below.

Preconditioning with No Block-Splitting

First, we studied how preconditioning affects the convergence rate of the basic primal and dual ADMM algorithms in (**??**) and (**??**). For simplicity, we did not apply block splitting. We evaluated the two algorithms over 10 random problems of size 50×300.

We evaluated how sensitive these ADMM LP algorithms are to the β parameter in the augmented Lagrangian. For each algorithm, we recorded the number of iterations required to converge as a function of β. The results are shown in Figure **??**. We observe that all of the ADMM approaches are relatively insensitive to β. For the rest of our experiments, we fixed $\beta = 0.9$. Choosing a different value of β should not have much effect on the rest of our results.

Additionally, we notice that the basic primal ADMM implementation converges in fewer iterations than the basic dual ADMM, but when preconditioning is applied (on the same problems), the dual converges in fewer iterations than the primal.

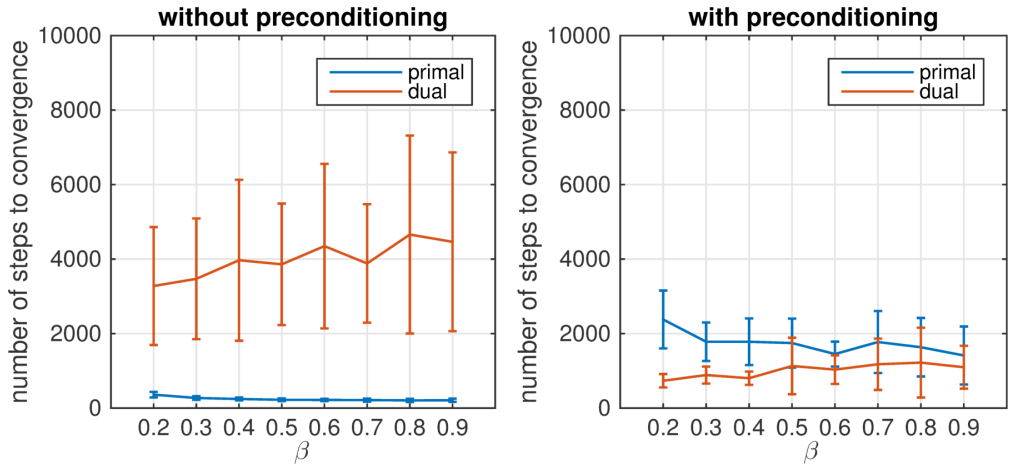

Figure 1: Results on 10 random LP problem of size 50×300.

Baseline Block-splitting (No Random Permutation, No Preconditioning)

To establish a baseline, we evaluated the ADMM solvers using sequential updating (i.e. we did not randomly permute the block update order) and without using preconditioning. The results of this experiment are shown in Figure **??**. In the figure, B represents the number of blocks ($B = 1$ indicates that no splitting was performed). Note that when $B > 2$, the problem does not necessarily converge. As we will see in the following experiments, we can improve performance by introducing random permutations or preconditioning.

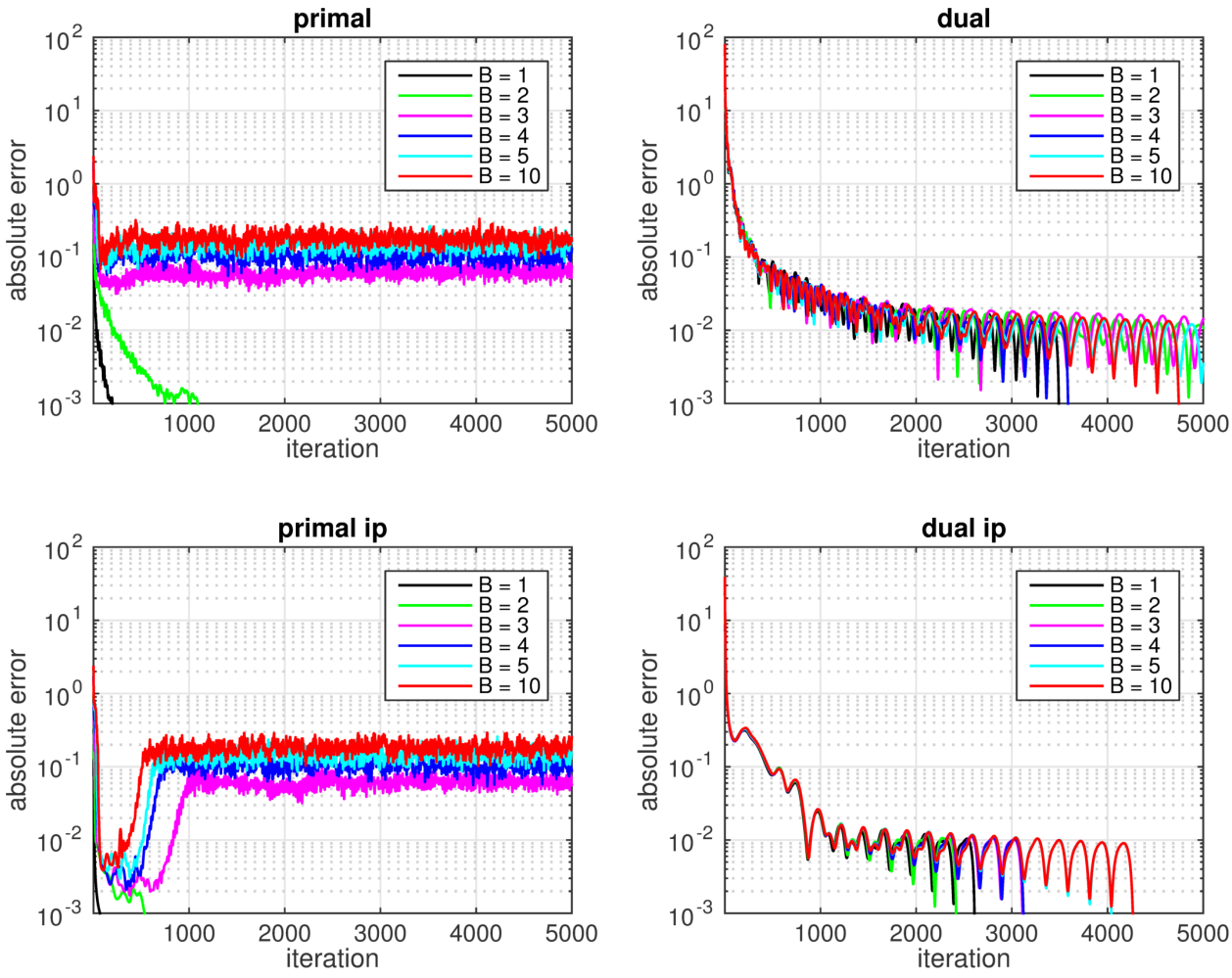

Figure 2: Results on a random LP problem of size 50×300. No preconditioning and no random permutation.

Block-splitting with Random Permutation (No Preconditioning)

In this set of experiments, we randomly permuted the update order of the block variables at each iteration. The results are shown in Figure **??**. When using random permutation, the primal now converges for all block sizes. Note that splitting the primal into more blocks requires more iterations. However, performing block splitting may still result in lower overall runtime because of the faster inverse computations.

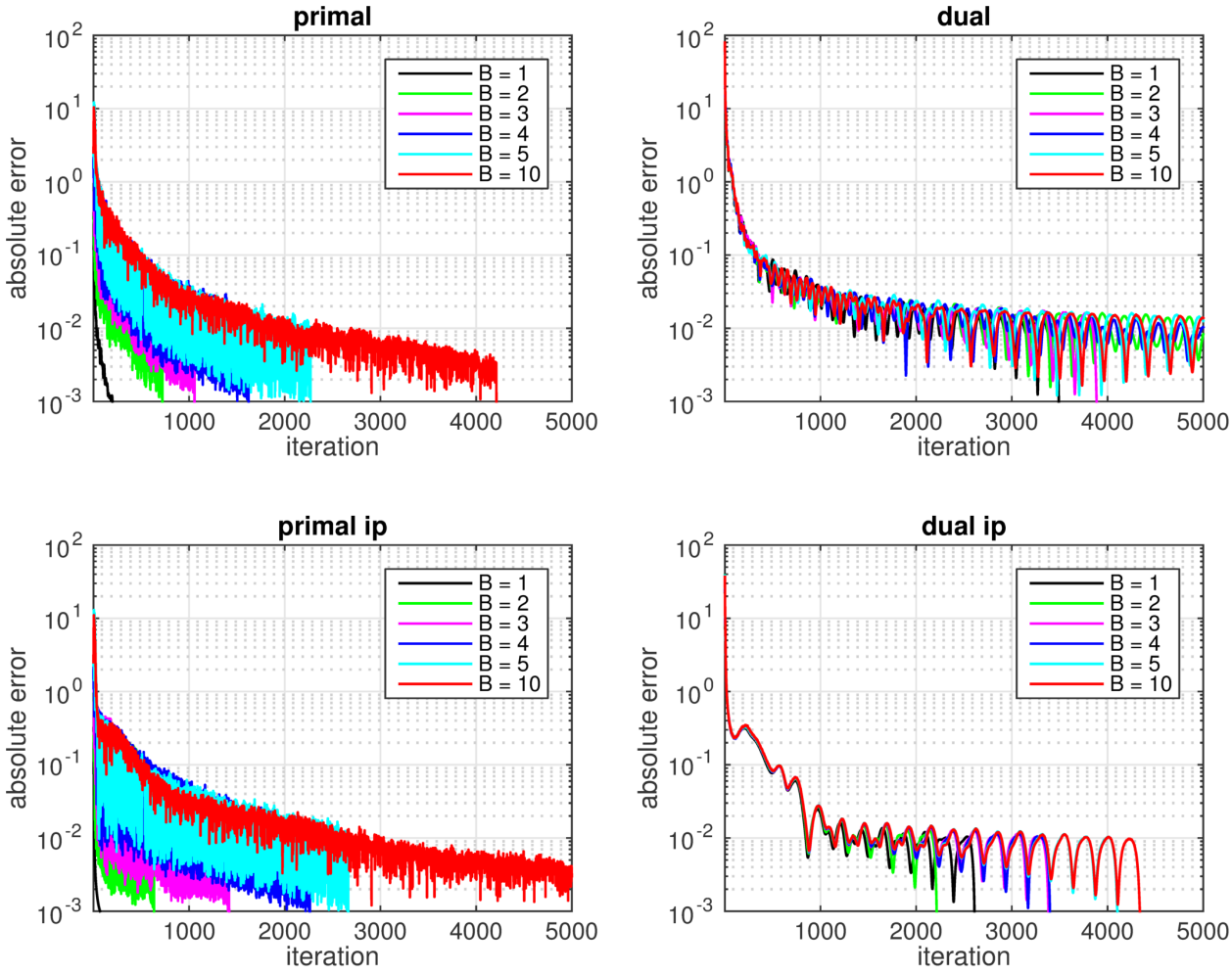

Figure 3: Results on a random LP problem of size 50×300. Only random permutation and no preconditioning.

Block-splitting with Preconditioning (No Random Permutation)

In this set of experiments, we applied preconditioning to each problem and used sequential block updates (no random permutation). The results are shown in Figure **??**. Note that the number of blocks has no effect on the number of iterations required for either dual algorithm. This verifies our claim at the end of Section 3 that preconditioning applied to the dual causes $\mathbf{y}$ to be separable. Interestingly, the number of blocks has very little effect on the number of iterations for the primal.

These results suggest that one should prefer to use the maximum number of blocks possible (n for the primal and m for the dual) when using preconditioning. In this case, the matrices that need to be inverted in the decision variable updates become scalars, so the updates can be computed rapidly.

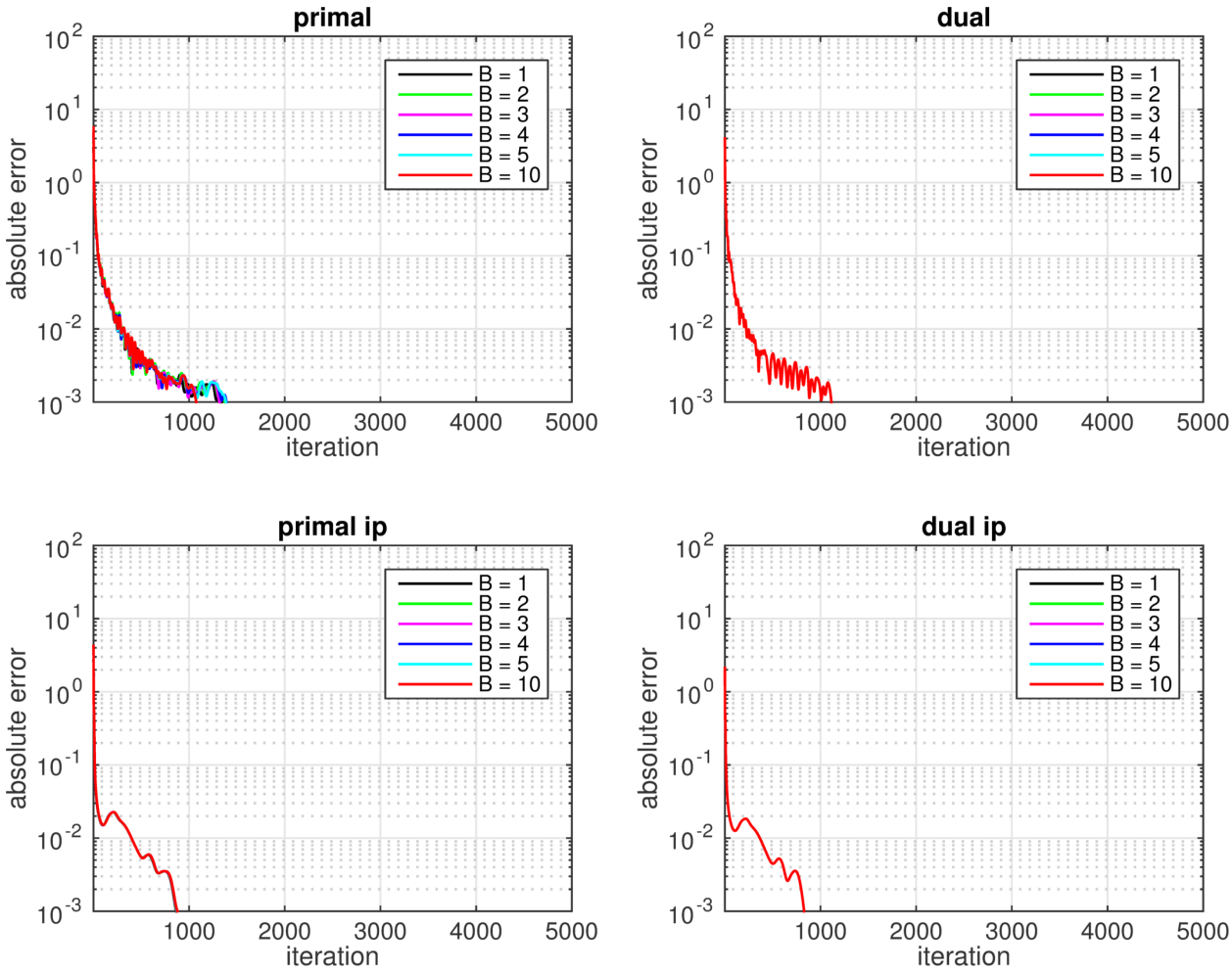

Figure 4: Results on a random LP problem of size 50×300. Only preconditioning and no random permutation.

Analysis of Non-Converging Primal

In Figure **??**, we observed that the absolute error oscillates for $B > 2$ when random permutation and preconditioning are not applied. One may ask whether this a result of the decision variable cycling around the optimal solution but never converging. To address this question, we considered the results shown in Figure **??** for $B = 2, 3, 10$.

To visualize the cycling behavior of $\mathbf{x}$, we found the component of $\mathbf{x}$ with the highest variance over the last 1000 iterations of the $B = 3$ case. Figure **??** below shows the value of this variable at each iteration for each block size $B = 2, 3, 10$. Note that the right plot is simply a zoomed in version of the left plot. Further note that the horizontal black line at 0 represents the value of this variable at the optimal solution, which is approximately 0.

We observe that the variable value indeed oscillates around the optimal value over time, but the oscillation is not symmetric. We also observe that the variable undergoes the largest swings for the $B = 10$ case, even though this variable was selected based upon having a high variance in the $B = 3$ case. This suggests that introducing more blocks may result in larger swings.

Based on the intuition that $\mathbf{x}$ may be cycling "around" the optimal solution, we computed the mean value of $\mathbf{x}$ over the last 1000 iterations and computed the corresponding objective value. However, the objective value only improved very slightly, and it was still far from the optimal objective value.

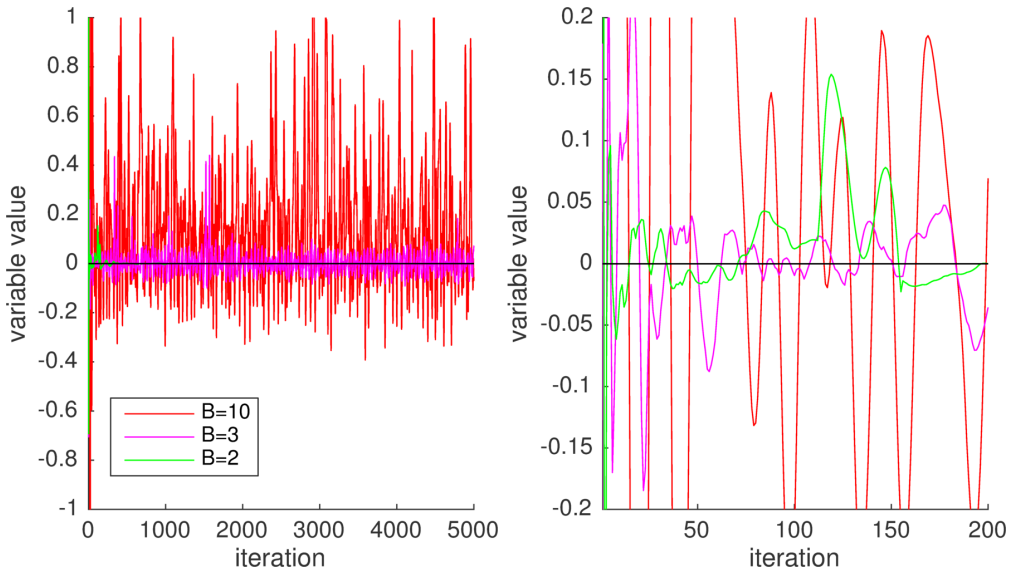

Figure 5: Variable Value over Time

Structured Problems

In this set of experiments, we evaluated the performance of each ADMM algorithm when the matrix A is composed of block structures. Figure **??** visualizes A as a heatmap. This type of matrix structure is common in practice. Often, groups of related constraints only involve a subset of the decision variables, resulting in the "staircase" block structure in Figure **??**. Furthermore, there are often "coupling" constraints involving many decision variables, which is reflected in the last 10 constraints of A. It is natural to study whether performing block splitting along A's block structures will improve the performance of ADMM. Note that $m = 100$, $n = 200$ for this set of experiments.

We reran each of the previous experiments on primal ADMM using the structured matrix A. Instead of performing arbitrary block splitting, we set $B = 10$ and split the matrix A along its block structures. The results of each experiment are shown in Figures **??**-**??**.

In Figure **??**, the problem diverges when applying block splitting for both the primal and the interior point primal formulations. This issue is not necessarily specific to structured matrices. We've observed that

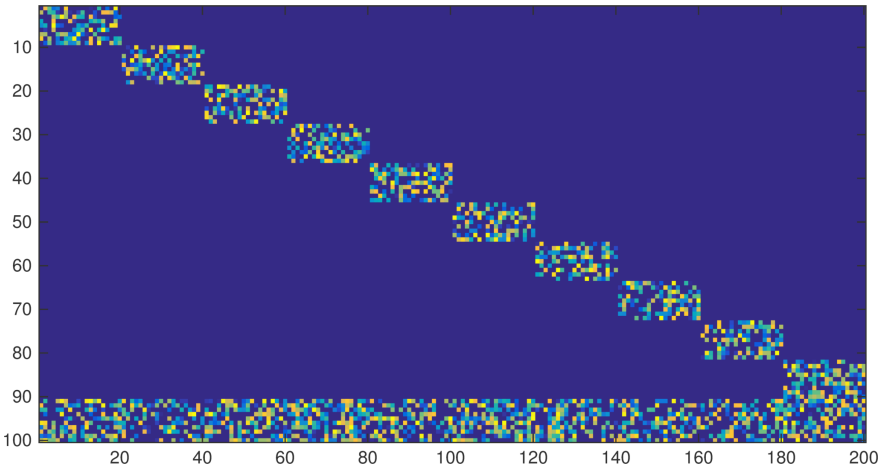

Figure 6: Heatmap of Structured Matrix

ADMM with block splitting (for $B > 2$) will either fail to converge to the optimal solution as in Figure **??** or diverge as in Figure **??**.

As in the non-structured case, we can resolve this issue by randomly permuting the block update order or applying preconditioning. The results for each approach are shown in Figures **??** and **??**. In both cases, the problem converges in approximately the same number of iterations as the single block case. The results are particularly striking for the preconditioning approach. These results suggest that performing optimal block splitting along a structured matrix may improve the performance of ADMM. In our future work, we plan to study whether this holds true in general for structured matrices.

6 Conclusions and Future Work

We evaluated the performance of several different versions of ADMM for linear programming problems. Our experiments show that one can split the problem into blocks without significantly increasing the number of iterations required to find an optimal solution. However, one must be sure to apply preconditioning to the problem or use a randomly permuted update order. Since block splitting allows one to avoid a large matrix inversion, our results suggest that these techniques may reduce the overall runtime of ADMM algorithms. We also found that block splitting seems especially well suited to problems with an inherent block structure, which commonly arise in practice.

In our future work, we intend to run similar experiments on more problems to verify that the results are robust. We also intend to test each approach on problems with various sizes. It will be interesting to see how each approach scales as problems grow large. We plan to compare the runtime (not just the number of iterations) of each algorithm, accounting for the time to take large matrix inversions. This will enable us to determine whether avoiding the large matrix inverse actually leads to a speedup. Finally, we intend to experiment with inverse-free preconditioning methods, such as Cholesky preconditioning, to avoid the matrix inversion involved in our current preconditioning approach. Indeed, we have shown cases where LP problems cannot be solved correctly without preconditioning or random-permutation on the block updates, and the results we present here on preconditioning make inverse-free preconditioning methods promising for multi-block ADMM.

7 Acknowledgements

We'd like to thank Professor Yinyu Ye for his mentorship and guidance throughout the project.

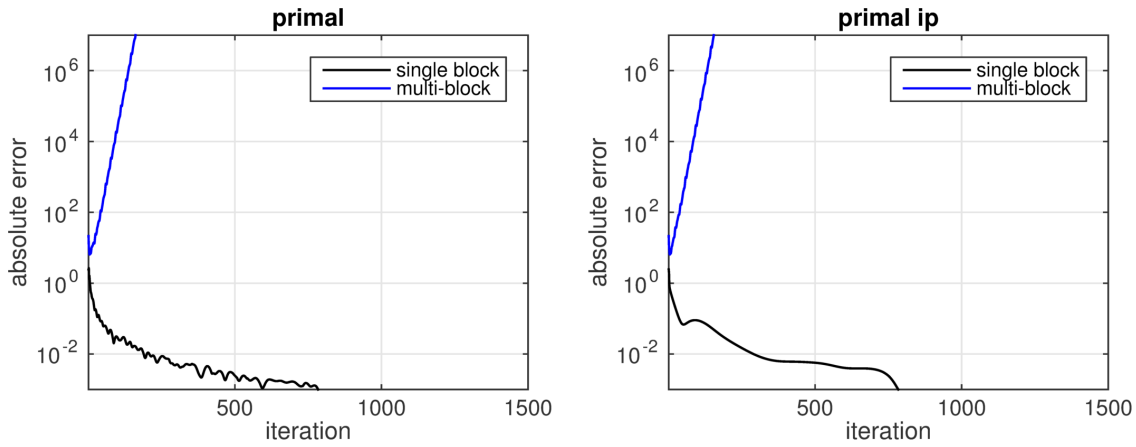

Figure 7: Structured problem. No preconditioning and no random permutation.

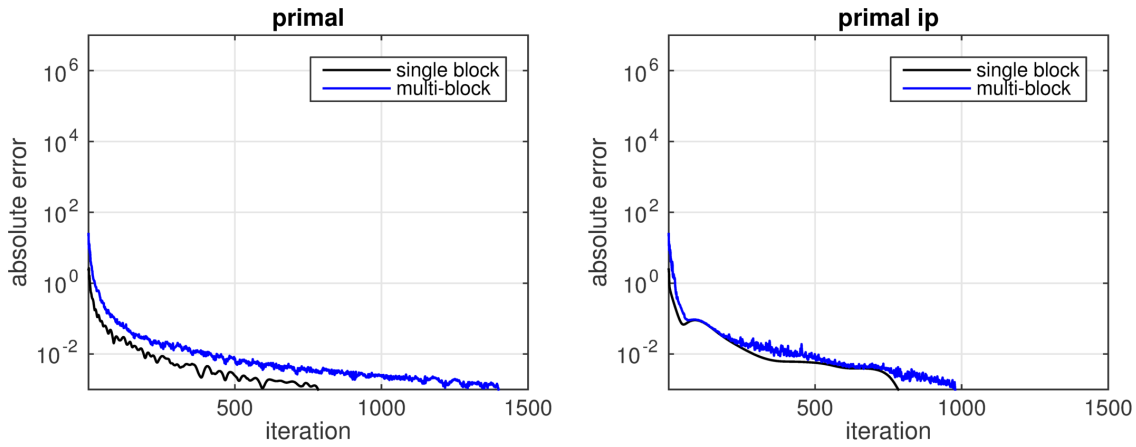

Figure 8: Structured problem. Only random permutation and no preconditioning.

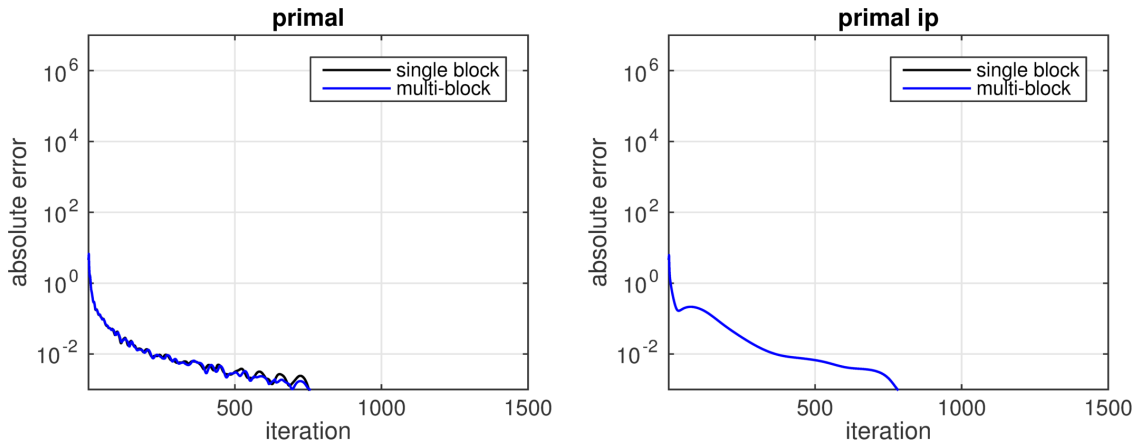

Figure 9: Structured problem. Only preconditioning and no random permutation