The Alternating Direction Method of Multipliers for Linear Programming

Jonathan Tuck

December 10, 2017

Abstract

In this paper, we derive the Alternating Direction Method of Multipliers (ADMM) algorithm for the specific case of linear programming. In addition to this derivation, we also implement this algorithm for the dual of the linear programming problem. We formulate the ADMM solution in an interior-point context, and analyze its performance to the implementation on the classical linear programming problem, and to an open-source solver. Lastly, we compare ADMM when the input data are not preconditioned to when the input data are preconditioned, and analyze the results.

Contents

1 Introduction

Consider the generic, primal linear program [BV04, LY15]

$$\begin{array}{ll} \text{minimize}_x & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array} \tag{1}$$

and its corresponding dual

$$\begin{array}{ll} \text{maximize}_{y,s} & b^T y \\ \text{subject to} & A^T y + s = c \\ & s \geq 0. \end{array} \tag{2}$$

Here, $A \in \mathbf{R}^{m \times n}, b \in \mathbf{R}^m$, and $c \in \mathbf{R}^n$ are the problem data, $x \in \mathbf{R}^n$ is the primal optimization variable, and $y \in \mathbf{R}^m, s \in \mathbf{R}^n$ are the dual optimization variables corresponding to the objective and the equality constraints of the primal problem (1), respectively. Problems (1) and (2) will henceforth be referred to as the *primal problem* and the *dual problem*, respectively.

2 ADMM for the Primal

Let us define the *augmented Lagrangian* [PB14] of the primal problem (1), $L^p : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ as

$$L^p(x, y) = c^T x - y^T (Ax - b) + (\beta/2)\|Ax - b\|_2^2. \tag{3}$$

Typically, x and y represent the primal and dual variables, respectively, and $\beta > 0$ is a penalty parameter. In addition, if there are more than one (say, k) primal decision variables, $x_1, \ldots, x_k$, then we can extend the augmented Lagrangian to incorporate these variables. In such a case, we denote the augmented Lagrangian by $L^p(x_1, \ldots, x_k, y)$.

Now, let us first convert the primal problem (1) into an equivalent problem, what we refer to as *ADMM standard form*:

$$\begin{array}{ll} \text{minimize}_{x_1, x_2} & c^T x_1 \\ \text{subject to} & Ax_1 = b \\ & x_1 - x_2 = 0 \\ & x_2 \geq 0. \end{array} \tag{4}$$

Note that we now have an additional constraint, which in turn means that we have a new dual variable, s, in this problem's corresponding augmented Lagrangian. The augmented Lagrangian for the problem (4) is then

$$L^p(x_1, x_2, y, s) = c^T x_1 - y^T (Ax_1 - b) - s^T (x_1 - x_2) + (\beta/2)\left(\|Ax_1 - b\|_2^2 + \|x_1 - x_2\|_2^2\right). \tag{5}$$

Then, given starting points $x_1^0, x_2^0 \geq 0$, and initial starting points for the dual variables (y^0, s^0), the ADMM solves the problem (4) by the following update scheme:

$$\begin{aligned} x_1^{k+1} &= \operatorname*{argmin}_{x_1} L^p(x_1, x_2^k, y^k, s^k) \\ x_2^{k+1} &= \operatorname*{argmin}_{x_2 \geq 0} L^p(x_1^{k+1}, x_2, y^k, s^k) \\ y^{k+1} &= y^k - \beta(Ax_1^{k+1} - b) \\ s^{k+1} &= s^k - \beta(x_1^{k+1} - x_2^{k+1}). \end{aligned} \tag{6}$$

This update scheme is advantageous because it converts solving a constrained optimization problem into solving a series of unconstrained optimization problems, which are typically much easier to solve than the original constrained problem. In face, in all cases that we analyze, all update steps have closed form solutions.

Update on x_1. The update on x_1^{k+1} can be further simplified into a closed-form update by noting that

$$\nabla_{x_1} L^p(x_1, x_2^k, y^k, s^k) = 0.$$

From this, we immediately see that

$$c - A^T y^k - s^k + \beta A^T(Ax_1 - b) + (x_1 - x_2^k) = 0,$$

which in turn gives us the update

$$x_1^{k+1} = (\beta A^T A + I)^{-1}(A^T y - c + s^k + \beta A^T b + 2x_2^k). \tag{7}$$

Update on x_2. The updates on x_2^{k+1} must be derived differently, due to the constraint that $x_2^{k+1} \geq 0$. Although the update for x_2^{k+1} involves solving a quadratic program over the nonnegative cone, a closed form exists. In fact, it can be shown that the update is [PB14]

$$x_2^{k+1} = \max\{x_1 - s/\beta, 0\}, \tag{8}$$

where max is taken elementwise. The interpretation for this update is natural: we are projecting the solution of $\nabla_{x_2} L^p(x_1^{k+1}, x_2, y^k, s^k) = 0$ onto the nonnegative cone.

3 ADMM for the dual

ADMM can, in a similar way as in the primal problem, be implemented for the dual problem. We start, like in the primal case, by noting that the augmented Lagrangian for the dual problem, $L^d : \mathbf{R}^m \times \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$, is

$$L^d(y, s, x) = -b^T y - x^T(A^T y + s - c) + (\beta/2)\|A^T y + s - c\|_2^2. \tag{9}$$

We update y, then s, then x, iteratively until a specified tolerance has been reached. That is, we repeat the following steps until convergence:

$$\begin{aligned} y^{k+1} &= \operatorname*{argmin}_{y} L^d(y, s^k, x^k) \\ s^{k+1} &= \operatorname*{argmin}_{s \geq 0} L^d(y^{k+1}, s, x^k) \\ x^{k+1} &= x^k - \beta(A^T y^{k+1} + s^{k+1} - c). \end{aligned} \tag{10}$$

These updates are relatively easy to handle; in fact, there exists closed form expressions for all of these updates (the update on x is already in closed form.)

Update on y. The update on y^{k+1} can be further simplified into a closed-form update by noting that

$$\nabla_y L^d(y, s^k, x^k) = 0.$$

From this, we immediately see that

$$-b - Ax^k + \beta A(A^T y + s^k - c),$$

which in turn gives us the update

$$y^{k+1} = (1/\beta)(AA^T)^{-1}(b + Ax^k + \beta A(c - s^k)). \tag{11}$$

Update on s. The update on s is the projection of $\nabla_s L^d(y^{k+1}, s, x^k) = 0$ onto the non-negative cone. Thus, the update becomes

$$s^{k+1} = \max\{(1/\beta)x^k - A^T y^{k+1} + c, 0\}, \tag{12}$$

where max is taken elementwise.

4 Interior-point ADMM

Consider the generic, primal linear program with a logarithmic barrier function

$$\begin{array}{ll} \text{minimize}_x & c^T x - \mu \sum_j \log x_j \\ \text{subject to} & Ax = b \\ & x > 0, \end{array} \tag{13}$$

or its dual,

$$\begin{array}{ll} \text{maximize}_x & b^T y + \mu \sum_j \log s_j \\ \text{subject to} & A^T y + s = c \\ & s > 0. \end{array} \tag{14}$$

As $\mu \to 0$, the solutions to problems (13) and (14) approach the solutions to the primal and dual problems, respectively [Ren01].

4.1 Interior-point ADMM for the primal

We solve the problem (13) using the identical ADMM framework for solving the primal problem. Like in §2, we convert the problem (13) to ADMM standard form:

$$\begin{array}{ll} \text{minimize}_x & c^T x_1 - \mu \sum_j \log (x_2)_j \\ \text{subject to} & Ax_1 = b \\ & x_1 - x_2 = 0 \\ & x_1 > 0, \end{array} \tag{15}$$

which in turn gives us an augmented Lagrangian, with dual variables y and s,

$$L_\mu^p(x_1, x_2, y, s) = c^T x_1 - \mu \sum_j \log(x_2)_j - y^T(Ax_1 - b) - s^T(x_1 - x_2) + (\beta/2)(\|Ax_1 - b\|_2^2 + \|x_1 - x_2\|_2^2). \tag{16}$$

Note that since $x_1 = x_2$ in the constraints, we can interchangably switch x_1 and x_2 so to make the updates for each step easier to compute. We can now solve the problem (15) using ADMM. The updates are identical as in the problem (6), so all that is left to find are the appropriate closed-form updates for x_1 and x_2.

x_1 update. To find the closed-form update for x_1, notice that the only difference between this problem's augmented Lagrangian and the augmented Lagrangian of (5) is the term $-\mu \sum_j \log(x_2)_j$, whose gradient with respect to x_1 vanishes. Thus, the x_1 update for this problem is identical to the x_1 update in (6).

x_2 update. As for the x_2 update, we take $\nabla_{x_2} L_\mu^p(x_1, x_2, y, s)$ elementwise and equate it to zero:

$$\begin{aligned}\left\{\nabla_{x_2} L_\mu^p(x_1^{k+1}, x_2, y^k, s^k)\right\}_j &= \frac{-\mu}{(x_2)_j} + s_j + \beta((x_1)_j - (x_2)_j) \\ &= 0,\end{aligned}$$

which in turn leads to the update

$$(x_2)_j^{k+1} = (1/2\beta)\left(s_j + \beta(x_1)_j + \sqrt{(s_j + \beta(x_1)_j)^2 + 4\mu\beta}\right), \quad j = 1, \ldots, n. \tag{17}$$

Outer iteration. In order to achieve a stable solution, we gradually reduce μ as an outer iteration. That is, we start with some $\mu = \mu^0$, and solve the problem (13) using ADMM. Then, we decrease μ by some constant $\gamma \in (0, 1)$ and repeat the process. This process allows us, in a stable fashion, to have $\mu \to 0$, ultimately leading us to the solution of the primal problem (1).

4.2 Interior-point ADMM for the dual

The methodology for deriving ADMM for the problem (14) is nearly identical to its primal counterpart. The augmented Lagrangian of the problem (14) is

$$L_\mu^d(y, s, x) = -b^T y - \mu \sum_j \log(s_j) - x^T(A^T y + s - c) + (\beta/2)\|A^T y + s - c\|_2^2. \tag{18}$$

Like in §3, we update y, then s, then x, iteratively until a specified tolerance has been reached. That is, we repeat the steps outlined in (10) until convergence, replacing L^d with L_μ^d. The update on the x variable is trivial and already in closed form, and we identically utilize an outer iteration on μ as we did for solving (13).

Update on y. The update on the y variable can be further simplified into a closed-form update by noting that

$$\nabla_y L_\mu^d(y, s^k, x^k) = 0,$$

from which we immediately see that

$$-b - Ax^k + \beta A(A^T y + s^k - c) = 0.$$

In turn, this gives us the update

$$y^{k+1} = (1/\beta)(AA^T)^{-1}(b + Ax^k + \beta A(c - s^k)). \tag{19}$$

Notice that this update is identical to the y update in ADMM for the original dual problem (2).

Update on s. In a similar fashion to the x_2 update for the primal barrier problem (13), we take $\nabla_s L_\mu^d(y^{k+1}, s, x^k)$ and equate it to zero:

$$\begin{aligned}\{\nabla_s L_\mu^d(y^{k+1}, s, x^k)\}_j &= \frac{-\mu}{s_j} - x_j^k + \beta((A^T y^{k+1})_j + s_j - c_j) \\ &= 0,\end{aligned}$$

which in turn leads to the update

$$s_j^{k+1} = (1/2)(c_j + x_j^k/\beta - (A^T y^{k+1})_j) + (1/2\beta)\sqrt{\beta^2((A^T y^{k+1})_j - x_j^k/\beta - c_j)^2 + 4\beta\mu}. \tag{20}$$

5 Examples

For these examples, the ground-truth optimal values for the problems (1) and (2) were obtained using CVX [GB14], an open-source convex optimization package.

Problem instance. For the experiments that follow, we let $n = 50$ and $m = 5$, and randomly generate problem instances by randomly generating $A \in \mathbf{R}^{m\times n}, b \in \mathbf{R}^m$, and $c \in \mathbf{R}^n$. In addition, we let $\beta = 1$ and $\gamma = 0.75$ unless stated otherwise.

5.1 Implementation of ADMM on the primal and dual problems

Figure 1 below shows the residual of ADMM versus the iteration when solving problems (1) and (2), respectively. From the results, ADMM seems to work well, albeit at a relatively slow rate: on average, it takes about 1000 iterations to get approximately 10^{-5} away from the optimal solution. However, for modest accuracy (*i.e.*, 10^{-2} away from the optimal solution,) we see that it emperically takes approximately 200 iterations.

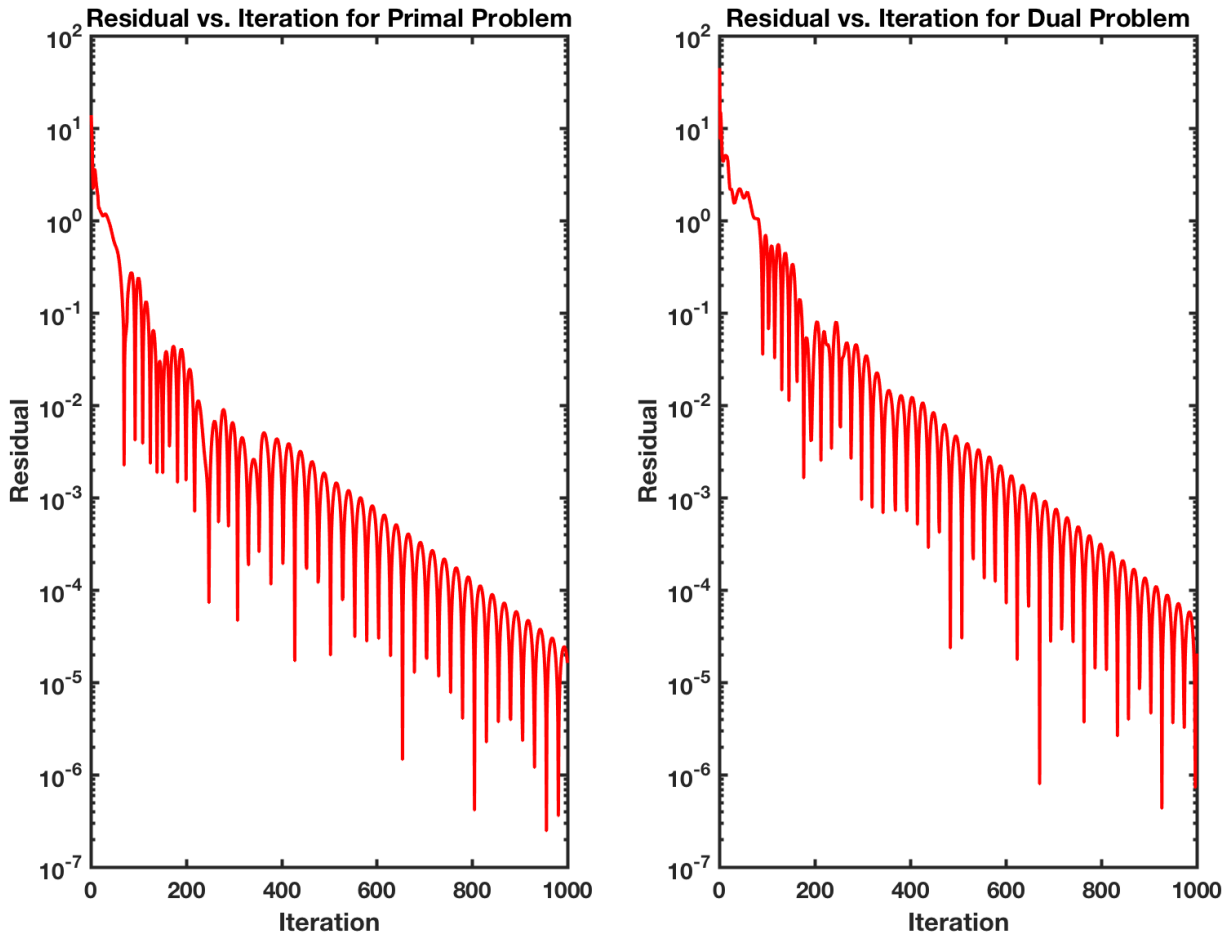

Figure 1: Graph of residual versus iteration when solving problems (1) and (2), respectively from left to right, using ADMM.

5.2 Implementation of interior-point ADMM

Figure 2 below shows the residual of solving the problems (13) and (14), respectively. From this graph, it is clear that solving (13) and (14) using ADMM is a valid and viable choice, as we see that the solution after 1000 iterations is approximately 10^{-7} away from the optimal solution. In addition, we implement interior-point ADMM using an Outer-Iteration process, with fixed γ. Comparing Figure 2 to Figure 1, we find that using ADMM to solve the barrier problems (13) and (14) achieves a slightly higher, but still noticable, degree of optimality than simply using ADMM to solve problems (1) and (2), respectively.

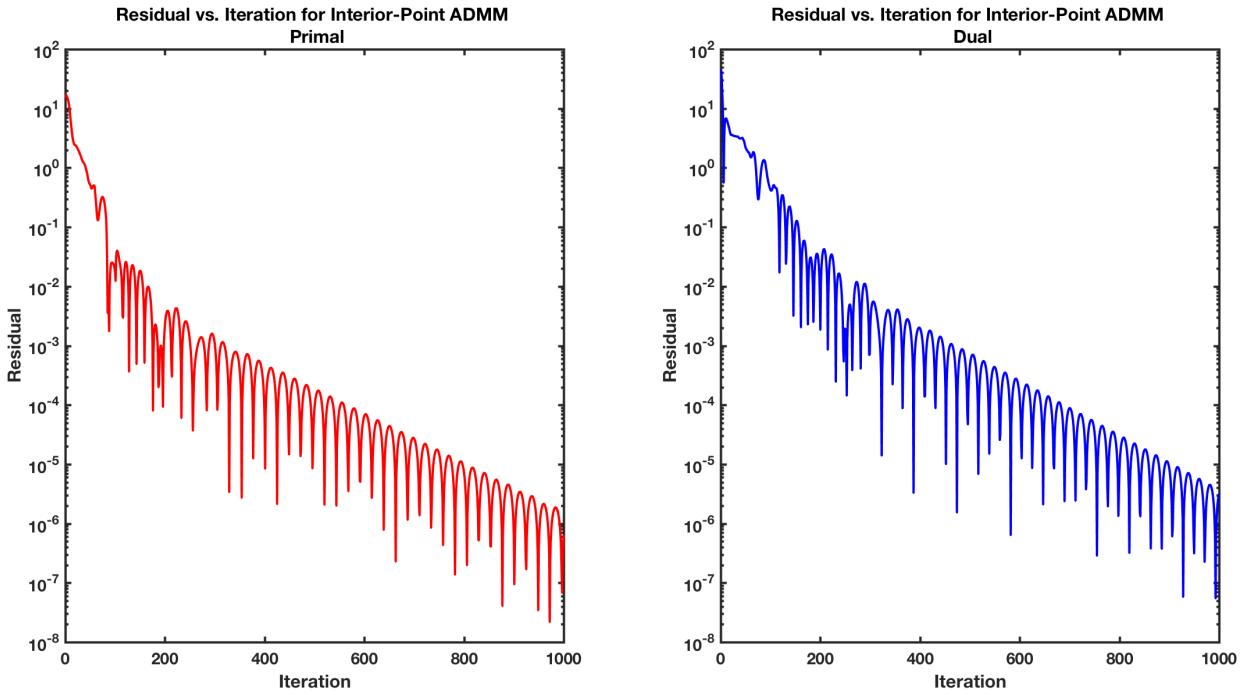

Figure 2: Graph of residual versus iteration when solving problems (13) and (14) (from left to right, respectively) using ADMM.

Varying γ. Here, we implement ADMM for the primal barrier problem (13) and vary γ. Specifically, we let $\gamma \in \{0.01, 0.2, 0.4, 0.6, 0.8, 0.99\}$. Figure 3 shows the residual versus iteration for solving the primal barrier problem (13) for the various γ. Emperically, small values of γ, specifically those close to 1, have relatively bad convergence rates, whereas all of the values of γ greater than or equal to 0.8 have similar or slightly varying convergence rates, with the fastest convergence rate belonging to $\gamma = 0.8$.

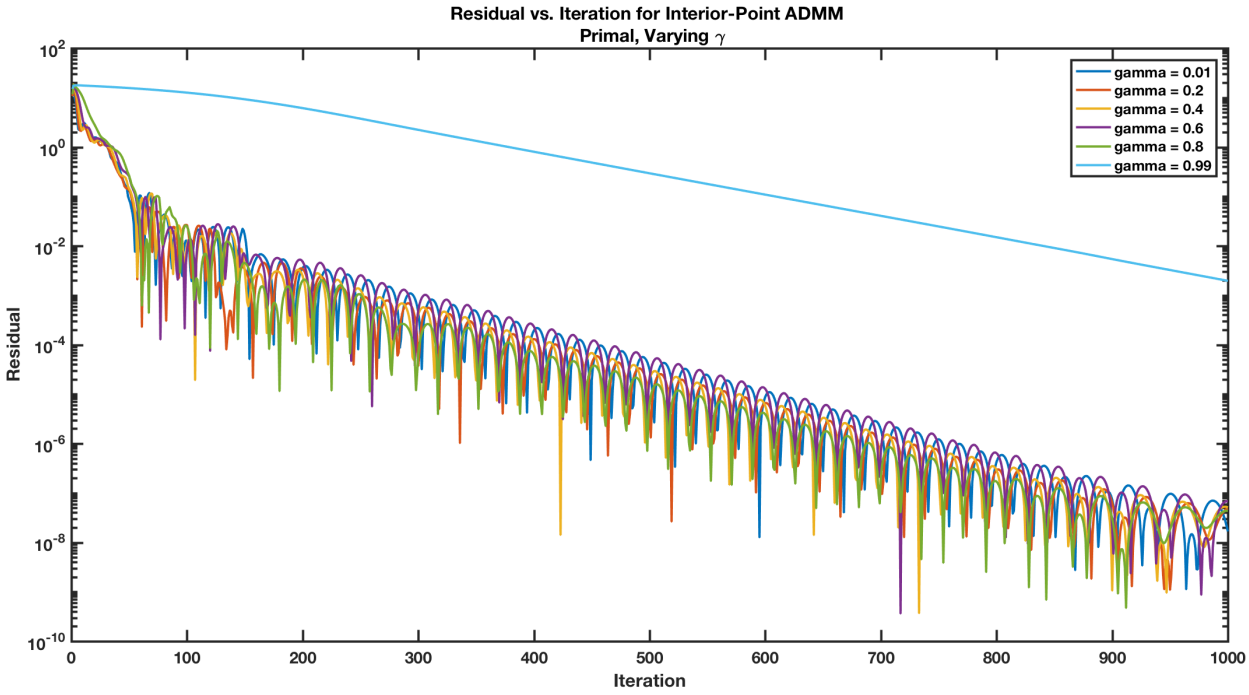

Figure 3: Graph of residual versus iteration when solving problems (13) using ADMM, and for $\gamma \in \{0.01, 0.2, 0.4, 0.6, 0.8, 0.99\}$.

Varying β. Here, we implement ADMM for the primal barrier problem (13) and vary β. Specifically, we let $\beta \in \{1, 10, 100, 500, 1000\}$. Figure 4 illustrates the residual versus iteration for solving the primal barrier problem (13) for the various β. Emperically, increasing β increases the convergence rate of the algorithm, up to a point: we see that the convergence rate of the algorithm gets better for increasing $\beta \in \{1, 10, 100\}$, but for $\beta = 500$, the convergence rate is markedly worse, and moreso for $\beta = 1000$.

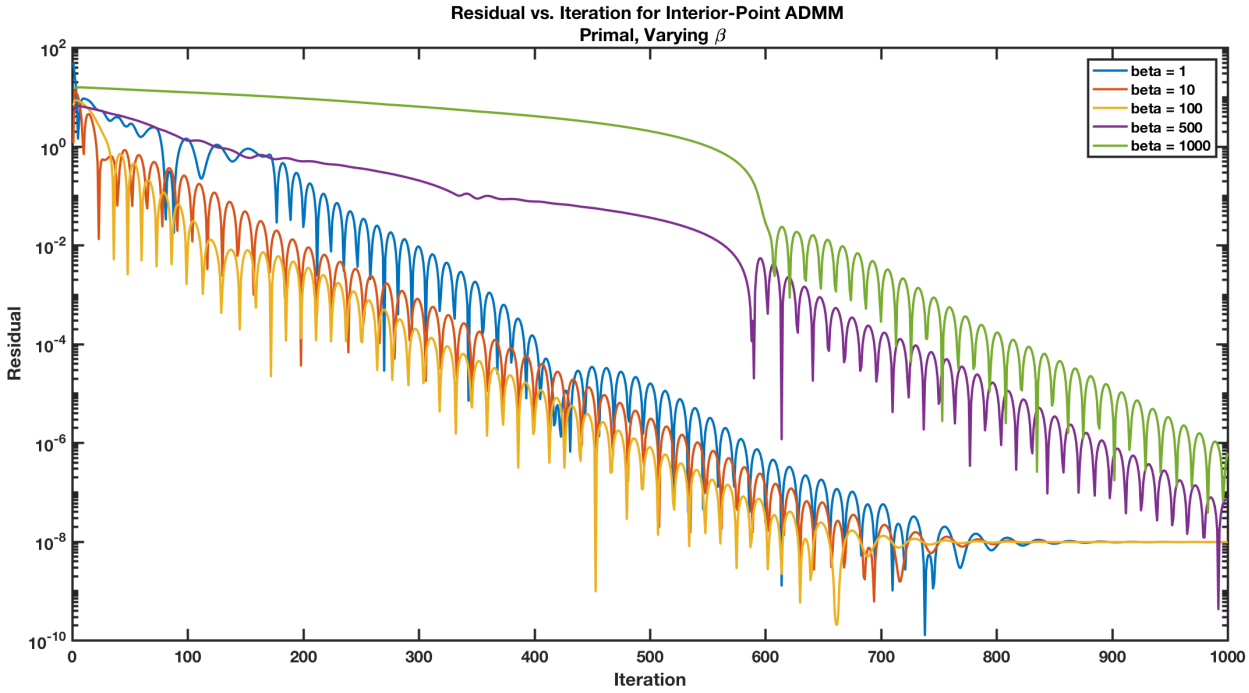

Figure 4: Graph of residual versus iteration when solving problems (13) using ADMM, and for $\beta \in \{1, 10, 100, 500, 1000\}$.

5.3 Preconditioned ADMM

In this case, we let $A' = (AA^T)^{-1/2}A$ and $b' = (AA^T)^{-1/2}b$, and consider the primal problem

$$\begin{array}{ll} \text{minimize}_x & c^T x_1 \\ \text{subject to} & A'x_1 = b' \\ & x_1 - x_2 = 0 \\ & x_2 \geq 0. \end{array} \tag{21}$$

Note that we only consider the primal problem, although the dual problem is trivially similar. Problem (21) is equivalent to problem (4), but now A' and b' are *preconditioned* [TB97], *i.e.*, the condition number of the system is smaller, or a small perturbation in the input will similarly only perturb the optimal solution by a small amount. Figure 5 below compares the performance to solving the non-conditioned problem instance with solving the preconditioned problem instance for both the primal and the dual. Notice that for the primal, the convergence is much smoother, and convergence occurs at a faster rate. As for the dual, both problem instances have approximately the same convergence rate. Our justification for why the preconditioned dual does not see better convergence rates is that the preconditioning of A to A' and b to b' acts like a pseudo dual update. As a benchmark, the preconditioned problem instance for the primal gets within 10^{-6} of the optimal solution approximately 350 iterations quicker than its non-conditioned problem instance counterpart. This emperical evidence shows that there exists problem instances in which preconditioning the data results in faster convergence rates.

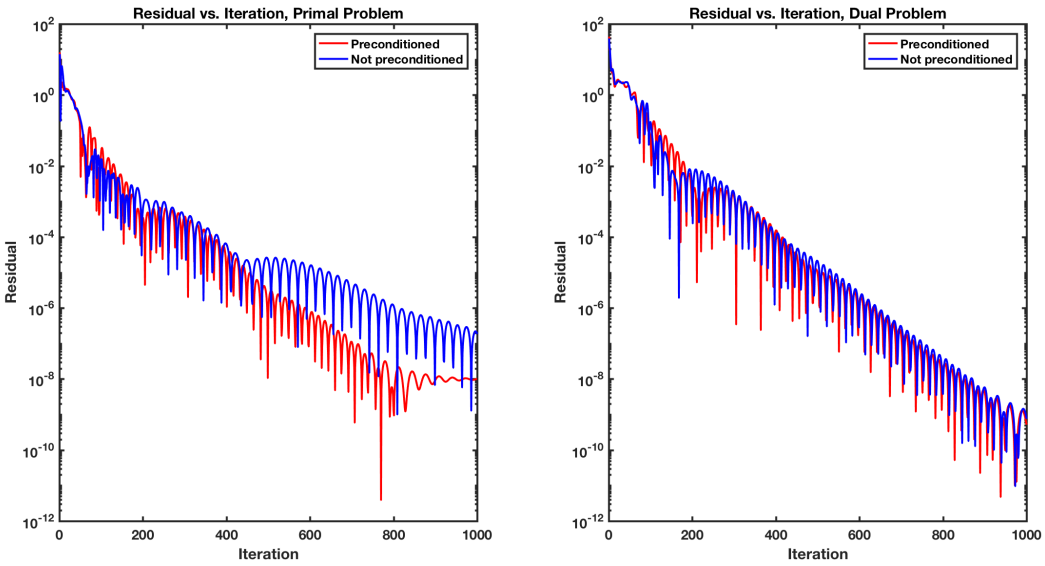

Figure 5: Graph of residual versus iteration when solving problems (1) and (2) (left to right, respectively) using ADMM, for both a non-conditioned case and a preconditioned case.

In addition, we precondition A and b and solve the barrier problem (13). Figure 6 below compares the performance to solving the non-conditioned problem instance with solving the preconditioned problem instance (We ignore the dual case for reasons explained above.) Again, notice that the convergence is smoother, and convergence occurs at a faster rate. For example, the preconditioned problem instance gets within 10^{-8} of the optimal solution 300 iterations quicker than the non-conditioned problem instance.

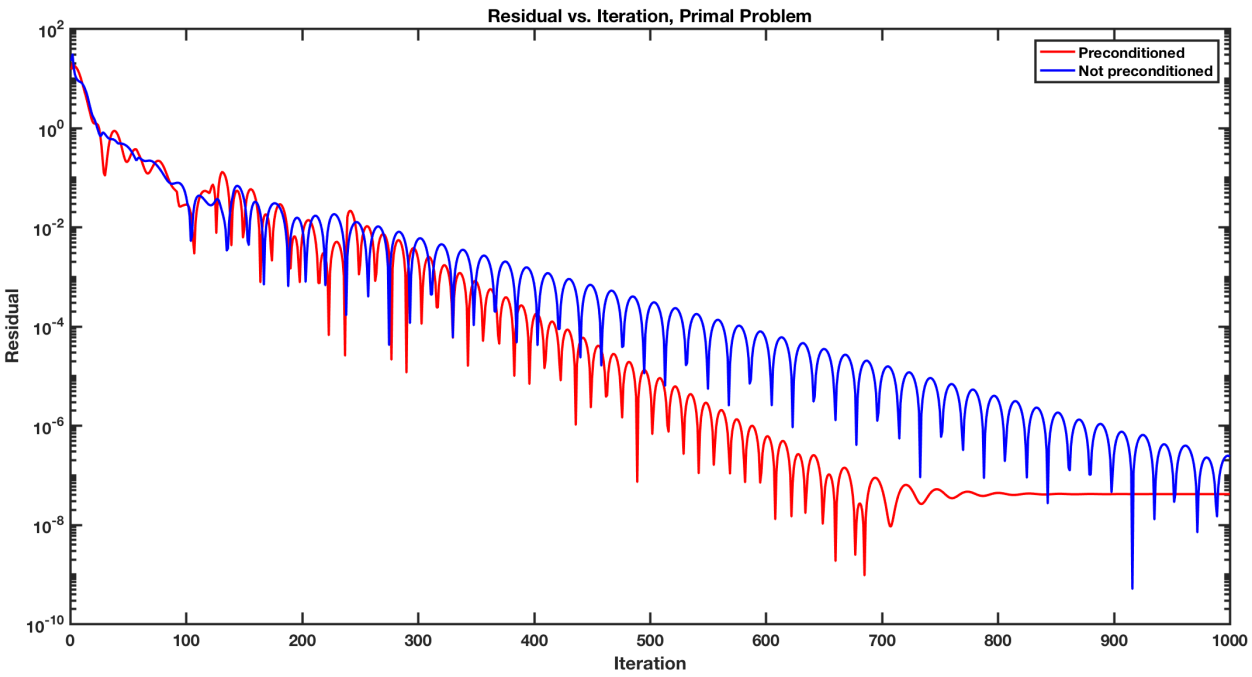

Figure 6: Graph of residual versus iteration when solving problem (13) using ADMM, for both a non-conditioned case and a preconditioned case.

6 Conclusion

In this paper, we have derived the Alternating Direction Method of Multipliers (ADMM) algorithm for the specific case of linear programming. In addition to this derivation, we also implemented this algorithm for the dual of the linear programming problem. We formulated the ADMM solution in an interior-point context, and analyze its performance to the implementation on the classical linear programming problem, and to an open-source solver. Lastly, we compare ADMM when the input data are not preconditioned to when the input data are preconditioned, and analyze the results. For the linear programming case, our results show that we can achieve modest accuracy in only a few hundred iterations, and can achieve high accuracy in less than 1000 iterations. In addition, we find that preconditioning our data can achieve us the same accuracy in a fashion that requires upwards of 350 iterations less than before.

Acknowledgements

The author would like to thank Professor Yinyu Ye for his insightful conversations regarding preconditioning and his advice in how to refine this paper.

A MATLAB code for projects

Attached to this paper are the MATLAB functions used to generate the results in this paper. Note that the code does not stop iterating at a given tolerance; this is intentional, and is done to equivalently see how much accuracy one can achieve for a given number of iterations.

```
function [x y s res_hist] = ADMM_primal(A, b, c, beta, max_iter)

[m n] = size(A);

x1 = abs(randn(n,1));
x2 = abs(randn(n,1));
y = abs(randn(m,1));
s = abs(randn(n,1));
res_hist = [];

for k = 1:max_iter
%% x1-update
x1 = inv(beta.*(A'*A) + eye(n))*(A'*y - c + s + beta.*A'*b + 1*x2);

%% x2-update
x2 = max(x1-s./beta, 0);
%% y-update
y = y - beta.*(A*x1-b);

%% s-update
s = s - beta.*(x1-x2);

res_hist = [res_hist; c'*x1];
end

x = x1;

end
```

```
function [x y s res_hist] = ADMM_dual(A, b, c, beta, max_iter)

[m n] = size(A);

x = abs(randn(n,1));
y = abs(randn(m,1));
s = abs(randn(n,1));
res_hist = [];

for k = 1:max_iter
    %% Update y
    y = (1./beta).*inv(A * A')*(b + A*x + beta.*A*(c-s));
    %% Update s
    %s = x./beta - A'*y + c;
    s = max(x./beta-A'*y +c, 0);
    %% Update x
    x = x - beta.*(A'*y + s - c);

    res_hist = [res_hist; b'*y];
end
```

```
function [x y s res_hist] = ADMM_primal_IP(A, b, c, beta, max_iter, gamma)

[m n] = size(A);
mu = 1;

x1 = abs(randn(n,1));
x2 = abs(randn(n,1));
y = abs(randn(m,1));
s = abs(randn(n,1));
res_hist = [];

for k = 1:max_iter
%% x1-update
x1 = inv(beta.*(A'*A) + 1*eye(n))*(A'*y - c + s + beta.*A'*b + 1*x2);

%% x2-update
for ii = 1:n
    x2(ii) = ((-s(ii)+ beta.*x1(ii)) + sqrt( (-s(ii) + beta.*x1(ii)).^2 + 4.*mu.*beta ))./
/(2.*beta);
end

%% y-update
y = y - beta.*(A*x1-b);

%% s-update
s = s - beta.*(x1-x2);

res_hist = [res_hist; c'*x1];

mu = gamma.*mu;
end

x = x1;

end
```

```
function [x y s res_hist] = ADMM_dual_IP(A, b, c, beta, max_iter)

[m n] = size(A);
gamma = 0.75;
mu = 1;

x = abs(randn(n,1));
y = abs(randn(m,1));
s = abs(randn(n,1));
res_hist = [];

for k = 1:max_iter
    %% Update y
    y = (1./beta).*inv(A * A')*(b + A*x + beta.*A*(c-s));
    %% Update s
    At_y = A' * y;
    for jj = 1:n
        s(jj) = (c(jj) + x(jj)./beta - At_y(jj))./2;
        s(jj) = s(jj) + sqrt(beta.^2 .*(At_y(jj) - x(jj)./beta - c(jj)).^2 + 4.*beta.
*mu)./(2.*beta);
    end
    %% Update x
    x = x - beta.*(A'*y + s - c);

    res_hist = [res_hist; b'*y];
    mu = gamma.*mu;

end
```

References

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[GB14] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, March 2014.

[LY15] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer Publishing Company, Incorporated, 2015.

[PB14] Neal Parikh and Stephen Boyd. Proximal algorithms. *Found. Trends Optim.*, 1(3):127–239, January 2014.

[Ren01] James Renegar. *A Mathematical View of Interior-point Methods in Convex Optimization*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[TB97] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.