MS&E 310, Fall 2017
Linear Programming
Prof. Yinyu Ye

Lorenzo Limonta – Project 4

Theroetical Solution

Question 1

Let's formulate the explicit update for $\mathbf{x}_1$ and $\mathbf{x}_2$ for the following LP problem

$$\begin{aligned} \text{minimize}_{\mathbf{x}_1,\mathbf{x}_2} \quad & \mathbf{c}^T\mathbf{x}_1 \\ s.t. \quad & A\mathbf{x}_1 = \mathbf{b} \\ & \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0} \\ & \mathbf{x}_2 \geq \mathbf{0} \end{aligned} \tag{1}$$

From the above system we have the following Lagrangian function:

$$L(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{c}^T\mathbf{x}_1 - \boldsymbol{\lambda}^T\left(A\boldsymbol{x}_1 - \boldsymbol{b}\right) - \boldsymbol{\mu}^T\left(\mathbf{x}_1 - \mathbf{x}_2\right) + \frac{\beta}{2}\left(||A\mathbf{x}_1 - \mathbf{b}||_2^2 + ||\mathbf{x}_1 - \mathbf{x}_2||_2^2\right)$$

Rewriting it without the norm 2, we have:

$$\begin{aligned} L(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{c}^T\mathbf{x}_1 - \boldsymbol{\lambda}^T\left(A\mathbf{x}_1 - \mathbf{b}\right) - \boldsymbol{\mu}^T\left(\mathbf{x}_1 - \mathbf{x}_2\right) + \\ + \frac{\beta}{2}\left(\mathbf{x}_1^T A^T A\mathbf{x}_1 - 2\mathbf{b}^T A\mathbf{x}_1 + \mathbf{b}^t\mathbf{b} + \mathbf{x}_1^T\mathbf{x}_1 + \mathbf{x}_2^T\mathbf{x}_2 - 2\mathbf{x}_1^T\mathbf{x}_2\right) \end{aligned} \tag{2}$$

Let's look then at the update scheme:

- Update variable $\mathbf{x}_1$:

$$\mathbf{x}_1^{k+1} = \arg\min\, L^P(\mathbf{x}_1, \mathbf{x}_2^k, \boldsymbol{\lambda}^k)$$

  In the writing above $\mathbf{x}_2$, $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ are constant. Thus, to find the analytical value for $\mathbf{x}_1$ we simply take the derivative of our Lagrangian w.r.t to it.

$$\begin{aligned} &\frac{\partial L}{\partial \mathbf{x}_1} = \mathbf{c} - A^T\boldsymbol{\lambda} - \boldsymbol{\mu} + \beta\left[\left(A^TA\right)\mathbf{x}_1 - A^T\mathbf{b} + \mathbf{x}_1 - \mathbf{x}_2\right] = \mathbf{0} \\ &\implies \left[\left(A^TA\right)\mathbf{x}_1 + \mathbf{x}_1\right] = \mathbf{x}_2 + A^T\mathbf{b} + \frac{1}{\beta}\left(A^T\boldsymbol{\lambda} + \boldsymbol{\mu} - \mathbf{c}\right) \end{aligned}$$

  This gives our analytical solution for $\mathbf{x}_1$ under the assumption that $(A^TA + I)$ is invertible

$$\mathbf{x}_1^{k+1} = \left(A^TA + I\right)^{(-1)}\left[\mathbf{x}_2^k + A^T\mathbf{b} + \frac{1}{\beta}\left(A^T\boldsymbol{\lambda}^k + \boldsymbol{\mu}^k - \mathbf{c}\right)\right] \tag{3}$$

- Update variable $\mathbf{x}_2$:

$$\mathbf{x}_2^{k+1} = \arg\min\ L^P(\mathbf{x}_1^{k+1}, \mathbf{x}_2, \boldsymbol{\lambda}^k)$$

This time $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$, $\mathbf{x}_1$ are constants, with $\mathbf{x}_1$ being the updated value found in eq. 3. As before, to find the new updated value we proceed by taking the derivative of the Lagrangian w.r.t. $\mathbf{x}_2$

$$\frac{\partial L}{\partial \mathbf{x}_2} = \boldsymbol{\mu} + \frac{\beta}{2}\left[2\mathbf{x}_2 - 2\mathbf{x}_1\right] = 0$$

Which gives us:

$$\mathbf{x}_2^{k+1} = \ \max\ \{\mathbf{0}, \mathbf{x}_1^{k+1} - \frac{\boldsymbol{\mu}^k}{\beta}\}_i \tag{4}$$

Equations 3, 4 are therefore our explicit formula for the iterative xs

Question 2

Let's now develop the explicit formula for the dual problem. From our original LP problem, we have the following dual:

$$\begin{aligned} \text{maxmimize}_{\mathbf{y},\mathbf{s}} \quad & \mathbf{b}^T\mathbf{y} \\ s.t. \quad & A^T\mathbf{y} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{5}$$

The above system thus gives us the following Lagrangian:

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

Which can be explicitly rewritten as

$$\begin{aligned} L(\mathbf{y}, \mathbf{s}, \boldsymbol{\lambda}) = & -\mathbf{b}^T\mathbf{y} - \boldsymbol{\lambda}^T\left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) \\ & + \frac{\beta}{2}\left(\mathbf{y}^T A A^T\mathbf{y} + \mathbf{s}^T\mathbf{s} + \mathbf{c}^T\mathbf{c} + 2\mathbf{y}^T A\mathbf{s} - 2\mathbf{s}^T\mathbf{c} - 2\mathbf{y}^T A\mathbf{c}\right) \end{aligned} \tag{6}$$

Looking at the update scheme we have:

- Update variable $\mathbf{y}$:

$$\mathbf{y}^{k+1} = \arg\min\ L^d(\mathbf{y}, \mathbf{s}^k, \boldsymbol{\lambda}^k)$$

In the writing above $\mathbf{s}$, $\boldsymbol{\lambda}$, are constant. Thus, to find the analytical value for $\mathbf{y}$ we simply take the derivative of our Lagrangian w.r.t to it.

$$\frac{\partial L}{\partial \mathbf{y}} = -\mathbf{b} - A\boldsymbol{\lambda} + \beta\left(AA^T\mathbf{y} + A\mathbf{s} - A\mathbf{c}\right) = \mathbf{0}$$
$$\implies \left(AA^T\right)\mathbf{y} = A\mathbf{c} - A\mathbf{s} + \frac{1}{\beta}\left(\mathbf{b} + A\boldsymbol{\lambda}\right)$$

Our updated $\mathbf{y}^{k+1}$ is:

$$\mathbf{y}^{k+1} = \left(AA^T\right)^{(-1)}\left[A\mathbf{c} - A\mathbf{s}^k + \frac{1}{\beta}\left(\mathbf{b} + A\boldsymbol{\lambda}^k\right)\right] \tag{7}$$

- Update variable $\mathbf{s}$:

$$\mathbf{s}^{k+1} = \arg\min\ L^d(\mathbf{y}^{k+1}, \mathbf{s}, \boldsymbol{\lambda}^k)$$

In the writing above $\mathbf{y}^{k+1}$, $\boldsymbol{\lambda}$, are constant. Thus, to find the analytical value for $\mathbf{s}$ we simply take the derivative of our Lagrangian w.r.t to it.

$$\frac{\partial L}{\partial \mathbf{s}} = -\boldsymbol{\lambda} + \beta\left(\mathbf{s} + A^T\mathbf{y} - \mathbf{c}\right) = 0$$
$$\implies \mathbf{s} = \mathbf{c} - A^T\mathbf{y} + \frac{\boldsymbol{\lambda}}{\beta}$$

Our updated $\mathbf{s}$ is given by:

$$\mathbf{s}^{k+1} = \ \max\ \{\mathbf{0}, \mathbf{c} - A^T\mathbf{y}^{k+1} + \frac{\boldsymbol{\lambda}^k}{\beta}\}_i \tag{8}$$

Question 3,4

For reference, we rewrite the problem when a barrier function is added. Notice the change in coefficient from μ in the guidelines to ξ in our formulation.

Primal problem

$$\begin{aligned} \text{minimize}_{\mathbf{x}} \quad & \mathbf{c}^T - \xi\sum_j ln(x_j)\mathbf{x}_1 \\ s.t. \quad & A\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned} \tag{9}$$

Where the lagrangian is

$$L(\mathbf{x}_1, \mathbf{x}_2, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \mathbf{c}^T\mathbf{x}_1 - \xi\sum_j ln(x_j) - \boldsymbol{\lambda}^T\left(A\boldsymbol{x}_1 - \boldsymbol{b}\right) - \boldsymbol{\mu}^T\left(\mathbf{x}_1 - \mathbf{x}_2\right) + \frac{\beta}{2}\left(||A\mathbf{x}_1 - \mathbf{b}||_2^2 + ||\mathbf{x}_1 - \mathbf{x}_2||_2^2\right)$$

The above representation doesn't have an analytical close form solution, therefore when updating our variables with the same scheme as in question 1, an additional numerical minimization technique will be needed. In our case, we choose matlab fmincon.

Dual problem

$$\begin{aligned} \text{maxmimize}_{\mathbf{y},\mathbf{s}} \quad & \mathbf{b}^T\mathbf{y} + \xi \sum_j ln(s_j) \\ s.t. \quad & A^T\mathbf{y} + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{10}$$

With its Lagrangian being:

$$L(\mathbf{y}, \mathbf{s}, \boldsymbol{\lambda}) = -\mathbf{b}^T\mathbf{y} - \xi \sum_j ln(s_j) - \boldsymbol{\lambda}^T \left(A^T\mathbf{y} + \mathbf{s} - \mathbf{c}\right) + \frac{\beta}{2}||A^T\mathbf{y} + \mathbf{s} - \mathbf{c}||_2^2$$

Just like with the primal, the above representation doesn't have an analytical close form solution. We again chose matlab fmincon for the minimization needed when updating our variables.

Question 5

Choosing the first formulation for our multi-block ADMM, we can re-write its algorithm in a similar fashion to what done in question 2:

- Update variable $\mathbf{y}_1$:

$$\mathbf{y}_1^{k+1} = \arg\min\, L^d(\mathbf{y}_1, \mathbf{y}_2^k, \mathbf{s}^k, \boldsymbol{\lambda}^k)$$

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{y}_1} &= -\mathbf{b}_1 - A_1\boldsymbol{\lambda} + \beta\left(A_1A_1^T\mathbf{y}_1 + A_1A_2^T\mathbf{y}_2 + A_1\mathbf{s} - A_1\mathbf{c}\right) = \mathbf{0} \\ \implies & \left(A_1A_1^T\right)\mathbf{y}_1 = A_1\mathbf{c} - A_1A_2^T\mathbf{y}_2 - A_1\mathbf{s} + \frac{1}{\beta}\left(\mathbf{b}_1 + A_1\boldsymbol{\lambda}\right) \end{aligned}$$

  Our update for $\mathbf{y}_1$ then becomes:

$$\mathbf{y}_1^{k+1} = \left(A_1A_1^T\right)^{(-1)}\left[A_1\mathbf{c} - A_1A_2^T\mathbf{y}_2^k - A_1\mathbf{s}^k + \frac{1}{\beta}\left(\mathbf{b}_1 + A_1\boldsymbol{\lambda}^k\right)\right] \tag{11}$$

- Update variable $\mathbf{y}_2$:

$$\mathbf{y}_2^{k+1} = \arg\min\, L^d(\mathbf{y}_1^{k+1}, \mathbf{y}_2, \mathbf{s}^k, \boldsymbol{\lambda}^k)$$

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{y}_2} &= -\mathbf{b}_2 - A_2\boldsymbol{\lambda} + \beta\left(A_2A_2^T\mathbf{y}_2 + A_2A_1^T\mathbf{y}_1 + A_2\mathbf{s} - A_2\mathbf{c}\right) = \mathbf{0} \\ \implies & \left(A_2A_2^T\right)\mathbf{y}_2 = A_2\mathbf{c} - A_2A_1^T\mathbf{y}_1 - A_2\mathbf{s} + \frac{1}{\beta}\left(\mathbf{b}_2 + A_2\boldsymbol{\lambda}\right) \end{aligned}$$

  Our update for $\mathbf{y}_2$ then becomes:

$$\mathbf{y}_2^{k+1} = \left(A_2A_2^T\right)^{(-1)}\left[A_2\mathbf{c} - A_2A_1^T\mathbf{y}_1^{k+1} - A_2\mathbf{s}^k + \frac{1}{\beta}\left(\mathbf{b}_2 + A_2\boldsymbol{\lambda}^k\right)\right] \tag{12}$$

- Update variable $\mathbf{s}$:

$$\mathbf{s} = \arg\min L^d(\mathbf{y}_1^{k+1}, \mathbf{y}_2^{k+1}, \mathbf{s}, \boldsymbol{\lambda}^k)$$

$$\frac{\partial L}{\partial \mathbf{s}} = -\boldsymbol{\lambda} + \beta\left(A_1^T\mathbf{y}_1 + A_2^T\mathbf{y}_2 + \mathbf{s} - \mathbf{c}\right) = 0$$
$$\implies \mathbf{s} = \frac{\boldsymbol{\lambda}}{\beta} + \mathbf{c} - A_1^T\mathbf{y}_1 - A_2^T\mathbf{y}_2$$

Thus, our update for $\mathbf{s}$ will be:

$$\mathbf{s}^{k+1} = \max\left\{\mathbf{0}, \frac{\boldsymbol{\lambda}^k}{\beta} + \mathbf{c} - A_1^T\mathbf{y}_1^{k+1} - A_2^T\mathbf{y}_2^{k+1}\right\} \tag{13}$$

- Update variable $\boldsymbol{\lambda}$:

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k - \beta\left(A_1\mathbf{y}_1^{k+1} + A_2\mathbf{y}_2^{k+1} + \mathbf{s}^{k+1} - \mathbf{c}\right) \tag{14}$$

Given equations 11-14 we can now easily solve our block 3 ADMM

Multi-Block

We can easily generalize our three blcok ADMM to a multi-block one as follows. Let the problem be

$$\begin{aligned} \text{maxmimize}_{\mathbf{y}_i,\mathbf{s}} \quad & \sum_i \mathbf{b}_i^T\mathbf{y}_i \\ s.t. \quad & \sum_i A_i^T\mathbf{y}_i + \mathbf{s} = \mathbf{c} \\ & \mathbf{s} \geq \mathbf{0} \end{aligned} \tag{15}$$

We can then write the following Lagrangian function:

$$L(\mathbf{y}_i, \mathbf{s}, \boldsymbol{\lambda}) = -\sum_i \mathbf{b}_i^T\mathbf{y}_i - \boldsymbol{\lambda}^T\left(\sum_i A_i^T\mathbf{y}_i + \mathbf{s} - \mathbf{c}\right) + \frac{\beta}{2}||\sum_i A_i^T\mathbf{y}_i + \mathbf{s} - \mathbf{c}||_2^2$$

Our ADMM then becomes:

- Update variable $\mathbf{y}_i$

$$\begin{aligned} y_i^{k+1} &= \left(A_iA_i^T\right)^{(-1)}\sum_{j\neq i}\left[A_j\mathbf{c} - A_jA_i^T\mathbf{y}_j^* - A_j\mathbf{s}^k + \frac{1}{\beta}\left(b_j + A_j\boldsymbol{\lambda}^k\right)\right] \\ \mathbf{y}_j^* &= \mathbf{y}_j^k \quad \text{if } j > i \\ \mathbf{y}_j^* &= \mathbf{y}_j^{k+1} \text{ if } j < i \end{aligned} \tag{16}$$

- Update variable $\mathbf{s}$

$$\mathbf{s}^{k+1} = \max\{\mathbf{0}, \frac{\boldsymbol{\lambda}^k}{\beta} + \mathbf{c} - \sum_i A_i^T \mathbf{y}_i^{k+1}\} \tag{17}$$

- Update variable $\boldsymbol{\lambda}$

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^k - \beta \left[\sum_i \left(A_i \mathbf{y}_i^{k+1}\right) + \mathbf{s}^{k+1} - \mathbf{c}\right] \tag{18}$$

Practical Implementation - Validation

Let's first look at a toy model to validate our code

$$\begin{aligned} \text{minimize} \quad & -5x_1 + 4x_2 + 0x_3 + 0x_4 \\ \text{s.t.} \quad & 6x_1 + 4x_2 + x_3 + 0x_4 = 24 \\ & x_1 + 2x_2 + 0x_3 + 1x_4 = 6 \\ & x_1 \geq 0 \end{aligned} \tag{19}$$

The above problem has primal solution $\mathbf{x}_1 = 3$, $\mathbf{x}_2 = 1.5$, $\mathbf{x}_3 = 0$, $\mathbf{x}_4 = 0$ which gives f_{min}=-21

Throughout the validation process — unless stated otherwise — we will use three stopping criteria: norm 2 difference between exact solution and numerical solution ($||f_{exact} - f_{x_k}||_2$), norm 2 difference between solutions at two iterative steps ($||f_{x_{k+1}} - f_{x_k}||_2$, referred to as criterion 1, and norm 2 difference between the variables value at each iteration ($||x_{k+1} - x_k||_2$, referred to as criterion 2. Since for a general problem the exact solution is not known a priori, we do this to establish a baseline validity on the convergence of each of the two criteria.

The following constants will be used throughout our validation process:

- $\beta = 0.5$
- Tolerance = 1e-14
- Error: $\frac{||f_{exact} - f_k||_2}{|f_{exact}|}$
- Max-iter = 10^8

Question 1

Re-writing the problem in matrix form similarly to eq. 1, we have:

$$\begin{aligned} \text{minimize} \quad & \left| \begin{matrix} -5 & 4 & 0 & 0 \end{matrix} \right| \mathbf{x}_1 \\ \text{s.t.} \quad & \left| \begin{matrix} 6 & 4 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{matrix} \right| \mathbf{x}_1 = \left| \begin{matrix} 24 \\ 6 \end{matrix} \right| \\ & \mathbf{x}_1 - \mathbf{x}_2 = \mathbf{0} \\ & \mathbf{x}_2 \geq 0 \end{aligned}$$

Let's look at the solution for the primal problem, with and without pre-conditioner. Tab. 1 and fig. 1 show that the solution without preconditioner requires less iterations than the one with preconditioner but the preconditioned solution satisfies both criterion 1 and 2 simultaneously. The decrease in performance of the pre-condtioned case in this situation needs further investigation, as no reasonable explanation can be proposed.

Toy problem primal solution					
Stopping Criterion	Required Tol	Computed Tol	Iter	Time (s)	Pre-Conditioner
$\|\|f_{exact} - f_{x_k}\|\|_2$	1e-14	0	121	0.002186933	
	1e-14	3.5527e-15	185	0.003334006	✓
$\|\|f_{x_{k+1}} - f_{x_k}\|\|_2$	1e-14	0	123	0.001967325	
	1e-14	7.1054e-15	187	0.002997109	✓
$\|\|x_{k+1} - x_k\|\|_2$	1e-14	0	143	0.002194761	
	1e-14	7.3726e-15	187	0.002857202	✓

Table 1: We show the results from the implementation of our toy problem. While the non-preconditioned case is solved with fewer iterations, convergence in both metric is not achieve at the same time. Computation time is averaged over 1000 tries.

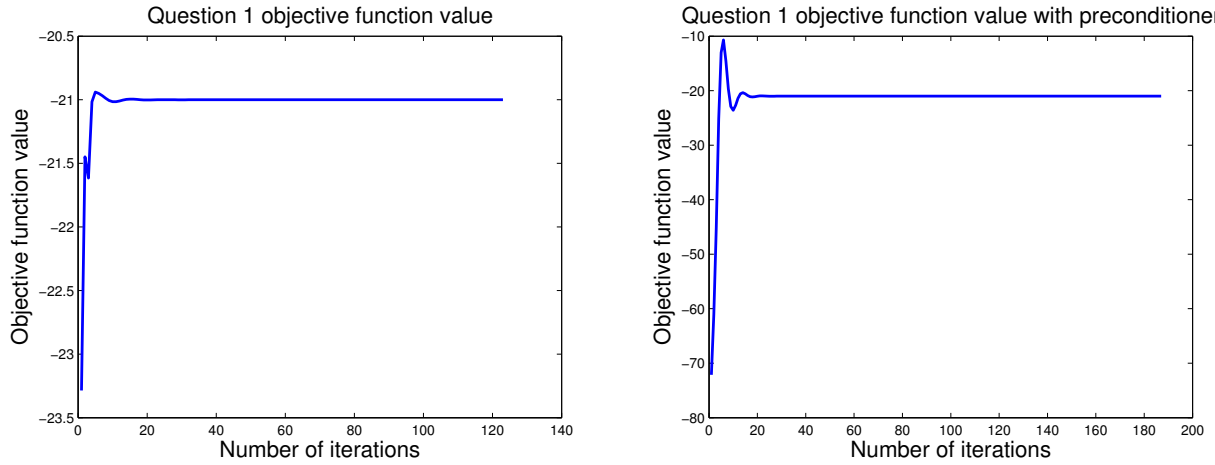

Figure 1: We show the convergence of ADMM for the problem solved when in form of eq. 1 with and without preconditioner. For the simple toy problem we obtain convergence rate much faster for the non-preconditioned problem. Nonetheless, the pre-condition solution looks "smoother".

Question 2

Writing out the dual of eq.19 as shown in eq.5 we obtain the following system of equation

$$\begin{aligned} \text{minimize} & \left| \begin{array}{cc|c} 24 & 6 & \mathbf{y} \end{array} \right. \\ \text{s.t.} & \left| \begin{array}{cc} 6 & 1 \\ 4 & 2 \\ 1 & 0 \\ 0 & 1 \end{array} \right| \mathbf{y} + \mathbf{s} = \left| \begin{array}{c} -5 \\ 4 \\ 0 \\ 0 \end{array} \right| \\ & \mathbf{s} \geq 0 \end{aligned}$$

The exact solutions for the dual are given by $\mathbf{y}_1 = -0.75$, $\mathbf{y}_2 = -0.5$,$\mathbf{y}_3 = 0$, $\mathbf{y}_4 = 0$ $\mathbf{s}_1 = 0$, $\mathbf{s}_2 = 0$, $\mathbf{s}_3 = 0.75$, $\mathbf{s}_4 = 0.5$.

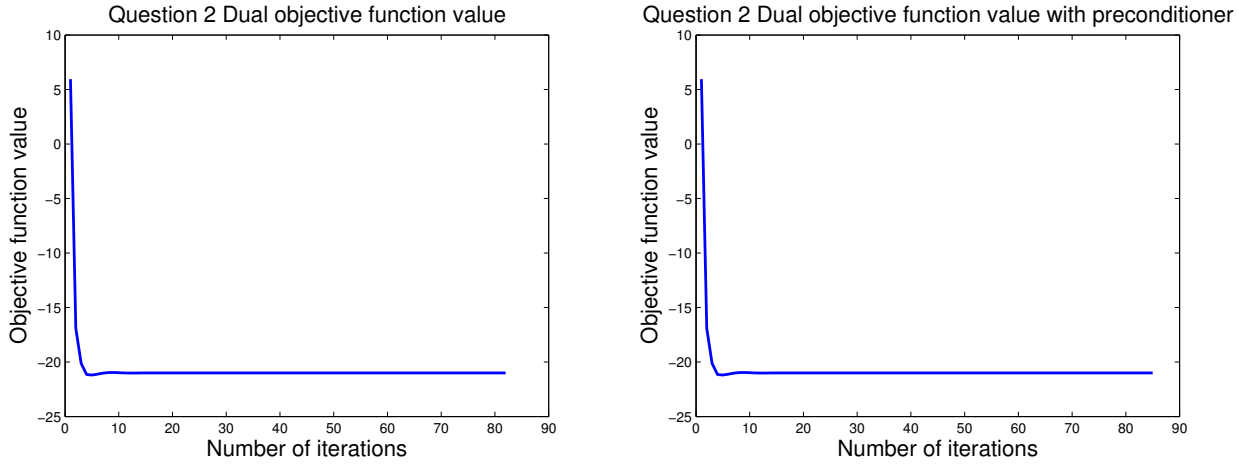

Figure 2: We show the convergence of ADMM for the dual problem solved when in form of eq.5 with and without preconditioner. The convergence rate of the two problem is substantially the same.

Tab.2 and fig.2 show the results for this situation. We can see how solving the dual problem with pre-conditioning on the primal doesn't significantly modify our results. With the same number of iteration, the pre-conditioned one is slightly faster due to the easiness in (pre) computing its inverse. (N.B. the inverse in eq.7 is precomputed as it is a constant throughout the process)

Toy problem dual solution					
Stopping Criterion	Required Tol	Computed Tol	Iter	Time (s)	Pre-Conditioner
$\|\|f_{exact} - f_{x_k}\|\|_2$	1e-14	7.1054e-15	81	0.002186933	
	1e-14	0	85	0.003334006	✓
$\|\|f_{x_{k+1}} - f_{x_k}\|\|_2$	1e-14	7.1054e-15	82	0.001480335	
	1e-14	7.1054e-15	85	0.001523686	✓
$\|\|y_{k+1} - y_k\|\|_2$	1e-14	3.7942e-15	82	0.001405075	
	1e-14	6.4047e-15	82	0.001404704	✓

Table 2: We show the results from the implementation of our toy problem for the dual. As expected, there is no substantial difference when preconditioning. Computation time is averaged over 1000 tries.

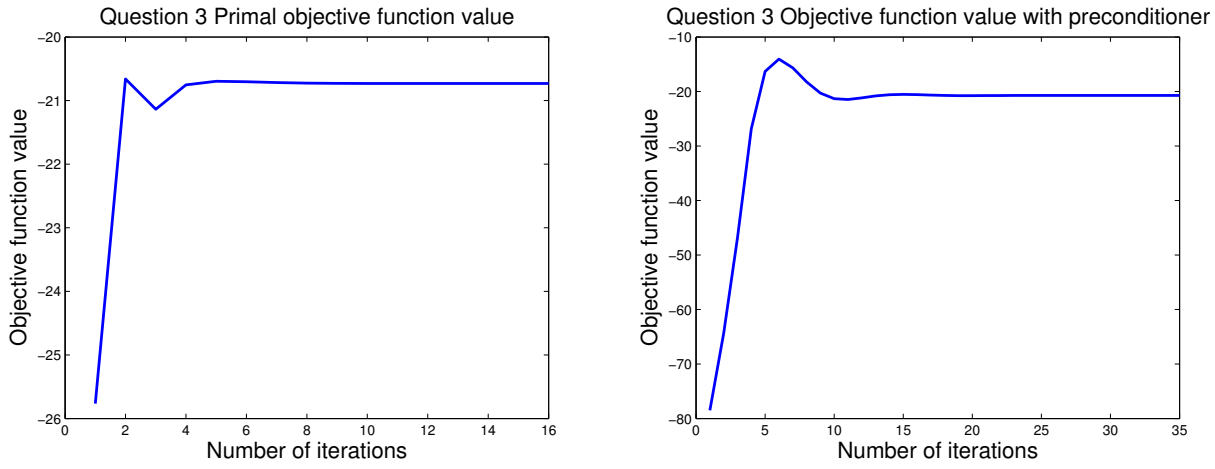

Figure 3: We show the convergence of ADMM for the primal problem solved with a barrier function. Preconditioning our problem seem to cause the convergence to take much longer.

Question 3

Let's look at the performance when implementing the interior-point case. To amplify the effects of adding a barrier, we will choose a relatively high value of ξ=0.9. (**N.B.** ξ stands in for the μ in the prompt, as μ has been used as one of the Lagrangian multipliers in section one)

Primal problem

Toy problem primal with barrier, ξ= 0.9					
Stopping Criterion	Required Tol	Computed Tol	Iter	Error (%)	Pre-Conditioner
$\|\|f_{x_{k+1}} - f_{x_k}\|\|_2$	1e-8	0	16	1.2798	
	1e-8	0	35	1.2802	✓
$\|\|x_{k+1} - x_k\|\|_2$	1e-8	0	16	1.2798	
	1e-8	0	35	1.2802	✓

Table 3: We show the results from the implementation of our toy problem with barrier function and a relatively large value for the barrier coefficient. Again, the preconditioned case seem to be taking significantly longer to solve than the non-preconditioned one.

As shown in fig.3, we are solving a slightly modified problem where our final objective value will not be -21. Depending on the value of ξ, the solution for the interior point ADMM will converge to different values. Tab.3 shows the necessary iterations to converge to the optimal solution of our problem. It seem that preconditioning worsen the convergence rate of our solver. Another factor that strongly influences the rate of convergence are the starting value of our arg min Lagrangian. The further away from the optimal solution of each iteration, the noisier the inner optimization will be. This translates in inaccurate values for our variables at each iteration, which will cause the process to take longer to converge.

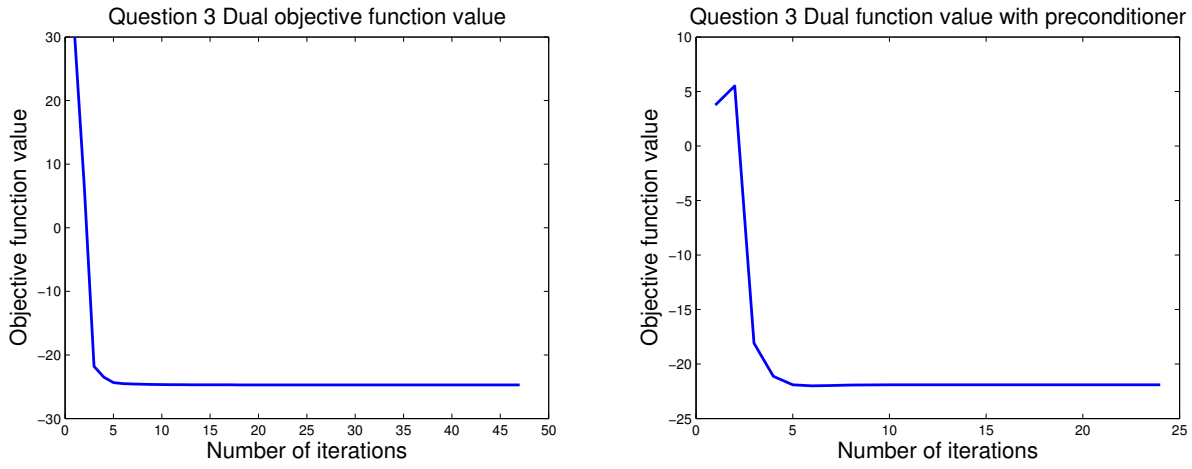

Figure 4: We show the convergence of ADMM for the primal problem solved with a barrier function. As with the previous case, preconditioning doesn't change the time required to reach a solution.

Dual problem

Toy problem dual with barrier, ξ= 0.9					
Stopping Criterion	Required Tol	Computed Tol	Iterations	Error (%)	Pre-Conditioner
$\|\|f_{x_{k+1}} - f_{x_k}\|\|_2$	1e-8	0	46	17.6692	
	1e-8	0	46	17.6692	✓
$\|\|x_{k+1} - x_k\|\|_2$	1e-8	0	46	17.6692	
	1e-8	0	46	17.6693	✓

Table 4: We show the results from the implementation of our toy problem with barrier function and a relatively large value for the barrier coefficient. As before, preconditioning doesn't affect our solution.

As seen in question 3, pre-conditioning doesn't modify the solution of our dual problem significantly, therefore we will not implement it when looking at bigger problems. Interestingly to notice, both stopping criteria are satisfied at the same iteration in the dual. The high value of ξ causes our interior point method to converge to a solution far from optimal

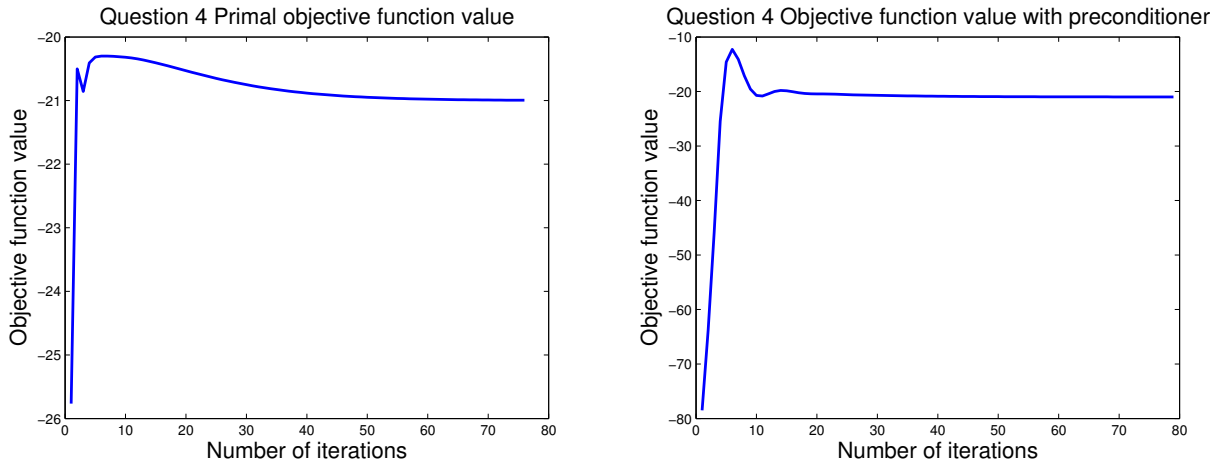

Figure 5: We show the convergence of ADMM for the primal problem solved with an outer iteration barrier function. Unlike in previous situations, where our objective value would oscillate around our solution, it decreases smoothly while converging to the exact values. This is due to the iterative decreasing value of ξ.

Question 4

Let's look at the performance when implementing the case with decreasing barrier function. To magnify its effects, we choose relatively big starting value for ξ: 0.9 and $\gamma = 0.9$.

Primal problem

This time around, our performance is much better than before. We converge approximately to the exact solution of negative 21, and we do so in a reasonable number of iterations. The reason is rather simple: as the log approaches infinity it's coefficient approaches zero, nullifying its effects.
Once again, we want to point out the sensitivity of the convergence rate to the inner minimization algorithm and its starting value. Depending on the chosen conditions it may takes longer to solve the inner arg min problem.

Toy problem primal with barrier, ξ= 0.9,γ= 0.9					
Stopping Criterion	Required Tol	Computed Tol	Iterations	Error (%)	Pre-Conditioner
$\|\|f_{x_{k+1}} - f_{x_k}\|\|_2$	1e-8	0	76	0.0255	
	1e-8	0	79	0.0212	✓
$\|\|x_{k+1} - x_k\|\|_2$	1e-8	0	76	0.0255	
	1e-8	0	79	0.0212	✓

Table 5: We show the results from the implementation of our toy problem for an outer iteration barrier function. We use a relatively large value for ξ and gamma to magnify the effects of this method.

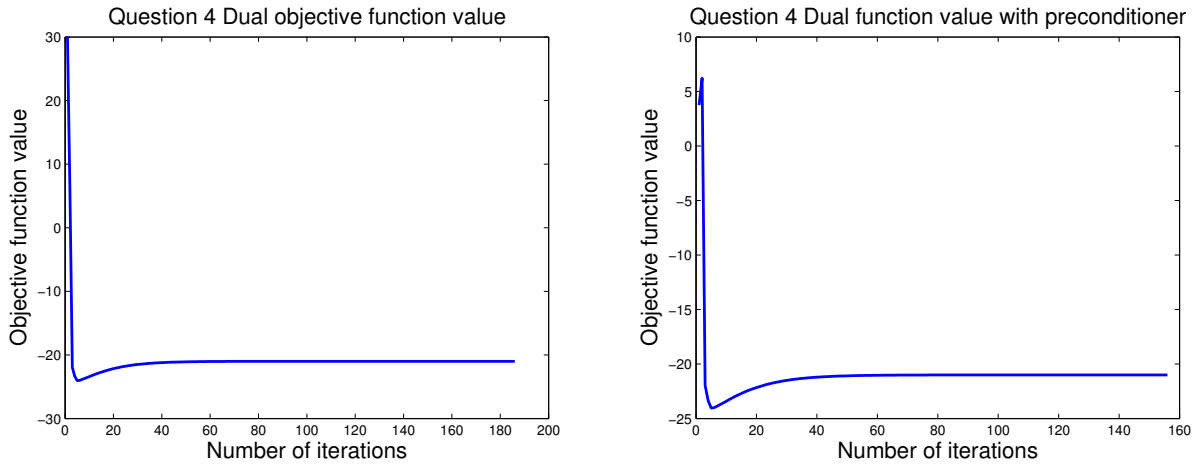

Figure 6: We show the convergence of ADMM for the dual problem solved with a barrier function. We witness a smooth approach to our solution.

Dual problem

Toy problem dual with barrier, ξ= 0.9,γ= 0.9					
Stopping Criterion	Required Tol	Computed Tol	Iterations	Error (%)	Pre-Conditioner
$\|\|f_{y_{k+1}} - f_{y_k}\|\|_2$	1e-8	5.9969e-09	189	2.0589e-04	
	1e-8	1.2974e-09	155	2.3224e-04	✓
$\|\|y_{k+1} - y_k\|\|_2$	1e-8	0	108	0.0015	
	1e-8	0	109	0.0014	✓

Table 6: We show the results from the implementation of our toy problem for an outer iteration barrier function. We use a relatively large value for ξ and gamma to magnify the effects of this method.

Changes of β, γ

Let's analyze the performance of our interior point ADMM for different values of β, γ when ξ=0.9. For brevity, we will do so for the primal problem, with stopping criteria based on the absolute error being above 0.00125 (i.e. $||f_{x_{k+1}} - f_{x_{exact}}||_2 \geq 1e-3$). In order to amplify the effects of our choice of β,γ, rather than accurate guesses for the inner function, we will use (constant) numbers drawn from the uniform distribution (0 1] as our guess at each inner iteration.

Fig.7 shows that increasing β increases the number of iterations needed to reach the solution. This is to be expected: the smaller is beta, the smaller the effects of the constraints are on the Lagrangian. This makes the minimization of the Lagrangian equivalent to solving an unconstrained minimization of the objective function. Which in turn increases the rate of convergence in situations where we are alway on the interiors of our problem (such as this). On the opposite side, decreasing the value of gamma increases the rate of convergence. Once again, this behaviour is to be expected: the lower the gamma means that at each iteration the barrier function has less influence on our objective function and we end up solving a similar problem to the one described in questions 1 or 2.

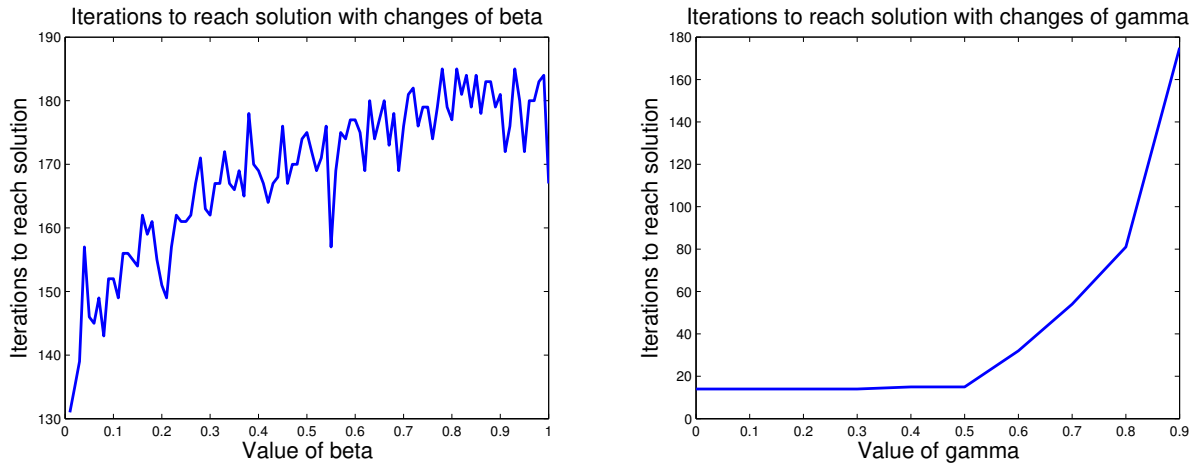

Figure 7: We show the convergence of ADMM to the desired error tolerance when changing parameters beta and gamma.

Question 5

We implement ADMM multi-block and see how it performs on our toy problem. We will implement both the standard multi-block ADMM and the randomly permuted one, based on the solution in eq.12-14.

- Standard Multi-Block

  We can see that the implementation of our multi-block method converges in this situation. Due to the structure of the problem (the original matrix is small), we have no time savings. Moreover, more iterations seem to be required for convergence.

Toy problem dual solution				
Stopping Criterion	Required Tol	Computed Tol	Iterations	Time (s)
$\|\|f_{exact} - f_{y_k}\|\|_2$	1e-14	7.1054e-15	272	0.012607
$\|\|f_{y_{k+1}} - f_{y_k}\|\|_2$	1e-14	0	269	0.028785
$\|\|y_{k+1} - y_k\|\|_2$	1e-14	6.5653e-15	262	0.013769

Table 7: We show the results from the implementation of our toy problem for the dual. No time savings is introduced by using multiblocks. Computation time is averaged over 1000 iterations.

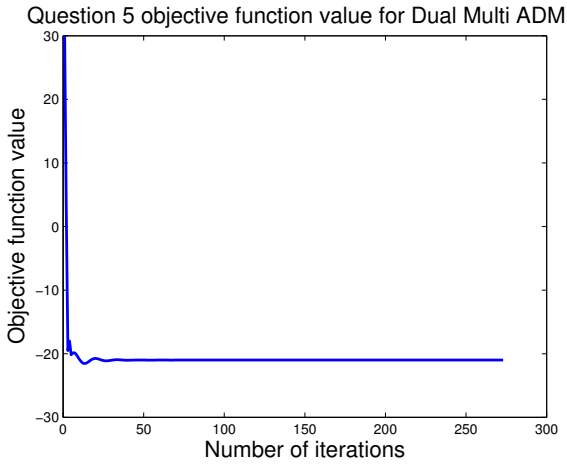

Figure 8: We show the convergence to solution of our multi-block ADMM under the first criteria. For the toy model we have no problem in converging and the convergence of our objective function is similar to the one in question 2.

- Randomly Permuted

  Randomly permuting the order of our update increases the number of iterations required for convergence, while adding noise in our convergence sequence. This behaviour can be seen in fig.9

Toy problem dual solution				
Stopping Criterion	Required Tol	Computed Tol	Iterations	Time (s)
$\|\|f_{exact} - f_{y_k}\|\|_2$	1e-14	7.1054e-15	375	0.023406
$\|\|f_{y_{k+1}} - f_{y_k}\|\|_2$	1e-14	7.1054e-15	415	0.025617
$\|\|y_{k+1} - y_k\|\|_2$	1e-14	4.8928e-15	350	0.022351

Table 8: We show the results from the implementation of our toy problem for the dual. As expected, there is no substantial difference when preconditioning. Computation time is averaged over 1000 iterations.

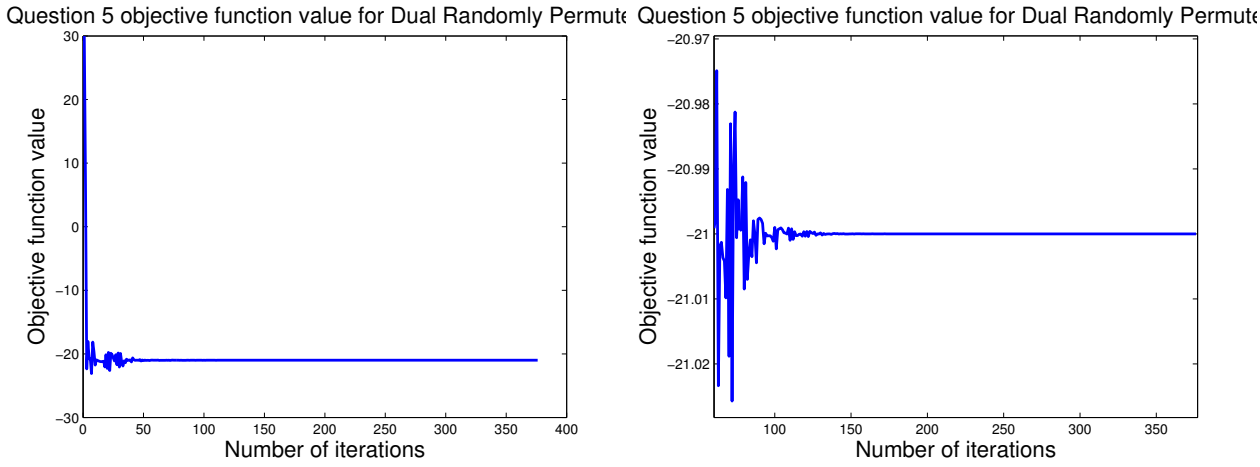

Figure 9: We show the convergence to solution of our randomly permuted multi-block ADMM under the first criteria. Randomly changing which variable to solve first causes wild oscillations that take a number of iterations to die out. These oscillations are characteristic of the process, as clearly shown in the second image, where the same structure can be recognize at smaller scale later in the process

Practical Implementation - Real Problem

Having validated the correctness of our code, we proceed on implementing it on bigger scale problem. From http://www.netlib.org/lp/data/ we proceed to solve the following problems: afiro, adlitlle, blend, sc205. Each problem has an exact solution.
Unless otherwise specified, we will use the following values to solve the relative ADMM problem:

- $\beta = 0.5$
- Stopping Criterion: $||f_{k+1} - f_k||_2$
- Error: $\frac{||f_{exact} - f_k||_2}{|f_{exact}|}$
- Tolerance = 1e-11
- Max-iter = 10^8
- Problem = not pre-condtioned

The fist stopping criterion is chosen for two reasons: to highlighted some of the quirkness of iterative methods, and to lessen the computational requirements. Moreover, all comparison are made on non-preconditioned problems.

Question 1

As expected, by applying ADMM on the primal we observe that it always converges to the exact solution. Unlike with out toy model, preconditioning speeds up our solution considerably, we see gains up to 70% of total computig time when using preconditioning. When comparing the number of iterations needed to reach a solution via stopping criteria number one or two, we see the same effect witnessed with our toy model (**N.B.** not shown here): when the problem is preconditioned, the difference between the number of iterations needed to satisfy either requirement is much less than when not. We can thus infer that preconditioning also regularizes our problem, i.e. changes in our variables will be matched by equivalent changes in the objective function.

Primal Solution						
Problem	Size	Sparsity (%)	Iter	Error (%)	Time (s)	Pre-Cond Iter
afiro	27x51	92.59	1512	4.7337e-10	0.1111	1476
adlittle	56x138	94.51	131785	0.0878	10.0823	103549
blend	74x114	93.81	147667	1.8926e-05	10.5930	32283
sc205	205x317	98.97	1298343	1.6471e-07	78.899630	930398

Table 9: We show the results from the implementation of standard ADMM for a variety of sample problems in their primal form.

Question 2

Applying ADMM on the dual presents risks, as evidenced by the results reported in tab.10. Two of our four problem, afiro and sc205, stop early and seemingly converge to the wrong solution. From theory, we know that two block ADMM always converges, therefore our objective function must be rather flat at the point of stopping. This is indeed the case, limiting ourselves to sc205 for brevity, we show in fig.10 that its objective function is rather flat for numerous iterations, and then jumps towards the exact solution. The slow decreases "tricks" our chosen stopping criteria into delivery the wrong solution.
To obviate this problem we may want to look at different solutions: either changing the value of β (1) or choice of stopping criterion (2).

1. For the dual of sc205, increasing β does indeed alleviate the problems caused by stopping criteria 1. It increases our stepsize and the costs of the constraints on the Lagrangian, converging quickly to the exact solution by avoiding any flat valley. Nonetheless, generally speaking, this choice cannot be made a priori. Augmenting beta such that the dual for sc205 converges, makes the primal not convergent within the maximum number of iterations. This leads us to make a crucial observation regarding the role of our tuning parameter β, if its change in one directions increases the convergence of the primal, it decreases the convergence rate of the dual and vice-versa. There are techniques to optimize the value of β and thus convergence. When problems are not excessively large, we could make use of the duality gap, by simultaneously solving both the primal and dual. Ref.[1] has a detailed algorithm on optimal choice for β using this idea.

2. Imposing the stopping criterion on the changes of the variables, it seems to generally perform better. Unfortunately, the literature has extensively shown this is not necessary true. In addition, a stopping criterion on the changes of variables, for non-preconditioned problem, greatly increases the number of iterations needed to converge (see tab.1).

Dual Solution						
Problem	Size	Sparsity (%)	Iter	Error (%)	Time	Pre-Cond Iter
afiro	51x27	92.59	1264	59.17		1255
adlittle	138x56	94.51	66026	7.8834e-08	3.6737	56079
blend	114x74	93.81	16530	1.2600e-05	1.0631	16530
sc205	317x205	98.97	18	99.7029		18

Table 10: We show the results from the implementation of standard ADMM on the dual of the aforementioned problems. Time is not shown for afiro and sc205 since convergence is not reached

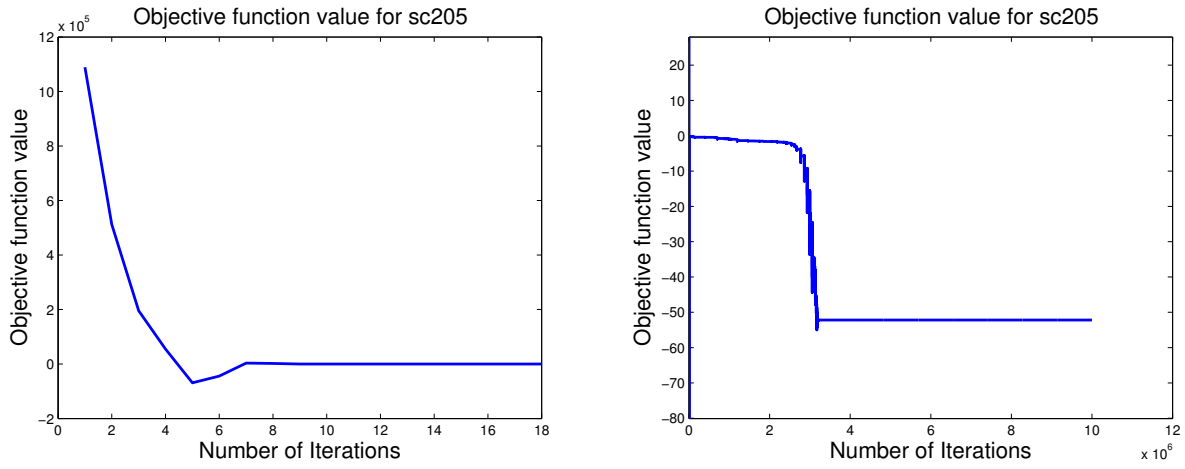

Figure 10: We show the convergence to solution of ADMM for the problem sc205. The figure on the left shows the objective function under testing conditions: it seemingly quickly reaches convergence to the wrong value. The right figure shows what happens when the algorithm is allowed to run until the maximum number of iterations is reached. It clearly shows a slow decay which could be mistaken as a solution by our algorithm, to then jump to the exact and correct one.

Question 3

For brevity, we will investigate ADMM with barrier function only for the primal of afiro, as it is the smallest problem and the one with the fastest convergence rate. Studying the other problems becomes cost prohibitive due to the solver chosen for the interior point method.

We easily and smoothly find a solution close to the exact one, especially thanks to the low coefficient associated with our barrier function. Comparing our results with those found in question 1, we can see that the interior point methods takes fewer iterations than the standard ADMM. Nevertheless, the cost of solving two non-linear minimization problems is significant and is reflected in a high computation time.

Primal Solution ξ= 0.1					
Problem	Tol	Iterations	Error (%)	Time (s)	Pre-Cond Iter
Afiro	1e-3	754	0.0492	178.8527	720

Table 11: We show the results from the implementation of the interior point method on the primal of afiro. It takes fewer iteration to converge to an acceptable result, nonetheless the time taken to reach it is longer.

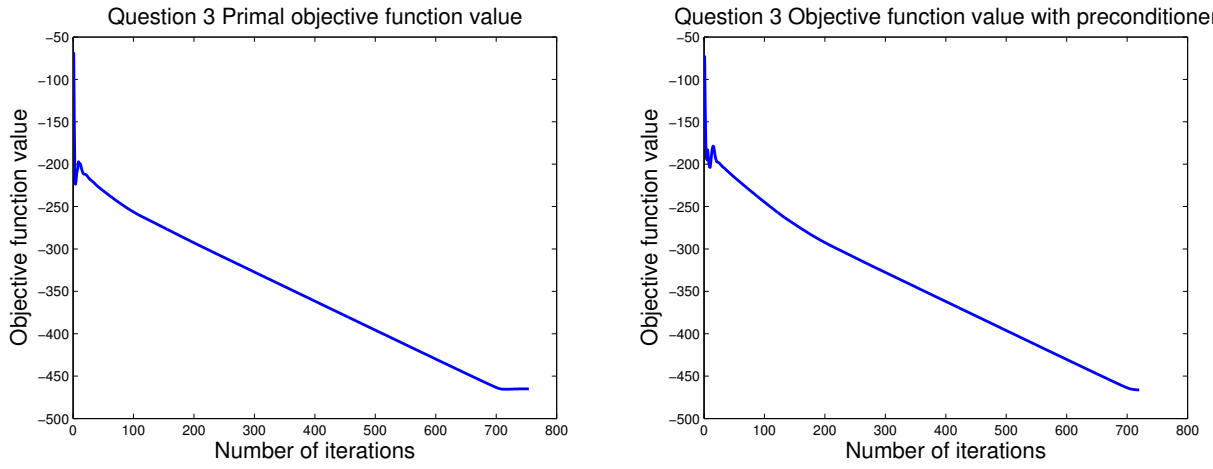

Figure 11: We show the results for the interior point ADMM on the objective function of Afiro. As it can be see from the right picture, the preconditioned problem is better behaved. Just like with our toy model, we see a smooth decrease towards the exact solution for both cases.

Question 4

As expected from what seen in the toy problem, we smoothly converge towards the exact solution. In this situation preconditioning does not influence our results significantly as we are introducing a barrier function which modify the problem and should nullify any positive effect of preconditioning. As before, it takes fewer iteration to converge, nonetheless, due to the Lagrangian not being solved analytically, it takes longer time.

Primal Solution, ξ= 0.9,γ= 0.9					
Problem	Tol	Iterations	Error (%)	Time (s)	Pre-Cond Iter
Afiro	1e-3	727	0.05	120.6948	732

Table 12: We show the results from the implementation of iterative barrier ADMM on the primal of afiro. We see that the required time to converge is significantly less for this methodology than for the prior interior point method.

Question 5

Applying ADMM multi-block presents the same problems outlined in the straight up dual section. Dividing the starting matrix in equal block doesn't seem to speed up the time required to reach a solution, nonetheless it make memory management much better. The same ideas as before can be imposed to ease convergence.

- Multi-Block ADMM

  By using adlittle as our testing problem, we investigate if there is an optimal solution on how to split the matrix A for most efficient computation time. Looking at fig.13 it seems that not splitting is always the most efficient situation as long as we don't run

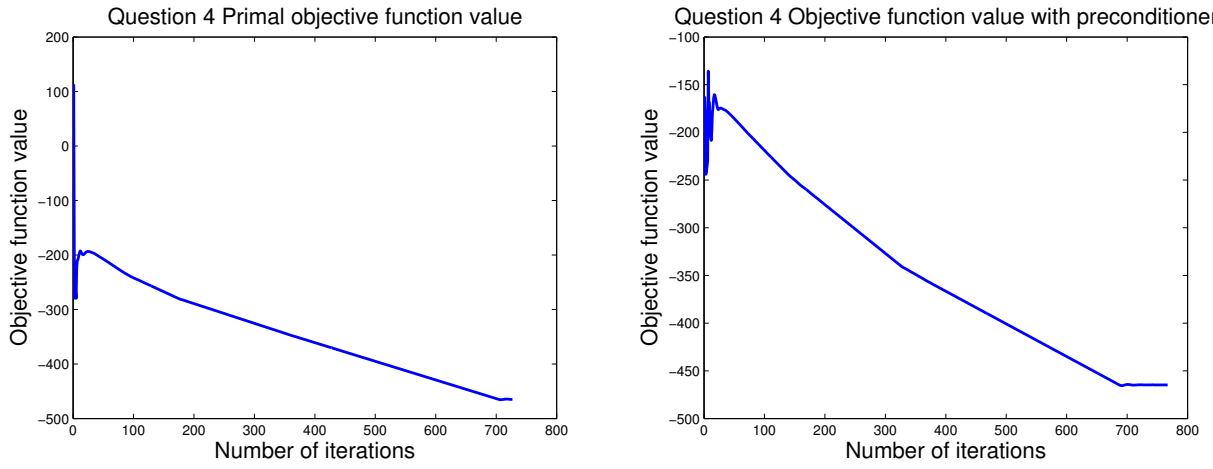

Figure 12: We show the results for barrier ADMM with outer iteration on the objective function of Afiro. Just like with our toy model, we see a smooth decrease towards the exact solution for both cases.

Dual Multi-Block					
Problem	Size	Sparsity (%)	Iter	Error (%)	Time
afiro	51x27	92.59	1660	59.17	
adlittle	138x56	94.51	76356	9.6035e-09	9.4241
blend	114x75	93.81	542707	0.0064	61.8577
sc205	317x205	98.97	644	99.7029	

Table 13: We show the results from the implementation of ADMM multi-block on the dual of the problems considered at the beginning of this section. Time is not shown for afiro and sc205 since convergence is not reached

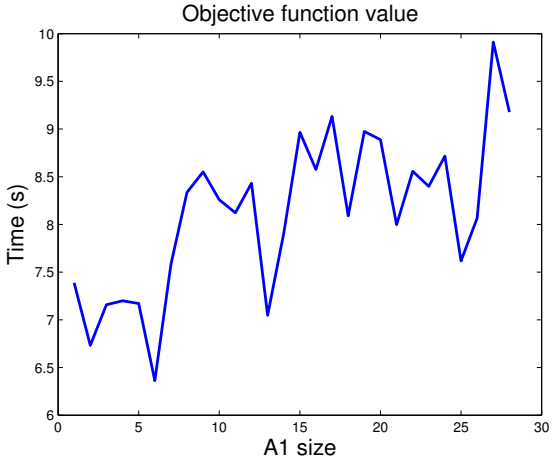

Figure 13: We show the time to convergence for adlittle as A1 rows increase. Conversely the row size of A2 is decreasing.

into memory problems. As A1 size increases (and conversely A2 size decreases) and reaches A2's size, time to solve increases.

- Random Permutation

Dual Multi Block Rand Perm				
Problem	Size	Sparsity (%)	Iter	Error (%)
afiro	51x27	92.59	1660	59.17
adlittle	138x56	94.51	111556	3.3292e-09
blend	114x74	93.81	48	84.2364
sc205	317x205	98.97	795	99.7029

Table 14: We show the results from the implementation of random permutation to ADMM multi-block on the dual of the problems considered at the beginning of this section.

Random permutation, in addition to the same problems as regular two block ADMM, presents a new fascinating one: blend seem not to converge. Let's investigate this situation further: not only does blend not converge, as shown in fig.14 it also seemingly converges to different solutions every time. In these situation, neither decreasing the tolerance nor increasing the value of β alleviate the problem. These event seems to arise when the last updated variable is also the firs new updated one. One simple way to eliminate this aberration lies in changing stopping criterion. We show this in fig.15. Applying criterion 2, we see that — as per theory — the algorithm does indeed converge to the exact solution. Again we notice how random permutation takes longer to converge as it oscillates significantly around the exact solution.

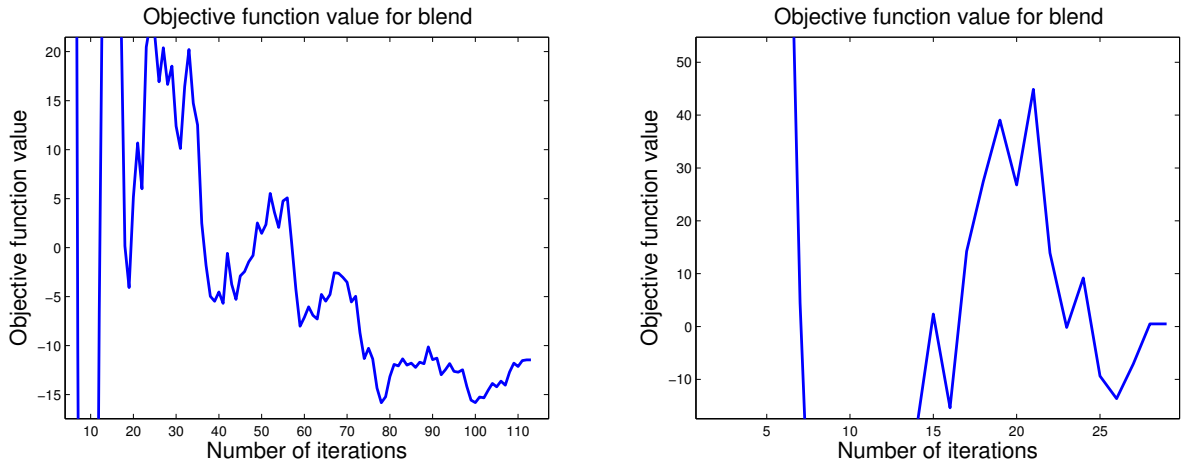

Figure 14: We show the results of different Multi Block ADMM with random permutation runs for blend. The algorithm seem to never converge to the exact solution as demonstrated by the different objective value it stops at.

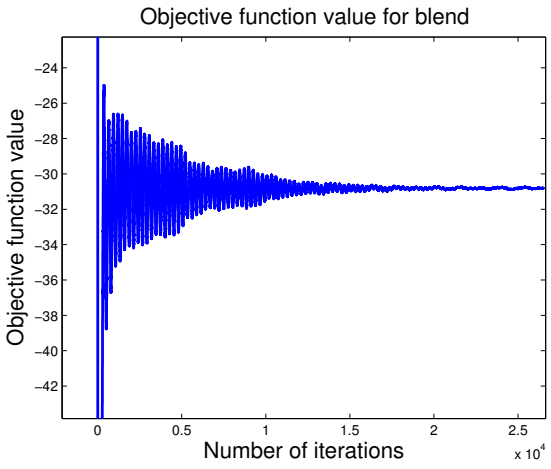

Figure 15: We show the results of using stopping criterion number 2 (i.e. $||y_{k+1} - y_k||_2$) to reach convergence for the Multi Block ADMM with random permutation for blend.

Conclusions

Having used ADMM on a variety of problem and in a variety of situations we recommend the following implementations techniques:

- Always precondition your problem: preconditioning increases speed of convergence and regularizes the problem. I.e. changes in $\mathbf{x}_k$ and f_k are more closely realated, which means that satisfying stopping criterion 1 may is more likely to also satisfy stopping criterion 2
- If unsure on the behavior of your objective function, use a blended tolerance approach for stopping criterion: $\alpha||f_{k+1} - f_k||_2 + (1-\alpha)\,||x_{k+1} - x_k||_2$
  This way you can reach a satisfactory compromise between speed of convergence and accuracy in both objective function and variables.
- When feasible, solve both primal and dual. As shown, it is possible that either one converges to the wrong solution, but it's unlikely that both will (from strong duality you know that can't be the case whenever a feasible optimal solution exists). Ref. [1] recommends a stopping criterion based un the residual of both the primal and dual problem to make use of the aforementioned statement.
- Carefully choose the value of β as it will strongly influence your rate of convergence [1].

References

[1] 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):1–122, 2011.

APPENDIX - Matlab Code
Main

```
clear all
close all

% Setting up system of equations to be solved
% System taken from http://www.ifp.illinois.edu/~angelia/ge330fall09_stform4.
% Pag.11
%
% A=[6  4  1  0 ;
%    1  2  0  1] ;
% b=[24;6];
% c=-[5;4;0;0]; % check sign for plus or minus depending if you are solving a
% load afiromatlab.mat
% load adlittlematlab.mat
% load blendmatlab.mat
% load sc205matlab.mat

% Aprime=sparse([Aineq,eye(size(Aineq,1))]); % This is adding the slack varia
% Asecond=sparse([Aeq,zeros(size(Aeq,1),size(Aineq,1))]); %This is adding the
% A=sparse([Aprime;Asecond]);
% b=sparse([bineq;beq]); % Writing the b vector
% c=sparse([f;zeros(size(Aineq,1),1)]); %Adding slack variable to objective f
beta=0.5;
% Setting beta, tolerance, counters
min_tol = 1e-11;
max_counter = 10000000;
xsi=0.9 ; % How much we need to consider barrier function
gamma = 0.9;


%%
% ================================================================================
%
%              QUESTION 1
%
% ================================================================================
%


%
% General prolbem is the primal with the following form
%   minimize c'*x
%       s.t. Ax1=b
```

```
%              x1-x2=0
%              x2 \ge 0
    temp_no_p=[]
    temp_p=[]
    % for iii = 1:100
    preconditioner = 0;
    temp1=Project4Q1Matlab(A,b,c,beta,min_tol,max_counter,preconditioner); %
    temp_no_p=[temp_no_p,temp1];
    preconditioner = 1;
    temp2=Project4Q1Matlab(A,b,c,beta,min_tol,max_counter,preconditioner); %
    temp_p=[temp_p,temp2];
% end
%%
% ================================================================================
%
%              QUESTION 2
%
% ================================================================================
%
%
% Solving the dual with the following form
%   maximize b'*y
%       s.t. A'y + s=c
%            s \ge 0
temp_no_p=[]
temp_p=[]
% for iii = 1:1000
preconditioner = 0;
temp1=Project4Q2Matlab(A,b,c,beta,min_tol,max_counter,preconditioner)
temp_no_p=[temp_no_p,temp1];
preconditioner = 1;
temp2=Project4Q2Matlab(A,b,c,beta,min_tol,max_counter,preconditioner)
temp_p=[temp_p,temp2];
% end
%%
% ================================================================================
%
%              QUESTION 3
%
% ================================================================================
%
%
% Solving the primal with barrier function
%   minimize c'*x-xsi*(sum(log(x_j)))
%       s.t. Ax1=b
```

```
%               x1-x2=0
%               x2 \ge 0

outer=0;
min_tol_barrier=1e-8;
xsi=0.1
%PRIMAL
% NO PRECONDITIONER
preconditioner = 0;
warning('off')
Project4Q3Q4Primal(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,
%WITH PRECONDITIONER
preconditioner = 1;
warning('off')
Project4Q3Q4Primal(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,

% Solving the dual with barrier function
%   maximize b'*y + xsi*sum(log(x_j))
%       s.t. A'y + s=c
%               s \ge 0
% DUAL
% NO PRECONDITIONER
preconditioner = 0;
warning('off')
Project4Q3Q4Dual(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,ou
% WITH PRECONDITIONER
preconditioner = 1;
warning('off')
Project4Q3Q4Dual(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,ou
%%
% ========================================================================
%
%               QUESTION 4
%
% ========================================================================
%
%
% Solving the primal with barrier function with outer iteration
%   minimize c'*x-xsi*(sum(log(x_j)))
%       s.t. Ax1=b
%               x1-x2=0
%               x2 \ge 0

outer=1;
xsi=0.9;
```

```
gamma=0.9;
min_tol_barrier=1e-8;
counter_all=[];
beta=0.5;
for beta =0.01:0.01:0.91
%PRIMAL
% NO PRECONDITIONER
preconditioner = 0;
warning('off')
[counter]=Project4Q3Q4Primal(A,b,c,beta,min_tol_barrier,max_counter,precondit
counter_all=[counter_all,counter];
end
% PRIMAL WITH PRECONDITIONER
preconditioner = 1;
warning('off')
Project4Q3Q4Primal(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,

% DUAL
% NO PRECONDITIONER
preconditioner = 0;
warning('off')
Project4Q3Q4Dual(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,ou
% WITH PRECONDITIONER
preconditioner = 1;
warning('off')
Project4Q3Q4Dual(A,b,c,beta,min_tol_barrier,max_counter,preconditioner,xsi,ou

%%
% ========================================================================
%
%              QUESTION 5
%
% ========================================================================
%
%
% Solving multi-block dual
%   maximize b1'*y1+b2*y2
%       s.t. A1'y1+A2'y2+s=c
%            s \ge 0
% total_time=[];
% for iii=1:round(size(A,1)/2)
    A1=A(1:iii,:);
    A2=A(iii+1:end,:);
    b1=b(1:iii,1);
    b2=b(iii+1:end,1);
```

```
    [tempo]=Project4Q5Dual(A1,A2,b1,b2,c,beta,min_tol,max_counter)
%       total_time=[total_time,tempo];
% end

%               QUESTION 5 - RANDOM PERMUTATION
%

Project4Q5DualRandPerm(A1,A2,b1,b2,c,beta,min_tol,max_counter)
```

Q1

```
function [temporale]=Project4Q1Matlab(A,b,c,beta,min_tol,max_counter,precondi
% close all

% Initialization of your starting vectors
mu=ones(size(A,2),1); %initializiation of lagrange multiplier for (x1-x2)
lambda=ones(size(A,1),1); %initilization of lagrange multiplier for (Ax-b)
x1=ones(size(A,2),1);
x2=ones(size(A,2),1);
tol=100; % initial tolerance
counter=0;
fnew=[]; %creation of vector containing result of objective function

s = sprintf('Question 1 objective function value');
% Do you want to precondition your system;
if preconditioner == 1
    b=(full(A)*full(A'))^(-1/2)*b;
    A=(full(A)*full(A'))^(-1/2)*A;
    s = sprintf('Question 1 objective function value with preconditioner');
end

% fprintf( 'A ', A,'\n b ',b,'\n c ',c,' \n mu ',mu,'\n lambda ',lambda,'\n b

temp_inv = (A'*A+eye(size(A'*A)))\eye(size(x2,1));
cond((A'*A+eye(size(A'*A))))
% zzznorm(temp_inv-inv(A'*A+eye(size(A'*A))))
% pause
tic
while tol>=min_tol && counter<=max_counter
    x_old=x1;
    counter=counter+1;
    fold=c'*x1;

    % Analytical solution for x1_k+1
    x1_k_1= temp_inv*(x2+A'*b+1/beta*(A'*lambda+mu-c));
```

```
    % Analytical solution for x2_k+1
    x2_k_1 =max([zeros(size(A,2),1),x1_k_1-mu/beta],[],2);

    % Updating multipliers

    lambda = lambda - beta*(A*x1_k_1-b);
    mu=mu-beta*(x1_k_1-x2_k_1);
    x2=x2_k_1;
    x1=x1_k_1;
    fnew=[fnew,c'*x1];
    tol=norm(fnew(end)-fold,2);
%      tol=norm(x1-x_old,1);
end
temporale=toc;
% figure
% plot(fnew,'LineWidth',2)

% title(s,'fontsize',16)
% xlabel('Number of iterations','fontsize',16)
% ylabel('Objective function value','fontsize',16)
fprintf('\n Question 1 \n')
fprintf('We needed %d iterations to obtain the following solution: %15.10f \n
tol
%  x2,lambda
fprintf('\n Taking this much time %f \n', temporale)

%Testing solution is correct
% solution=linprog(c,-eye(size(A,2),size(A,2)),zeros(size(A,2),1),A,b) %-ey;e
```

Q2

```
function [temporale]=Project4Q2Matlab(A,b,c,beta,min_tol,max_counter,precondi

% Initialization of your starting vectors
y=-ones(size(A,1),1);
s=ones(size(A,2),1);
lambda=ones(size(A,2),1);
tol=100; % initial tolerance
counter=0;
fnew=[]; %creation of vector containing result of objective function


% Do you want to precondition your system
testo = sprintf('Question 2 Dual objective function value');
% Do you want to precondition your system;
```

```
if preconditioner == 1
    b=(full(A)*full(A'))^(-1/2)*b;
    A=(full(A)*full(A'))^(-1/2)*A;
    testo = sprintf('Question 2 Dual objective function value with preconditi
end

temp_inv=(A*A')\eye(size(y,1));
cond(temp_inv)

% fprintf( 'A ', A,'\n b ',b,'\n c ',c,' \n mu ',mu,'\n lambda ',lambda,'\n b
tic
while tol>=min_tol && counter<= max_counter

    y_old=y;
    counter=counter+1;
    fold=b'*y;

%     % Analytical solution for y_k+1
    y_k_1 = temp_inv*(A*c-A*s+1/beta*(b+A*lambda));


     % Solution via fmincon for lagrangian for general problem
%     options=optimset('fminunc');options.TolFun = 1e-10;options.TolX = 1e-10
%     myfuny=@(y)yLagMinimizerDual(y,A,b,c,lambda,beta,s);
%     y_k_1=fminunc(myfuny,y,options);
%
%     % Analytical solution for s_k+1
    s_k_1 = max([zeros(size(A,2),1),c-A'*y_k_1+lambda/beta],[],2);


    % Solution via fmincon for lagrangian for general problem
%     myfuns=@(s)sLagMinimizerDual(y_k_1,A,b,c,lambda,beta,s);
%     options=optimset('fmincon');options.TolFun = 1e-10;options.TolX = 1e-10
%     s_k_1=fmincon(myfuns,s,[],[],[],[],zeros(size(A,2),1),[],[],options);

    % Updating multipliers
    lambda = lambda - beta*(A'*y_k_1+s_k_1-c);
    s=s_k_1;
    y=y_k_1;
    fnew=[fnew,b'*y];
    fnew(end);
%     tol=norm(fnew(end)-fold,1);
    tol=norm(y-y_old,2);
%     pause
end
```

```
temporale=toc;
figure
plot(fnew,'LineWidth',2)
title(testo,'fontsize',16)
tol
xlabel('Number of iterations','fontsize',16)
ylabel('Objective function value','fontsize',16)
fprintf('\n Question 2 \n')
fprintf('We needed %d iterations to obtain the following solution: %15.10f \n
% y
fprintf('\n Taking this much time %f \n', temporale)
% s

%Solution via linpro
%[x,fval,exitflag,output] = linprog([-Newb;zeros(size(NewA,2),1)],[],[],[NewA
```

Q3Q4 Primal

```
function [counter]=Project4Q3Q4Primal(A,b,c,beta,min_tol,max_counter,precondi
preconditioner

% Initialization of your starting vectors
mu=ones(size(A,2),1); %initializiation of lagrange multiplier for (x1-x2)
lambda=ones(size(A,1),1); %initilization of lagrange multiplier for (Ax-b)
x1=ones(size(A,2),1);
x2=ones(size(A,2),1);
tol=100; % initial tolerance
counter=0;
fnew=[]; %creation of vector containing result of objective function

%
domanda = ('Question 3');
if outer == 1
    domanda = sprintf('Question 4');
end

testo = [domanda,sprintf(' Primal objective function value')];
% Do you want to precondition your system;
if preconditioner == 1
    b=(full(A*A'))^(-1/2)*b;
    A=(full(A*A'))^(-1/2)*A;
    testo = [domanda,sprintf(' Objective function value with preconditioner')
end
tic
while tol>=1e-7 && counter<max_counter
    x_old=x1;
```

```
    counter=counter+1;
    fold=c'*x1-xsi*sum(log(x1));

%     Solution via fmincon for lagrangian for general problem
    myfunx1=@(x1)x1LagMinimizerPrimalBarrier(x1,A,b,c,mu,lambda,beta,xsi,x2);
    options=optimset('fmincon');options.TolFun = 1e-10;options.TolX = 1e-10;o
    x1_k_1 = fmincon(myfunx1,x1,[],[],[],[],0.00001*ones(length(A),1),[],[],o
    myfunx2=@(x2)x2LagMinimizerPrimalBarrier(x1_k_1,A,b,c,mu,lambda,beta,xsi,
    options=optimset('fmincon');options.TolFun = 1e-10;options.TolX = 1e-10;o
    x2_k_1 = fmincon(myfunx2,x2,[],[],[],[],0.00001*ones(length(A),1),[],[],o

    % Updating multipliers

    lambda = lambda - beta*(A*x1_k_1-b);
    mu=mu-beta*(x1_k_1-x2_k_1);
    x2=x2_k_1;
    x1=x1_k_1;
     fnew=[fnew,c'*x1-xsi*sum(log(x1))];
     fnew(end);
    tol=norm(fnew(end)+21);
%     tol=norm(x1-x_old);
    if outer ==1 % Reduction algorithm outer iteration
        xsi=gamma*xsi;
    end
end
toc
% figure
plot(real(fnew),'LineWidth',2)

title(testo,'fontsize',16)
xlabel('Number of iterations','fontsize',16)
ylabel('Objective function value','fontsize',16)
fprintf(['\n', domanda,' Primal \n'])
fprintf('We needed %d iterations to obtain the following solution: %15.10f',
% x2
tol
% error = norm(fnew(end)+21)
%Testing solution is correct
% xsi=0;
% solution = fmincon(@(x) c'*x-xsi*sum(log(x)),ones(size(A,2),1),[],[],A,b,0.0
%
```

Q3Q4 Dual

```
function []=Project4Q3Q4Dual(A,b,c,beta,min_tol,max_counter,preconditioner,xs
preconditioner
```

```
% Initialization of your starting vectors
y=ones(size(A,1),1);
s=ones(size(A,2),1);
lambda=ones(size(A,2),1);
tol=100; % initial tolerance
counter=0;
fnew=[]; %creation of vector containing result of objective function

%
domanda = ('Question 3');
if outer == 1
    domanda = sprintf('Question 4');
end

testo = [domanda,sprintf(' Dual objective function value')];
% Do you want to precondition your system;
if preconditioner == 1
    b=(A*A')^(-1/2)*b;
    A=(A*A')^(-1/2)*A;
    testo = [domanda,sprintf(' Dual function value with preconditioner')];
end
%

% fprintf( 'A ', A,'\n b ',b,'\n c ',c,' \n mu ',mu,'\n lambda ',lambda,'\n b
fnew=[fnew,b'*y+xsi*sum(log(s))];
tic
while tol>=min_tol && counter <= max_counter
    counter=counter+1
    y_old=y;

    % Solution via fmincon for lagrangian for general problem
    options=optimset('fminunc');options.TolFun = 1e-10;options.TolX = 1e-10;o
    myfuny=@(y)yLagMinimizerDualQ3(y,A,b,c,lambda,beta,s,xsi);
    y_k_1=fminunc(myfuny,y,options);

    % Solution via fmincon for lagrangian for general problem
    myfuns=@(s)sLagMinimizerDualQ3(y_k_1,A,b,c,lambda,beta,s,xsi);
    options=optimset('fmincon');options.TolFun = 1e-10;options.TolX = 1e-10;o
    s_k_1=fmincon(myfuns,s,[],[],[],[],0.00001*ones(size(A,2),1),[],[],option

    % Updating multipliers
    lambda = lambda - beta*(A'*y_k_1+s_k_1-c);
    s=s_k_1;
    y=y_k_1;
    fnew=[fnew,b'*y+xsi*sum(log(s))];
```

```
    fnew(end)
%       tol=abs((-4.6475314286e02+fnew(end)))
%       tol=norm(fnew(end)-fnew(end-1),2)
    tol=norm(y_old-y);
    if outer ==1
        xsi=gamma*xsi;
    end
%       pause
end
toc
figure
plot(real(fnew),'LineWidth',2)
title(testo,'fontsize',16)
xlabel('Number of iterations','fontsize',16)
ylabel('Objective function value','fontsize',16)
fprintf([domanda, 'Dual'])
fprintf('We needed %d iterations to obtain the following solution: %15.10f',
y
tol
norm(fnew(end)-fnew(end-1))
```

Q5

```
function [temporale]=Project4Q5Dual(A1,A2,b1,b2,c,beta,min_tol,max_counter)
% Initialization of your starting vectors
y1=ones(size(A1,1),1);
y2=ones(size(A2,1),1);
s=ones(size(A1,2),1);
lambda=ones(size(A1,2),1);

tol=100; % initial tolerance
counter=0;
fnew=[]; %creation of vector containing result of objective function
y_old=[y1;y2];
% fprintf( 'A ', A,'\n b ',b,'\n c ',c,' \n mu ',mu,'\n lambda ',lambda,'\n b
fnew=[fnew,b1'*y1+b2'*y2];
temp_inv1=(A1*A1')\eye(size(y1,1));
temp_inv2=(A2*A2')\eye(size(y2,1));
tic
while tol>=min_tol && counter <=max_counter
    counter=counter+1;

    % Analytical solution for x1_k+1
    y1_k_1 = temp_inv1*(A1*c-A1*s-A1*A2'*y2+1/beta*(b1+A1*lambda));

    y2_k_1 = temp_inv2*(A2*c-A2*s-A2*A1'*y1_k_1+1/beta*(b2+A2*lambda));
```

```
    s_k_1   = max([zeros(length(lambda),1),lambda/beta+c-A1'*y1_k_1-A2'*y2_k_1

    % Updating multipliers
    lambda = lambda - beta*(A1'*y1_k_1+A2'*y2_k_1+s_k_1-c);
    s=s_k_1;
    y1=y1_k_1;
    y2=y2_k_1;
    fnew=[fnew,b1'*y1+b2'*y2];
    tol=norm(fnew(end)-fnew(end-1),2);
%       tol=norm([y1;y2]-y_old,2);
    y_old=[y1;y2];
%       pause
end
temporale=toc;
% figure
% plot(real(fnew),'LineWidth',2)
% title('Question 5 objective function value for Dual Multi ADMM','fontsize',
% xlabel('Number of iterations','fontsize',16)
% ylabel('Objective function value','fontsize',16)
% fprintf('Question 5 Dual1 \n')
fprintf('We needed %d iterations to obtain the following solution: %15.10f \n
% tol
% y1,y2
% printf('\n With value: %f \n',fnew(end))
fprintf('\n Taking this much time %f \n', temporale)
```