

Alternate method of multipliers

Sivamaran M.A.C





### Acknowledgement

The success and final outcome of this assignment required a lot of guidance and assistance from many people and I am extremely fortunate to have got this support along the completion of our assignment work

I respect and thank Dr. Soman K.P for giving me the opportunity to do this project and providing me all support and guidance required to complete the project.

### Abstract

The Alternating Direction Method of Multipliers (ADMM) has recently received lots of application as a powerful algorithm to solve convex problems due to its simplicity, fast convergence, and easily decomposable structure. It is vastly applied in optimization problems. ADMM applies to a more general class of convex and nonsmoothed objective functions, beyond the smooth and separable least squares loss function. We will optimize the following cost functions using ADMM: Basis pursuit, LASSO, LAD, Huber loss, Linear programming and quadratic programming.

Table of Contents

[Acknowledgement 2](#_Toc43211988)

[Abstract 3](#_Toc43211989)

[Introduction 5](#_Toc43211990)

[Lagrangian, dual decomposition and Method of multipliers 6](#_Toc43211991)

[Dual decomposition 7](#_Toc43211992)

[Method of Multipliers 8](#_Toc43211993)

[The Alternating Method of Multipliers (ADMM) 9](#_Toc43211994)

[Drawbacks 10](#_Toc43211995)

[Basis pursuit 10](#_Toc43211996)

[LASSO (Least Absolute shrinkage and Selection Operator): 11](#_Toc43211997)

[Huber Loss: 13](#_Toc43211998)

[Least Absolute Deviations (LAD): 15](#_Toc43211999)

[Quadratic Programming: 16](#_Toc43212000)

[Linear Programming: 17](#_Toc43212001)

[MATLAB code 18](#_Toc43212002)

### Introduction

In all applied fields, it is now common to approach problems through tools such as data analysis, statistics and machine learning algorithms on often very large datasets. In industry, this trend has been referred to as ‘Big Data’, and it has had a significant impact in areas as varied as artificial intelligence, internet applications, computational biology, medicine, finance, marketing, journalism, network analysis, and logistics.

First, the datasets are often extremely large, consisting of hundreds of millions or billions of training examples; second, the data is often very high-dimensional, because it is now possible to measure and store very detailed information about each example; and third, because of the large scale of many applications, the data is often stored or even collected in a distributed manner. As a result, it has become of central importance to develop algorithms that are both rich enough to capture the complexity of modern data, and scalable enough to process huge datasets in a parallelized or fully decentralized fashion. Indeed, some researchers have suggested that even highly complex and structured problems may succumb most easily to relatively simple models trained on vast datasets.

Many such problems can be posed in the framework of convex optimization. Given the significant work on decomposition methods and decentralized algorithms in the optimization community, it is natural to look to parallel optimization algorithms as a mechanism for solving large-scale statistical tasks. This approach also has the benefit that one algorithm could be flexible enough to solve many problems.

Alternating direction method of multipliers (ADMM), a simple but powerful algorithm that is well suited to distributed convex optimization .ADMM was first developed in the 1970s, with roots in the 1950s, and is equivalent or closely related to many other algorithms, such as dual decomposition, the method of multipliers, Douglas-Rachford splitting, Spingarn's method of partial inverses, Dykstra's alternating projections, proximal methods, and others. Many problems of recent interest in statistics and machine learning can be posed in the framework of convex optimization. The generalization of ADMM’s usage is in solving convex optimization problems where the data can be arbitrarily large. In such extreme cases, the traditional techniques for minimization may be too slow, despite how fast they may be on normal sized problems. Using ADMM the optimization would be decentralized. This optimization technique is robust and splits the problem into smaller pieces that can be optimized in parallel. ADMM extends the method of multipliers in such a way that we get back some of the decomposability (i.e. ability to parallelize) of standard dual ascent algorithms. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization. It also gives us a flexible framework for incorporating many types of convex constraints. We will first give some background on ADMM, then describe how it works, with a MATLAB code implementation of how it is used to solve problems in practice.

### Lagrangian, dual decomposition and Method of multipliers

Consider the following equality-constrained convex optimization problem:

This is referred to as the primal problem (for a primal function f) and x is referred to as the primal variable. To help us solve this, we formulate a different problem using the Lagrangian and solve that. The Lagrangian is defined as:



We call the dual function  and the dual problem , where y is the dual variable. With this formulation, we can recover , where  represents the optimum value of x; f’s minimizer. The solution can be acquired through the Dual Ascend method (DAM). The variables are iterated until convergence. Characterization at iteration k is given as:



Here, α is a step size for the iteration k (commonly used in many optimization algorithms such as gradient descent)and we note that ∇g(y (k) ) = Ax\* − b, and . If g is differentiable, this algorithm strictly converges and seeks out the gradient of g. If g is not differentiable, then we do not have monotone convergence and the algorithm seeks out the negative of a sub-gradient of −g. Note that the term  acts as a penalty function that guarantees minimization occurs on the given constraint.

### Dual decomposition

For high-dimensional input we parallelize DAM for better performance. Suppose that our objective is separable; i.e. , and . Then we can say the same for the Lagrangian. We have: , where . Thus, our x-minimization step in the DAM is split into n separate minimizations that can be carried out in parallel. Our x-minimization step in the DAM is split into n separate minimizations that can be carried out in parallel:





### Method of Multipliers

Augmented Lagrangian methods were developed in part to bring robustness to the dual ascent method, and in particular, to yield convergence without assumptions like strict convexity or finiteness of f. we swap the Lagrangian for an Augmented Lagrangian:



ρ is the penalization parameter. With ρ=0 we have L0 which is the standard langrangian for the problem. Through un-augmented langrangian it can be viewed as:



Note the addition of another penalty term that penalizes straying too far from the constraint during minimization over the length of ρ. Now our iteration computes until convergence:



Here, ρ is the dual update step length, chosen to be the same as the penalty coefficient ρ



At each iteration k,  , we can arrive at the solution analytically by solving augmented lagrangian as following:



As the method of multipliers proceeds, the primal residual Axk+1 − b converges to zero, yielding optimality. The greatly improved convergence properties of the method of multipliers over dual ascent comes at a cost. When f is separable, the augmented Lagrangian Lρ is not separable, so the x-minimization step cannot be carried out separately in parallel for each xi. This means that the basic method of multipliers cannot be used for decomposition. We will see how to address this issue next. This is where ADMM steps in.

### The Alternating Method of Multipliers (ADMM)

ADMM is a hybrid algorithm which utilizes decomposability of dual ascent and uses lagrangian multiplier. Its convergence rate is superior to method of multipliers as it combines the advantages of dual ascent and Method of multiplier. Let us the take the following example:



we form the augmented Lagrangian.



Iterative step for ADMM:  


ρ > 0.

where ρ > 0. Being very similar to dual ascent and the method of multipliers, the algorithm consists of an x-minimization step, a z-minimization step and a dual variable update. As in the method of multipliers, the dual variable update uses a step size equal to the augmented Lagrangian parameter ρ. The algorithm works such that we do not explicitly require to objective function; it only requires the constraint variables (A, B, and c), minimizing functions argminX and argminZ, and a stopping condition.

### Drawbacks

* It converges very fast initially, but tends to slow down as it

approaches the solution.

* It is generally not as fast as traditional convex optimization methods. With careful choices of step size ρ, however, it can be competitive. The issue then lies in how to choose the step size. This may require specialized knowledge about each problem being solved

### Basis pursuit

Basis pursuit is the equality-constrained l1 minimization problem. It is used for finding the sparsest solution to an underdetermined system of equations Ax = b is to solve. It is applied in modern statistical signal processing.



ADMM form,



Where f is a indicator function of {x ∈ Rn | Ax = b}.

The iterative step for ADMM is:



where Π is projection onto {x ∈ Rn | Ax = b}.

The x-update, which explicitly can be written as:  


We can interpret ADMM for basis pursuit as reducing the solution of a least l1 norm problem to solving a sequence of minimum Euclidean norm problems. For a discussion of similar algorithms for related problems in image processing.

### LASSO (Least Absolute shrinkage and Selection Operator):

It is an l1 regularized linear regression that involves solving,



where λ > 0 is a scalar regularization parameter that is usually chosen by cross-validation. The lasso has been widely applied, particularly in the analysis of biological data, where only a small fraction of a huge number of possible factors are actually predictive of some outcome of interest.

ADMM form:









Iterative steps for ADMM:



Note that (AT A + ρI) is always invertible, since ρ > 0. The x-update is essentially a ridge regression (i.e., quadratically regularized least squares) computation, so ADMM can be interpreted as a method for solving the lasso problem by iteratively carrying out ridge regression.

### Huber Loss:

Huber fitting is a function that includes least squares and least absolute deviations. As in for the deviation from target in the range -1 to 1 it is least score loss. Beyond that, loss increases linearly with deviation.



This can be put into ADMM form since, the ADMM algorithm is the same except that the z-update involves the proximity operator of the Huber function rather than that of the l1 norm:



ADMM formulation:











Where S is a shrinkage function.

Iterative steps for ADMM:



### Least Absolute Deviations (LAD):

A simple variant on least squares fitting is least absolute deviations, in which we minimize . Least absolute deviations provides a more robust fit than least squares when the data contains large outliers, and has been used extensively in statistics and econometrics.



Iterative steps for ADMM:



You can refer to Huber loss for deriving the update for x.

Instead of naively solving this system, we use a more efficient approach by finding the Cholesky decomposition RRT=ATA, where R and RT are lower and upper triangular matrices respectively. Due to R’s triangularity we introduce a y such that  and then finding our minimizing x such that RTx=y.

### Quadratic Programming:

The standard form of Quadratic programming:



When P = 0, this reduces to the standard form linear program (LP).

ADMM form:



Where 





Iterative steps for ADMM:



### Linear Programming:

The standard form of linear programming:



ADMM form:



Iterative steps for ADMM:



### MATLAB code

#### Basis pursuit

% basis\_pursuit Solve basis pursuit via ADMM

%

% [x, history] = basis\_pursuit(A, b, rho, alpha)

%

% Solves the following problem via ADMM:

%

% minimize ||x||\_1

% subject to Ax = b

%

% The solution is returned in the vector x.

% rho is the augmented Lagrangian parameter.

rand('seed', 0);

randn('seed', 0);

n = 10;

m = 6;

A = randn(m,n);

x = sprandn(n, 1, 0.1\*n);

b = A\*x;

xtrue = x;

%calling basis\_pusuit function

[x z u r\_norm s\_norm g obj] = basis\_pursuit(A, b, 1.0, 1.0)

% for printing 'x^(k+1)','z^(k+1)','u^(k+1)'

fprintf('%20s\t%10s\t%10s\t \n', 'x^(k+1)','z^(k+1)','u^(k+1)');

for i =1:length(x)

fprintf('%20f\t%10.4f\t%10.4f\t \n', x(i),z(i),u(i));

end

figure

%plotting difference between current z and old z

subplot(2,1,1)

plot(1:g,r\_norm)

ylabel('||r||\_2'); xlabel('iter (k)');

%plotting difference between z and x

subplot(2,1,2)

plot(1:g,s\_norm)

ylabel('||s||\_2'); xlabel('iter (k)');

figure

%plotting norm with iteration

plot(1:g,obj)

ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');

function [X\_hat z u r\_norm s\_norm g obj] = basis\_pursuit(A, b, rho, alpha)

MAX\_ITER = 1000;

[m n] = size(A);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k = 1:MAX\_ITER

%giving tolerance value

RELTOL = 1e-4;

% z-update with relaxation

zold = z;

% x-update

x = shrinkage(zold - u, 1/rho);

X\_hat = alpha\*x +(1-alpha)\*zold;

z= (X\_hat+ u) + pinv(A)\*(b - A\*(X\_hat+u));

u = u + (X\_hat - z);

%norm of the difference between x and z

r\_norm(k) = norm(X\_hat - z);

%norm of the difference between zold and z

s\_norm(k) = norm(-rho\*(z - zold));

%norm of z

obj(k)=norm(z,1)

%if the difference is less than tolerance,then it will break the loop

if(z-zold<RELTOL )

g=k

break;

end

end

function y = shrinkage(a, kappa)

y = max(0, a-kappa) - max(0, -a-kappa);

end

end

#### LASSO

% lasso Solve lasso problem via ADMM

% [z, history] = lasso(A, b, lambda, rho, alpha);

% Solves the following problem via ADMM:

% minimize 1/2\*|| Ax - b ||\_2^2 + \lambda || x ||\_1

% The solution is returned in the vector x.

% rho is the augmented Lagrangian parameter.

% alpha is the over-relaxation parameter (typical values for alpha are

% between 1.0 and 1.8).

randn('seed', 0);

rand('seed',0);

m = 1500; % number of examples

n = 5000; % number of features

p = 10/n; % sparsity density

x0 = sprandn(n,1,p);

A = randn(m,n);

A = A\*spdiags(1./sqrt(sum(A.^2))',0,n,n); % normalize columns

b = A\*x0 + sqrt(0.001)\*randn(m,1);

lambda\_max = norm( A'\*b, 'inf' );

lambda = 0.1\*lambda\_max;

%calling basis\_pusuit function

[x z u r\_norm s\_norm g obj] = lasso(A, b, lambda, 1.0, 1.0);

% for printing 'x^(k+1)','z^(k+1)','u^(k+1)'

fprintf('%20s\t%10s\t%10s\t \n', 'x^(k+1)','z^(k+1)','u^(k+1)');

for i =1:length(x)

fprintf('%20f\t%10.4f\t%10.4f\t \n', x(i),z(i),u(i));

end

figure

%plotting difference between current z and old z

subplot(2,1,1)

plot(1:g,r\_norm)

ylabel('||r||\_2'); xlabel('iter (k)');

%plotting difference between z and x

subplot(2,1,2)

plot(1:g,s\_norm)

ylabel('||s||\_2'); xlabel('iter (k)');

figure

%plotting norm with iteration

plot(1:g,obj)

ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');

function [X\_hat z u r\_norm s\_norm g obj] = lasso(A, b,lambda, rho, alpha)

MAX\_ITER = 100;

[m n] = size(A);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k = 1:MAX\_ITER

%giving tolerance value

RELTOL = 1e-4;

% z-update with relaxation

zold = z;

% x-update

x= pinv(A'\*A+(rho)\*speye(n)) \*( A'\*b+(rho)\*(zold-u));

X\_hat = alpha\*x +(1-alpha)\*zold;

z = shrinkage(X\_hat + u,lambda);

u = u + (X\_hat - z);

%norm of the difference between x and z

r\_norm(k) = norm(X\_hat - z);

%norm of the difference between zold and z

s\_norm(k) = norm(-rho\*(z - zold));

%norm of z

obj(k)=norm(z,1)

%if the difference is less than tolerance

g=k

if(z-zold<RELTOL )

g=k

break;

end

end

function y = shrinkage(a, kappa)

y = max(0, a-kappa) - max(0, -a-kappa);

end

end

#### Huber Loss

% Huber Loss via ADMM

%

% [x, history] = basis\_pursuit(A, b, rho, alpha)

%

% Solves the following problem via ADMM:

%

% minimize ||x||\_1

% subject to Ax = b

%

% The solution is returned in the vector x.

% rho is the augmented Lagrangian parameter.

randn('seed', 0);

rand('seed',0);

m = 5000; % number of examples

n = 200; % number of features

x0 = randn(n,1);

A = randn(m,n);

A = A\*spdiags(1./norms(A)',0,n,n); % normalize columns

b = A\*x0 + sqrt(0.01)\*randn(m,1);

b = b + 10\*sprand(m,1,200/m);

%calling basis\_pusuit function

[x z u r\_norm s\_norm g obj] = lasso(A, b, 1.0, 1.0);

% for printing 'x^(k+1)','z^(k+1)','u^(k+1)'

fprintf('%20s\t%10s\t%10s\t \n', 'x^(k+1)','z^(k+1)','u^(k+1)');

for i =1:length(x)

fprintf('%20f\t%10.4f\t%10.4f\t \n', x(i),z(i),u(i));

end

figure

%plotting difference between current z and old z

subplot(2,1,1)

plot(1:g,r\_norm)

ylabel('||r||\_2'); xlabel('iter (k)');

%plotting difference between z and x

subplot(2,1,2)

plot(1:g,s\_norm)

ylabel('||s||\_2'); xlabel('iter (k)');

figure

%plotting norm with iteration

plot(1:g,obj)

ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');

function [Ax\_hat z u r\_norm s\_norm g obj] = lasso(A, b, rho, alpha)

QUIET = 0;

MAX\_ITER = 1000;

ABSTOL = 1e-4;

RELTOL = 1e-2;

[m, n] = size(A);

Atb = A'\*b;

[L U] = factor(A);

[m n] = size(A);

x = zeros(n,1);

z = zeros(m,1);

u = zeros(m,1);

for k = 1:MAX\_ITER

%giving tolerance value

RELTOL = 1e-4;

% x-update

q = Atb + A'\*(z - u);

x = U \ (L \ q);

% z-update with relaxation

zold = z;

Ax\_hat = alpha\*A\*x + (1-alpha)\*(zold + b);

tmp = Ax\_hat - b + u;

z = rho/(1 + rho)\*tmp + 1/(1 + rho)\*shrinkage(tmp, 1 + 1/rho);

u = u + (Ax\_hat - z - b);

% diagnostics, reporting, termination checks

obj(k) = ( 1/2\*sum(huber(z)) );;

r\_norm(k) = norm(A\*x - z - b);

s\_norm(k) = norm(-rho\*A'\*(z - zold));

g=k

if(z-zold<RELTOL )

g=k

break;

end

end

function z = shrinkage(x, kappa)

z = pos(1 - kappa./abs(x)).\*x;

end

function [L U] = factor(A)

[m, n] = size(A);

if ( m >= n ) % if skinny

L = chol( A'\*A, 'lower' );

end

% force matlab to recognize the upper / lower triangular structure

L = sparse(L);

U = sparse(L');

end

end

#### LAD

%

% lad Least absolute deviations fitting via ADMM

%

% [x, history] = lad(A, b, rho, alpha)

%

% Solves the following problem via ADMM:

%

% minimize ||Ax - b||\_1

%

% The solution is returned in the vector x.

%

% history is a structure that contains the objective value, the primal and

% dual residual norms, and the tolerances for the primal and dual residual

% norms at each iteration.

%

% rho is the augmented Lagrangian parameter.

%

% alpha is the over-relaxation parameter (typical values for alpha are

% between 1.0 and 1.8).

%

%

% More information can be found in the paper linked at:

% http://www.stanford.edu/~boyd/papers/distr\_opt\_stat\_learning\_admm.html

%

rand('seed', 0);

randn('seed', 0);

m = 1000; % number of examples

n = 100; % number of features

A = randn(m,n);

x0 = 10\*randn(n,1);

b = A\*x0;

idx = randsample(m,ceil(m/50));

b(idx) = b(idx) + 1e2\*randn(size(idx));

[x objval r\_norm s\_norm eps\_pri eps\_dual] = lad(A, b, 1.0, 1.0);

K = length(objval);

h = figure;

plot(1:K, objval, 'k', 'MarkerSize', 10, 'LineWidth', 2);

ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');

g = figure;

subplot(2,1,1);

semilogy(1:K, max(1e-8,r\_norm), 'k', ...

1:K, eps\_pri, 'k--', 'LineWidth', 2);

ylabel('||r||\_2');

subplot(2,1,2);

semilogy(1:K, max(1e-8, s\_norm), 'k', ...

1:K, eps\_dual, 'k--', 'LineWidth', 2);

ylabel('||s||\_2'); xlabel('iter (k)');

function [x,objval,r\_norm,s\_norm,eps\_pri,eps\_dual] = lad(A, b, rho, alpha)

t\_start = tic;

QUIET = 0;

MAX\_ITER = 1000;

ABSTOL = 1e-4;

RELTOL = 1e-2;

[m n] = size(A);

x = zeros(n,1);

z = zeros(m,1);

u = zeros(m,1);

if ~QUIET

fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...

'r norm', 'eps pri', 's norm', 'eps dual', 'objective');

end

for k = 1:MAX\_ITER

if k > 1

x = R \ (R' \ (A'\*(b + z - u)));

else

R = chol(A'\*A);

x = R \ (R' \ (A'\*(b + z - u)));

end

zold = z;

Ax\_hat = alpha\*A\*x + (1-alpha)\*(zold + b);

z = shrinkage(Ax\_hat - b + u, 1/rho);

u = u + (Ax\_hat - z - b);

% diagnostics, reporting, termination checks

objval(k) = objective(z);

r\_norm(k) = norm(A\*x - z - b);

s\_norm(k) = norm(-rho\*A'\*(z - zold));

eps\_pri(k) = sqrt(m)\*ABSTOL + RELTOL\*max([norm(A\*x), norm(-z), norm(b)]);

eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*A'\*u);

if ~QUIET

fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...

r\_norm(k), eps\_pri(k), ...

s\_norm(k), eps\_dual(k), objval(k));

end

if (r\_norm(k) < eps\_pri(k) && ...

s\_norm(k) < eps\_dual(k))

break;

end

end

if ~QUIET

toc(t\_start);

end

end

function obj = objective(z)

obj = norm(z,1);

end

function y = shrinkage(a, kappa)

y = max(0, a-kappa) - max(0, -a-kappa);

end

#### Quadratic programing

randn('state', 0);

rand('state', 0);

n = 100;

% generate a well-conditioned positive definite matrix

% (for faster convergence)

P = rand(n);

P = P + P';

[V D] = eig(P);

P = V\*diag(1+rand(n,1))\*V';

q = randn(n,1);

r = randn(1);

l = randn(n,1);

u = randn(n,1);

lb = min(l,u);

ub = max(l,u);

[x z u r\_norm s\_norm g obj] = quadprog(P, q, r, lb, ub, 1.0, 1.0);

fprintf('%20s\t%10s\t%10s\t \n', 'x^(k+1)','z^(k+1)','u^(k+1)');

for i =1:length(x)

fprintf('%20f\t%10.4f\t%10.4f\t \n', x(i),z(i),u(i));

end

figure

%plotting difference between current z and old z

subplot(2,1,1)

plot(1:g,r\_norm)

ylabel('||r||\_2'); xlabel('iter (k)');

%plotting difference between z and x

subplot(2,1,2)

plot(1:g,s\_norm)

ylabel('||s||\_2'); xlabel('iter (k)');

figure

%plotting norm with iteration

plot(1:g,obj)

ylabel('f(x^k) + g(z^k)'); xlabel('iter (k)');

function [x\_hat z u r\_norm s\_norm g obj] = quadprog(P, q, r, lb, ub, rho, alpha)

t\_start = tic;

QUIET = 0;

MAX\_ITER = 1000;

ABSTOL = 1e-4;

RELTOL = 1e-2;

n = size(P,1);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

for k = 1:MAX\_ITER

if k > 1

x = R \ (R' \ (rho\*(z - u) - q));

else

R = chol(P + rho\*eye(n));

x = R \ (R' \ (rho\*(z - u) - q));

end

% z-update with relaxation

zold = z;

x\_hat = alpha\*x +(1-alpha)\*zold;

z = min(ub, max(lb, x\_hat + u));

% u-update

u = u + (x\_hat - z);

% diagnostics, reporting, termination checks

obj(k) = objective(P, q, r, x);

r\_norm(k) = norm(x - z);

s\_norm(k) = norm(-rho\*(z - zold));

if(z-zold<RELTOL )

g=k

break;

end

end

if ~QUIET

toc(t\_start);

end

end

function obj = objective(P, q, r, x)

obj = 0.5\*x'\*P\*x + q'\*x + r;

end

#### Linear programing

function [z, history] = linprog(c, A, b, rho, alpha)

% linprog Solve standard form LP via ADMM

%

% [x, history] = linprog(c, A, b, rho, alpha);

%

% Solves the following problem via ADMM:

%

% minimize c'\*x

% subject to Ax = b, x >= 0

%

% The solution is returned in the vector x.

%

% history is a structure that contains the objective value, the primal and

% dual residual norms, and the tolerances for the primal and dual residual

% norms at each iteration.

%

% rho is the augmented Lagrangian parameter.

%

% alpha is the over-relaxation parameter (typical values for alpha are

% between 1.0 and 1.8).

%

%

% More information can be found in the paper linked at:

% http://www.stanford.edu/~boyd/papers/distr\_opt\_stat\_learning\_admm.html

%

t\_start = tic;

QUIET = 0;

MAX\_ITER = 1000;

ABSTOL = 1e-4;

RELTOL = 1e-2;

[m n] = size(A);

x = zeros(n,1);

z = zeros(n,1);

u = zeros(n,1);

if ~QUIET

fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...

'r norm', 'eps pri', 's norm', 'eps dual', 'objective');

end

for k = 1:MAX\_ITER

% x-update

tmp = [ rho\*eye(n), A'; A, zeros(m) ] \ [ rho\*(z - u) - c; b ];

x = tmp(1:n);

% z-update with relaxation

zold = z;

x\_hat = alpha\*x + (1 - alpha)\*zold;

z = pos(x\_hat + u);

u = u + (x\_hat - z);

% diagnostics, reporting, termination checks

history.objval(k) = objective(c, x);

history.r\_norm(k) = norm(x - z);

history.s\_norm(k) = norm(-rho\*(z - zold));

history.eps\_pri(k) = sqrt(n)\*ABSTOL + RELTOL\*max(norm(x), norm(-z));

history.eps\_dual(k)= sqrt(n)\*ABSTOL + RELTOL\*norm(rho\*u);

if ~QUIET

fprintf('%3d\t%10.4f\t%10.4f\t%10.4f\t%10.4f\t%10.2f\n', k, ...

history.r\_norm(k), history.eps\_pri(k), ...

history.s\_norm(k), history.eps\_dual(k), history.objval(k));

end

if (history.r\_norm(k) < history.eps\_pri(k) && ...

history.s\_norm(k) < history.eps\_dual(k))

break;

end

end

if ~QUIET

toc(t\_start);

end

end

function obj = objective(c, x)

obj = c'\*x;

end