ADMM Project Report

This report is regarding the observations, calculations and scripting of ADMM algorithm used in various daily life applications. The algorithm shows how simple optimization can be even when extended to a million variables.

ADMM Project Report

Table of Contents

Acknowledgement

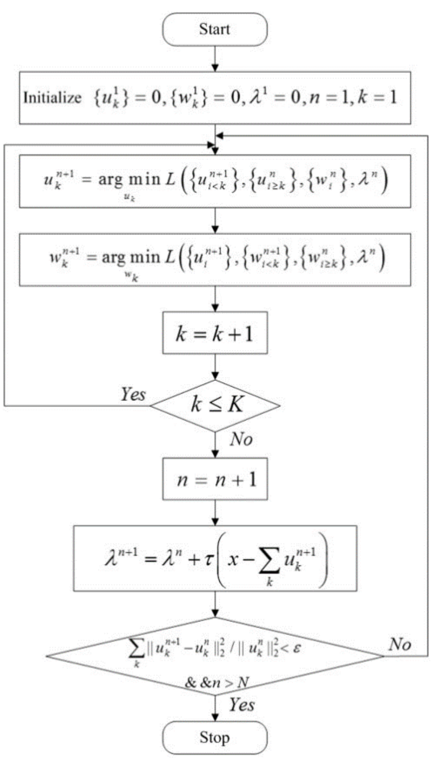
I would like to thank Mr. K.P Soman to grant me an opportunity to make this project report analysis and helping throughout the course. I would also like to thank Mr. Premjith for guiding and helping me throughout course.

What is ADMM?

The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which are then easier to handle. It has recently found wide application in a number of areas.

Lagrangian Multiplier

In mathematical optimization, the method of Lagrange multipliers is a strategy for finding the local maxima and minima of a function subject to equality constraints. The basic idea is to convert a constrained problem into a form such that the derivative test of an unconstrained problem can still be applied.



Topics Chosen

1. Linear Programming
2. Huber fitting
3. SVM
4. Intersection of Polyhedra
5. Quadratic Programming
6. Lasso

Linear Programming

Code:

```
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
```

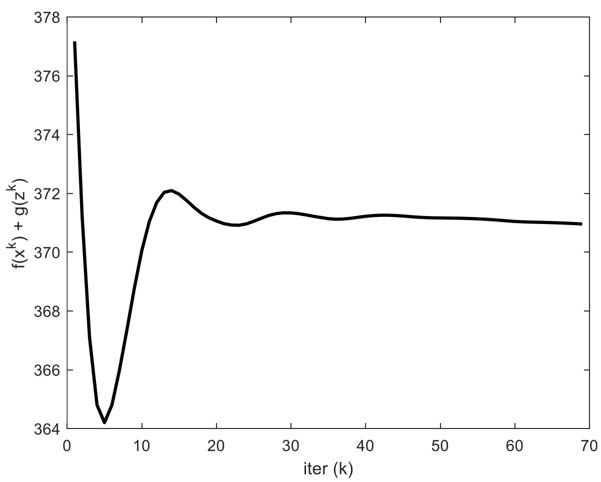
```
    history.objval(k)  = objective(c, x);

    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
```

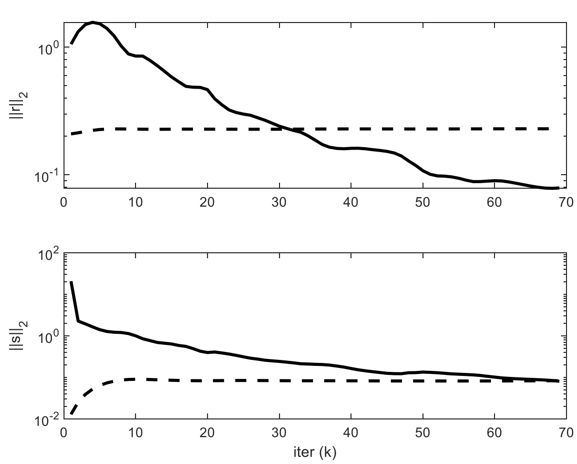
Example

```
randn('state', 0);
rand('state', 0);
n = 500;  % dimension of x
m = 400;  % number of equality constraints
c  = rand(n,1) + 0.5;    % create nonnegative price vector with mean 1
x0 = abs(randn(n,1));    % create random solution vector

A = abs(randn(m,n));     % create random, nonnegative matrix A
b = A*x0;


[x history] = linprog(c, A, b, 1.0, 1.0);

K = length(history.objval);
h = figure;
```

```
plot(1:K, history.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, history.r_norm), 'k', ...
    1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
    1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:





Intersection of Polyhedra

Code:

```
function [z, history] = polyhedra_intersection(A1, b1, A2, b2, rho, alpha)

t_start = tic;
QUIET    = 0;
MAX_ITER = 1000;
ABSTOL   = 1e-4;
RELTOL   = 1e-2;

n = size(A1,2);
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
   % use cvx to find point in first polyhedra
   cvx_begin quiet
      variable x(n)
      minimize (sum_square(x - (z - u)))
      subject to
         A1*x <= b1
   cvx_end

   % z-update with relaxation
   zold = z;
   x_hat = alpha*x + (1 - alpha)*zold;
   % use cvx to find point in second polyhedra
   cvx_begin quiet
      variable z(n)
      minimize (sum_square(x_hat - (z - u)))
      subject to
         A2*z <= b2
   cvx_end

   u = u + (x_hat - z);

   % diagnostics, reporting, termination checks
```

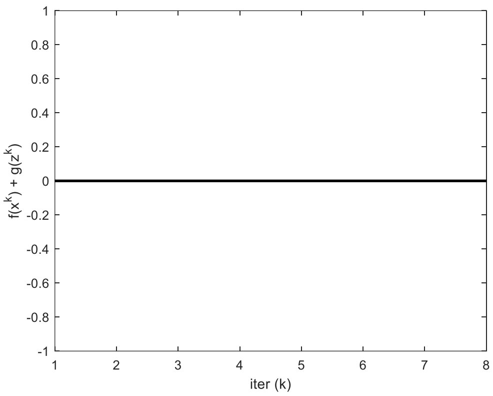
```
    history.objval(k)  = 0;
    history.r_norm(k)  = norm(x - z);
    history.s_norm(k)  = norm(-rho*(z - zold));

    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
```

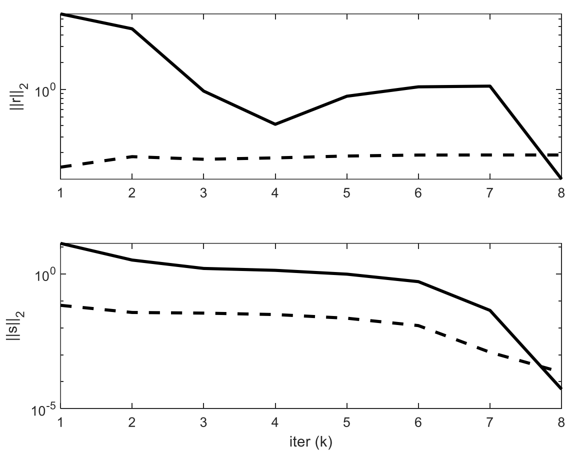
Example:

```
randn('state', 0);
rand('state', 0);

n = 5;      % dimension of variable
m1 = 10;    % number of faces for polyhedra 1
m2 = 12;    % number of faces for polyhedra 2

c1 = 10*randn(n,1);       % center of polyhedra 1
c2 = -10*randn(n,1);      % center of polyhedra 2

% consider the following picture:
%
%      a1
% c ---------> x
%
% from the center "c", we travel along vector "a1" (not necessarily a unit
% vector) until we reach x. at "x", a1'x = b. a point y is to the left of x
% if a1'y <= b.
%

% pick m1 random directions with different magnitudes
A1 = diag(1 + rand(m1,1))*randn(m1,n);
% the value of b is found by traveling from the center along the normal
```

```
% vectors in A1 and taking its inner product with A1.
b1 = diag(A1*(c1*ones(1,m1) + A1'));

% pick m2 random directions with different magnitudes
A2 = diag(1 + rand(m2,1))*randn(m2,n);
% the value of b is found by traveling from the center along the normal
% vectors in A1 and taking its inner product with A1.
b2 = diag(A2*(c2*ones(1,m2) + A2'));

% find the distance between the two polyhedra--make sure they overlap by
% checking if the distance is 0
cvx_begin quiet
   variables x(n) y(n)
   minimize sum_square(x - y)
   subject to
      A1*x <= b1
      A2*y <= b2
cvx_end

% if the distance is not 0, expand A1 and A2 by a little more than half the
% distance
if norm(x-y) > 1e-4,
   A1 = (1 + 0.5*norm(x-y))*A1;
   A2 = (1 + 0.5*norm(x-y))*A2;
   % recompute b's as appropriate
   b1 = diag(A1*(c1*ones(1,m1) + A1'));
   b2 = diag(A2*(c2*ones(1,m2) + A2'));
end

[x history] = polyhedra_intersection(A1, b1, A2, b2, 1.0, 1.0);

K = length(history.objval);

h = figure;
plot(1:K, history.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, history.r_norm), 'k', ...
   1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
   1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
f(x^k) + g(z^k)
1 2 3 4 5 6 7 8
iter (k)


||r||2
10^0
1 2 3 4 5 6 7 8
||s||2
10^0
10^-5
1 2 3 4 5 6 7 8
iter (k)

Support Vector Machine

Code:

```
function [xave, history] = linear_svm(A, lambda, p, rho, alpha)
% linear_svm   Solve linear support vector machine (SVM) via ADMM
%
% [x, history] = linear_svm(A, lambda, p, rho, alpha)
%
% Solves the following problem via ADMM:
%
%   minimize   (1/2)||w||_2^2 + \lambda sum h_j(w, b)
%
% where A is a matrix given by [-y_j*x_j -y_j], lambda is a
% regularization parameter, and p is a partition of the observations in to
% different subsystems.
%
% The function h_j(w, b) is a hinge loss on the variables w and b.
% It corresponds to h_j(w,b) = (Ax + 1)_+, where x = (w,b).
%
% This function implements a *distributed* SVM that runs its updates
% serially.
% The solution is returned in the vector x = (w,b).
%
% 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);
N = max(p);
% group samples together
for i = 1:N,
    tmp{i} = A(p==i,:);
end
A = tmp;
x = zeros(n,N);
z = zeros(n,N);
u = zeros(n,N);
```

```
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
    for i = 1:N,
        cvx_begin quiet
            variable x_var(n)
            minimize ( sum(pos(A{i}*x_var + 1)) + rho/2*sum_square(x_var - z(:,i) + u(:,i)) )
        cvx_end
        x(:,i) = x_var;
    end
    xave = mean(x,2);

    % z-update with relaxation
    zold = z;
    x_hat = alpha*x +(1-alpha)*zold;
    z = N*rho/(1/lambda + N*rho)*mean( x_hat + u, 2 );
    z = z*ones(1,N);

    % u-update
    u = u + (x_hat - z);

    % diagnostics, reporting, termination checks
    history.objval(k)  = objective(A, lambda, p, x, z);

    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;
```

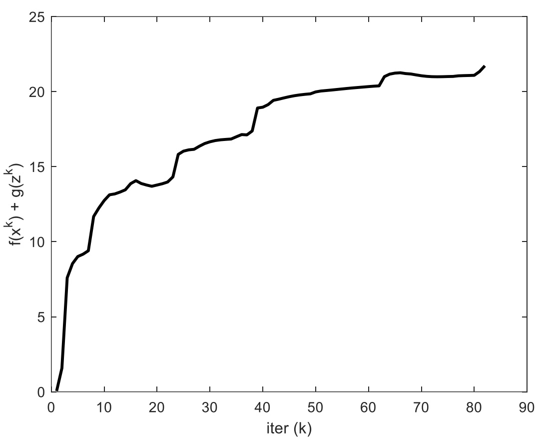
```
    end
end

if ~QUIET
    toc(t_start);
end
end

function obj = objective(A, lambda, p, x, z)
    obj = hinge_loss(A,x) + 1/(2*lambda)*sum_square(z(:,1));
end

function val = hinge_loss(A,x)
    val = 0;
    for i = 1:length(A)
        val = val + sum(pos(A{i}*x(:,i) + 1));
    end
end
```

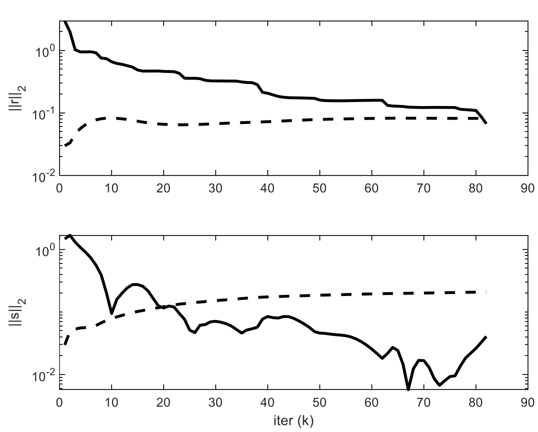
Example:

```
rand('seed', 0);
randn('seed', 0);

n = 2;
m = 200;
N = m/2;
M = m/2;

% positive examples
Y = [1.5+0.9*randn(1,0.6*N), 1.5+0.7*randn(1,0.4*N);
2*(randn(1,0.6*N)+1), 2*(randn(1,0.4*N)-1)];

% negative examples
X = [-1.5+0.9*randn(1,0.6*M),  -1.5+0.7*randn(1,0.4*M);
2*(randn(1,0.6*M)-1), 2*(randn(1,0.4*M)+1)];

x = [X Y];
y = [ones(1,N) -ones(1,M)];
A = [ -((ones(n,1)*y).*x)' -y'];
xdat = x';
lambda = 1.0;

% partition the examples up in the worst possible way
% (subsystems only have positive or negative examples)
p = zeros(1,m);
p(y == 1)  = sort(randi([1 10], sum(y==1),1));
p(y == -1) = sort(randi([11 20], sum(y==-1),1));
```

```
[x history] = linear_svm(A, lambda, p, 1.0, 1.0);

K = length(history.objval);

h = figure;
plot(1:K, history.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, history.r_norm), 'k', ...
    1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
    1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:





Quadratic Programming

Code:

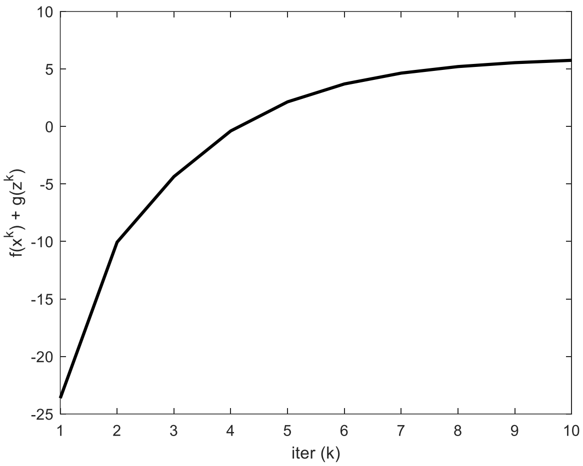
```
function [z, history] = quadprog(P, q, r, lb, ub, rho, alpha)
% quadprog  Solve standard form box-constrained QP via ADMM
%
% [x, history] = quadprog(P, q, r, lb, ub, rho, alpha)
%
% Solves the following problem via ADMM:
%
%   minimize     (1/2)*x'*P*x + q'*x + r
%   subject to   lb <= x <= ub
%
% 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;
n = size(P,1);
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

    if k > 1
```

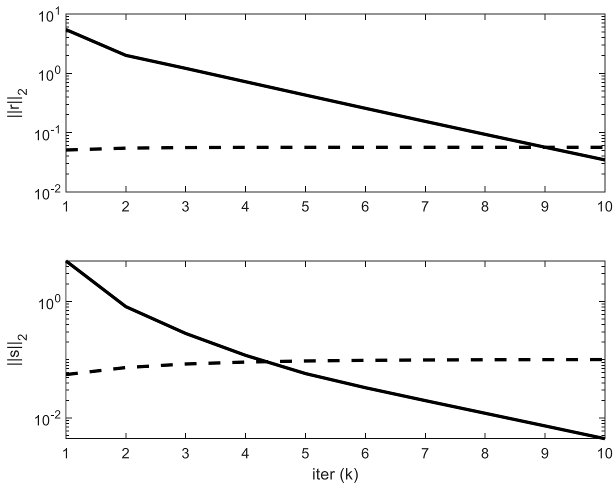
```
        x = R \ (R' \ (rho*(z - u) - q));
    else
        R = chol(P + rho*eye(n));
        x = R \ (R' \ (rho*(z - u) - q));
    end

    % 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
    history.objval(k)  = objective(P, q, r, 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(P, q, r, x)
    obj = 0.5*x'*P*x + q'*x + r;
end
```

Example:

```
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 history] = quadprog(P, q, r, lb, ub, 1.0, 1.0);
K = length(history.objval);

h = figure;
plot(1:K, history.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, history.r_norm), 'k', ...
    1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
    1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:


10
5
0
-5
-10
-15
-20
-25
$f(x^k) + g(z^k)$
1
2
3
4
5
6
7
8
9
10
iter (k)


10^1
10^0
10^{-1}
10^{-2}
$||r||_2$
1
2
3
4
5
6
7
8
9
10
10^0
10^{-2}
$||s||_2$
1
2
3
4
5
6
7
8
9
10
iter (k)

Huber fitting

Code:

```
function [x, history] = huber_fit(A, b, rho, alpha)
% huber_fit  Solves a robust fitting problem
%
% [z, history] = huber_fit(A, b, rho, alpha);
%
% solves the following problem via ADMM:
%
%   minimize 1/2*sum(huber(A*x - b))
%
% with variable x.
%
% 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);

% save a matrix-vector multiply
Atb = A'*b;
x = zeros(n,1);
z = zeros(m,1);
u = zeros(m,1);

% cache factorization
[L U] = factor(A);
```

```
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
    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
    history.objval(k)  = objective(z);

    history.r_norm(k)  = norm(A*x - z - b);
    history.s_norm(k)  = norm(-rho*A'*(z - zold));

    history.eps_pri(k) = sqrt(n)*ABSTOL + RELTOL*max([norm(A*x), norm(-z), norm(b)]);
    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
```

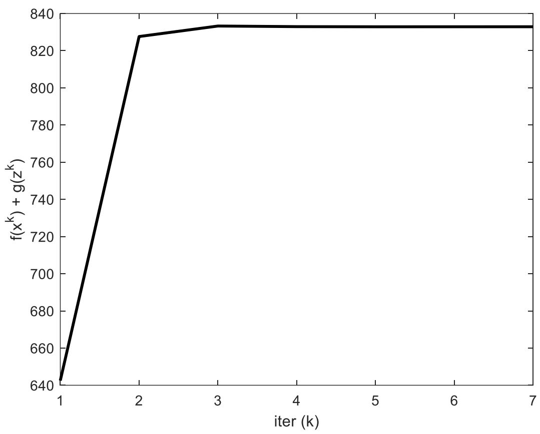
```
function p = objective(z)
   p = ( 1/2*sum(huber(z)) );
end

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
```

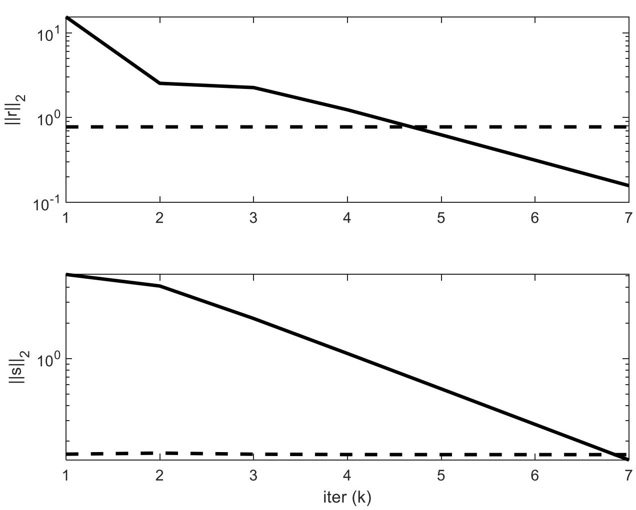
Example:

```
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);      % add sparse, large noise

[x history] = huber_fit(A, b, 1.0, 1.0);
K = length(history.objval);

h = figure;
plot(1:K, history.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, history.r_norm), 'k', ...
   1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
```

```
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
    1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:





Lasso

Code:

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

% save a matrix-vector multiply
Atb = A'*b;
x = zeros(n,1);
z = zeros(n,1);
u = zeros(n,1);

% cache the factorization
[L U] = factor(A, rho);

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
   q = Atb + rho*(z - u);    % temporary value
   if( m >= n )    % if skinny
     x = U \ (L \ q);
   else            % if fat
     x = q/rho - (A'*(U \ ( L \ (A*q) )))/rho^2;
   end

   % z-update with relaxation
   zold = z;
   x_hat = alpha*x + (1 - alpha)*zold;
   z = shrinkage(x_hat + u, lambda/rho);

   % u-update
   u = u + (x_hat - z);

   % diagnostics, reporting, termination checks
   history.objval(k)  = objective(A, b, lambda, x, z);

   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 p = objective(A, b, lambda, x, z)
```

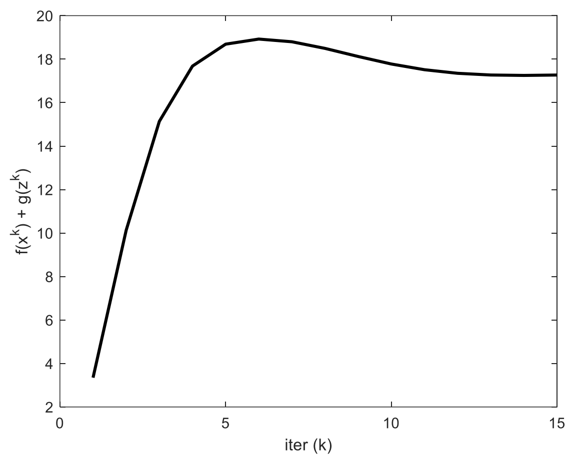
```
   p = ( 1/2*sum((A*x - b).^2) + lambda*norm(z,1) );
end

function z = shrinkage(x, kappa)
   z = max( 0, x - kappa ) - max( 0, -x - kappa );
end

function [L U] = factor(A, rho)
   [m, n] = size(A);
   if ( m >= n )    % if skinny
      L = chol( A'*A + rho*speye(n), 'lower' );
   else            % if fat
      L = chol( speye(m) + 1/rho*(A*A'), 'lower' );
   end

   % force matlab to recognize the upper / lower triangular structure
   L = sparse(L);
   U = sparse(L');
end
```

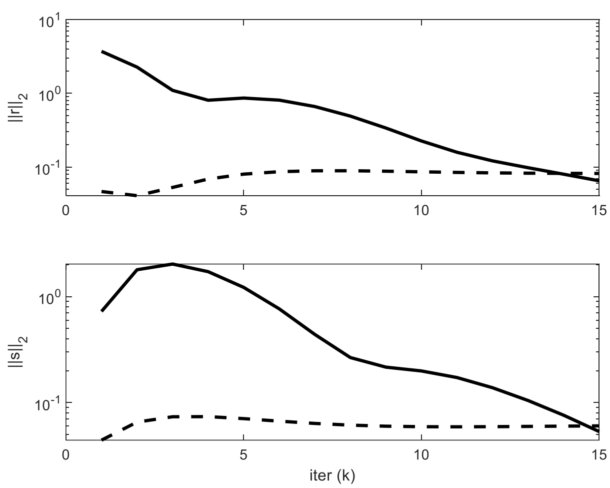
Example:

```
randn('seed', 0);
rand('seed',0);

m = 1500;       % number of examples
n = 5000;       % number of features
p = 100/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;

[x history] = lasso(A, b, lambda, 1.0, 1.0);

K = length(history.objval);

h = figure;
plot(1:K, history.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, history.r_norm), 'k', ...
```

```
    1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

subplot(2,1,2);
semilogy(1:K, max(1e-8, history.s_norm), 'k', ...
    1:K, history.eps_dual, 'k--', 'LineWidth', 2);
ylabel('||s||_2'); xlabel('iter (k)');
```

Plots:





References

https://web.stanford.edu/~boyd/papers/admm/