AMRITA
VISHWA VIDYAPEETHAM

19 MAT204 MATHEMATICS FOR INTELLIGENT SYSTEMS - III

ADMM AND ITS APPLICATIONS

GROUP 3
TEAM MEMBERS

Dinesh Kumar M R	- CB.EN.U4AIE20011
Rishi Karthigayan S	- CB.EN.U4AIE20057
N T Shrish Surya	- CB.EN.U4AIE20067
Tsaliki Satya Ganesh Kumar	- CB.EN.U4AIE20073

TABLE OF CONTENTS

Contents

Declaration

We hereby declare that the project titled “ADMM and its applications" submitted to the Center for Excellence in Computational Engineering and Networking is a record of the original work done under the guidance of Dr. Neethu Mohan, Assistant Professor in Computational Engineering and Networking, Amrita University.

Signature of the faculty

Acknowledgment

The completion of this project has been made possible by the efforts of many. We would like to thank Dr. Sasangan Ramanathan, Dean - Engineering, and our Mathematics for Intelligent Systems-III faculty, Dr.Neethu Mohan, for their unstinting support. We would also like to extend our gratitude to our friends and family who have aided us throughout the project.

Abstract

Many problems of recent interest in statistics and machine learning can be posed in the framework of convex optimization. Due to the explosion in size and complexity of modern datasets, it is increasingly important to be able to solve problems with a very large number of features or training examples. As a result, both the decentralized collection or storage of these datasets as well as accompanying distributed solution methods are either necessary or at least highly desirable. In this review, we argue that the alternating direction method of multipliers is well suited to distributed convex optimization, and in particular to large-scale problems arising in statistics, machine learning, and related areas. The method was 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, Bregman iterative algorithms for l1 problems, proximal methods, and others. After briefly surveying the theory and history of the algorithm, we discuss applications to a wide variety of statistical and machine learning problems of recent interest, including the basis pursuit, regularised logistic regression, sparse inverse covariance selection, support vector machines, and many others. We also discuss linear programming, quadratic programming, and the intersection of polyhedra.

Introduction:

The alternating direction method of multipliers (ADMM) is an algorithm that solves convex optimization problems by breaking them into smaller pieces, each of which is then easier to handle. It has recently found wide application in a number of areas. ADMM Is a simple and powerful iterative algorithm for convex optimization problems. It is much faster than conventional methods. It has emerged as a powerful technique for largescale structured optimization.

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. It also gives us a flexible framework for incorporating many types of convex constraints. ADMM can be viewed as an attempt to blend the benefits of dual decomposition and augmented Lagrangian methods for constrained optimization.

Augmented Lagrangian and the Method of Multipliers:

Augmented lagrangian method was developed to bring robustness to the dual ascent method. The augmented lagrangian is denoted as $L_\rho(x,y)$. The augmented lagrangian for minimizing the function $f(x)$ subjected to $Ax = b$ is

$$L_\rho(x,y) = f(x) + \lambda(Ax - b) + (\rho/2) \parallel Ax - b \parallel_2^2$$

where ρ >0, it is called penalty parameter and $(\rho/2) \parallel Ax - b \parallel_2^2$ is called a penalty term after modifying this problem is clearly equivalent to the original problem since for any feasible x the term added to the objective is zero.

Now we have to minimize the x value . for that we can apply dual ascent to our modified problem.

$$x^{k+1} = L_\rho\ (x, z^k, \lambda^k) z^{k+1} = L_\rho\ (x^{k+1}, z\ \ , \lambda^k) \lambda^{k+1}$$
$$= \lambda^k + \rho(Ax^{k+1} + Bz^{k+1} - c)$$

At first we have to perform x-minimization and then z-minimization step. Thus they update alternatively

ADMM and its various forms:

ADMM Form-1:

Consider a need to minimize f(x)+g(x),

ADMM works on the principle of increasing the number of variables and solving each of them via multiple iteration. Hence the problem can be re-formulated to,

min f(x)+g(z)

Subject to x-z=0

The augmented Lagrangian can be written as :

$$L(x,y,z) = f(x)+g(z)+ \langle y,x-z\rangle+(\rho/2).||x - z||_2^2$$

$(\rho/2).||x - z||_2^2$ is the augmented Lagrangian which is used to unconstrained the optimization and decrease the iterations to obtain the solution. While obtaining for nth iteration for a variable , the rest are treated as constants.

$$x^{k+1} = \arg\min_x \left(f(x) + \underbrace{\langle y^k , x \rangle}_{y^T x} + \frac{\rho}{2}\|x - z^k\|_2^2 \right)$$

In this iteration for x , the constant and other variables are neglected. Hence,

$$x^{k+1} = \arg\min_x \left(f(x) + \frac{\rho}{2} \left\| x - z^k + \frac{1}{\rho} y^k \right\|_2^2 \right)$$
$$z^{k+1} = \arg\min_z \left(g(z) + \frac{\rho}{2} \left\| x^{k+1} - z + \frac{1}{\rho} y^k \right\|_2^2 \right)$$
$$y^{k+1} = y^k + \rho\left(x^{k+1} - z^{k+1}\right)$$

Substituting,

$$u^k = \frac{1}{\rho} y^k, \quad \lambda = \frac{1}{\rho}$$

$$x^{k+1} = \arg\min_x \left(f(x) + \frac{1}{2\lambda} \| x - (z^k - u^k) \|_2^2 \right)$$
$$z^{k+1} = \arg\min_z \left(g(z) + \frac{1}{2\lambda} \| z - (x^{k+1} + u^k) \|_2^2 \right)$$
$$y^{k+1} = y^k + \rho\left(x^{k+1} - z^{k+1}\right)$$

ADMM Form-2:

Consider, min f(x)

Subject to $x \varepsilon C$

This above problem is converted to min f(x) + g(z), Subject to x-z=0
Where g(x) is an indicator function which takes the values 0 if $x \varepsilon C$ and infinity otherwise.

The augmented Lagrangian is written as ,

$$L(x,z,u) = f(x)+g(z)+({}^{1}/_{2\lambda}).||x - z + u||_2^2$$

$$x^{k+1} = \arg\min_x \left(f(x) + \frac{1}{2\lambda} \| x - z^k + u^k \|_2^2 \right)$$
$$z^{k+1} = \Pi_C\left(x^{k+1} + u^k\right)$$
$$u^{k+1} = u^k + x^{k+1} - z^{k+1}$$

Where π_c represents Projection onto C.

Shrinkage function in a Nutshell :

When a threshold is fixed, any input above the threshold will be unaffected and if the input is less than the threshold it will be equated to 0.

Applications of ADMM:

Linear Programming:

In linear programming, the objective is to minimize a closed convex function

$f(x) = \frac{1}{2}x^T.P.x + q^Tx + r$; Subject to $Ax=B$

The problem will be converted to the form 2 of ADMM :

min f(x)+g(z)

Subject to x - z = 0

Where g(z) will be the indicator function.

$L(x,z,u) = f(x)+g(z)+(^1/_{2\lambda}).||x - z + u||_2^2$ |Lagrangian

Partial differentiation w.r.t x,z,u and substituting to 0

$$\frac{\partial L}{\partial x} = 0 \Rightarrow Px^{k+1} + q + \frac{1}{\lambda}\left(x^{k+1} - z^k + u^k\right) + A^T v = 0$$

$$\left(P + \frac{1}{\lambda} I\right)x^{k+1} + q + A^T v - \frac{1}{\lambda}\left(z^k - u^k\right) = 0 \qquad \text{--(1)}$$

$$\frac{\partial L}{\partial v} = 0 \Rightarrow Ax^{k+1} - b = 0 \qquad \text{--(2)}$$

$$\begin{bmatrix} P+\rho.I & A^T \\ A & 0 \end{bmatrix} . \begin{bmatrix} x^{k+1} \\ u \end{bmatrix} = \begin{bmatrix} \rho(z^k - u^k) - q \\ b \end{bmatrix}$$

$$\begin{bmatrix} P+\rho.I & A^T \\ A & 0 \end{bmatrix} . \begin{bmatrix} x^{k+1} \\ v \end{bmatrix} = \begin{bmatrix} P+\rho.I & A^T \\ A & 0 \end{bmatrix}^{-1} . \begin{bmatrix} \rho(z^k - u^k) - q \\ b \end{bmatrix}$$

| EQ-1

-- (From 1 and 2)

where ρ = 1/λ

The very same methodology can be implemented for solving Quadratic equations on x when subject to *Ax=b*. In case of a subject of inequality, the methodology will differ. Usually solving for a linear program, P will be a 0 matrix which reduces f(x) to c'(x)+r;

MATLAB Implementation :

```
function [z, history] = linprog(c, A, b, rho, alpha)
t_start = tic;

% Global Constants and Defaults

QUIET    = 0;
MAX_ITER = 1000;
ABSTOL   = 1e-4;
RELTOL   = 1e-2;
% Data preprocessing
[m n] = size(A);

% ADMM Solver
x = zeros(n,1);
z = zeros(n,1);
u = zeros(n,1);
```

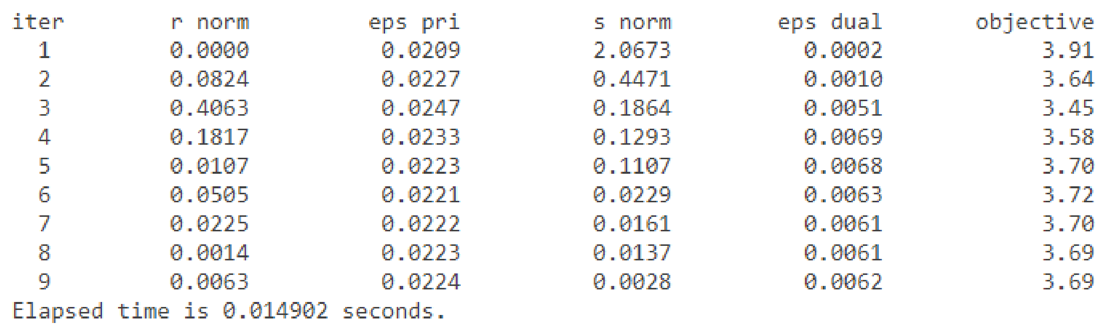
```
if ~QUIET
    fprintf('%3s\t%10s\t%10s\t%10s\t%10s\t%10s\n', 'iter', ...
      'r norm', 'eps pri', 's norm', 'eps dual', 'objective');
end

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 for solving a LP:

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

n = 5;  % dimension of x
```

```
m = 4;  % 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);
```

Output:



```
iter         r norm        eps pri        s norm       eps dual      objective
  1          0.0000        0.0209         2.0673        0.0002          3.91
  2          0.0824        0.0227         0.4471        0.0010          3.64
  3          0.4063        0.0247         0.1864        0.0051          3.45
  4          0.1817        0.0233         0.1293        0.0069          3.58
  5          0.0107        0.0223         0.1107        0.0068          3.70
  6          0.0505        0.0221         0.0229        0.0063          3.72
  7          0.0225        0.0222         0.0161        0.0061          3.70
  8          0.0014        0.0223         0.0137        0.0061          3.69
  9          0.0063        0.0224         0.0028        0.0062          3.69
Elapsed time is 0.014902 seconds.
```

Quadratic Programming:

For solving Quadratic equations Subject to lower and upper bound.

$$f(x) = \frac{1}{2} x^T . P . x \; + \; q^T x + r \quad ; \text{Subject to : } lb<x<ub$$

The problem will be converted to the form 2 of ADMM :

min f(x)+g(z)

Subject to x - z = 0

where g(z) will be the indicator function.

$$L(x,z,u) = f(x)+g(z)+(\rho/_2).||x - z + u||_2^2 \qquad |\text{Lagrangian}$$

From EQ-1 : When A,b and v are null –

$x = [P + \rho.I]^{-1} * [\rho.(z - u) - q]$ | I is an Identity Matrix

For increasing efficiency and speed of computation , Consider R to be an upper triangular matrix obtained from Cholesky Factorisation of [P+p.I].

Cholesky Factorisation :

When A is a Hermitian and Positive Deficient Matrix on factorisation we get R, Where A = $R*R^T$

Hence to increase efficiency ,

$x^{k+1} = R * R^{-1} * [\rho.(z^k – u^k) - q]$ | Where $[P + \rho.I]^{-1} = R*R^T$

$z = \min(ub , \max(lb , x^{k+1}+ u^k))$

$u^{k+1} = u^k + (x^{k+1} - z)$

MATLAB Implementation :

```
function [z, history] = quadprog(P, q, r, lb, ub, rho, alpha)

t_start = tic;

%Global constants and defaults
QUIET    = 0;
MAX_ITER = 1000;
ABSTOL   = 1e-4;
RELTOL   = 1e-2;

%Data preprocessing
n = size(P,1);

%ADMM solver
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
```

```matlab
    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 for solving a QP:

```
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);

%Solve problem
[x history] = quadprog(P, q, r, lb, ub, 1.0, 1.0);
```

Output:

```
iter        r norm       eps pri        s norm       eps dual       objective
  1         5.4585        0.0508        4.9772        0.0556          -23.62
  2         2.0050        0.0546        0.8151        0.0734          -10.06
  3         1.2096        0.0558        0.2823        0.0845           -4.36
  4         0.7197        0.0562        0.1188        0.0912           -0.39
  5         0.4287        0.0564        0.0578        0.0952            2.14
  6         0.2564        0.0564        0.0331        0.0976            3.70
  7         0.1541        0.0564        0.0199        0.0990            4.64
  8         0.0931        0.0564        0.0120        0.0999            5.21
  9         0.0566        0.0564        0.0073        0.1004            5.55
 10         0.0345        0.0564        0.0044        0.1007            5.75
Elapsed time is 0.067481 seconds.
```

Basis Pursuit:

The objective is very simple to minimise the values satisfying $Ax = b$

minimize $\|x\|_1$ subject to $Ax = b$

ADMM formulation:

minimize f(x)g(x) subject to x-z vector

where,

$$f(x) = \|x\|_1$$

$$g(z) = \begin{cases} 0, & Az = b \\ \infty & \text{otherwise} \end{cases}$$

g(z) is like a penalty function when it satisfies the condition Az=b , it will be 0 otherwise it will shoot up to a large value.

The ADMM Algorithm will be :

$$x^{k+1} := \Pi(z^k - u^k)$$

$$z^{k+1} := S_{1/\rho}(x^{k+1} + u^k)$$

$$u^{k+1} := u^k + x^{k+1} - z^{k+1},$$

where Π is projection onto $\{x \in R^n \mid Ax = b\}$.

The x-update, which involves solving a linearly-constrained minimum Euclidean norm problem, can be written explicitly as

$$x^{k+1} := (I - A^T(AA^T)^{-1}A)*(z^k - u^k) + A^T(AA^T)^{-1}b.$$

Now , let $P = (I - A^T(AA^T)^{-1}A)$ and $q = A^T(AA^T)^{-1}b$.

Hence , $x^{k+1} = P*(z^k - u^k) + q$

MATLAB Implementation :

```
function [z, history] = basis_pursuit(A, b, rho, alpha)


t_start = tic;


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

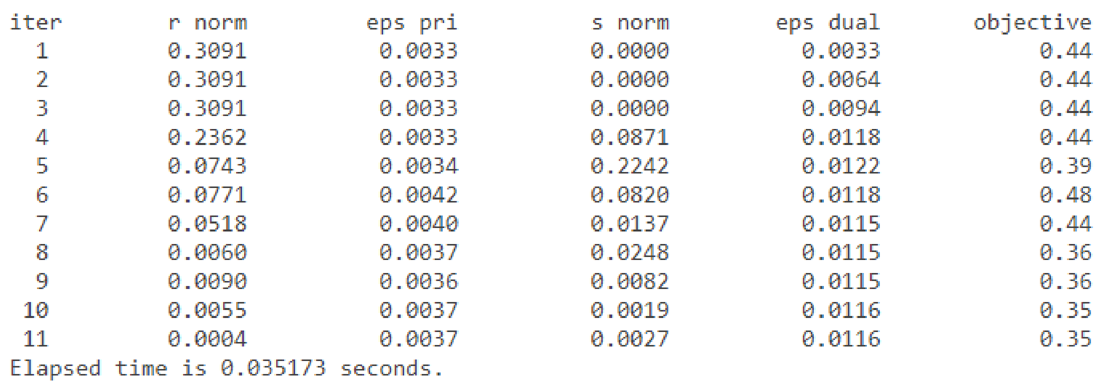
```
[m n] = size(A);


x = zeros(n,1);
z = zeros(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


% precompute static variables for x-update (projection on to Ax=b)
AAt = A*A';
P = eye(n) - A' * (AAt \ A);
q = A' * (AAt \ b);


for k = 1:MAX_ITER
    % x-update
    x = P*(z - u) + q;


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


    u = u + (x_hat - z);


    % diagnostics, reporting, termination checks
    history.objval(k)  = objective(A, b, 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(A, b, x)
    obj = norm(x,1);
end


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

end
```

Example problem solved using ADMM:

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

n = 3;
m = 1;
A = randn(m,n);

x = sprandn(n, 1, 0.1*n);
b = A*x;
xtrue = x;
[x history] = basis_pursuit(A, b, 1.0, 1.0);
```

Output:



```
iter         r norm        eps pri          s norm        eps dual        objective
  1          0.3091         0.0033          0.0000         0.0033            0.44
  2          0.3091         0.0033          0.0000         0.0064            0.44
  3          0.3091         0.0033          0.0000         0.0094            0.44
  4          0.2362         0.0033          0.0871         0.0118            0.44
  5          0.0743         0.0034          0.2242         0.0122            0.39
  6          0.0771         0.0042          0.0820         0.0118            0.48
  7          0.0518         0.0040          0.0137         0.0115            0.44
  8          0.0060         0.0037          0.0248         0.0115            0.36
  9          0.0090         0.0036          0.0082         0.0115            0.36
 10          0.0055         0.0037          0.0019         0.0116            0.35
 11          0.0004         0.0037          0.0027         0.0116            0.35
Elapsed time is 0.035173 seconds.
```

L1 – Regularized Logistic Regression:

Logistic regression is widely used in machine learning for classification problems. It is well-known that regularization is required to avoid over-fitting, especially when there is only a small number of training examples, or when there are a large number of parameters to be learned. In particular, L1 regularized logistic regression is often used for feature selection and has been shown to have good generalization performance in the presence of many irrelevant features.

L1 regularized logistic regression is a workhorse of machine learning: it is widely used for many classification problems, particularly ones with many features. L1 regularized logistic regression requires solving a convex optimization problem. However, a drawback is that standard algorithms for solving convex optimization problems do not scale well enough to handle the large datasets encountered in many practical settings.

Matlab Implementation:

```
function [z, history] = logreg(A, b, mu, rho, alpha)

t_start = tic;
```

Global constants and defaults

```
QUIET = 0;

MAX_ITER = 1000;

ABSTOL   = 1e-4;

RELTOL   = 1e-2;
```

Data preprocessing

```
[m, n] = size(A);
```

ADMM solver

```
x = zeros(n+1,1);
```

```
z = zeros(n+1,1);
u = zeros(n+1,1);

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
      x = update_x(A, b, u, z, rho);
      % z-update with relaxation
      zold = z;
      x_hat = alpha*x + (1-alpha)*zold;
      z = x_hat + u;
      z(2:end) = shrinkage(z(2:end), (m*mu)/rho);
      u = u + (x_hat - z);
      % diagnostics, reporting, termination checks
      history.objval(k)  = objective(A, b, mu, 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, b, mu, x, z)

      m = size(A,1);

      obj = sum(log(1 + exp(-A*x(2:end) - b*x(1)))) + m*mu*norm(z,1);

end

 function x = update_x(A, b, u, z, rho, x0)

      % solve the x update

      %   minimize [ -logistic(x_i) + (rho/2)||x_i - z^k + u^k||^2 ]

      % via Newton's method; for a single subsystem only.

      alpha = 0.1;

      BETA  = 0.5;

      TOLERANCE = 1e-5;

      MAX_ITER = 50;

      [m n] = size(A);

      I = eye(n+1);

      if exist('x0', 'var')

      x = x0;

      else

      x = zeros(n+1,1);

      end
      C = [-b -A];

      f = @(w) (sum(log(1 + exp(C*w))) + (rho/2)*norm(w - z + u).^2);

      for iter = 1:MAX_ITER

      fx = f(x);

      g = C'*(exp(C*x)./(1 + exp(C*x))) + rho*(x - z + u);

      H = C' * diag(exp(C*x)./(1 + exp(C*x)).^2) * C + rho*I;

      dx = -H\g;    % Newton step

      dfx = g'*dx; % Newton decrement

      if abs(dfx) < TOLERANCE

            break;

      end

      % backtracking

      t = 1;

      while f(x + t*dx) > fx + alpha*t*dfx

            t = BETA*t;

      end

      x = x + t*dx;

      end

end

function z = shrinkage(a, kappa)

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

end
```

Example Problem using ADMM:

Generate problem data

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

randn('seed', 0);



n = 5;
```

```
m = 20;


w = sprandn(n, 1, 0.1);  % N(0,1), 10% sparse

v = randn(1);            % random intercept

X = sprandn(m, n, 10/n);

btrue = sign(X*w + v);


% noise is function of problem size use 0.1 for large problem

b = sign(X*w + v + sqrt(0.1)*randn(m,1)); % labels with noise

A = spdiags(b, 0, m, m) * X;

ratio = sum(b == 1)/(m);

mu = 0.1 * 1/m * norm((1-ratio)*sum(A(b==1,:),1) + ratio*sum(A(b==-1,:),1),
'inf');

x_true = [v; w];
```

Solve problem

```
[x history] = logreg(A, b, mu, 1.0, 1.0);
```

```
iter        r norm      eps pri       s norm     eps dual    objective

  1         0.3774      0.0199      1.8188      0.0040         5.56

  2         0.0416      0.0276      0.9767      0.0042         4.78

  3         0.0202      0.0340      0.6855      0.0041         4.32

  4         0.0472      0.0390      0.5355      0.0038         4.06

  5         0.0515      0.0431      0.4411      0.0036         3.89

  6         0.0486      0.0466      0.3754      0.0034         3.78

  7         0.0435      0.0497      0.3265      0.0033         3.70

  8         0.0380      0.0524      0.2886      0.0033         3.64

  9         0.0328      0.0549      0.2581      0.0033         3.60

 10         0.0283      0.0571      0.2331      0.0033         3.57

 11         0.0565      0.0591      0.2047      0.0030         3.55

 12         0.0262      0.0609      0.1815      0.0029         3.54
```

13	0.0142	0.0625	0.1636	0.0029	3.53
14	0.0106	0.0640	0.1487	0.0029	3.53
15	0.0089	0.0653	0.1360	0.0028	3.52
16	0.0076	0.0666	0.1250	0.0028	3.52
17	0.0066	0.0677	0.1152	0.0028	3.52
18	0.0058	0.0688	0.1067	0.0028	3.52
19	0.0051	0.0698	0.0990	0.0028	3.53
20	0.0046	0.0707	0.0921	0.0028	3.53
21	0.0041	0.0715	0.0859	0.0028	3.53
22	0.0037	0.0723	0.0803	0.0028	3.53
23	0.0033	0.0731	0.0753	0.0028	3.54
24	0.0030	0.0738	0.0706	0.0028	3.54
25	0.0027	0.0744	0.0664	0.0028	3.55
26	0.0025	0.0751	0.0625	0.0028	3.55
27	0.0023	0.0756	0.0589	0.0028	3.55
28	0.0021	0.0762	0.0556	0.0028	3.56
29	0.0019	0.0767	0.0525	0.0028	3.56
30	0.0018	0.0772	0.0496	0.0028	3.56
31	0.0016	0.0777	0.0470	0.0028	3.57
32	0.0015	0.0781	0.0445	0.0028	3.57
33	0.0014	0.0785	0.0422	0.0028	3.58
34	0.0013	0.0789	0.0401	0.0028	3.58
35	0.0012	0.0793	0.0381	0.0028	3.58
36	0.0011	0.0797	0.0362	0.0028	3.59
37	0.0011	0.0800	0.0344	0.0028	3.59
38	0.0010	0.0804	0.0327	0.0028	3.59
39	0.0009	0.0807	0.0312	0.0028	3.59
40	0.0009	0.0810	0.0297	0.0028	3.60

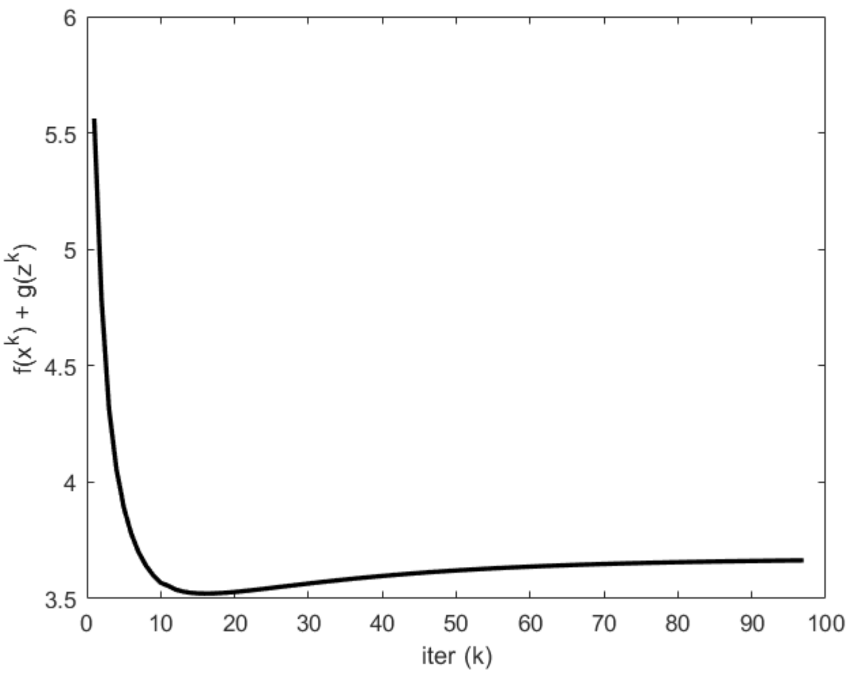
41	0.0008	0.0812	0.0283	0.0028	3.60
42	0.0008	0.0815	0.0270	0.0028	3.60
43	0.0007	0.0818	0.0257	0.0028	3.61
44	0.0007	0.0820	0.0246	0.0028	3.61
45	0.0006	0.0823	0.0234	0.0028	3.61
46	0.0006	0.0825	0.0224	0.0028	3.61
47	0.0006	0.0827	0.0214	0.0028	3.61
48	0.0005	0.0829	0.0204	0.0028	3.62
49	0.0005	0.0831	0.0196	0.0028	3.62
50	0.0005	0.0833	0.0187	0.0028	3.62
51	0.0005	0.0835	0.0179	0.0028	3.62
52	0.0004	0.0836	0.0171	0.0028	3.62
53	0.0004	0.0838	0.0164	0.0028	3.63
54	0.0004	0.0839	0.0157	0.0028	3.63
55	0.0004	0.0841	0.0150	0.0028	3.63
56	0.0004	0.0842	0.0144	0.0028	3.63
57	0.0003	0.0844	0.0138	0.0028	3.63
58	0.0003	0.0845	0.0132	0.0028	3.63
59	0.0003	0.0846	0.0126	0.0028	3.64
60	0.0003	0.0848	0.0121	0.0028	3.64
61	0.0003	0.0849	0.0116	0.0028	3.64
62	0.0003	0.0850	0.0111	0.0028	3.64
63	0.0003	0.0851	0.0107	0.0028	3.64
64	0.0002	0.0852	0.0102	0.0028	3.64
65	0.0002	0.0853	0.0098	0.0028	3.64
66	0.0002	0.0854	0.0094	0.0028	3.64
67	0.0002	0.0855	0.0090	0.0028	3.65
68	0.0002	0.0856	0.0087	0.0028	3.65

69	0.0002	0.0856	0.0083	0.0028	3.65
70	0.0002	0.0857	0.0080	0.0028	3.65
71	0.0002	0.0858	0.0077	0.0028	3.65
72	0.0002	0.0859	0.0074	0.0028	3.65
73	0.0002	0.0859	0.0071	0.0028	3.65
74	0.0002	0.0860	0.0068	0.0028	3.65
75	0.0001	0.0861	0.0065	0.0028	3.65
76	0.0001	0.0861	0.0063	0.0028	3.65
77	0.0001	0.0862	0.0060	0.0028	3.65
78	0.0001	0.0863	0.0058	0.0028	3.66
79	0.0001	0.0863	0.0056	0.0028	3.66
80	0.0001	0.0864	0.0053	0.0028	3.66
81	0.0001	0.0864	0.0051	0.0028	3.66
82	0.0001	0.0865	0.0049	0.0028	3.66
83	0.0001	0.0865	0.0047	0.0028	3.66
84	0.0001	0.0866	0.0046	0.0028	3.66
85	0.0001	0.0866	0.0044	0.0028	3.66
86	0.0001	0.0866	0.0042	0.0028	3.66
87	0.0001	0.0867	0.0040	0.0028	3.66
88	0.0001	0.0867	0.0039	0.0028	3.66
89	0.0001	0.0868	0.0037	0.0028	3.66
90	0.0001	0.0868	0.0036	0.0028	3.66
91	0.0001	0.0868	0.0034	0.0028	3.66
92	0.0001	0.0869	0.0033	0.0028	3.66
93	0.0001	0.0869	0.0032	0.0028	3.66
94	0.0001	0.0869	0.0031	0.0028	3.66
95	0.0001	0.0870	0.0029	0.0028	3.66
96	0.0001	0.0870	0.0028	0.0028	3.66

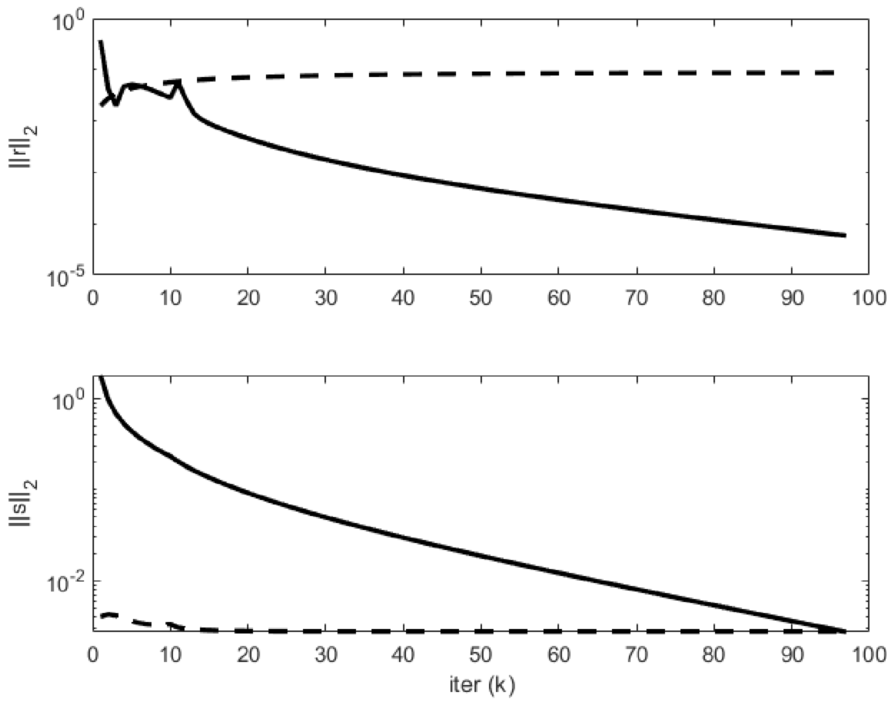
```
 97       0.0001      0.0870      0.0027      0.0028        3.66
Elapsed time is 0.258555 seconds.
```

Reporting

```
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)');
```



6
5.5
5
4.5
4
3.5
$f(x^k) + g(z^k)$
0
10
20
30
40
50
60
70
80
90
100
iter (k)



10^0
10^{-5}
$\|r\|_2$
10^0
10^{-2}
$\|s\|_2$
0
10
20
30
40
50
60
70
80
90
100
iter (k)

Support Vector Machine(SVM):

Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems it is best suited for classification also. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult when the number of features exceeds three.

Matlab Implementation:

```
function [xave, history] = linear_svm(A, lambda, p, rho, alpha)

t_start = tic;
```

Global constants and defaults

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

Data preprocessing

```
[m, n] = size(A);
N = max(p);
% group samples together
for i = 1:N,
    tmp{i} = A(p==i,:);
end
A = tmp;
```

ADMM solver

```
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 Problem using ADMM:

```
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));
```

Solve problem

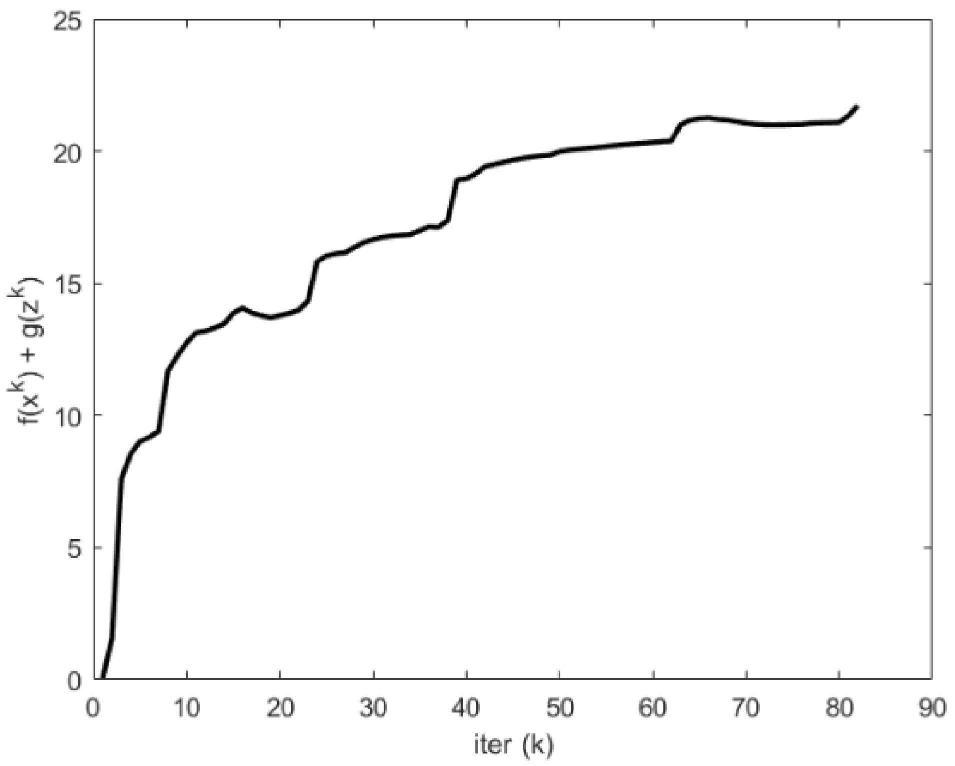
```
[x history] = linear_svm(A, lambda, p, 1.0, 1.0);
```

```
iter        r norm        eps pri         s norm        eps dual       objective
  1         2.9530        0.0297          1.4676        0.0297           0.05
  2         1.9801        0.0331          1.6816        0.0488           1.58
  3         1.0197        0.0458          1.3087        0.0532           7.60
  4         0.9436        0.0561          1.0746        0.0554           8.54
  5         0.9444        0.0649          0.9008        0.0562           9.01
  6         0.9496        0.0719          0.7339        0.0560           9.17
  7         0.9143        0.0770          0.5603        0.0605           9.40
  8         0.7525        0.0804          0.3914        0.0668          11.68
  9         0.7373        0.0821          0.2032        0.0728          12.25
 10         0.6569        0.0823          0.0943        0.0782          12.75
 11         0.6171        0.0816          0.1587        0.0832          13.13
 12         0.5971        0.0803          0.1992        0.0881          13.18
 13         0.5683        0.0783          0.2423        0.0929          13.31
 14         0.5399        0.0759          0.2734        0.0975          13.46
 15         0.4846        0.0735          0.2743        0.1016          13.87
 16         0.4656        0.0712          0.2565        0.1054          14.07
 17         0.4655        0.0691          0.2169        0.1092          13.88
 18         0.4664        0.0675          0.1637        0.1131          13.78
 19         0.4666        0.0663          0.1247        0.1171          13.69
 20         0.4604        0.0655          0.1159        0.1210          13.78
 21         0.4584        0.0650          0.1244        0.1251          13.86
 22         0.4558        0.0646          0.1200        0.1291          13.98
 23         0.4295        0.0644          0.0946        0.1330          14.32
 24         0.3584        0.0644          0.0765        0.1358          15.82
 25         0.3566        0.0646          0.0512        0.1385          16.04
 26         0.3570        0.0649          0.0467        0.1411          16.12
 27         0.3523        0.0653          0.0613        0.1438          16.16
 28         0.3316        0.0658          0.0627        0.1464          16.36
 29         0.3251        0.0663          0.0694        0.1490          16.54
 30         0.3242        0.0669          0.0708        0.1514          16.66
 31         0.3236        0.0675          0.0686        0.1539          16.74
 32         0.3229        0.0682          0.0649        0.1564          16.79
 33         0.3222        0.0687          0.0596        0.1588          16.82
 34         0.3217        0.0692          0.0522        0.1614          16.84
 35         0.3124        0.0697          0.0457        0.1638          16.99
 36         0.3055        0.0701          0.0513        0.1662          17.14
 37         0.3044        0.0706          0.0537        0.1686          17.12
 38         0.2834        0.0711          0.0566        0.1708          17.37
 39         0.2130        0.0716          0.0769        0.1723          18.91
 40         0.2058        0.0722          0.0844        0.1737          18.96
41        0.1941       0.0728       0.0810       0.1750        19.13
42        0.1833       0.0735       0.0798       0.1762        19.42
43        0.1779       0.0742       0.0842       0.1773        19.50
44        0.1749       0.0749       0.0842       0.1785        19.58
45        0.1741       0.0756       0.0790       0.1796        19.66
46        0.1735       0.0763       0.0716       0.1807        19.73
47        0.1726       0.0769       0.0641       0.1817        19.78
48        0.1714       0.0774       0.0566       0.1828        19.82
49        0.1706       0.0779       0.0483       0.1839        19.85
50        0.1602       0.0783       0.0459       0.1849        19.98
51        0.1571       0.0787       0.0453       0.1859        20.04
52        0.1563       0.0791       0.0442       0.1869        20.07
53        0.1560       0.0795       0.0433       0.1878        20.11
54        0.1560       0.0798       0.0426       0.1887        20.14
55        0.1562       0.0801       0.0416       0.1897        20.18
56        0.1567       0.0804       0.0398       0.1906        20.21
57        0.1572       0.0807       0.0369       0.1915        20.25
```

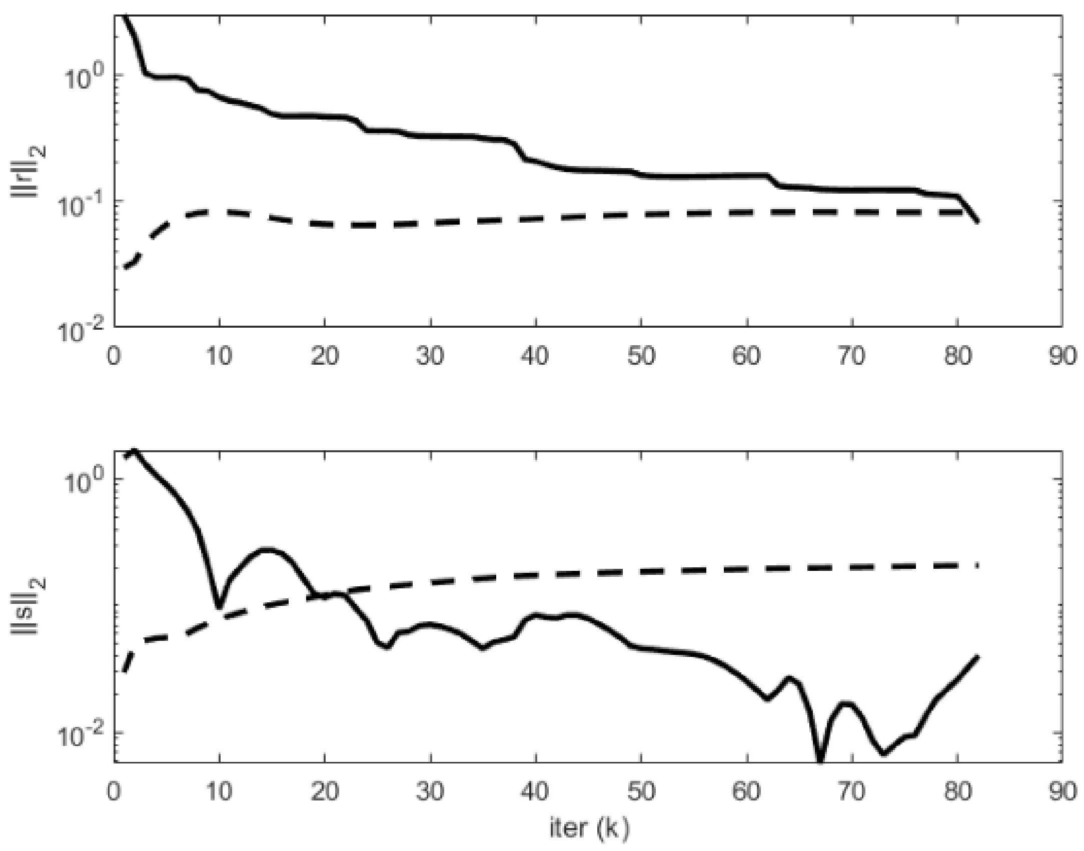
```
 58        0.1578        0.0810        0.0334        0.1924          20.28
 59        0.1583        0.0812        0.0295        0.1933          20.31
 60        0.1586        0.0814        0.0254        0.1942          20.34
 61        0.1587        0.0816        0.0215        0.1950          20.36
 62        0.1587        0.0818        0.0181        0.1959          20.39
 63        0.1318        0.0819        0.0216        0.1965          21.00
 64        0.1285        0.0820        0.0270        0.1971          21.17
 65        0.1276        0.0821        0.0242        0.1976          21.23
 66        0.1263        0.0821        0.0146        0.1981          21.26
 67        0.1235        0.0821        0.0057        0.1986          21.20
 68        0.1226        0.0820        0.0124        0.1991          21.18
 69        0.1219        0.0820        0.0168        0.1997          21.11
 70        0.1217        0.0819        0.0166        0.2003          21.06
 71        0.1219        0.0819        0.0130        0.2010          21.02
 72        0.1221        0.0818        0.0085        0.2016          21.00
 73        0.1221        0.0817        0.0067        0.2023          20.99
 74        0.1219        0.0817        0.0079        0.2029          21.00
 75        0.1218        0.0816        0.0092        0.2036          21.01
 76        0.1217        0.0815        0.0095        0.2043          21.01
 77        0.1139        0.0815        0.0136        0.2049          21.05
 78        0.1126        0.0815        0.0183        0.2055          21.07
 79        0.1110        0.0815        0.0219        0.2061          21.08
 80        0.1092        0.0816        0.0261        0.2068          21.09
 81        0.0872        0.0817        0.0325        0.2073          21.34
 82        0.0669        0.0817        0.0405        0.2076          21.72
Elapsed time is 779.912628 seconds.
```

Reporting

```
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)');
```



Intersection of Polyhedra:

The intersection detection problem is a subset of collision detection for more complex forms, which plays a major role in robotics, computer animation, and mechanical simulation. Intersection detection is also used in computer graphics as a component of accelerated data structures such as bounding volume hierarchies and Kd-trees. In the latter scenario, an item of interest (such as a ray, a view frustum, or another hierarchy) is compared to the geometric forms that bind each node in the hierarchy to see if they overlap. Simple forms (boxes, spheres) for which quick intersection detection algorithms exist are used as bounding shapes.

Matlab Implementation:

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

t_start = tic;
```

Global constants and defaults

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

n = size(A1,2);
```

ADMM solver

```
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 Problem using ADMM:

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

n = 5;       % dimension of variable
m1 = 16;     % number of faces for polyhedra 1
m2 = 18;     % 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
```

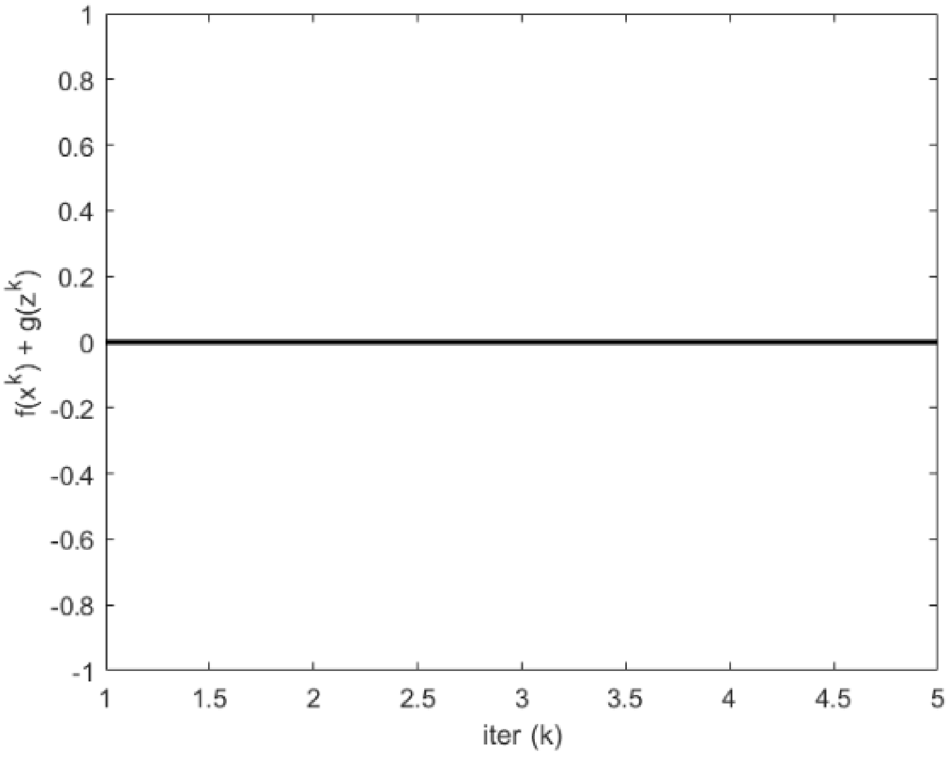
Solve problem

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

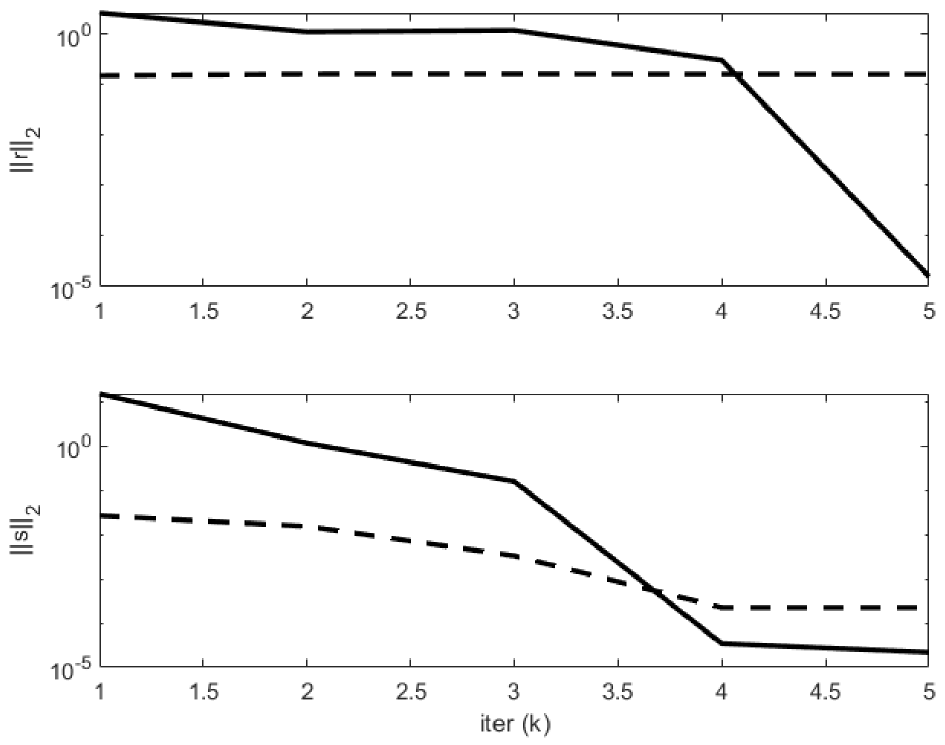
```
iter        r norm        eps pri        s norm       eps dual      objective
  1         2.6152        0.1501        14.9907        0.0264           0.00
  2         1.1149        0.1612         1.1554        0.0152           0.00
  3         1.1944        0.1618         0.1560        0.0033           0.00
  4         0.3059        0.1600         0.0000        0.0002           0.00
  5         0.0000        0.1595         0.0000        0.0002           0.00
Elapsed time is 5.307592 seconds.
```

Reporting

```
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)');
```



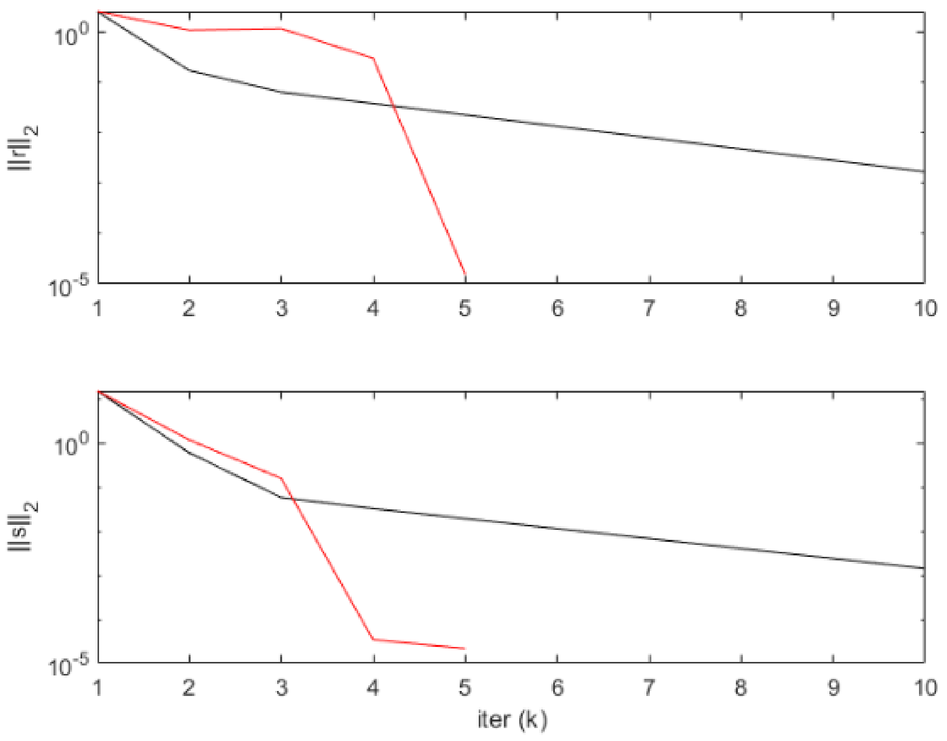
```
g = figure;
subplot(2,1,1);
semilogy(1:K, max(1e-8, history.r_norm), 'k', ...
    1:K, history.eps_pri, 'k--',  'LineWidth', 2);
ylabel('||r||_2');

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)');
```



Compare to alternating projections

```
% Compare to alternating projections

MAX_ITER = 10;

x = zeros(n,1);

z = zeros(n,1);

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))
        subject to
            A1*x <= b1
      cvx_end
      % z-update with relaxation
      zold = z;
      % use cvx to find point in second polyhedra
      cvx_begin quiet
        variable z(n)
        minimize (sum_square(x - z))
        subject to
            A2*z <= b2
      cvx_end
    history1.r_norm(k)  = norm(x - z);
    history1.s_norm(k)  = norm((z - zold));
end
g = figure;
subplot(2,1,1);
semilogy(1:MAX_ITER, max(1e-8, history1.r_norm), 'k', 1:K, max(1e-8,
history.r_norm), 'r');
ylabel('||r||_2');
subplot(2,1,2);
semilogy(1:MAX_ITER, max(1e-8, history1.s_norm), 'k', 1:K, max(1e-8,
history.s_norm), 'r');
ylabel('||s||_2'); xlabel('iter (k)');
```



Sparse Inverse Covariance Selection:

It is a sequence of positive definite inverse covariance matrices that converge to a sparse matrix, while the sequence of Y k's is a sequence of sparse matrices that converges to a positive definite inverse covariance matrix. To estimate a probabilistic model (e.g. a Gaussian model), estimating the precision matrix, that is the inverse covariance matrix, is as important as estimating the covariance matrix. Indeed a Gaussian model is parametrized by the precision matrix. Given an empirical covariance matrix S,

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top.$$

we find a sparse inverse covariance matrix P to represent the data.

$$\min_{\mathbf{P}\succ 0} F(\mathbf{P}) \overset{\text{def}}{=} L(\mathbf{P}) + \lambda \|\mathrm{vec}(\mathbf{P})\|_1, \quad L(\mathbf{P}) = -\log\det(\mathbf{P}) + \mathrm{trace}(\mathbf{SP}).$$

where L is the negative log-likelihood function and the l_1 term is a sparsity inducing regularizer.

Matlab Implementation:

```
function [Z, history] = covsel(D, lambda, rho, alpha)

t_start = tic;
```

Global constants and defaults

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

Data preprocessing

```
S = cov(D);
n = size(S,1);
```

ADMM solver

```
X = zeros(n);
Z = zeros(n);
U = zeros(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
    [Q,L] = eig(rho*(Z - U) - S);
    es = diag(L);
    xi = (es + sqrt(es.^2 + 4*rho))./(2*rho);
    X = Q*diag(xi)*Q';

    % z-update with relaxation
    Zold = Z;
    X_hat = alpha*X + (1 - alpha)*Zold;
    Z = shrinkage(X_hat + U, lambda/rho);

    U = U + (X_hat - Z);

    % diagnostics, reporting, termination checks

    history.objval(k)  = objective(S, X, Z, lambda);

    history.r_norm(k)  = norm(X - Z, 'fro');
    history.s_norm(k)  = norm(-rho*(Z - Zold),'fro');

    history.eps_pri(k) = sqrt(n*n)*ABSTOL + RELTOL*max(norm(X,'fro'),
norm(Z,'fro'));
    history.eps_dual(k)= sqrt(n*n)*ABSTOL + RELTOL*norm(rho*U,'fro');


    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(S, X, Z, lambda)
    obj = trace(S*X) - log(det(X)) + lambda*norm(Z(:), 1);
end

function y = shrinkage(a, kappa)
    y = max(0, a-kappa) - max(0, -a-kappa);
end
```

Example Problem using ADMM:

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

n = 100;   % number of features
N = 10*n;  % number of samples

% generate a sparse positive definite inverse covariance matrix
Sinv      = diag(abs(ones(n,1)));
idx       = randsample(n^2, 0.001*n^2);
Sinv(idx) = ones(numel(idx), 1);
Sinv = Sinv + Sinv';   % make symmetric
if min(eig(Sinv)) < 0  % make positive definite
    Sinv = Sinv + 1.1*abs(min(eig(Sinv)))*eye(n);
end
S = inv(Sinv);

% generate Gaussian samples
D = mvnrnd(zeros(1,n), S, N);
```

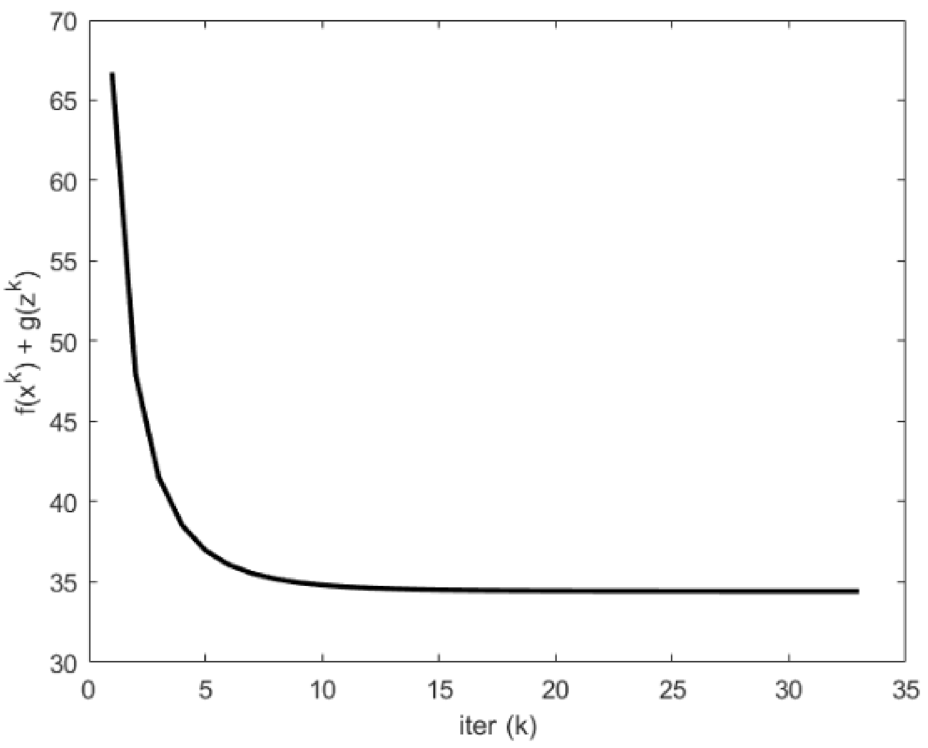
Solve problem

```
[X, history] = covsel(D, 0.01, 1, 1);
```

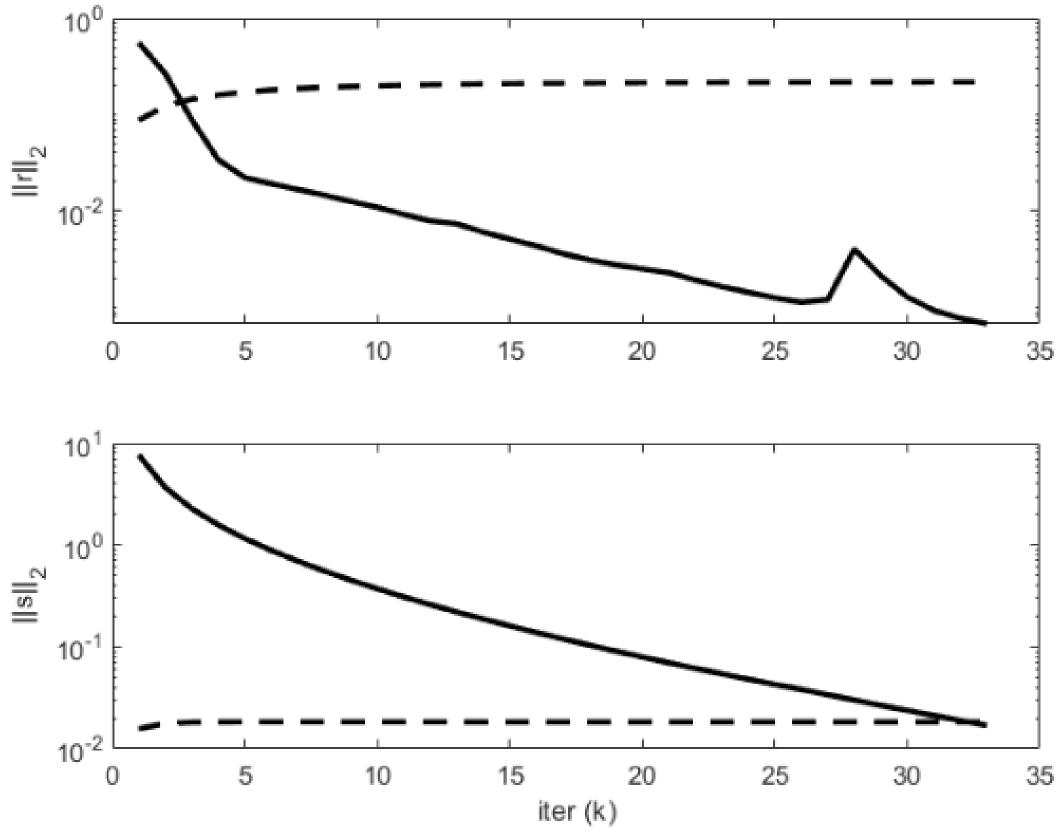
```
iter        r norm       eps pri        s norm       eps dual      objective
  1         0.5604        0.0878        7.6521        0.0156        66.74
  2         0.2677        0.1230        3.6651        0.0176        47.97
  3         0.0890        0.1453        2.2727        0.0181        41.51
  4         0.0341        0.1608        1.5731        0.0182        38.54
  5         0.0223        0.1720        1.1563        0.0183        36.98
  6         0.0193        0.1806        0.8831        0.0183        36.08
  7         0.0168        0.1872        0.6927        0.0183        35.54
  8         0.0147        0.1924        0.5544        0.0183        35.19
  9         0.0126        0.1967        0.4507        0.0183        34.97
 10         0.0110        0.2001        0.3710        0.0183        34.82
 11         0.0092        0.2029        0.3087        0.0183        34.71
 12         0.0079        0.2053        0.2592        0.0183        34.64
 13         0.0074        0.2072        0.2192        0.0183        34.59
 14         0.0060        0.2088        0.1867        0.0183        34.55
 15         0.0051        0.2102        0.1600        0.0183        34.52
 16         0.0044        0.2114        0.1378        0.0183        34.50
 17         0.0036        0.2124        0.1192        0.0183        34.48
 18         0.0031        0.2132        0.1036        0.0183        34.47
 19         0.0028        0.2140        0.0904        0.0183        34.46
 20         0.0025        0.2146        0.0792        0.0182        34.46
 21         0.0023        0.2151        0.0696        0.0182        34.45
 22         0.0019        0.2156        0.0613        0.0182        34.45
 23         0.0016        0.2160        0.0542        0.0182        34.44
 24         0.0014        0.2163        0.0480        0.0182        34.44
 25         0.0013        0.2166        0.0426        0.0182        34.44
 26         0.0011        0.2169        0.0379        0.0182        34.44
 27         0.0012        0.2171        0.0337        0.0182        34.44
 28         0.0040        0.2173        0.0299        0.0182        34.44
 29         0.0021        0.2175        0.0265        0.0182        34.44
 30         0.0013        0.2177        0.0236        0.0182        34.43
 31         0.0009        0.2178        0.0211        0.0182        34.43
 32         0.0008        0.2179        0.0188        0.0182        34.43
 33         0.0007        0.2181        0.0169        0.0182        34.43
Elapsed time is 0.256118 seconds.
```

Reporting

```
K = length(history.objval);
X_admm = X;

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)');
```



10^0
10^{-2}
$||r||_2$
0
5
10
15
20
25
30
35
10^1
10^0
10^{-1}
10^{-2}
$||s||_2$
iter (k)

Inference:

- ADMM has the benefit of being extremely simple to implement, and it maps onto several standard distributed programming models reasonably well.

- ADMM was developed over a generation ago, with its roots stretching far in advance of the Internet, distributed and cloud computing systems, massive high-dimensional datasets, and the associated large scale applied statistical problems.

- Despite this, it appears to be well suited to the modern regime and has the important benefit of being quite general in its scope and applicability.

- ADMM builds on existing algorithms for single machines, and so can be viewed as a modular coordination algorithm that ‘incentivizes’ a set of simpler algorithms to collaborate to solve much larger global problems together than they could on their own.

Conclusion: