MATEMATICS OF INTELLIGENT SYSTEMS 3

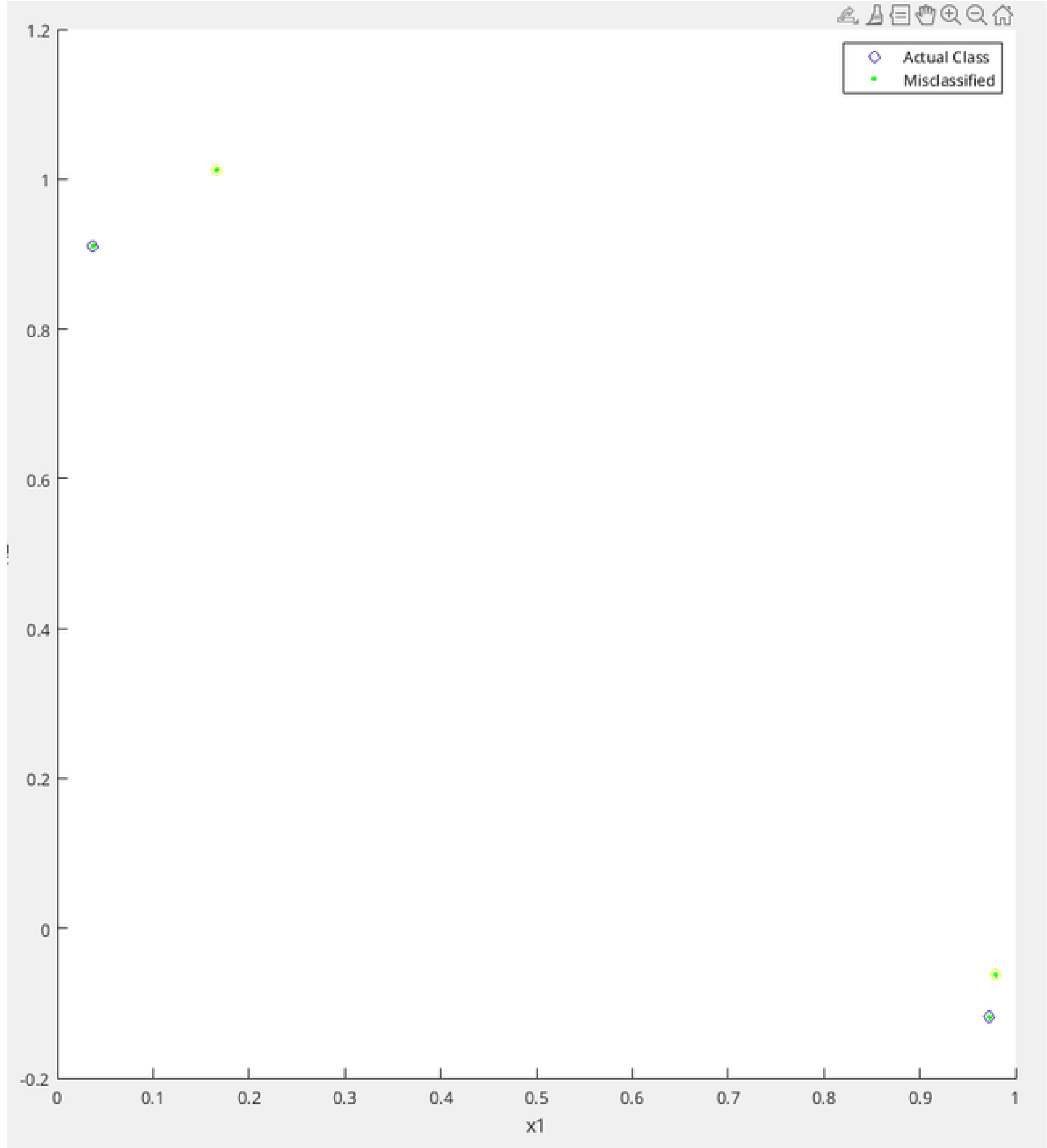
Sarvesh ShashiKumar
CB.EN.U4AIE21163

LINEAR SVM USING HARD MARGIN

$$\min_{\boldsymbol{w},b} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$$

$$\textbf{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1$$

```matlab
load checkerboard_dataset.mat

X = X;
y = y;
% Define the optimization problem
cvx_begin
    variables w(2) b xi(size(X,1))
    minimize(norm(xi,1))
    subject to
        y.*(X*w + b) >= 1 - xi
        xi >= 0
cvx_end

% Extract the weights and bias from the solution
w =w;
b = b;
% Classify the data points using the hyperplane equation
predictions = sign(X*w + b);

% Plot the results
scatter(X(:,1), X(:,2), [], y)
hold on
plot(X(predictions ~= y,1), X(predictions ~= y,2), 'xr')
plot(X(predictions == y,1), X(predictions == y,2), '.g')
xlabel('x1')
ylabel('x2')
legend('Actual Class', 'Misclassified', 'Correctly Classified')
```

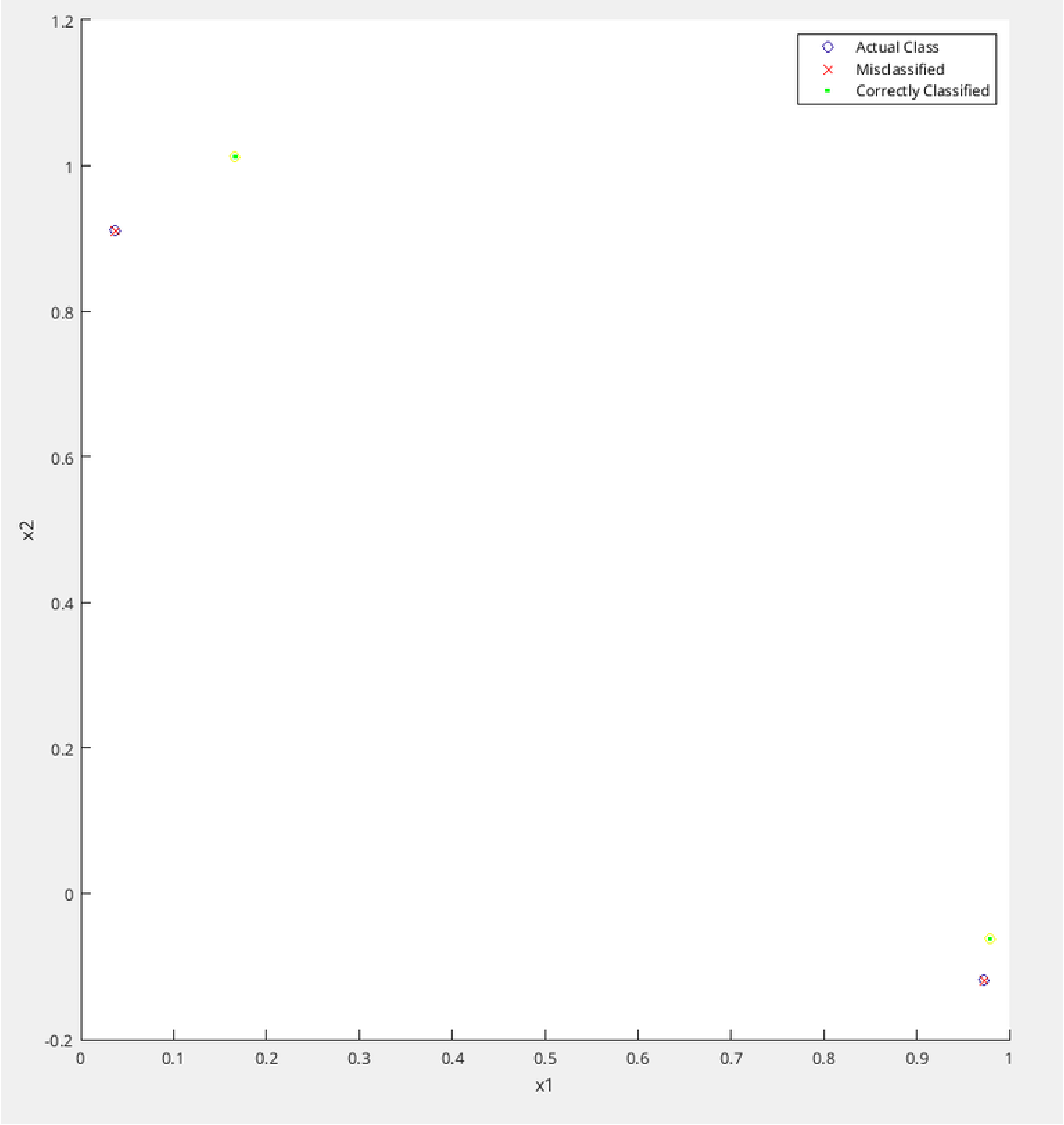


LINEAR SVM USING SOFT MARGIN

$$\min \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\zeta_i$$

$$\textbf{s.t.} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \zeta_i \quad \forall i = 1, \ldots, n, \; \zeta_i \geq 0$$

```
load checkerboard_dataset.mat

X = X;
y = y;
% Set the regularization parameter
lambda = 1;

% Define the optimization problem
cvx_begin
    variables w(2) b xi(size(X,1))
    minimize(norm(w,2) + lambda*sum(xi))
    subject to
        y.*(X*w + b) >= 1 - xi
        xi >= 0
cvx_end

% Extract the weights and bias from the solution
w=w;
b=b;

% Classify the data points using the hyperplane equation
predictions = sign(X*w + b);

% Plot the results
scatter(X(:,1), X(:,2), [], y)
hold on
plot(X(predictions ~= y,1), X(predictions ~= y,2), 'xr')
plot(X(predictions == y,1), X(predictions == y,2), '.g')
xlabel('x1')
ylabel('x2')
legend('Actual Class', 'Misclassified', 'Correctly Classified')

%print accuracy
accuracy = sum(predictions == y)/length(y);
fprintf('Accuracy: %f percent \n', accuracy*100);
```



LINEAR L2 SVM

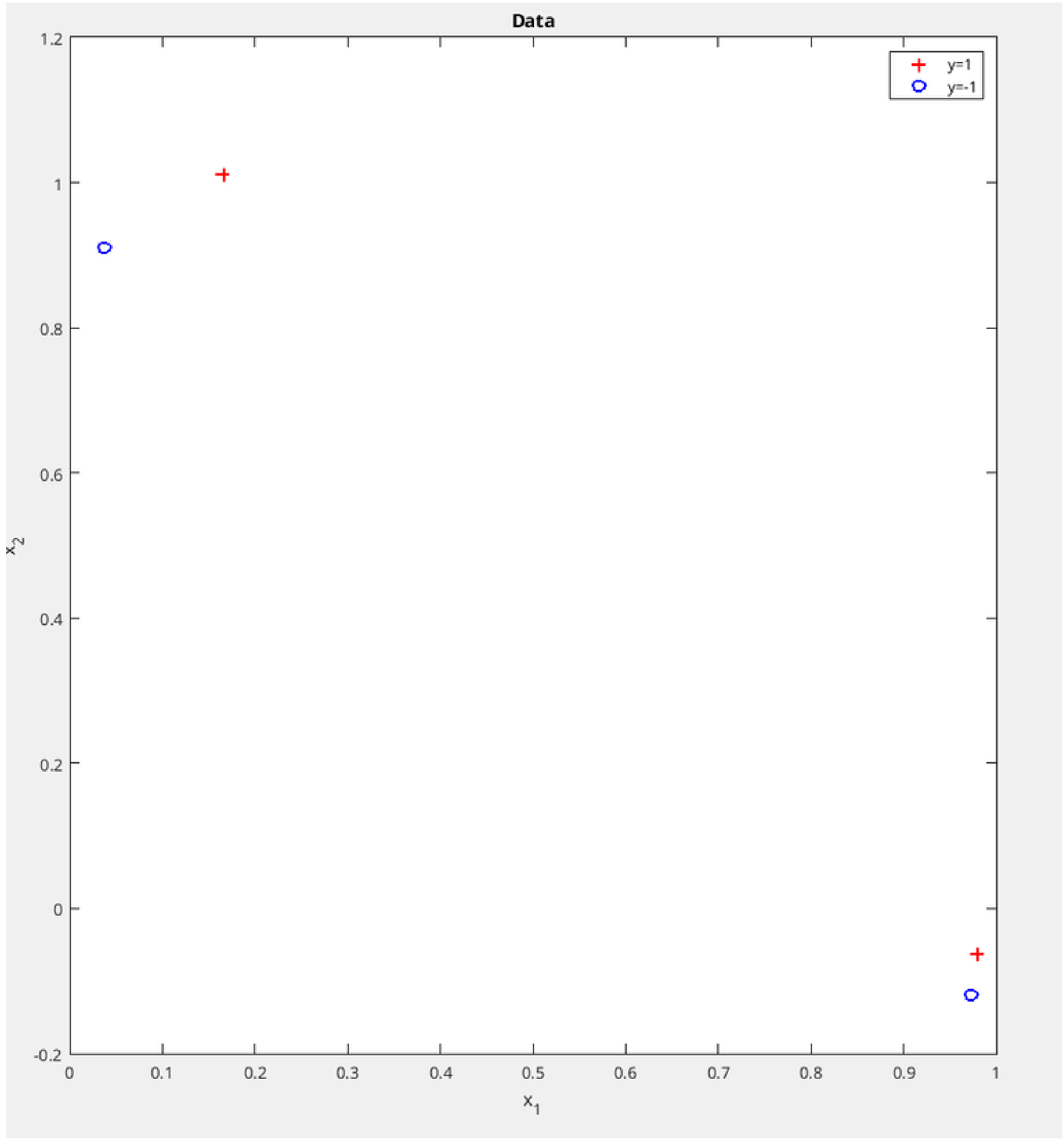
$$\min_{\mathbf{w},\gamma,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2$$

subject to

$$d_i(\mathbf{w}^T\mathbf{x}_i - \gamma) + \xi_i - 1 \geq 0, \quad 1 \leq i \leq m$$

$$\xi_i \geq 0, \quad 1 \leq i \leq m$$

```
load checkerboard_dataset.mat

X = X;
y = y;

C = 1;

% Define the variables
n = size(X,1); % number of samples
d = size(X,2); % number of features

% Define the objective function and constraints
cvx_begin
    variable w(d)
    variable b
    variable e(n)
    minimize( 0.5 * w'*w + 0.5 * C * sum(e)^2 )
    subject to
        y.*(X*w - b)+ e -1 >= 0
cvx_end

%plotting the data
figure(1)
plot(X(y==1,1),X(y==1,2),'r+','LineWidth',2,'MarkerSize',7);
hold on
plot(X(y==-1,1),X(y==-1,2),'bo','LineWidth',2,'MarkerSize',7);
hold on
legend('y=1','y=-1')
xlabel('x_1')
ylabel('x_2')
title('Data')
```



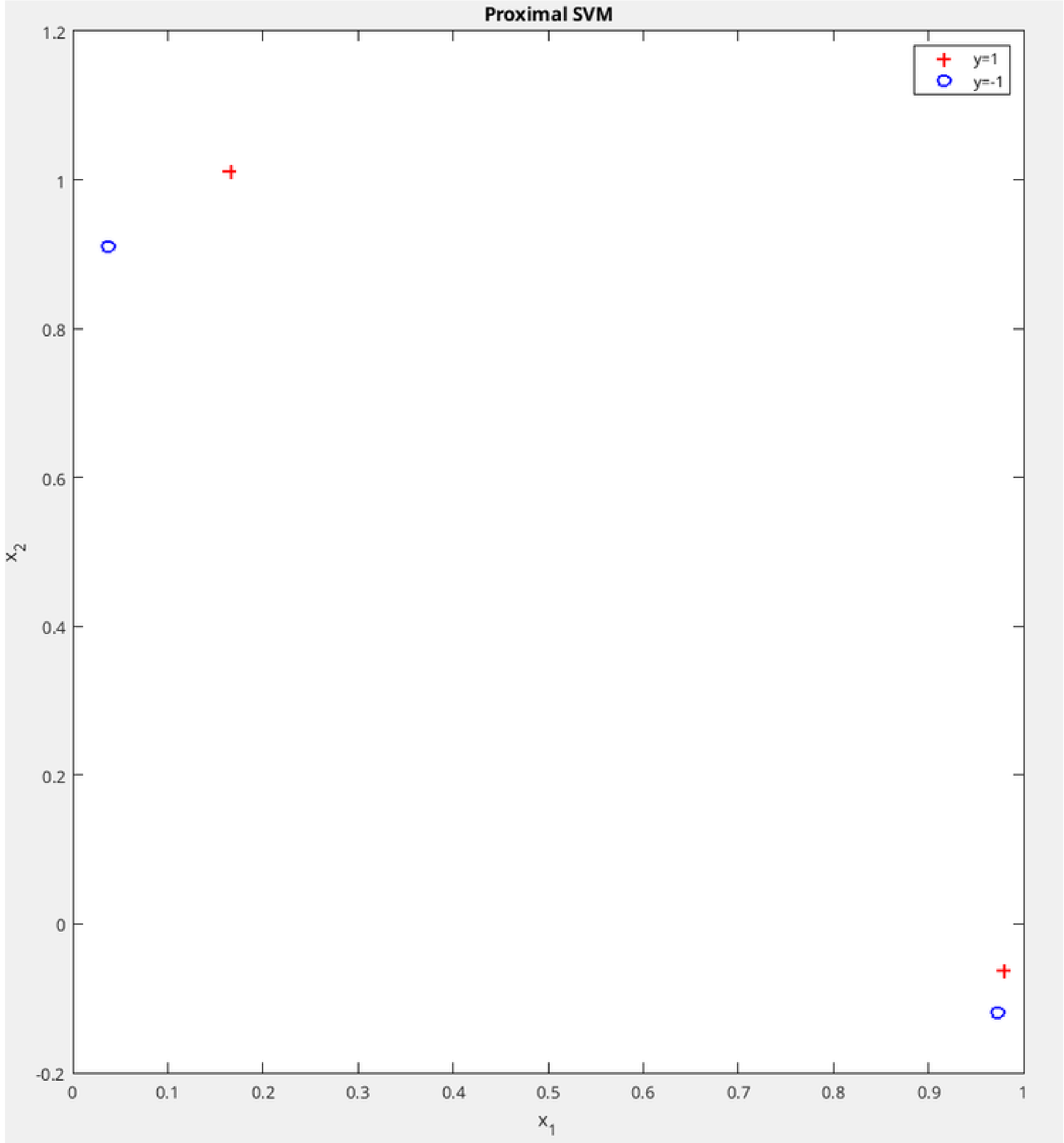
PROXIMAL SVM

$$\min_{\mathbf{w},\gamma,\xi} \frac{1}{2}(\mathbf{w}^T\mathbf{w}+\gamma^2)+\frac{C}{2}\sum_{i=1}^{m}\xi_i^2$$

subject to

$$d_i(\mathbf{w}^T\mathbf{x}_i-\gamma)+\xi_i-1=0,\quad 1\le i\le$$

```
load checkerboard_dataset.mat

X = X;
y = y;
C = 1;

% Define the variables
n = size(X,1); % number of samples
d = size(X,2); % number of features

% Define the objective function and constraints
cvx_begin
    variable w(d)
    variable b
    variable e(n)
    minimize( 0.5 * ( w'*w + b^2 )+ 0.5 * C * sum(e)^2 )
    subject to
        y.*(X*w - b)+ e -1 >= 0
cvx_end

%plotting the data
figure(1)
plot(X(y==1,1),X(y==1,2),'r+','LineWidth',2,'MarkerSize',7);
hold on
plot(X(y==-1,1),X(y==-1,2),'bo','LineWidth',2,'MarkerSize',7);
hold on
legend('y=1','y=-1')
xlabel('x_1')
ylabel('x_2')
title('Proximal SVM')
```



RKS SVM

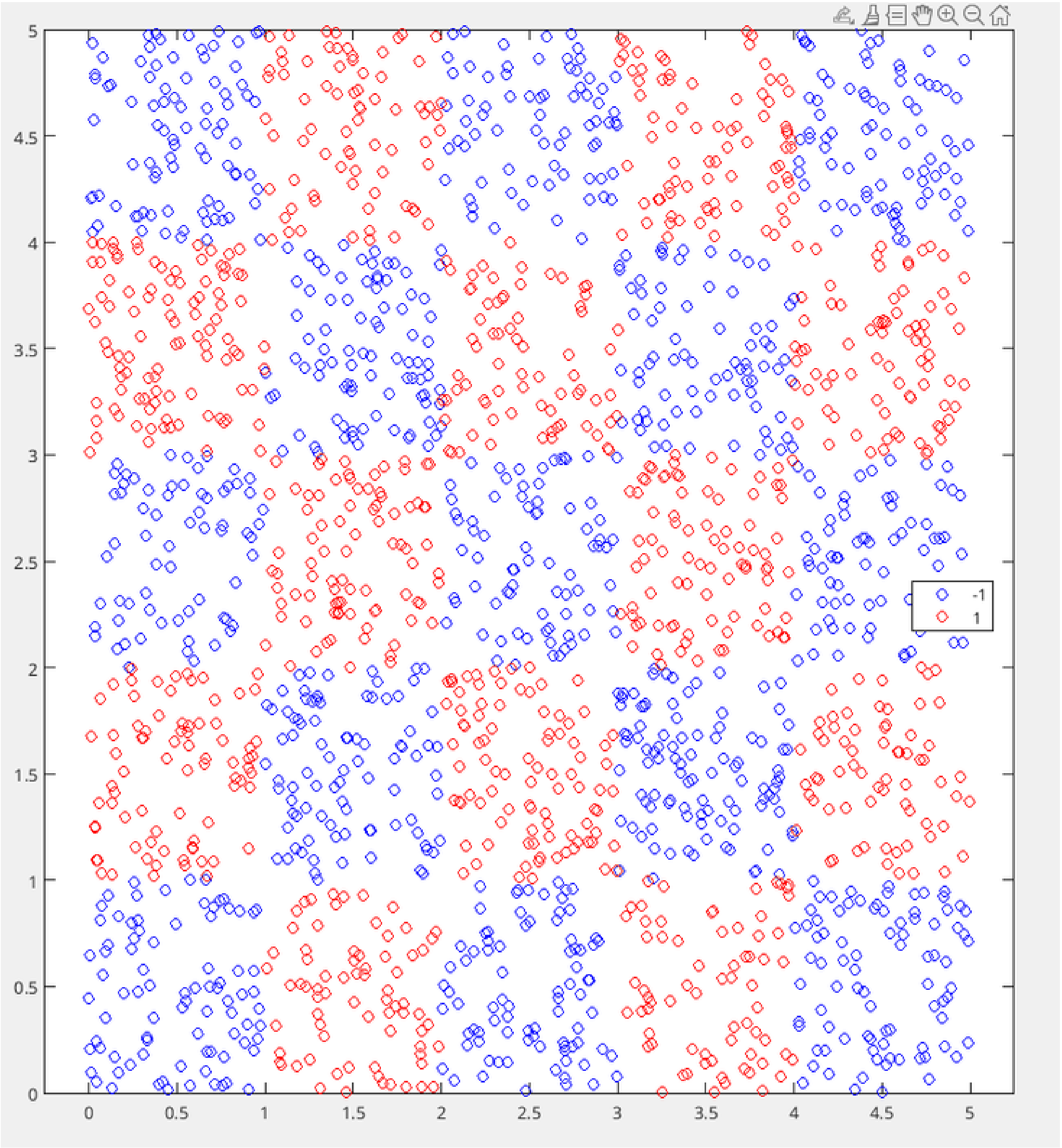
$$\min_{\mathbf{w},\gamma,\xi} \frac{1}{2}(\mathbf{w}^T\mathbf{w}+\gamma^2)+\frac{C}{2}\sum_{i=1}^{m}\xi_i^2$$

subject to

$$d_i(\mathbf{w}^T\mathbf{x}_i-\gamma)+\xi_i-1=0, \quad 1\le i\le$$

```
x = rand(2000,1)*5;
y = rand(2000,1)*5;
c = mod((floor(x)+floor(y)),2);
ind = find(c); a = [x(ind),y(ind)];
ind1 = find(c==0);
b = [x(ind1),y(ind1)];
numRows = size(a, 1);
numRowsi = size(b, 1);
A =cat(1, a, b);
di = ones(numRows, 1);
d2 = -1*ones(numRowsi, 1);
d = cat(1, di, d2);
numRows2 = size(d, 1);
a = 5;
D = diag(d);
figure;
gscatter(A(:,1),A(:,2),d,'br',"o");
hold on
n = size(A,2);
m = size(A,1);
e = ones(m,1);
c = 10000000;
cvx_begin
  variables w(n) g Psi(m)
  minimize ((0.5*w'*w)+(c*sum(Psi)))
  subject to
    D*(A*w-g*e)+Psi-e >= 0;
    Psi >= 0;
cvx_end
% accuracy
z = sign(A*w-g); r = sum(d==z); Acc = (r/m)*100
```

```
Acc =

   52.5500
```



Linear prog

```matlab
function [z, history] = linprog(c, 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

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
```

```matlab
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);
```

Basis Pursit

```matlab
function [z, history] = basis_pursuit(A, b, rho, alpha) %function declaration
t_start = tic; %to measure the time taken by the function
QUIET    = 0; %to print the output or not
MAX_ITER = 1000; %maximum number of iterations
ABSTOL   = 1e-4; %absolute tolerance
RELTOL   = 1e-2; %relative tolerance

[m n] = size(A); %size of the matrix A
x = zeros(n,1); %initializing x,y,z
z = zeros(n,1);
u = zeros(n,1);

if ~QUIET %if QUIET is not 0 then print the following
    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'; %AAt is the matrix A multiplied by its transpose
P = eye(n) - A' * (AAt \ A); %P is the projection matrix
q = A' * (AAt \ b); %q is the projection vector

for k = 1:MAX_ITER %for loop to iterate for MAX_ITER times or until the termination condition is satisfied
    % x-update
    x = P*(z - u) + q;

    % z-update with relaxation
    zold = z; %zold is the previous value of z
    x_hat = alpha*x + (1 - alpha)*zold; %x_hat is the relaxation parameter
    z = shrinkage(x_hat + u, 1/rho); %shrinkage is the soft thresholding function used to update z

    u = u + (x_hat - z); %u is the dual variable which is updated using the relaxation parameter

    % diagnostics, reporting, termination checks
    history.objval(k)  = objective(A, b, x); %objective function is the L1 norm of x

    history.r_norm(k)  = norm(x - z); %r_norm is the norm of the difference between x and z
    history.s_norm(k)  = norm(-rho*(z - zold)); %s_norm is the norm of the difference between z and zold

    history.eps_pri(k) = sqrt(n)*ABSTOL + RELTOL*max(norm(x), norm(-z)); %eps_pri is the tolerance for the primal variable
```

```matlab
    history.eps_pri(k) = sqrt(n)*ABSTOL + RELTOL*max(norm(x), norm(-z)); %eps_pri is the tolerance for the primal variable
    history.eps_dual(k)= sqrt(n)*ABSTOL + RELTOL*norm(rho*u); %eps_dual is the tolerance for the dual variable

    if ~QUIET %if QUIET is not 0 then print the following
        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) %function to calculate the objective function
    obj = norm(x,1);
end

function y = shrinkage(a, kappa) %function to calculate the soft thresholding function
    y = max(0, a-kappa) - max(0, -a-kappa); %soft thresholding function
end
```

Implementation

```
clear all;clc;
rand('seed', 0);
randn('seed', 0);

n = 30;
m = 10;
A = randn(m,n);

x = sprandn(n, 1, 0.1*n);
b = A*x;

xtrue = x;
```

We then call the function Basis_pursuit.m, which contains the ADMM algorithm. We set the parameters rho, alpha to be 1.0. We then save the output of the function as x, and the history of the algorithm as history.

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

.We then plot the objective function value f(x^k) + g(z^k) versus the number of iterations k.

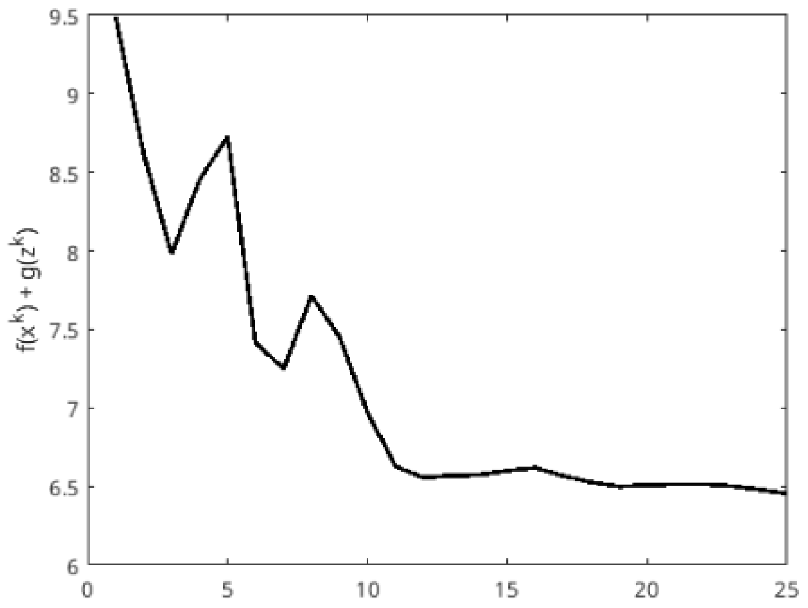
We then plot the residual norm ||r||_2 versus the number of iterations k.

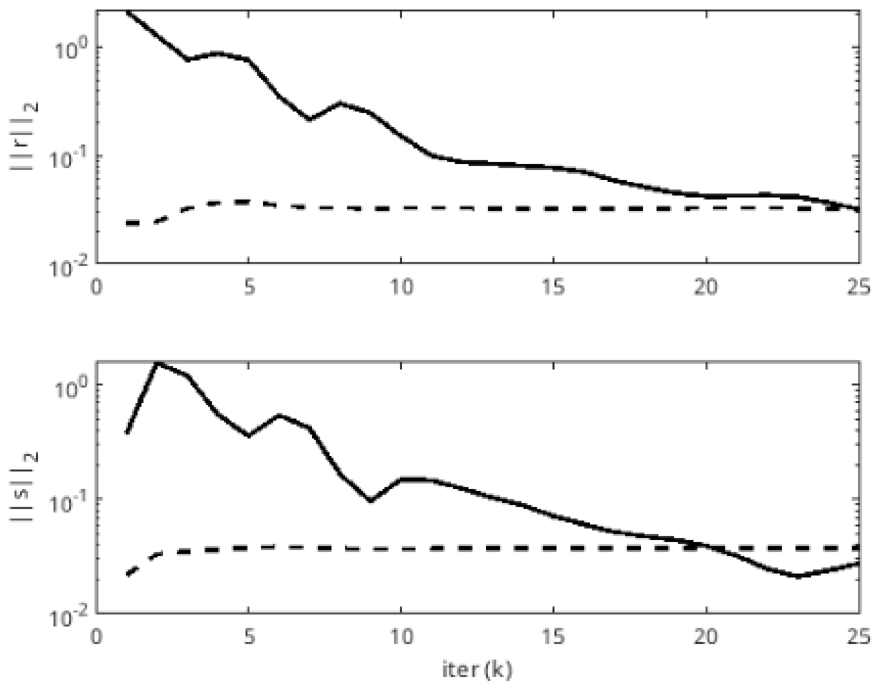
We then plot the residual norm ||s||_2 versus the number of iterations k.

```
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)');
```

2	1.2821	0.0246	1.5779	0.0332
3	0.7697	0.0326	1.2111	0.0354
4	0.8756	0.0371	0.5554	0.0368
5	0.7662	0.0373	0.3617	0.0382
6	0.3543	0.0349	0.5447	0.0387
7	0.2137	0.0334	0.4236	0.0381
8	0.3035	0.0329	0.1683	0.0374
9	0.2479	0.0327	0.0969	0.0371
10	0.1514	0.0328	0.1500	0.0371
11	0.1002	0.0329	0.1479	0.0373
12	0.0870	0.0328	0.1249	0.0375
13	0.0840	0.0327	0.1039	0.0378
14	0.0804	0.0325	0.0895	0.0380
15	0.0774	0.0324	0.0715	0.0381
16	0.0711	0.0324	0.0610	0.0381
17	0.0590	0.0325	0.0522	0.0380
18	0.0513	0.0326	0.0477	0.0381
19	0.0452	0.0327	0.0443	0.0381
20	0.0424	0.0329	0.0394	0.0381
21	0.0426	0.0329	0.0323	0.0381
22	0.0433	0.0329	0.0248	0.0381
23	0.0417	0.0328	0.0214	0.0381
24	0.0373	0.0327	0.0240	0.0382
25	0.0319	0.0326	0.0277	0.0382

Elapsed time is 0.018553 seconds.





Lasso

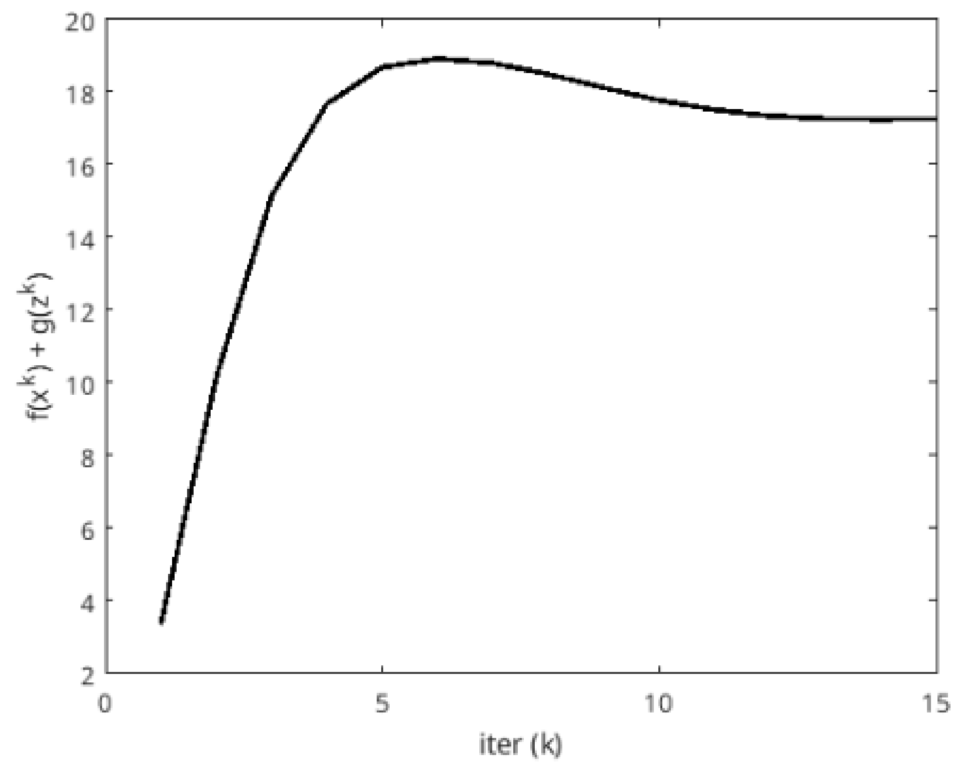
```matlab
function [z, history] = lasso(A, b, lambda, rho, alpha)
t_start = tic;
QUIET    = 0;
MAX_ITER = 1000;
ABSTOL   = 1e-4;
RELTOL   = 1e-2;

[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)...
```

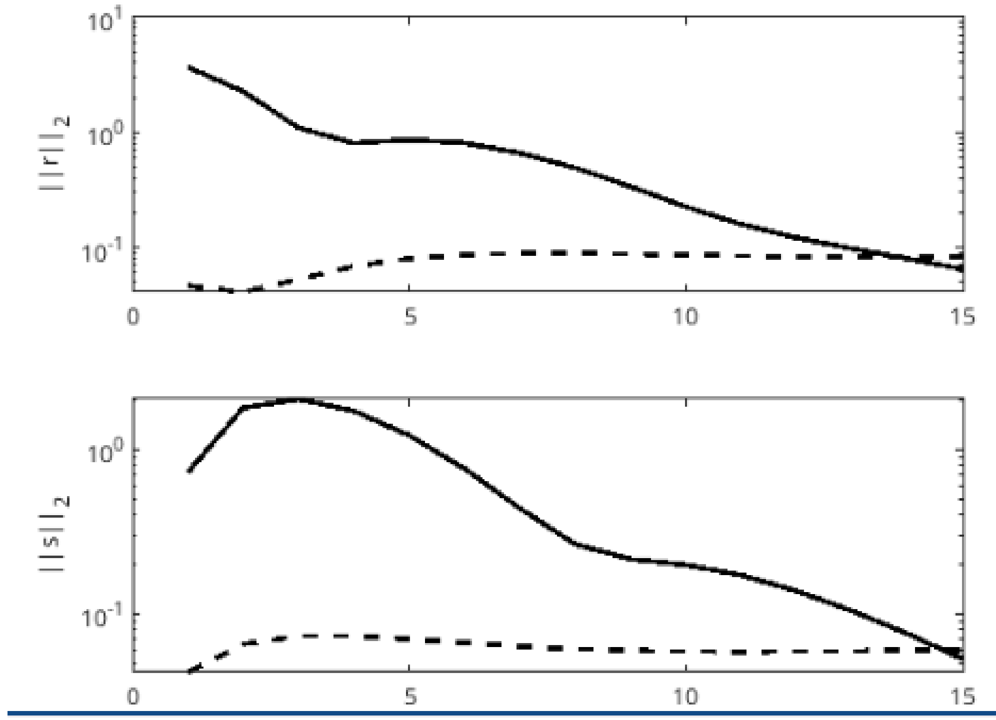
Lasso Implementation

```
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)');
```

iter	r norm	eps pri	s norm	eps dual
1	3.7048	0.0465	0.7250	0.0441
2	2.2654	0.0409	1.7960	0.0653
3	1.0958	0.0529	2.0325	0.0734
4	0.8050	0.0687	1.7219	0.0736
5	0.8619	0.0801	1.2234	0.0704
6	0.8078	0.0864	0.7669	0.0667
7	0.6611	0.0889	0.4398	0.0635
8	0.4906	0.0890	0.2659	0.0612
9	0.3379	0.0878	0.2159	0.0598
10	0.2255	0.0861	0.1987	0.0591
11	0.1585	0.0845	0.1721	0.0590
12	0.1212	0.0833	0.1379	0.0591
13	0.0979	0.0825	0.1044	0.0595
14	0.0799	0.0820	0.0759	0.0598
15	0.0650	0.0819	0.0532	0.0602

Elapsed time is 0.383823 seconds.





Thank You