21MAT204

Mathematics For Intelligent Systems-3

KALYANA SUNDARAM
CB.EN.U4AIE21120

QUESTION 1

Implement the following SVM versions using CVX for binary datasets (use appropriate binary datasets. Examples: Checkerboard, Spiral, Ring data or any 2 class datasets from net

1. Linear Hard-Margin SVM (L1 SVM)
2. Linear Soft-Margin SVM (L1 SVM)
3. Linear L2 SVM
4. Proximal SVM
5. Non-linear SVM using RBF, Polynomial Kernels
6. Non-linear SVM using RKS

LINEAR HARD-MARGIN SVM (L1 SVM)

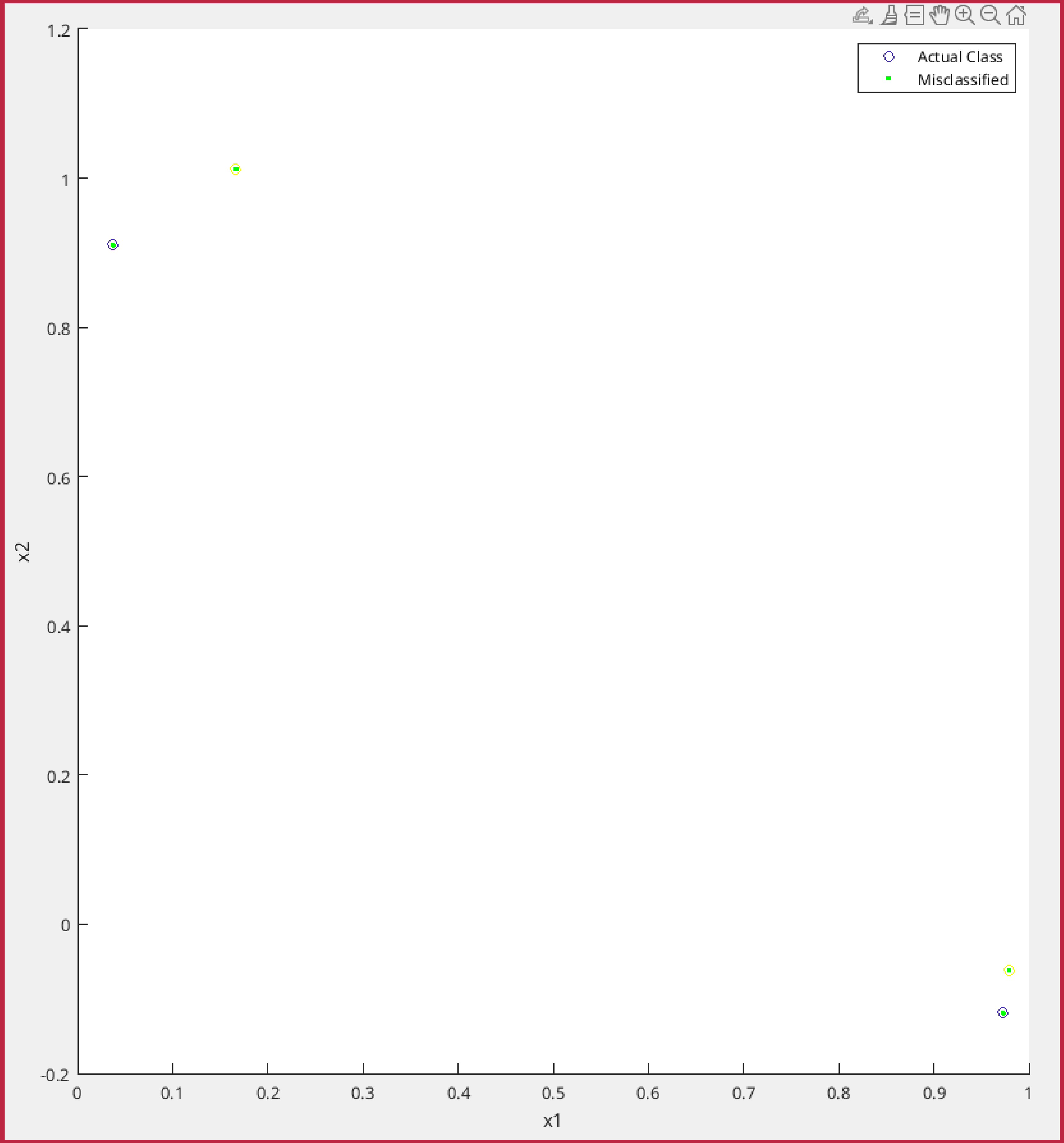
- Linear Hard-Margin Support Vector Machines (L1 SVM) is a classification algorithm that tries to find the hyperplane in an N-dimensional space (N — the number of features) that maximally separates the two classes.
- The distance between the hyperplane and the nearest data point from either class is known as the margin.
- Hard-margin SVM can only be used when the data is linearly separable, meaning that it can be separated into two classes by a single straight line.
- One of the main characteristics of hard-margin SVM is that it tries to maximize the margin between the two classes, which makes it more robust to noise.

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

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

l1svm-har-margin.m

```
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 SOFT-MARGIN SVM (L1 SVM)

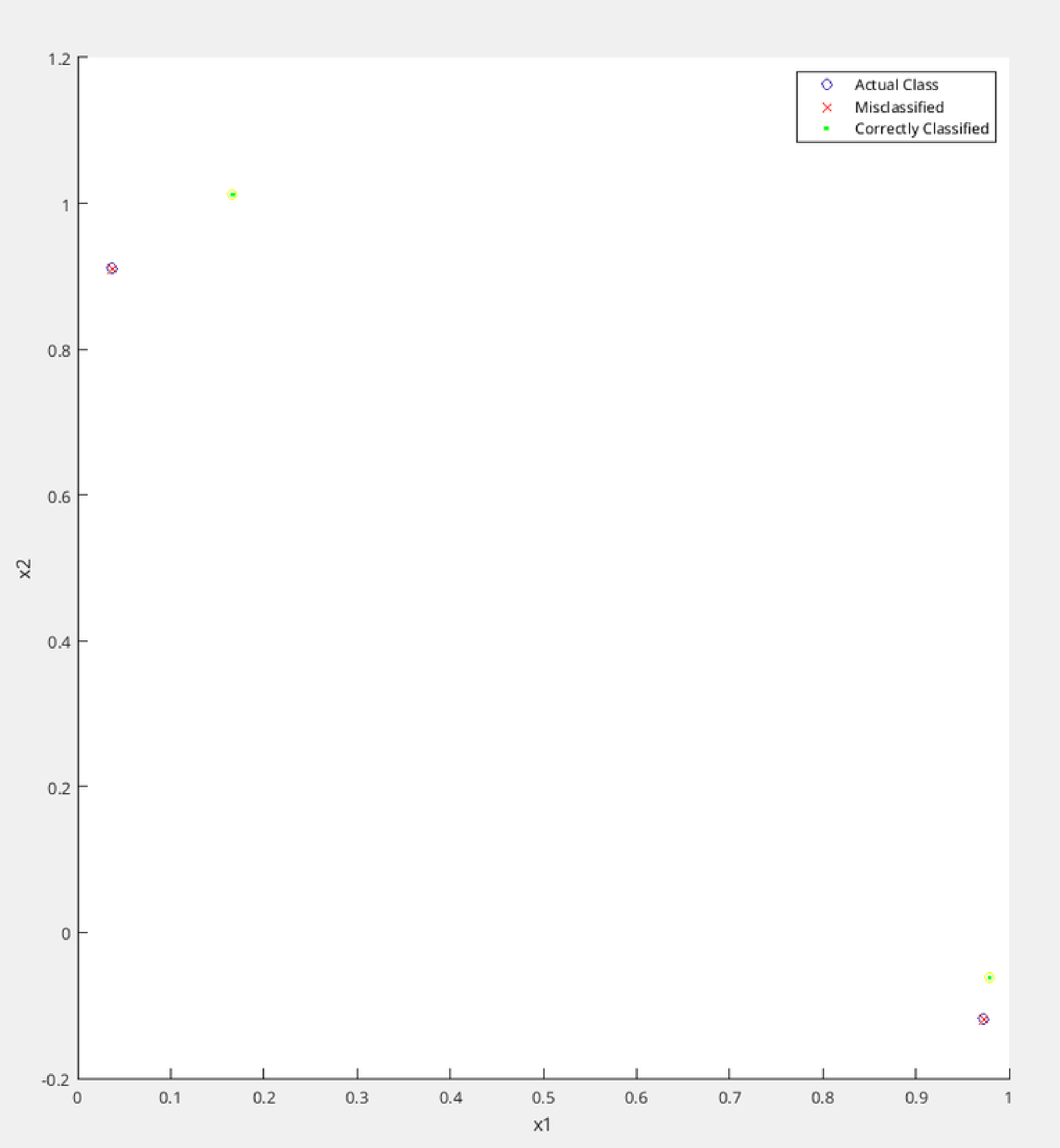
- Linear Soft-Margin Support Vector Machines (L1 SVM) is a classification algorithm that tries to find the hyperplane in an N-dimensional space (N — the number of features) that maximally separates the two classes while allowing for some misclassification of the data points.
- The distance between the hyperplane and the nearest data point from either class is known as the margin.
- One of the main characteristics of soft-margin SVM is that it can handle non-linearly separable data by allowing for some misclassification.

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

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

l1svm-har-margin.m   l1svm-soft-margin.m

```matlab
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);
```




```
Status: Solved
Optimal value (cvx_optval): +4

Accuracy: 50.000000 percent
```

LINEAR L2 SVM

- L2 support vector machine is a soft-margin SVM which optimizes the sum of squared errors.
- Just as in L1 SVM, the first term in the objective function minimizes the margin, the second term controls the number of misclassifications and C represents the trade-off between the two.
- The only difference between L1 and L2 SVM is that L2 SVM uses the sum of squared slack variables.
- This difference leads to some interesting properties in dual space.

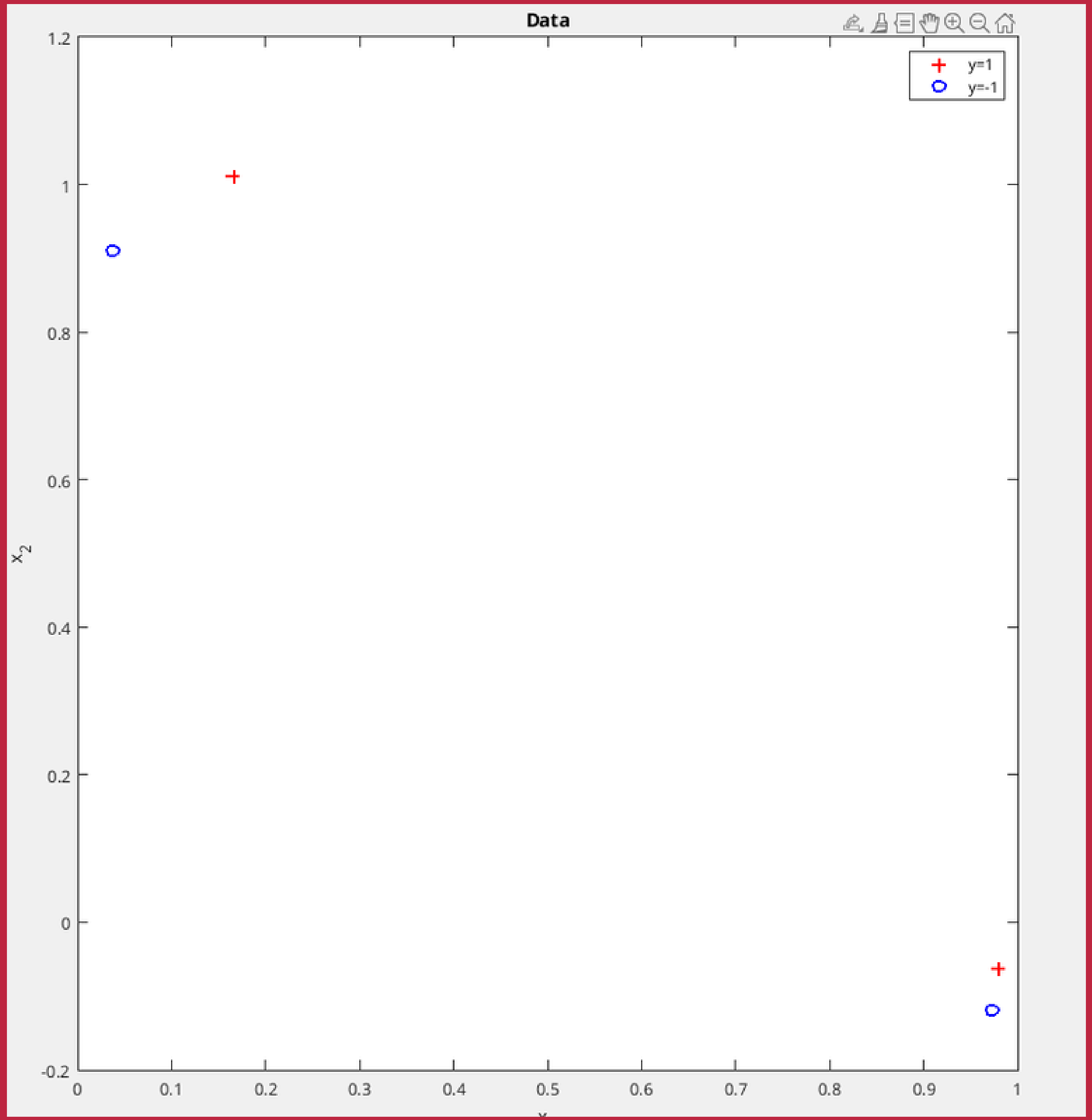
$$\min_{\mathbf{w},\gamma,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2$$

subject to

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

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

```matlab
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')
```




```
Status: Solved
Optimal value (cvx_optval): +7.66645
```

PROXIMAL SVM

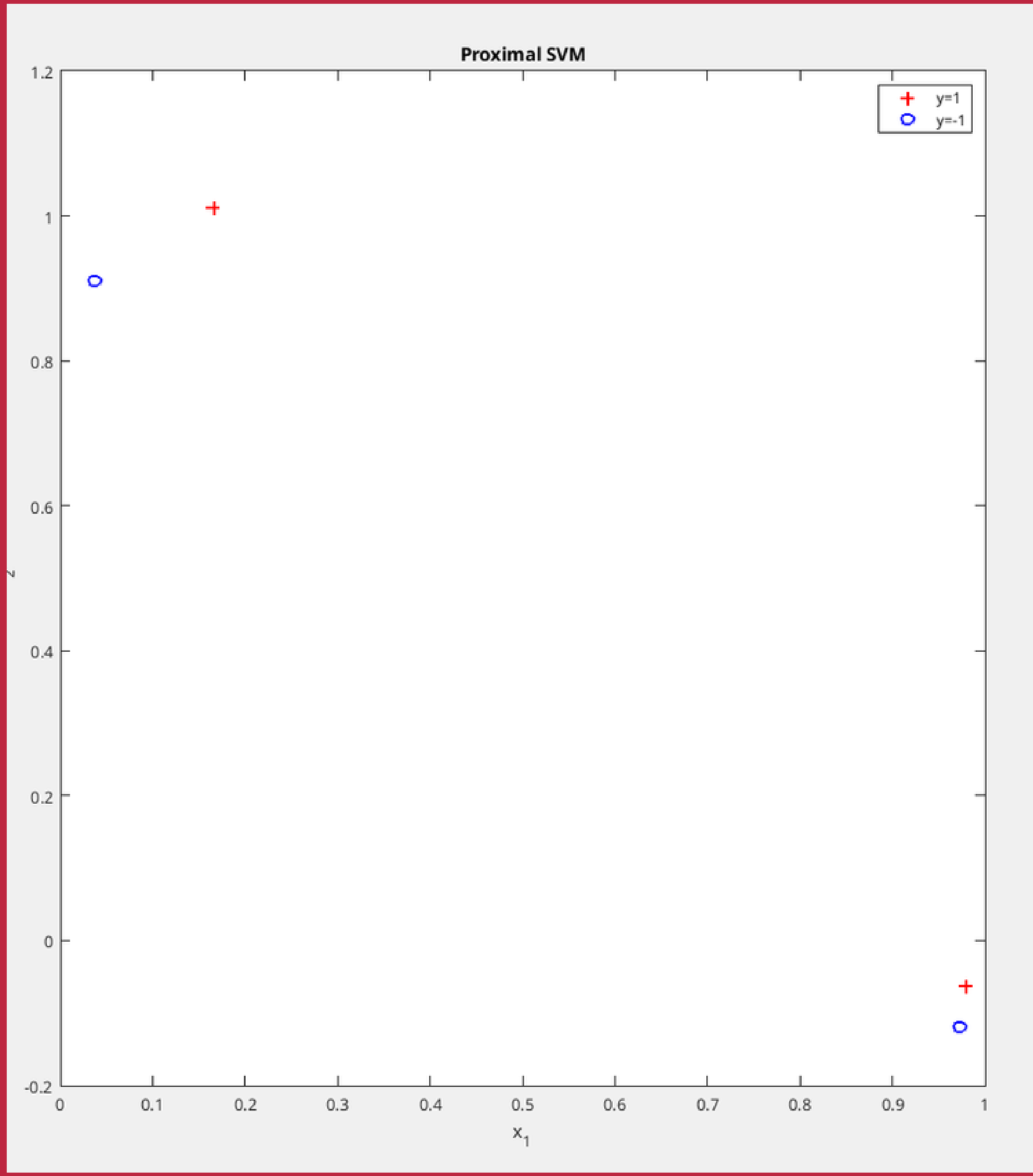
- Instead of a standard support vector machine (SVM) that classifies points by assigning them to one of two disjoint half-spaces, points are classified by assigning them to the closest of two parallel planes (in input or feature space) that are pushed apart as far as possible.

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

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

l1svm-har-margin.m | l1svm-soft-margin.m | linearl2svm.m | ProximalSVM.m

```matlab
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')

```




```
Status: Solved
Optimal value (cvx_optval): +7.66645
```

NON-LINEAR SVM USING RKS

```
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

% plot
x1 = linspace(0,5,100);
y1 = -(w(1)*x1+g)/w(2);
plot(x1,y1,'k-','LineWidth',2)
hold off
```

Command Window

```
Acc =

   52.9000
```

Implement standard LP form using ADMM

1. The first line of code is the declaration of the function linprog in matlab. The function linprog takes in 6 parameters: the objective function coefficients (c), the constraint matrix (A), the constraint vector (b), the penalty parameter (rho), the relaxation parameter (alpha) and the return value (z). It returns the value of the objective function (z and history).
2. The second line is the declaration of the variable t_start. It is used to record the time when the function is called.
3. The next three lines are the parameters for the algorithm. The first parameter QUIET is used to control the output of the algorithm. If QUIET=0, the algorithm will output the number of iterations, the objective function value and the norms of the residuals. The second parameter MAX_ITER is the maximum number of iterations. The third parameter ABSTOL and RELTOL are the absolute and relative tolerances for the stopping criteria.
4. The next two lines are the declaration of the variables m and n. The variable m is the number of rows of the constraint matrix. The variable n is the number of columns of the constraint matrix.
5. The next three lines are the declaration of the variables x, z and u. The variable x is the primal variable. The variable z is the dual variable. The variable u is the auxiliary variable.
6. The next line is the declaration of the variable history. The variable history is used to store the objective function value, the norm of the residual, the primal and the dual tolerances for each iteration.
7. The next line is the for loop. The for loop is used to iterate the algorithm. The for loop will execute MAX_ITER times or until the stopping criteria is met.

+11 q1_1.mlx q2.mlx q3.mlx q4.mlx DataGen.m rbf_tst.m Basis_pursuit.m

```
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
```

```
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);
```

running file

Implement Basis Pursuit using ADMM

```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
    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
```

We first import the data from the data file, and then we generate a random vector x with 10% of the entries non-zero. The vector blis generated by multiplying the matrix A and the vector x.

```matlab
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.

```matlab
[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.

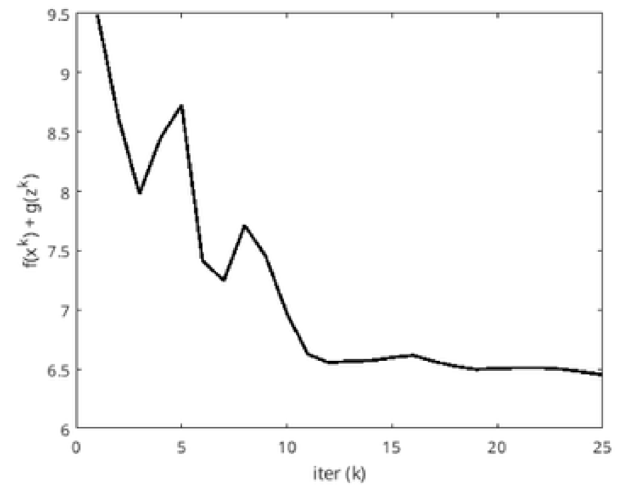
We then plot the residual norm ||r||_2 versus the number of iterations k.

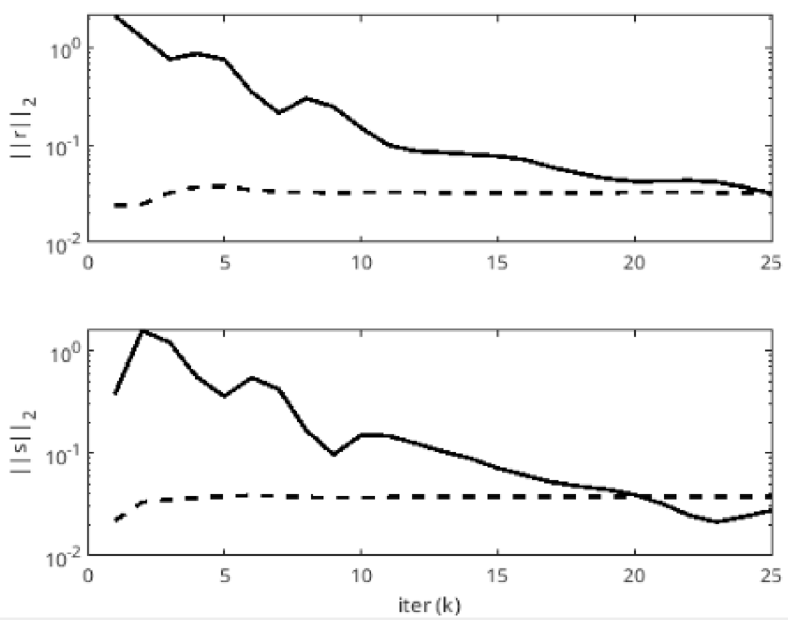
We then plot the residual norm ||s||_2 versus the number of iterations k.

```matlab
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	2.1455	0.0240	0.3756	0.0220
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.





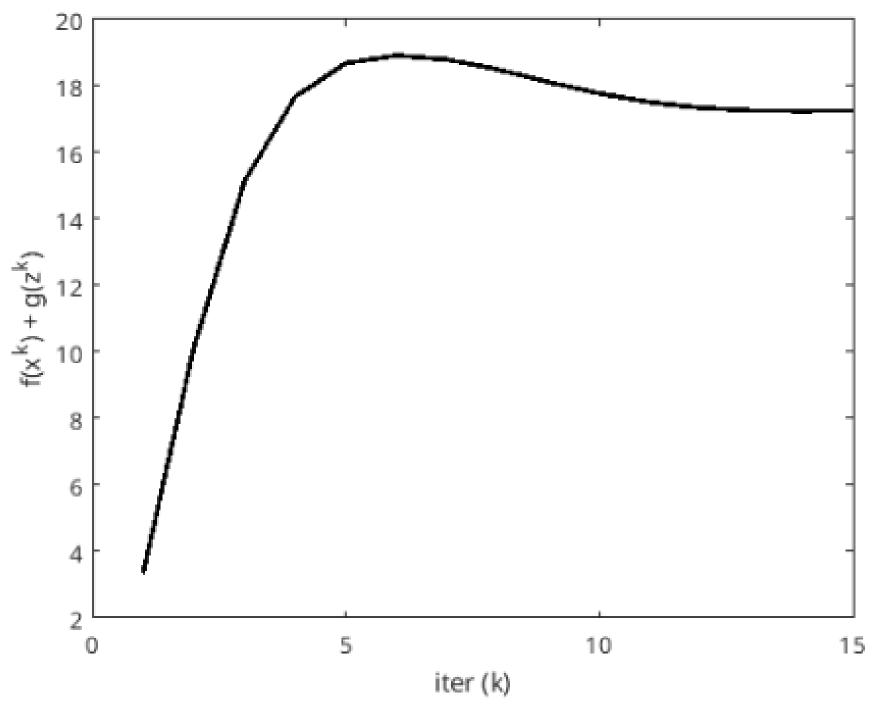
Implement LASSO using ADMM

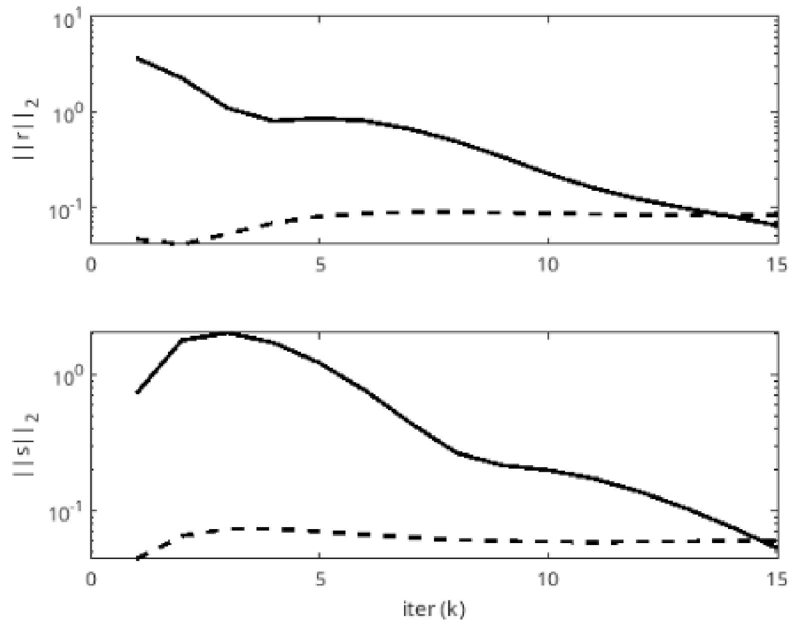
1. The first two lines set the random seed, so that the results are reproducible.
2. We create an m-by-n matrix A, where m is the number of examples and n is the number of features. The matrix is drawn from a standard normal distribution, and the columns are normalized to have unit length.
3. We create a sparse vector x0, where only a fraction p of the entries are non-zero.
4. We create the observation vector b by computing A*x0 + noise, where noise is drawn from a standard normal distribution and scaled by 0.001.
5. We compute the maximum value of the l1 norm of A'*b, which is the largest lambda for which the lasso problem is feasible.
6. We set the regularization parameter lambda to 0.1 * lambda_max.
7. We call the Lasso solver. The first argument is the matrix A, the second argument is the vector b, the third argument is the regularization parameter lambda, the fourth argument is the rho parameter, and the fifth argument is the alpha parameter.
8. We plot the objective function value versus the number of iterations.
9. We plot the primal and dual residuals versus the number of iterations.

```
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