```
function [beta,status,history] = l1_norm_ls_solver(X, y, lambda)
% @brief l1-Regularized Least Squares Solver:
%   l1_ls solves problems of the following form:
%       minimize ||y - X * beta||_2^2 + lambda * ||beta_i||_1,
%   where X and y are problem data and beta is variable (described
 below).
%
% this code is based on https://web.stanford.edu/~boyd/l1_ls/
% I am extremely grateful to the author Kwangmoo Koh
 <deneb1@stanford.edu>
%
% @author Xiaoyun Yuan,
% @date   Oct 15, 2017
%


% interior points parameters
mu = 2;        % updating parameter of t
max_iter = 100;      % maximum IPM (Newton) iteration

% backtracking line search parameters
alpha = 0.01;     % minimum fraction of decrease in the objective
nu2 = 0.5;      % stepsize decrease factor
max_iter_linesearch = 100;      % maximum backtracking line search
 iteration

% data size
n = size(X, 1);
p = size(X, 2);

% set initial value
t = 5;
reltol = 1e-3;
u = ones(p, 1);
beta = zeros(p, 1);
dobj = -Inf;
history = zeros(100, 5);

fprintf('\nSolving a problem of size (n=%d, p=%d), with lambda=%.5e
\n',...
            n, p, lambda);
fprintf('-----------------------------------------------------------------------------
\n');
fprintf('%5s %9s %15s %15s %13s %11s', ...
            'iter','gap','primobj','dualobj','reltot');
fprintf('\n');

% main loop
for ntiter = 0:max_iter
    % calcualte z
    z = y - X * beta;
    % calculate duality gap
```

```
    s = min(lambda * 1 ./ abs(2 * X' * (y - X * beta)));
    nu = 2 * s * z;

    pobj  =  z' * z + lambda * norm(beta, 1);
    dobj  =  max(-0.25 * nu' * nu + nu' * y, dobj);
    gap   =  pobj - dobj;

    % check result
    fprintf('%4d %12.2e %15.5e %15.5e %15.5e\n',...
        ntiter, gap, pobj, dobj, gap / dobj);
    history(ntiter + 1, :) = [ntiter, gap, pobj, dobj, gap / dobj];
    if (gap / dobj < reltol)
        status  = 'Solved';
        fprintf('Absolute tolerance reached.\n');
        break;
    end

    % update t
    if (s >= 0.5)
        %t = max(min(2 * p * mu / gap, mu * t), t);
        t = mu * t;
    end

    % newton step
    q1 = 1 ./ (u + beta);      q2 = 1 ./ (u - beta);
    d1 = q1 .^ 2 + q2 .^ 2;   d2 = q1 .^ 2 - q2 .^ 2;

    % calculate gradient
    gradient = [-2 * t * X' *(y - X * beta) + (q2 - q1);
        lambda * t * ones(p,1) - (q1 + q2)];

    % calculate hessian matrix
    H = [2 * t * X' * X + diag(d1), diag(d2);
        diag(d2), diag(d1)];

    % calculate dbeta and du
    dbetau = -pinv(H) * gradient;
    dbeta  = dbetau(1 : p);
    du  = dbetau(p + 1 : end);

    % line search
    phi = t * ( z' * z + lambda*sum(u)) - sum(log(u + beta)) -
sum(log(u - beta));
    s = 1.0;
    for iter = 1:max_iter_linesearch
        new_beta = beta + s * dbeta;
        new_u = u + s * du;
        if (min(new_u + new_beta) > 0) && (min(new_u - new_beta) > 0)
            newz = y - X * new_beta;
            newphi =  t * (newz' * newz + lambda * sum(new_u)) - ...
                sum(log(new_u + new_beta)) - sum(log(new_u -
new_beta));
            if (newphi - phi <= alpha * s * gradient' * dbetau)
                break;
```

```
            end
        end
        s = nu2 * s;
    end
    beta = new_beta;
    u = new_u;
end

history = history(1:(ntiter + 1), :);
```

```
Not enough input arguments.

Error in l1_norm_ls_solver (line 25)
n = size(X, 1);
```

Published with MATLAB® R2016a