APPC
TP 3 Component Wise Lasso

Thomas Robert

Prepare data

First we prepare the data : we standardize the data and create 2 datasets for learning and testing.

```
close all;
data = load('housing.data');

% make X and y matrices
[n,d] = size(data);
p = d-1;
X = data(:, 1:p);
y = data(:,d);

% standardize feature values and center target
mu_y = mean(y);
y = y - mu_y;
[X, mu, sigma] = standardizeCols(X);

% Split learn and test
[Xlearn, ylearn, Xtest, ytest] = splitdata(X, y, 0.5);
```

Proximal Gradient method

For this pratical session, we need to code a function that will compute an Elastic Net regression (parameters β) for a given λ and μ. Because fixed step can be painful to fix by hand, we will use a simple variable step (ρ) method ($\rho \times 1.15$ if cost descrease, $\rho/2$ and backtrack if cost increase).

```
function [b, Js] = proximalElastic(X, y, rho, lambda, mu)

p = size(X,2);

b = zeros(p,1);

XX = X'*X;
Xy = X'*y;

i = 1;
JOld = Inf;
J = 1/2*norm(X*b-y,2)^2 + lambda * norm(b, 1) + mu / 2 * norm(b)^2;

if (nargout > 1)
    Js = J;
end

while (JOld - J > 1e-7 && i < 1e6)

    % try compute beta
    bhatNew = b - rho * (XX*b - Xy + mu*b);
    bNew = sign(bhatNew) .* max(0, abs(bhatNew) - rho * lambda);

    % cost
    JNew = 1/2*norm(X*bNew-y,2)^2 + lambda * norm(bNew, 1) + mu / 2 * norm(bNew)^2;

    % save if cost decrease
    if (JNew < J)
        b = bNew;
```

```
        bhat = bhatNew;
        JOld = J;
        J = JNew;
        rho = rho * 1.15;

        if (nargout > 1)
            Js = [Js J];
        end
    % forget if cost increase
    else
        rho = rho / 2;
    end

    i = i + 1;
end
```

Lasso

Let's compute the Lasso using Proximal Gradient (PG) method. For Lasso, we set $\mu = 0$ and $\lambda = 10$.

We compare Proxial Gradient method result with Component-Wise Lasso, and can see that they give the same result (difference of the norm arround 10^{-4}).

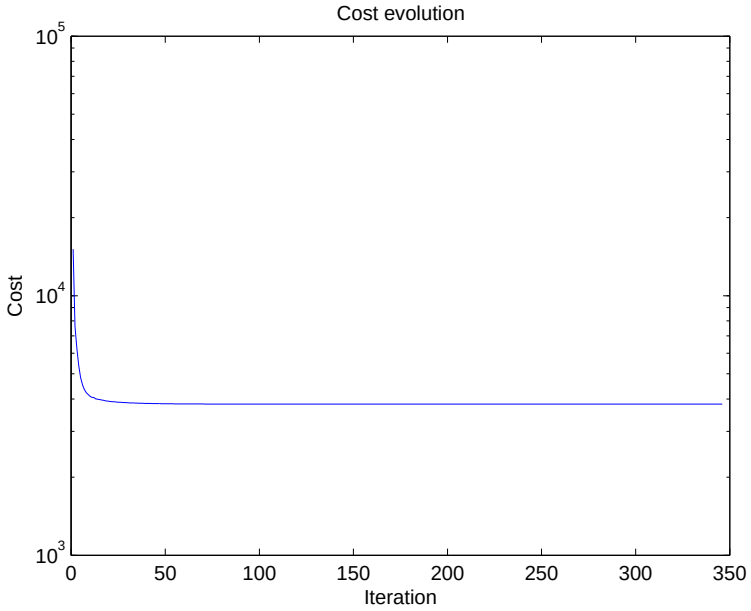
We can check that indeed the cost decrease when using PG method, which is forced by step variation that will decrease the step until it find a step that will make the cost decrease.

We can also see that PG method is much faster (100 times faster) than CW method, which were already quite quick. Indeed, there is not a lot to compute at each interation.

```
rho = 0.0005;
lambda = 10;
mu = 0;

% Proximal
tic
[bProx, Js] = proximalElastic(Xlearn, ylearn, rho, lambda, mu);
disp(['Proximal time : ' int2str(toc*1000) ' ms']);
% CW Lasso to compare
tic
bCW = CWLasso(Xlearn, ylearn, lambda, zeros(p,1));
disp(['CW Lasso time : ' int2str(toc*1000) ' ms']);

% Plot cost evolution
semilogy(Js);
title('Cost evolution');
xlabel('Iteration');
ylabel('Cost');

bs = [bProx bCW]
diffBtwB = norm(bProx - bCW)
```

```
Proximal time : 10 ms
CW Lasso time : 440 ms

bs =

   -0.7927   -0.7928
    0.7723    0.7724
    0.0048    0.0051
    0.8299    0.8299
   -2.8594   -2.8594
    2.6779    2.6778
    0.2960    0.2961
   -3.4367   -3.4367
    2.4069    2.4076
   -1.4339   -1.4346
```

```
   -1.9429   -1.9429
    0.8974    0.8974
   -4.1915   -4.1915


diffBtwB =

    0.0010
```



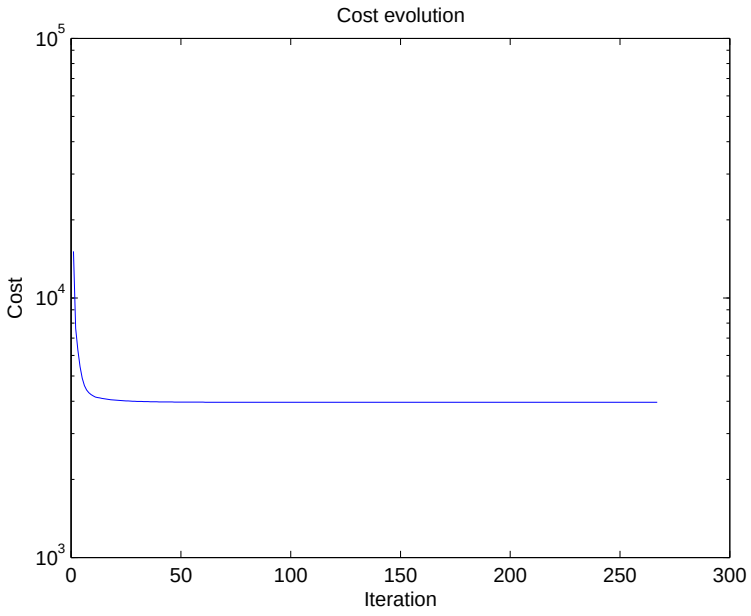
Elastic Net

We now test the function with $\mu = 5$ and $\lambda = 10$ to Elastic Net regression method.

We compare the results given by PG method with results given by CVX and still find the same results.

```
rho = 0.0005;
lambda = 10;
mu = 5;

% Proximal
tic
[bProx, Js] = proximalElastic(Xlearn, ylearn, rho, lambda, mu);
disp(['Proximal time : ' int2str(toc*1000) ' ms']);
% CVX
tic
cvx_quiet(true);
cvx_begin
    variable b(p)
    minimize(1/2 * sum_square(Xlearn*b - ylearn) + lambda * norm(b, 1) + mu / 2 * sum_square(b))
cvx_end
disp(['CVX time : ' int2str(toc*1000) ' ms']);
bCVX = b;

% Plot cost evolution
semilogy(Js);
title('Cost evolution');
xlabel('Iteration');
ylabel('Cost');

bs = [bProx bCVX]
diffBtwB = norm(bProx - bCVX)
```

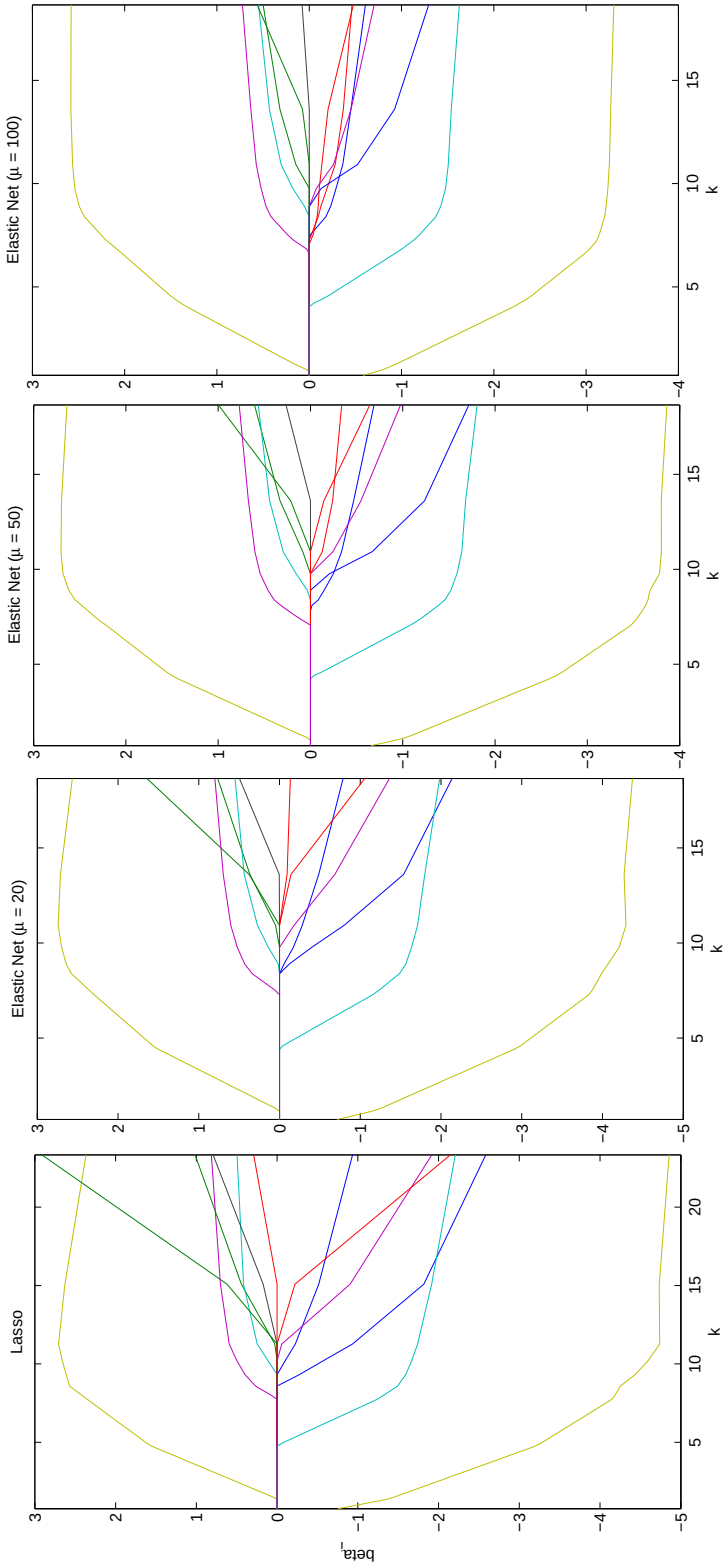
```
Proximal time : 6 ms
CVX time : 324 ms

bs =

   -0.7318   -0.7319
    0.6979    0.6979
         0   -0.0000
    0.8190    0.8190
   -2.5992   -2.5992
    2.7476    2.7475
    0.2310    0.2310
   -3.1857   -3.1857
    2.0239    2.0242
   -1.1828   -1.1832
   -1.8943   -1.8942
    0.8982    0.8982
   -4.0889   -4.0889


diffBtwB =

   5.1976e-04
```



Regularization path

We now compute the regularization path for both Lasso and Elastic Net to compare how the evolution of beta components differ.

We can see that the L2 penalization of the Elastic Net prevent some variables from growing too much for important values of k, when those values can become really important in Lasso.




```
mu1 = 20;
mu2 = 50;
mu3 = 100;

Lvals = [0:50:2200];
errors = zeros(length(Lvals), 1);
bsLasso = zeros(length(Lvals), p);
bsElast1 = zeros(length(Lvals), p);
```

```
bsElast2 = zeros(length(Lvals), p);
bsElast3 = zeros(length(Lvals), p);

for i = 1:length(Lvals)
    L = Lvals(i);
    bsLasso(i,:) = proximalElastic(Xlearn, ylearn, rho, L, 0);
    bsElast1(i,:) = proximalElastic(Xlearn, ylearn, rho, L, mu1);
    bsElast2(i,:) = proximalElastic(Xlearn, ylearn, rho, L, mu2);
    bsElast3(i,:) = proximalElastic(Xlearn, ylearn, rho, L, mu3);
end

ksLasso = sum(abs(bsLasso), 2);
ksElast1 = sum(abs(bsElast1), 2);
ksElast2 = sum(abs(bsElast1), 2);
ksElast3 = sum(abs(bsElast1), 2);

figure;
subplot(1,4,1);
plot(ksLasso, bsLasso);
xlim([min(ksLasso) max(ksLasso)]);
title('Lasso');
ylabel('beta_i');
xlabel('k');

subplot(1,4,2);
plot(ksElast1, bsElast1);
xlim([min(ksElast1) max(ksElast1)]);
title(['Elastic Net (\mu = ' int2str(mu1) ')'])
ylabel('beta_i');
xlabel('k');

subplot(1,4,3);
plot(ksElast2, bsElast2);
xlim([min(ksElast2) max(ksElast2)]);
title(['Elastic Net (\mu = ' int2str(mu2) ')'])
ylabel('beta_i');
xlabel('k');

subplot(1,4,4);
plot(ksElast3, bsElast3);
xlim([min(ksElast3) max(ksElast3)]);
title(['Elastic Net (\mu = ' int2str(mu3) ')'])
ylabel('beta_i');
xlabel('k');
```