# LABORATORY EXPERIENCE #1
# (Linear Regression)

## Introduction

The first laboratory experience requests to perform **regression** algorithms in order to predict the grade of the **Parkinson disease** in a list of patients, starting from a dataset of measurements taken by the medical doctor.
The dataset that has been subject of study is public and can be downloaded at the following link https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring.
Each entry of the dataset is described by the following parameters, evaluated for 42 patients during a 6 months monitoring period:

- **subject#** - Integer that uniquely identifies each subject
- **age** - Subject age
- **sex** - Subject gender '0' - male, '1' - female
- **test_time** - Time since recruitment into the trial. The integer part is the number of days since recruitment.
- **motor_UPDRS** - Clinician's motor UPDRS score, linearly interpolated
- **total_UPDRS** - Clinician's total UPDRS score, linearly interpolated
- **Jitter (%)** - Several measures of variation in fundamental frequency
- **Jitter (Abs)**
- **Jitter: RAP**
- **Jitter: PPQ5**
- **Jitter: DDP**
- **Shimmer** - Several measures of variation in amplitude
- **Shimmer(dB)**
- **Shimmer: APQ3**
- **Shimmer: APQ5**
- **Shimmer: APQ11**
- **Shimmer: DDA**
- **NHR**
- **HNR -** Two measures of ratio of noise to tonal components in the voice
- **RPDE -** A nonlinear dynamical complexity measure
- **DFA** - Signal fractal scaling exponent
- **PPE** - A nonlinear measure of fundamental frequency variation

The parameter that indicates the grade of the Parkinson disease is the UPDRS, that stands for Unified Parkinson’s Disease Rating Scale. In our case we will take into consideration the **motor_UPDRS** parameter, that will be our regressand.
Regression will be performed through **Minimum Square Error (MSE) algorithm**, **gradient algorithm** and **steepest descent algorithm.**

## Data preparation

The first phase of the algorithm is dedicated to “clean” the input dataset, in order to ease the process of regression. In particular, for each patient, we aggregate the measurements taken

## Table of Contents

# Data preparation

```
clear all
close all

load('updrs.mat')

A=parkinsonsupdrs;

num_patients=max(A(:,1));

new_matrix=ones(1,22); % we define the matrix in which we insert
 grouped values

for ind=1:num_patients
    sub_rows=find(A(:,1)==ind); % indexes of the matrix corresponding
 to the 'ind' patient
    pat=A(sub_rows, :); % sub_matrix of 'ind' patient
    time=sort(abs(fix(pat(:,4)))); % we define an array of the days in
 which measurements were taken for patient 'ind'
    % we had to take the integer part of that values and make the
 absolute
    % value because there were some negative time values. Then I
 ordered
    % them by increasing order
    time=unique(time); % we delete duplicates in the array of days
    for i=1:length(time) %we iterate within the days
        day=time(i);
        time_rows=find(A(:,1)==ind & abs(fix(A(:,4)))==day); % we find
 the rows with all the measurements taken on patient 'ind' in the day
 'day'
        pat_time=A(time_rows, :); % we build a matrix of the
 measurements of patient 'ind' on day 'day'
        pat_time(:,4)=abs(fix(pat_time(:,4))); % we adjust the time
 column in order to have integer day indexes
        mean_row=mean(pat_time,1);
        new_matrix=[new_matrix;mean_row];
    end
end

pazienti=new_matrix(2:end,:);

save('parkinson.mat','new_matrix');
```

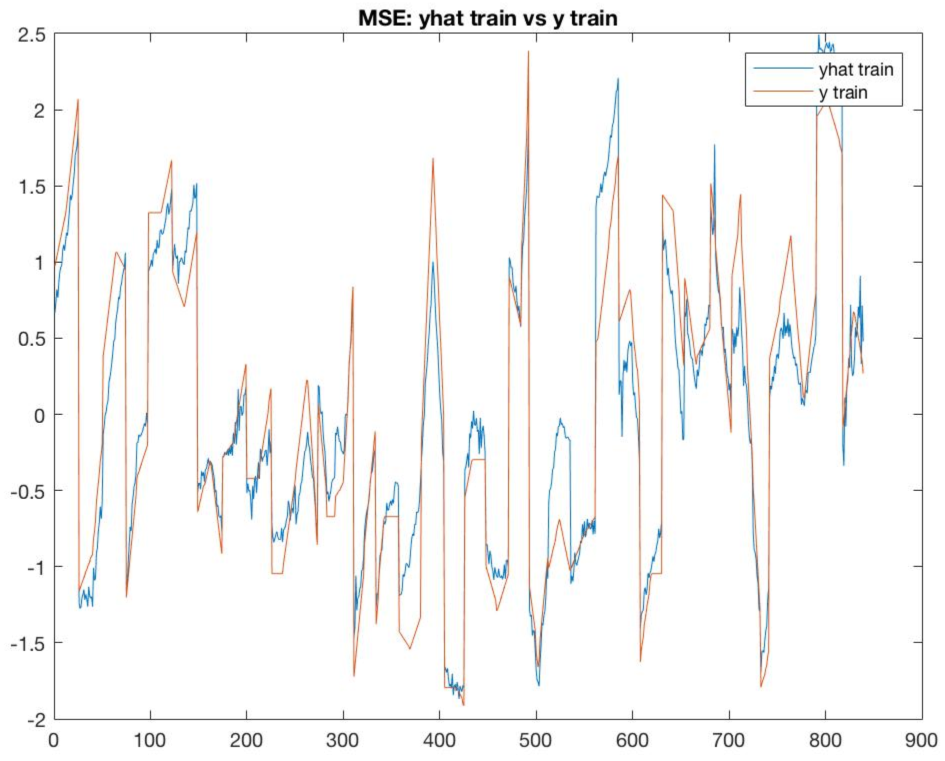
```
%-------  we build data_train and data_test matrices-----

% data_train=ones(1,22);
% data_test=ones(1,22);
% for i=1:length(new_matrix(:,1))
%     if new_matrix(i,1)<=36
%         data_train=[data_train;new_matrix(i,:)];
%     else
%         data_test=[data_test;new_matrix(i,:)];
%     end
% end
%
% data_train=data_train(2:end,:);
% data_test=data_test(2:end,:);

data_train = pazienti(pazienti(:,1)<37,:);
data_test = pazienti(pazienti(:,1)>36,:);


% ---- applying normalization to data_train matrix

m_data_train=mean(data_train,1);
v_data_train=var(data_train,1);
s_v_data_train=sqrt(v_data_train);

% data_train_norm=ones(length(data_train(:,1)),22);
% for i=1:length(data_train(:,1))
%     data_train_norm(i,1:4)=data_train(i,1:4);
%     for f=5:22
%         data_train_norm(i,f)=(data_train(i,f)-m_data_train(f))/
s_v_data_train(f);
%     end
% end

o = ones(size(data_train,1),1);
data_train_norm = data_train;
data_train_norm(:,5:end) = (data_train(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_train_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_train_norm = var(data_train_norm,1);


% ---- applying normalization to data_test matrix

% data_test_norm=ones(length(data_test(:,1)),22);
% for i=1:length(data_test(:,1))
%     data_test_norm(i,1:4)=data_test(i,1:4);
%     for f=5:22
%         data_test_norm(i,f)=(data_test(i,f)-m_data_train(f))/
s_v_data_train(f);
```

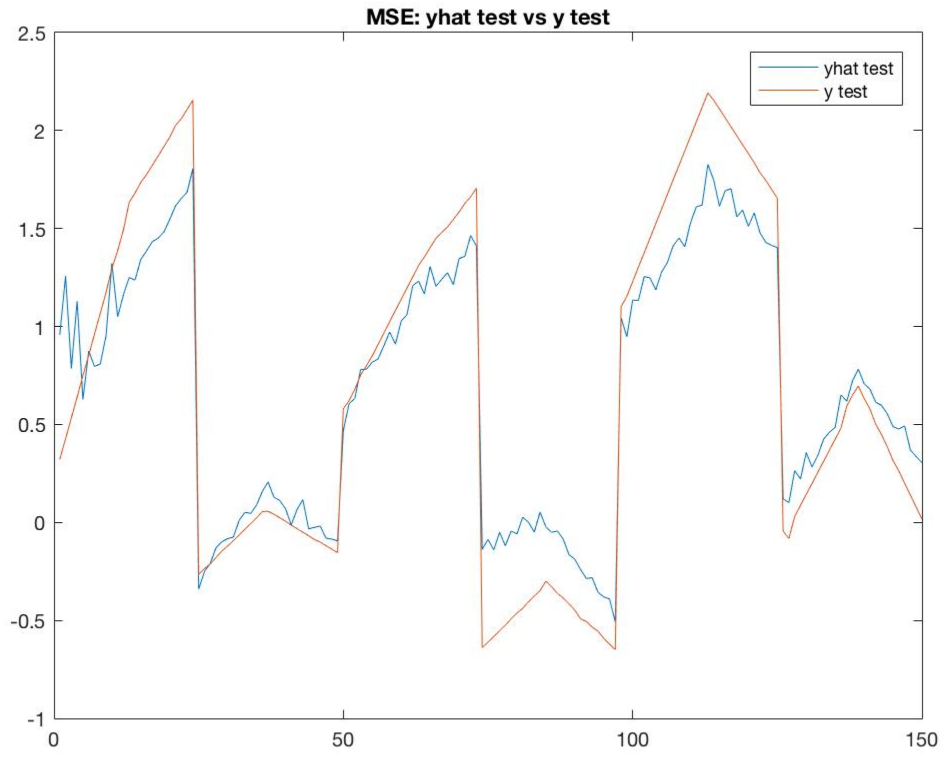
```
%     end
% end

o = ones(size(data_test,1),1);
data_test_norm = data_test;
data_test_norm(:,5:end) = (data_test(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_test_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_test_norm = var(data_train_norm,1);

% DEFINITION OF REGRESSOR AND REGRESSAND

F0=5; % we have to try with feature 7 and feature 5

y_train=data_train_norm(:,F0);
X_train=data_train_norm;
X_train(:,F0)=[];

y_test=data_test_norm(:,F0);
X_test=data_test_norm;
X_test(:,F0)=[];
```

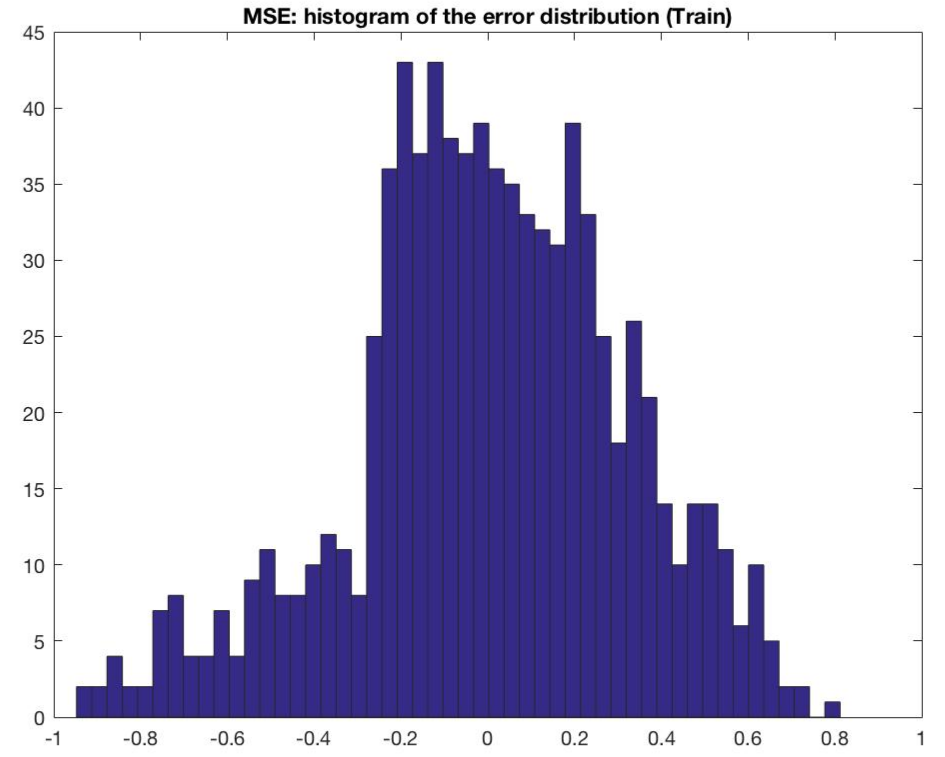
## Minimum Square Error (MSE)

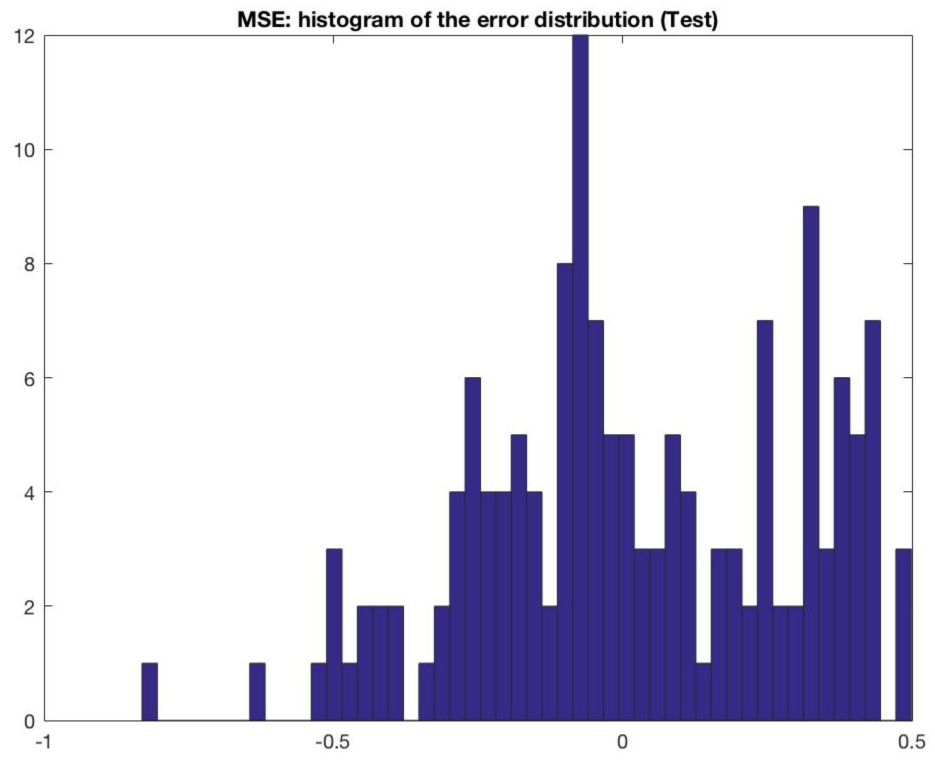
```
X_T=X_train(:,5:end).';

a_hat=inv(X_T*X_train(:,5:end))*X_T*y_train;

yhat_train=X_train(:,5:end)*a_hat;
yhat_test=X_test(:,5:end)*a_hat;

error_test=y_test-yhat_test;
error_train=y_train-yhat_train;

figure
plot(yhat_train)
hold on
plot(y_train)
hold off
title('MSE: yhat train vs y train')
legend('yhat train','y train')

figure
plot(yhat_test)
hold on
plot(y_test)
hold off
title('MSE: yhat test vs y test')
legend('yhat test','y test')

figure
```

```
hist(error_train,50) % for the feature 5 the error is very variable,
 that means that feature 5 is less correlated to others
% for the feature 7 the error is distributed according something that
 is
% like a gaussian
title('MSE: histogram of the error distribution (Train)')

figure
hist(error_test,50) % for the feature 5 the error is very variable,
 that means that feature 5 is less correlated to others
% for the feature 7 the error is distributed according something that
 is
% like a gaussian
title('MSE: histogram of the error distribution (Test)')
```




MSE: yhat test vs y test
yhat test
y test
2.5
2
1.5
1
0.5
0
-0.5
-1
0
50
100
150

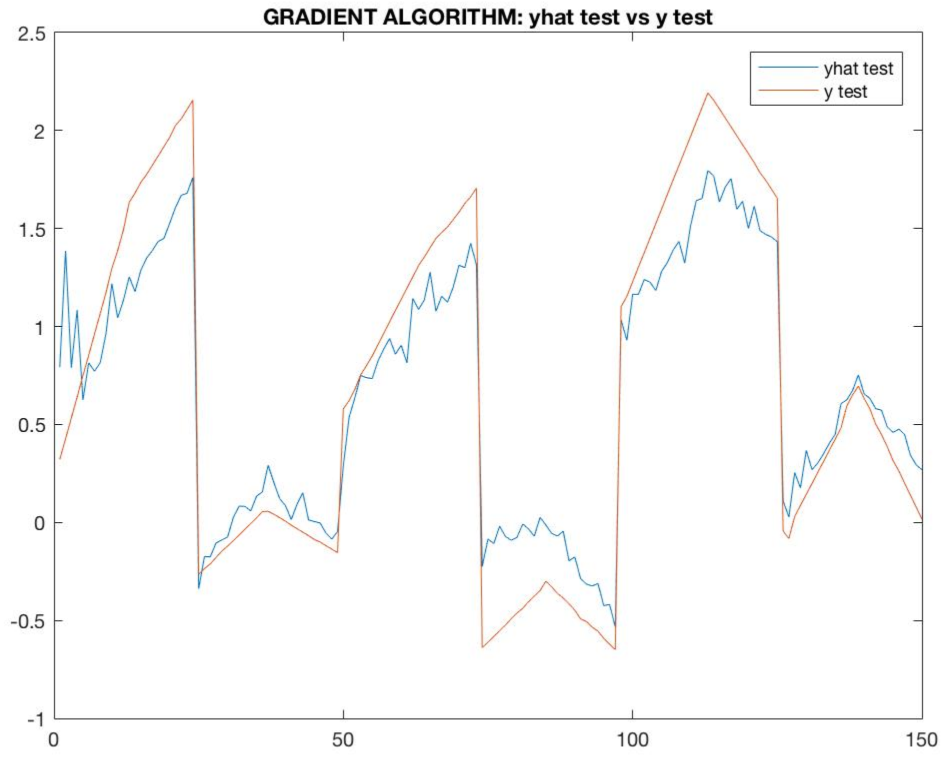

MSE: histogram of the error distribution (Train)
45
40
35
30
25
20
15
10
5
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1



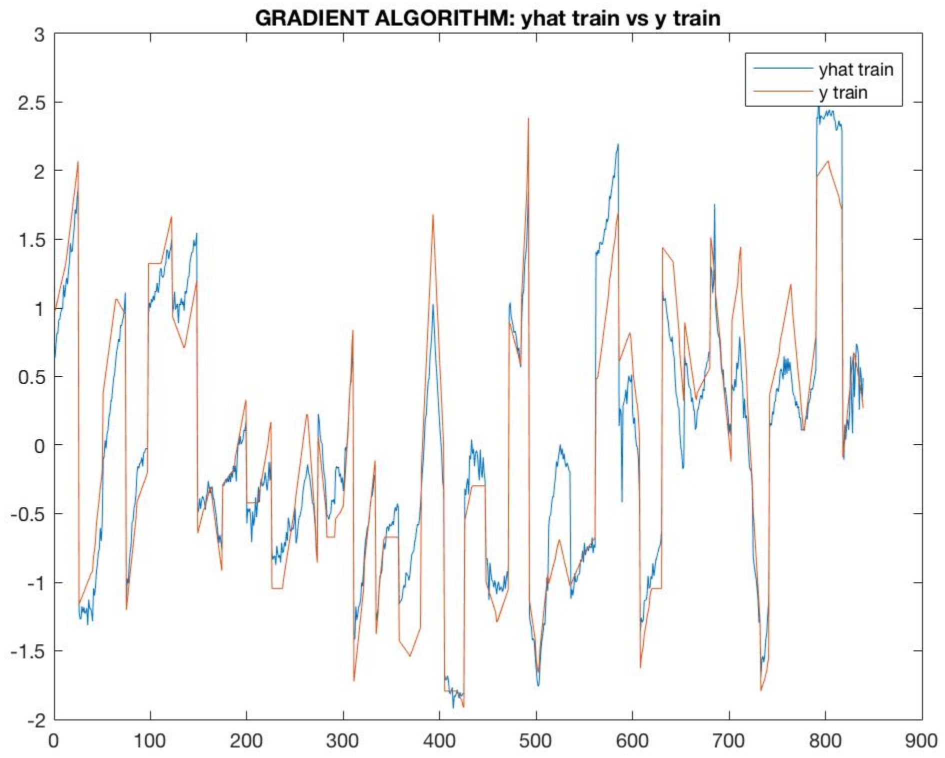
## Gradient Algorithm

```
rng('default') % allows to have the same random vector each time we
 run the script
a_i = rand(17,1); % generates a 21x1 matrix (a vector) of uniformly
 distributed random numbers
gamma=0.000001; % learning coefficient (usually 0.01/0.001/0.0001)
epsilon=0.00001; % stopping rule: a_hat(i+1)-a_hat(i)<epsilon
a_old=zeros(17,1);
n=0;
a_story=a_i';

while (norm(a_i - a_old) > epsilon)
    grad= - 2 * X_T * y_train + 2 * X_T * X_train(:,5:end) * a_i;
    a_story=[a_story;a_i'];
    a_old=a_i;
    a_i = a_i - (gamma * grad);
    n=n+1;
end

yhat_ga_test=X_test(:,5:end)*a_i;
error_ga_test=y_test-yhat_ga_test;

yhat_ga_train=X_train(:,5:end)*a_i;
error_ga_train=y_train-yhat_ga_train;
```

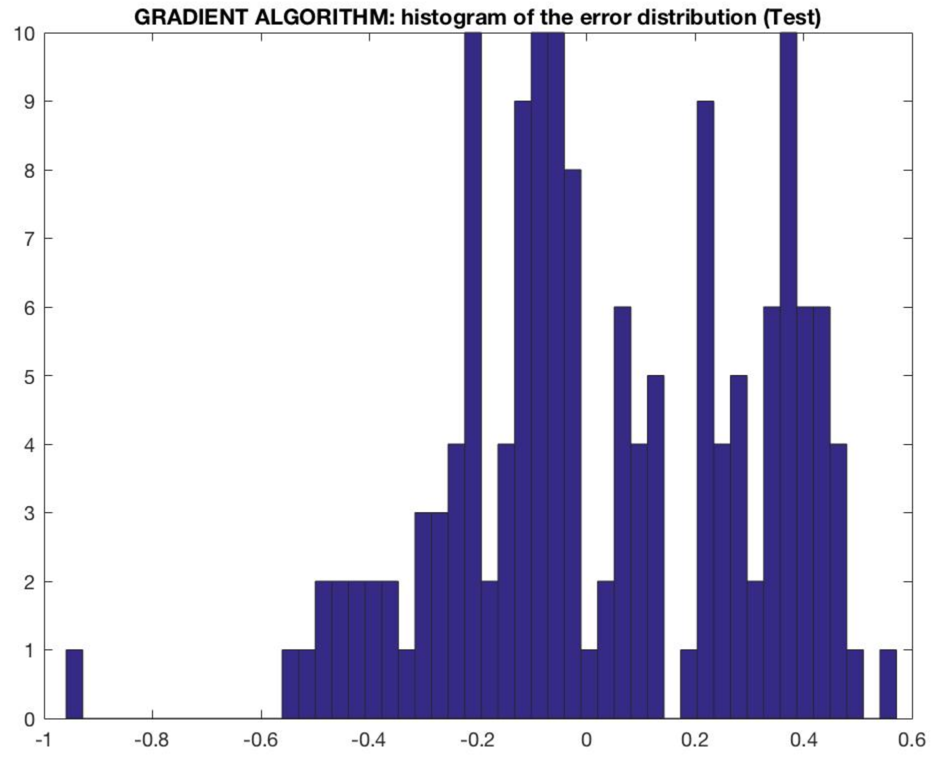
```
figure
plot(yhat_ga_test)
hold on
plot(y_test)
hold off
title('GRADIENT ALGORITHM: yhat test vs y test')
legend('yhat test','y test')

figure
plot(yhat_ga_train)
hold on
plot(y_train)
hold off
title('GRADIENT ALGORITHM: yhat train vs y train')
legend('yhat train','y train')

figure
hist(error_ga_test,50)
title('GRADIENT ALGORITHM: histogram of the error distribution
 (Test)')

figure
hist(error_ga_train,50)
title('GRADIENT ALGORITHM: histogram of the error distribution
 (Train)')
```

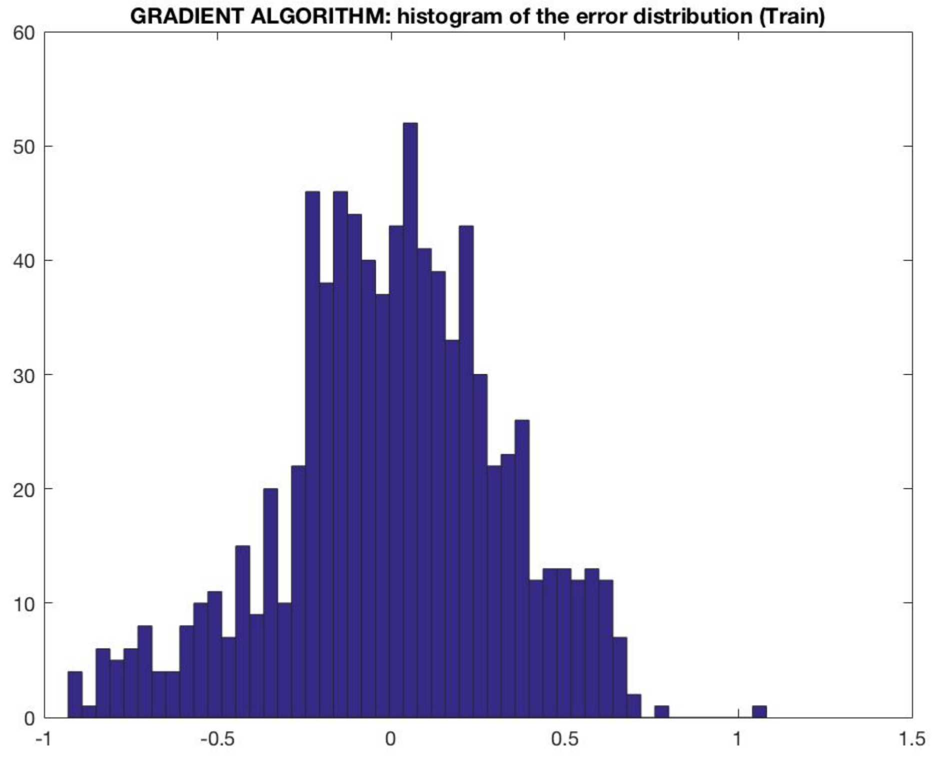


GRADIENT ALGORITHM: yhat test vs y test
yhat test
y test



GRADIENT ALGORITHM: yhat train vs y train
yhat train
y train

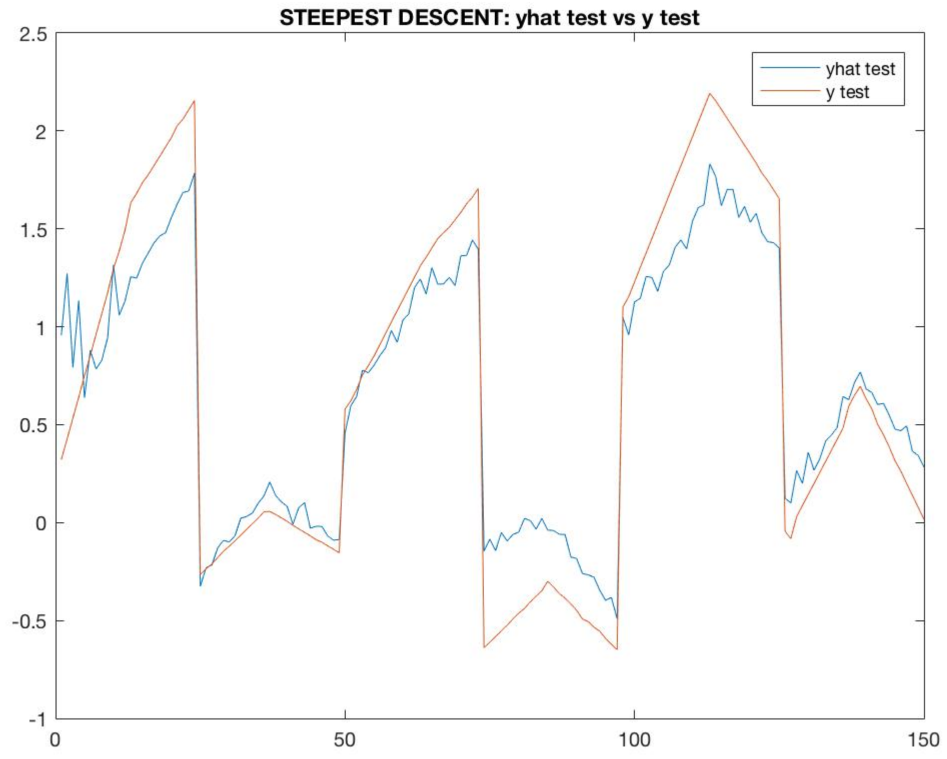

GRADIENT ALGORITHM: histogram of the error distribution (Test)
10
9
8
7
6
5
4
3
2
1
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

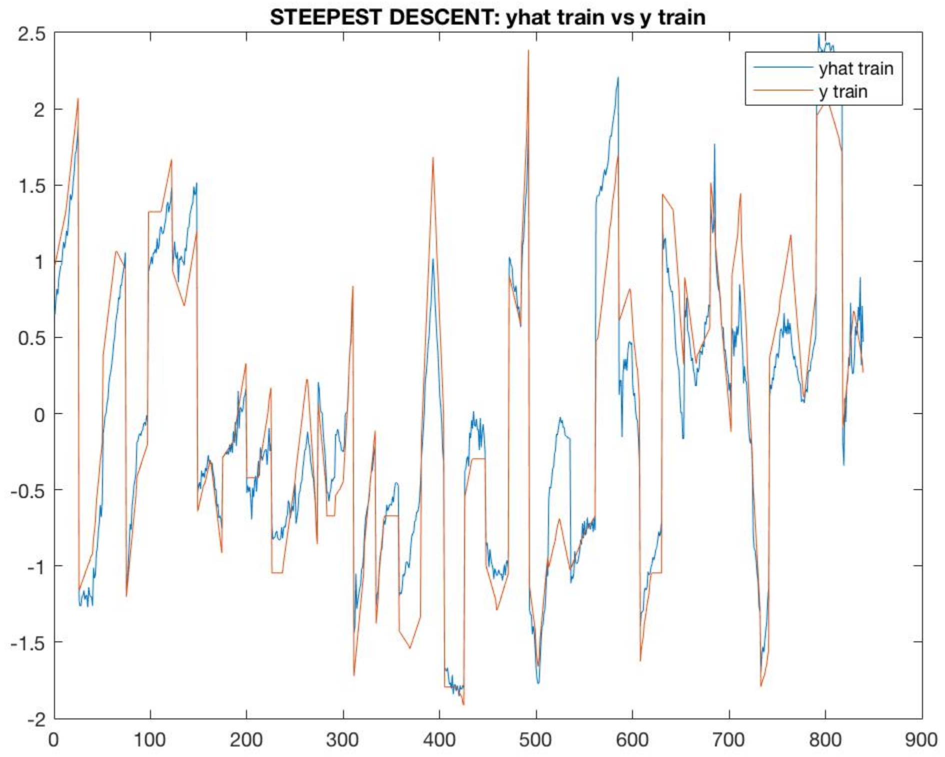

GRADIENT ALGORITHM: histogram of the error distribution (Train)
60
50
40
30
20
10
0
-1
-0.5
0
0.5
1
1.5

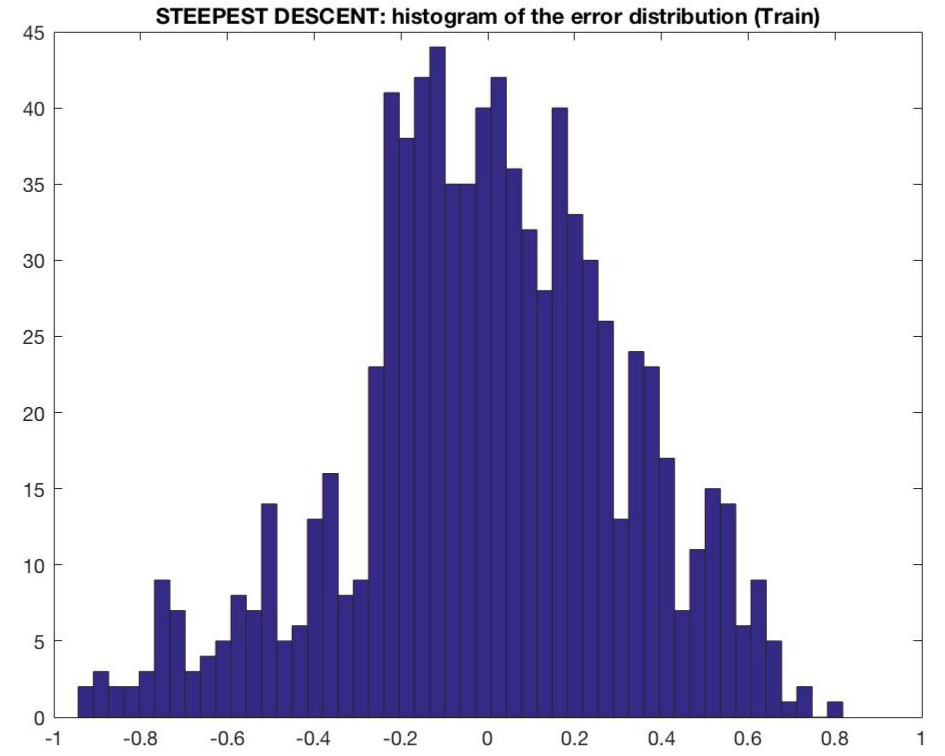
## Steepest Descent Algorithm

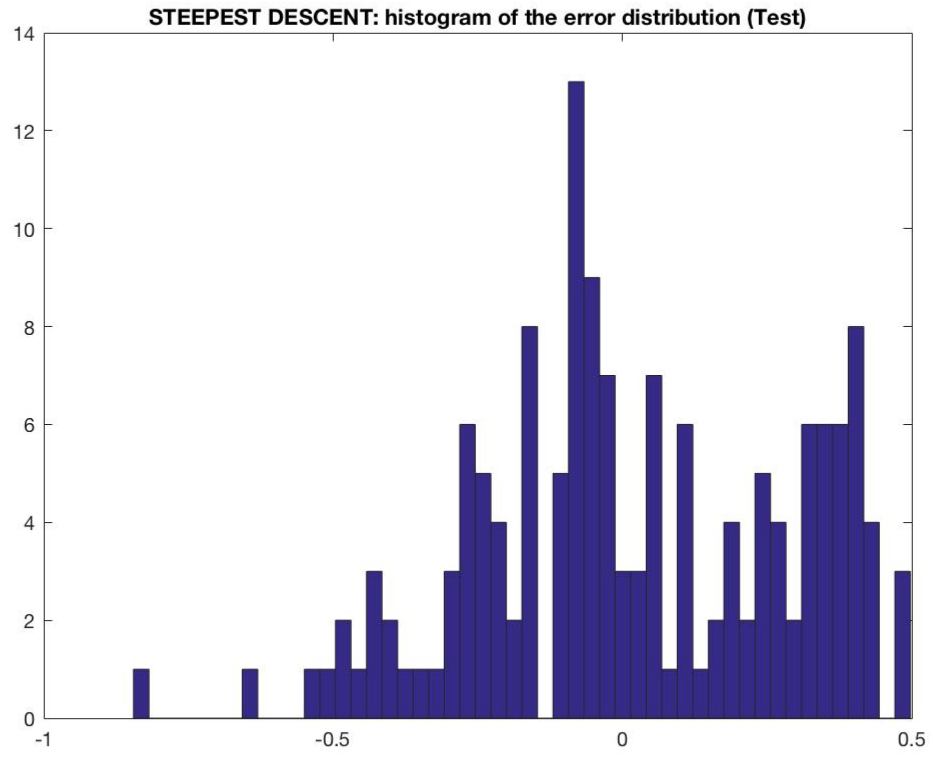
```
rng('default') % allows to have the same random vector each time we
 run the script
a_i_sd = rand(17,1); % generates a 21x1 matrix (a vector) of uniformly
 distributed random numbers
gamma=0.000001; % learning coefficient (usually 0.01/0.001/0.0001)
epsilon=0.00001; % stopping rule: a_hat(i+1)-a_hat(i)<epsilon
a_old_sd=zeros(17,1);
n=0;
a_story=a_i';

while (norm(a_i_sd - a_old_sd) > epsilon)
    grad_sd= - 2 * X_T * y_train + 2 * X_T * X_train(:,5:end) *
 a_i_sd;
    hessian = 4 * X_T * X_train(:,5:end);
    a_old_sd = a_i_sd;
    a_i_sd = a_i_sd - ((norm(grad_sd)^2 * grad_sd)/(grad_sd.' *
 hessian * grad_sd));
end

yhat_sd_train=X_train(:,5:end)*a_i_sd;
error_sd_train=y_train-yhat_sd_train;

yhat_sd_test=X_test(:,5:end)*a_i_sd;
error_sd_test=y_test-yhat_sd_test;

figure
plot(yhat_sd_test)
hold on
plot(y_test)
hold off
title('STEEPEST DESCENT: yhat test vs y test')
legend('yhat test','y test')


figure
plot(yhat_sd_train)
hold on
plot(y_train)
hold off
title('STEEPEST DESCENT: yhat train vs y train')
legend('yhat train','y train')

figure
hist(error_sd_train,50)
title('STEEPEST DESCENT: histogram of the error distribution (Train)')

figure
hist(error_sd_test,50)
title('STEEPEST DESCENT: histogram of the error distribution (Test)')
```

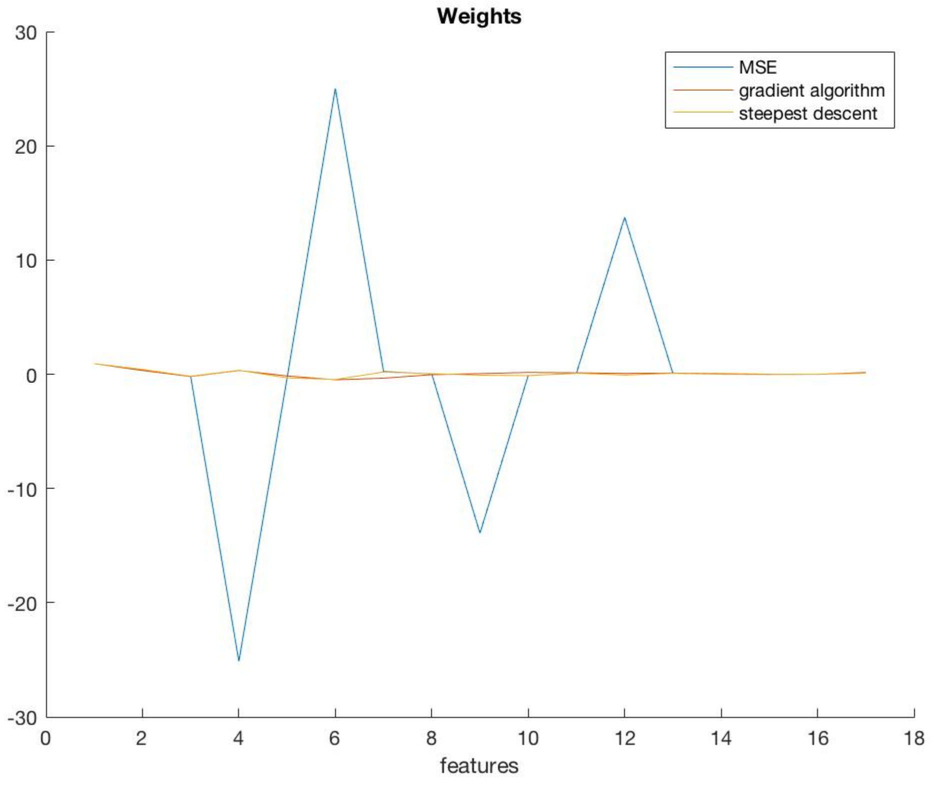
```
figure
hold on
plot(a_hat)
plot(a_i)
plot(a_i_sd)
hold off
title('Weights')
xlabel('features')
legend('MSE','gradient algorithm','steepest descent')
```




STEEPEST DESCENT: yhat train vs y train
yhat train
y train
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900


STEEPEST DESCENT: histogram of the error distribution (Train)
45
40
35
30
25
20
15
10
5
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


STEEPEST DESCENT: histogram of the error distribution (Test)
0
2
4
6
8
10
12
14
-1
-0.5
0
0.5


Weights
MSE
gradient algorithm
steepest descent
30
20
10
0
-10
-20
-30
0
2
4
6
8
10
12
14
16
18
features

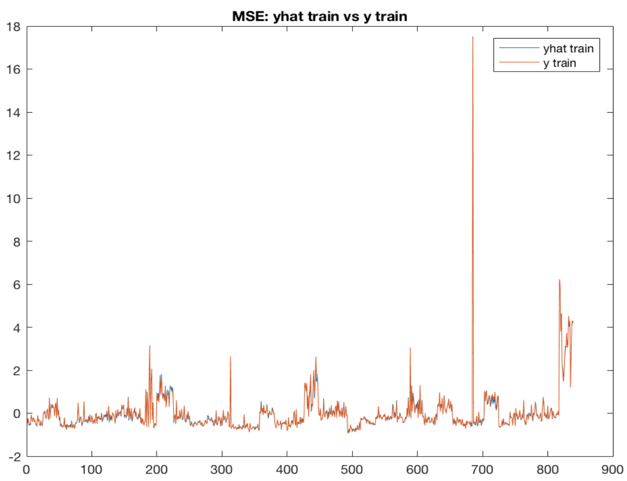
*Published with MATLAB® R2016a*

within each day evaluating the mean of the entries corresponding to that day. As some test_time values were wrongly registered as negative numbers, the absolute value of that number was considered. Then, data is normalized and divide in two matrices: the first, containing the measurement of the first 36 patients, will be used as regressor in the training phase of the algorithms, while the second, corresponding to the patients from 37 to 42 will be used in the test phase.

## Algorithms implementation and results

The idea of linear regression starts from the hypothesis that we can obtain our regressand vector **y** is a linear combination of our regressor matrix **X**, properly weighted (coefficients vector **a**), plus a measurement error. The scope of the linear regression algorithm is to find the weights that better describe this relation between regressor and regressand.
**MSE algorithm** simply finds those weights as the values that minimizes the square error $e(a) = \|y - Xa\|^2$ finding the values of a for which the gradient of e(a) is zero. This method allows to evaluate the unknown through a closed form expression, but requires the inversion of a matrix, that might be too complex in some cases. So, in those cases might be useful to perform iterative algorithms like the **gradient algorithm**, in which, starting from an initial guess of the weights, computes step by step the gradient of the error function and tries to "follow" the direction towards the minimum until a stopping function is reached. This algorithm requires the definition of a learning coefficient and of the stopping rule. **Steepest descent algorithm** allows to reach a faster convergence finding at each step the optimum value of the learning coefficient, instead of defining it a priori.

We are able to obtain in a similar way good performance for each of the three algorithms, especially if we try to predict feature 7 (**Jitter (%)**). Plotting the predicted values against the true ones we can see that all the algorithms are working quite well, both in the training phase and learning phase. Actually, plotting the histograms of the error distribution is it possible to observe a greater dispersion of the error in the test phase, as we could expect.
The confrontation of the weights shows that the iterative algorithms find a different solution to the MSE algorithm, as well as the overall performance of the regression remains very good.
Performing regression with feature 5 (motor_UPDRS) we can see a slight decrease of the performances, expecially in the test phase. This could be due to the fact that Jitter is more related to the other features rather then the motor_UPDRS, as the first one is, in line the other features, related to speech impairments, while the second one is dealing with motor impairments, and It is evaluated at discretion of the medical doctor.



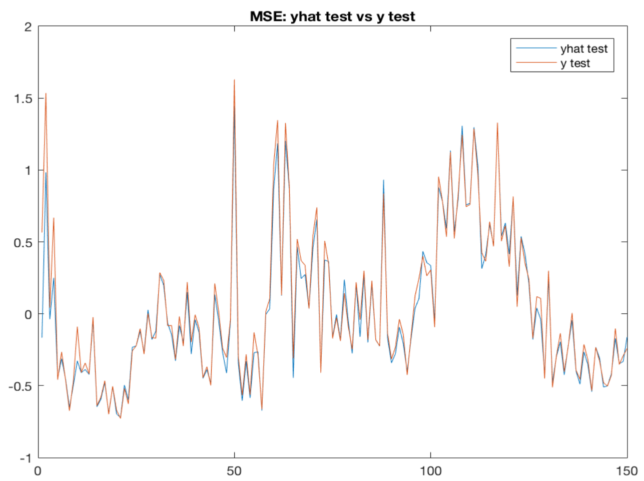


*Figure 1: Regression results vs. real values in train and test phase - MSE feature 7*

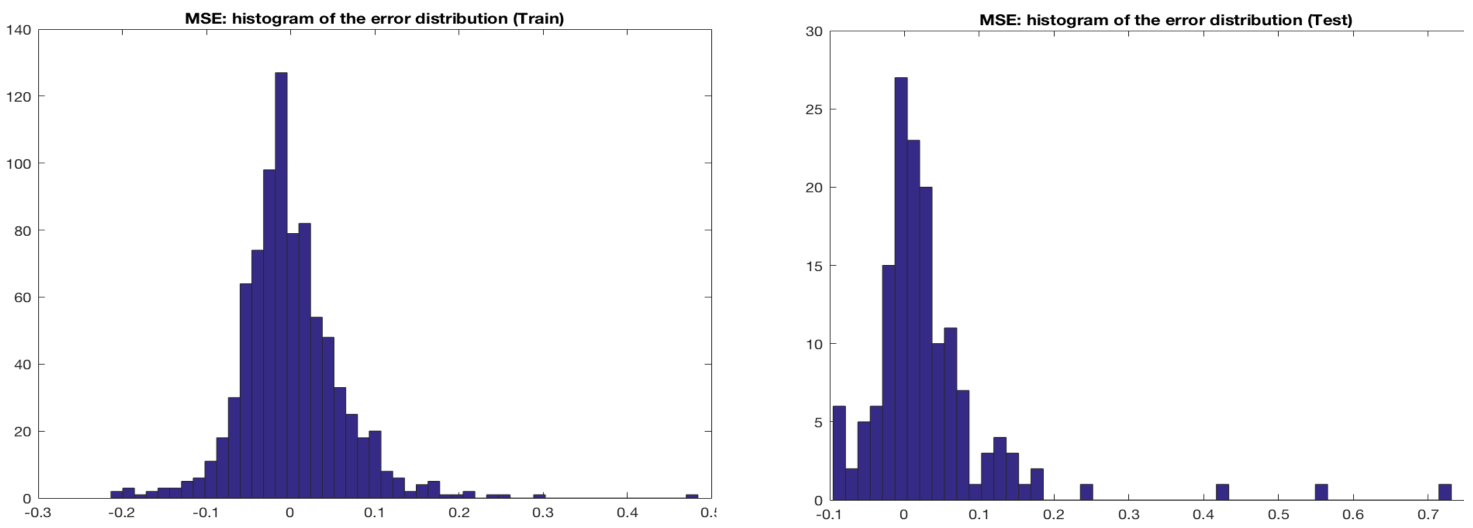


*Figure 2: Comparison of error distribution in train and test phase - MSE feature 7*

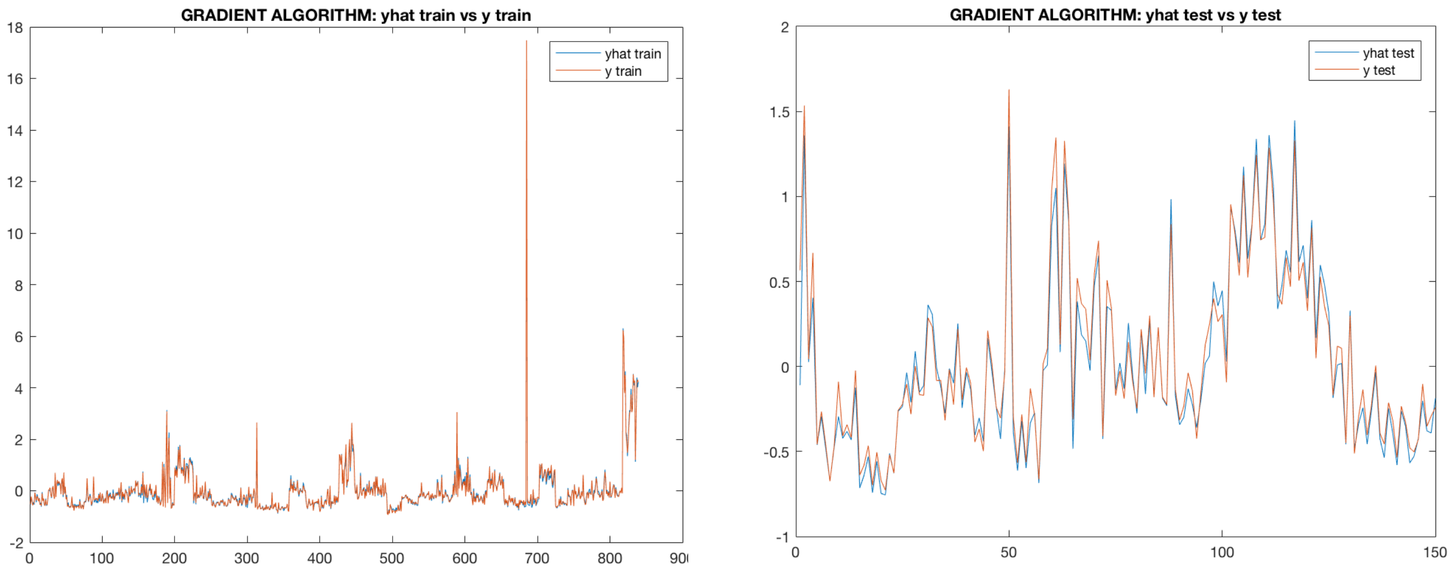


*Figure 3: Regression results vs. real values in train and test phase - Gradient algorithm, feature 7*

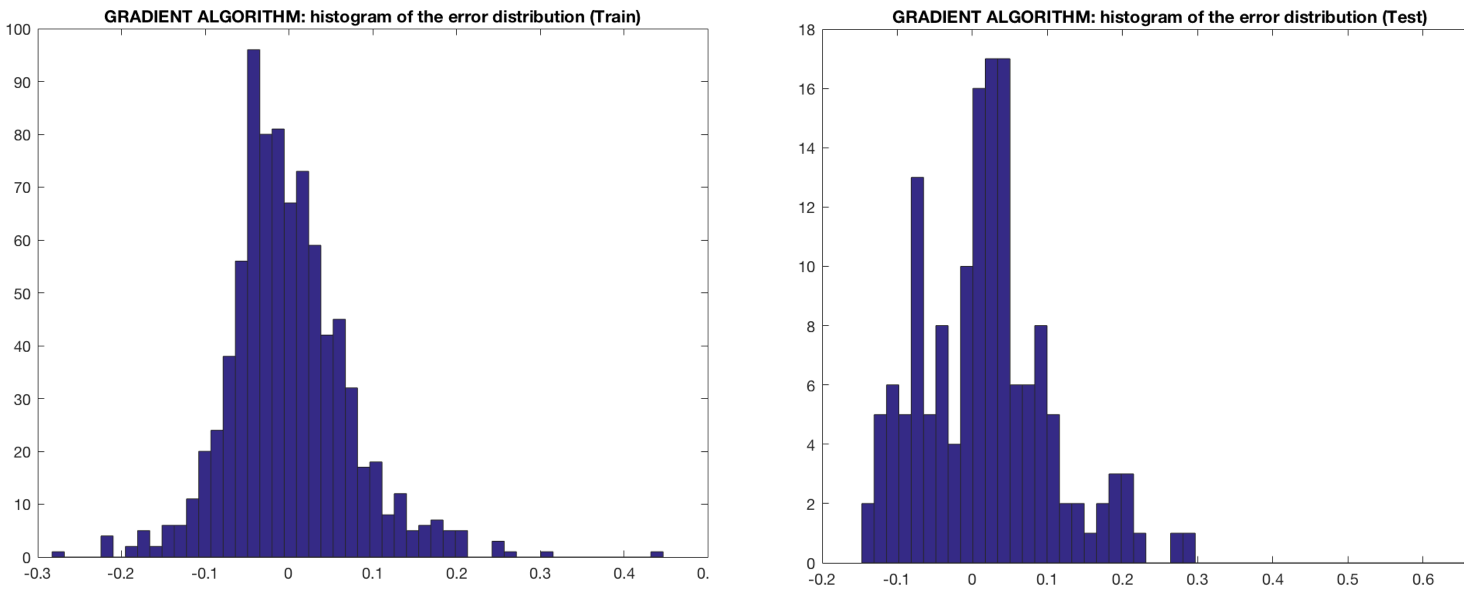


*Figure 4: Comparison of error distribution in train and test phase - Gradient algorithm, feature 7*

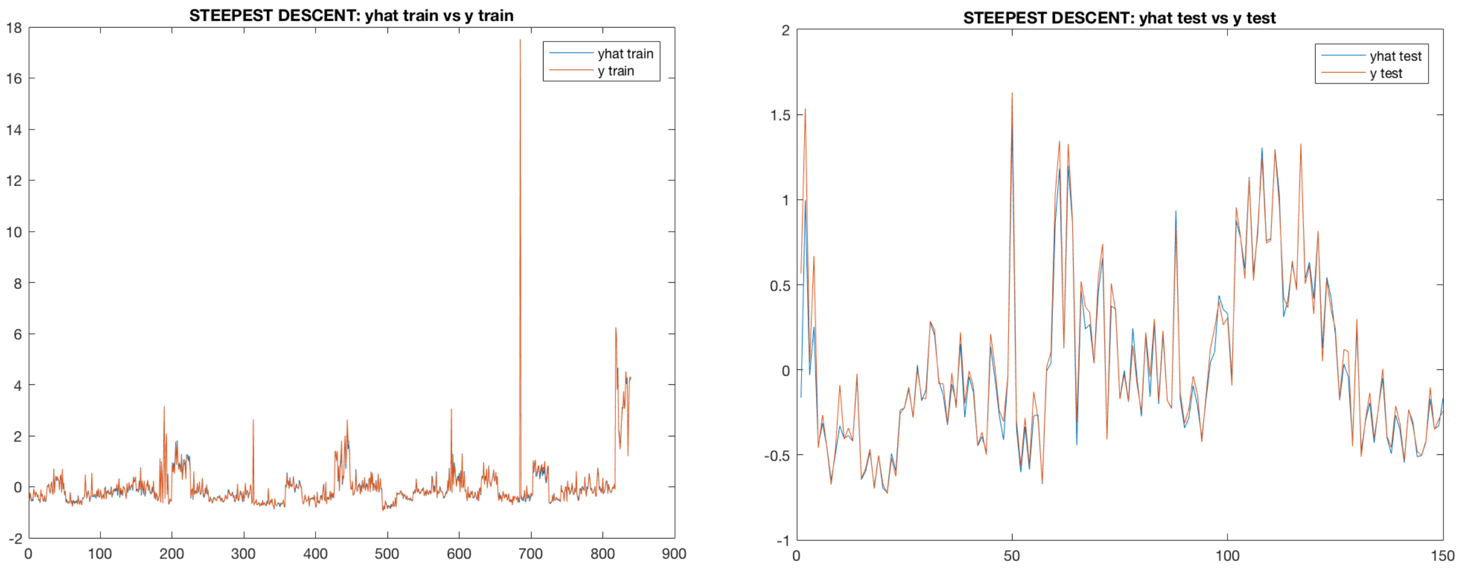


*Figure 5: Regression results in train and test phase - Steepest descent, feature 7*

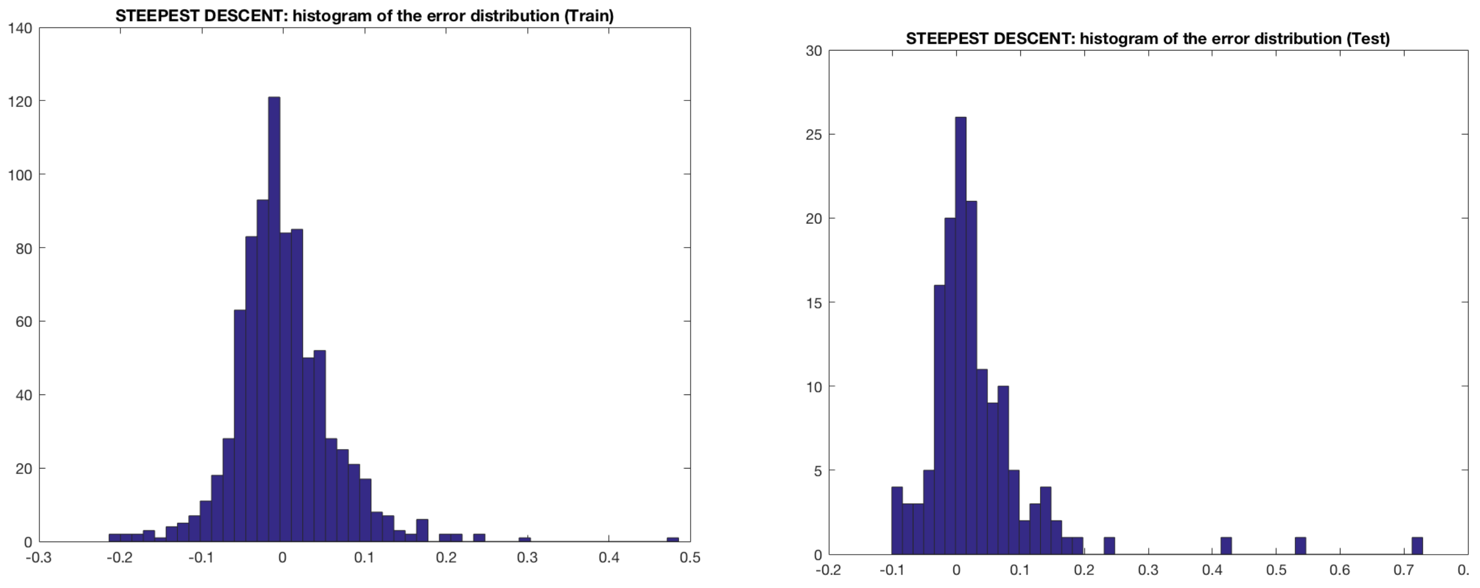
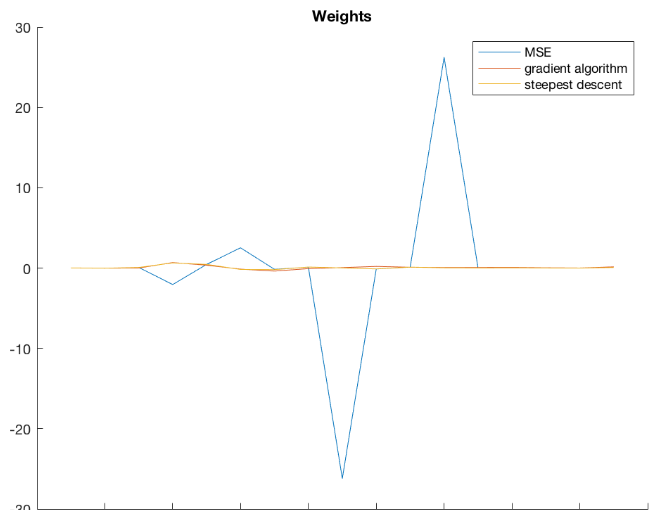


*Figure 6: Comparison of error distribution in train and test phase - Steepest descent, feature 7*



Pictures relative to regression with feature 5 are present in the publish of the Matlab scripts of Lab 01.

## Table of Contents

# Data preparation

```
clear all
close all

load('updrs.mat')

A=parkinsonsupdrs;

num_patients=max(A(:,1));

new_matrix=ones(1,22); % we define the matrix in which we insert
 grouped values

for ind=1:num_patients
    sub_rows=find(A(:,1)==ind); % indexes of the matrix corresponding
 to the 'ind' patient
    pat=A(sub_rows, :); % sub_matrix of 'ind' patient
    time=sort(abs(fix(pat(:,4)))); % we define an array of the days in
 which measurements were taken for patient 'ind'
    % we had to take the integer part of that values and make the
 absolute
    % value because there were some negative time values. Then I
 ordered
    % them by increasing order
    time=unique(time); % we delete duplicates in the array of days
    for i=1:length(time) %we iterate within the days
        day=time(i);
        time_rows=find(A(:,1)==ind & abs(fix(A(:,4)))==day); % we find
 the rows with all the measurements taken on patient 'ind' in the day
 'day'
        pat_time=A(time_rows, :); % we build a matrix of the
 measurements of patient 'ind' on day 'day'
        pat_time(:,4)=abs(fix(pat_time(:,4))); % we adjust the time
 column in order to have integer day indexes
        mean_row=mean(pat_time,1);
        new_matrix=[new_matrix;mean_row];
    end
end

pazienti=new_matrix(2:end,:);

save('parkinson.mat','new_matrix');
```

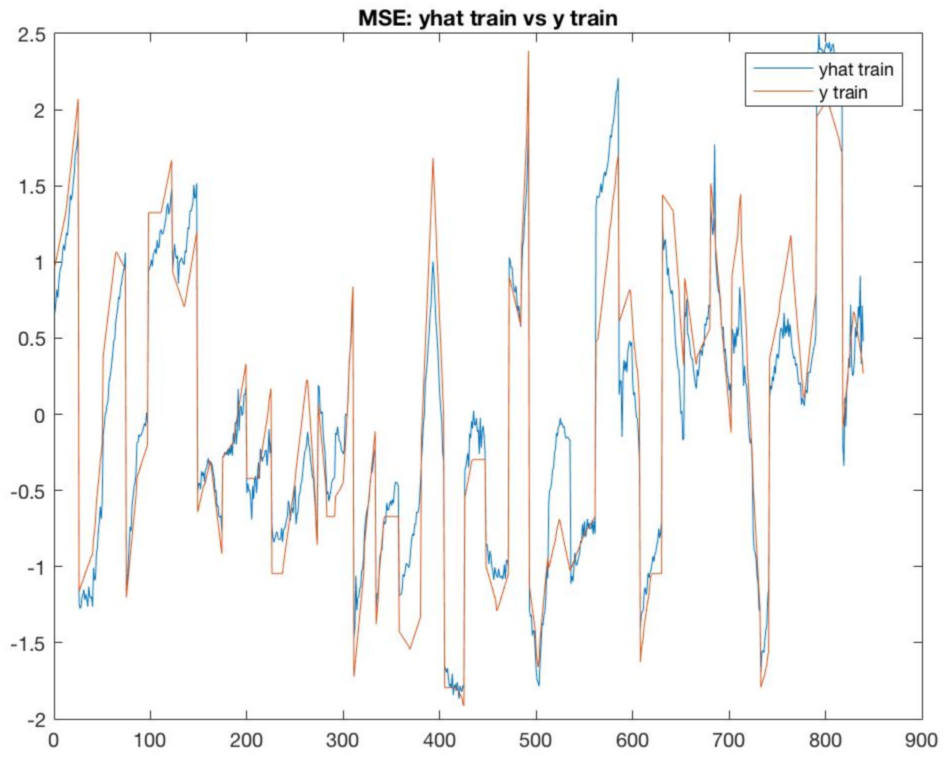
```
%-------  we build data_train and data_test matrices-----

% data_train=ones(1,22);
% data_test=ones(1,22);
% for i=1:length(new_matrix(:,1))
%     if new_matrix(i,1)<=36
%         data_train=[data_train;new_matrix(i,:)];
%     else
%         data_test=[data_test;new_matrix(i,:)];
%     end
% end
%
% data_train=data_train(2:end,:);
% data_test=data_test(2:end,:);

data_train = pazienti(pazienti(:,1)<37,:);
data_test = pazienti(pazienti(:,1)>36,:);


% ---- applying normalization to data_train matrix

m_data_train=mean(data_train,1);
v_data_train=var(data_train,1);
s_v_data_train=sqrt(v_data_train);

% data_train_norm=ones(length(data_train(:,1)),22);
% for i=1:length(data_train(:,1))
%     data_train_norm(i,1:4)=data_train(i,1:4);
%     for f=5:22
%         data_train_norm(i,f)=(data_train(i,f)-m_data_train(f))/
s_v_data_train(f);
%     end
% end

o = ones(size(data_train,1),1);
data_train_norm = data_train;
data_train_norm(:,5:end) = (data_train(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_train_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_train_norm = var(data_train_norm,1);


% ---- applying normalization to data_test matrix

% data_test_norm=ones(length(data_test(:,1)),22);
% for i=1:length(data_test(:,1))
%     data_test_norm(i,1:4)=data_test(i,1:4);
%     for f=5:22
%         data_test_norm(i,f)=(data_test(i,f)-m_data_train(f))/
s_v_data_train(f);
```

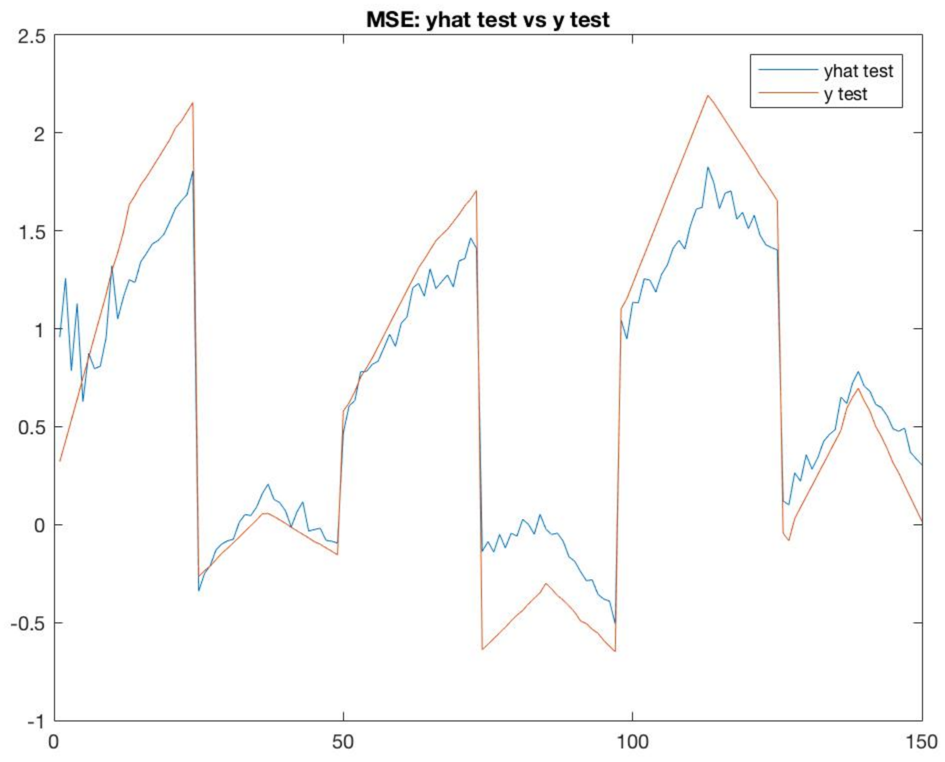
```
%     end
% end

o = ones(size(data_test,1),1);
data_test_norm = data_test;
data_test_norm(:,5:end) = (data_test(:,5:end) -
 o*m_data_train(:,5:end)) ./ (o*s_v_data_train(:,5:end));

mean_test_norm = mean(data_train_norm,1); % verify that normalization
 is effective
var_test_norm = var(data_train_norm,1);

% DEFINITION OF REGRESSOR AND REGRESSAND

F0=5; % we have to try with feature 7 and feature 5

y_train=data_train_norm(:,F0);
X_train=data_train_norm;
X_train(:,F0)=[];

y_test=data_test_norm(:,F0);
X_test=data_test_norm;
X_test(:,F0)=[];
```

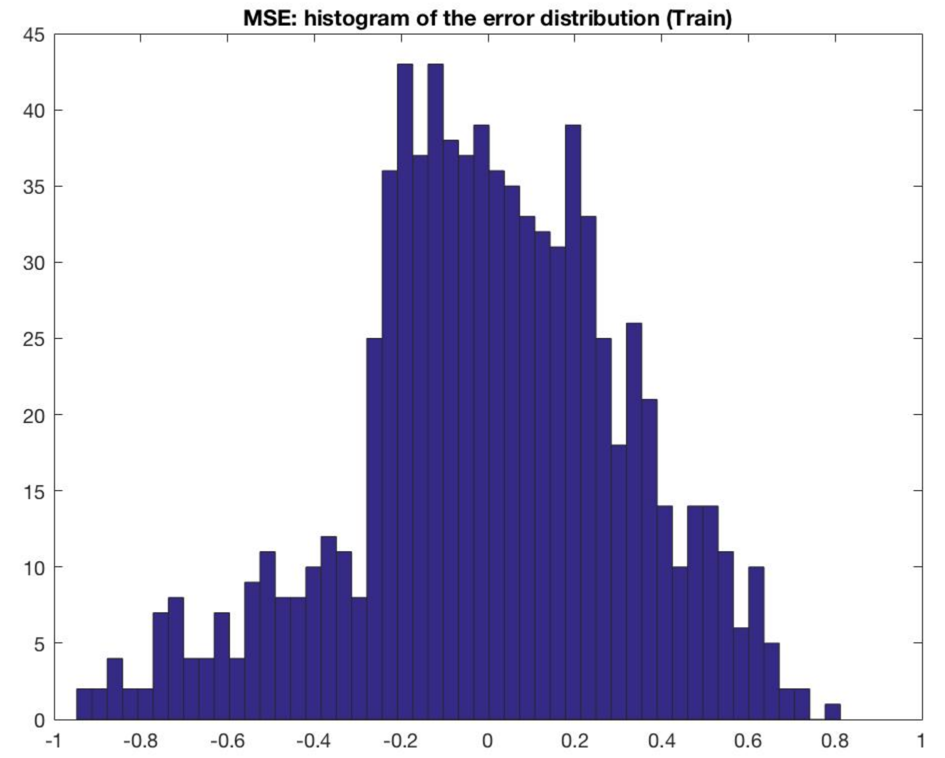
## Minimum Square Error (MSE)

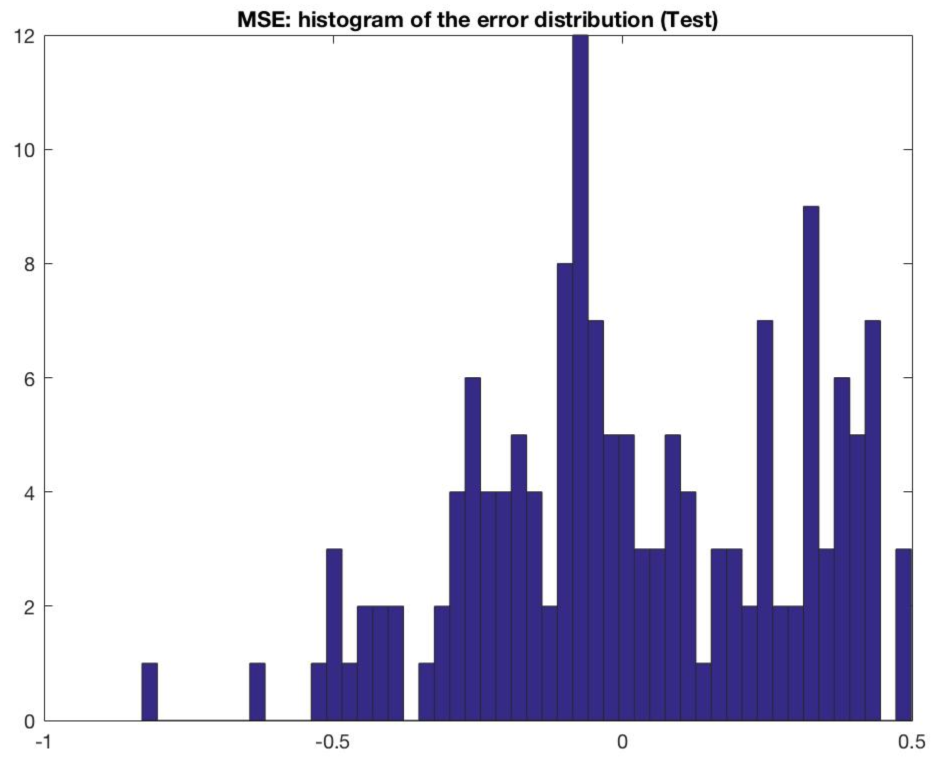
```
X_T=X_train(:,5:end).';

a_hat=inv(X_T*X_train(:,5:end))*X_T*y_train;

yhat_train=X_train(:,5:end)*a_hat;
yhat_test=X_test(:,5:end)*a_hat;

error_test=y_test-yhat_test;
error_train=y_train-yhat_train;

figure
plot(yhat_train)
hold on
plot(y_train)
hold off
title('MSE: yhat train vs y train')
legend('yhat train','y train')

figure
plot(yhat_test)
hold on
plot(y_test)
hold off
title('MSE: yhat test vs y test')
legend('yhat test','y test')

figure
```

```
hist(error_train,50) % for the feature 5 the error is very variable,
 that means that feature 5 is less correlated to others
% for the feature 7 the error is distributed according something that
 is
% like a gaussian
title('MSE: histogram of the error distribution (Train)')

figure
hist(error_test,50) % for the feature 5 the error is very variable,
 that means that feature 5 is less correlated to others
% for the feature 7 the error is distributed according something that
 is
% like a gaussian
title('MSE: histogram of the error distribution (Test)')
```




MSE: yhat test vs y test
yhat test
y test
2.5
2
1.5
1
0.5
0
-0.5
-1
0
50
100
150

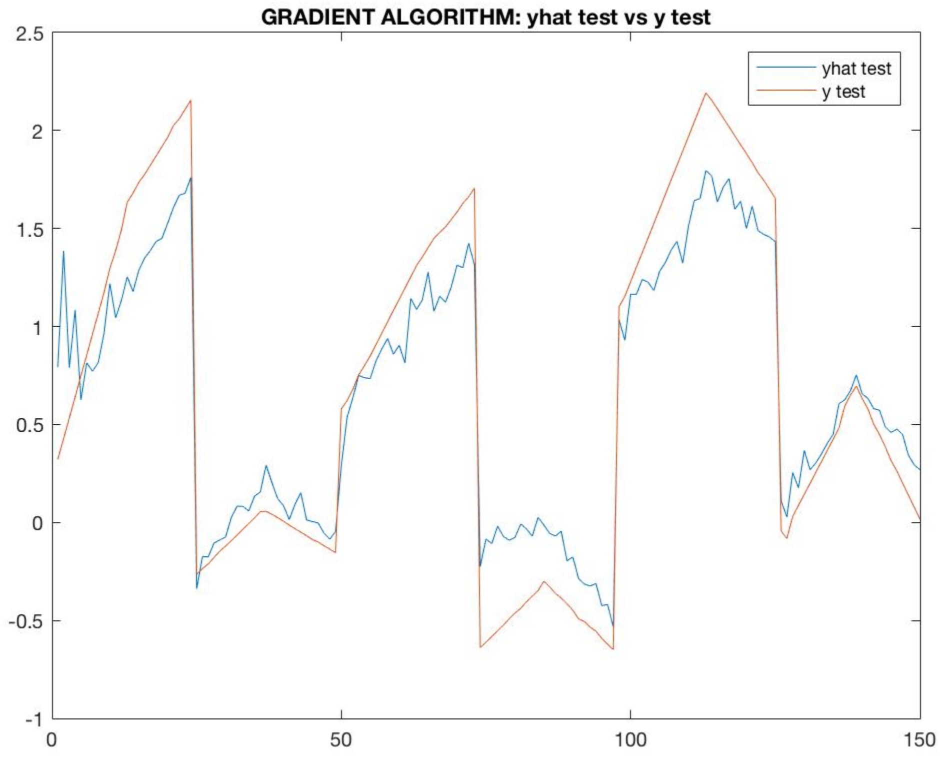

MSE: histogram of the error distribution (Train)
45
40
35
30
25
20
15
10
5
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

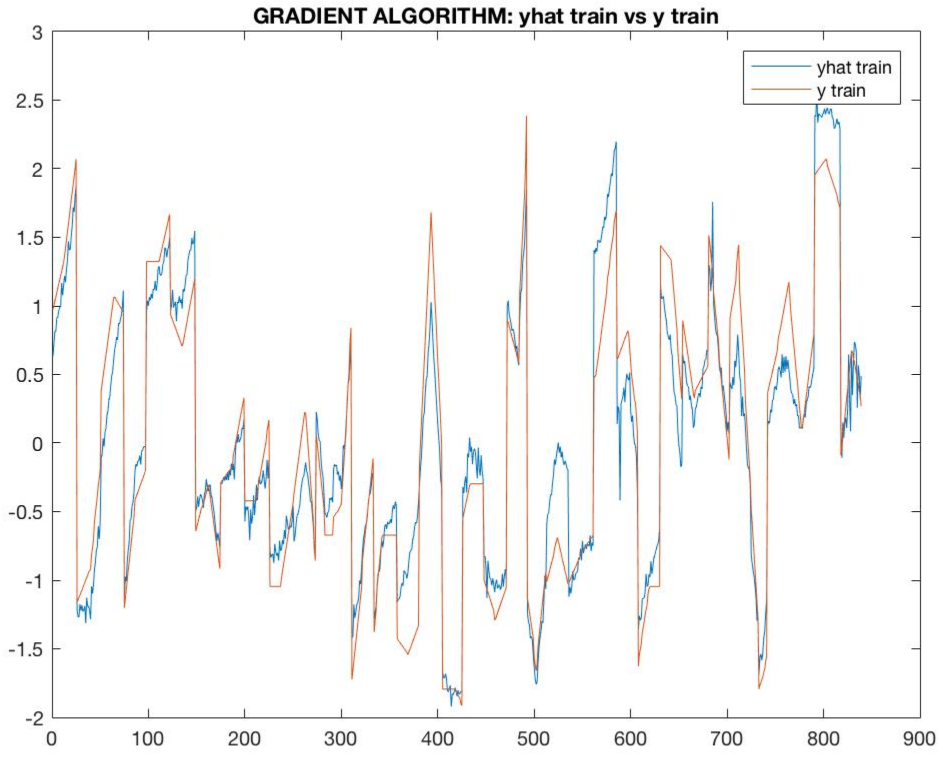


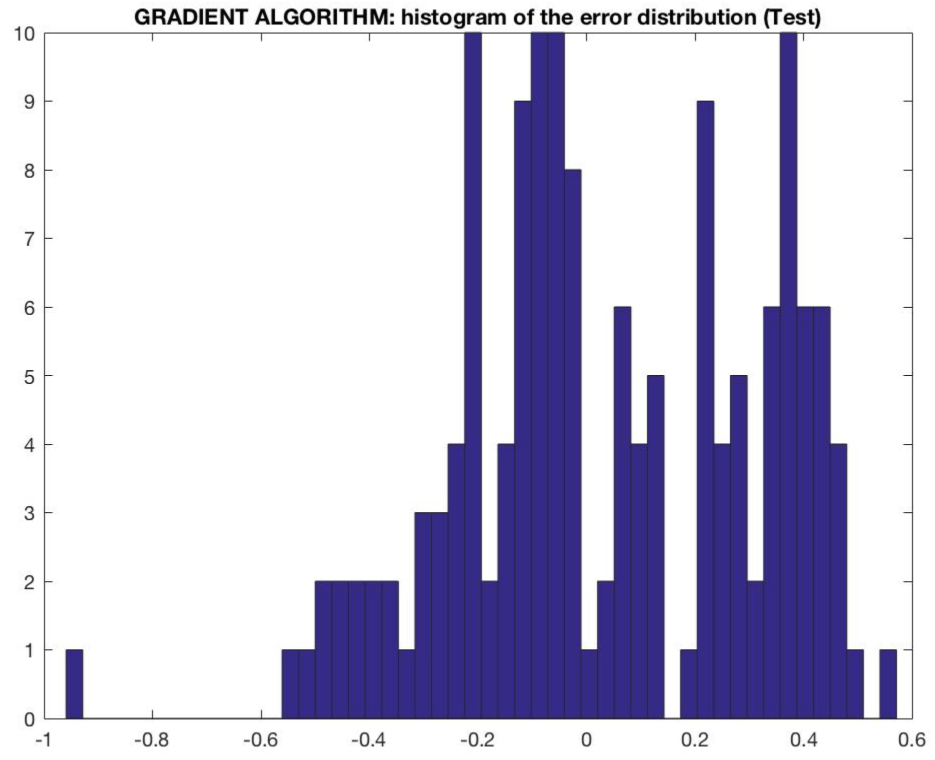
## Gradient Algorithm

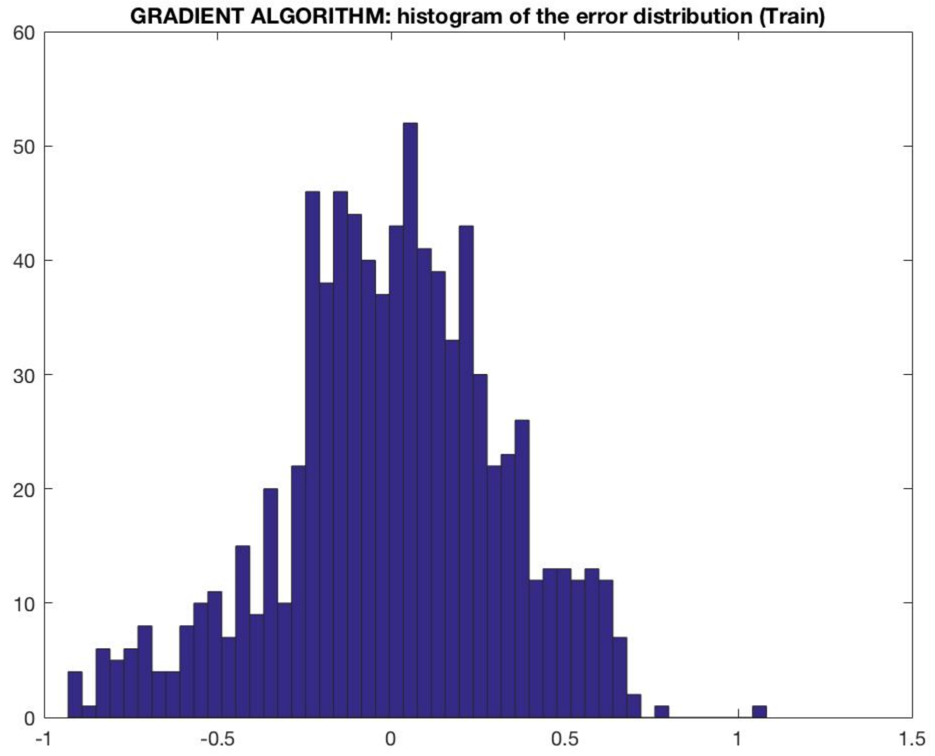
```
rng('default') % allows to have the same random vector each time we
 run the script
a_i = rand(17,1); % generates a 21x1 matrix (a vector) of uniformly
 distributed random numbers
gamma=0.000001; % learning coefficient (usually 0.01/0.001/0.0001)
epsilon=0.00001; % stopping rule: a_hat(i+1)-a_hat(i)<epsilon
a_old=zeros(17,1);
n=0;
a_story=a_i';

while (norm(a_i - a_old) > epsilon)
    grad= - 2 * X_T * y_train + 2 * X_T * X_train(:,5:end) * a_i;
    a_story=[a_story;a_i'];
    a_old=a_i;
    a_i = a_i - (gamma * grad);
    n=n+1;
end

yhat_ga_test=X_test(:,5:end)*a_i;
error_ga_test=y_test-yhat_ga_test;

yhat_ga_train=X_train(:,5:end)*a_i;
error_ga_train=y_train-yhat_ga_train;
```

```
figure
plot(yhat_ga_test)
hold on
plot(y_test)
hold off
title('GRADIENT ALGORITHM: yhat test vs y test')
legend('yhat test','y test')

figure
plot(yhat_ga_train)
hold on
plot(y_train)
hold off
title('GRADIENT ALGORITHM: yhat train vs y train')
legend('yhat train','y train')

figure
hist(error_ga_test,50)
title('GRADIENT ALGORITHM: histogram of the error distribution
 (Test)')

figure
hist(error_ga_train,50)
title('GRADIENT ALGORITHM: histogram of the error distribution
 (Train)')
```


GRADIENT ALGORITHM: yhat test vs y test
yhat test
y test
2.5
2
1.5
1
0.5
0
-0.5
-1
0
50
100
150


GRADIENT ALGORITHM: yhat train vs y train
yhat train
y train
3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900

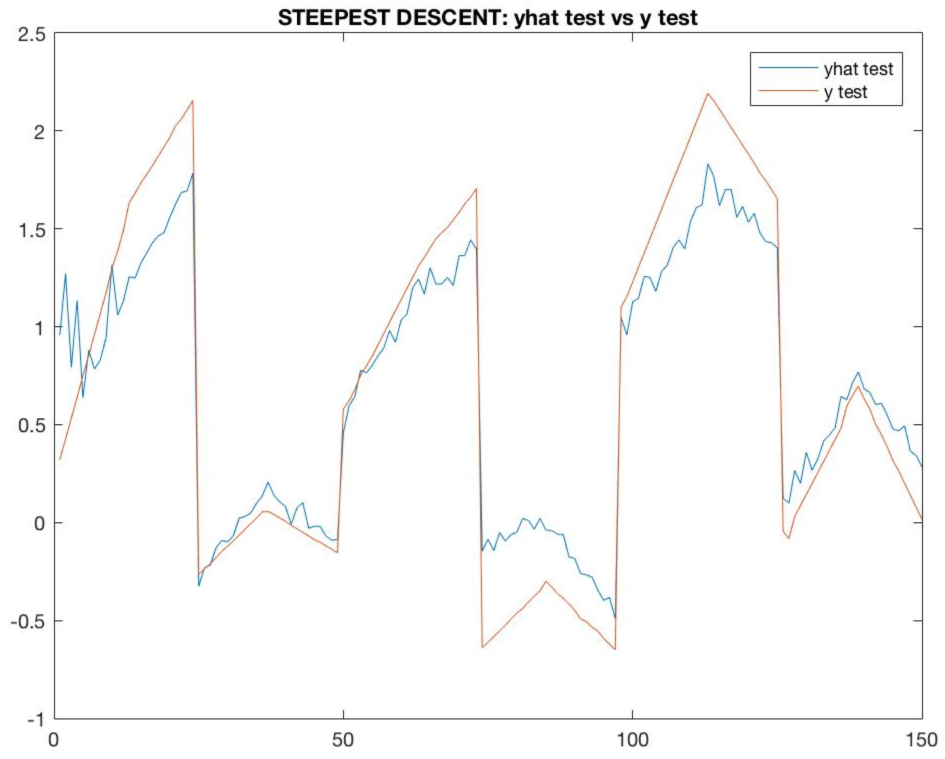

GRADIENT ALGORITHM: histogram of the error distribution (Test)

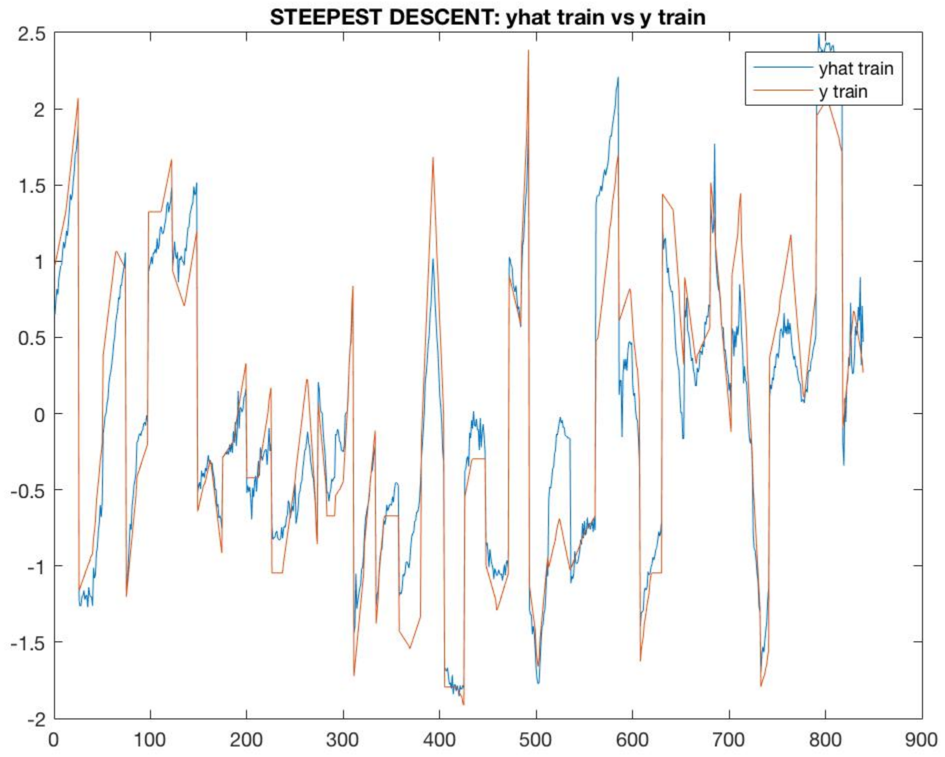

GRADIENT ALGORITHM: histogram of the error distribution (Train)

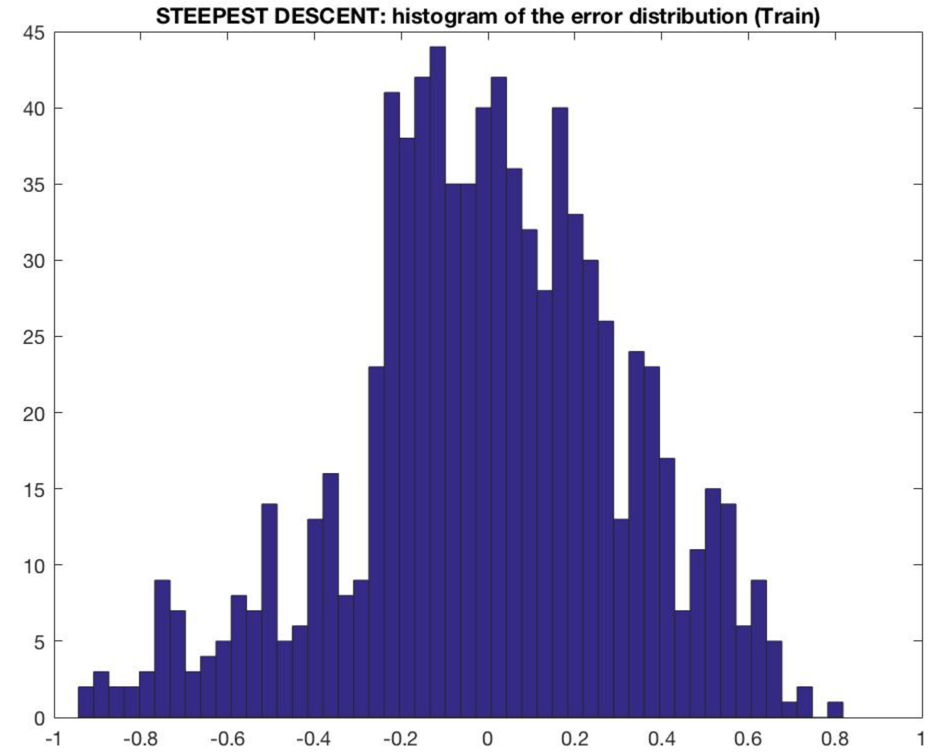
## Steepest Descent Algorithm

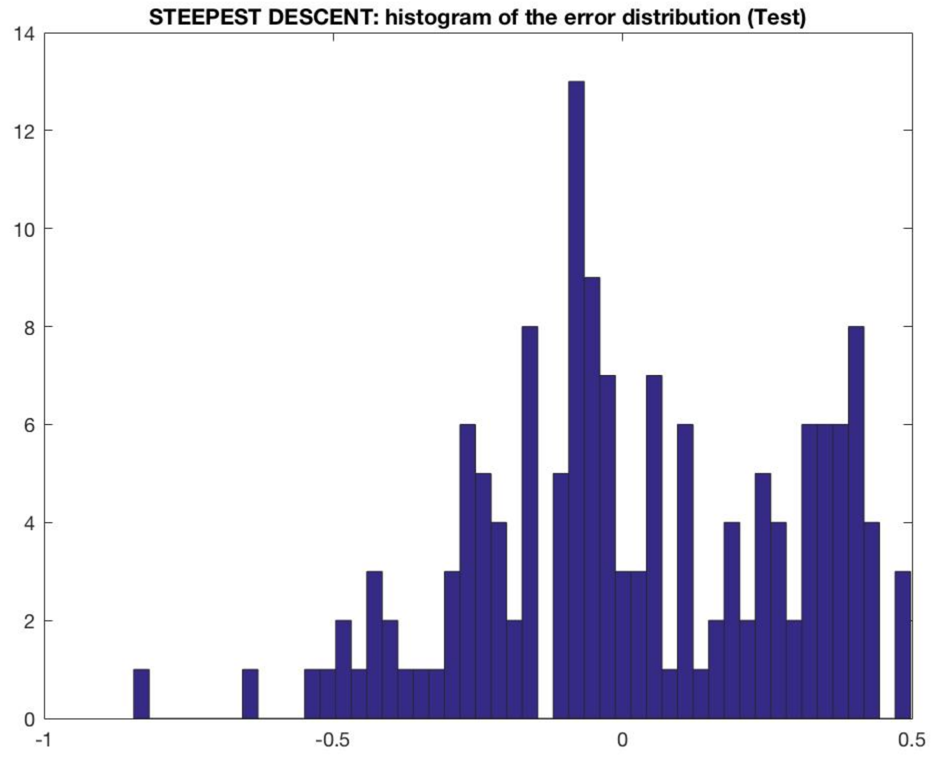
```
rng('default') % allows to have the same random vector each time we
 run the script
a_i_sd = rand(17,1); % generates a 21x1 matrix (a vector) of uniformly
 distributed random numbers
gamma=0.000001; % learning coefficient (usually 0.01/0.001/0.0001)
epsilon=0.00001; % stopping rule: a_hat(i+1)-a_hat(i)<epsilon
a_old_sd=zeros(17,1);
n=0;
a_story=a_i';

while (norm(a_i_sd - a_old_sd) > epsilon)
    grad_sd= - 2 * X_T * y_train + 2 * X_T * X_train(:,5:end) *
 a_i_sd;
    hessian = 4 * X_T * X_train(:,5:end);
    a_old_sd = a_i_sd;
    a_i_sd = a_i_sd - ((norm(grad_sd)^2 * grad_sd)/(grad_sd.' *
 hessian * grad_sd));
end

yhat_sd_train=X_train(:,5:end)*a_i_sd;
error_sd_train=y_train-yhat_sd_train;

yhat_sd_test=X_test(:,5:end)*a_i_sd;
error_sd_test=y_test-yhat_sd_test;

figure
plot(yhat_sd_test)
hold on
plot(y_test)
hold off
title('STEEPEST DESCENT: yhat test vs y test')
legend('yhat test','y test')


figure
plot(yhat_sd_train)
hold on
plot(y_train)
hold off
title('STEEPEST DESCENT: yhat train vs y train')
legend('yhat train','y train')

figure
hist(error_sd_train,50)
title('STEEPEST DESCENT: histogram of the error distribution (Train)')

figure
hist(error_sd_test,50)
title('STEEPEST DESCENT: histogram of the error distribution (Test)')
```

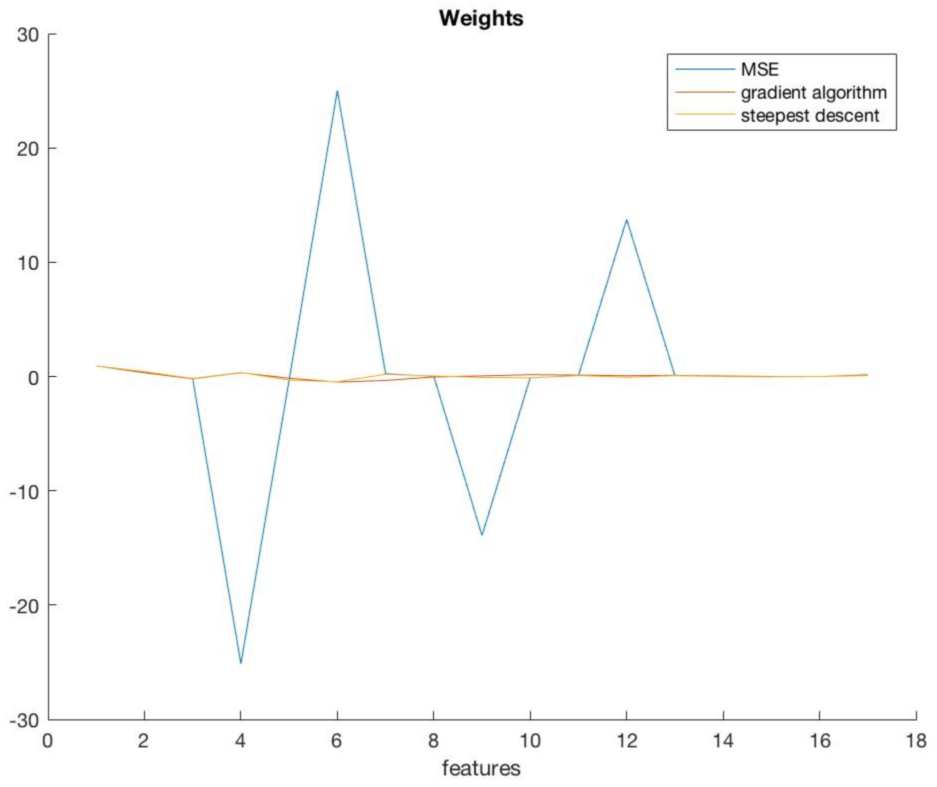
```
figure
hold on
plot(a_hat)
plot(a_i)
plot(a_i_sd)
hold off
title('Weights')
xlabel('features')
legend('MSE','gradient algorithm','steepest descent')
```




STEEPEST DESCENT: yhat train vs y train
yhat train
y train
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
100
200
300
400
500
600
700
800
900


STEEPEST DESCENT: histogram of the error distribution (Train)
45
40
35
30
25
20
15
10
5
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


STEEPEST DESCENT: histogram of the error distribution (Test)
0
2
4
6
8
10
12
14
-1
-0.5
0
0.5


Weights
MSE
gradient algorithm
steepest descent
30
20
10
0
-10
-20
-30
0
2
4
6
8
10
12
14
16
18
features

*Published with MATLAB® R2016a*