

- 1 **Parkinson's disease**
- 2 **Laboratory # 1**
 - Prepare and analyze the data
 - Perform regression
- 3 **Structure of the report**

Parkinson's disease [1]

Very short description:

- Patients affected by Parkinson's disease cannot exactly control their muscles. In particular they show tremor, they walk with difficulties and, in general, they have problems in starting a movement. Many of them cannot speak correctly, since they cannot control the vocal chords and the vocal tract. It has been shown that they overcome the illness if they dance or have an external clock that gives the time
- Levodopa is prescribed to the patients, but most of the medicine, which should be absorbed in the intestine, is absorbed by the stomach; as the movements become slower and slower, levodopa stays more and more in the stomach and cannot reach the intestine
- The beneficial effects of levodopa last for some time, and then a new dose of levodopa should be taken. The neurologist decides when the patient should take levodopa and how much levodopa he/she should take, but it is difficult for the neurologist to optimize the treatment, because of the continuous progression of the illness.

Parkinson's disease [2]

- The severity of the illness is measured by neurologists, who judge the patients by asking them to perform many movements (for example tapping the other four fingers with the thumb, or rising from a chair, or walking a short distance, or saying some words). Adding together the scores of each single movement gives the final grade, which is called total **UPDRS** (Unified Parkinson's Disease Rating Scale). The visit takes a lot of time, different neurologists may give slightly different scores.
- It would be useful to find an automatic way to give the patient an objective score, which can be measured several times during the day and help the neurologist to optimize the treatment.

Parkinson's disease [3]

- In the literature, articles have been published in which the authors claim that it is possible to use parameters of voice to predict the total UPDRS: it is then sufficient to record voice samples (for example using a smartphone), generate these voice parameters (features) and then use a regression technique to predict UPDRS. The claim is not so correct, since Parkinson's disease not always affects voice.
- Goal of the lab is to use linear regression to predict UPDRS from a set of voice parameters.

Table of Contents

1 Parkinson's disease

2 Laboratory # 1

- Prepare and analyze the data
- Perform regression

3 Structure of the report

Prepare and analyze the data [1]

- Download from <https://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring> the Data Folder and Data Set Description. In particular, download files `parkinsons_updrs.data` and `parkinsons_updrs.names` from <https://archive.ics.uci.edu/ml/machine-learning-databases/parkinsons/telemonitoring/>. The data were obtained by some researchers who evaluated in the same day total UPDRS and motor UPDRS of some patients and recorded the speech of the patients to measure voice parameters. The patients were analyzed several times in 6 months and several records exist for each patients, at different times. The outcome is a matrix with many rows (one for each measurement) and many columns (one for each feature).
- File `parkinsons_updrs.data` stores F columns and N rows; the columns are separated by commas (csv file). Change the extension of the file from `.data` to `.csv`, so that you can have a look at the file using Excel or LibreOfficeCalc

Prepare and analyze the data [2]

- A useful Python library that can be used to analyze data is Pandas: download it
- Open Spyder3 or Pycharm. Start a new script in which you import pandas, matplotlib.pyplot, NumPy:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

- Read the Parkinson's data file using Pandas function `pd.read_csv('filename')` .

```
1 x=pd.read_csv("parkinsons_updrs.csv")
```

- Search the web for the usage of Pandas. The main things you have to know is that x is a **DataFrame**, and that many attributes and methods exist for DataFrames (see <https://pandas.pydata.org/pandas-docs/stable/api.html#id2>).

Prepare and analyze the data [3]

- Examples of methods associated with DataFrames:

```
1 x.info()#gives you information about the data
2 x.describe()#descr. of dataset (min, max, mean, etc of each feat.)
3 x.plot.hist(bins=50)#plots the histograms of all the features
4 x.plot.scatter()#plots the scatter plot, i.e. the data of column i
5 #versus the data of column k
6 features=list(x.columns)}#list with the names of the features
7 x.cov()}#gives the covariance matrix for the features in the columns
8 x.values()}#gives the NumPy Nddarray with the data
```

Prepare and analyze the data [4]

- Start checking the data (mandatory step each time you work with a new dataset):

- Write in your code

```
1 x.describe.T
2 x.info()
```

and look at the printed values. Check that there are no major problems with the data (no missing values, no out-of-scale values, etc). `x.info()` gives the following output

```
1 subject#          5875 non-null int64
2 age             5875 non-null int64
3 sex             5875 non-null int64
4 test_time       5875 non-null float64
5 motor_UPDRS     5875 non-null float64
6 total_UPDRS     5875 non-null float64
7 Jitter(%)       5875 non-null float64
8 Jitter(Abs)     5875 non-null float64
9 Jitter:RAP      5875 non-null float64
10 Jitter:PPQ5     5875 non-null float64
11 Jitter:DDP      5875 non-null float64
12 Shimmer         5875 non-null float64
```

Prepare and analyze the data [5]

13	Shimmer (dB)	5875	non-null	float64
14	Shimmer:APQ3	5875	non-null	float64
15	Shimmer:APQ5	5875	non-null	float64
16	Shimmer:APQ11	5875	non-null	float64
17	Shimmer:DDA	5875	non-null	float64
18	NHR	5875	non-null	float64
19	HNR	5875	non-null	float64
20	RPDE	5875	non-null	float64
21	DFA	5875	non-null	float64
22	PPE	5875	non-null	float64

which means that there are 5875 row with no missing values and the read values are either integer or float numbers.

Prepare and analyze the data [6]

- 2 Check the names/meanings of the available features:

```
1 features=list(x.columns)
2 print(features)
```

The list of features is 'subject#', 'age', 'sex', 'test_time', 'motor_UPDRS', 'total_UPDRS', 'Jitter(%)', 'Jitter(Abs)', 'Jitter:RAP', 'Jitter:PPQ5', 'Jitter:DDP', 'Shimmer', 'Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5', 'Shimmer:APQ11', 'Shimmer:DDA', 'NHR', 'HNR', 'RPDE', 'DFA', 'PPE'.

- It is stupid to use 'subject#' as a regressor: a new patient will have another ID and we cannot rely on IDs of previous patients to predict UPDRS of a new patient. Therefore, we need to remove 'subject#' from the features.
- The new patient will be analyzed, not necessarily knowing how much time he/she has been suffering from the illness. Again we cannot rely on 'test_time' to perform regression and also this feature should be removed.

Prepare and analyze the data [7]

- The other features are all reasonable. **Jitter** is the variation of the fundamental **frequency** (or, conversely, its period) in signals that should be periodic but are not (it is impossible that the frequency of a sinusoidal signal generated by an electronic equipment never changes; it is impossible that a periodic vocal signal like 'a' is perfectly periodic). **Shimmer** is the variation of **amplitude** in signals that should be periodic but are not. NHR is the noise to harmonics ratio; HNR is the harmonics to noise ratio. RPDE is Recurrence Period Density Entropy, DFA is the Detrended Fluctuation Analysis, PPE is Perceived Vocal Effort. All these features/parameters are related to voice and are automatically evaluated by specific software that uses a voice signal as input.

Prepare and analyze the data [8]

3 Write in your script

```
1 X=x.drop(['subject#','test_time'],axis=1)# drop unwanted features
2 Np,Nc=X.shape
3 # Np = number of rows/patients
4 # Nf=number of regressors + 1(regressand)
5 features=list(X.columns)
```

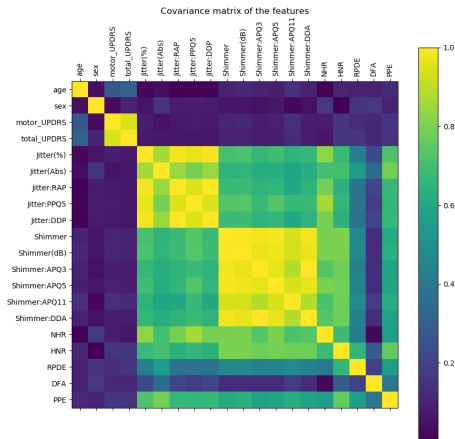
to remove the unwanted features

4 Let us check if features are correlated. First normalize all the data, then use the DataFrame method cov to evaluate the covariance matrix and plot it using matplotlib

```
1 xnorm=(x-x.mean())/x.std()#normalize the entire dataset
2 c=xnorm.cov()#measure the covariance
3 plt.figure(figsize=(10,10))
4 plt.matshow(np.abs(c.values),fignum=0)
5 plt.xticks(np.arange(len(features)), features, rotation=90)
6 plt.yticks(np.arange(len(features)), features, rotation=0)
7 plt.colorbar()
8 plt.title('Covariance matrix of the features',pad=70)
```

Look also at the values in DataFrame c.

Prepare and analyze the data [9]



Prepare and analyze the data [10]

- Clearly motor_UPDRS is correlated to total_UPDRS, while the correlation between total_UPDRS and voice features is not so large. Moreover jitter features are all correlated among each other, the same for shimmer features. This might give rise to **collinearity** or multicollinearity: one feature among the regressors can be linearly derived from other regressors, which means that many different vectors \mathbf{w} exist that solve the problem with the same value of the objective function. In such cases it might be convenient to remove all but one of the linearly dependent features, selecting the one that has the highest correlation coefficient with the regressand (total_UPDRS in our case). However we will keep all the features to see what happens.

Prepare and analyze the data [11]

- Prepare a new DataFrame in which you **randomly permute (shuffle)** the rows of the original DataFrame. This operation avoids that the data of only the first patients appear in the training dataset. The rationale behind this operation is that we pretend that all the measurements are related to different patients at different stages in the illness evolution and we take the first rows of the DataFrame to train/validate the regression model and the remaining for rows to test the performance of the found model.

```
1 np.random.seed(1) # set the seed for random shuffling
2 indexsh=np.arange(Np)
3 np.random.shuffle(indexsh)
4 Xsh=X.copy(deep=True)
5 Xsh=Xsh.set_axis(indexsh,axis=0,inplace=False)
6 Xsh=Xsh.sort_index(axis=0)
```

- The regressand will be **total UPDRS**, whereas all the other features (including motor UPDRS) will be regressors.

Perform regression [1]

- It is now time to start generating the regression model. We assume that random variable `total_UPDRS` linearly depends on the random variables `sex`, `age`, `motor_UPDRS`, `shimmer`, `jitter`, etc:

$$y = w_1x_1 + w_2x_2 + \cdots + w_{N_f}x_{N_f}$$

where y, x_1, \dots, x_{N_f} are all random variables and w_1, \dots, w_{N_f} are the weights to be found.

Perform regression [2]

- In the previous model we assume that all the random variables have zero mean. A more complete model is

$$y = w_1x_1 + w_2x_2 + \cdots + w_{N_f}x_{N_f} + C$$

but it is convenient to work with features with zero-mean and variance one (setting variance to one reduces numerical problems). Therefore we must perform **normalization** of the data, by removing the mean and divide by the standard deviation each random variable.

In order to be consistent with our scenario, we must consider that the feature means and standard deviations **can only be measured on the training dataset** (we do not know in advance the parameters of future patients, can we?).

Perform regression [3]

- We will use the first 50% of the rows of the shuffled matrix as **training** points (define the new DataFrame as `x_tr`), the subsequent 25% for **validation** (define the new DataFrame as `x_va`), and the remaining 25% for **testing** (define the new DataFrame as `x_te`). This is accomplished with the instructions

```
1 Ntr=int(Np*0.5) # number of training points
2 Nva=int(Np*0.25) # number of validation points
3 Nte=Np-Ntr-Nva # number of test points
4 X_tr=Xsh[0:Ntr]# dataframe that contains only the training data
5 mm=X_tr.mean()# mean (series)
6 ss=X_tr.std()# standard deviation (series)
7 my=mm['total_UPDRS']# mean of total UPDRS
8 sy=ss['total_UPDRS']# st.dev of total UPDRS
```

A series is substantially a DataFrame with just one column (not exact, just to give an idea).

Perform regression [4]

- Now we can normalize the data and split them into training, validation, test subsets:

```
1 Xsh_norm=(Xsh-mm)/ss# normalized data
2 ysh_norm=Xsh_norm['total_UPDRS']# regressand only
3 Xsh_norm=Xsh_norm.drop('total_UPDRS',axis=1)# regressors only
4
5 X_tr_norm=Xsh_norm[0:Ntr]# training regressors
6 X_va_norm=Xsh_norm[Ntr:Ntr+Nva]# validation regressors
7 X_te_norm=Xsh_norm[Ntr+Nva:]# test regressors
8 y_tr_norm=ysh_norm[0:Ntr]# training regressand
9 y_va_norm=ysh_norm[Ntr:Ntr+Nva]# validation regressand
10 y_te_norm=ysh_norm[Ntr+Nva:]# test regressand
```

Perform regression [5]

- Regression procedure:

- 1 DataFrames `X_tr_norm` and `Y_tr_norm` will be used to find:

$$\hat{\mathbf{w}} = \arg \min \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

(`X_tr_norm=X`, `Y_tr_norm=y`). With LLS:

```
1 w_hat=np.linalg.inv(X_tr_norm.T@X_tr_norm)@(X_tr_norm.T@y_tr_norm)
```

- 2 Depending on the method used to find $\hat{\mathbf{w}}$, validation dataset (i.e `X_va_norm` and `Y_va_norm`) will be used to avoid overfitting. With LLS, there is no validation procedure and therefore `X_va_norm` and `Y_va_norm` will not be used.

Perform regression [6]

- ④ Then, having found $\hat{\mathbf{w}}$, the test dataset will be used to evaluate

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$$

($\mathbf{X}_{te_norm} = \mathbf{X}$). Goodness of $\hat{\mathbf{w}}$ will be measured by comparing $\hat{\mathbf{y}}$ and \mathbf{Y}_{te_norm} . For the LLS case,

```
1 y_hat_te_norm=X_te_norm@w_hat
2 MSE_norm=np.mean((y_hat_te_norm-y_te_norm)**2)
```


Perform regression [7]

- ④ However, note that Y_te_norm is normalized, and therefore the value of MSE does not say much to a medical doctor (what is the unit of measurement?) and it is necessary to de-normalize \hat{y} . For the LLS case:

```
1 Y_hat_te=Y_hat_te_norm*sy+my
2 Y_te=Y_te_norm*sy+my
3 MSE=np.mean((Y_hat_te-Y_te)**2)
```

Alternatively, using random variables (subscript N means "normalized"):

$$y_N = \frac{y - \mu}{\sigma} \longrightarrow y = \sigma y_N + \mu$$

$$\hat{y}_N = \hat{\mathbf{w}}^T \mathbf{x}_N, \quad \hat{y} = \sigma \hat{y}_N + \mu$$

$$\text{MSE} = \mathbb{E}\{(\hat{y} - y)^2\} = \mathbb{E}\{[\sigma \hat{y}_N + \mu - (\sigma y_N + \mu)]^2\} = \sigma^2 \mathbb{E}\{(\hat{y}_N - y_N)^2\}$$

$$\text{MSE} = \sigma^2 \text{MSE}_N$$

Therefore, you can also write

```
1 MSE=sy**2*MSE_norm
```

Perform regression [8]

- Final measurements.

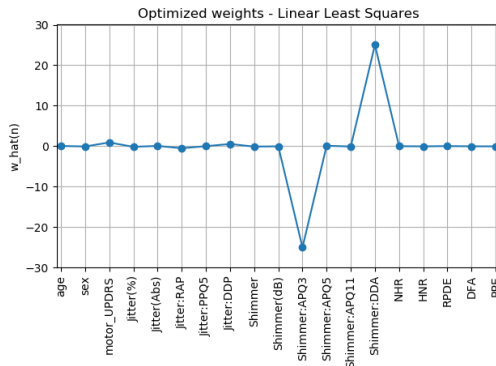
- Once you find \hat{w} , you must plot it in order to find potential problems

```

1 regressors=list(X_tr_norm.columns)
2 Nf=len(w_hat)
3 nn=np.arange(Nf)
4 plt.figure(tight_layout=True)
5 plt.plot(nn,w_hat,'-o')
6 ticks=nn
7 plt.xticks(ticks,regressors,rotation=90)#, **kwargs)
8 plt.ylabel('w_hat(n)')
9 plt.title('Optimized_weights_Linear_Least_Squares')
10 plt.margins(0.01,0.1)# leave some space
11 plt.grid()
12 plt.show()

```

Perform regression [9]



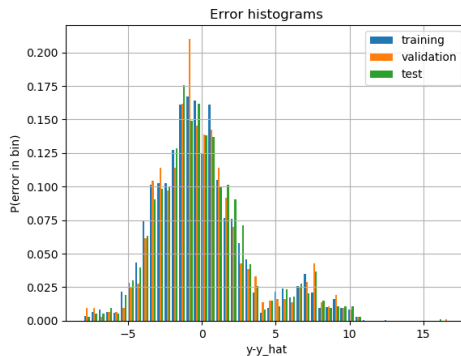
The effect of **collinearity** can be seen since the weight associated with 'Shimmer:APQ3' and 'Shimmer:DDA' are very large and with opposite signs. One of the two features should be removed.

Perform regression [10]

- ② You must check if the regression error shows peculiar trends, which might reveal an error in the script.

```
1 E_tr=(y_tr_norm-X_tr_norm@w_hat)*sy# training
2 E_va=(y_va_norm-X_va_norm@w_hat)*sy# validation
3 E_te=(y_te_norm-X_te_norm@w_hat)*sy# test
4 e=[E_tr,E_va,E_te]
5 plt.figure()
6 plt.hist(e,bins=50,density=True,histtype='bar',
7 label=['training','validation','test'])
8 plt.xlabel('y-y_hat')
9 plt.ylabel('P(error_in_bin)')
10 plt.legend()
11 plt.grid()
12 plt.title('Error_histograms')
```

Perform regression [11]



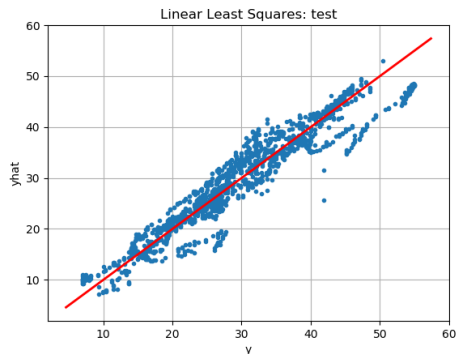
Clearly, the error does not have a Gaussian pdf, it is a mixture of two Gaussian pdfs, and there is not much difference in the three subsets (no overfitting).

Perform regression [12]

- ③ You must compare the true and the regressed value of y . Note: when you plot a versus b , a is on the y-axis, then b is on the x-axis.

```
1 Y_hat_te=(X_te_norm@w_hat)*sy+my
2 Y_te=Y_te_norm*sy+my
3 plt.figure()
4 plt.plot(Y_te,Y_hat_te,'.')
5 plt.xlabel('y')
6 plt.ylabel('yhat')
7 plt.grid()
8 plt.title('Linear Least Squares: test')
```

Perform regression [13]



Note that the estimated values \hat{y} are close to the true values y apart from some cases in which y is very large and \hat{y} takes smaller values with an error around 7-8 UPDRS points (which justifies the second Gaussian-like part of the histogram for error values around 8).

Perform regression [14]

- 4 Other important parameters are mean, standard deviation, mean square value, parameter R^2 of the error in each of the three subsets:

```
1 E_tr_mu=E_tr.mean()
2 E_tr_sig=E_tr.std()
3 E_tr_MSE=np.mean(E_tr**2)
4 y_tr=y_tr_norm*sy+my
5 R2_tr=1-E_tr_sig**2/np.mean(y_tr**2)
6 E_va_mu=E_va.mean()
7 E_va_sig=E_va.std()
8 E_va_MSE=np.mean(E_va**2)
9 y_va=y_va_norm*sy+my
10 R2_va=1-E_va_sig**2/np.mean(y_va**2)
11 E_te_mu=E_te.mean()
12 E_te_sig=E_te.std()
13 E_te_MSE=np.mean(E_te**2)
14 y_te=y_te_norm*sy+my
15 R2_te=1-E_te_sig**2/np.mean(y_te**2)
```

To show the results in a better form, a DataFrame can be generated and printed:

Perform regression [15]

```

1 rows=[ 'Training ', 'Validation ', 'test ' ]
2 cols=[ 'mean', 'std', 'MSE', 'R^2' ]
3 p=np.array([[ E_tr_mu, E_tr_sig, E_tr_MSE, R2_tr ],
4              [ E_va_mu, E_va_sig, E_va_MSE, R2_va ],
5              [ E_te_mu, E_te_sig, E_te_MSE, R2_te ]])
6 results=pd.DataFrame(p, columns=cols, index=rows)
7 print(results)

```

The printed output is

	mean	std	MSE	R^2
Training	3.013110e-13	3.243811	10.518730	0.988988
Validation	1.341350e-01	3.298703	10.892018	0.988534
test	1.333404e-01	3.265545	10.674310	0.988964



Conclusions that can be drawn looking at the above results are:

- The error mean in the training subset is zero (as it should be); the error mean is slightly positive in the other two subsets because the mean of total_UPDRS were evaluated using only the training subset, which means that total_UPDRS means in test and validation subsets are not exactly zero, after normalization.
- The error standard deviation is practically the same in the three subsets (no overfitting)

Perform regression [16]

- The coefficient of determination R^2 is close to 1, which means that the regression is pretty good.
- Since the standard deviation is around 3.2 points, this means that most of the times the regression error is around 3-6 points (see also the histogram), which might still be accepted by a medical doctor. Regression is not very precise, but having an error of 6 points when total_UPDRS is equal to 50 points can be accepted.

What you have to do [1]

- Repeat regression (exactly the same steps shown for LLS) using
 - ① **stochastic gradient algorithm** with **adam** method 
 - ② **ridge regression** (optimize λ) 

Try and use classes and inherited methods (for example to plot histograms, to plot \hat{y} versus y , to generate the DataFrame with the results, etc), exploit the code you already wrote to implement the gradient algorithm.

What you have to do [2]

- Note that you wrote the algorithms assuming **Ndarrays** as inputs. Therefore, you must pass to your methods `X_tr_norm.values` and `y_tr_norm.values`
- Generate at least the following plots for regression of total_UPDRS:
 - the **estimated regressand versus the true regressand** for the test dataset (note: the de-normalized values, not the normalized values)
 - the **histograms** (de-normalized), of the estimation error for training validation and test datasets
 - the **values** of **w**
 - for **ridge regression**, plot the mean square error $\mathbb{E}\{[\hat{y}(n) - y(n)]^2\}$ for the training dataset and the validation dataset as a function of λ , so that you can show how you set the value of λ .
- Fill in a **table** (DataFrame) with the values of the measured mean, standard deviation, mean square value for the regression errors (de-normalized) and R^2 for training, validation and test datasets, and draw your conclusions.

What you have to do [3]

- **VERY IMPORTANT:** in the applications analyzed in this course **execution speed** is not an issue (we are not dealing with big data); algorithm **complexity** is not an issue (think of the cost of a "simple" ultrasound machine, around 60-100 kEuros, and the cost of a "good" server with CUDAs etc, let's say 5-10 KEuros). The true issue is **reliability** and goodness of the result.
- Include your results and comments in the Latex report that you can download from the class material, where only LLS results have been included.
- **DEADLINE: 14 days from now at 11.59 PM, before next lab.**
- Once you are ready with the lab, generate a **pdf** file with your report; the filename **must** be `yoursurname_yourmatricola_report_1.pdf`.
- Generate a folder with your Python code (with subfolders if you used subfolders); the folder name **must** be `yoursurname_yourmatricola_report_1_code`.

What you have to do [4]

- The main python file must be `main.py`. Input data must be inside folder `yoursurname_yourmatricola_report_N_code/data`. If you write a python library that you call in your `main.py` script, then the library must be in folder `yoursurname_yourmatricola_report_N_code/sub`.
- Folder `yoursurname_yourmatricola_report_1_code` must include all the required files, so that it is possible to run the script from a shell, i.e. using `$ python3 main.py`.
- Zip together the folder and the report pdf file and generate file `yoursurname_yourmatricola_report_1.zip`
Compression systems other than zip will not be accepted.
- If you do not exactly follow these instruction, the grade of the report is 0.

Table of Contents

1 Parkinson's disease

2 Laboratory # 1

- Prepare and analyze the data
- Perform regression

3 Structure of the report

The report [1]

- The report main requirements (name of files, upload, font, etc) are written in the previous slides. Read them again.
- **Spell checking** shall be used in order to remove most of the typos. Careful reading is suggested, to remove the remaining typos. If you are not sure of a word, search for it on the web/dictionary.
- Never include **figures** that are not referred to in the text.
- Never write sentences like “In the figure below” or similar; open a technical book written by an English or American author, search how the author refers to figures, copy the same sentences. Give **numbers to figures** and refer to them in the text through the figure number. The figure number must be in the caption below the figure (not in the title).
- Never use “I” in the report (it is not professional).
- The plots shall have the **xlabels** and **ylabels**, the **grids**, the **title**. If you want to compare two plots, use the **same ranges** for the x and y axes.

40/41

- The report must be **self-consistent**, it must be understood by a professor that teaches regression in another university, maybe using a different notation (you must specify the meaning of each symbol you use in the report). You can refer to books, papers, documents available in the web (correctly cite the documents). You cannot refer to the slides of the class, because they are not public.
- The Python scripts shall include **comments** in English (not in Chinese, Spanish or Italian), to help understand the flow of instructions
- The report is practically written (download it from the class material), and you just have to add the missing figures and your final comments. You can install Latex on your PC or use the online Latex editor **Overleaf**.