

# Report 2 : Forecasting and Analysis of the Polish Power System Time Series

Piero Paialunga and Simone Chierichini

October 2020

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Data and Method Description</b>	<b>2</b>
2.1	Data overview . . . . .	2
2.2	Data preprocessing . . . . .	2
2.3	Fourier Transform . . . . .	3
2.3.1	Uniform varying threshold . . . . .	5
2.3.2	Maximum related threshold . . . . .	5
2.3.3	Time dependent Fourier Transform . . . . .	11
2.4	Wavelet . . . . .	12
2.5	Arima and Sarima Processes . . . . .	14
2.6	Deconvolution . . . . .	20
<b>3</b>	<b>Results and Conclusion</b>	<b>23</b>
<b>4</b>	<b>Appendix</b>	<b>25</b>
4.1	datapreprocessing.py . . . . .	25
4.2	fouriermtehod.ipynb . . . . .	27
4.3	fouriermethod2.ipynb . . . . .	35
4.4	waveletfiltering.ipynb . . . . .	38
4.5	SARIMA.ipynb . . . . .	43
4.6	Deconvolution.ipynb . . . . .	53

### **Abstract**

The Load of the Polish Power System Dataset has been analyzed with various methods and a forecast have been made out of this analysis. Two Fourier methods (time independent and time dependent) have been applied to extract information about the periodical behavior of the signal. The wavelet has been used to clean the signal from the noise, and the cleaned signal has been used to apply prediction with ARIMA and SARIMA processes and deconvolution methods. The best forecast gave a RMSE equal to the 9.7% of the maximum value.

# Chapter 1

## Introduction

An appealing kind of dataset that is worth studying in order to obtain a predictive ability about nature are the Time Series. The one studied in this report is about **the load of the Polish Power System**.

One of the most powerful tool of Data Analysis of a Time Series consists on using the Fourier transform on a periodic data. In fact, the Fourier Theorem states that a periodic function can be decomposed in sines and cosines terms. In this sense it is possible to perform a variable space from the original one (the  $x$  space) and analyze the behavior of the function with respect of the  $u$  frequency, thus going from  $f(x)$  to  $F(u)$ . In particular, the absolute value of this function computed at a specific frequency  $u^*$  furnishes the amplitude that  $u^*$  has in the spectrum. By the distinction of the frequencies that could be considered as a form of "noise" and the frequencies of the "proper" signal, and by the accurate treatment of the non-stationary nature of the signal, it is possible to use the information of the frequency spectrum to filter the signal and gain a forecast ability.

Another important tool to clean the signal consists in the use of the Wavelet transform. As the Fourier Analysis, the wavelet trans-

form compute the projection of the original signal on an orthogonal basis of a function called wavelet. By using this wavelet at different scales, it is possible to detect the noise scale as it has frequencies that can be assumed to be above the typical band of frequency of the signal.

A predictive method that is more powerful of the Fourier Analysis consists in the use of ARIMA and SARIMA processes. By the use of this processes it is possible to construct an approximated "auto-regressive" and "moving average" model of the original signal. In particular, the main difference between this method and the other ones that has been used is the ability to use a different numbers of values to look at both in the auto-regressive and the moving-average processes to obtain a prediction. Moreover, in the SARIMA processes, the non stationary nature of the signal could be treated with an extra-attention as the method considers the seasonal elements of the data.

The last method that has been used to forecast and analyze the signal is the deconvolution. This approach considers the signal as the convolution of one single shape known as kernel with a series of pulses. A refinement of this method has been applied considering three different kernels.

# Chapter 2

## Data and Method Description

### 2.1 Data overview

The analyzed data is the **load of Polish Power System time series**. The dataset is extremely simple as it consists in a time report of all the loads in the Polish Power System. The columns of the dataset are:

- **Day**, Calendar Day in the following format ('YYYY/MM/DD')
- **Hour**, Hour of the end of the detection
- **Minute**, Minute of the end of the detection
- **Load**, Load (MW) during that time interval

Measurements are taken each 15 minutes and the Load column reports, in the  $(n)$ -th row, the load between the  $(n - 1)$ th and the  $(n)$ th row (i.e. in 15 minutes of time). The **site** that has been used to extract the dataset uploads real time information about the Polish Load. Due to computational limits, our specific dataset is limited from 2008 (2008-01-01) to 2016 (2016-12-31).

### 2.2 Data preprocessing

Even if the initial format of the dataset is immediate to read and to understand, it is convenient to convert the temporal scale in a way that the temporal continuity is visible and appreciable. To do so, a 5-th column has been added to the dataset, consisting in a temporal line from 900 seconds (15 minutes) and increasing by the same factor (900 seconds) for each row: (900,1800,2700,...). Using this temporal scale, an effective visualization of the load can be obtained:

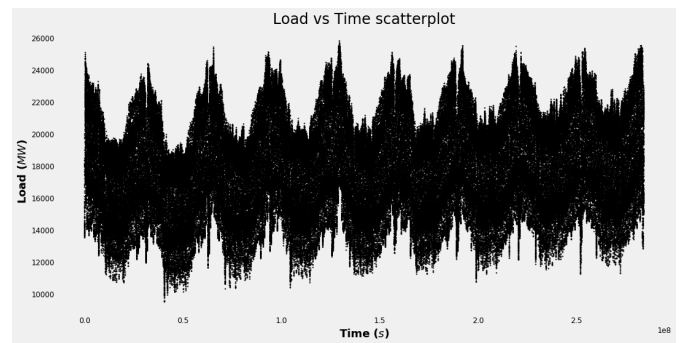


Figure 2.1: **Load(MW)vs Time(s)** plot.

The analysis we want to perform about the dataset is based on the Fourier Transform. As it has already been said, the goal of this analysis is to detect the frequencies that

characterize the phenomenon of interest. In this sense, the mean value and the global temporal trend of the signal are not interesting and tend to disturb the target analysis. As it can be appreciated from Figure 2.1 the mean value of the signal is not 0 and a global linear ascending trend can be easily identify. The first step is thus "de-trending" the original signal, and the final result is expressed in Figure 2.2:

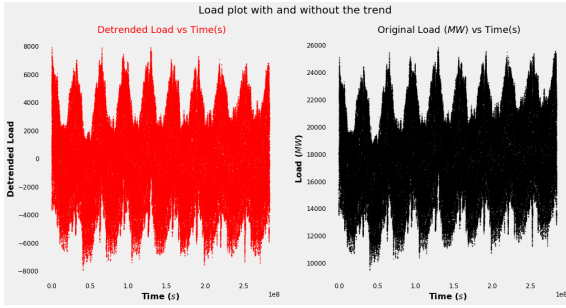


Figure 2.2: **De-trended Load vs Time(s) plot** to the left and **Load (MW) vs Time (s) plot** to the right.

The mean value of the red signal is extremely low ( $\approx 10^{-11}$ ). To make it even lower, the mean value is subtracted from the red signal, thus obtaining a signal with mean value=0 (numerically the mean value is the lowest possible:  $\approx 10^{-13}$ ). From now on, the "detrended" mean=0 Load will be simply intended as load. As it has already been said in chapter 1, the Fourier Analysis is made for periodical signal. The first important thing to notice is if the first point ( $i = 0$ ) and the last point ( $i = N$ ) of the dataset has the same Load quantity. As there is in fact a notable difference between  $Load_0$  and  $Load_N$  ( $|Load_0 - Load_N| = 2267.67$  that is almost the 30% of the maximum load), a window function

needs to be applied to smooth the signal and make the endpoint of the signal meet. The first standard windowing function that has been applied to do so is the Hanning function [6] and the result of the product between the signal and the hanning function is shown in Figure 2.3

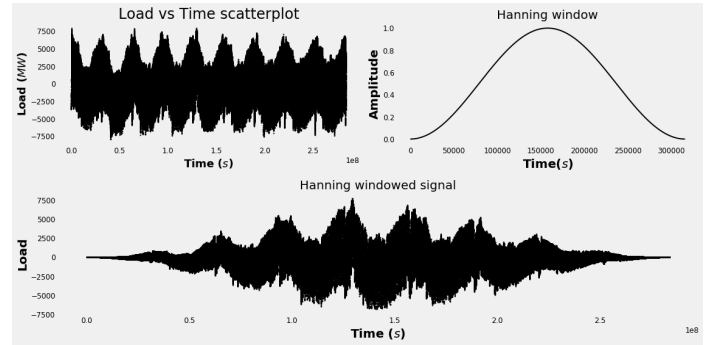


Figure 2.3: **Detended Signal Load vs Time scatterplot (up-left)**, **Hanning window (up-right)** and their product **Hanning windowed signal**. As it is possible to see, the windowed signal has the same value on its border.

## 2.3 Fourier Transform

As it has already been said in the introduction, the signals that can be analyzed using Fourier Transform are by definition stationary. This condition appears to be a forced one when the signal is extracted by some real world data like the one that it has been considered. For example the behavior of the Load of a civilized country during the Christmas Holidays is not equal to the behavior during the rest of the year. For this reason, the blind application of the (Fast) Fourier Transform may seem a weak ap-

proach. Nonetheless some basic time period could still be verified from the Fourier transform of the Load. In fact by just looking at the plot Amplitude vs Period ( $h$ ) it is possible to notice some natural order periods like:

- 12h (the lenght of a day divided by 2)
- 24h (the lenght of a day)
- $\approx 33h$  (1 day + 8h )
- 168h (7 days, a week)

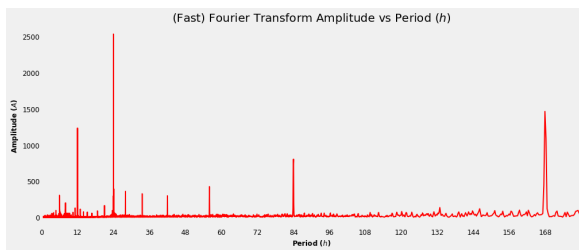


Figure 2.4: **Amplitude vs Period ( $h$ ) plot.** As it is possible to notice some relevant peaks appear in some specific time ticks (12h,24h,33h,168h).

This specific periods seem to be high correlated with the working hours and the natural duration of a day. In fact, it is possible to assume that the 12h periodicity could be related to the day and the night hours of a day. 24h is the duration of a day, so it is still a natural period. 33h is the sum of 24h (the lenght of the day) + 8h (the mean working hours per day) and it could be related to some working periodicity. Even the week and the middle week periods are stressed by the 84h and 168h peaks. Thus, even if the signal is not stationary and some extra work needs to be

done in order to interpret the signal correctly and thus to be able to predict the following year, **it is still important to analyze this periodicity and understand if it is possible to reconstruct the signal starting with few frequencies values.** As the numpy algorithm that has been used is a robust one, transforming the original signal using Fourier transform and then inverse transforming the Fourier transform, the original signal is accurately reproduced. The mean absolute difference between the reconstructed signal and the original one is in fact almost 0 ( $\approx 10^{-13}$ ) and the signal are almost unrecognizable from each other as it is possible to see from Figure 2.5:

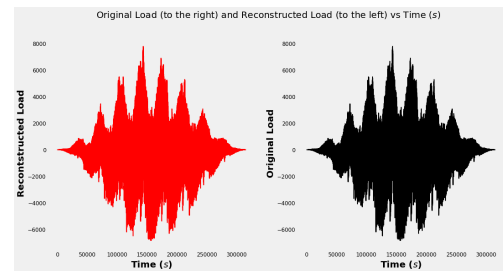


Figure 2.5: **Reconstructed (to the left) and original signal (to the right) comparison.** As it is possible to see, the signal are unrecognizable from each other, as the numpy FFT algorithm is robust.

After that the robustness of the algorithm has been tested, it is interesting to analyze which frequencies are really informative in terms of the reconstruction of the original signal. In particular, a threshold has been applied to the frequency spectrum, setting to 0 the values that are greater than that threshold. After this operation the inverse fourier transform has been applied, thus obtaining a reconstructed signal. The root

mean squared error (RMSE) has been computed between the original signal and the reconstructed one.

### 2.3.1 Uniform varying threshold

Varying the threshold ( $k$ ) in an uniform manner ( $k \in \{0, 2475\}$ ) the RMSE becomes larger as one could expect. To deeply understand the meaning of this RMSE and how the information is filtered out by the threshold value, some  $k$  values filtered plot has been shown in Figure 2.6

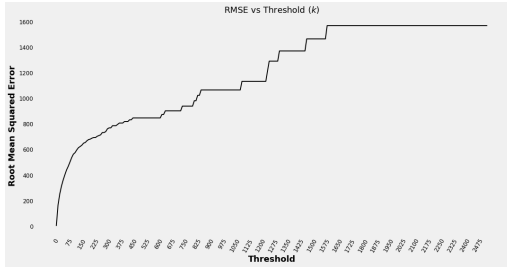


Figure 2.6: **RMSE vs  $k$  plot.** As the threshold ( $Th_k = k$ ) increases the RMSE increases too, thus highlighting the loss of information that it is verified during the reconstruction. The highest growth rate is in the first zone of the graph, where  $k \in \{0, 300\}$

As Figure 2.7 suggests, as the  $k$  value increases, only the frequencies with highest amplitude are not filtered out. In fact, when  $k$  becomes bigger than a certain value, only the highest peak, that is the one related to the 24h period, "survives".

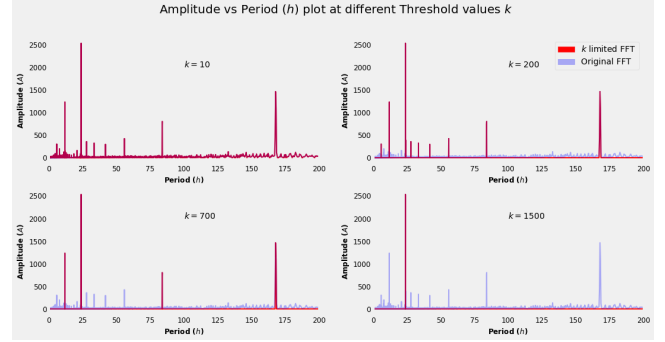


Figure 2.7: **FFT of the original signal and  $k$ -filtered FFT vs Period ( $h$ ) plot.** As the threshold ( $Th_k = k$ ) increases only the frequencies with the highest amplitude are not filtered out.

### 2.3.2 Maximum related threshold

The analysis suggests that, to understand the relevance of the sinusoidal components, it could be more convenient to vary the threshold  $k$  scale with respect to the maximum amplitude value. As it is possible to see from Figure 2.8, if the frequencies with amplitude under the 30% of the maximum are set to 0, a big portion of the noise is removed, and the waving behavior of the signal is still visible and cleaned. As the threshold becomes higher, only the 24h period survives. Its frequency, as it is possible to see from Figure 2.4, is in fact so high to be indistinguishable from a full colored square, as the last subplot in Figure 2.8 highlights. A clearer version of Figure 2.6 has been reproduced, computing the RMSE at the varying of the  $k$  with respect to the maximum amplitude value (Figure 2.9). Even if, in general the RMSE furnishes an efficient comparison method be-



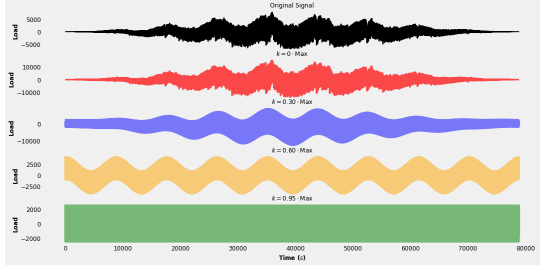


Figure 2.8: **Reconstructed Load vs Time at various  $k$  threshold value.** If  $k = 0.3 \cdot \text{Max}$ , the original signal can be reconstructed with a discrete level of accuracy.

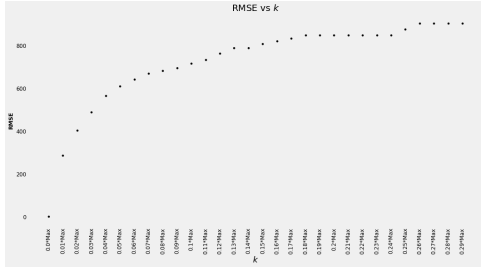


Figure 2.9: **RMSE vs  $k$  plot.** This plot is analog to the one reproduced in Figure 2.6, but it is simpler to read as it is related to the maximum amplitude value.

tween a real signal and a reconstructed one (low RMSE implies good correspondence between the two), it is important to understand if the RMSE value is dominated by the background noise of the original signal. In fact, while reconstructing the signal with the threshold limited fourier transform, the noisy behavior of the original Load is set to zero and a more clean signal is reconstructed (example in Figure 2.10). This process summarize the purpose of this part of the report, as it cleans the original data and set to zero the not relevant information,

but increases the RMSE.

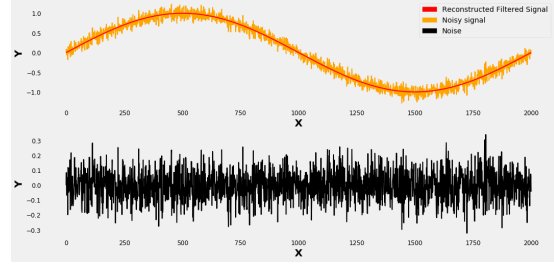


Figure 2.10: **Example of the noise phenomenon.** Filtering the fourier spectrum with a threshold, the noisy sinus wave loses its background noise. As it is possible to see, the noise is not related to the original signal.

To explore this phenomenon, it is important to understand how much the error is related or not to the original signal. The plot of the correlation value with respect to  $k$  predictably highlights that the correlation increases with the  $k$  threshold. The  $k$  value can thus be chosen with respect to this correlation coefficient. It is important to highlight the trade off that needs to be obtained here:

- An high threshold value assures that a rigid selection of the frequencies has been applied and the noise is deleted
- A low correlation coefficient assures that the residual signal is not correlated with the original signal and it can be properly assume as noise.

As it is possible to see from Figure 2.11, as the correlation decreases the threshold decreases too and vice versa. It is thus important, in order to delete only the signal, to keep the threshold as high as possible while the correlation doesn't increase over

a certain value. As it is possible to appreciate from Figure 2.11, the correlation between the error and the signal is not 0 even when the threshold is set to 0, thus highlighting that even if the error that is generated by the Fourier Transform is extremely low it is still slightly correlated with the signal ( $C = 4.3\%$  correlation). This

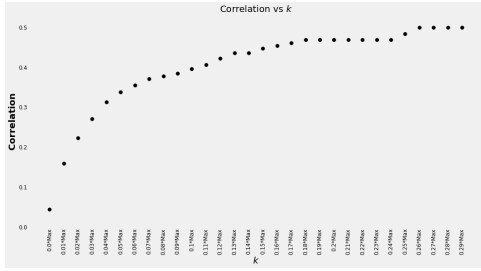


Figure 2.11: **Correlation vs  $k$  scatter plot.** As it is possible to see from the plot, as the  $k$  threshold value increases, the error becomes more correlated with the original signal.

consideration, together with the correlation instantly increasing in the first 3 thresholds ( $C_{k=0.05*Max} = 16\%$ ,  $C_{k=0.02*Max} = 23\%$ ,  $C_{k=0.03*Max} = 27\%$ ), furnishes a relevant warning to further decrease the  $k$  lower bound. As it is possible to see from Figure 2.12 this detailed observation permits to choose a less correlated signal reconstruction by setting the optimal threshold as the highest one with correspondent correlation lower than 10% ( $k_{opt} = 0.004 \cdot Max$ ).

The optimal signal reconstruction is thus obtained by inverse transforming the filtered power spectrum and the results are shown in Figure 2.13.

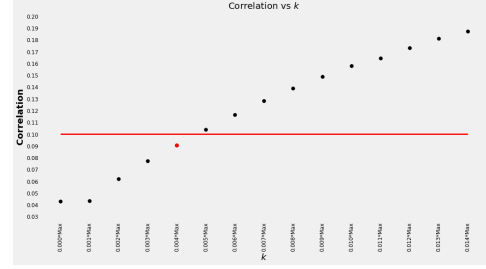


Figure 2.12: **Correlation vs  $k$  scatter plot.** As it is possible to see from the plot, as the  $k$  threshold value increases, the error becomes more correlated with the original signal. The optimal threshold

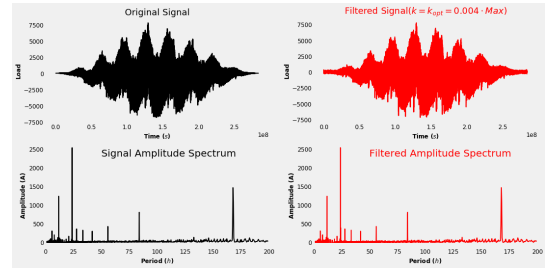


Figure 2.13: **Original (Up-left) and Filtered (Up-right) Signal with correspondent amplitude spectra (Down-left and Down-right).** As it is possible to see, the signals appear similar but does have some differences in the highest frequencies.

A double check of the analysis has been done to verify the distribution plot of the difference between the reconstructed signal and the original one. In fact in order to be considered white gaussian noise, it is expected to be fitted with low  $\chi^2$  value to a gaussian [5]. As it is possible to see from Figure 2.14 the error appears to have a gaussian shape. It is thus interesting

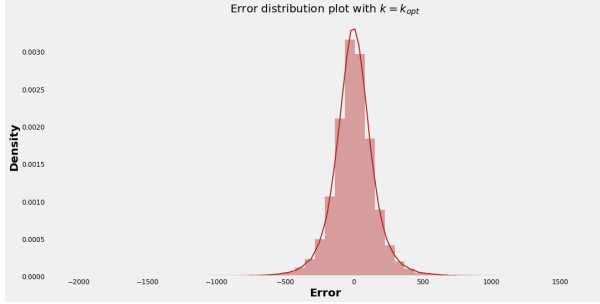


Figure 2.14: **Error Distribution plot with  $k = k_{opt}$ .** As it is possible to see, the distribution has a gaussian shape in its core.

to perform the  $\chi^2$  test on a gaussian distribution, as it has been shown in Figure 2.15. Unsurprisingly, the  $\chi^2$  value is low  $\chi^2 = 6.27 \times 10^{-2}$  while the p value [7] is extremely high  $p \approx 1.0$ , thus highlighting the excellent correlation between the gaussian fit and the original data. Even if this

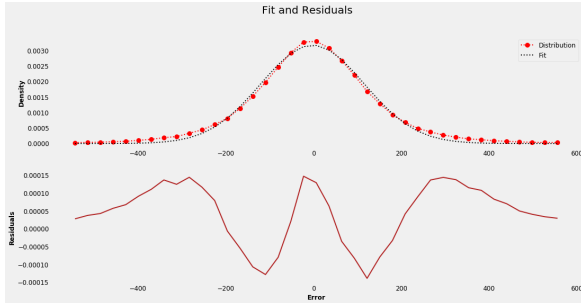


Figure 2.15: **Error Distribution plot with  $k = k_{opt}$  and gaussian fit (Up), Residuals plot (Down).** As it is possible to see, the gaussian fit with high accuracy the distribution plot. The residuals ( $R$ ) are at a significant lower scale with respect to the distribution values ( $D$ ):  $\frac{D_{max}}{R_{max}} = 22.360$

method shows promising statistical properties, **the important test that the recon-**

**structed signal has to pass is the prediction of the new data.** As it is important to highlight as a background clarification, **the prediction that can be done is about the oscillatory behavior of the core of the signal.** In fact, the linear trend (the growth of the signal) and the mean value are both subtracted from the original signal in order to process it correctly with the Fourier analysis. Moreover, two important hypothesis that has been made are the following:

- **The nature of the signal is periodical:** all the events can be accurately modeled as sines and cosines terms.
- **The signal is stationary:** the frequencies of the fourier spectrum are time invariant.

While the first assumption is a crucial one when the Fourier transform is applied, the second one will be relaxed with the application of the **spectrogram method**. In order to perform the prediction test, the dataset has been split in two sets. Borrowing in a not rigorous manner the Machine Learning notation they have been defined as following:

- **The training set**, where the algorithm that has been described in the previous page has been applied and the prediction has been performed.
- **The test set**, that is the set of data that has been used for testing the prediction.

In order to apply the algorithm in the most efficient way as possible, the training set has been obtained by extracting the first 8 years

of the dataset, while the remaining year (2016) has been used as test set. The previous described algorithm has been applied in the training set, but as it is necessary to predict the real values no smoothing has been applied to the original signal. In order to keep both RMSE and Correlation values under control ( $RMSE \leq 300$ ,  $C \leq 10\%$ ), the optimal threshold as been reduced :  $k_{opt} = 0.004 \cdot Max$ . The plot of both RMSE and Correlation values are reported on Figure 2.16 while the spectrum with the  $k_{opt}$  threshold has been reported in Figure 2.17.

The spectrum in Figure 2.17 furnishes a

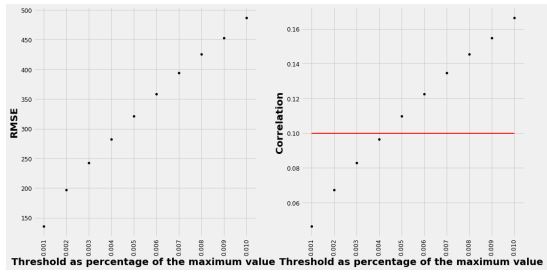


Figure 2.16: **RMSE and Correlation vs  $k$  plot**). In order to keep  $RMSE \leq 300$  and  $C \leq 10\%$ , the  $k_{opt}$  value has been chosen to be  $k_{opt} = 0.004 \cdot Max$ .

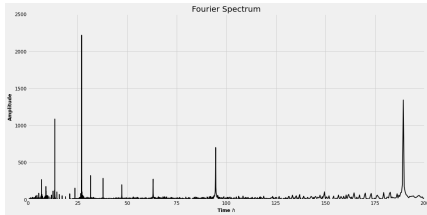


Figure 2.17: **RMSE and Correlation vs  $k$  plot**). In order to keep  $RMSE \leq 300$  and  $C \leq 10\%$ , the  $k_{opt}$  value has been chosen to be  $k_{opt} = 0.004 \cdot Max$ .

set of sines and cosines that can be defined

by their frequency  $\omega$  and their complex correspondent value ( $F(\omega) = F_R(\omega) + jF_I(\omega)$ ). Given a specific frequency  $\omega$  and a specific time  $t$  the correspondent waving function is given by:

$$x(t, \omega) = \frac{2}{N} (F_R(\omega) \cos(2\pi\omega t) + F_I(\omega) \sin(2\pi\omega t))$$

In general, for all the frequencies that has not been filtered out ( $\omega \in \Omega$ ) we have:

$$x(t) = \sum_{\omega \in \Omega} \frac{2}{N} (F_R(\omega) \cos(2\pi\omega t) + F_I(\omega) \sin(2\pi\omega t))$$

By applying this rule with  $t \in T_{train}$  with  $T_{train}$  that is defined by all the time steps ( $s$ ) of the training data, the reconstructed signal is simply the inverse (filtered) fourier transform of the original one. Nonetheless, when  $t \notin T_{train}$  but  $t \in T_{test}$  the signal is extended over its inverse fourier definition by using the above rule. In this sense, as the algorithm "didn't see" the test set, a prediction has been made. As the reconstruction has been done on the training set frequencies, the difference between the original signal and the reconstructed one is at its minimum values in the training set and it increases on the test set. Indeed, it is possible to see that the increase of the difference between the predicted signal and the real one doubles in the test set.

In particular the RMSE in the training set is:

$$RMSE_{Train} = 278.32$$

While the RMSE in the test set is:

$$RMSE_{Test} = 2624.89$$

This result comes with little surprise. In fact no temporal information has been given

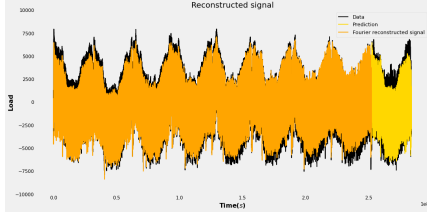


Figure 2.18: **Original Signal, Reconstructed Signal and Predicted Signal vs Time(s)**

about the frequencies and the signal is considered to be stationary. A close look of the original signal and the reconstructed one in the test set highlights that the errors of the prediction become significantly higher with respect to the time.

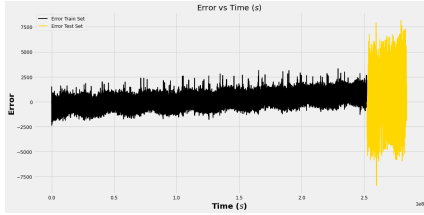


Figure 2.19: **Error vs Time**. The error becomes consistently higher when the test set is explored. In both training set and test set the signal appears to be slightly correlated to the error.

Moreover **the error increases with respect to the time, thus highlighting the weaknesses of the stationarity assumption**. As it has already been told, the RMSE could be a inappropriate metric. A deeper analysis has been made to see whether or not high correlation could be found between the original signal and the error. Unfortunately the signal and the error are highly anticorrelated ( $C_{data,error} \approx$

$-0.6$ ) and the  $P$ -value test highly rejects the gaussian hypothesis ( $P \approx 0$ ), thus confirming the failure of the method. The

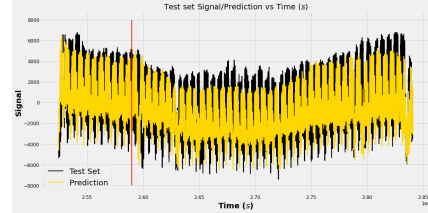


Figure 2.20: **Signal/Prediction vs Time** ). It is possible to see that the prediction becomes worse at high time scale, while it seems to be acceptable for the first entries.

RMSE has been computed reducing the time range and it can be seen that the first prediction have the lowest error values. In particular the prediction for the first 30 days is both the most precise one in terms of RMSE and the less error-related. As the

	$T_f/n(s)$	RMSE	$\Delta T(s)$	$ C_{data,error} $	$\Delta Days$
0	254188800	2569.99	1727100	0.64	20
1	255052800	2439.43	2591100	0.61	30
2	257644800	2458.33	5183100	0.62	60
3	260236800	2780.33	7775100	0.65	90
4	262828800	2686.71	10367100	0.64	120

Figure 2.21: **Summary of the RMSE at different time ranges**). As it is possible to see, the prediction gets better for lower values of time step. In particular the best RMSE and correlation (the lowest) are verified for a 30 days time step.

(best) RMSE is the 33% of the maximum value of the signal, the method has to be considered as failed.

### 2.3.3 Time dependent Fourier Transform

The algorithm that has been presented in Chapter 2.3.2 had the important defect of being completely time independent. In fact the signal has been interpreted as a stationary one, thus obtaining a weak reconstruction and an even weaker prediction. The following approach [4] is used to consider the non stationary nature of the dataset, and it is based on the following division of the original data:

- **The training set**, that contains the signal from 2008 to 2015
- **The validation set**, that contains the signal data of the year 2015
- **The test set** that contains the signal data of the year 2016

The training set has been divided in 5 annual periods. **For each one of those, the fourier transform has been applied. The mean of all these transform has been applied, thus obtaining a mean fourier transform of the first 5 years of the dataset.**

The validation set has been used to apply the algorithm that has been described in chapter 2.3.2 to the mean fourier transform that has been obtained from the training set. **The big difference between this approach and the one described in chapter 2.3.2 is that the RMSE and the correlation are computed with respect to a portion of the dataset that the algorithm "does not know": the algorithm is not been trained on this portion of the dataset.** To consider

this difference, the RMSE threshold and the correlation one are less strict than the one consider in chapter 2.3.2 ( $RMSE_{max} < 2000, |C_{min}| < 0.82$ ).

The threshold value with the lowest  $C_{min}$  and within the  $RMSE_{max}$  value has been considered as the **optimal threshold**. **The mean fourier transform that has been computed in the training set has thus been filtered with the optimal threshold, and the result has been compared with the test set signal.** These has been summed together and divided by 8, thus obtaining a mean fourier transform that has been adopted to predict the values of the 9th year.

	RMSE	Days	$C_{data,error}$
0	1693.717665	10	-0.499780
1	1653.486625	20	-0.584994
2	1625.299205	30	-0.551923
3	1750.071345	40	-0.572060
4	1705.124815	50	-0.576772
5	1742.991733	60	-0.582522
6	1739.389256	70	-0.587244
7	1757.974530	80	-0.591003
8	1945.728359	90	-0.620888

Figure 2.22: **Summary of the RMSE after a certain number of days of observation.** Even if the RMSE is in general still remarkably high the values are lower than the ones that have been obtained with the previous method. Even the correlation values are lower than the ones of the previous method.

This method remarkably outclass the previous one in terms of RMSEs, obtaining lower RMSEs even for larger period of time. The method is better in terms of correlation between the original signal and the error too. In fact data are almost 10% less correlated



with the error with respect to the previous method.

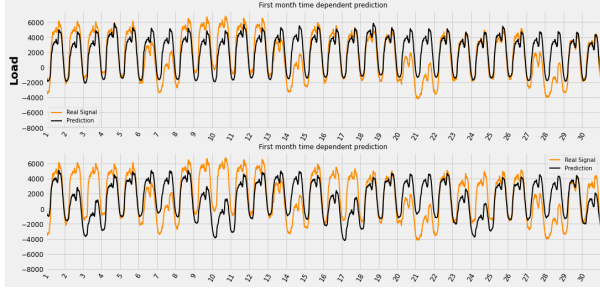


Figure 2.23: First month prediction using Time dependent Fourier Transform method).

The best RMSE assumes a value that is the 21% of the maximum value thus obtaining a more accurate method with respect of the (filtered) global fourier transform one.

## 2.4 Wavelet

A wide part of the algorithm in section 2.3.2 and 2.3.1 is based on cleaning the original signal from the (presumed) noise. **In general, an important step in order to have an accurate prediction of a time-series is based on cleaning the original data.** So far, this cleaning process has been done following the step indicated above:

- Analyzing the fourier spectrum
- Applying a threshold to it
- Selecting the upper bound of the threshold based on the RMSE
- Selecting the optimal threshold as the highest one with the correlation value as close as possible to 10%

This methodology has not been proved helpful even if it is based on the reasonable assumption that the residual signal has to be not correlated with the signal to be considered as noise. A different approach to clean the signal is based on using the so called **wavelet transform**. The **wavelet transform** is in some way similar to the Fourier one as it **computes the product between the original signal and a basis function** ( $\psi$ ). Nonetheless this basis function is stretched and contracted in relation with a parameter called **scale** ( $s$ ). Thus, at a fixed scale  $s$ , the **basis function will "slide" on the original signal time steps outputting the projection between the signal  $f$  at that specific time and the wavelet computed at that specific scale and that specific time.** The sum of all this time contribute will give the wavelet coefficient at a specific time ( $u$ ) and at a specific scale ( $s$ ):

$$W_f(s, u) = \int_{-\infty}^{+\infty} f(t) \frac{1}{\sqrt{s}} \psi^*\left(\frac{t-u}{s}\right) dt$$

The power of this method relies on the ability of discern and decompose the original signal at different scale. Indeed, when the scale decreases, higher frequencies of the original signal are detected and vice versa. By using low value of scale, the wavelet method is able to discern the highest frequencies of the signal, that are the one that can be considered to belong to the white noise frequency range. In particular, the method that has been used to implement this idea is the **discrete wavelet transform** [2]. This method is based on the decomposition of the signal in **detail** and **approximation coefficients** on various level. The first level is obtained by using

the wavelet with the lowest possible scale: i.e. the highest frequency ( $\nu_{s_{min}}$ ) with respect to the highest frequency of the signal ( $\nu_{s_{max}}$ ) according to the Nyquist Shannon theorem ( $\frac{\nu_{s_{max}}}{2} = \nu_{s_{min}}$ ) [1]. In particular, **the detail coefficients of the first level are represented by  $W_f(s_{min}, u)$  and the approximation coefficients are represented by the residual between the original signal and the detail coefficients**. Both the detail and the approximation coefficients, according to the Nyquist Shannon theorem can be downsampled by a factor  $2^{n_{level}}$  (2 for the first level, 4 for the second level, 8 for the third level). **As it can be assumed that the denoised signal will have specific band limited frequency range, the highest frequencies of the studied signal can be considered as the white noise frequencies [3].** That means that it is possible to detect the noise signal in the lowest level of the discrete wavelet transform detail coefficients.

The first level presents a not negligible excess of kurtosis, thus discouraging from looking at other levels, as too much relevant information about the signal would be lost. Moreover, in order to select the part of the detail coefficient that can't be considered as "noise", a certain portion of the detail coefficient has been set to 0. This portion has been chosen with respect of a threshold value  $\theta_k$  that is a function of  $k$ , that is a real number between 0 and 13.25 ( $k \in [0, 13.25]$ ) and the fitted sigma value from the gaussian fit:

$$\theta_k = k\sigma$$

This value has to be intended as a threshold in the following sense: **all the values of**

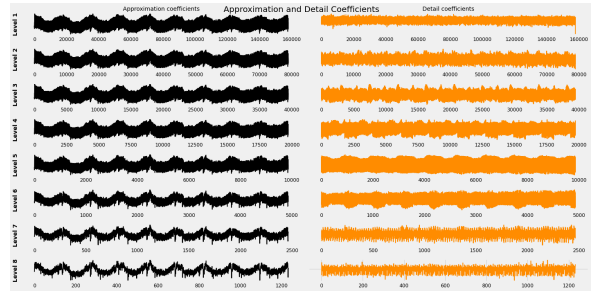


Figure 2.24: **Approximation and Detail Coefficients**). As it is possible to see, the periodical behavior of the signal does not present itself in the first detail coefficients, while it starts appear when the  $n_{level}$  increases. At the same time the approximation coefficients are really similar to the original signal for the first level, and they loose accuracy while the level increases.

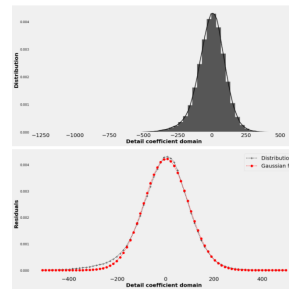


Figure 2.25: **Detail Coefficient distribution**). As it is possible to see, the distribution is almost gaussian, but presents an excess of kurtosis on its tail.

**the details coefficient that are found between  $-\theta_k$  and  $+\theta_k$  are set to 0.** The  $k$  upper bound is due to the fact that if the threshold is chosen to be that high, the entire first detail coefficient is set to 0. While  $k$ , (and  $\theta_k$ ) increases, so does the RMSE. Indeed if  $\theta_0 = 0$  the approximation and detail coefficient are naively summed, as no value



of the detail coefficient is set to 0. Nevertheless, if the RMSE increments are due to the noise they are harmless as they don't imply a loss of information about the original signal. **In order to detect whether or not the filtering is actually cutting only the noise out, the correlation coefficient between the original signal and the difference between the latter and the reconstructed one (intended as the sum of the approximation coefficient and the filtered detail coefficient both of the first level) has been computed. The best threshold has been chosen to be the one with the lowest correlation.** In particular it has been proven to be 1:

$$\theta_{opt} = \sigma$$

. This optimal threshold permitted to have the following error-signal correlation

$$C = 0.2\%$$

The effect of the threshold on the original detail coefficient domain has been shown in Figure 2.26. This optimal threshold has been used to construct a reconstructed signal, that is as less as possible influenced by the noise. As it is possible to see from Figure 2.28, it is really difficult to spot the differences between the original signal and the reconstructed one in general terms. On the other hand the smoothing effect can be appreciated at lower scales, where the original signal and the reconstructed one are still similar, but the reconstructed signal follows a smoother line as it is not 'disturbed' by the noise.

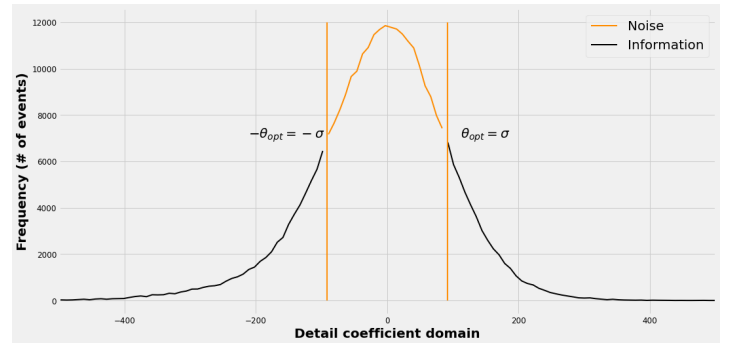


Figure 2.26: **Detail Coefficient distribution filter**). In orange it is possible to see the range of points that has been filtered out from the original detail coefficient signal. In black, it is possible to see the remaining part, that has been summed to the approximation coefficient signal to obtain the reconstructed filtered signal.

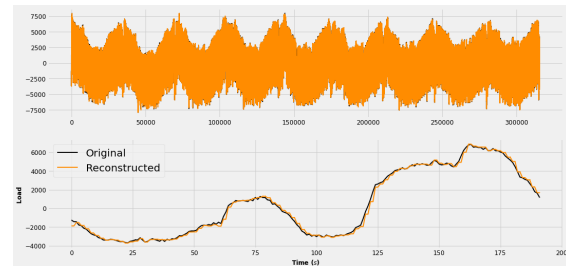


Figure 2.27: **Reconstructed Signal (Orange) and Original Signal (Black) at different time scales**. While the signals appear to be indistinguishable in the larger time scales, the effect of the de-noising wavelet process appears to be visible at lower scales.

## 2.5 Arima and Sarima Processes

In this report we wanted to test the ability of a class of models widely used

in forecasting, the Autoregressive–moving–average (ARMA) models. The basic idea is to use these models to describe the dataset and extrapolate a forecast.

The main goal in time series analysis is identifying an appropriate stochastic process that has trajectories that adapt to the data, in order to then be able to formulate forecasts. An important class of stochastic processes that allows us to uniquely identify the process and obtain a consistent estimate is represented by stationary processes. In particular, if what we observe is interpreted as a finite realization of a stochastic process that enjoys particular properties, then it is possible to find a single model suitable to represent the temporal evolution of the phenomenon under study. Intuitively, a stochastic process is stationary if its probabilistic structure (average value, variance, etc.) is invariant over time. Generally a stochastic process  $(X_t)_{t \in \mathbb{Z}}$  is said to be *stationary* or *weakly stationary* if it satisfies the following conditions:

- 1.  $E[|X_t|^2] < \infty$
- 2.  $E[X_t] = m \quad \forall t \in \mathbb{Z}$
- 3.  $\gamma_X(r, s) = \gamma_X(r + t, s + t) \quad \forall r, s, t \in \mathbb{Z}$

Considering a white noise with zero mean and variance  $\sigma^2$  identified as  $WN(0, \sigma^2)$  and introducing the delay operator  $B$  defined as

$$BX_t = X_{t-1} \quad (2.1)$$

we can begin to define some of the fundamental characteristics of these processes. An **Autoregressive process** of order  $p$ , identified as AR ( $p$ ) can be written as:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \dots + \phi_p x_{t-p} + w_t \quad (2.2)$$

formally

$$W_t = \phi_0 X_t - \sum_{i=1}^p \phi_i X_{t-i} = \phi(B)X_t \quad (2.3)$$

In particular,  $\phi(B)$  is a polynomial of degree  $p$ , given by

$$\phi(z) = \phi_0 - \phi_1 z - \dots - \phi_p z^p$$

Where  $x_t$  is stationary and  $\phi_1, \phi_2, \dots, \phi_p$  are constant ( $\phi_p \neq 0$ ). It is generally assumed that  $w_t$  is a Gaussian white noise with zero mean and variance  $\sigma_{w_t}^2$ .

In a very similar way it is possible to define a **Moving Average** process of order  $q$ , with  $q \in \mathbb{N}$ , briefly MA( $q$ ), if it is of the form:

$$\begin{aligned} X_t &= \theta_0 + \theta_1 W_t + \theta_2 W_{t-1} + \dots + \theta_q W_{t-q} = \\ &= \sum_{i=1}^q \theta_i W_{t-i} = \theta(B)W_t \end{aligned}$$

where  $\theta(B)$  is the polynomial of degree  $q$ , given by

$$\theta(z) = \theta_0 + \theta_1 z + \dots + \theta_q z^q$$

A stationary process  $(X_t)_{t \in \mathbb{Z}}$  is called *Autoregressive Moving Average*, more briefly ARMA( $p, q$ ), if there are coefficients  $\phi_1, \dots, \phi_p, \theta_1, \dots, \theta_q$  such that

$$x_t - \phi_1 x_{t-1} - \dots - \phi_p x_{t-p} = w_t + \theta_1 w_{t-1} + \dots + \theta_q w_{t-q} \quad (2.4)$$

Using the polynomials AR and MA defined above and the delay operator  $B$ , we can write the equation in compact form:

$$\phi(B)X_t = \theta(B)W_t$$

These models are widely used to study and model time series in particular they are very much related to stationary processes due to their nature. Furthermore, they are very simple models which, however, manage to capture complex data behaviors. However, the specific dataset we have analyzed is very extensive and we have decided to test the power of these models on a **monthly average** of the original dataset, for two reasons: first of all this would have considerably reduced the computational complexity and would have allowed us to compare different configurations, but also because in order to be able to describe all the facets of the original dataset, the model would have required an enormous amount of parameters, and this would have been in contradiction with the nature of these models.

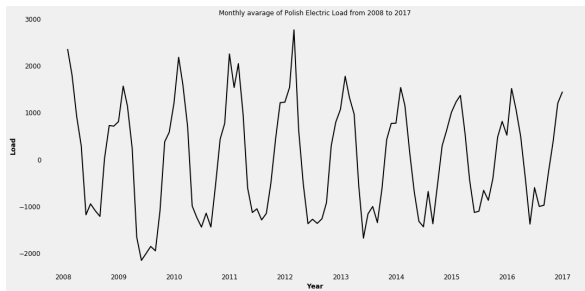


Figure 2.28: Monthly average of original dataset

## Arima modeling

The first step in analyzing the time series is to check its stationarity, and it was done using Augmented Dickey–Fuller test which tests that the time series can be represented by a unit root (has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time

series is stationary. The values of the parameters relating to this test are shown in the following table under the "monthly average" column.

Differencing	ADF Statistic	P value
monthly average	-2.64	0.085
1st order	-2.27	0.18
2nd order	-10.99	$6.80 \times 10^{-20}$

This shows us that according to the criteria of the hypothesis test used, it is necessary to reject the null hypothesis, i.e. that the series is stationary. Table 2.5 also shows the values of the test parameters for the first and second differencing order; In order to make the series stationary, the series has been differenced up to 2 times.

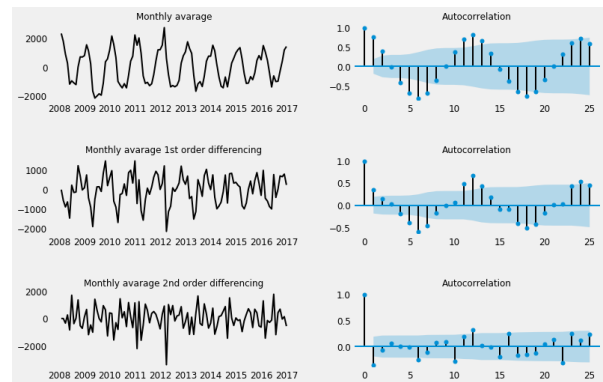


Figure 2.29: Lineplot and autocorrelation plot for the time series, for the 1st. and 2nd order differencing of the Monthly average of Polish Electric Load from 2008 to 2017.

The test is actually successful only for the second order differencing, moreover Figure 2.29 shows how differentiation manages to regularize the series by making it stationary, in particular, the autocorrelation

plot for the 2nd differencing is very similar to the autocorrelation of a white noise; These facts would suggest that the differentiation did the trick! However, a closer look at the autocorrelation plot for the 2nd differencing the lag goes into the far negative zone fairly quick, which indicates, the series might have been over differenced. Considering what has been said above, we decided to build two ARIMA models and compare them, one with a differentiation parameter  $d$  equal to 0 and the other equal to 2. The parameters relating to the auto-regression  $p$  and to the moving average  $q$  were selected by comparing the Akaike Information Criterion (AIC) of different models and have been fitted to the data. The main information relating to the two models has been reported in the following table:

Coeff	d=0	d=2
ar.L1	0.0819	0.5279
ar.L2	0.1028	-0.2791
ar.L3	-0.2692	-0.0252
ar.L4	0.3284	0.3296
ar.L5	-0.3148	-0.5174
ar.L6	-0.7677	-0.3888
ma.L1	0.3635	-1.9547
ma.L2	0.1579	1.3792
ma.L3	0.4566	-0.5751
ma.L4	-0.2928	-0.3030
ma.L5	0.4209	1.1751
ma.L6	0.8783	-0.7160
ma.L7	0.2458	Na
ma.L8	0.3423	Na
AIC	1385.880	1382.601
BIC	1423.707	1415.099

In Table 3 are shown values of autoregressive and moving average parameters of the models; in particular the comparison of the

AIC value for the two models suggested that the best configuration for the integrated one was ARIMA( $p=6, d=2, q=6$ ) and ARMA( $p=6, q=8$ ). Furthermore, it is interesting to note that the two models differ very little in terms of AIC and BIC score. The models were built using a statsmodel function called **SARIMAX**. One of the built-in methods of this function allows to get a diagnostic plot of the model which is very useful to get an idea of the agreement between models and data.

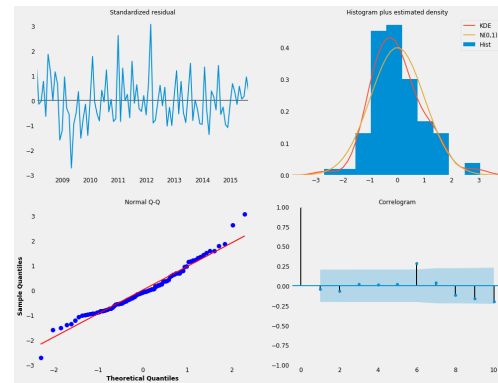


Figure 2.30: Diagnostic plot of the ARIMA(6,2,6) model. The plots show a good match between the model and the data as the residuals are very similar to Gaussian white noise

Figure 2.30 shows the agreement that exists between the ARIMA(6,2,6) model and the data and suggests that the chosen model could be an acceptable representation of the process represented by the data (Diagnostic for the ARMA(6,8) is very similar). Once the information about the two models is gathered, it's time to see how they perform in the forecast; For both models, a training set equal to about 80% of the volume of the series was used to calculate the parameters

and was asked to predict the last recorded cycle (still monthly average).

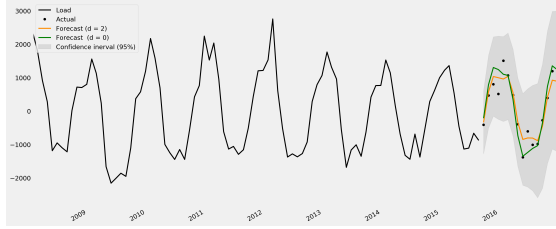


Figure 2.31: **Forecasting of the monthly average of Polish Electric Load for 2016.** Two different models were used: Arima(6,2,6)[Orange] and Arma(6,8)[Green].

The two models behave similarly on unexplored data, in particular the integrated model performs slightly better as it is given an RMSE of 270.22 which is approximately 9% of the global maximum of the data; The ARMA model, on the other hand, has an RMSE equal to 345.64 (12% of the maximum).

To conclude, it is interesting to note how the non-integrated model seems to be able to better capture the characteristic frequency of the data at this level, this prompted us to investigate this class of models better to verify if there was a way to better represent the time series under exam.

## Sarima modeling

The results obtained by building the ARIMA model just described have highlighted some very interesting aspects of the dataset. First of all, the fact that the autoregressive terms necessary to describe the data are a fairly high number suggests that there is a correlation between the data at least 6 lag apart. In

fact, we know that the studied dataset has recurring patterns with an annual period and it might be more appropriate to build a model that takes into account the **seasonal component** to obtain accurate forecasts. For this purpose, a model called SARIMA was taken into consideration which represents a more sophisticated version of the ARMA models and introduces parameters to manage the seasonality of the dataset, generally indicated with the acronym SARIMA(p,d,q)x(P,D,Q,s) where

- **p** and seasonal **P**: indicate number of autoregressive terms (lags of the stationarized series)
- **d** and seasonal **D**: indicate differencing that must be done to stationarize series
- **q** and seasonal **Q**: indicate number of moving average terms (lags of the forecast errors)
- **s**: indicates seasonal length in the data

Retracing the steps described above, two approaches were taken to determine the ideal SARIMA parameters: ACF and PACF plots, and a grid search. Let's proceed step by step, first we used a function of statsmodel called *seasonal decompose* which allows to obtain a decomposition of the time series in an additive sense.

$$y(t) = T(t) + S(t) + N(t)$$

where:

- $y(t)$  is the time series at the time step  $t$
- $T(t)$  is the **trend** component at the time step  $t$

- $S(t)$  is the **seasonal** component at the time step  $t$
- $N(t)$  is the **noise** component at the time step  $t$

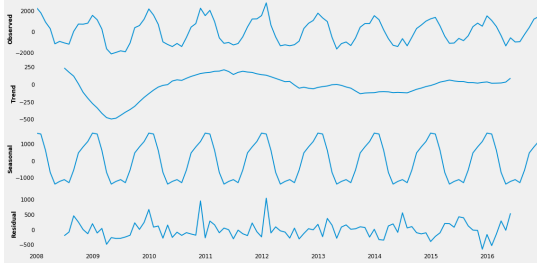


Figure 2.32: Series Additive decomposition of monthly average dataset.

Figure 2.32 actually shows that there is a strong seasonal component of length equal to 7 time steps. The idea now, as already mentioned, is to identify the regular (**d**) and seasonal (**D**) integration parameters and the seasonal one (**s**), then extrapolate the others through grid search. We already have information on the parameter **d** thanks to the previous analysis, to try to identify **D** we proceed by carrying out a hypothesis test AD Fuller on the seasonally differentiated series to defend seasonal lag values.

The P values are shown in figure 2.33, which shows how according to this test the best value for parameter **s** should be 6. It is important to underline that this hypothesis was subsequently confirmed by carrying out a bit of grid search also on this parameter, in fact the configurations with the lowest AIC are those that have the parameter **s** equal to 6. Going over again the process applied for the ARMA model, the series seems

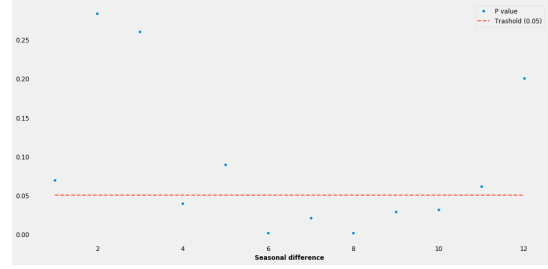


Figure 2.33: Scatter plot for P Values related to AD-Fuller test for stationarity of differentiated series. On X axis the differencing order taken for the time series.

to assume a stationary behavior by applying a second order differentiation to the seasonally differentiated series at the first order, in short, the best configuration for the integration parameters appears to be  $d=2$  and  $D=1$ .

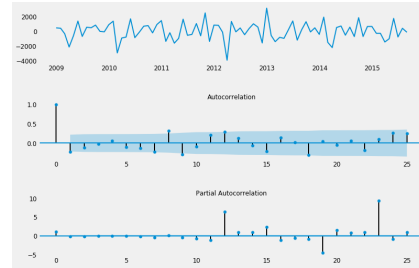


Figure 2.34: Autocorrelation and Partial Autocorrelation plots for the differentiated series of order  $d = 2$  and  $D = 1$ .

Recalling what has been said for the ARIMA model, let's proceed by testing the behavior of both cases. The grid search on the parameters  $p$ ,  $q$ ,  $P$ ,  $Q$  showed that the models which are in best agreement with the time series are  $SARIMA(3, 2, 1) \times (1, 1, 2, 6)$  and  $SARIMA(3, 0, 1) \times (1, 1, 2, 6)$ . The following table shows the values of the

parameters relating to both models. Once again the integrated model appears to be better according to the information criteria and diagnostic confirms the agreement of the model to the time series.

Coeff	d=0	d=2
ar.L1	-0.7437	-0.7343
ar.L2	0.6337	-0.3098
ar.L3	0.3774	0.0616
ma.L1	1	0.9992
ar.S.L1	-1	-0.9992
ma.S.L1	-0.1438	-0.2194
ma.S.L2	-0.8533	-0.7738
AIC	1293.954	1269.353
BIC	1313.589	1288.799

Once again the integrated model appears to be better according to the information criteria and diagnostic confirms the agreement of the model to the time series.

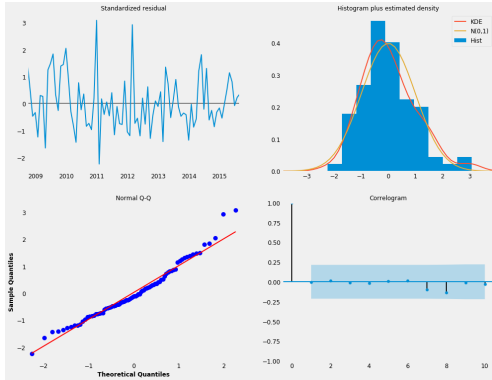


Figure 2.35: Diagnostic plot for SARIMA(3, 2, 1)x(1, 1, 2, 6) model.

At this point it remains only to check the behavior on the test set of these models. Finally, both models seem to learn better how to model the frequency that characterizes the time series, despite the fact that

the error is higher than that estimated by not considering seasonality; in fact RMSE for the integrated model is equal to 498.38 which is approximately 18% of the maximum, while the other one is 332.94, approximately 12% of the maximum.

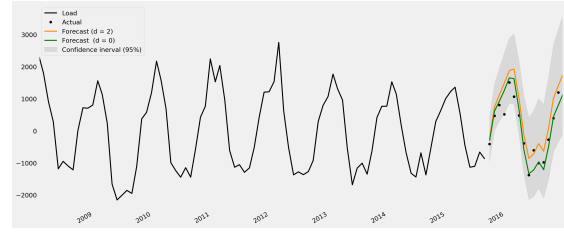


Figure 2.36: Forecasting of the monthly average of Polish Electric Load for 2016. Two different models were used: SARIMA(3, 2, 1)x(1, 1, 2, 6)[Orange] and SARIMA(3, 0, 1)x(1, 1, 2, 6)[Green].

## 2.6 Deconvolution

The last method that was used in the time series analysis was a deconvolution. More in detail, the idea was to hypothesize that the signal was the result of a convolution between a series of pulses and a kernel function that modeled its shape.

$$TS = d \circledast K$$

where TS represents the time series, d the pulse signal, which for convenience will be called Delta signal and K the kernel function.

The main purpose of this approach is to obtain a model capable of making predictions, to achieve this, it is necessary to know the delta signal and have an estimate of the Kernel function form.



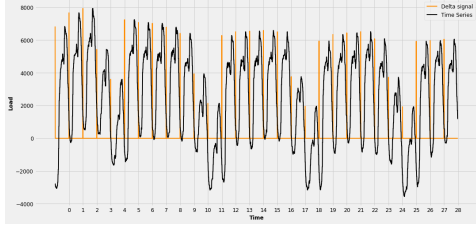


Figure 2.37: Lineplot of the first 30 days of of Polish Electric Load[Black], Representation of the pulse signal through a series of deltas centered at the starting point of each day and with an amplitude equal to the maximum relative to that day[Orange].

For simplicity, it has been assumed that the pulse signal was composed of a series of delta functions linked together with an amplitude equal to the maximum value of the time series during the day to which they refer; as regards the kernel function, it was obtained by considering an average of the time series at a daily level, in order to capture the behavior of the "average day". In fact, the figure shows how generally weekdays have a very similar trend, while holidays seem to have their own. This way we get 3 different kernels, one for weekdays, one for Saturday and one for Sunday.

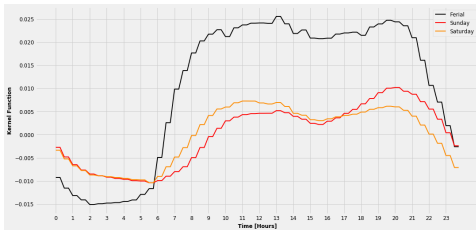


Figure 2.38: Kernel functions used for convolution.

stake, the model was built in the following way: The Delta signal, which represents the average year, was obtained by taking the average over all years of the maximums relating to each specific day, i.e. the amplitude of the delta relating to the first Monday of January is the average of the maximums of the first Monday of January from 2008 to 2015, the amplitude of the first Tuesday is the average of the highs of all the first Tuesdays, etc. The convolution of this delta signal with a kernel function that discriminates the weekdays from Saturday and Sunday, allows to obtain a representation of a "typical" year of the time series, thus obtaining the realization of an "average" full year. the convolution was obtained by exploiting the fourier space in which, as we know, the convolution becomes a product between the delta function and the kernel function:

$$F[TS] = F[d \otimes K] = F[d] \cdot F[K]$$

In particular we compared the model with the year 2016, which is used as a test set to see if it could actually be compared with the average behavior of the previous 8 years.

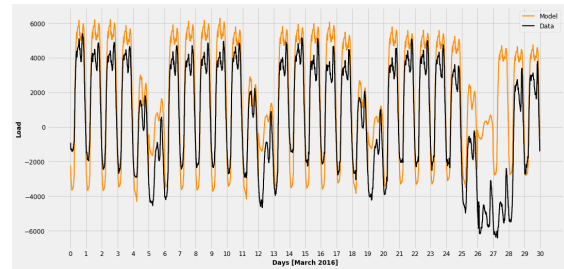


Figure 2.39: Comparison between model and time series for March 2016

Obviously this model, for how it was built is an approximation of the average behavior



of the Polish electric load, figure 2.39 shows its limits; It can be seen clearly in the figure that the model manages to capture the behavior of the data, but it remains quite inaccurate. Quantitatively, The correlation between the model residuals and the time series is 0.55 and the RMSE for the data showed in 2.39 is approximately 2150 which is the 31.39% of the maximum.

# Chapter 3

## Results and Conclusion

A wide analysis has been made about the load of the Polish Power Systems using different techniques. The information that has been extracted by this analysis has been used to gain some forecasting abilities about the dataset that has been considered.

As it was predictable, the time series presents a recurrent periodical behavior. This periodicity of the data permitted to use the **Fourier Transform method**. In order to make the Fourier Analysis more robust and clean, the signal has been preprocessed in the following way:

- **De-Trend:** A linear fit has been subtracted from the signal in order to make it "more stationary".
- **De-Mean:** The mean of the signal has been subtracted from the original one in order not to disturb the frequency spectrum ( $F(w = 0) = Mean = 0$ )
- **Smoothing:** In order to make the endpoints of the signal meet and make the periodic assumption more valid, the hanning window has been applied to the signal

After this preprocessing part, the signal has been analyzed in the Fourier spectrum,

thus retrieving life ticked frequencies like the 12h, 24h and 168h ones. The Fourier spectrum has been filtered by setting the frequencies with amplitude under a certain threshold to 0. This threshold has been chosen to vary as a certain fraction of the maximum amplitude of the original spectrum. In particular, a reconstruction algorithm that considered the correlation between the original signal and the reconstructed one permitted to obtain an optimal reconstruction by using the threshold  $k_{opt} = 0.004 \cdot Max$ . The same reconstruction algorithm has been applied to the first 7 years of the dataset (training set) in order to obtain the optimal frequency spectrum. **This spectrum has been used to predict the last year (test set) signal obtaining an RMSE up to the 33% of the maximum value.**

A refinement of this algorithm [4] has been used to consider the non-stationary nature of the signal. **This new approach considered a tripartite division (training/validation/test) and computed annual Fourier transforms, thus obtaining an RMSE up to the 21% of the original maximum.**

A more robust cleaning method on the signal that has been applied is the **Wavelet**

one. Under the safe assumption that the "real" signal is band-limited and the highest frequencies of the signal could be related to the noise, an algorithm has been developed in order to reconstruct a signal that could be as much as possible entirely and exclusively not influenced by the noise [3]. **This algorithm permitted to have a signal that has an error almost uncorrelated with the original signal:  $C = 0.2\%$ .**

Furthermore, the dataset has been studied using a class of models widely used to model time series, the ARMA models. This analysis showed how these models are able to adapt to the low frequencies of the dataset and manage to capture their behavior in a convincing way. In particular, different models were built and compared, starting from simpler models up to more sophisticated ones. It is interesting to note that in terms of RMSE the one that seems to perform better is the ARIMA(6,2,6) which does not take into account seasonality directly, but the information of the correlation between the data is mainly entrusted to the autoregressive terms. However, the two SARIMA models considered seem to be able to better capture the frequency-level behavior of the dataset by not forcibly chasing those data that could be associated with fluctuations. Probably a more precise tuning of the parameters could highlight other aspects, however this analysis has shown how also relatively simple models can be extremely powerful in this field. The last analysis performed on the time series was deconvolution, assuming that the series could be represented by the convolution of a Kernel function with a pulse function on a daily scale. The aim was to be able to create an

impulse signal that represented the average annual behavior of the series, in order to use it to make predictions. Unfortunately this method did not prove to be particularly efficient, although it is possible to obtain some information; the match between the data and the model is not satisfactory for several reasons. However, we must think that there have been different degrees of approximation and the impulse signal has been constructed on the basis of a priori assumptions that do not necessarily reflect the nature of the process. Anyway looking closely at the comparison between the data and the model, it would seem that one of the problems is related to the kernel function, in fact it is possible that the average behavior of weekdays and holidays is not a good approximation of the signal behavior at the daily level because this changes consistently for each specific day. The following table shows the RMSE values related to the various methods used for forecasting the series

Method	RMSE
Fourier	
Wavelet	
ARMA(6,8)	. 345.64
ARIMA(6,2,6)	270.22
SARIMA(3, 2, 1)x(1, 1, 2, 6)	498.38
SARIMA(3, 0, 1)x(1, 1, 2, 6)	332.94
Deconvolution	2150

# Chapter 4

## Appendix

In this section, the codes that permitted to obtain the output shown in the report have been reported. The codes are collected in their relative [GitHub page](#). The description of the notebook is the following:

1. "datapreprocessing.ipynb"

In this notebook data has been pre-processed: the mean value of the data and a linear fit has been subtracted by the original data. Moreover an hanning window has been applied.

2. "fouriermethod.ipynb"

Time independent Fourier Analysis has been performed. By the usage of an opportune filter, a forecast has been made.

3. "fouriermethod2.ipynb"

Time dependent Fourier Analysis has been performed. By the usage of an opportune filter, a forecast has been made.

4. "waveletfiltering.ipynb"

A wavelet filter has been adopted to clean the signal.

5. "SARIMA.ipynb"

Sarima and Arima processes have been

made in order to perform a forecast on the dataset that has been cleaned by the wavelet filtering.

6. "Deconvolution.ipynb"

A deconvolution algorithm has been applied and a forecast has been made on the last year available (2016)

### 4.1 datapreprocessing.py

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Visualization Notebook: Pre-
5 # # processing
6 # In[4]:
7
8
9 #Importing the libraries to watch
10 # the 'fits' image and get the
11 # data array
12 import astropy
13 import plotly.graph_objects as go
14 from astropy.io import fits
15 #Importing a library that is useful
16 # to read the original file
17 import pandas as pd
18 import pylab as plb
19 import matplotlib.pyplot as plt
20 from scipy.optimize import
21 curve_fit
```

```

18 from scipy import asarray as ar,exp
19 #Importing a visual library with
    some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 import math
25 import seaborn as sns
26 plt.style.use('fivethirtyeight')
27 plt.rcParams['font.family'] = 'sans
    -serif'
28 plt.rcParams['font.serif'] = '
    Ubuntu'
29 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
30 plt.rcParams['font.size'] = 14
31 plt.rcParams['axes.labelsize'] = 12
32 plt.rcParams['axes.labelweight'] =
    'bold'
33 plt.rcParams['axes.titlesize'] = 12
34 plt.rcParams['xtick.labelsize'] =
    12
35 plt.rcParams['ytick.labelsize'] =
    12
36 plt.rcParams['legend.fontsize'] =
    12
37 plt.rcParams['figure.titlesize'] =
    12
38 plt.rcParams['image.cmap'] = 'jet'
39 plt.rcParams['image.interpolation'] =
    'none'
40 plt.rcParams['figure.figsize'] =
    (16, 8)
41 plt.rcParams['lines.linewidth'] = 2
42 plt.rcParams['lines.markersize'] =
    8
43 plt.rcParams["axes.grid"] = False
44
45 colors = ['xkcd:pale orange', 'xkcd
    :sea blue', 'xkcd:pale red', '
    xkcd:sage green', 'xkcd:terra
    cotta', 'xkcd:dull purple', '
    xkcd:teal', 'xkcd: goldenrod',
    'xkcd:cadet blue',
46 'xkcd:scarlet']
47 cmap_big = cm.get_cmap('Spectral',
    512)
48 cmap = mcolors.ListedColormap(
    cmap_big(np.linspace(0.7, 0.95,
    256)))
49 bbox_props = dict(boxstyle="round,
    pad=0.3", fc=colors[0], alpha
    =.5)
50
51
52 # In[8]:
53
54 data=pd.read_csv('data.csv',sep=';',
    )
55
56
57
58 # In[10]:
59
60 data=data.rename(columns={'Data': '
    Day', 'Godzina': 'hour', 'Minuty':
    'minutes', 'Wolumen': 'Load'})
61
62
63
64 # In[11]:
65
66 #Building a continous time array
67 data['seconds']=np.arange(0,len(
    data)*900,900)
68
69
70
71 # In[15]:
72
73 plt.plot(data.seconds,data.Load,',',
    ,color='k')
74
75 plt.grid(True)
76 plt.xlabel('Time (s)',fontsize=20)
77 plt.ylabel('Load (MW)',fontsize=20)
78 plt.title('Load vs Time scatterplot
    ',fontsize=20)
79
80
81 # In[16]:
82
83
84 from scipy import signal
85
86
87 # In[17]:

```

```

88 1 #!/usr/bin/env python
89 2 # coding: utf-8
90 3
91 4 # # Time Independent Fourier
92 5 Transform
93 6 # In[2]:
94 7
95 8
96 9 #Importing the libraries to watch
97 10 the 'fits' image and get the
98 11 data array
99 12 import astropy
100 13 import plotly.graph_objects as go
101 14 from astropy.io import fits
102 15 #Importing a library that is useful
103 16 to read the original file
104 17 import pandas as pd
105 18 import pylab as plb
106 19 import matplotlib.pyplot as plt
107 20 from scipy.stats import chisquare
108 21
109 22 from scipy.optimize import
110 23 curve_fit
111 24 from scipy import asarray as ar,exp
112 25 #Importing a visual library with
113 26 some illustrative set up
114 27 import matplotlib.pyplot as plt
115 28 import matplotlib.colors as mcolors
116 29 from matplotlib import cm
117 30 import numpy as np
118 31 import math
119 32 from sklearn.metrics import
120 33 mean_squared_error
121 34 import seaborn as sns
122 35 from scipy import signal
123 36 plt.style.use('fivethirtyeight')
124 37 plt.rcParams['font.family'] = 'sans
125 38 -serif'
126 39 plt.rcParams['font.serif'] = '
127 40 Ubuntu'
128 41 plt.rcParams['font.monospace'] = '
129 42 Ubuntu Mono'
130 43 plt.rcParams['font.size'] = 14
131 44 plt.rcParams['axes.labelsize'] = 12
132 45 plt.rcParams['axes.labelweight'] =
133 46 'bold'
134 47 plt.rcParams['axes.titlesize'] = 12
135 48 plt.rcParams['xtick.labelsize'] =
136 49
137 50
138 51
139 52
140 53
141 54
142 55
143 56
144 57
145 58
146 59
147 60
148 61
149 62
150 63
151 64
152 65
153 66
154 67
155 68
156 69
157 70
158 71
159 72
160 73
161 74
162 75
163 76
164 77
165 78
166 79
167 80
168 81
169 82
170 83
171 84
172 85
173 86
174 87
175 88
176 89
177 90
178 91
179 92
180 93
181 94
182 95
183 96
184 97
185 98
186 99
187 100
188 101
189 102
190 103
191 104
192 105
193 106
194 107
195 108
196 109
197 110
198 111
199 112
200 113
201 114
202 115
203 116
204 117
205 118
206 119
207 120
208 121
209 122
210 123
211 124
212 125
213 126
214 127
215 128
216 129
217 130
218 131
219 132
220 133
221 134
222 135
223 136
224 137
225 138
226 139
227 140
228 141
229 142
230 143
231 144
232 145
233 146
234 147
235 148
236 149
237 150
238 151
239 152
240 153
241 154
242 155
243 156
244 157
245 158
246 159
247 160
248 161
249 162
250 163
251 164
252 165
253 166
254 167
255 168
256 169
257 170
258 171
259 172
260 173
261 174
262 175
263 176
264 177
265 178
266 179
267 180
268 181
269 182
270 183
271 184
272 185
273 186
274 187
275 188
276 189
277 190
278 191
279 192
280 193
281 194
282 195
283 196
284 197
285 198
286 199
287 200
288 201
289 202
290 203
291 204
292 205
293 206
294 207
295 208
296 209
297 210
298 211
299 212
300 213
301 214
302 215
303 216
304 217
305 218
306 219
307 220
308 221
309 222
310 223
311 224
312 225
313 226
314 227
315 228
316 229
317 230
318 231
319 232
320 233
321 234
322 235
323 236
324 237
325 238
326 239
327 240
328 241
329 242
330 243
331 244
332 245
333 246
334 247
335 248
336 249
337 250
338 251
339 252
340 253
341 254
342 255
343 256
344 257
345 258
346 259
347 260
348 261
349 262
350 263
351 264
352 265
353 266
354 267
355 268
356 269
357 270
358 271
359 272
360 273
361 274
362 275
363 276
364 277
365 278
366 279
367 280
368 281
369 282
370 283
371 284
372 285
373 286
374 287
375 288
376 289
377 290
378 291
379 292
380 293
381 294
382 295
383 296
384 297
385 298
386 299
387 300
388 301
389 302
390 303
391 304
392 305
393 306
394 307
395 308
396 309
397 310
398 311
399 312
400 313
401 314
402 315
403 316
404 317
405 318
406 319
407 320
408 321
409 322
410 323
411 324
412 325
413 326
414 327
415 328
416 329
417 330
418 331
419 332
420 333
421 334
422 335
423 336
424 337
425 338
426 339
427 340
428 341
429 342
430 343
431 344
432 345
433 346
434 347
435 348
436 349
437 350
438 351
439 352
440 353
441 354
442 355
443 356
444 357
445 358
446 359
447 360
448 361
449 362
450 363
451 364
452 365
453 366
454 367
455 368
456 369
457 370
458 371
459 372
460 373
461 374
462 375
463 376
464 377
465 378
466 379
467 380
468 381
469 382
470 383
471 384
472 385
473 386
474 387
475 388
476 389
477 390
478 391
479 392
480 393
481 394
482 395
483 396
484 397
485 398
486 399
487 400
488 401
489 402
490 403
491 404
492 405
493 406
494 407
495 408
496 409
497 410
498 411
499 412
500 413
501 414
502 415
503 416
504 417
505 418
506 419
507 420
508 421
509 422
510 423
511 424
512 425
513 426
514 427
515 428
516 429
517 430
518 431
519 432
520 433
521 434
522 435
523 436
524 437
525 438
526 439
527 440
528 441
529 442
530 443
531 444
532 445
533 446
534 447
535 448
536 449
537 450
538 451
539 452
540 453
541 454
542 455
543 456
544 457
545 458
546 459
547 460
548 461
549 462
550 463
551 464
552 465
553 466
554 467
555 468
556 469
557 470
558 471
559 472
560 473
561 474
562 475
563 476
564 477
565 478
566 479
567 480
568 481
569 482
570 483
571 484
572 485
573 486
574 487
575 488
576 489
577 490
578 491
579 492
580 493
581 494
582 495
583 496
584 497
585 498
586 499
587 500
588 501
589 502
590 503
591 504
592 505
593 506
594 507
595 508
596 509
597 510
598 511
599 512
600 513
601 514
602 515
603 516
604 517
605 518
606 519
607 520
608 521
609 522
610 523
611 524
612 525
613 526
614 527
615 528
616 529
617 530
618 531
619 532
620 533
621 534
622 535
623 536
624 537
625 538
626 539
627 540
628 541
629 542
630 543
631 544
632 545
633 546
634 547
635 548
636 549
637 550
638 551
639 552
640 553
641 554
642 555
643 556
644 557
645 558
646 559
647 560
648 561
649 562
650 563
651 564
652 565
653 566
654 567
655 568
656 569
657 570
658 571
659 572
660 573
661 574
662 575
663 576
664 577
665 578
666 579
667 580
668 581
669 582
670 583
671 584
672 585
673 586
674 587
675 588
676 589
677 590
678 591
679 592
680 593
681 594
682 595
683 596
684 597
685 598
686 599
687 600
688 601
689 602
690 603
691 604
692 605
693 606
694 607
695 608
696 609
697 610
698 611
699 612
700 613
701 614
702 615
703 616
704 617
705 618
706 619
707 620
708 621
709 622
710 623
711 624
712 625
713 626
714 627
715 628
716 629
717 630
718 631
719 632
720 633
721 634
722 635
723 636
724 637
725 638
726 639
727 640
728 641
729 642
730 643
731 644
732 645
733 646
734 647
735 648
736 649
737 650
738 651
739 652
740 653
741 654
742 655
743 656
744 657
745 658
746 659
747 660
748 661
749 662
750 663
751 664
752 665
753 666
754 667
755 668
756 669
757 670
758 671
759 672
760 673
761 674
762 675
763 676
764 677
765 678
766 679
767 680
768 681
769 682
770 683
771 684
772 685
773 686
774 687
775 688
776 689
777 690
778 691
779 692
780 693
781 694
782 695
783 696
784 697
785 698
786 699
787 700
788 701
789 702
790 703
791 704
792 705
793 706
794 707
795 708
796 709
797 710
798 711
799 712
800 713
801 714
802 715
803 716
804 717
805 718
806 719
807 720
808 721
809 722
810 723
811 724
812 725
813 726
814 727
815 728
816 729
817 730
818 731
819 732
820 733
821 734
822 735
823 736
824 737
825 738
826 739
827 740
828 741
829 742
830 743
831 744
832 745
833 746
834 747
835 748
836 749
837 750
838 751
839 752
840 753
841 754
842 755
843 756
844 757
845 758
846 759
847 760
848 761
849 762
850 763
851 764
852 765
853 766
854 767
855 768
856 769
857 770
858 771
859 772
860 773
861 774
862 775
863 776
864 777
865 778
866 779
867 780
868 781
869 782
870 783
871 784
872 785
873 786
874 787
875 788
876 789
877 790
878 791
879 792
880 793
881 794
882 795
883 796
884 797
885 798
886 799
887 800
888 801
889 802
890 803
891 804
892 805
893 806
894 807
895 808
896 809
897 810
898 811
899 812
900 813
901 814
902 815
903 816
904 817
905 818
906 819
907 820
908 821
909 822
910 823
911 824
912 825
913 826
914 827
915 828
916 829
917 830
918 831
919 832
920 833
921 834
922 835
923 836
924 837
925 838
926 839
927 840
928 841
929 842
930 843
931 844
932 845
933 846
934 847
935 848
936 849
937 850
938 851
939 852
940 853
941 854
942 855
943 856
944 857
945 858
946 859
947 860
948 861
949 862
950 863
951 864
952 865
953 866
954 867
955 868
956 869
957 870
958 871
959 872
960 873
961 874
962 875
963 876
964 877
965 878
966 879
967 880
968 881
969 882
970 883
971 884
972 885
973 886
974 887
975 888
976 889
977 890
978 891
979 892
980 893
981 894
982 895
983 896
984 897
985 898
986 899
987 900
988 901
989 902
990 903
991 904
992 905
993 906
994 907
995 908
996 909
997 910
998 911
999 912
1000 913
1001 914
1002 915
1003 916
1004 917
1005 918
1006 919
1007 920
1008 921
1009 922
1010 923
1011 924
1012 925
1013 926
1014 927
1015 928
1016 929
1017 930
1018 931
1019 932
1020 933
1021 934
1022 935
1023 936
1024 937
1025 938
1026 939
1027 940
1028 941
1029 942
1030 943
1031 944
1032 945
1033 946
1034 947
1035 948
1036 949
1037 950
1038 951
1039 952
1040 953
1041 954
1042 955
1043 956
1044 957
1045 958
1046 959
1047 960
1048 961
1049 962
1050 963
1051 964
1052 965
1053 966
1054 967
1055 968
1056 969
1057 970
1058 971
1059 972
1060 973
1061 974
1062 975
1063 976
1064 977
1065 978
1066 979
1067 980
1068 981
1069 982
1070 983
1071 984
1072 985
1073 986
1074 987
1075 988
1076 989
1077 990
1078 991
1079 992
1080 993
1081 994
1082 995
1083 996
1084 997
1085 998
1086 999
1087 1000
1088 1001
1089 1002
1090 1003
1091 1004
1092 1005
1093 1006
1094 1007
1095 1008
1096 1009
1097 1010
1098 1011
1099 1012
1100 1013
1101 1014
1102 1015
1103 1016
1104 1017
1105 1018
1106 1019
1107 1020
1108 1021
1109 1022
1110 1023
1111 1024
1112 1025
1113 1026
1114 1027
1115 1028
1116 1029
1117 1030
1118 1031
1119 1032
1120 1033
1121 1034
1122 1035
1123 1036
1124 1037
1125 1038
1126 1039
1127 1040
1128 1041
1129 1042
1130 1043
1131 1044
1132 1045
1133 1046
1134 1047
1135 1048
1136 1049
1137 1050
1138 1051
1139 1052
1140 1053
1141 1054
1142 1055
1143 1056
1144 1057
1145 1058
1146 1059
1147 1060
1148 1061
1149 1062
1150 1063
1151 1064
1152 1065
1153 1066
1154 1067
1155 1068
1156 1069
1157 1070
1158 1071
1159 1072
1160 1073
1161 1074
1162 1075
1163 1076
1164 1077
1165 1078
1166 1079
1167 1080
1168 1081
1169 1082
1170 1083
1171 1084
1172 1085
1173 1086
1174 1087
1175 1088
1176 1089
1177 1090
1178 1091
1179 1092
1180 1093
1181 1094
1182 1095
1183 1096
1184 1097
1185 1098
1186 1099
1187 1100
1188 1101
1189 1102
1190 1103
1191 1104
1192 1105
1193 1106
1194 1107
1195 1108
1196 1109
1197 1110
1198 1111
1199 1112
1200 1113
1201 1114
1202 1115
1203 1116
1204 1117
1205 1118
1206 1119
1207 1120
1208 1121
1209 1122
1210 1123
1211 1124
1212 1125
1213 1126
1214 1127
1215 1128
1216 1129
1217 1130
1218 1131
1219 1132
1220 1133
1221 1134
1222 1135
1223 1136
1224 1137
1225 1138
1226 1139
1227 1140
1228 1141
1229 1142
1230 1143
1231 1144
1232 1145
1233 1146
1234 1147
1235 1148
1236 1149
1237 1150
1238 1151
1239 1152
1240 1153
1241 1154
1242 1155
1243 1156
1244 1157
1245 1158
1246 1159
1247 1160
1248 1161
1249 1162
1250 1163
1251 1164
1252 1165
1253 1166
1254 1167
1255 1168
1256 1169
1257 1170
1258 1171
1259 1172
1260 1173
1261 1174
1262 1175
1263 1176
1264 1177
1265 1178
1266 1179
1267 1180
1268 1181
1269 1182
1270 1183
1271 1184
1272 1185
1273 1186
1274 1187
1275 1188
1276 1189
1277 1190
1278 1191
1279 1192
1280 1193
1281 1194
1282 1195
1283 1196
1284 1197
1285 1198
1286 1199
1287 1200
1288 1201
1289 1202
1290 1203
1291 1204
1292 1205
1293 1206
1294 1207
1295 1208
1296 1209
1297 1210
1298 1211
1299 1212
1300 1213
1301 1214
1302 1215
1303 1216
1304 1217
1305 1218
1306 1219
1307 1220
1308 1221
1309 1222
1310 1223
1311 1224
1312 1225
1313 1226
1314 1227
1315 1228
1316 1229
1317 1230
1318 1231
1319 1232
1320 1233
1321 1234
1322 1235
1323 1236
1324 1237
1325 1238
1326 1239
1327 1240
1328 1241
1329 1242
1330 1243
1331 1244
1332 1245
1333 1246
1334 1247
1335 1248
1336 1249
1337 1250
1338 1251
1339 1252
1340 1253
1341 1254
1342 1255
1343 1256
1344 1257
1345 1258
1346 1259
1347 1260
1348 1261
1349 1262
1350 1263
1351 1264
1352 1265
1353 1266
1354 1267
1355 1268
1356 1269
1357 1270
1358 1271
1359 1272
1360 1273
1361 1274
1362 1275
1363 1276
1364 1277
1365 1278
1366 1279
1367 1280
1368 1281
1369 1282
1370 1283
1371 1284
1372 1285
1373 1286
1374 1287
1375 1288
1376 1289
1377 1290
1378 1291
1379 1292
1380 1293
1381 1294
1382 1295
1383 1296
1384 1297
1385 1298
1386 1299
1387 1300
1388 1301
1389 1302
1390 1303
1391 1304
1392 1305
1393 1306
1394 1307
1395 1308
1396 1309
1397 1310
1398 1311
1399 1312
1400 1313
1401 1314
1402 1315
1403 1316
1404 1317
1405 1318
1406 1319
1407 1320
1408 1321
1409 1322
1410 1323
1411 1324
1412 1325
1413 1326
1414 1327
1415 1328
1416 1329
1417 1330
1418 1331
1419 1332
1420 1333
1421 1334
1422 1335
1423 1336
1424 1337
1425 1338
1426 1339
1427 1340
1428 1341
1429 1342
1430 1343
1431 1344
1432 1345
1433 1346
1434 1347
1435 1348
1436 1349
1437 1350
1438 1351
1439 1352
1440 1353
1441 1354
1442 1355
1443 1356
1444 1357
1445 1358
1446 1359
1447 1360
1448 1361
1449 1362
1450 1363
1451 1364
1452 1365
1453 1366
1454 1367
1455 1368
1456 1369
1457 1370
1458 1371
1459 1372
1460 1373
1461 1374
1462 1375
1463 1376
1464 1377
1465 1378
1466 1379
1467 1380
1468 1381
1469 1382
1470 1383
1471 1384
1472 1385
1473 1386
1474 1387
1475 1388
1476 1389
1477 1390
1478 1391
1479 1392
1480 1393
1481 1394
1482 1395
1483 1396
1484 1397
1485 1398
1486 1399
1487 1400
1488 1401
1489 1402
1490 1403
1491 1404
1492 1405
1493 1406
1494 1407
1495 1408
1496 1409
1497 1410
1498 1411
1499 1412
1500 1413
1501
```

```

12
39 plt.rcParams['ytick.labelsize'] = 12
40 plt.rcParams['legend.fontsize'] = 12
41 plt.rcParams['figure.titlesize'] = 12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation'] = 'none'
44 plt.rcParams['figure.figsize'] = (16, 8)
45 plt.rcParams['lines.linewidth'] = 2
46 plt.rcParams['lines.markersize'] = 8
47 plt.rcParams["axes.grid"] = False
48
49 colors = ['xkcd:pale orange', 'xkcd:sea blue', 'xkcd:pale red', 'xkcd:sage green', 'xkcd:terra cotta', 'xkcd:dull purple', 'xkcd:teal', 'xkcd:goldenrod', 'xkcd:cadet blue', 'xkcd:scarlet']
50 cmap_big = cm.get_cmap('Spectral', 512)
51 cmap = mcolors.ListedColormap(cmap_big(np.linspace(0.7, 0.95, 256)))
52 bbox_props = dict(boxstyle="round, pad=0.3", fc=colors[0], alpha=.5)
53
54
55 # In[3]:
56
57
58
59 data=pd.read_csv('data.csv',sep=';')
60 data=data.rename(columns={'Data':'Day','Godzina':'hour','Minuty':'minutes','Wolumen':'Load'})
61 data['seconds']=np.arange(0,len(data)*900,900)
62 detrended_sig=signal.detrend(data.Load)
63 sig=detrended_sig*np.hanning(len(detrended_sig))
64
65
66 # In[4]:
67
68 data=data.rename(columns={'Data':'Day','Godzina':'hour','Minuty':'minutes','Wolumen':'Load'})
69
70
71
72 # In[5]:
73
74 #Building a continous time array
75 data['seconds']=np.arange(0,len(data)*900,900)
76
77
78
79 # In[7]:
80
81
82 detrended_sig=signal.detrend(data.Load)
83 sig=detrended_sig*np.hanning(len(detrended_sig))
84
85
86 # ## Uniform varying threshold
87
88 # In[8]:
89
90
91 #Computing the FFT
92 FFT=np.fft.fft(sig)
93
94
95 # In[9]:
96
97
98 #Constructing the period x axis with the hours
99 new_N=int(len(FFT)/2)
100 f_nat=1/900
101 new_X = np.linspace(10**-12, f_nat/2, new_N, endpoint=True)
102 new_Xph=1.0/(new_X*60*60)
103
104
105 # In[10]:
106

```

```

107 FFT_abs=np.abs(FFT)
108 plt.plot(new_Xph,2*FFT_abs[0:int(
109     len(FFT)/2.)/len(new_Xph),
110     color='red')
111 plt.xlabel('Period ($h$)',fontsize=
112     20)
113 plt.ylabel('Amplitude',fontsize=20)
114 plt.title('(Fast) Fourier Transform
115     Method Algorithm',fontsize=20)
116 plt.grid(True)
117 plt.xlim(0,200)
118
119 # In[11]:
120
121 #Minimal difference has been shown
122 in the reconstruction
123 plt.subplot(1,2,1)
124 plt.plot(data.seconds,np.fft.ifft(
125     FFT),'',color='red')
126 plt.grid(True)
127 plt.xlabel('Time ($s$)')
128 plt.ylabel('Reconstructed Signal')
129 plt.subplot(1,2,2)
130 plt.plot(data.seconds,sig,'',color
131     ='k')
132 plt.grid(True)
133 plt.xlabel('Time ($s$)')
134 plt.ylabel('Original Signal')
135
136 # In[12]:
137
138 #Defining the filtering function
139 def fft_filter(th):
140     fft_tof=FFT.copy()
141     fft_tof_abs=np.abs(fft_tof)
142     fft_tof_abs=2*fft_tof_abs/len(
143     new_Xph)
144     fft_tof[fft_tof_abs<=th]=0
145     return fft_tof
146
147 # In[13]:
148
149 #Computing the RMSE for each
150 reconstruction
151 K=np.arange(0,2475+75,75)
152 RMSE=[]
153 for k in K:
154     rec_four=fft_filter(k)
155     rec=np.fft.ifft(rec_four)
156     RMSE.append(np.sqrt(
157     mean_squared_error(rec.real,sig
158     )))
159
160 # In[14]:
161
162 plt.plot(K,RMSE,color='k')
163 plt.xlabel('Threshold')
164 plt.ylabel('RMSE')
165 plt.grid(True)
166 plt.xticks(K,rotation=45)
167
168 # In[15]:
169
170 #Showing the plots at different
171 thresholds values
172 #Defining the amplitude filtering
173 function
174 def fft_filter_amp(th):
175     fft_tof=FFT.copy()
176     fft_tof_abs=np.abs(fft_tof)
177     fft_tof_abs=2*fft_tof_abs/len(
178     new_Xph)
179     fft_tof_abs[fft_tof_abs<=th]=0
180     return fft_tof_abs[0:int(len(
181     fft_tof_abs)/2.)]
182
183 # In[16]:
184
185 K_plot=[10,200,700,1500]
186 j=0
187 for k in K_plot:
188     j=j+1
189     plt.subplot(2,2,j)
190     plt.title('k=%i'%(k))
191     plt.xlim(0,200)

```



```

190 plt.plot(new_Xph,2*FFT_abs[0: 229
int(len(FFT)/2.)/len(new_Xph), 230 # In[19]:
color='navy',alpha=0.5,label=' 231
Original') 232
191 plt.grid(True) 233 #Performing the same RMSE process
192 plt.plot(new_Xph,fft_filter_amp as before, but relating to the
(k),'red',label='Filtered') maximum value
193 plt.xlabel('Time($h$)') 234 #Computing the RMSE for each
194 plt.ylabel('Amplitude') reconstruction
195 plt.legend() 235 K=np.arange(0.0,0.31,0.01)
196 plt.subplots_adjust(hspace=0.5) 236 RMSE=[]
197 237 for k in K:
198 238     rec_four=fft_filter(k)
199 # ## Maximum related threshold 239     rec=np.fft.ifft(rec_four)
200 240     RMSE.append(np.sqrt(
201 # In[17]: mean_squared_error(rec.real,sig
202 )))
203 241
204 #Maximum relate filter function 242
205 def fft_filter(perc): 243 # In[20]:
206     th=perc*(2*FFT_abs[0:int(len( 244
FFT)/2.)/len(new_Xph)).max() 245
207     fft_tof=FFT.copy() 246 plt.plot(K,RMSE,'.',color='k')
208     fft_tof_abs=np.abs(fft_tof) 247 plt.grid(True)
209     fft_tof_abs=2*fft_tof_abs/len( 248 plt.xlabel('Threshold as percentage
new_Xph) of the maximum value ',
210     fft_tof[fft_tof_abs<=th]=0 fontsize=20)
211     return fft_tof 249 plt.xticks(K,rotation=90)
212 250 plt.ylabel('RMSE',fontsize=20,
213 rotation=90)
214 # In[18]: 251
215 252
216 253 # In[21]:
217 #Showing some plots at different 254
threshold values 255
218 K_plot_values=[0.0,0.30,0.60,0.95] 256 #Computing the correlation between
219 j=0 the original signal and its
220 for k in K_plot_values: error
221     j+=1 257 CORR=[]
222     plt.subplot(4,1,j) 258 for k in K:
223     plt.plot(data.seconds,np.fft. 259     rec_four=fft_filter(k)
ifft(fft_filter(k)),color= 260     rec=np.fft.ifft(rec_four)
colors[j]) 261     error=rec.real-sig
224     plt.title('k=%.2f of the 262     CORR.append(np.abs(np.corrcoef(
maximum' %(k)) sig,error)[0][1]))
225     plt.xlabel('Time ($s$)') 263
226     plt.ylabel('Load') 264
227 plt.subplots_adjust(hspace=0.8) 265 # In[22]:
228 266

```

```

267
268 #Plotting the correlation between
    the signal and its error
269 plt.plot(K,CORR, '.',color='k')
270 plt.grid(True)
271 plt.xlabel('Threshold as percentage
    of the maximum value ',
    fontsize=20)
272 plt.xticks(K,rotation=90)
273 plt.ylabel('Correlation Coefficient',
    ',fontsize=20,rotation=90)
274
275
276 # In[23]:
277
278
279 #Reducing the range and selecting
    the best K value
280 K=np.arange(0.001,0.015,0.001)
281 CORR=[]
282 for k in K:
283     rec_four=fft_filter(k)
284     rec=np.fft.ifft(rec_four)
285     error=rec.real-sig
286     CORR.append(np.abs(np.corrcoef(
        sig,error)[0][1]))
287 plt.plot(K,CORR, '.',color='k')
288 plt.plot(K,np.zeros(len(K))+0.10,
    color='red')
289 plt.grid(True)
290 plt.xlabel('Threshold as percentage
    of the maximum value ',
    fontsize=20)
291 plt.xticks(K,rotation=90)
292 plt.ylabel('Correlation Coefficient',
    ',fontsize=20,rotation=90)
293
294
295 # In[24]:
296
297
298 #Displaying the optimum values
299 opt_perc=0.004
300 plt.subplot(2,2,1)
301 plt.plot(data.seconds,np.fft.ifft(
    fft_filter(opt_perc).real),',',
    color='red')
302 plt.grid(True)
303 plt.xlabel('Time($s$)')
304 plt.ylabel('Reconstructed Signal')
305 plt.subplot(2,2,2)
306 plt.plot(data.seconds,sig,',',color=
    ='k')
307 plt.grid(True)
308 plt.xlabel('Time($s$)')
309 plt.ylabel('Original Signal')
310 plt.subplot(2,2,4)
311 plt.xlim(0,200)
312 plt.plot(new_Xph,2*FFT_abs[0:int(
    len(FFT)/2.)/len(new_Xph),
    color='k',alpha=1.0,label='
    Original')
313 plt.legend()
314
315 plt.grid(True)
316 plt.xlabel('Time($h$)')
317 plt.ylabel('Original Fourier
    Transform amplitude')
318 plt.subplot(2,2,3)
319 plt.xlim(0,200)
320 plt.plot(new_Xph,2*np.abs(
    fft_filter(opt_perc))[0:int(len
    (FFT)/2.)/len(new_Xph),color='
    red',alpha=1.0,label='
    Reconstructed')
321 plt.grid(True)
322 plt.xlabel('Time($h$)')
323 plt.ylabel('Reconstructed Fourier
    Transform amplitude ')
324 plt.legend()
325
326
327 # In[25]:
328
329
330 #Distribution of the error
331 sns.distplot(sig-np.fft.ifft(
    fft_filter(opt_perc)).real,
    color='firebrick')
332 plt.grid(True)
333 plt.xlabel('Error distribution
    values',fontsize=20)
334 plt.ylabel('Distribution',fontsize
    =20)
335
336
337 # In[26]:
338

```

```

339
340 y_1=np.histogram(sig-np.fft.ifft( 374 ax_11.grid(True)
    fft_filter(opt_perc)).real,500) 375 ax_11.set_xlabel('Distribution
    [0]                               Values')
341 x_1=np.histogram(sig-np.fft.ifft( 376 ax_11.set_ylabel('Distribution')
    fft_filter(opt_perc)).real,500) 377
    [1][0:len(y_1)]                 378 #plt.y_1label('Residuals')
342                               379 ax_11.plot(x_1,gaus(x_1,*popt),'red
343                               ',label='fit')
344 # In[27]:                       380 plt.grid(True)
345                               381 ax_11.legend()
346                               382 res = y_1 - gaus(x_1,*popt)
347                               383 ax_12.plot(x_1,res,color='
    def gaus(x,a,x0,sigma):          darkorange',label='Residuals')
    return a*np.exp(-(x-x0)         384 ax_11.set_xlabel('Distribution
    **2/(2*sigma**2))              Values')
350                               385 ax_11.set_ylabel('Residuals')
351                               386 ax_12.legend()
352 # In[28]:                       387 plt.show()
353                               388
354                               389
355 #x_1=x_1[np.where((x_1>-500) & (x_1 390 # ## Prediction Part
    <500))]                          391
356 y_1=y_1[np.where((x_1>-500) & (x_1 392 # # Train Test Split
    <500))]                          393
357 x_1=np.linspace(-500,500,len(y_1)) 394 # In[30]:
358                               395
359                               396
360 # In[29]:                       397 #Selecting the first 80% of the
361                               signal
362                               398 sig=data.Load
363 #Plot of the values             399 sig=signal.detrend(sig)
364 val_medio=0                     400 sig=sig[0:365*96*8].copy()
365 n = len(x_1)                   401 FFT=np.fft.fft(sig)
    #the number of data              402 FFT_abs=np.abs(FFT)
366 mean = sum(x_1*y_1)/n           403
    #note this correction            404
367 sigma = sum(y_1*(x_1-val_medio)**2) 405 # In[31]:
    /n                               #note this correction 406
368 p0 = [max(y_1),val_medio,10]    407
369 popt,pcov = curve_fit(gaus,x_1,y_1 408 #Maximum relate filter function
    p0=p0)                          409 def fft_filter(perc):
370 fig, (ax_11, ax_12) = plt.subplots 410
    (2, 1)                          sig=data.Load
371 plt.suptitle('Fit and rediduals', 411
    fontsize=20)                    sig=signal.detrend(sig)
372 #ax_11.set_y_1_label('Intensity_1 412
    ADU'])                          sig=sig[0:365*96*8].copy()
373 ax_11.plot(x_1,y_1,'navy',label=' 413
    Distribution')                  FFT=np.fft.fft(sig)
                                    414 FFT_abs=np.abs(FFT)
                                    th=perc*(2*FFT_abs[0:int(len(
                                    FFT)/2.)]/len(new_Xph)).max()

```

```

416     fft_tof=FFT.copy()
417     fft_tof_abs=np.abs(fft_tof)
418     fft_tof_abs=2*fft_tof_abs/len(new_Xph)
419     fft_tof[fft_tof_abs<=th]=0
420     return fft_tof
421
422
423 # In[32]:
424
425
426 #Performing the same RMSE process
427 #as before, but relating to the
428 #maximum value
429 #Computing the RMSE for each
430 #reconstruction
431 K=np.arange(0.001,0.011,0.001)
432 RMSE=[]
433 CORR=[]
434 for k in K:
435     rec_four=fft_filter(k)
436     rec=np.fft.ifft(rec_four)
437     RMSE.append(np.sqrt(
438         mean_squared_error(rec.real,sig
439         )))
440     rec=np.fft.ifft(rec_four)
441     error=rec.real-sig
442     CORR.append(np.abs(np.corrcoef(
443         sig,error)[0][1]))
444
445 # In[33]:
446
447 plt.subplot(1,2,1)
448 plt.plot(K,RMSE,'.',color='k')
449 plt.grid(True)
450 plt.xlabel('Threshold as percentage
451 of the maximum value ',
452           fontsize=20)
453 plt.xticks(K,rotation=90)
454 plt.ylabel('RMSE',fontsize=20,
455           rotation=90)
456 plt.subplot(1,2,2)
457 plt.plot(K,CORR,'.',color='k')
458 plt.grid(True)
459 plt.xlabel('Threshold as percentage
460 of the maximum value ',
461           fontsize=20)
462
463 plt.plot(K,np.zeros(len(K))+0.10,
464         color='red')
465 plt.xticks(K,rotation=90)
466 plt.ylabel('Correlation',fontsize
467           =20,rotation=90)
468
469 # In[34]:
470
471 opt_perc=0.004
472 fft_filter(opt_perc)
473 plt.xlim(0,200)
474 plt.ylim(0,2500)
475 plt.plot(new_Xph,np.abs(fft_filter(
476     opt_perc))[0:len(new_Xph)]/len(
477     new_Xph),color='k',alpha=1.,
478     label='Original')
479 plt.xlabel('Time $h$')
480 plt.ylabel('Amplitude')
481 plt.grid(True)
482 plt.title('Fourier Spectrum',
483         fontsize=20)
484
485 # In[203]:
486
487 #Prediction:
488 #Using the Fourier formula
489 fourier=fft_filter(opt_perc)
490 space=np.array(data.seconds.tolist
491                ())
492 f_nat=1/(space[1]-space[0])
493 P=np.pi*2
494 N=int(len(space)/2)
495 freq=np.linspace(0,f_nat/2,len(
496     fourier))*2*np.pi
497 #fourier=f_try[1]
498 J_LIST=np.where(fourier!=0)[0]
499 real=fourier.real
500 imag=fourier.imag
501 T=np.linspace(0,f_nat/2,len(freq),
502             endpoint=True)
503 I=np.arange(1,101,1)
504 x_t=np.zeros(len(space))
505 q=0
506 SEEN=[]
507 for j in J_LIST:

```

```

493     q=q+1
494     x_t=x_t+2/(len(space))*(np.abs(
fourier[j].real)*np.cos(freq[j
]*2*space)-fourier[j].imag*np.
sin(2*freq[j]*space))
495 #print(i,len(space))
496     if int(100*q/len(J_LIST)) in I
and int(100*q/len(J_LIST)) not
in SEEN:
497         print('%i'%(int(100*q/len(
J_LIST)))+ ' % of the
frequencies reconstructed')
498         SEEN.append(int(100*q/len(
J_LIST)))
499         #X_T.append(x_t)
500
501
502 # In[200]:
503
504
505 x_t=np.array(x_t)
506 x_t[np.where(x_t>sig.max())]=sig.
max()
507
508
509 # In[312]:
510
511
512 #plt.plot(np.fft.ifft(fourier))
513 sig=data.Load
514 sig=signal.detrend(sig)
515 plt.plot(space,sig,color='k')
516 plt.plot(space[0:len(fourier)],x_t
[0:len(fourier)],color='
darkorange')
517 plt.plot(space[len(fourier):len(
data)],x_t[len(fourier):len(
data)],color='gold')
518 plt.grid(True)
519 plt.xlabel('Time($s$)')
520 plt.ylabel('Load (MW)')
521 #plt.plot(space[len(fourier)+10:len
(data)],exp,color='black')
522 #plt.plot(data.seconds[0:365*8*96],
np.fft.ifft(fourier))
523 ##plt.plot(data.seconds
[365*8*96:],sig[365*8*96:])
524 #plt.plot(space,x_t*max(sig)/max(
x_t))
525
526
527 # In[307]:
528
529
530 RMSE=mean_squared_error(sig
[365*8*96:len(data)],np.array(
x_t[365*8*96:len(data)]).real)
531
532
533 # In[309]:
534
535
536 print('RMSE is '+ str(RMSE))
537
538
539 # In[314]:
540
541
542 plt.plot(data.seconds[365*8*96:len(
data)],sig[365*8*96:len(data)],
'.',color='gold',label='
Prediction',markersize=5)
543 plt.plot(data.seconds[365*8*96:len(
data)],x_t[365*8*96:len(data)],
'.',color='k',label='Real',
markersize=5)
544 plt.legend()
545 plt.grid(True)
546 plt.xlabel('Time($s$)')
547 plt.ylabel('Signal')
548
549
550 # In[321]:
551
552
553 #Plotting the errors
554 plt.plot(np.array(data.seconds
[0:365*8*96] ,(sig-x_t)
[0:365*8*96] ,color='k')
555 plt.plot(np.array(data.seconds
[365*8*96:len(data)],(sig-x_t)
[365*8*96:len(data)],color='
gold')
556 plt.xlabel('Time($s$)')
557 plt.ylabel('Load($MW$)')
558 plt.grid(True)
559
560

```

```

561 # In[352]:
562
563
564 #RMSE for each interval
565
566 D=[0,20,30,60,90,120]
567 D=np.array(D)*900
568 start=365*900*8
569 T_in=(D+start)[: -1]
570 T_fin=(D+start)[1::]
571 D=[0,20,30,60,90,120]
572 new_D=np.array(D)
573 new_D=new_D*96
574 new_start=365*96*8
575 RMSE=[]
576 C=[]
577 for d in range(len(D)-1):
578     x_t=np.array(x_t[(new_start+
579 new_D[d]):(new_start+new_D[d
580 +1)])])
579     sig_=sig[(new_start+new_D[d]):(
580 new_start+new_D[d+1])]
581     error=x_t_.real-sig_
582     RMSE.append(np.sqrt(
583 mean_squared_error(sig_,x_t_.
584 real)))
585     C.append(np.corrcoef(error,sig_
586 ) [0] [1])
587
588 # In[357]:
589
590
591 RMSE_data=pd.DataFrame()
592 RMSE_data['$T_fin$(s)']=T_fin
593 RMSE_data['RMSE']=RMSE
594 RMSE_data['$\Delta T$(s)']=T_fin-
595 T_in
596 RMSE_data['$C_{data,error}$']=C
597 RMSE_data['$\Delta Days$']=D[1::]
598
599 # In[360]:
600
601 RMSE_data

```

## 4.3 fouriermethod2.ipynb

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Time Dependent method
5
6 # In[2]:
7
8
9 #Importing the libraries to watch
10 the 'fits' image and get the
11 data array
12 import astropy
13 import plotly.graph_objects as go
14 from astropy.io import fits
15 #Importing a library that is useful
16 to read the original file
17 import pandas as pd
18 import pylab as plb
19 import matplotlib.pyplot as plt
20 from scipy.stats import chisquare
21
22 from scipy.optimize import
23 curve_fit
24 from scipy import asarray as ar,exp
25 #Importing a visual library with
26 some illustrative set up
27 import matplotlib.pyplot as plt
28 import matplotlib.colors as mcolors
29 from matplotlib import cm
30 import numpy as np
31 import math
32 from sklearn.metrics import
33 mean_squared_error
34 import seaborn as sns
35 from scipy import signal
36 plt.style.use('fivethirtyeight')
37 plt.rcParams['font.family'] = 'sans
38 -serif'
39 plt.rcParams['font.serif'] = '
40 Ubuntu'
41 plt.rcParams['font.monospace'] = '
42 Ubuntu Mono'
43 plt.rcParams['font.size'] = 14
44 plt.rcParams['axes.labelsize'] = 12
45 plt.rcParams['axes.labelweight'] =
46 'bold'
47 plt.rcParams['axes.titlesize'] = 12

```

```

38 plt.rcParams['xtick.labelsize'] = 64
   12
39 plt.rcParams['ytick.labelsize'] = 66 # In[4]:
   12
40 plt.rcParams['legend.fontsize'] = 68
   12
41 plt.rcParams['figure.titlesize'] = 69 load_no_line=signal.detrend(data.
   12 Load,type='linear')
42 plt.rcParams['image.cmap'] = 'jet' 70 clean_load=np.array(load_no_line)-
43 plt.rcParams['image.interpolation'] 71 np.array(load_no_line).mean()
   = 'none' 72 data['clean_load']=clean_load
44 plt.rcParams['figure.figsize'] = 73 data=data.drop(columns=['Load']).
   (16, 8) rename(columns={'clean_load': '
45 plt.rcParams['lines.linewidth'] = 74 Load'})
46 plt.rcParams['lines.markersize'] = 75 # In[5]:
   8
47 plt.rcParams["axes.grid"] = True 76
48 #plt.rcParams[''] 77
49 colors = ['xkcd:pale orange', 'xkcd: 78 #Mean FFT on Training set
   :sea blue', 'xkcd:pale red', ' 79
   xkcd:sage green', 'xkcd:terra 80 data.Day=pd.to_datetime(data.Day)
   cotta', 'xkcd:dull purple', ' 81 data['Year']=data.Day.dt.year #
   xkcd:teal', 'xkcd: goldenrod', 82 Years
   'xkcd:cadet blue', 83 data['Month']=data.Day.dt.month #
50 'xkcd:scarlet'] Months
51 cmap_big = cm.get_cmap('Spectral', 84 YEARS=np.sort(list(set(data.Year.
   512)))).tolist() #Year list
52 cmap = mcolors.ListedColormap( 85 load_yf=0
   cmap_big(np.linspace(0.7, 0.95, 86 for y in range(len(YEARS)-2): #len
   256))) of the training set
53 bbox_props = dict(boxstyle="round, 87 df_year=data[data['Year']==
   pad=0.3", fc=colors[0], alpha 88 YEARS[y]]
   =.5) if y==0 or y==4: #leap years :)
54 89 df_year=df_year.drop(
55 df_year[df_year.Day==(str(YEARS
56 # In[3]: [y])+'-02-29')].index)
57 90 load_yf=load_yf+np.fft.fft(
58 91 df_year.Load)
59 data=pd.read_csv('data.csv',sep=';', 92 four_year=load_yf/(len(YEARS)-2)
   )
60 data=data.rename(columns={'Data': ' 93 # In[6]:
   Day', 'Godzina': 'hour', 'Minuty': 94
   'minutes', 'Wolumen': 'Load'}) 95
61 data['seconds']=np.arange(0,len( 96 #Fourier transform is defined on
   data)*900,900) the frequency axis.
62 detrended_sig=signal.detrend(data. 97 #In order to make it more visible,
   Load) the conversion on the period
63 sig=detrended_sig*np.hanning(len( has been applied
   detrended_sig))

```



```

98 new_N=int(len(four_year)/2)
99 f_nat=1/900
100 new_X = np.linspace(10**-12, f_nat
    /2, new_N, endpoint=True)
101 new_Xph=1.0/(new_X*60*60)
102 plt.xlim(0,40)
103 plt.plot(new_Xph,np.abs(2*four_year
    [:int(len(four_year)/2.)]/len(
    four_year)),color='red')
104 plt.ylabel('Amplitude')
105 plt.xlabel('Period ($h$)')
106 plt.ylim(0,3000)
107 plt.title('Mean Fourier Spectrum',
    fontsize=20)
108
109
110 # In[7]:
111
112
113 #Validation
114 TH=np.arange(0.01,0.5,0.01) #
    threshold Values
115 opt_t=[] #Threshold values that are
    acceptable in terms of RMSE
    and Correlation values
116 opt_corr=[] #Acceptable Correlation
    values
117 abs_year=np.array(abs(four_year))
118 for t in TH:
119     t_abs_year=abs_year.copy()
120     t_abs_year[t_abs_year<(t*
    t_abs_year.max())]=0 #Apply the
    threshold on the absolute
    value
121     t_four_year=four_year.copy()
122     t_four_year[t_abs_year==0]=0 #
    Apply the threshold on Fourier
    transform
123     RMSE=np.sqrt(mean_squared_error(
    np.fft.ifft(t_four_year).real,
    data[data.Year==2015].Load)) #
    inverse transform and compute
    RMSE
124     if RMSE<2000: #First
    requirement: LOW RMSE
125         new_TH=np.arange(t*0.1,t,
    *0.1) #More specific threshold
126         for t_new in new_TH:
127             t_new_abs_year=abs_year
    .copy()
            t_new_abs_year[
    t_new_abs_year<(t_new*
    t_new_abs_year.max())]=0
            t_new_four_year=
    four_year.copy()
            t_new_four_year[
    t_new_abs_year==0]=0
            E=np.fft.ifft(
    t_new_four_year).real-df_year.
    Load
            corr=pd.DataFrame({'A':
    df_year.Load,'B':E}).corr().
    values[0][1]
            if abs(corr)<=0.54: #
    Second requirement: LOW
    CORRELATION
                opt_t.append(t_new)
                opt_corr.append(
    corr)
    opt_corr=np.abs(opt_corr)
# In[8]:
#Selecting the best reconstruction
best_th=opt_t[opt_corr.argmin()] #
    Best threshold value
abs_year=np.array(abs(four_year))
t_abs_year=abs_year.copy()
t_abs_year[t_abs_year<(best_th*
    t_abs_year.max())]=0
med_four=four_year.copy()
med_four[t_abs_year==0]=0 #Best
    fourier transform
# In[9]:
RMSEs=[]
test=data[data['Year']==2016] #Test
    year
test=test.drop(test[test.Day==(
    '2016-02-29')].index) #Again,
    leap year
C=[] #Correlation list
TIME=np.arange(0,100,10) #Timestep

```



```

    we want to check
159 rmse_four=np.fft.ifft(med_four) #
    Take our prediction
160 test=np.array(test.Load)
161
162 for t in range(len(TIME)-1):
163     Rmse_test=test[96*(TIME[0])
    :96*(TIME[t+1])] #Take the load
    between two timestep
164     Rmse_four=rmse_four[96*(TIME
    [0]):(96*(TIME[t+1]))] #Take
    the prediction between the same
    timestep
165     RMSEs.append(np.sqrt(
    mean_squared_error(Rmse_four.
    real,Rmse_test))) #Compute the
    RMSE
166     C.append(np.corrcoef(Rmse_four.
    real-Rmse_test,Rmse_test)
    [0][1]) #Compute the
    correlation
167 res_data=pd.DataFrame({'RMSE':RMSEs
    , 'Days':TIME[1:], '$C_{data,
    error}$': C}) #Store them in a
    dataframe
168 res_data #Et voila
169
170
171 # In[11]:
172
173
174 print('Percentage of the maximum: ',
    + str(100*np.array(RMSEs).min()
    /data.Load.max())+ '%')
175
176
177 # In[17]:
178
179
180 plt.plot(rmse_four[0:30*96],color='
    k')
181 plt.plot(test[0:30*96],color='
    darkorange')
182 plt.xlabel('Time(Day)')
183 plt.ylabel('Load')
184 plt.xticks(np.arange(0,30*96,96),np
    .arange(0,30))

```

## 4.4 waveletfiltering.ipynb

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Wavelet Filtering
5
6 # In[4]:
7
8
9 #Importing the libraries to watch
    the 'fits' image and get the
    data array
10 import astropy
11 from astropy.io import fits
12 #Importing a library that is useful
    to read the original file
13 import pandas as pd
14 import pylab as plb
15 import matplotlib.pyplot as plt
16 from scipy.optimize import
    curve_fit
17 import pywt
18 from scipy.stats import chisquare
19 from scipy import asarray as ar,exp
20 #Importing a visual library with
21 from sklearn.metrics import
    mean_squared_error
22 import matplotlib.pyplot as plt
23 import matplotlib.colors as mcolors
24 import random
25 from matplotlib import cm
26 import matplotlib.patches as
    mpatches
27
28 import numpy as np
29 import math
30 import seaborn as sns
31 import datetime
32 plt.style.use('fivethirtyeight')
33 plt.rcParams['font.family'] = 'sans
    -serif'
34 plt.rcParams['font.serif'] = '
    Ubuntu'
35 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
36 plt.rcParams['font.size'] = 14
37 plt.rcParams['axes.labelsize'] = 12
38 plt.rcParams['axes.labelweight'] =

```

```

    'bold'
39 plt.rcParams['axes.titlesize'] = 12
40 plt.rcParams['xtick.labelsize'] =
    12
41 plt.rcParams['ytick.labelsize'] =
    12
42 plt.rcParams['legend.fontsize'] =
    12
43 plt.rcParams['figure.titlesize'] =
    12
44 plt.rcParams['image.cmap'] = 'jet'
45 plt.rcParams['image.interpolation']
    = 'none'
46 plt.rcParams['figure.figsize'] =
    (16, 8)
47 plt.rcParams['lines.linewidth'] = 2
48 plt.rcParams['lines.markersize'] =
    8
49 plt.rcParams["axes.grid"] = False
50
51 colors = ['xkcd:pale orange', 'xkcd:
    :sea blue', 'xkcd:pale red', '
    xkcd:sage green', 'xkcd:terra
    cotta', 'xkcd:dull purple', '
    xkcd:teal', 'xkcd: goldenrod',
    'xkcd:cadet blue',
52 'xkcd:scarlet']
53 cmap_big = cm.get_cmap('Spectral',
    512)
54 cmap = mcolors.ListedColormap(
    cmap_big(np.linspace(0.7, 0.95,
    256)))
55 bbox_props = dict(boxstyle="round,
    pad=0.3", fc=colors[0], alpha
    =.5)
56
57
58 # # Pre-processing steps
59
60 # In[5]:
61
62
63 data=pd.read_csv('data.csv',sep=';')
64
65
66 # In[6]:
67
68
69 data=data.rename(columns={'Data':'
    Day','Godzina':'hour','Minuty':
    'minute','Wolumen':'Load'})
70
71
72 # In[7]:
73
74
75 SECONDS=np.arange(900,900*len(data)
    +900,900)
76
77
78 # In[8]:
79
80
81 data['seconds']=SECONDS
82
83
84 # In[9]:
85
86
87 LOAD=data.Load
88
89
90 # In[10]:
91
92
93 from scipy import signal
94 load_no_line=signdetrend(LOAD,
    type='linear')
95 #load_no_constant=signdetrend(
    load_no_line,type='constant')
96
97
98 # In[11]:
99
100
101 clean_load=np.array(load_no_line)-
    np.array(load_no_line).mean()
102
103
104 # In[12]:
105
106
107 data['clean_load']=clean_load
108
109
110 # In[13]:
111

```

```

112 data=data.drop(columns=['Load']).
113     rename(columns={'clean_load':'
114         Load'})
115
116 # # Wavelet Transform
117
118 # In[14]:
119
120
121 #Performing the Wavelet transform
122     using a sym2
123 week=data
124 time=week.seconds.max()
125 sample_rate=1/900.
126 size= int(sample_rate*time)
127 t = np.linspace(0, time, num=size)
128 dataset = np.array(week.Load.tolist
129     ())
130 waveletname = 'sym2'
131 levels=8
132 fig, axarr = plt.subplots(nrows=
133     levels, ncols=2, figsize
134     =(20,10))
135 COEFF_D=[]
136 DATASET=[]
137 k=1
138 for ii in range(levels):
139     (dataset, coeff_d) = pywt.dwt(
140     dataset, waveletname,mode='per'
141     )
142     axarr[ii, 0].plot(dataset, '
143     black')
144     axarr[ii, 1].plot(coeff_d, '
145     darkorange')
146     axarr[ii, 0].set_ylabel("Level
147     {}".format(ii + 1), fontsize
148     =14, rotation=90)
149     axarr[ii, 0].set_yticklabels
150     ([])
151     if ii == 0:
152         axarr[ii, 0].set_title("
153         Approximation coefficients",
154         fontsize=14)
155         axarr[ii, 1].set_title("
156         Detail coefficients", fontsize
157         =14)
158     axarr[ii, 1].set_yticklabels
159
160     ([])
161     #print(len(coeff_d))
162     COEFF_D.append(np.repeat(
163     coeff_d,2**k))
164     DATASET.append(np.repeat(
165     dataset,2**k))
166     k=k+1
167 plt.tight_layout()
168 plt.show()
169
170 # In[15]:
171
172 #Preparing the gaussian fit
173 def gaus(x,a,x0,sigma):
174     return a*np.exp(-(x-x0)
175     **2/(2*sigma**2))
176
177 # In[16]:
178
179 #Histogram of the first coefficient
180 x=np.histogram(COEFF_D[0],500)
181     [1][0:500]
182 y=np.histogram(COEFF_D[0],500)[0]
183
184 # In[17]:
185
186 #To restrict the number of values
187     and fit the gaussian correctly,
188     this function has been used
189 def takeClosest(num,collection):
190     collection=collection.tolist()
191     collection=np.array(collection)
192     if num>=0:
193         collection=np.abs(
194     collection[np.where(collection
195     >0)])
196         a= min(collection,key=
197     lambda x:abs(x-abs(num)))
198     else:
199         collection=np.abs(
200     collection[np.where(collection

```

```

182     <0]])
183         a= -min(collection, key=
184             lambda x: abs(x-abs(num)))
185         return a
186
187 # In[18]:
188
189 #Gaussian Fit
190 x_try=x
191 b=x.tolist().index(takeClosest(500,
192     x))
193 a=x.tolist().index(takeClosest
194     (-500,x))
195 x=x[a:b]
196 y=y[a:b]
197
198 val_medio=x[int(len(x)/2)]
199 n = len(x)
200     the number of data
201 mean = sum(x*y)/n
202     #note this correction
203 sigma = sum(y*(x-val_medio)**2)/n
204     #note this correction
205 p0 = [max(y),val_medio,10]
206 popt,pcov = curve_fit(gaus,x,y,p0=
207     p0)
208 fig, (ax1, ax2) = plt.subplots(2,
209     1)
210 ax1.set_ylabel('Distribution ')
211 ax1.plot(x,y,'navy',label='First
212     Coefficient Distribution')
213 ax1.grid(True)
214 plt.ylabel('Residuals')
215 ax1.plot(x,gaus(x,*popt),'ro:',
216     label='Gaussian Fit')
217 ax1.legend()
218 res = y - gaus(x,*popt)
219 ax2.plot(res,color='darkorange')
220 ax2.grid(True)
221 plt.show()
222
223 # In[19]:
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

259 TH=np.arange(0.0,13.25,0.5)
260 C_TH=[]
261 for t in TH:
262     rec=recons_from_th_zero(t)
263     C_TH.append(np.abs(np.corrcoef(
264         rec-test,test)[0][1]))
265
266 # In[29]:
267
268
269 #Threshold, narrow range
270 TH=np.arange(0.0,1.01,0.01)
271 C_TH=[]
272 for t in TH:
273     rec=recons_from_th_zero(t)
274     C_TH.append(np.abs(np.corrcoef(
275         rec-test,test)[0][1]))
276
277 # In[37]:
278
279
280 #Best threshold
281 print('The Correlation has the
282     following percentage: '+str
283     (100*C_TH[-1]) +'%')
284
285
286 # In[53]:
287
288
289 #Mean and extreme value
290 mean=popt[1]
291 lim_neg=mean-1*Sigma
292 lim_pos=mean+1*Sigma
293
294 # In[78]:
295
296 lim_1=-1*Sigma
297 lim_2=-lim_1
298 coeff_0=np.array(COEFF_D[0].copy())
299 #mask = np.where((coeff_0<lim_2) &
300     (coeff_0>lim_1))
301 #coeff_0[np.where((coeff_0<lim_2) &
302     (coeff_0>lim_1))]=0
303
304 # In[101]:
305
306 x_hist=np.histogram(coeff_0,199)[1]
307 y_hist=np.histogram(coeff_0,200)[0]
308 filt_coeff_0=coeff_0[np.where((
309     coeff_0<lim_2) & (coeff_0>lim_1
310     ))]
311
312 # In[104]:
313
314 x_left=x_hist[np.where((x_hist<
315     lim_neg) & (x_hist>-500))]
316 x_right=x_hist[np.where(x_hist>
317     lim_pos)]
318 y_left=y_hist[np.where((x_hist<
319     lim_neg) & (x_hist>-500))]
320 y_right=y_hist[np.where(x_hist>
321     lim_pos)]
322 x_mid=x_hist[np.where((x_hist>
323     lim_neg) & (x_hist<lim_pos))]
324 y_mid=y_hist[np.where((x_hist>
325     lim_neg) & (x_hist<lim_pos))]
326
327 # In[111]:
328
329 plt.plot(x_left,y_left,color='k')
330 plt.plot(x_right,y_right,color='k',
331     label='Information')
332 plt.plot(x_mid,y_mid,color='
333     darkorange',label='Noise')
334 plt.plot(np.zeros(12000)+lim_neg,np
335     .arange(0,12000),color='red')
336 plt.plot(np.zeros(12000)+lim_pos,np
337     .arange(0,12000),color='red')
338
339 plt.grid(True)
340 plt.xlabel('Coefficient values')
341 plt.ylabel('Coefficient
342     Distribution')
343 plt.legend(fontsize=20)

```

## 4.5 SARIMA.ipynb

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # SARIMA MODELING
5
6 # Import dataset
7
8 # In[2]:
9
10
11 #Importing the libraries to watch
    the 'fits' image and get the
    data array
12 import astropy
13 import plotly.graph_objects as go
14 from astropy.io import fits
15 #Importing a library that is useful
    to read the original file
16 import pandas as pd
17 import pylab as plb
18 import matplotlib.pyplot as plt
19 from scipy.optimize import
    curve_fit
20 from scipy import asarray as ar,exp
21 #Importing a visual library with
    some illustrative set up
22 import matplotlib.pyplot as plt
23 import matplotlib.colors as mcolors
24 from matplotlib import cm
25 import numpy as np
26 import math
27 import seaborn as sns
28 plt.style.use('fivethirtyeight')
29 plt.rcParams['font.family'] = 'sans
    -serif'
30 plt.rcParams['font.serif'] = '
    Ubuntu'
31 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
32 plt.rcParams['font.size'] = 14
33 plt.rcParams['axes.labelsize'] = 12
34 plt.rcParams['axes.labelweight'] =
    'bold'
35 plt.rcParams['axes.titlesize'] = 12
36 plt.rcParams['xtick.labelsize'] =
    12
37 plt.rcParams['ytick.labelsize'] =
    12
38 plt.rcParams['legend.fontsize'] =
    12
39 plt.rcParams['figure.titlesize'] =
    12
40 plt.rcParams['image.cmap'] = 'jet'
41 plt.rcParams['image.interpolation']
    = 'none'
42 plt.rcParams['figure.figsize'] =
    (16, 8)
43 plt.rcParams['lines.linewidth'] = 2
44 plt.rcParams['lines.markersize'] =
    8
45 plt.rcParams["axes.grid"] = False
46
47 colors = ['xkcd:pale orange', 'xkcd:
    sea blue', 'xkcd:pale red', '
    xkcd:sage green', 'xkcd:terra
    cotta', 'xkcd:dull purple', '
    xkcd:teal', 'xkcd: goldenrod',
    'xkcd:cadet blue',
48 'xkcd:scarlet']
49 cmap_big = cm.get_cmap('Spectral',
    512)
50 cmap = mcolors.ListedColormap(
    cmap_big(np.linspace(0.7, 0.95,
    256)))
51 bbox_props = dict(boxstyle="round,
    pad=0.3", fc=colors[0], alpha
    =.5)
52
53
54 # In[3]:
55
56
57 from statsmodels.graphics.tsaplots
    import plot_pacf
58 from statsmodels.graphics.tsaplots
    import plot_acf
59 from statsmodels.tsa.arima_process
    import ArmaProcess
60 from statsmodels.stats.diagnostic
    import acorr_ljungbox
61 from statsmodels.tsa.statespace.
    sarimax import SARIMAX
62 from statsmodels.tsa.stattools
    import adfuller
63 from statsmodels.tsa.stattools
    import pacf
```

```

64 from statsmodels.tsa.stattools
    import acf
65 from tqdm import tqdm_notebook
66 import matplotlib.pyplot as plt
67 import numpy as np
68 import pandas as pd
69
70 from itertools import product
71
72
73 # In[4]:
74
75
76 dataset = pd.read_csv('/Users/
    Simone/Desktop/University/DDA/
    Lab/Relazione_2_TimesSeries/
    reconstruction.csv')
77
78 dataset = dataset.drop(columns=['
    Unnamed: 0'])
79
80 full_load = pd.Series(dataset.R)
81
82
83 # In[5]:
84
85
86 import csv
87 import datetime as dat
88 import pandas as pd
89 import numpy as np
90 import matplotlib.pyplot as plt
91
92 dir='/Users/Simone/Desktop/
    University/DDA/Lab/
    Relazione_2_TimesSeries/'
93 file='daneOkresoweKSE.csv'
94
95 # Read data from file 'filename.csv'
96
97 # (in the same directory that your
    python process is based)
98 # Control delimiters, rows, column
    names with read_csv (see later)
99 data = pd.read_csv(dir+file)
100 # Preview the first 5 lines of the
    loaded data
101 data.head()
102
103 data = pd.read_csv(dir+file,
    skiprows=0,sep=';')
104 data.head()
105
106
107 data.columns= ['Day','hour','minute
    ','Load']
108 #plt.figure(figsize=(10, 10))
109 #plt.plot(F606,F814,'k')
110 #plt.ylim(32,12)
111 #plt.xlim(32,12)
112 #plt.show()
113 #plt.close
114
115
116 # In[6]:
117
118
119 N= 10000# data.shape[0]
120 print("Using {} entries on {}
    available".format(N, data.shape
    [0]) )
121 load=np.zeros(N) #vogliamo mettere
    i vettori tempo in formato np,
    per fare successive analisi, in
    formato datetime pu essere
    scomodo
122 time=np.zeros(N)
123
124
125 txt0 = "{} {:02d}:{:02d}:00".format
    (data['Day'][0],data['hour'
    ][0],0)
126 T0= dat.datetime.strptime(txt0,'%Y
    -%m-%d %H:%M:%S')
127
128 dday=0
129 for i in range(N):
130     load[i]=data['Load'][i]
131
132     if data['hour'][i]==24:
133         data['hour'][i]=0
134         dday=1
135
136     txt = "{} {:02d}:{:02d}:00".
    format(data['Day'][i],data['
    hour'][i],data['minute'][i])
137     date= dat.datetime.strptime(txt

```

```

138 , '%Y-%m-%d %H:%M:%S')
139 #Serve questo loop perch i
140 polacchi segnano la mezzanotte
141 in maniera differente
142 if dday==1:
143     date=date+dat.timedelta(
144     days=1)
145     Secs=(date-T0).total_seconds()
146     time[i]=Secs
147     dday=0
148 plt.figure(figsize=(10,10))
149 plt.plot(time,load,'.k')
150 plt.xlabel('Time [s]')
151 plt.ylabel('Load [MW]')
152 plt.title('Polish Electric Load
153 since {} {}:02d}:{:02d}:00'.
154 format(data['Day'][0],data['
155 hour'][0],0) )
156 plt.show()
157 #
158 # In[8]:
159 s = data.Day.loc[0]
160 data.iloc[96]
161
162 # In[9]:
163
164 data.tail()
165
166 # In[10]:
167
168 from scipy import signal
169 dir='/Users/Simone/Desktop/
170 University/DDA/Lab/
171 Relazione_2_TimesSeries/'
172 file='daneOkresoweKSE.csv'
173
174 # Read data from file 'filename.csv'
175 # (in the same directory that your
python process is based)
176 # Control delimiters, rows, column
177 names with read_csv (see later)
178 data = pd.read_csv(dir+file, sep =
179 ;')
180 data=data.rename(columns={'Data':
181 'Day', 'Godzina': 'hour', 'Minuty':
182 'minute', 'Wolumen': 'Load'})
183 SECONDS=np.arange(900,900*len(data)
184 +900,900)
185 data['seconds']=SECONDS
186 LOAD=data.Load
187 load_no_line=signal.detrend(LOAD,
188 type='linear')
189 clean_load=np.array(load_no_line)-
190 np.array(load_no_line).mean()
191 data['clean_load']=clean_load
192 data=data.drop(columns=['Load']).
193 rename(columns={'clean_load':
194 'Load'})
195
196 # In[11]:
197
198 s = '2008-01-01 00:15:00'
199 end = '2017-01-01 00:00:00'
200
201 # In[12]:
202
203 index = pd.date_range(start = s ,
204 end = end , freq='15T')
205 series = pd.Series(data.Load.tolist
206 (), index=index)
207 series.resample('1W').mean()
208
209 # In[13]:
210
211 W=pd.DataFrame({'Time':series.
212 resample('1W').mean().index,
213 'Load':series.resample('1W').
214 mean()})
215 M=pd.DataFrame({'Time':series.
216 resample('1M').mean().index,
217 'Load':series.resample('1M').

```



```

    mean()})
209 D=pd.DataFrame({'Time':series.
    resample('1D').mean().index,'
    Load':series.resample('1D').
    mean()})
210
211
212 # In[14]:
213
214
215 M = M.drop(M.index[-1])
216
217
218 # In[15]:
219
220
221 plt.plot(M.Load,color = 'k')
222 plt.xlabel('Year')
223 plt.ylabel('Load')
224 plt.title('Monthly avarage of
    Polish Electric Load from 2008
    to 2017')
225
226
227 # # Stationarity
228
229 # Let's check stationary properties
    , let's plot the series, the
    series differencied 1 and 2
    times, and test stationarity
    with ADF test
230
231 # In[56]:
232
233
234 M['L_diff'] = M.Load.diff()
235
236
237 # In[18]:
238
239
240 plt.plot(M.L_diff)
241
242
243 # In[19]:
244
245
246 M['L_diff_2'] = M.L_diff.diff()
247
248
249 # In[20]:
250
251
252 plt.plot(M.L_diff_2,color = 'k')
253 plt.xlabel('Year')
254 plt.ylabel('Load')
255 plt.title('Monthly avarage of
    Polish Electric Load from 2008
    to 2017')
256
257
258 # In[64]:
259
260
261 M = M.fillna(0)
262
263
264 # In[65]:
265
266
267 import statsmodels.api as sm
268 from statsmodels.tsa.arima_model
    import ARIMA
269
270 fig = plt.figure(figsize=(12,8))
271 ax1 = fig.add_subplot(211)
272 fig = sm.graphics.tsa.plot_acf(M.
    Load, ax=ax1)
273 ax2 = fig.add_subplot(212)
274 fig = sm.graphics.tsa.plot_pacf(M.
    Load, ax=ax2)
275
276
277 # In[66]:
278
279
280 fig = plt.figure(figsize=(12,8))
281 ax1 = fig.add_subplot(211)
282 fig = sm.graphics.tsa.plot_acf(M.
    L_diff, ax=ax1)
283 ax2 = fig.add_subplot(212)
284 fig = sm.graphics.tsa.plot_pacf(M.
    L_diff, ax=ax2)
285
286
287 # In[70]:
288
289

```

```

290 fig = plt.figure(figsize=(12,8))
291 ax1 = fig.add_subplot(211)
292 fig = sm.graphics.tsa.plot_acf(M.
    L_diff_2, ax=ax1, lags =25)
293 ax2 = fig.add_subplot(212)
294 fig = sm.graphics.tsa.plot_pacf(M.
    L_diff_2, ax=ax2, lags = 25)
295
296
297 # In[75]:
298
299
300 fig = plt.figure(figsize=(12,8))
301 ax11 = fig.add_subplot(321)
302 plt.plot(M.Load,color = 'k')
303
304 plt.title('Monthly avarage')
305
306 ax12 = fig.add_subplot(322)
307 fig = sm.graphics.tsa.plot_acf(M.
    Load, ax=ax12, lags =25)
308 ax21 = fig.add_subplot(323)
309 plt.plot(M.L_diff,color = 'k')
310
311 plt.title('Monthly avarage 1st
    order differencing')
312 ax22 = fig.add_subplot(324)
313 fig = sm.graphics.tsa.plot_acf(M.
    L_diff, ax=ax22, lags =25)
314 ax21 = fig.add_subplot(325)
315 plt.plot(M.L_diff_2,color = 'k')
316
317 plt.title('Monthly avarage 2nd
    order differencing')
318 ax22 = fig.add_subplot(326)
319 fig = sm.graphics.tsa.plot_acf(M.
    L_diff_2, ax=ax22, lags =25)
320 fig.tight_layout(pad=3.0)
321
322
323 # In[69]:
324
325
326 result = adfuller(M.Load)
327 print('ADF Statistic: {}'.format(
    result[0]))
328 print('p-value: {}'.format(result
    [1]))
329 print('Critical Values:')
330 for key, value in result[4].items()
    :
331     print('\t{}: {}'.format(key,
    value))
332
333 # In[76]:
334
335
336 result = adfuller(M.L_diff)
337 print('ADF Statistic: {}'.format(
    result[0]))
338 print('p-value: {}'.format(result
    [1]))
339 print('Critical Values:')
340 for key, value in result[4].items()
    :
341     print('\t{}: {}'.format(key,
    value))
342
343 # In[77]:
344
345 result = adfuller(M.L_diff_2)
346 print('ADF Statistic: {}'.format(
    result[0]))
347 print('p-value: {}'.format(result
    [1]))
348 print('Critical Values:')
349 for key, value in result[4].items()
    :
350     print('\t{}: {}'.format(key,
    value))
351
352 # # ARMA
353
354 # First of all we try to build An
    ARMA model on the data
355
356 # In[24]:
357
358 N = len(M.Load)
359 split = 0.85
360 training_size = round(split*N)
361 test_size = round((1-split)*N)

```

```

368 series = M.Load[:training_size]
369
370
371 # In[25]:
372
373
374 def optimize_ARIMA(order_list, exog
375 ):
376     """
377     Return dataframe with
378     parameters and corresponding
379     AIC
380
381     order_list - list with (p,
382     d, q) tuples
383     exog - the exogenous
384     variable
385     """
386
387     results = []
388
389     for order in tqdm_notebook(
390 order_list):
391         try:
392             model = SARIMAX(exog,
393 order=order).fit(dis=-1)
394         except:
395             continue
396
397         aic = model.aic
398         results.append([order,
399 model.aic])
400
401     result_df = pd.DataFrame(
402 results)
403     result_df.columns = ['(p, d, q)',
404 'AIC']
405     #Sort in ascending order, lower
406     AIC is better
407     result_df = result_df.
408 sort_values(by='AIC', ascending
409 =True).reset_index(drop=True)
410
411     return result_df
412
413 # In[26]:
414
415
416 ps = range(0, 10, 1)
417 d = 0
418 qs = range(0, 10, 1)
419
420 # Create a list with all possible
421 combination of parameters
422 parameters = product(ps, qs)
423 parameters_list = list(parameters)
424
425 order_list = []
426
427 for each in parameters_list:
428     each = list(each)
429     each.insert(1, d)
430     each = tuple(each)
431     order_list.append(each)
432
433 result_df = optimize_ARIMA(
434 order_list, exog = series)
435
436 result_df
437
438 # Try to fit the best model
439 according to AIC
440
441 # In[76]:
442
443 best_model = SARIMAX(series, order
444 =(6,2,6)).fit()
445 print(best_model.summary())
446 s_best_model = SARIMAX(series,
447 order=(6,0,8)).fit()
448 print(s_best_model.summary())
449
450 # In[77]:
451
452 best_model.plot_diagnostics(figsize
453 =(15,12))
454 s_best_model.plot_diagnostics(
455 figsize=(15,12))
456
457 # In[29]:

```

```

446 original_1 = M.L_diff_2.cumsum() +
    M.L_diff.max()-M.L_diff_2.
    cumsum().max()
447 original_1[0]=M.L_diff[0]
448 original_2 = original_1.cumsum() +
    M.Load.max()- original_1.cumsum
    ().max()
449 original_2[0] = M.Load[0]
450 plt.plot(M.Load)
451 plt.plot(original_2)
452
453 # Let's see how forecast performs
454 # In[86]:
455
456
457
458
459
460 fore_1= test_size -1
461 forecast = best_model.
    get_prediction(start=
    training_size, end=
    training_size+fore_1)
462 forec = forecast.predicted_mean
463 ci = forecast.conf_int(alpha=0.05)
464
465 s_forecast = s_best_model.
    get_prediction(start=
    training_size, end=
    training_size+fore_1)
466 s_forec = s_forecast.predicted_mean
467 s_ci = forecast.conf_int(alpha
    =0.05)
468
469 fig, ax = plt.subplots(figsize
    =(16,8), dpi=300)
470 x0 = M.Load.index[0:training_size]
471 x1=M.Load.index[training_size:
    training_size+fore_1+1]
472 #ax.fill_between(forec, ci['lower
    Load'], ci['upper Load'])
473 plt.plot(x0, M.Load[0:training_size],
    'k', label = 'Load')
474
475 plt.plot(M.Load[training_size:
    training_size+fore_1], '.k',
    label = 'Actual')
476
477 forec = pd.DataFrame(forec, columns
    =['f'], index = x1)
478 forec.f.plot(ax=ax,color = '
    Darkorange',label = 'Forecast (
    d = 2)')
479 ax.fill_between(x1, ci['lower Load'
    ], ci['upper Load'],alpha=0.2,
    label = 'Confidence interval
    (95%)',color='grey')
480
481 s_forec = pd.DataFrame(s_forec,
    columns=['f'], index = x1)
482 s_forec.f.plot(ax=ax,color = 'green
    ',label = 'Forecast (d = 0)')
483 #ax.fill_between(x1, s_ci['lower
    Load'], s_ci['upper Load'],
    alpha=0.2, label = 'Confidence
    interval (95%)',color='grey')
484
485 plt.legend(loc = 'upper left')
486 plt.show()
487
488
489 # In[89]:
490
491
492
493 from sklearn.metrics import
    mean_squared_error
494 RMSE_Arima = np.sqrt(
    mean_squared_error(M.Load[
    training_size:training_size+
    fore_1],forec.f[0:len(forec)
    -1]))
495 print('RMSE : {:.02f}'.format(
    RMSE_Arima))
496
497 # In[90]:
498
499
500 RMSE_Arima/M.Load.max()
501
502 # In[91]:
503
504
505 from sklearn.metrics import
    mean_squared_error
506 RMSE_Arima = np.sqrt(

```

```

mean_squared_error(M.Load[
training_size:training_size+
fore_1],s_forec.f[0:len(forec)
-1]))
509 print('RMSE : {:.02f}'.format(
    RMSE_Arima))
510
511
512 # In[92]:
513
514
515 RMSE_Arima/M.Load.max()
516
517
518 # # SARIMA
519
520 # Let's see if taking care of
    seasonality could help building
    a better model, tre process is
    very similar to the one used
    for ARIMA modeling, however a
    decomposition tool helped us
    checking the stationarity
    components.
521
522 # In[100]:
523
524
525 def get_stationarity(timeseries,
    window):
526
527     # rolling statistics
528     rolling_mean = timeseries.
    rolling(window=window).mean()
529     rolling_std = timeseries.
    rolling(window=window).std()
530
531     # rolling statistics plot
532     original = plt.plot(timeseries,
    color='blue', label='Original'
    )
533     mean = plt.plot(rolling_mean,
    color='red', label='Rolling
    Mean')
534     std = plt.plot(rolling_std,
    color='black', label='Rolling
    Std')
535     plt.legend(loc='best')
536     plt.title('Rolling Mean &
    Standard Deviation')
    plt.show(block=False)
537
538     # Dickey Fuller test:
539     result = adfuller(timeseries)
540     print('ADF Statistic: {}'.
    format(result[0]))
541     print('p-value: {}'.format(
    result[1]))
542     print('Critical Values:')
543     for key, value in result[4].
    items():
544         print('\t{}: {}'.format(key
    , value))
545
546
547
548 # In[95]:
549
550 from statsmodels.tsa.seasonal
    import seasonal_decompose
551
552 result = seasonal_decompose(M.Load,
    model='additive')
553 result.plot()
554 plt.show()
555
556
557
558 # In[101]:
559
560 len(S_series),len(RM_Load)
561
562
563
564 # In[102]:
565
566 N = len(M.Load)
567 split = 0.85
568 training_size = round(split*N)
569 test_size = round((1-split)*N)
570
571 S_series = M.Load[:training_size]
572
573
574
575 # In[103]:
576
577 len(RM_Load)
578

```

```

579
580
581 # In[105]:
582
583
584 window = 6
585 rolling_mean = S_series.rolling(
586     window=window).mean()
587 RM_Load = S_series.diff(window)
588 RM_Load.dropna(inplace=True)
589 get_stationarity(RM_Load,window)
590 plt.plot(RM_Load)
591 plt.plot(S_series)
592
593 # In[106]:
594
595
596 validates = range(1,13,1)
597 p_0 = []
598 for v in validates:
599     rolling_mean = S_series.rolling(
600         (window=v).mean()
601     RM = S_series.diff(v)
602     RM.dropna(inplace=True)
603     result = adfuller(RM)
604     p_0.append(result[1])
605
606 # In[111]:
607
608
609 p_value = np.full(len(validates),
610     fill_value= 0.05)
611 plt.plot(validates,p_0,'.',label =
612     'P value')
613 plt.plot(validates,p_value,'--',
614     label= 'Trashold (0.05)')
615 plt.xlabel('Seasonal difference')
616 plt.legend()
617 #plt.ylim(0,0.01)
618
619 # In[644]:
620
621 fig = plt.figure(figsize=(12,8))
622 ax1 = fig.add_subplot(211)
623 fig = sm.graphics.tsa.plot_acf(
624     RM_Load, ax=ax1, lags =25)
625
626
627 # In[645]:
628
629
630 fig = plt.figure(figsize=(12,8))
631 ax3 = fig.add_subplot(311)
632 ax3.plot(RM_Load.diff().dropna().
633     diff().dropna())
634 ax1 = fig.add_subplot(312)
635 fig = sm.graphics.tsa.plot_acf(
636     RM_Load.diff().dropna().diff().
637     dropna(), ax=ax1,lags =25)
638 ax2 = fig.add_subplot(313)
639 fig = sm.graphics.tsa.plot_pacf(
640     RM_Load.diff().dropna().diff().
641     dropna(), ax=ax2,lags=25)
642 fig.tight_layout(pad=3.0)
643
644 # In[121]:
645
646
647 def optimize_SARIMA(parameters_list
648     , d, D, s, exog):
649     """
650     Return dataframe with
651     parameters, corresponding AIC
652     and SSE
653
654     parameters_list - list with
655     (p, q, P, Q) tuples
656     d - integration order
657     D - seasonal integration
658     order
659     s - length of season
660     exog - the exogenous
661     variable
662     """
663
664     results = []
665
666     for param in tqdm_notebook(
667         parameters_list):
668         try:

```

```

658         model = SARIMAX(exog,
        order=(param[0],d, param[1]),
        seasonal_order=(param[2], D,
        param[3], s)).fit(dis=-1)
659     except:
660         continue
661
662     aic = model.aic
663     results.append([param, aic
664 ])
665
666     result_df = pd.DataFrame(
667 results)
668     result_df.columns = ['(p,q)x(P,
669 Q)', 'AIC']
670
671     #Sort in ascending order, lower
672     AIC is better
673     result_df = result_df.
674     sort_values(by='AIC', ascending
675 =True).reset_index(drop=True)
676
677     return result_df
678
679 # In[122]:
680
681 p = range(0, 4, 1)
682 d = 0
683 q = range(0, 4, 1)
684 P = range(0, 4, 1)
685 D = 1
686 Q = range(0, 4, 1)
687 s = 6
688 parameters = product(p, q, P, Q)
689 parameters_list = list(parameters)
690 print(len(parameters_list))
691
692 # In[123]:
693
694 result_df = optimize_SARIMA(
695     parameters_list, d, D, s ,
696     S_series)
697 result_df
698
699 # Again building two models for
700
701 comparison
702 # In[113]:
703
704 best_model = SARIMAX(S_series,
705     order=(3, 2, 1), seasonal_order
706     =(1, 1, 2, 6)).fit(dis=-1)
707 print(best_model.summary())
708 s_best_model = SARIMAX(S_series,
709     order=(3,0, 1), seasonal_order
710     =(1, 1, 2, 6)).fit(dis=-1)
711 print(s_best_model.summary())
712
713 # In[120]:
714
715 best_model.plot_diagnostics(figsize
716     =(15,12));
717
718 # In[115]:
719
720 fore_1= test_size -1
721 forecast = best_model.
722     get_prediction(start=
723     training_size, end=
724     training_size+fore_1)
725 forec = forecast.predicted_mean
726 ci = forecast.conf_int(alpha=0.05)
727
728 s_forecast = s_best_model.
729     get_prediction(start=
730     training_size, end=
731     training_size+fore_1)
732 s_forec = s_forecast.predicted_mean
733 s_ci = forecast.conf_int(alpha
734     =0.05)
735
736 fig, ax = plt.subplots(figsize
737     =(16,8), dpi=300)
738 x0 = M.Load.index[0:training_size]
739 x1=M.Load.index[training_size:
740     training_size+fore_1+1]
741 #ax.fill_between(forec, ci['lower
742     Load'], ci['upper Load'])

```

```

729 plt.plot(x0, M.Load[0:training_size], 'k', label = 'Load')
730
731 plt.plot(M.Load[training_size:
training_size+fore_1], '.k',
label = 'Actual')
732
733 forec = pd.DataFrame(forec, columns
=[ 'f'], index = x1)
734 forec.f.plot(ax=ax,color = '
Darkorange',label = 'Forecast (
d = 2)')
735 ax.fill_between(x1, ci['lower Load'], ci['upper Load'],alpha=0.2,
label = 'Confidence interval
(95%)',color='grey')
736
737 s_forec = pd.DataFrame(s_forec,
columns=[ 'f'], index = x1)
738 s_forec.f.plot(ax=ax,color = 'green',label = 'Forecast (d = 0)')
739 #ax.fill_between(x1, s_ci['lower
Load'], s_ci['upper Load'],
alpha=0.2, label = 'Confidence
interval (95%)',color='grey')
740
741
742 plt.legend(loc = 'upper left')
743 plt.show()
744
745
746
747 # In[116]:
748
749
750 RMSE_Sarima = np.sqrt(
mean_squared_error(M.Load[
training_size:training_size+
fore_1],forec.f[0:len(forec)
-1]))
751 print('RMSE : {:.02f}'.format(
RMSE_Sarima))
752
753
754 # In[117]:
755
756
757 RMSE_Sarima/M.Load.max()
758

```

```

# In[124]:
RMSE_Sarima = np.sqrt(
mean_squared_error(M.Load[
training_size:training_size+
fore_1],s_forec.f[0:len(forec)
-1]))
print('RMSE : {:.02f}'.format(
RMSE_Sarima))

# In[125]:
RMSE_Sarima/M.Load.max()

```

## 4.6 Deconvolution.ipynb

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Deconvolution
5
6 # In[79]:
7
8
9 #Importing the libraries to watch
the 'fits' image and get the
data array
10 import astropy
11 #import plotly.graph_objects as go
12 from astropy.io import fits
13 #Importing a library that is useful
to read the original file
14 import pandas as pd
15 import pylab as plb
16 import matplotlib.pyplot as plt
17 from scipy.optimize import
curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np

```



```

24 import math
25 import seaborn as sns
26 plt.style.use('fivethirtyeight')
27 plt.rcParams['font.family'] = 'sans-
    serif'
28 plt.rcParams['font.serif'] = '
    Ubuntu'
29 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
30 plt.rcParams['font.size'] = 14
31 plt.rcParams['axes.labelsize'] = 12
32 plt.rcParams['axes.labelweight'] = '
    bold'
33 plt.rcParams['axes.titlesize'] = 12
34 plt.rcParams['xtick.labelsize'] =
    12
35 plt.rcParams['ytick.labelsize'] =
    12
36 plt.rcParams['legend.fontsize'] =
    12
37 plt.rcParams['figure.titlesize'] =
    12
38 plt.rcParams['image.cmap'] = 'jet'
39 plt.rcParams['image.interpolation']
    = 'none'
40 plt.rcParams['figure.figsize'] =
    (16, 8)
41 plt.rcParams['lines.linewidth'] = 2
42 plt.rcParams['lines.markersize'] =
    8
43 plt.rcParams["axes.grid"] = False
44
45 colors = ['xkcd:pale orange', 'xkcd:
    sea blue', 'xkcd:pale red', '
    xkcd:sage green', 'xkcd:terra
    cotta', 'xkcd:dull purple', '
    xkcd:teal', 'xkcd: goldenrod',
    'xkcd:cadet blue',
46 'xkcd:scarlet']
47 cmap_big = cm.get_cmap('Spectral',
    512)
48 cmap = mcolors.ListedColormap(
    cmap_big(np.linspace(0.7, 0.95,
    256)))
49 bbox_props = dict(boxstyle="round,
    pad=0.3", fc=colors[0], alpha
    =.5)
50
51
52 # First step we import filtered
    dataset. (wavelet)
53
54 # In[80]:
55
56
57 dataset = pd.read_csv('
    reconstruction.csv')
58 dataset = dataset.drop(columns=['
    Unnamed: 0'])
59
60
61 # In[81]:
62
63
64 data=pd.read_csv('deconv.csv',sep='
    ;')
65
66
67 # In[82]:
68
69
70 data=data.rename(columns={'Data': '
    Day', 'Godzina': 'hour', 'Minuty':
    'minute', 'Wolumen': 'Load'})
71
72
73 # In[83]:
74
75
76 data['Load'] = dataset.R
77
78
79 # In[84]:
80
81
82 data.head()
83
84
85 # Covertng data into DateTime
    objects.
86
87 # In[85]:
88
89
90 data.Day = pd.to_datetime(data.Day)
91
92
93 # In[86]:

```

```

94
95
96 data.Day = pd.to_datetime(data.Day)
97 data['year'] = data.Day.dt.year
98 data['month'] = data.Day.dt.month
99 data['day'] = data.Day.dt.day
100
101
102 # In[87]:
103
104
105 data['month'] = data.Day.dt.month
106
107
108 # In[88]:
109
110
111 data['day'] = data.Day.dt.day
112
113
114 # In[89]:
115
116
117 data['weekday'] = data.Day.dt.day_name()
118
119
120 # In[91]:
121
122
123 data.head()
124
125
126 # Distinguish Holidays from weekdays
127
128 # In[92]:
129
130
131 import holidays
132 pl_holidays = holidays.Poland()
133
134
135 # In[93]:
136
137
138 from datetime import date
139 Days = data.day.tolist()
140 Months = data.month.tolist()

141 Years = data.year.tolist()
142 hd = []
143 for i in range(len(data)):
144     if date(Years[i], Months[i], Days[i]) in pl_holidays :
145         hd.append(1)
146     else :
147         hd.append(0)
148 data['holidays'] = hd
149
150
151 # In[94]:
152
153
154 data.head()
155
156
157 # In[95]:
158
159
160 weekdays = data.weekday.drop_duplicates().tolist()
161 weekdays = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
162
163
164 # Building kernel functions for weekdays and weekend without holidays.
165
166 # In[97]:
167
168
169 Mean = []
170 for w in weekdays:
171     W = data[data['weekday'] == w]
172     W = W[W['holidays'] == 0]
173     W_Day = W.Day.drop_duplicates().tolist()
174     mean = pd.DataFrame()
175     mean = np.zeros(96)
176     k = 0
177     for d in W_Day:
178         value = W[W['Day'] == d].Load
179         try :
180             mean = mean + np.asarray(value)

```

```

181         k = k+1
182     except :
183         continue
184
185     mean = mean/k
186     Mean.append(mean)
187
188
189 # In[98]:
190
191
192 plt.figure(1)
193 plt.subplot(711)
194 plt.plot(Mean[0])
195 plt.subplot(712)
196 plt.plot(Mean[1])
197 plt.subplot(713)
198 plt.plot(Mean[2])
199 plt.subplot(714)
200 plt.plot(Mean[3])
201 plt.subplot(715)
202 plt.plot(Mean[4])
203 plt.subplot(716)
204 plt.plot(Mean[5])
205 plt.subplot(717)
206 plt.plot(Mean[6])
207
208
209 # In[99]:
210
211
212 d=data.Day.drop_duplicates()
213
214
215 # In[101]:
216
217
218 Ferial = np.array(Mean[0]) + np.
219         array(Mean[1]) + np.array(Mean
220         [2]) + np.array(Mean[3]) + np.
221         array(Mean[4])
222
223 Ferial=Ferial/5.
224 Ferial=Ferial/Ferial.sum()
225
226
227 # In[102]:
228
229
230 plt.plot(Ferial),Ferial.mean()
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269

```

```

270 OLD_PROB_D=PROB_D.copy()
271
272
273 # In[107]:
274
275
276 fix_data=data.copy()
277 I=np.arange(0,len(OLD_PROB_D)+1,1)
278 for i in range(len(I)-1):
279     if i%2==0:
280         to_add=fix_data.loc[
281             fix_data[(fix_data.Day==
282                 OLD_PROB_D[I[i+1]]) & (fix_data
283                 .hour==2)].index.tolist()
284                 [0:4]].Load
285         line = fix_data.loc[to_add
286             index[0]:to_add.index[0]+3]
287         start=fix_data[(fix_data.
288             Day==OLD_PROB_D[I[i]]) & (
289             fix_data.hour==1)].index.max()
290         +1
291         try_data=fix_data.copy()
292         try_data=try_data.drop(np.
293             arange(start,len(data)))
294         fix_data=try_data.append(
295             line,ignore_index=True)
296         fix_data.loc[len(fix_data)
297             -4:len(fix_data)].Day=np.repeat
298             (OLD_PROB_D[I[i]],4)
299         fix_data=fix_data.append(
300             data.loc[fix_data.index.max()
301                 -3:len(data)])
302         fix_data=fix_data.drop(line
303             .index)
304
305 # Fix data is used to build Delta
306     Signal
307
308 # In[108]:
309
310
311 data=fix_data
312
313 # In[109]:
314
315 MAX=[]
316
317
318 PROB_D=[]
319 #data=data.reset_index()
320 RANGE=np.arange(0,len(fix_data),96)
321 for t in RANGE:
322     MAX.append(fix_data.loc[t:t
323         +96-1].Load.max())
324
325 # In[110]:
326
327 def build_day(k):
328     day=np.zeros(96)
329     day[0]=k
330     return day
331
332 # In[111]:
333
334 DELTA=[]
335 for m in MAX:
336     DELTA.append(build_day(m))
337 #DELTA=np.abs(DELTA)
338
339 # In[112]:
340
341 DELTA=np.array(DELTA).ravel()
342
343 # In[118]:
344
345 data['r']=DELTA
346
347 # In[119]:
348
349 fix_data['r']=DELTA
350
351 # In[143]:
352
353 plt.plot(np.array(fix_data.r
354     [96:96*30]),'Darkorange', label

```

```

    = 'Delta signal')
350 plt.plot(np.array(fix_data.Load
    [96:96*30]), 'k', label = 'Time
    Series')
351 plt.grid(True)
352 plt.ylabel('Load')
353 plt.xlabel('Time')
354 plt.xticks(np.arange(96,96*30,96),
    np.arange(0,30))
355 plt.legend()
356 plt.show()
357
358
359 # In[121]:
360
361
362 data=fix_data
363
364
365 # In[122]:
366
367
368 YEARS=data.year.drop_duplicates().
    tolist()
369
370
371 # Each specific day of the months
    for each year has been stored
372 # e.g.
373 # The first Monday of the third
    week of May
374 # the second Friday of the first
    week of April
375
376 # In[123]:
377
378
379 TOT=[]
380 for m in range(1,13):
381     DAYS=[]
382     for d in weekdays:
383         mon=data[data['weekday']==d
    ]
384         W=[]
385         J=np.arange(0,96*7,96)
386         K=[]
387         for j in range(len(J)):
388             j_rel=np.zeros(96)
389             k=0
390
    for y in range(len(
    YEARS)-1):
391         try:
392             add=np.array(
    mon[(mon['month']==m)&(mon.year
    ==YEARS[y])].reset_index().loc[
    J[j]:J[j]+95].r)
393             j_rel=j_rel+add
394             k=k+1
395         except:
396             continue
397         K.append(k)
398         W.append(j_rel/k)
399         DAYS.append(W)
400
401         TOT.append(DAYS)
402
403
404 # The calendar of 2016 has been
    buit
405
406 # In[144]:
407
408
409
410 cal=pd.DataFrame({'Day':data['Day'
    ],'month':data['month'],'year':
    data['year'],'Weekday':data['
    weekday']})
411 cal['hour']=data.hour
412 cal['minute']=data.minute
413 cal=cal[cal['year']==2016]
414
415
416 # In[145]:
417
418
419 NEW_Z=[]
420 for k in range(1,13):
421     mese=cal[cal['month']==k]
422     mese=mese.reset_index().drop(
    columns=['index'])
423     z=[]
424     for j in range(1,6):
425         sett=mese.loc[(j-1)
    *7*96:96*(j)*7]
426         #print(j-1)
427         ZERO=np.zeros(len(sett)-1)+

```

```

429     j-1
430     z.append(ZERO)
431     new_z=[]
432     for i in z:
433         for j in i:
434             new_z.append(j)
435     new_z=new_z[0:len(mese)]+[new_z[
436     [len(new_z)-1]]
437     NEW_Z.append(new_z)
438
439 # In[146]:
440
441
442
443 ZERO_FIN=[]
444 for i in NEW_Z:
445     for j in i:
446         ZERO_FIN.append(j)
447
448
449 # In[147]:
450
451
452 cal['n']=ZERO_FIN
453
454
455 # In[148]:
456
457
458 tot_cal=cal.drop(columns=['hour','
459     minute'])
460 tot_cal=tot_cal.drop_duplicates(
461     subset='Day')
462
463
464
465 tot_cal=tot_cal.reset_index()
466
467
468 # In[149]:
469
470
471 def week_to_number(day):
472     if day=='Monday':
473         i=0
474
475     if day=='Tuesday':
476         i=1
477     if day=='Wednesday':
478         i=2
479     if day=='Thursday':
480         i=3
481     if day=='Friday':
482         i=4
483     if day=='Saturday':
484         i=5
485     if day=='Sunday':
486         i=6
487     return i
488
489 # In[151]:
490
491
492 d_n=[]
493 for day in tot_cal.Weekday.tolist():
494     :
495     d_n.append(week_to_number(day))
496
497 # In[152]:
498
499
500 tot_cal['d_n']=d_n
501
502
503 # The model of each day has been
504     used to create a synthetic 2016
505
506 # In[153]:
507
508
509 SIGNAL=np.array([])
510 for m in range(1,13):
511     mon=tot_cal[tot_cal['month']==m
512     ]
513     mon=mon.reset_index()
514
515     for i in range(len(mon)):
516         dn=int(mon.loc[i]['d_n',
517         ])
518         N=int(mon.loc[i]['n'])
519         SIGNAL=np.append(SIGNAL
520         ,TOT[m-1][dn][N])

```

```

518 # Debugging the split years
519 # In[154]:
520
521 bis_1=data[data.year==2008]
522 bis_2=data[data.year==2012]
523 first=np.array(bis_1[bis_1['Day']=='2008-02-29'].r)
524 second=np.array(bis_2[bis_2['Day']=='2012-02-29'].r)
525 bis=(first+second)/2.
526
527 # In[155]:
528
529 a=np.argwhere(np.isnan(SIGNAL)).
530     ravel().min()
531 b=np.argwhere(np.isnan(SIGNAL)).
532     ravel().max()+1
533 SIGNAL[a:b]=bis
534
535 # In[157]:
536
537 OLD_SIGNAL=SIGNAL.copy()
538
539 # In[158]:
540
541 SIGNAL=np.abs(OLD_SIGNAL)
542
543 # In[159]:
544
545 Ferial = np.array(Mean[0]) + np.
546     array(Mean[1]) + np.array(Mean
547         [2]) + np.array(Mean[3]) + np.
548     array(Mean[4])
549 Ferial=Ferial/5.
550 Ferial=Ferial/Ferial.sum()
551 Old_Ferial=Ferial.copy()
552
553 # In[160]:
554
555 Ferial = np.array(Mean[0]) + np.
556     array(Mean[1]) + np.array(Mean
557         [2]) + np.array(Mean[3]) + np.
558     array(Mean[4])
559 Ferial=Ferial/5.
560 Ferial=Ferial/Ferial.sum()
561 Ferial=Ferial+Ferial.mean()-Ferial.
562     min()-Ferial.max()
563
564 # In[161]:
565
566 Ferial=Ferial/5.
567
568 # In[162]:
569
570 Ferial=Ferial/Ferial.sum()
571
572 # In[163]:
573
574 Ferial=Ferial+Ferial.mean()-Ferial.
575     min()-Ferial.max()
576
577 # Performing the convolution with
578     Ferial days
579
580 # In[166]:
581
582 X=signal.convolve(SIGNAL,Ferial)
583 X=X*(data.Load.max())/X.max()
584
585 # In[167]:
586
587 test=np.array(data[data.year
588     ==2016].Load)
589
590
591

```

```

602 # In[168]:
603
604
605 X=X[0:len(X)-95]
606
607
608 # In[169]:
609
610
611 plt.plot(X[60*96:90*96],color='blue',
612          ' ')
613 plt.plot(np.array(test)
614          [60*96:90*96],color='red')
615 #error=np.array(test)[12*96:50*96]-
616         original
617 #plt.plot(np.abs(error),color='
618         black')
619
620
621 # Building convolution for weekend
622         days
623
624 # In[170]:
625
626
627
628 start=20*96
629 end=(20+7*40)*96
630 prev=X[start:end]
631 t=np.arange(0,len(prev),1)
632
633
634 # In[173]:
635
636
637
638 sat= np.array(Mean[6])
639 sat=sat-sat.min()
640 sat=sat/sat.sum()
641 sat=sat-sat.mean()
642
643
644 # In[204]:
645
646
647 sun= np.array(Mean[6])
648 sun=sun-sun.min()
649 sun=sun/sun.sum()
650 sun=sun-sun.mean()
651 sat= np.array(Mean[5])
652 sat=sat-sat.min()
653
654
655 sat=sat/sat.sum()
656 sat=sat-sat.mean()
657 plt.plot(Ferial,'k',label = 'Ferial
658         ')
659 plt.plot(sun,'red',label = 'Sunday'
660         )
661 plt.plot(sat,'Darkorange',label = '
662         Saturday')
663 plt.legend()
664 plt.grid(True)
665 plt.ylabel('Kernel Function')
666 plt.xlabel('Time [Hours]')
667 plt.xticks(np.arange(0,96,4),np.
668         arange(0,24))
669 plt.show()
670
671 # Prediction using weekend days
672         kernels for entire calendar.
673
674 # In[175]:
675
676
677
678 SUN=signal.convolve(sun,SIGNAL)
679 SUN=SUN*data[data.year!=2016].Load.
680         max()/SUN.max()
681 SAT=signal.convolve(sat,SIGNAL)
682 SAT=SAT*data[data.year!=2016].Load.
683         max()/SAT.max()
684
685
686 # In[176]:
687
688
689
690 plt.plot(SAT[60*96:90*96])
691 plt.plot(test[60*96:90*96])
692
693
694 # Dataset of the 3 Kernels
695         reconstruction
696
697 # In[ ]:
698
699
700
701 recons_data=data[data['year'
702         ]==2016].copy()
703 recons_data['PSFFerial']=X[0:len(
704         recons_data)]
705 recons_data['PSFSunday']=SUN[0:len(

```



```

        recons_data)]
685 recons_data['PSFSaturday']=SAT[0:
        len(recons_data)]
686
687
688 # In[178]:
689
690 recons_data
691
692
693
694 # In[180]:
695
696
697 psf_data=pd.DataFrame()
698 psf_data['weekday']=recons_data['
        weekday']
699 psf_data['Ferial']=recons_data.
        PSFFerial
700 psf_data['Sunday']=recons_data.
        PSFSunday
701 psf_data['Saturday']=recons_data.
        PSFSaturday
702 psf_data=psf_data.reset_index()
703
704
705 # In[182]:
706
707
708 final_recons=np.array(psf_data.
        Ferial)
709 for i in range(35136):
710     day_data=str(psf_data.loc[i].
        weekday)
711     if day_data=='Sunday':
712         final_recons[i]=psf_data.
        loc[i].Sunday
713     if day_data=='Saturday':
714         final_recons[i]=psf_data.
        loc[i].Saturday
715
716
717
718 # Final Prediction
719
720 # In[226]:
721
722
723 plt.plot(final_recons[60*96:90*96]
        color='Darkorange',label='Model
        ')
724 plt.plot(test[60*96:90*96],color='k
        ',label = 'Data')
725 plt.grid(True)
726 plt.legend()
727 plt.xticks(np.arange(0*96,31*96,96)
        ,np.arange(0,31))
728 plt.xlabel('Days [March 2016]')
729 plt.ylabel('Load')
730
731
732 # In[214]:
733
734
735 from sklearn.metrics import
        mean_squared_error
736
737
738 # In[216]:
739
740
741 finaldata=data[data['year']==2016].
        copy()
742 #finaldata['Prediction']=
        final_recons
743
744
745 # In[217]:
746
747
748 finaldata.to_csv('pred.csv')
749
750
751 # In[218]:
752
753
754 finaldata=pd.read_csv('pred.csv')
755
756
757 # In[1158]:
758
759
760 test=np.array(finaldata.Load)
761 pred=np.array(final_recons)
762
763
764 # In[29]:
765

```

```

766
767 from sklearn.metrics import
    mean_squared_error as mse
768
769
770 # In[41]:
771
772
773 PERF=[np.corrcoef(test-pred,test)
    [0][1], np.sqrt(mse(pred
    [0*96:30*96],test[0*96:30*96]))
    ,np.sqrt(mse(pred[30*96:],test
    [30*96:]))/test[30*96:].max()
    ]
774
775
776 # In[45]:
777
778
779 print ('The correlation between the
    error and the original signal
    is '+str(PERF[0]))
780 print ('The RMSE for the first
    month of data is ' + str(PERF
    [1]))
781 print ('This RMSE is the '+str(PERF
    [2]*100) +'% of the maximum')

```

# Bibliography

- [1] Del Moro Dario, De Gasperis Giancarlo. *Lezione 2 - The Nyquist-Shannon sampling theorem*. 2020. URL: [http://people.fisica.uniroma2.it/~solare/en/wp-content/uploads/2018/11/Lez\\_2.pdf](http://people.fisica.uniroma2.it/~solare/en/wp-content/uploads/2018/11/Lez_2.pdf).
- [2] Del Moro Dario, De Gasperis Giancarlo. *Lezione 7- Wavelets*. 2020. URL: [http://people.fisica.uniroma2.it/~solare/en/wp-content/uploads/2018/11/Lez\\_7\\_Wavelets.pdf](http://people.fisica.uniroma2.it/~solare/en/wp-content/uploads/2018/11/Lez_7_Wavelets.pdf).
- [3] Piero Paialunga. “Hey wavelet, turn that noise down!” In: *Towards Data Science* (2020). URL: <https://towardsdatascience.com/hey-wavelet-turn-that-noise-down-80a30796dfcf>.
- [4] Piero Paialunga. “If history repeats itself, Fourier Transform is a key”. In: *Towards Data Science* (2020). URL: <https://towardsdatascience.com/if-history-repeats-itself-fourier-transform-is-a-key-a593ddfa246e>.
- [5] Wikipedia contributors. *Chi-squared test* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 4-November-2020]. 2020. URL: [https://en.wikipedia.org/w/index.php?title=Chi-squared\\_test&oldid=983024096](https://en.wikipedia.org/w/index.php?title=Chi-squared_test&oldid=983024096).
- [6] Wikipedia contributors. *Hann function* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 30-October-2020]. 2020. URL: [https://en.wikipedia.org/w/index.php?title=Hann\\_function&oldid=970339234](https://en.wikipedia.org/w/index.php?title=Hann_function&oldid=970339234).
- [7] Wikipedia contributors. *P-value* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 4-November-2020]. 2020. URL: <https://en.wikipedia.org/w/index.php?title=P-value&oldid=979081903>.