TÉCNICO LISBOA

Data Analitics for Smart Grids

MEEC

Lab 4 - Time Series

Authors:

Rodrigo Contreiras (90183) rodrigo.contreiras@tecnico.ulisboa.pt
Gonçalo Aniceto (96218) goncalo.aniceto@tecnico.ulisboa.pt

2022/2023 – 2nd Semester, P3

Contents

1 Problem description

The electric grid is a complex and dynamically structured system that delivers electricity from power plants to consumers. The grid is comprised of a network of power generation facilities, transmission lines, substations, transformers that work together to ensure a reliable and efficient supply of electricity, dependent on meteorological conditions.

Time can play a significant role, when analysing how the electrical grid is behaving to these requests. Patterns, trends, and connections between the data start to appear over time. Measurements that were once independent can now be considered to be intertwined. Time's inherent autocorrelation can produce biased parameter estimates and incorrect forecasts. Consequently, a variety of methods, such as differencing, autoregressive models, and moving average models, can be used to address it.

2 Objectives

Consider a set of stochastic variables X and an electrical quantity y dependent on such variables for which timeseries data are available (both for the variables and the quantity) and discover an accurate predictor for y based on the forecasts of X.

3 Time series problems

3.1 Initial Problem

To evaluate how time series impact the electrical grid, the previous "kite" grid will be analyzed, but with the addition of power injection (from where a huge wind farm is connected) on bus 1.

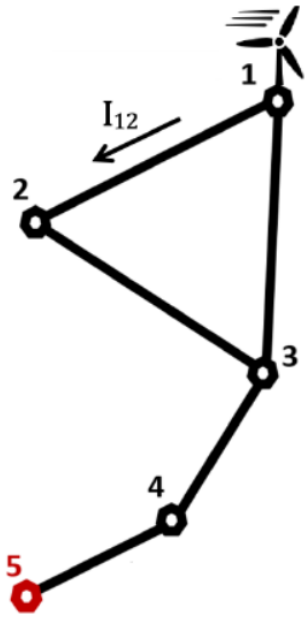

Figure 1: "Kite" grid with power injection on bus 1.

Predictors and forecasts will be made for the current flowing in line 1-2 using real-time measurements of the active power injection in bus 1 (P^1_{inj}) and the magnitude (and sign) of the current flowing there, I_{12}. To solve this problem, we began by using the same methodology as the earlier projects. Considering $y(t) = I_{12}(t)$ and $x(t) = P^1_{inj}(t)$ The results of an **OLS regression** that maps the power flow onto the current flow are as follows:

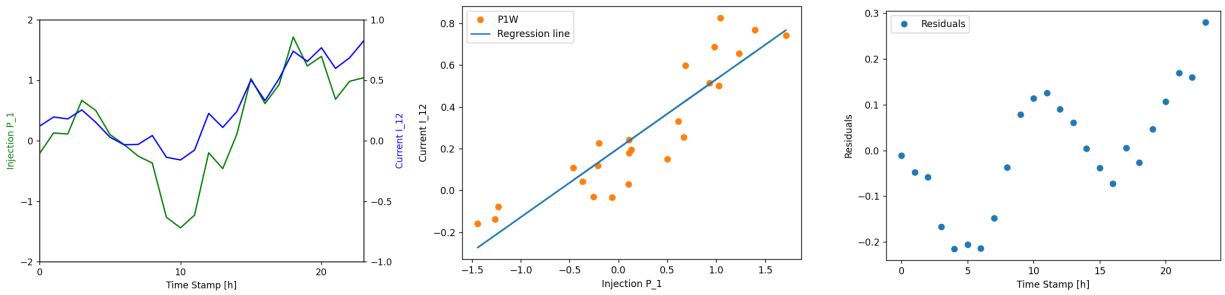

Figure 2: (a) Active power injection in the wind farm connection bus, P_{1inj}, and corresponding current flow magnitude (with sign) in the outgoing line, I_{12}, versus time; (b) the injected power versus the current flow with the ordinary least squares regression line overlaid; (c) scatter plot of the residuals versus time.

By analyzing the residuals of the **OLS regression**, it is clearly visible a pattern. This is suggestive of significant autocorrelation in the errors. To support this subjective approach, we can conduct statistical methods to confront our hypothesis.

The Durbin-Watson test, which assesses the correlation between the residuals in a regression analysis, is the one that we'll employ. Its test statistic spans the range of 0 to 4, with 2 denoting a lack of autocorrelation. Positive autocorrelation is indicated by numbers below 2 whereas negative autocorrelation is suggested by values over 2. There is no substantial autocorrelation if the statistic is between 1.5 and 2.5, but significant autocorrelation is present if the statistic is outside of that range. Its implementation is as follows:

$$DW = \frac{\sum_{t=2}^{T}(\epsilon_t - \epsilon_{t-1})^2}{\sum_{t=1}^{T}\epsilon_t^2} \tag{1}$$

The value of Durdin-Watson (DW) was: 0.1973, proving that our concern was correct and the data values are indeed positively autocorrelated. Therefore, if we use the obtained model to forecast the current flow in line 1-2, it's likely that the prediction won't be very accurate, as seen on the following figure:

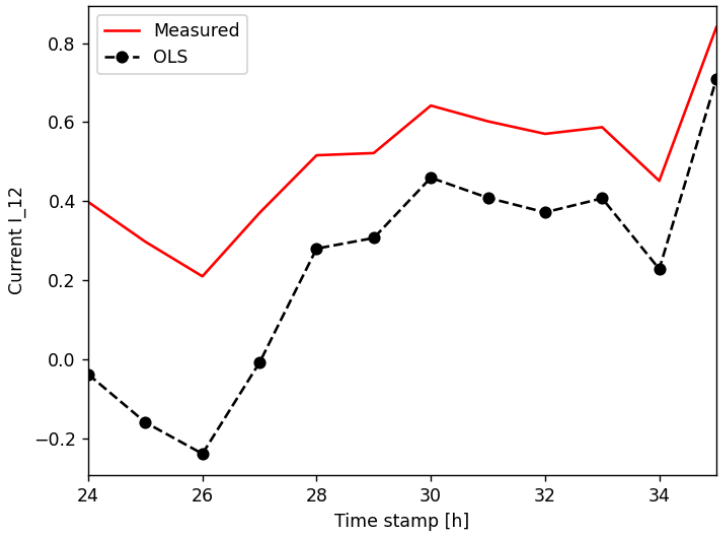

Figure 3: 12h-ahead projection for current flow I_{12}.

As a result, in order to produce better models, we must discover a technique to cope with this autocorrelation. When autocorrelated error terms are present, three basic remedial options are the addition of predictor variables to the regression model, the use of transformed variables or the use of autoregressive models.

3.2 Addition of Predictor Variables

3.2.1 Sum of Loads and Sum of Voltages

One of the primary causes of autocorrelated error terms is the absence of one or more significant predictor variables with time-ordered effects on the response variable from the model. Because several significant dynamic variables, such as the loads, are not explicitly addressed on the preceding model, it cannot be anticipated to perform well in projecting the future current flows if the wind forecast is correct and the residuals from the linear regression are small.

Note that the sum of voltages were chosen as a predictor variable in an effort to incorporate injection dynamics. In fact, if we consider constant power factor, $cos(\phi)$, reactive power consumption will have the same auto-correlated dynamic as the active power, $Q(t) \propto P(t)$. Since the grid is mostly inductive, nodal voltage drop is directly related to the reactive load, $\Delta V \approx X\Delta Q$, and the sum of injection's dynamics could be expressed on sum of voltage (drops):

$$\sum P(t) \propto \sum Q(t) \propto \sum 1 - V_k(t) \propto \sum V_k(t)$$

To evaluate this metric, we chose to repeat the **OLS regression** while including additional variables in each time period that would assist to better capture the network structure. The results were as follows:

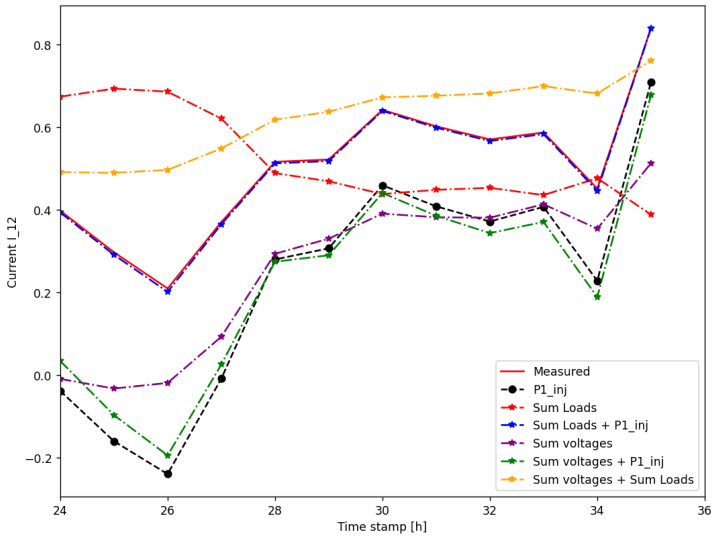

Figure 4: 12h-ahead projection for current flow I_{12} with different predictor variables.

When we only take into account the P^1_{inj} as before, which is represented by the **black dashed line**, it strongly matches the actual measures, as can be seen in figure 4. The results were very poor when we only took into account the total loads (P^1_{inj} included), as shown by the **red dashed line**. However, the **blue dashed line**, which almost perfectly predicts the current flow in lines 1-2, illustrates that the best results were obtained when using both of these variables as independent predictors together.

This shows that the P^1_{inj} is crucial if we want to forecast the current flow, but it also shows that other variables are required to accurately fit the model to the data. Even though the sum of all

loads alone didn't seem to offer any useful insight into how the grid operates, when added to P^1_{inj}, it provided useful complementing information that helped to predict the model perfectly.

The voltages, however, do not fit this description. When we add up all of the voltages, the results first appear optimistic, as shown by the **purple dashed line**, which somewhat mimics the measures. However, the **green dashed line** indicates that the results are worse when we combine it with P^1inj than when we use P^1inj alone. This demonstrates that the voltages don't offer helpful features that could supplement those offered by P^1_{inj}. Lastly by looking at **orange dashed line**, is clearly visible that the sum of loads and the sum of voltages, without P^1_{inj} can't forecast the current flow.

Lastly when analysing the residuals from all these methods, as well as the values for Durbin-Watson:

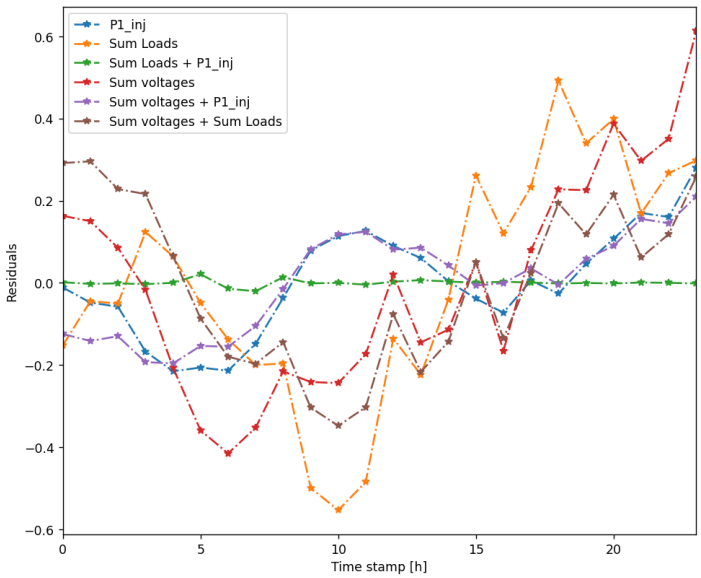

Figure 5: Scatter plot of the residuals versus time for the different models.

	P1_inj	**Sum Loads**	**Sum Loads + P1_inj**	**Sum voltages**	**Sum voltages + P1_inj**	**Sum voltages + Sum Loads**
DW	0.23	0.33	2.4	0.27	0.15	0.37

Table 1: Durdin-Watson values for the different models.

It is visible that all of the models have a trend that can be seen to indicate the autocorrelation, with the exception of the one represented by the **green trend-line**. The Durbin-Watson statistic makes it clear that once more, our presumptions were accurate. Every model still exhibits a positive autocorrelation, with the exception of the one that utilizes both the total number of loads and the power injected onto bus 1. The single exception has a DW value about 2.4, which is within acceptable bounds, and it is possible to conclude that there is no substantial autocorrelation in this case.

3.2.2 Adjacent Branch Currents

Kirchhoff's Current Law (KCL), states that the sum of all currents entering and exiting a node must equal zero. As a result, we may use the nearby currents that flow between the adjacent nodes of the line we wish to measure as predictor variables. Therefore, we developed a model that utilized the sum of the currents flowing through lines 1-3 and 2-3. With the same data has before, we obtained the following result:

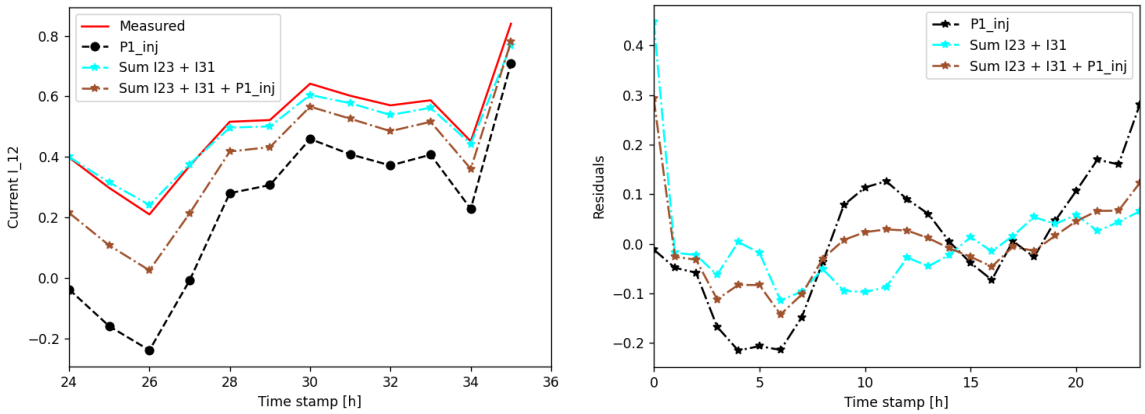

Figure 6: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

In addition to Figure 6, the **Durbin-Watson** test returned 0.9 when only the sum of the currents was considered and 0.7 when the P^1_{inj} was also considered, indicating that it was successful in minimizing autocorrelation in the residuals. According to the results, the currents utilized as predictors are fairly indicative of the current flow between lines 1-2. But, once again, we may presume that it adds no useful information to the one provided by P^1_{inj}.

3.2.3 Adjacent Nodal Voltages

Autocorrelation in injected power on an electrical grid can cause fluctuations in voltage magnitude, $\Delta V \approx X\Delta Q + R\Delta P$, and angles, $\Delta\theta \approx X\Delta P - R\Delta Q$, which can result in voltage autocorrelation. Therefore, we will evaluate if the voltage drop and phase difference between bus 1 and 2 are good predictor variables. With the same data has before, we obtained the following result:

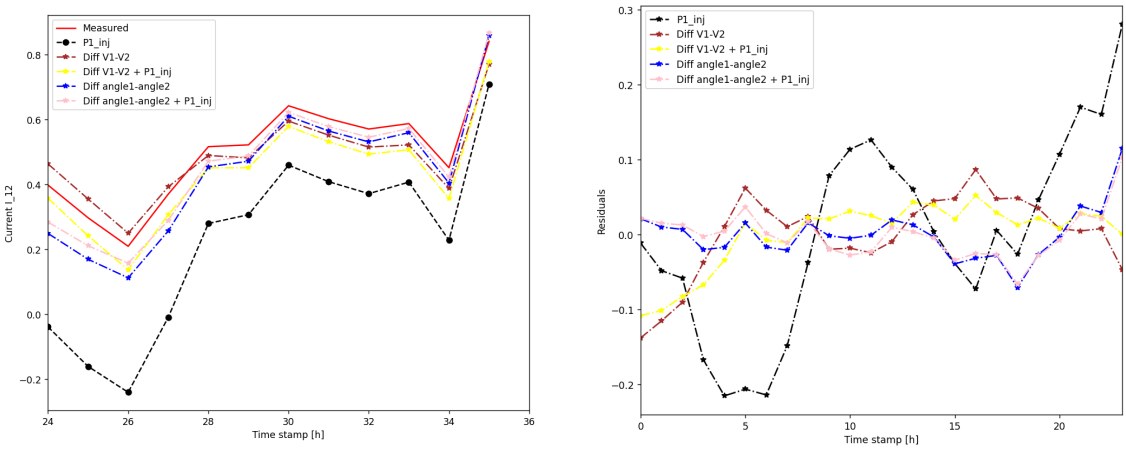

Figure 7: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

In this example, using all of the predictor variables results in a far more accurate forecast than only using P^1_{inj}. To determine which model performed the best, we calculated the **sum of the squared errors**, which yielded:

	Diff V1-V2	**Diff V1-V2 + P1_inj**	**Diff angle1-angle2**	**Diff angle1-angle2 + P1_inj**
RSS	0.0695	0.0471	0.0275	0.0239

Table 2: Sum of the squared errors for the forecast of each model.

As seen in **Table 2**, the results for the **RSS** for each model are quite small, as expected. Also, by adding P^1_{inj}, to complement both the magnitude and angles of the voltages on the models, the forecasts where improved, showing that the information supplied was relevant. Another conclusion we may draw is that using **angles** instead of **magnitudes** of the voltages results in a more accurate forecast. This makes sense because the electrical grid under consideration is more **inductive** than **reactive** and due to the high power factor, active power changes will be bigger then reactive, $\Delta P > \Delta Q$. As a result, angle variation will be bigger than magnitude variation, $\Delta\theta \approx X\Delta P > \Delta V \approx X\Delta Q$, providing more meaningful information about grid dynamics.

3.3 Use of Transformed Variables

Remedial measure based on transformed variables should only be used when using more predictor variables does not solve the issue of autocorrelated errors. We generated three separate models to test each method. We decided to use P^1_{inj} because it is the baseline of our case study, **sum of loads** because it didn't provide much information about grid dynamics in previous examples and we want to see if we can improve it, and **sum of I23 and I31** because this variables forecast almost perfectly the current flow between line 1-2. We also plot the **OLS regression** forecast, for P^1_{inj}, to compare the results.

3.3.1 Cochrane-Orcutt Procedure

In the Cochrane-Orcutt approach, the initial regression model is estimated using the ordinary least squares (OLS) method, and the residuals from the original regression model are computed. The residuals and the Durbin-Watson statistic are then used to estimate the autocorrelation parameter.

The data is then transformed using a first-order autoregressive (AR1) model utilizing the calculated autocorrelation parameter. The modified data are then used to re-estimate the regression model. Until the calculated autocorrelation parameter converges to a stable value, these stages are repeated. Since most models converge within three iterations, the procedure's maximum number of iterations is commonly set at three. Further iterations may raise the danger of the model being overfit to the data and are unlikely to meaningfully enhance estimates of the model parameters. Consequently, depending on the particular properties of the data and model being estimated, the maximum number of iterations can be changed. With the same data has before, we obtained the following result:

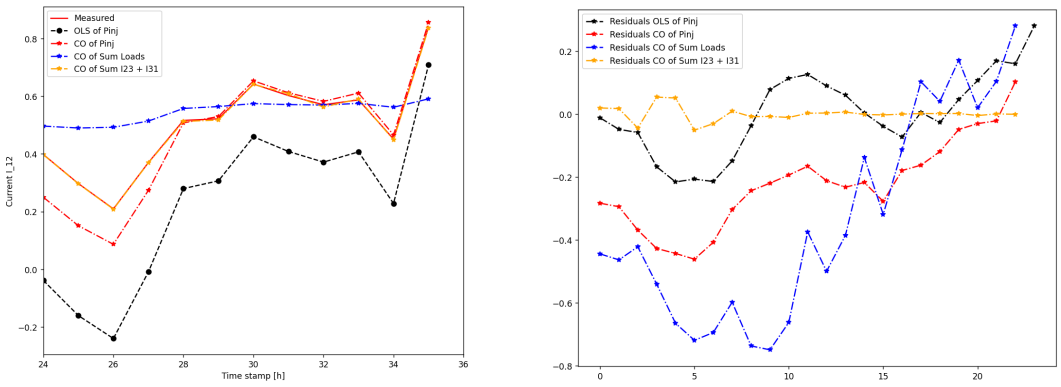

Figure 8: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

The **Cochrane-Orcutt** strategy gave accurate forecasts for the current flow as well as a good way to eliminate the effects of autocorrelation for P^1inj and **sum of I23 and I31** models in this case. The fact that the red-dashed line in figure 8 is far closer to the red line than the black-dashed line and that the orange-dashed line almost completely fits the red line provides proof for this. Moreover, while the **sum of loads** is still far from accurately predicting the current flow, the **sum of square errors** improved from when this method was not utilized, showing that it works as intended in enhancing the model's accuracy. Lastly, the **Durbin-watson** values, for all the models got closer to

the desired value (which is 2), and regarding the model **sum of I23 and I31**, the value was 2.2, which corresponds to a forecast with no autocorrelation on the residuals.

3.3.2 Hildreth-lu Procedure

The Hildreth-lu Procedure, applies repeatedly the **OLS regression** to:

$$y_t - \rho y_{t-1} = \beta_0(1-\rho) + (X_t - \rho X_{t-1})\beta + w_t \tag{2}$$

For different values of ρ between -1 and 1, one chooses the best estimate $\hat{\rho}$ that produces an auxiliary regression, (β_0, β), corresponding to the **smallest residual sum of squares (RSS)**. Note that **SSE** as a function of ρ is quite stable in a wide region around the minimum, as is often the case. It indicates that the numerical search for finding the best value of ρ need not be too fine unless there is particular interest in the intercept term β_0, since the estimate of β_0 is sensitive to the value of $\hat{\rho}$. With the same data has before, we obtained the following result:

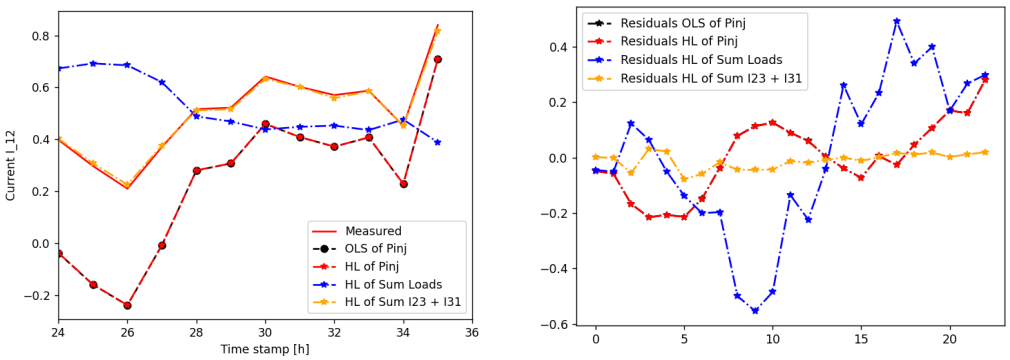

Figure 9: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

The **Hildreth-lu Procedure** generated some intriguing outcomes in this situation. The orange-dashed line indicates that this strategy was effective in eliminating autocorrelation for the model that employs the **sum of I23 and I31**. The *rho* value used on this model, was -1, which reflects that the previous values for the current are fundamental to the forecast.

Unfortunately, it had no effect on the other two models. The reason for this is that in both cases, the **smallest residual sum of squares** was close to the one obtained by the **OLS Regression**. As a result, the value assigned to *rho* was close to zero. This means that this technique almost didn't give any weight to the preceding measurements and, as a result, didn't remove the residual autocorrelation, thereby turning itself into a **OLS regression**.

3.3.3 First Differences Procedure

Differences Procedure are obtained by subtracting the value of a time series at a given time point from its value at the previous time point. By eliminating the trend component, non-stationary time series can be converted into stationary ones using this method. With the same data has before, we obtained the following result:

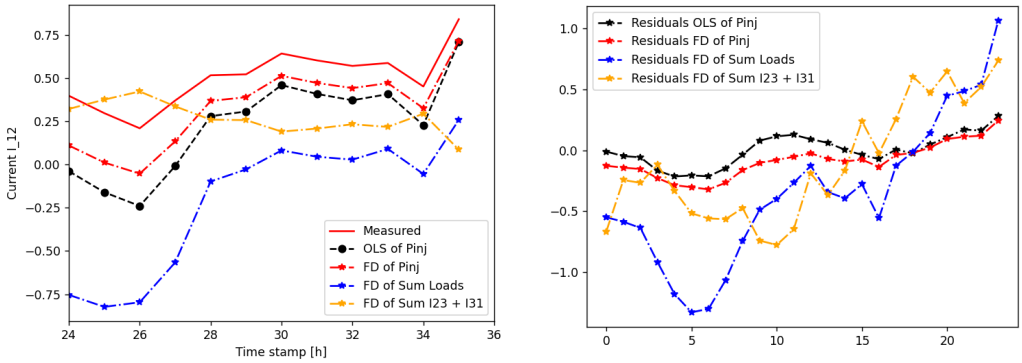

Figure 10: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

Something interesting happened again in this case with **First Differences Procedure**. The model that employed the **sum of I23 and I31** is now the poorest forecaster. We can infer that there is perfect negative autocorrelation because ρ = **-**1 on **Hildreth-lu Procedure**. When a time series has perfect negative autocorrelation, the values are perfectly predicted based on their prior values, and the series has no random variation. In such a circumstance, the original time series cannot be made stationary using the first differences approach, and the resulting results will be unsatisfactory.

The same doesn't happen with the other two models. Because there is no perfect autocorrelation, this method can capture the trend in residuals and improve the models. As seen by the red-dashed line, it is closer to the red line than the black. This strategy was quite successful in terms of the **sum of loads**. Despite the fact that the forecast is the furthest from the actual measured current flow (red line), this was the first method capable of attenuating the trend and hence the autocorrelation from the residuals. The forecasted trend follows the real one and its' accuracy could be increased with the use of a regression model, such as the **Ridge or Lasso regressors**.

3.3.4 Autoregressive model AR(1)

An autoregressive (AR) model is a time series model that ties a **current observation** of a time series to a linear combination of the series' **previous observations**. The amount of historical observations included in the model determines the rank of the AR model.

An AR(1) model is a sort of autoregressive model that uses the time series' first-order lag as a predictor. To put it another way, an AR(1) model expresses the current time series observation as a linear combination of the **preceding observation** multiplied by a **constant factor** (the autoregressive coefficient). This factor determines the strength and direction of the relationship between the current observation and the previous observation. If is positive, then a higher value of the previous

observation leads to a higher value of the current observation, and vice versa. If is negative, then a higher value of the previous observation leads to a lower value of the current observation, and vice versa. The magnitude of the factor determines the strength of the relationship. With the same data has before, we obtained the following result:

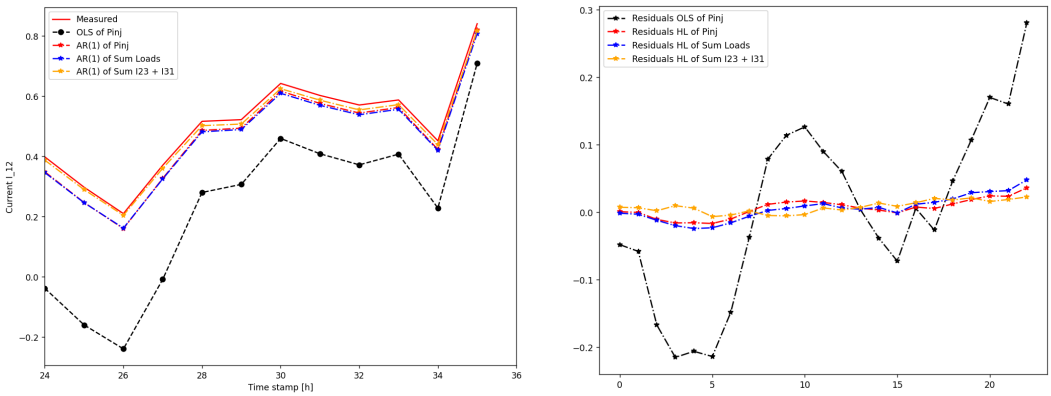

Figure 11: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

By examining Figure 11, we can see that by far the best results were obtained by treating this data as AR(1). The current observation of the time series is highly correlated with the prior observation, according to the results. Taking this autocorrelation into account by include the preceding observation as a predictor in the model, greatly improves the model's accuracy when estimating the current flow between lines 1-2.

It should be noted that these results are quite good due to the significant degree of autocorrelation between consecutive data. If that were not the case, this strategy would not perform as well, as shown in the image below:

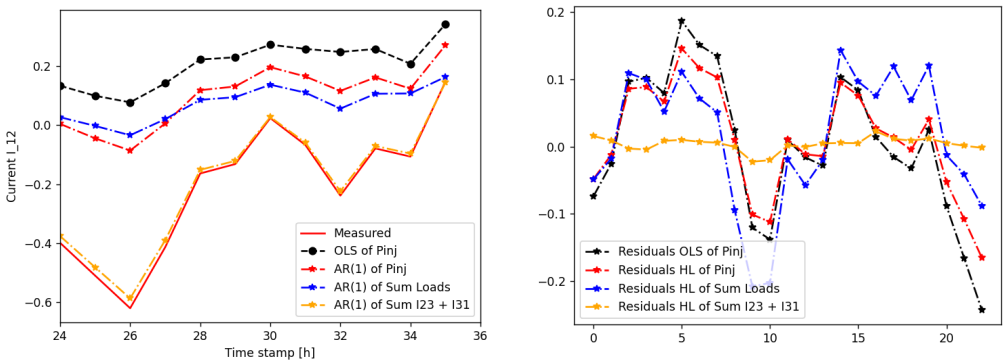

Figure 12: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

By introducing a bigger error on the model, the autocorrelation between two consecutive measures decreases. As a result, the AR(1) performance's suffers because the data is no longer as indicative

of that state as it once was. This produces results similar to earlier methods, in which the best performing model has the predictor variables that are most representational of the dynamics of the electric grid, in this case the **sum of the currents I23 and I31**.

3.4 Testing on other grids

3.4.1 Radial Grid

In terms of power flow, a meshed grid is considerably more complex to assess than a radial grid. This section will examine whether the same applies to the forecast of time series data. Considering the suggested "kite" grid, the radial grid obtained by disconnecting buses 1 and 3. Again, the power from the wind farm is being injected onto bus 1 and the current flow we want to measure is the one between line 1-2. First we tested with just the P^1_{inj}, to see how it would respond:

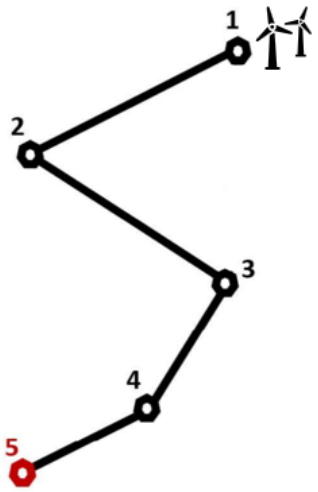

Figure 13: Radial grid with power injection on bus 1.

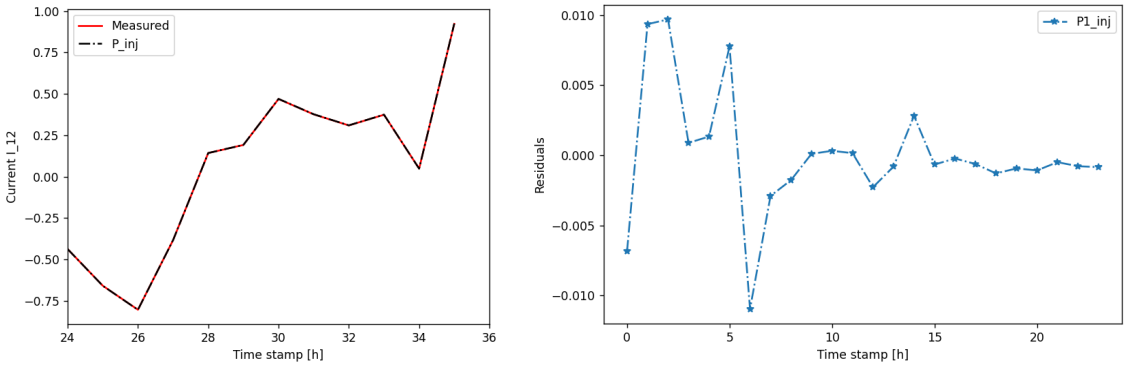

Figure 14: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

Figure 14 clearly demonstrates that accurate forecasting is considerably easier as expected. The model successfully predicted the current flow between lines 1-2 with remarkable precision using solely the information regarding the power generated by the wind farm. This was achieved by utilizing data that could not produce any good result in the previous examples, even though there is as much autocorrelation between consecutive measurements.

Since, the results achieved with the simplest data was satisfactory, no more tests where made, and we can conclude that it is indeed easier to forecast time series that on radial grids, in the presence of autocorrelation between measurements.

3.4.2 IEEE-33 Grid

We also wanted to test time series data forecasting on a larger network. We want to see if the same predictor variables used on the "kite" grid network perform similarly. To do so, we implemented a network grid with 33 buses. The wind farm's power was introduced on bus 22, and the current flow that we want to forecast is on lines 19-20.

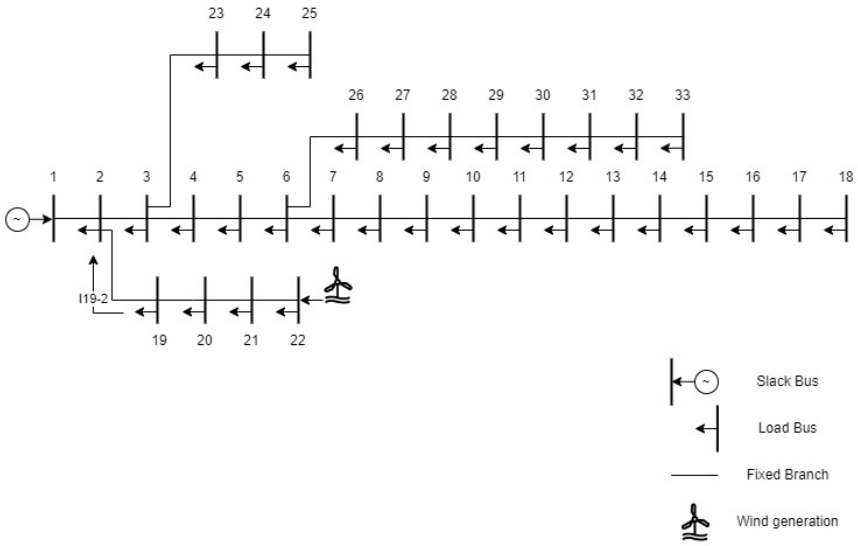

Figure 15: IEEE-33 grid with opened switchable branches and power injection on bus 22.

In order to compare the results appropriately, the data for this network was also AR(1), with the same degree of magnitude as the previous cases. We got the following results:

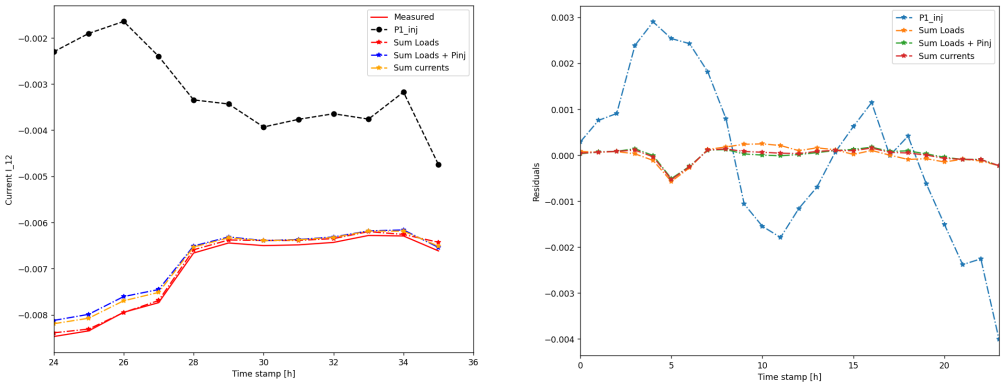

Figure 16: 12h-ahead projection for current flow I12 with different predictor variables (on the left) and Scatter plot of the residuals versus time for the different models (on the right).

From figure 16, it is possible to understand that some predictor variables perform similarly on larger networks, while others don't.

One of the ones that behave similarly is the **sum of adjacent currents**, illustrated by the orange-dashed line and consisting of the currents between lines 1-2, 2-3, and 20-19. This predictor performed admirably (considering the network's size), as it did on the other networks. While considering the electrical fundamentals, this conclusion can also be derived for the **adjacent voltages drop**.

Wind power injected, shown by the black-dashed line, and **Sum of Loads**, represented by the red-dashed lines, behave differently. In the prior examples, **wind power injected** was a valuable predictor variable that modeled the network dynamics very well, whereas **sum of Loads** didn't add much. On this network, however, the opposite is true. One of the reasons for this is that in the preceding example, the **wind power injected** surpassed in some circumstances, or at least had a significant impact on the network grid dynamics. However, its impact was diminished in this larger network, as illustrated in Figure 17:

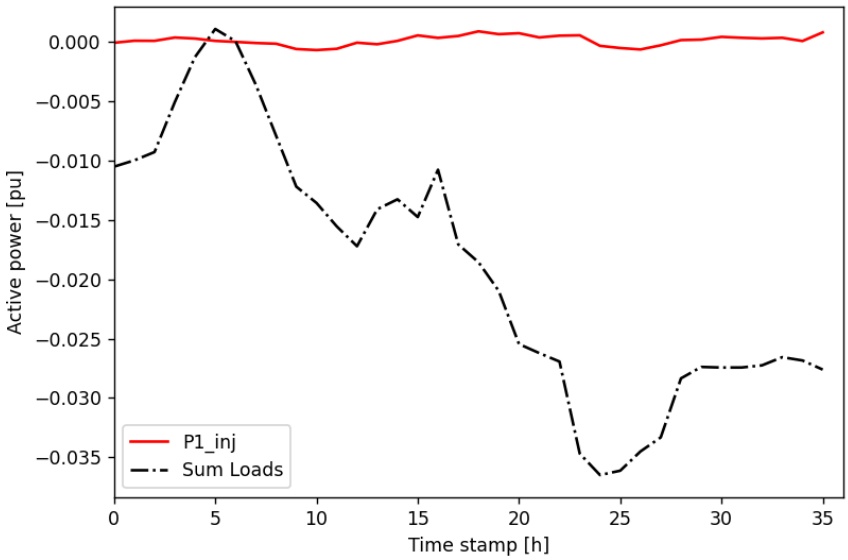

Figure 17: Active power injection in the wind farm connection bus, P_{1inj}, and sum of the active power of all loads, versus time.

Therefore, it is feasible to say that in a larger network, the **power loads** tend to be insignificant by themselves, when trying to make the forecast, because they will have little to no effect on the overall system dynamics. However, in some cases, as it happened on this example, when used together, they can bring useful information.

Overall on a larger electrical network, however, the forecasting problem becomes much more complex. There are many more variables to consider, including power generation, transmission, and distribution, as well as weather patterns, demand fluctuations, and other factors that can impact the performance of the network. In addition, the network may have a more complex underlying structure, with multiple interdependencies and feedback loops that can make it difficult to accurately predict future behavior. To address these challenges, more sophisticated forecasting techniques may be needed,

making use of more complete models that can capture the complex interactions between variables and generate more accurate predictions.

4 Conclusion

This assignment demonstrated that having a clear understanding of electrical grid operation and structure, can aid in formulating models that can forecast eletrical quantities. From the experiments done, the main findings to be retained are the following:

- By incorporating appropriate predictor variables that complement the model, it is possible to overcome, or at least limit, the autocorrelation bias and, as a result, improve model accuracy.
- On the "Kite" grid, the variables that best explained network behavior were P_{inj}^{1} and nearby voltages and currents. Because of the network's inductive nature, voltage angles provided better forecasts than voltage drops.
- The system complexity is so minimal on the smaller radial grid that the forecast for the current flow in line 1-2 was essentially correct with predictor variables that could barely model the "Kite" grid network.
- For a larger network, the predictor variable sum of all loads produced the most accurate projections. P_{inj}^{1}, on the other hand, produced the worst outcomes.
- Transformed variables can be useful for discovering autocorrelation patterns in the residuals of a time series model, leading to more accurate and reliable forecasts.
- The Cochrane-Orcutt Procedure performed as expected on all predictor variables, the Hildreth-lu Procedure only produced good results when the sum of currents I23 and I31 were employed and the First Differences Procedure produced good results when the sum of loads and P_{inj}^{1} were utilized.
- AR(1) can be effective in predicting future values based on past values, particularly when there is a clear linear relationship between the variable and its past values, however, it may not capture all of the complexities of the underlying data and may require additional explanatory variables.
- Forecasting on large electric grids is complex and requires advanced modeling techniques and expertise in electrical engineering, while forecasting on smaller grids can be accomplished using simpler models and a smaller set of predictor variables

References

1)**Lecture Notes: 3 Time series and dynamically structured problems**

2)**Applied Linear Statistical Models Michael H. Kutner Christopher J. Nachtsheim John Neter William Li**

Appendix

A Python code

```
    import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


def durbin_watson(residuals):
    """
    Compute the Durbin-Watson test statistic for residuals.

    Args:
        residuals (numpy.ndarray): Array of residuals.

    Returns:
        float: Durbin-Watson test statistic.
    """
    diff = np.diff(residuals)
    numerator = np.sum(diff ** 2)
    denominator = np.sum(residuals ** 2)
    dw = numerator / denominator
    return dw


networkFactor = 100  # To change the characteristics of the network (Y)
cosPhi = 0.95  # Value of teta
time = 24  # Training Period
timeForecast = 12  # Test Period

# Import data (From Excel file)

Info = np.array(pd.read_excel(r'\DASG_Prob2_new.xlsx', sheet_name='Info', header=None))
# Information about the slack bus
SlackBus = Info[0, 1]
print("Slack Bus: ", SlackBus, "\n")

# Network Information
Net_Info = np.array(pd.read_excel(r'DASG_Prob2_new.xlsx', sheet_name='Y_Data'))
print("Lines information (Admitances)\n", Net_Info, "\n")

# Power Information (Train)
Power_Info = np.array(pd.read_excel(r'DASG_Prob2_new.xlsx', sheet_name='Load(t,Bus)'))
Power_Info = np.delete(Power_Info, [0], 1)
print("Power consumption information (time, Bus)\n", Power_Info, "\n")

# Power Information (Test)
```

```
Power_Test = np.array(pd.read_excel(r'\DASG_Prob2_new.xlsx',
                                    sheet_name='Test_Load(t,Bus)'))
Power_Test = np.delete(Power_Test, [0], 1)

P = np.dot(-Power_Info, np.exp(complex(0, 1) * np.arccos(cosPhi)))
I = np.conj(P[2, :])

# <b>Admittance Matrix(<i>Y</i>); Conductance Matrix(<i>G</i>); Susceptance Matrix(<i>B</i>)

# Determine the number of Bus
nBus = max(np.max(Net_Info[:, 0]), np.max(Net_Info[:, 1]))

# Create the variable number of lines and the admitance matrix (Y)
nLines = Net_Info.shape[0]

Y = np.zeros((nBus, nBus), dtype=complex)

# Complete the Y matrix nad update the number of lines
for i in range(Net_Info.shape[0]):
    y_aux = Net_Info[i, 2].replace(",", ".")
    y_aux = y_aux.replace("i", "j")
    Y[Net_Info[i, 0] - 1, Net_Info[i, 0] - 1] = Y[Net_Info[i, 0] - 1, Net_Info[i, 0] - 1]
    + complex(y_aux) * networkFactor
    Y[Net_Info[i, 1] - 1, Net_Info[i, 1] - 1] = Y[Net_Info[i, 1] - 1, Net_Info[i, 1] - 1]
    + complex(y_aux) * networkFactor
    Y[Net_Info[i, 0] - 1, Net_Info[i, 1] - 1] = Y[Net_Info[i, 0] - 1, Net_Info[i, 1] - 1]
    - complex(y_aux) * networkFactor
    Y[Net_Info[i, 1] - 1, Net_Info[i, 0] - 1] = Y[Net_Info[i, 1] - 1, Net_Info[i, 0] - 1]
    - complex(y_aux) * networkFactor

# Remove the slack bus from the admitance matrix
Yl = np.delete(Y, np.s_[SlackBus - 1], axis=0)
Yl = np.delete(Yl, np.s_[SlackBus - 1], axis=1)

# Conductance Matrix
G = Yl.real

# Susceptance Matrix
B = Yl.imag

print("The admitance matrix Y is:\n", Y, "\n")
print("The conductance matrix G is\n", G, "\n")
print("The susceptance matrix B is\n", B, "\n")

# <b> Errors Definition

# Random values considering a normal distribution

np.random.seed(50)
```

```
e1 = np.random.randn(time + timeForecast) * 0.5  # Errors associated to Wind Generation
e = np.random.randn(time + timeForecast) * 0.25  # Errors associated to Power Injection

# To obtain the same values of lecture notes, we should use the following errors

e1 = [0.2878, 0.0145, 0.5846, -0.0029, -0.2718, -0.1411,
      -0.2058, -0.1793, -0.9878, -0.4926, -0.1480, 0.7222,
      -0.3123, 0.4541, 0.9474, -0.1584, 0.4692, 1.0173,
      -0.0503, 0.4684, -0.3604, 0.4678, 0.3047, -1.5098,
      -0.5515, -0.5159, 0.3657, 0.7160, 0.1407, 0.5424,
      0.0409, 0.0450, 0.2365, -0.3875, 1.4783, -0.8487]

e = [-0.0106, 0.0133, 0.2226, 0.2332, 0.1600, -0.0578,
     -0.2293, -0.2843, -0.2732, -0.1203, -0.1757, -0.1891,
     0.1541, -0.0093, -0.1691, 0.2211, -0.4515, -0.1786,
     -0.2031, -0.3634, -0.1105, -0.1413, -0.5900, -0.1729,
     -0.0810, -0.0023, -0.0556, 0.1858, -0.0324, -0.1071,
     -0.0845, -0.0743, -0.0479, -0.0870, -0.1834, -0.1432]

# <b> Determine the wind generation and the load flow in <i>I<sub>12

# Creation of Matrix
II = np.zeros((nBus - 1, time + timeForecast), dtype=complex)
i12 = np.zeros(time + timeForecast)
i1w = np.zeros(time + timeForecast)

# Initializing the process of data generation
II[:, 0] = I  # Power Injections
v = 1 + np.dot(np.linalg.inv(Yl), I)
i12[0] = np.absolute(np.dot(Y[0, 1], v[0] - v[1]))  # Current I12 in period t=0
i1w[0] = np.real(I[0])  # Injection in bus 1 (Wind) in period t=0

# Process of data generation
for t in range(time + timeForecast - 1):
    II[:, t + 1] = 0.95 * II[:, t] + e[t]
    # the values are more or less related considering
    # the value of 0.95. This value can change between 0 and 1.
    i1w[t + 1] = 0.75 * i1w[t] + e1[t]  # Wind power based on the previous periods
    II[0, t + 1] = i1w[t + 1] + complex(0, np.imag(II[0, t + 1]))  # Add the Wind generation
    v = 1 + np.dot(np.linalg.inv(Yl), II[:, t + 1])  # Compute the voltages
    I12 = np.dot(-Y[0, 1], v[0] - v[1])  # Compute the load flow in line 1-2 (Complex)
    i12[t + 1] = np.absolute(I12) * np.sign(np.real(I12))

print('The power injection in Bus 1 is:\n', II[1, :])
print('\nThe power flow in Line 1-2 is:\n', i12)

# <b>Ordinary Least Squares OLS regression

# Define the OLS regression relating the Current I12 with the Pinjection I1.
```

```
AA = np.ones((time, 2))  # Vector Xt with ones
AA[:, 1] = i1w[0:time]  # Vector Xt with ones in first column and wind injection in column 2
AATransp = np.transpose(AA)
beta = np.dot(np.dot(np.linalg.inv(np.dot(AATransp, AA)), AATransp), i12[0:time])  # Beta valu
print("AAA")
print("The value of Betas, using OLS, are:\n", beta)

# <b>Plot initial data

# Define the plots
x = range(time)
yy1 = i12[0:time]
yy2 = i1w[0:time]
rss_1 = beta[0] + np.dot(beta[1], i1w[0:time])  # OLS regresion line
yy3 = rss_1
yy4 = i12[0:time] - beta[0] - np.dot(beta[1], i1w[0:time])

# First Graph (Pinjection in bus 1 and Current I12)
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(x, yy2, 'g-')
ax2.plot(x, yy1, 'b-')
ax1.set_xlabel('Time Stamp [h]')
ax1.set_ylabel('Injection P_1', color='g')
ax2.set_ylabel('Current I_12', color='b')
ax1.set_ylim([-2, 2])
ax2.set_ylim([-1, 1])
plt.xlabel("Time Stamp [h]")

plt.xlim([0, 23])
plt.show()

# Second Graph (Relation I1 vs I12 and OLS regression)
plt.plot(yy2, yy1, 'C1o', label='P1W')
plt.plot(yy2, yy3, label='Regression line')
plt.legend()
plt.xlabel("Injection P_1")
plt.ylabel("Current I_12")
plt.show()

# Third Graph (Residuals - Difference between the real current I12 and the one obtained
plt.plot(x, yy4, 'C0o', label='Residuals')
plt.legend()
plt.xlabel("Time Stamp [h]")
plt.ylabel("Residuals")
plt.show()

# <b>Durbin-Watson statistic
```

```
D = durbin_watson(yy4)
ro = 1 - D / 2
print("The value of Durdin-Watson (DW) is:", D)
print("The value of rho is: ", ro)

# <b>Cochrane Orcutt

res_1 = i12[0:time] - rss_1
for k in range(3):  # According to "Applied Linear Statistical Models" if the OC methot does n
    # in three iterations, we should use other method
    r2 = res_1[0:time - 1]
    r1 = res_1[1:time]
    ro = 0.97 * np.dot(np.dot((np.dot(np.transpose(r2), r2)) ** (-1), np.transpose(r2)),
                       r1)  # Estimate Rho based on(28)
    i1w_s = i1w[1:time] - np.dot(ro, i1w[0:time - 1])  # Transform yt*=yt
    i12_s = i12[1:time] - np.dot(ro, i12[0:time - 1])  # Transform xt*=Xt
    B = np.ones((time - 1, 2))
    B[:, 1] = i1w_s
    b_s = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(B), B)), np.transpose(B)),
                 np.transpose(i12_s))  # Regress yt* over xt*

    b_s[0] = np.divide(b_s[0], 1 - ro)  # Transform Beta_0
    rss_s = b_s[0] + np.dot(b_s[1], i1w_s[0:time - 1])  # Update residuals
    rss_2 = b_s[0] + np.dot(b_s[1], i1w[0:time])
    res_2 = i12[0:time] - rss_2
    res_1 = res_2[:]
b_ss = b_s

# <b>Forecast Day-ahead (Current I_12)

I12f1 = beta[0] + np.dot(beta[1], i1w[time:time + timeForecast])
I12f2 = b_ss[0] + np.dot(b_ss[1], i1w[time:time + timeForecast])
print("Forecast Corrent I12 considering OLS:", I12f1, "\n")
print("Forecast Corrent I12 considering CO:", I12f2)

# Plot forecsated values

x = range(time)
xx = range(time - 1)
xxx = range(time, time + timeForecast)

yy1 = i12[0:time]
yy2 = i1w[0:time]
yy3 = rss_1
yy4 = rss_2
yy5 = i12[0:time] - rss_1
yy6 = i12[0:time - 1] - rss_2[0:time - 1]
yy7 = i12[time:time + timeForecast]
yy8 = I12f1
```

```
yy9 = I12f2
D = durbin_watson(yy6)
print("AAAAAAAAAAAAAAA")
print("The value of Durdin-Watson (DW) is:", D)
D = durbin_watson(yy5)
print("AAAAAAAAAAAAAAA")
print("The value of Durdin-Watson (DW) is:", D)

plt.plot(xxx, yy7, color='red', label='Measured')
plt.plot(xxx, yy8, color='black', linestyle='dashed', marker='o', label='OLS')
plt.plot(xxx, yy9, color='red', linestyle='-.', marker='*', label='CO')
plt.legend()
plt.xlabel('Time stamp [h]')
plt.ylabel('Current I_12')
plt.xlim([24, 35])
plt.show()

plt.plot(yy2, yy1, 'C1o', label='i12')
plt.plot(yy2, yy3, label='OLC Regression')
plt.plot(yy2, yy4, label='CO Regression')
plt.legend()
plt.xlabel("Injection P_1")
plt.ylabel("Current I_12")

plt.show()

plt.plot(x, yy5, 'C1o', label='Residuals OLS')
plt.plot(xx, yy6, 'C0*', label='Residuals CO')

plt.legend()
plt.show()

print(sum(np.square(yy5)))
print(sum(np.square(yy6)))

# Autocorrelation Method

# In this example, the input data is different because the error used to generate the values
# the same results, we should use the next values. To compare with previous example

ee1 = [0.2878, 0.0145, 0.5846, -0.0029, -0.2718, -0.1411,
       -0.2058, -0.1793, -0.9878, -0.4926, -0.1480, 0.7222,
       -0.3123, 0.4541, 0.9474, -0.1584, 0.4692, 1.0173,
       -0.0503, 0.4684, -0.3604, 0.4678, 0.3047, -1.5098,
       -0.5515, -0.5159, 0.3657, 0.7160, 0.1407, 0.5424,
       0.0409, 0.0450, 0.2365, -0.3875, 1.4783, -0.8487]

ee = [0.2226, -0.2293, -0.1757, -0.1691, -0.2031, -0.5900,
      -0.0556, -0.0845, -0.1834, 0.2798, 0.1534, 0.0751,
```

```
      -0.1089, 0.3545, 0.0228, -0.2139, 0.4409, 0.6044,
      -0.2187, -0.1233, 0.0026, 0.4980, 0.3703, 0.0812,
      0.1183, 0.2486, -0.0686, -0.0727, -0.0009, -0.1180,
      0.2443, 0.6224, -0.4600, -0.3878, 0.4734, -0.4050]

II = np.zeros((nBus - 1, time + timeForecast), dtype=complex)
II[:, 0] = I
i12 = np.zeros(time + timeForecast)
i1w = np.zeros(time + timeForecast)

v = 1 + np.dot(np.linalg.inv(Yl), I)
i12[0] = np.absolute(np.dot(Y[0, 1], v[0] - v[1]))
i1w[0] = np.real(I[0])
for t in range(time + timeForecast - 1):
    II[:, t + 1] = 0.95 * II[:, t] + ee[t]  # meter ee
    i1w[t + 1] = 0.75 * i1w[t] + ee1[t]  # meter ee1
    II[0, t + 1] = i1w[t + 1] + complex(0, np.imag(II[0, t + 1]))
    v = 1 + np.dot(np.linalg.inv(Yl), II[:, t + 1])
    I12 = np.dot(-Y[0, 1], v[0] - v[1])
    i12[t + 1] = np.absolute(I12) * np.sign(np.real(I12))

# <b> Models
# - 1 - OLS
# - 2 - Cochrane Orcutt (CO)
# - 3 - Autorregration AR(1)
# - 4 - Autorregration with Loads AR(1)+Load Sum


## 1 - OLS
AA[:, 1] = i1w[0:time]  # Vector Xt with ones in first column and wind injection in column 2
AATransp = np.transpose(AA)
beta = np.dot(np.dot(np.linalg.inv(np.dot(AATransp, AA)), AATransp), i12[0:time])
print("The value of Betas, using OLS, are:\n", beta)

## 2 - Cochrane Orcutt (CO)
rss_1 = beta[0] + np.dot(beta[1], i1w[0:time])  # OLS regresion line
res_1 = i12[0:time] - rss_1
for k in range(3):
    # in three iterations, we should use other method
    r2 = res_1[0:time - 1]
    r1 = res_1[1:time]
    ro = 0.97 * np.dot(np.dot((np.dot(np.transpose(r2), r2)) ** (-1), np.transpose(r2)),
                       r1)  # Estimate Rho based on (28)
    i1w_s = i1w[1:time] - np.dot(ro, i1w[0:time - 1])  # Transform yt*=yt
    i12_s = i12[1:time] - np.dot(ro, i12[0:time - 1])  # Transform xt*=Xt
    B = np.ones((time - 1, 2))
    B[:, 1] = i1w_s
    b_s = np.dot(np.dot(np.linalg.inv(np.dot(np.transpose(B), B)), np.transpose(B)),
                 np.transpose(i12_s))  # Regress yt* over xt*
```

```
    b_s[0] = np.divide(b_s[0], 1 - ro)  # Transform Beta_0
    rss_s = b_s[0] + np.dot(b_s[1], i1w_s[0:time - 1])  # Update residuals
    rss_2 = b_s[0] + np.dot(b_s[1], i1w[0:time])
    res_2 = i12[0:time] - rss_2
    res_1 = res_2[:]
b_ss = b_s

## 3 - Autorregration AR(1)
# Definition of Matrix Xt
X = np.ones((time, 3))  # Vector Xt with ones
X[:, 1] = i1w[1:time + 1]
X[:-1, 2] = i12[0:time - 1]  # Vector Xt with previous time current i12 injection
XTransp = np.transpose(X)

# Definition of Matrix y
# Compute Beta using 32
Beta_AR1 = np.dot(np.dot(np.linalg.inv(np.dot(XTransp, X)), XTransp), i12[0:time])

# Equation 31
# Residuals

## 4 - Autorregration with Loads AR(1)+Load Sum
soma = np.zeros(time + timeForecast)
for j in range(time + timeForecast):
    soma[j] = sum(np.real(II[:, j]))

# Definition of Matrix Xt
X2 = np.ones((time, 3))  # Vector Xt with ones
X2[:, 1] = i1w[0:time]  # Vector Xt with ones in first column and wind injection in column 2
X2[:, 2] = soma[0:time]  # Vector Xt with previous time current i12 injection
XTransp2 = np.transpose(X2)
# Definition of Matrix y
# Compute Beta using 32
Beta_AR2 = np.dot(np.dot(np.linalg.inv(np.dot(XTransp2, X2)), XTransp2), i12[0:time])
# Equation 31
# Residuals

# Definition of Matrix Xt
X3 = np.ones((time - 1, 4))  # Vector Xt with ones
X3[:, 1] = i1w[1:time]  # Vector Xt with ones in first column and wind injection in column 2
X3[:, 2] = i12[0:time - 1]  # Vector Xt with previous time current i12 injection
X3[:, 3] = soma[1:time]  # Vector Xt with previous time current i12 injection
XTransp3 = np.transpose(X3)
# Definition of Matrix y
# Compute Beta using 32
Beta_AR3 = np.dot(np.dot(np.linalg.inv(np.dot(XTransp3, X3)), XTransp3), i12[1:time])
# Equation 31
# Residuals
```

```
# <b>Forecast Day-ahead (Current I_12)

# 1 - OLS

I12f1 = beta[0] + np.dot(beta[1], i1w[time:time + timeForecast])
print("Forecast Corrent I12 considering OLS:", I12f1, "\n")

# 2 - Cochrane Orcutt (CO)
I12f2 = b_ss[0] + np.dot(b_ss[1], i1w[time:time + timeForecast])
print("Forecast Corrent I12 considering CO:", I12f2)

# 3 - Autorregration AR(1)
I12f3 = Beta_AR1[0] + np.dot(Beta_AR1[1], i1w[time:time + timeForecast]) +
np.dot(Beta_AR1[2], i12[time: time +
timeForecast])

# 4 - Autorregration with Load Sum
I12f4 = np.zeros(timeForecast)
I12f4[0] = Beta_AR2[0] + Beta_AR2[1] * i1w[time] + Beta_AR2[2] * soma[time]
for i in range(1, timeForecast):
    I12f4[i] = Beta_AR2[0] + Beta_AR2[1] * i1w[time + i] + Beta_AR2[2] * soma[time + i]

# 5 - Autorregration with Loads AR(1)+Load Sum
I12f5 = np.zeros(timeForecast)
I12f5[0] = Beta_AR3[0] + Beta_AR3[1] * i1w[time] + Beta_AR3[2] * i12[time - 1] +
Beta_AR3[3] * soma[time]
for i in range(1, timeForecast):
    I12f5[i] = Beta_AR3[0] + Beta_AR3[1] * i1w[time + i] + Beta_AR3[2] * I12f5[i - 1] +
    Beta_AR3[3] * soma[time + i]

# <b>Plot forecsated values
yy7 = i12[time:time + timeForecast]
yy8 = I12f1
yy9 = I12f2
yy10 = I12f3
yy11 = I12f4
yy12 = I12f5

plt.plot(xxx, yy7, color='red', label='Measured')
plt.plot(xxx, yy8, color='black', linestyle='dashed', marker='o', label='OLS')
plt.plot(xxx, yy9, color='red', linestyle='-.', marker='*', label='CO')
plt.plot(xxx, yy10, color='green', linestyle='dashed', marker='o', label='AR(1)')
plt.plot(xxx, yy11, color='orange', linestyle='-.', marker='*', label='Loads')
plt.plot(xxx, yy12, color='purple', linestyle='-.', marker='*', label='AR(1)+Loads')
plt.xlim((24, 36))
plt.legend()
plt.xlabel('Time stamp [h]')
plt.ylabel('Current I_12')
```

```
plt.show()
```

B Data generation on radial grid Matlab code

```
    %% Time series
%Hw4 Analitica de dados para redes inteligentes
%P3 2022-2023
%G.Aniceto 96218

clc; clear all; close all

%% Power consuption - Time series
W=[-0.2128+0.06994398j
    0.1282+0.06644678j
    0.11065+0.06312444j
    0.6675875+0.05996822j
    0.49779063+0.05696981j
    0.10154297+0.05412132j
   -0.06494277+0.05141525j
   -0.25450708+0.04884449j
   -0.37018031+0.04640226j
   -1.26543523+0.04408215j
   -1.44167642+0.04187804j
   -1.22925732+0.03978414j
   -0.19974299+0.03779493j
   -0.46210724+0.03590519j
    0.10751957+0.03410993j
    1.02803968+0.03240443j
    0.61262976+0.03078421j
    0.92867232+0.029245j
    1.71380424+0.02778275j
    1.23505318+0.02639361j
    1.39468988+0.02507393j
    0.68561741+0.02382023j
    0.98201306+0.02262922j
    1.04120979+0.02149776j
   -0.72889265+0.02042287j
   -1.09816949+0.01940173j
   -1.33952712+0.01843164j
   -0.63894534+0.01751006j
    0.236791+0.01663456j
    0.31829325+0.01580283j
    0.78111994+0.01501269j
    0.62673995+0.01426205j
    0.51505496+0.01354895j
    0.62279122+0.0128715j
```

```
    0.07959342+0.01222793j
    1.53799506+0.01161653j].';

I =[-0.6726+0.22107293j
    -0.64957+0.21001928j
    -0.6037915+0.19951832j
    -0.35100192+0.1895424j
    -0.10025183+0.18006528j
     0.06476076+0.17106202j
     0.00372272+0.16250892j
    -0.22576341+0.15438347j
    -0.49877524+0.1466643j
    -0.74703648+0.13933108j
    -0.82998466+0.13236453j
    -0.96418542+0.1257463j
    -1.10507615+0.11945899j
    -0.89572234+0.11348604j
    -0.86023623+0.10781174j
    -0.98632442+0.10242115j
    -0.71590819+0.09730009j
    -1.13161278+0.09243509j
    -1.25363215+0.08781333j
    -1.39405054+0.08342267j
    -1.68774801+0.07925153j
    -1.71386061+0.07528896j
    -1.76946758+0.07152451j
    -2.2709942+0.06794828j
    -2.33034449+0.06455087j
    -2.29482727+0.06132333j
    -2.1823859+0.05825716j
    -2.12886661+0.0553443j
    -1.83662328+0.05257709j
    -1.77719211+0.04994823j
    -1.79543251+0.04745082j
    -1.79016088+0.04507828j
    -1.77495284+0.04282437j
    -1.7341052+0.04068315j
    -1.73439994+0.03864899j
    -1.83107994+0.03671654j].';

%Loads-expanded
for k = 1:4
    for kk = 1:36
        if k == 1
            I_load(k,kk) = W(1,kk);
        else
            I_load(k,kk) = I(1,kk);
        end
    end
```

```
end

%Loads-softer
for k = 1:4
    for kk = 1:36
            I_load(k,kk) = 0.6*I_load(k,kk);
    end
end

%% Grid
%Radial
Y = [ 1-10*j -1+10*j 0  0  0
    -1+10*j  4-30*j -3+20*j  0  0
      0 -3+20*j  6-40*j -3+20*j  0
      0  0 -3+20*j  5-40*j -2+20*j
      0  0  0 -2+20*j  2-20*j];

Z = Y(1:4,1:4)^-1;

%% State
%Generate voltage data
voltages = zeros(5,1);

for k = 1:36
v=1+Z*I_load(:,k);
v = [v;1];

    if k == 1
        voltages = v;
    else
        voltages = [voltages v];
    end
end

%Generate I1-2 data
current = zeros(1,36);
for k = 1:36
current(k) = (voltages(1,k) - voltages(2,k))*(1-10*j);
end

W2=0.6*W;
```

C Data generation on IEEE-33 grid Matlab code

```
   %% Time series
%Hw4 Analitica de dados para redes inteligentes
%P3 2022-2023
%G.Aniceto 96218
```

```
clc; clear all; close all

%% Power consuption - Time series
W=[-0.2128+0.06994398j
    0.1282+0.06644678j
    0.11065+0.06312444j
    0.6675875+0.05996822j
    0.49779063+0.05696981j
    0.10154297+0.05412132j
   -0.06494277+0.05141525j
   -0.25450708+0.04884449j
   -0.37018031+0.04640226j
   -1.26543523+0.04408215j
   -1.44167642+0.04187804j
   -1.22925732+0.03978414j
   -0.19974299+0.03779493j
   -0.46210724+0.03590519j
    0.10751957+0.03410993j
    1.02803968+0.03240443j
    0.61262976+0.03078421j
    0.92867232+0.029245j
    1.71380424+0.02778275j
    1.23505318+0.02639361j
    1.39468988+0.02507393j
    0.68561741+0.02382023j
    0.98201306+0.02262922j
    1.04120979+0.02149776j
   -0.72889265+0.02042287j
   -1.09816949+0.01940173j
   -1.33952712+0.01843164j
   -0.63894534+0.01751006j
    0.236791+0.01663456j
    0.31829325+0.01580283j
    0.78111994+0.01501269j
    0.62673995+0.01426205j
    0.51505496+0.01354895j
    0.62279122+0.0128715j
    0.07959342+0.01222793j
    1.53799506+0.01161653j].';

I =[-0.6726+0.22107293j
    -0.64957+0.21001928j
    -0.6037915+0.19951832j
    -0.35100192+0.1895424j
    -0.10025183+0.18006528j
     0.06476076+0.17106202j
     0.00372272+0.16250892j
```

```
    -0.22576341+0.15438347j
    -0.49877524+0.1466643j
    -0.74703648+0.13933108j
    -0.82998466+0.13236453j
    -0.96418542+0.1257463j
    -1.10507615+0.11945899j
    -0.89572234+0.11348604j
    -0.86023623+0.10781174j
    -0.98632442+0.10242115j
    -0.71590819+0.09730009j
    -1.13161278+0.09243509j
    -1.25363215+0.08781333j
    -1.39405054+0.08342267j
    -1.68774801+0.07925153j
    -1.71386061+0.07528896j
    -1.76946758+0.07152451j
    -2.2709942+0.06794828j
    -2.33034449+0.06455087j
    -2.29482727+0.06132333j
    -2.1823859+0.05825716j
    -2.12886661+0.0553443j
    -1.83662328+0.05257709j
    -1.77719211+0.04994823j
    -1.79543251+0.04745082j
    -1.79016088+0.04507828j
    -1.77495284+0.04282437j
    -1.7341052+0.04068315j
    -1.73439994+0.03864899j
    -1.83107994+0.03671654j].';

%Loads-expanded
for k = 1:32
    for kk = 1:36

        if k == 21
            I_load(k,kk) = W(1,kk);
        else
            I_load(k,kk) = I(1,kk);
        end
    end
end

%Loads-softer
for k = 1:32
    for kk = 1:36

            I_load(k,kk) = 0.0005*I_load(k,kk);

    end
```

```
end



%% Grid
%Branch Impedances
Z12 = 0.092 + 1i*0.047;
Z23 = 0.493 + 1i*0.2511;
Z34 = 0.366 + 1i*0.1864;
Z45 = 0.3811 + 1i*0.1941;
Z56 = 0.819 + 1i*0.707;
Z67 = 0.1872 + 1i*0.6188;
Z78 = 0.7114 + 1i*0.2351;
Z89 = 1.03 + 1i*0.74;
Z910 = 1.044 + 1i*0.74;
Z1011 = 0.1966 + 1i*0.065;
Z1112 = 0.3744 + 1i*0.1238;
Z1213 = 1.468 + 1i*1.155;
Z1314 = 0.5416 + 1i*0.7129;
Z1415 = 0.591 + 1i*0.526;
Z1516 = 0.7463 + 1i*0.545;
Z1617 = 1.289 + 1i*1.721;
Z1718 = 0.732 + 1i*0.574;
Z219 = 0.164 + 1i*0.1565;
Z1920 = 1.5042 + 1i*1.3554;
Z2021 = 0.4095 + 1i*0.4784;
Z2122 = 0.7089 + 1i*0.9373;
Z323 = 0.4512 + 1i*0.3083;
Z2324 = 0.898 + 1i*0.7091;
Z2425 = 0.896 + 1i*0.7011;
Z626 = 0.203 + 1i*0.1034;
Z2627 = 0.2842 + 1i*0.1447;
Z2728 = 1.059 + 1i*0.9337;
Z2829 = 0.8042 + 1i*0.7006;
Z2930 = 0.5075 + 1i*0.2585;
Z3031 = 0.9744 + 1i*0.963;
Z3132 = 0.3105 + 1i*0.3619;
Z3233 = 0.341 + 1i*0.5302;
Z821 = 10000000000;
Z1222 = 1000000000;
Z2529 = 1000000000;

%Admitance matrix
Y = zeros(33,33);
Y(1,1) = 1/Z12;
Y(1,2) = -1/Z12;

Y(2,1) = -1/Z12;
Y(2,2) = 1/Z12 + 1/Z23 + 1/Z219;
```

```
Y(2,3) = -1/Z23;
Y(2,19) = -1/Z219;

Y(3,2) = -1/Z23;
Y(3,3) = 1/Z23 + 1/Z323 + 1/Z34;
Y(3,4) = -1/Z34;
Y(3,23) = -1/Z323;

Y(4,3) = -1/Z34;
Y(4,4) =  1/Z34 + 1/Z45;
Y(4,5) = -1/Z45;

Y(5,4) = -1/Z45;
Y(5,5) =  1/Z45 + 1/Z56;
Y(5,6) = -1/Z56;

Y(5,4) = -1/Z45;
Y(5,5) =  1/Z45 + 1/Z56;
Y(5,6) = -1/Z56;

Y(6,5) = -1/Z56;
Y(6,6) =  1/Z56 + 1/Z67 + 1/Z626;
Y(6,7) = -1/Z67;
Y(6,26) = -1/Z626;

Y(7,6) = -1/Z67;
Y(7,7) =  1/Z67 + 1/Z78;
Y(7,8) = -1/Z78;

Y(8,7) = -1/Z78;
Y(8,8) =  1/Z89 + 1/Z78 + 1/Z821;
Y(8,9) = -1/Z89;
Y(8,21) = -1/Z821;

Y(9,8) = -1/Z89;
Y(9,9) =  1/Z89 + 1/Z910;
Y(9,10) = -1/Z910;

Y(10,9) = -1/Z910;
Y(10,10) =  1/Z1011 + 1/Z910;
Y(10,11) = -1/Z1011;

Y(10,9) = -1/Z910;
Y(10,10) =  1/Z1011 + 1/Z910;
Y(10,11) = -1/Z1011;

Y(11,10) = -1/Z1011;
Y(11,11) =  1/Z1011 + 1/Z1112;
Y(11,12) = -1/Z1112;
```

```
Y(12,11) = -1/Z1112;
Y(12,12) =  1/Z1112 + 1/Z1213 + 1/Z1222;
Y(12,13) = -1/Z1213;
Y(12,22) = -1/Z1222;

Y(13,12) = -1/Z1213;
Y(13,13) =  1/Z1213 + 1/Z1314;
Y(13,14) = -1/Z1314;

Y(14,13) = -1/Z1314;
Y(14,14) =  1/Z1415 + 1/Z1314;
Y(14,15) = -1/Z1415;

Y(15,14) = -1/Z1415;
Y(15,15) =  1/Z1415 + 1/Z1516;
Y(15,16) = -1/Z1516;

Y(16,15) = -1/Z1516;
Y(16,16) =  1/Z1617 + 1/Z1516;
Y(16,17) = -1/Z1617;

Y(17,16) = -1/Z1617;
Y(17,17) =  1/Z1617 + 1/Z1718;
Y(17,18) = -1/Z1718;

Y(18,17) = -1/Z1718;
Y(18,18) =  1/Z1718;

Y(19,2) = -1/Z219;
Y(19,19) = 1/Z1920 + 1/Z219;
Y(19,20) = -1/Z1920;

Y(19,20) = -1/Z1920;
Y(20,20) = 1/Z1920 + 1/Z2021;
Y(20,21) = -1/Z1920;

Y(21,8) = -1/Z821;
Y(21,20) = -1/Z2021;
Y(21,21) = 1/Z2122 + 1/Z2021 +1/Z821;
Y(21,22) = -1/Z2122;

Y(22,12) = -1/Z1222;
Y(22,21) = -1/Z2122;
Y(22,22) = 1/Z2122 + 1/Z1222;

Y(23,3) = -1/Z323;
Y(23,23) = 1/Z323 + 1/Z2021;
Y(23,24) = -1/Z2324;
```

```
Y(24,23) = -1/Z2324;
Y(24,24) = 1/Z2324 + 1/Z2425;
Y(24,25) = -1/Z2425;

Y(25,24) = -1/Z2425;
Y(25,25) = 1/Z2425 +1/Z2529;
Y(25,29) = -1/Z2529;

Y(26,6) = -1/Z626;
Y(26,26) = 1/Z626 + 1/Z2627;
Y(26,27) = -1/Z2627;

Y(27,26) = -1/Z2627;
Y(27,27) = 1/Z2728 + 1/Z2627;
Y(27,28) = -1/Z2728;

Y(28,27) = -1/Z2728;
Y(28,28) = 1/Z2728 + 1/Z2728;
Y(28,29) = -1/Z2829;

Y(29,25) = -1/Z2529;
Y(29,28) = -1/Z2829;
Y(29,29) = 1/Z2829 + 1/Z2930 + 1/Z2529;
Y(29,30) = -1/Z2930;

Y(30,29) = -1/Z2930;
Y(30,30) = 1/Z3031 + 1/Z2930;
Y(30,31) = -1/Z3031;

Y(31,30) = -1/Z3031;
Y(31,31) = 1/Z3031 + 1/Z3132;
Y(31,32) = -1/Z3132;

Y(32,31) = -1/Z3132;
Y(32,32) = 1/Z3132 + 1/Z3233;
Y(32,33) = -1/Z3233;

Y(33,32) = -1/Z3233;
Y(33,33) = 1/Z3233;

%Susceptances
b12 = -imag(Y(1,2));
b23 = -imag(Y(2,3));
b34 = -imag(Y(3,4));
b45 = -imag(Y(4,5));
b56 = -imag(Y(5,6));
b67 = -imag(Y(6,7));
b78 = -imag(Y(7,8));
```

```
b89 = -imag(Y(8,9));
b910 = -imag(Y(9,10));
b1011 = -imag(Y(10,11));
b1112 = -imag(Y(11,12));
b1213 = -imag(Y(12,13));
b1314 = -imag(Y(13,14));
b1415 = -imag(Y(14,15));
b1516 = -imag(Y(15,16));
b1617 = -imag(Y(16,17));
b1718 = -imag(Y(17,18));
b219 = -imag(Y(2,19));
b1920 = -imag(Y(19,20));
b2021 = -imag(Y(20,21));
b2122 = -imag(Y(21,22));
b323 = -imag(Y(3,23));
b2324 = -imag(Y(23,24));
b2425 = -imag(Y(24,25));
b626 = -imag(Y(6,26));
b2627 = -imag(Y(26,27));
b2728 = -imag(Y(27,28));
b2829 = -imag(Y(28,29));
b2930 = -imag(Y(29,30));
b3031 = -imag(Y(30,31));
b3132 = -imag(Y(31,32));
b3233 = -imag(Y(32,33));
b821 = -imag(Y(8,21));
b1222 = -imag(Y(12,22));
b2529 = -imag(Y(25,29));

%% State
%All open
Z = Y(2:33,2:33)^-1;

%Generate voltage data
voltages = zeros(33,1);
for k = 1:36
v=1+Z*I_load(:,k);
v = [1; v];

    if k == 1
        voltages = v;
    else
        voltages = [voltages v];
    end
end

%Generate I19-2 data
current = zeros(1,36);
loads = zeros(1,36);
```

```
c1 = zeros(1,36);
c2 = zeros(1,36);
c3 = zeros(1,36);

%I forecast
for k = 1:36
current(k) = (voltages(19,k) - voltages(2,k))/Z219;
end

%P injected wind
W2=0.0005*W;

%Load sum
for k = 1:36
    for kk = 1:32
        loads(k) = loads(k) + conj(I_load(kk,k))
    end
end

%Currents
for k = 1:36
    c1(k) = (voltages(20,k)-voltages(19,k))/Z1920
    c2(k) = (voltages(3,k)-voltages(2,k))/Z23
    c3(k) = (voltages(1,k)-voltages(2,k))/Z12
end
```