TÉCNICO LISBOA

Data Analitics for Smart Grids

MEEC

Lab 2 - Losses Prediction

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

Power loss calculation is a classic problem in power systems engineering. Conventional power flow technique solutions, which have been around for a long time, rely on network element and load models, the parameters of which are only approximations, resulting in a discrepancy between observed and calculated values. The availability of high resolution smart metering data enabled significantly more exact grid modeling, without a sophisticated network model.

2 Objectives

The aim of this assignment is to address the losses prediction problem formulating it as a linear regression problem. There will be two sets of per bus power consumption data provided: a training set and a data set. The first will be used to produce a loss function, while the second will be utilized for performance evaluation. An algorithm will be proposed and tested on two grids.

3 Algorithm

Lets consider the DC power flow approximations, ie $V_i \approx V_j = 1\ pu$, $\theta_{ij} << 1\ rad\ and\ G_i = G_j = 0$. Note that the voltage angles, θ, can be expressed linearly on the provided active power data as $\theta = -B^{-1}P$. Given that the grid is not low voltage, branch susceptance will be an order of magnitude larger than branch condutance, ie $B_{ij} >> G_{ij}$, the power losses can be approximated by the quadratic function:

$$P_L \approx \sum_{i \neq j} G_{ij} \theta_{ij}^2$$

As an alternative, the losses can be computed through a quadratic magnitude voltage formula valid for LV grids:

$$P_L \approx \sum_{i \neq j} G_{ij} (V_i \ - V_i)^2$$

The power loss data may be structured in a matrix format across time to produce a multivariate regression equation:

$$y = \begin{bmatrix} P_L \\ \vdots \\ P_L \end{bmatrix}, k \in 1, 2, 3, \ldots, M$$

Considering $\beta = (B^{-1})^T C G_{ij}^{diag} C^T B^{-1}$, the power loss can be written as the inner product of coefficients by a vector of outer product terms:

$$y \approx P^T \beta P = \langle \beta_i, P_i \otimes P_i \rangle$$

The outer product expansion may be written in matrix form as a multivariate regression equation with M readings on both power loss, y, and power consumption outer product, X, considering a normally distributed error with null mean value, $\epsilon \sim \mathcal{N}(0, \sigma^2)$:

$$Y = BX + \epsilon$$

The loss function is given by least squares parameter estimates obtained through:

$$\beta = (X^T X)^{-1} X^T Y$$

Algorithm 1 Losses Prediction Model Training

Build grid matrices Y B G C
Approximate voltage diference
$\theta_{ij} \leftarrow C^T B^{-1} P_{test}$
$V_{ij} \leftarrow C^T G^{-1} P_{test}$
Approximate power loss
$y \leftarrow P_L \approx \sum_{i \neq j} G_{ij}[1 - cos(\theta_i - \theta_i)]$
$y \leftarrow P_L \approx \sum_{i \neq j} G_{ij}(V_i - V_i)^2$
Build readings matrix
$X \leftarrow P_{test} \otimes P_{test}$
Regress Y over X
$min_\beta \ (Y - \beta X)^2$

4 Losses Prediction Problems

4.1 "Kite" Grid

In this section, a simple situation will be examined in which customer data for a five bus "kite" grid is provided. The loss prediction task at hand, despite its small size, is difficult due to high load levels and volatility.

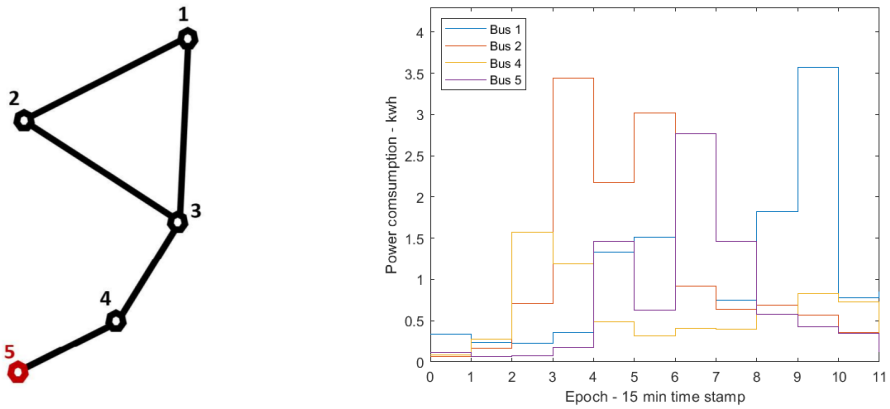

Figure 1: "Kite" grid and chronological representation of bus load measurement(training set)

In order to describe the network, we must first compute its **Admitance**, **Conductance**, and **Susceptance** matrices while keeping in mind that the Slack Bus was excluded from all calculations. This matrices have information about the physical properties of the network. The **Conductance** represents the real part of the **Admitance** matrix and retains information about the ability of the network to conduct electrical current, while the **Susceptance** represents the imaginary part and retains information about the ability of the network to store and release electrical energy. It resulted in the following matrices:

$$Admitance = Y = \begin{bmatrix} 3-30j & -1+10j & -2+20j & 0 \\ -1+10j & 4-30j & -3+20j & 0 \\ -2+20j & -3+20j & 8-60j & -3+20j \\ 0 & 0 & -3+20j & 5-40j \end{bmatrix}$$

$$Conductance = G = \begin{bmatrix} 3 & -1 & -2 & 0 \\ -1 & 4 & -3 & 0 \\ -2 & -3 & 8 & -3 \\ 0 & 0 & -3 & 5 \end{bmatrix}$$

$$Susceptance = B = \begin{bmatrix} -30 & 10 & 20 & 0 \\ 10 & -30 & 20j & 0 \\ 20 & 20 & -60 & 20 \\ 0 & 0 & 20 & -40 \end{bmatrix}$$

Next it was time to define the **Incidance matrix**, which represents the energy flow direction, from the edges that connect nodes. The matrix rows correspond to the nodes(bus) and the columns correspond to the edges(lines) that connect each node. So if two nodes aren't connected the matrix has a value of 0 and if they are connected, it has a value of 1, or -1 depending on whether the node is incident to the edge in a certain direction. It resulted in the following matrix:

$$IncidenceMatrix = C = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

4.1.1 Outer product expansion equation

To create the **Loss function**, that will be used to calculate and predict the **Power Losses**, based on the current power of each Bus, we will first use a **training set**. To do so, with this set, first we calculate **Power Losses**, using the two Algorithms shown on **Section 3**. It resulted in the following results:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
$\mathbf{P}_{L2}$	0.0040525	0.00665397	0.08342236	0.38413184	0.30832255	0.39635134	0.17306509	0.08269704	0.15806927	0.38605897	0.05446394	0.02942382	0.0388501
$\mathbf{P}_{L}$	0.00405272	0.00665461	0.0835298	0.38592245	0.3096688	0.39823976	0.1737622	0.08283	0.15840609	0.38795136	0.05450801	0.02943516	0.0388715

Table 1: Power losses calculated for the 13 periods of time. P_{L2} is calculated with equation (13), P_L is calculated with equation (15).

Subsequently it was time to calculate the **Loss function**, where we used P_{L2} values (after deliberation with the professor) and the outer product expansion equation. It resulted in the following results:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Beta	0.01635835	0.01489069	0.01214016	0.0041297	0.01684308	0.01340756	0.00481611	0.01173772	0.00610852	0.00519595

Table 2: Loss function values.

To see if the **Loss Function** obtained, was giving an correct estimation for the **Power losses**, over the training set, we used it to calculate the values and compare it with the ones obtained with equation (13). To mimic the error of the measurements we added a small normally distributed error of, $\epsilon \sim \mathcal{N}(0, 0.0025)$. We obtained the following values:

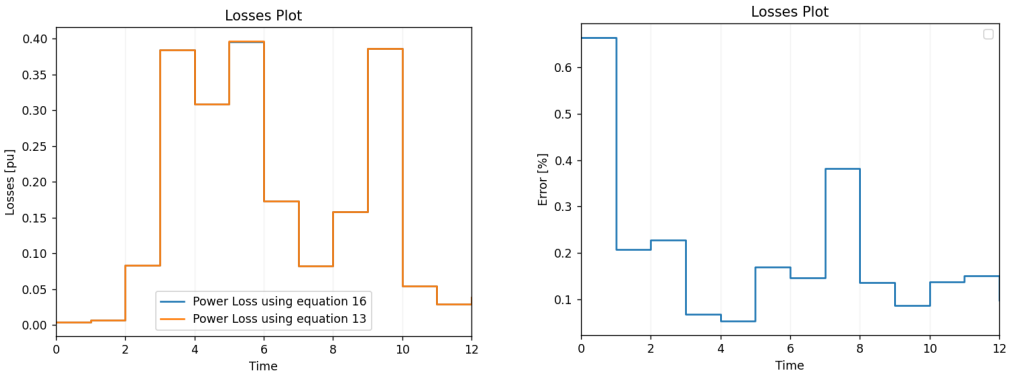

Figure 2: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Training set.

As we can see by analysing figure 2, even in the presence of measurement noise, our calculated Beta can predict the Power Losses values with barely any error. However, this doesn't mean that it will perform good in unseen data. Therefore, we calculated again the **Power losses** to the unseen **Test set**, using again equation (13), and the **Loss function** with the values that we derived earlier. We obtained the following values:

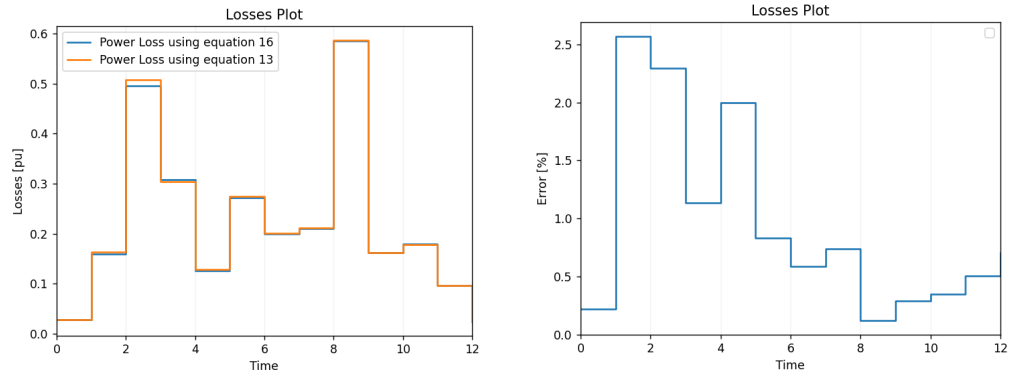

Figure 3: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Testing set

Through figure 3 it is possible to see that even for the unseen **Test Set**, equation (16), with the **Loss function** values calculate for the training set, still provides a good approximation to the **Power Losses**. As it is easy to understand, the errors obtained where bigger than the ones calculated before, because the **Loss function** was derived from that data. Nonetheless, we can say that this predictor is robust and accurate to predict unseen data.

However, there is something else that we need to worry about, when using the **outer expansion equation**, to calculate the **Loss function**. This method grows quadratically, with the number of buses added to the network. Considering a network with N buses, the size of the Matrix X (outer product expansion of the power injections P) will have size $\frac{n^2+n}{2}$.

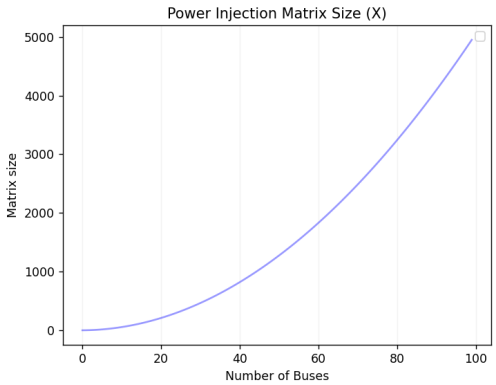

Figure 4: Outer product expansion of the power injections P matrix size, according to the number of buses the network.

Figure 4 clearly shows that it is too computationally heavy to use this method in big networks. With only 100 buses, the matrix X becomes of size 4950. Therefore it is necessary to implement other methods that trade accuracy for performance.

4.1.2 Squared injections only

To reduce the size of the matrix X, we implemented a method that only takes in consideration the value of the squared injections. Therefore the size of this matrix will be linearly related with the amount of buses on the network. Considering a network with N buses, the size of the Matrix X will be of size N.

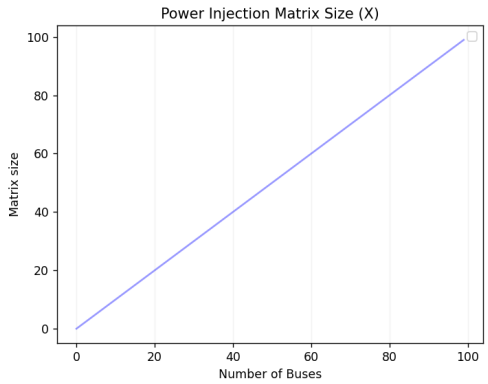

Figure 5: Squared injections only of the power injections P matrix size, according to the number of buses the network.

To get to the **Power losses**, we followed the same steps on **Section 4.1.1** and obtained the following values for the **training set** and the **test set**:

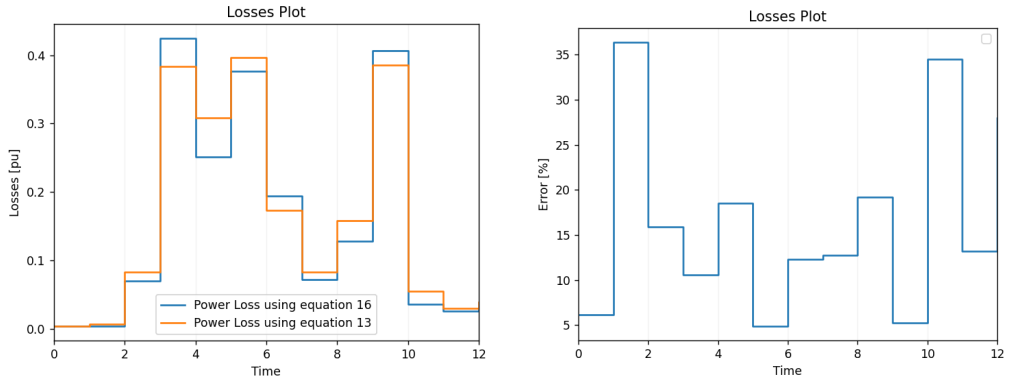

Figure 6: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Training set.

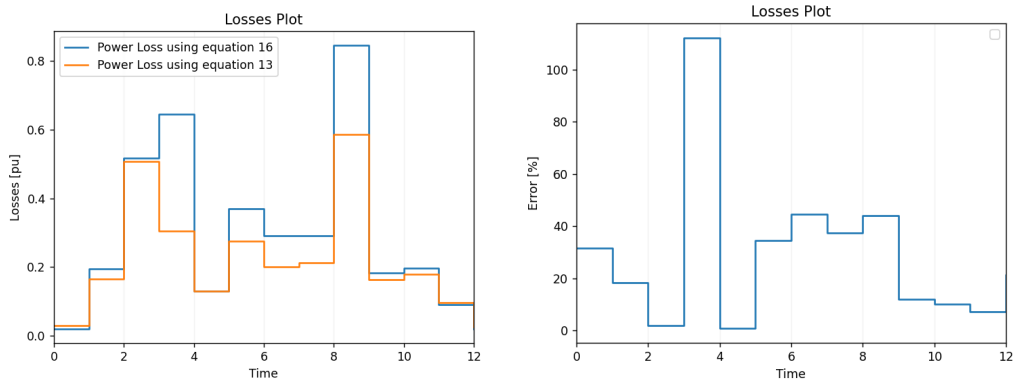

Figure 7: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Testing set.

By looking at both figures 6 and 7 it is easily visible that with the sizable reduction of matrix X, compared with the previous method, the errors for the prediction of the **Power Losses** where much higher. This is a case, where in the worst scenario, the error was over 90%, proving once again the trade-off between the accuracy of the model and the its computational difficulty. The average error was around 17 % for the **training test** and an average error of 26 % for the **test set**.

4.1.3 Summing electrically close bus injections

To further reduce the size of the matrix X, electrically close bus injections were considered. This reduction resulted in X having three inputs for the "kite" grid and can be considered as approximating buses 1, 2, and 3 to one "huge bus" and utilizing the outer-product expansion. In a distribution grid, this strategy can be utilized to treat each feeder as a "huge bus," greatly lowering computing complexity.

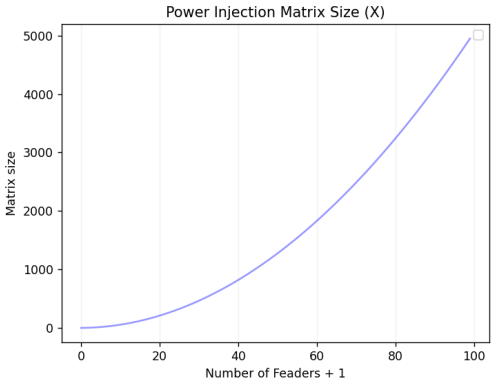

Figure 8: Summing electrically close bus injections of the power injections P matrix size, according to the number of buses the network.

To get to the **Power losses**, we followed the same steps on **Section 4.1.1** and obtained the following values for the **training set** and the **test set**:

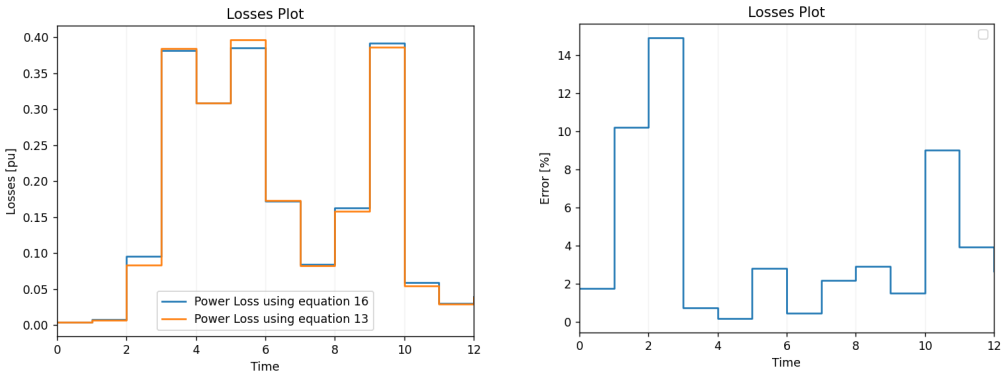

Figure 9: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Training set.

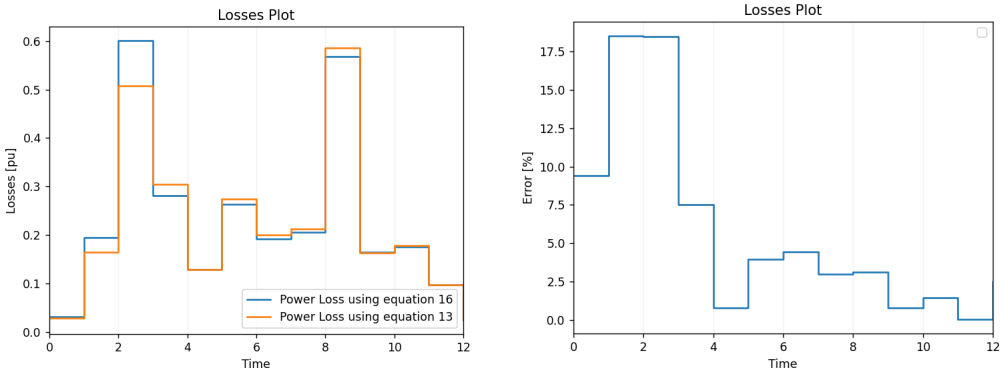

Figure 10: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Testing set.

From figures 9 and 10 it is visible that by preserving and using the **Power Injection** values, while still reducing the matrix X to only three rows, was possible to obtain much better values than the previous method. This shows us that to improve the accuracy of the utilized method. we need to not only focus on reducing the matrix, but also focus on searching for the right values for the power to use to calculate the **Loss function**. The average error was around 4 % for the **training test** and an average error of 5,5 % for the **test set**.

4.1.4 Considering the network structure

To reduce the size of the matrix X, we implemented a method where we considered the network structure. This means that we are only going to use 2*Pi*Pj when a line connection exists, on the outer expansion equation. Therefore the size of matrix X on this case will be dependent on the number of edges that connect each bus. On this case, since there are 4 connections and 4 buses, the matrix X will have size of 8. To get to the **Power losses**, we followed the same steps on **Section 4.1.1** and obtained the following values for the **training set** and the **test set**:

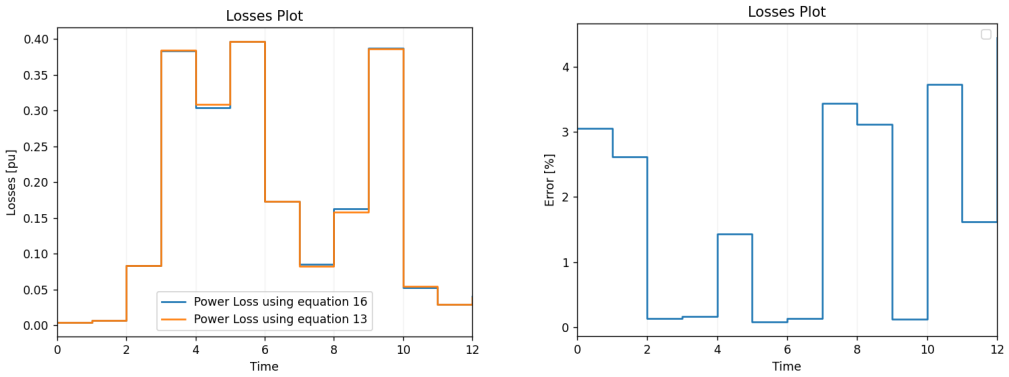

Figure 11: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Training set.

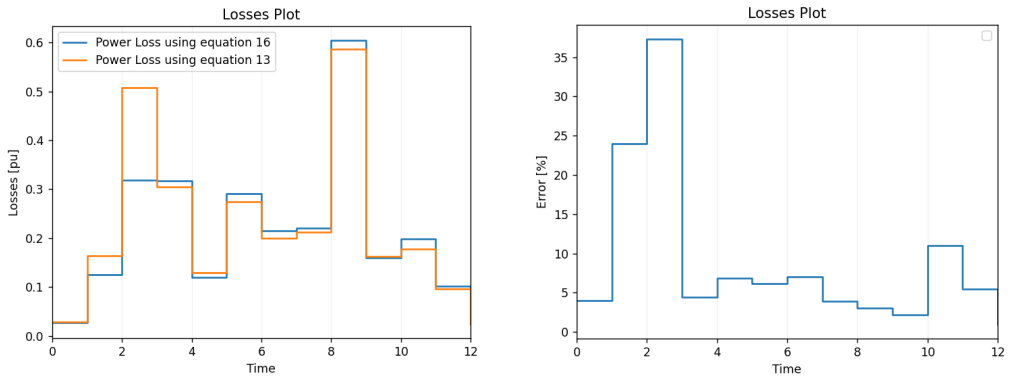

Figure 12: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), for the Testing set.

By looking at figure 11 and 12, we can perceive that this method becomes overfitted to the training data. The errors, for the **Power Losses**, regarding the train set, are minimal. Which at first glance

looks a good result, since we where able to reduce our matrix X and still preserve a small error. However, when we use the **Loss Function** achieved, to predict the unseen data from the **Test set**, it becomes visible that the method applied doesn't appear to have the same behavior. On this set the error becomes orders of magnitude higher. The average error was around 2 % for the **training test** and an average error of 9 % for the **test set**.

4.2 IEEE 33 bus Grid

The feasibility of adapting the proposed algorithm to a larger network will be explored in this section. The IEEE 33 bus system was employed in this study, where all load buses were assumed to be three-phase balanced. This is a typical medium voltage distribution grid and was schematically represented in Figure 13:

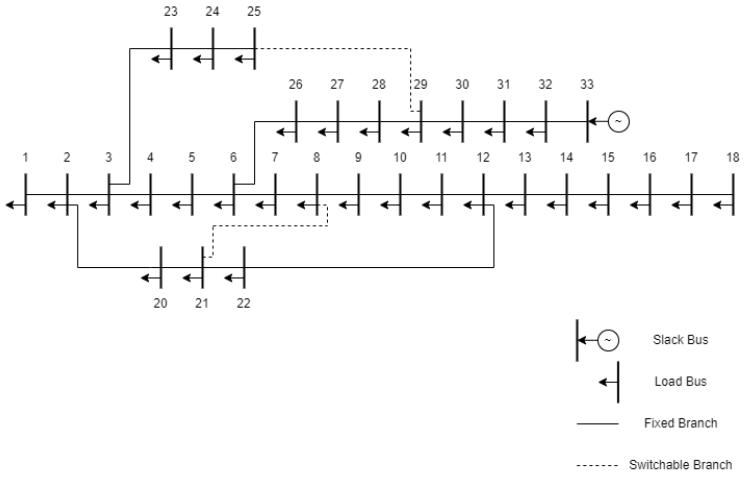

Figure 13: Schematic representation of the enhanced IEEE 33 bus distribution test system [2]

Note that bus 33 was designated as the slack bus for the benefit of ease. The grid's reactance to resistance (X/R) ratio of the branches is quite low (i.e. 0.33 to 3.31), as witnessed in practical distribution systems. Table 1 in the appendix displays the data for the proposed test benchmark's branches.

The data set was obtained by expanding the given training set, adding a normally distributed error, $\epsilon \sim \mathcal{N}(0.3, 0.2)$, and multiplying by a scaling factor, α, affecting the magnitude of the load. It should be noted that for the proposed active power demand profiles, the system will be heavily loaded, resulting in significant voltage drop and argument difference, ie, $V_i \not\approx V_j \not\approx 1$ and $\theta_{ij} > 1\ rad$, thus making the DC power flow approximation invalid.

4.2.1 Loss prediction under low and normal load conditions

The losses were computed making use of the quadratic functions of argument difference and magnitude difference:

$$P_L \approx \sum_{i \neq j} G_{ij}[1 - cos(\theta_i\ - \theta_i)]\ ,\ \theta = -B^{-1}P$$

$$P_L \approx \sum_{i \neq j} G_{ij}(V_i\ - V_i)^2\ ,\ V = G^{-1}P$$

Firstly, the system was tested for very light load conditions, with scaling factor, $\alpha = 0.0005$. The model was trained and losses were estimated using the training set. The voltage difference approach

produced higher predicted and exact values. Yet, there was no loss of performance because both models produced similar max mismatches of 5%.

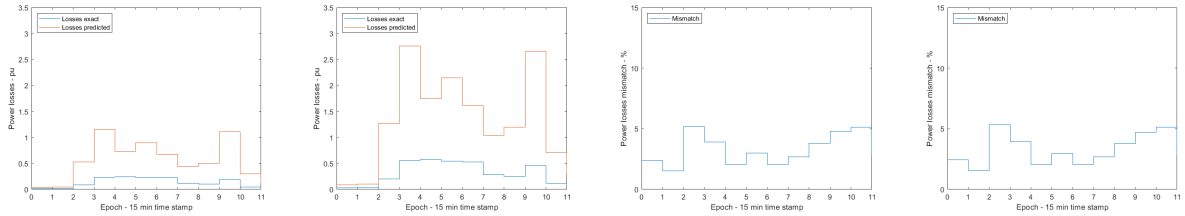

Figure 14: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), under low load circumstances.

When the system was tested under normal load conditions, $\alpha = 0.001$, all the previous observations held true. As the demand increased, so did the error, achieving max mismatches of 7%, significantly higher than the five bus "kite" grid example. Since losses are quadratic w.r.t load, the linear model becomes erratic for higher demand conditions. Note that, even though the scaling factor may assume a very small value, the losses were greater than in the previous grid, and the total power consumption was comparable to that proposed in [2].

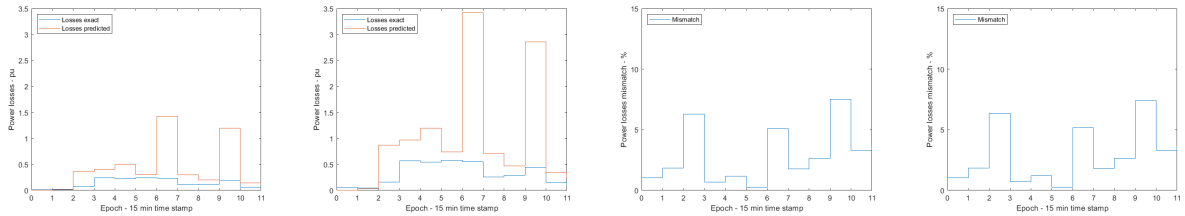

Figure 15: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), under normal load circumstances.

A slight error, $\epsilon \sim \mathcal{N}(0, 2 \times 10^{-3})$, was added to Y under normal load conditions. Although this noise level caused errors in study done in [1], this prediction was unaffected, and the mismatch was comparable to the one without mistake. This can be explained by the fact that the losses displayed were greater than those achieved for the "kite" grid, resulting in a smaller inaccuracy.

4.2.2 Power loss approximation validity

Since both methods yielded comparable mismatches, nothing can be said about the accuracy at face value. Both methods of predicting losses rely on power loss approximations, therefore the most precise approximation will result in the best estimation. To compare them, the power flow must be solved to determine the precise network state using a non-linear equation solving method. Jacobi-Gauss was chosen due to ease of implementation:

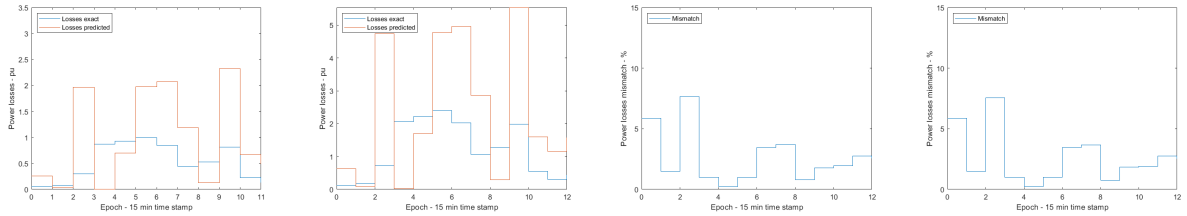

Figure 16: Loss prediction and mismatch calculated using argument difference (left) and magnitude difference (right), under normal load circumstances and error $\epsilon \sim \mathcal{N}(0,\, 2 \times 10^{-4})$

$$\begin{bmatrix} V_1 \\ \vdots \\ V_{32} \end{bmatrix}^{v+1} = [Y]^{-1} \begin{bmatrix} (\frac{S_1}{V_1})^* \\ \vdots \\ (\frac{S_{32}}{V_{32}})^* \end{bmatrix}^{v}$$

Upon obtaining precise nodal voltage values, the active power injected at the slack bus can be derived from the complex power injection, ie $\Re(V_{33}^* \sum_{j=1}^{33} Y_{33j} V_j)$, and losses may be described as the difference between generation and load.

$$P_L = \sum P_{generated} - \sum P_{load}$$

Under various load levels, the magnitude difference approximation yielded superior results. This result was already expected given the grid's resistive nature, the lack of transversal admittances (short line model). Given the grid's resemblance to an LV grid rather than an HV grid, the LV approximation proposed in [1] is more valid.

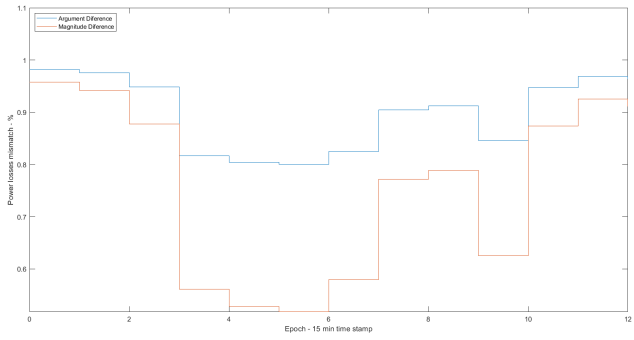

Figure 17: Error made by power loss approximation under normal load conditions

The model was tested using the most realistic losses approximation and a testing set generated in the same manner as the training set, using the same scaling factor, α. As expected, there was an

increase in the max prediction error from 7% to 13, 5%. The relative increase, 1, 92 was smaller than the one obtained in the Lectures Notes[1], 2, 5, but the percentage increase was bigger by 4%. It should be noted that when the model was trained with low load data and tested for normal to high load situations, very substantial errors were observed.

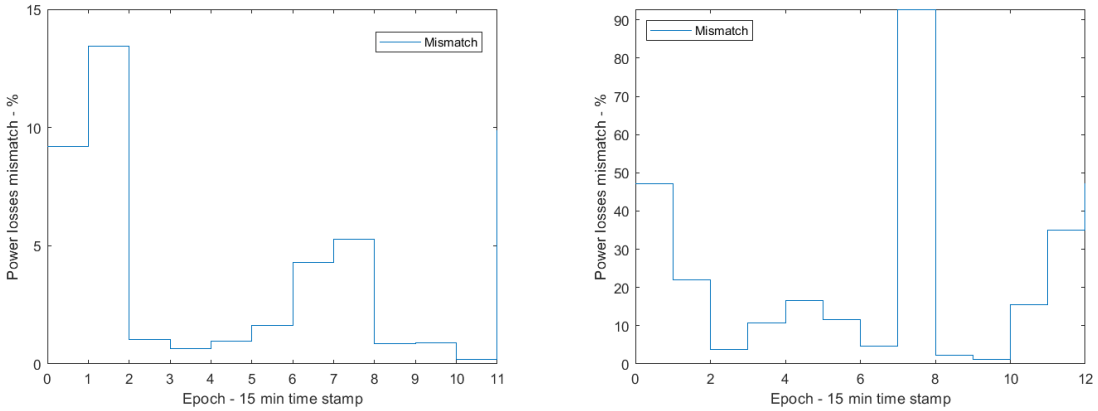

Figure 18: Loss prediction mismatch calculated for the testing set, under normal load circumstances, when the model was trained for normal load (left) and when the model was trained for low load(right)

4.2.3 Sensibility analyses on load magnitude

Many issues with the applicability of a model may arise, since it is necessary to thoroughly assess its assumptions, limitations, and performance against relevant data. Because losses vary quadratically with load, the loss function will vary substantially depending on the level of consumption. Sensitivity analysis on load magnitude is an effective way to determine the applicability of a model. The model was trained under ordinary load conditions before being tested for extremely low to extremely high power consumption.

The best results were obtained for similar scaling factors, indicating that the model is most effective for the load size for which it was trained. Since the training set loads are greater and more volatile, there is no perfect match, which explains why the minimum is attained where the scaling factor difference is $\frac{\alpha_{test}}{\alpha_{train}} = 0.556$.

4.2.4 Sensibility analyses on load volatility

Since load volatility can impair model performance, a second sensibility study was performed. The training set was changed to make it less consistent. The model was trained using increasingly smooth data and evaluated using the testing set, with scaling factors shared between sets. Loads below the

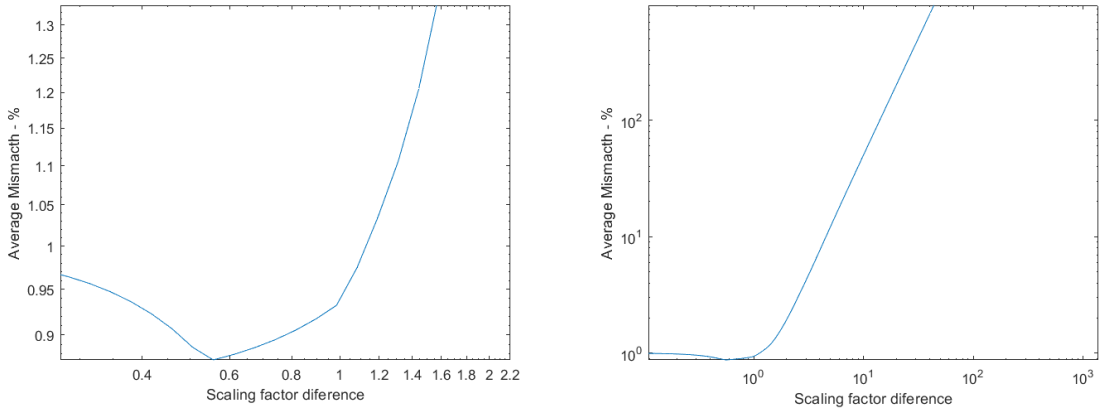

Figure 19: Average error in percentage in function of scaling factor disparity

average value are increased by 0.1α per iteration, while those above are decreased by the same amount.

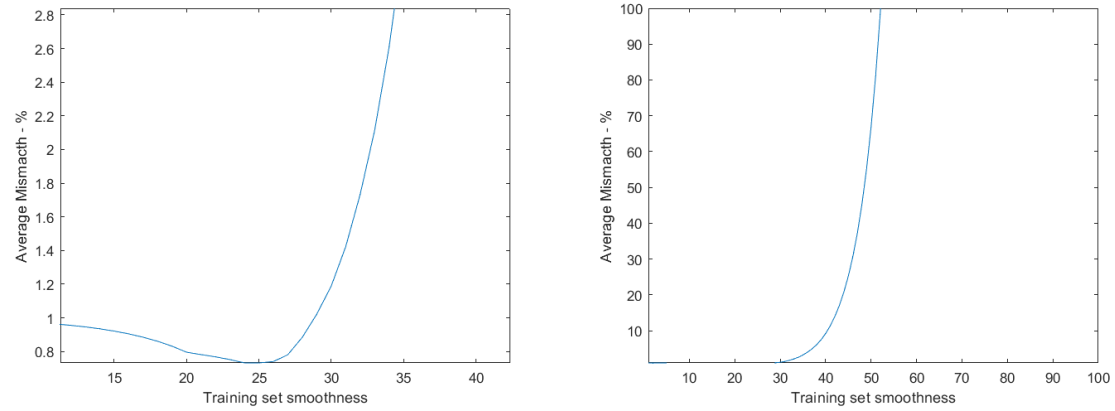

Figure 20: Average error in percentage in function of training set load volatility, low load conditions

The average inaccuracy reduced as volatility increased. The best performance was obtained when the load profiles of both sets were similar. It is worth noting that the loss functions derived with erratic load profiles result in consistent low inaccuracies ($< 1\%$), but the average error of models trainded with more consistent loads grew dramatically. This was expected because smoothing X increases the covariances of β, making the correlations between Y and X considerably more difficult to spot.

5 Conclusion

After testing the method and its versions on both grids, important observations must be made. It can clearly be stated, after carefully analyzing the results acquired throughout the assessment, that the availability of high resolution smart metering data allowed for far more accurate grid modeling in the absence of a comprehensive network model. However, this doesn't mean that the availability of this data, leads to solutions that are exact.

The **outer product expansion** method produced the best results by a wide margin. The complexity of the matrix X utilized in this strategy, however, increased quadratically with the number of buses added to the network, leading to a computational impossibility. Regarding the others, the one that presented a good trade between accuracy and performance was the **sum of close electrically bus connections**. By reducing the matrix X to size 3, it provided error values that where small enough. The **Squared Injections only** on the other end was a method that could only reduce the matrix X to a linear size and provided the worst results. Either it was on the training set and test set the error values was to big, for this method to be considered a good approximation. Lastly,when we considered the network structure, it provided good, small errors for the training set, but big ones when we computed the **Power Losses** for the test set. Taking all this in consideration, it is possible to say that the best method, among the tested ones, when we consider **accuracy**, was the **outer product expansion equation** and when we consider the **computational difficulty**, was the **sum of close electrically bus connections**.

The proposed method produced effective results in a MV distribution grid. Although the error was larger than on the "kite" grid, acceptable error was attained for the training set, testing set, and testing set with errors. This error could be reduced further by computing the exact power losses after solving the power flow or by taking advantage of the grid's characteristics and estimating the power losses using the quadratic expression of magnitude difference.

From the two sensibility analyses done, it should be noted that the losses model should be trained for the demand level it aims to predict. Volatile training sets produce good losses functions, whereas homogeneous training sets produce poor results.

References

1)**Lecture Notes: Chap 2.2 Losses Prediction**

2)**Dolatabadi, S.H.; Ghorbanian, M.; Siano, P.; Hatziargyriou, N.D. An enhanced IEEE 33 bus benchmark test system for distribution system studies. IEEE Trans. Power Syst. 2020, 36, 2565–2572.**

Appendix

A Branches data

Branch Number	From Bus	To Bus	Type	R (Ω)	X (Ω)
1	1	2	Fixed	0.0922	0.047
2	2	3	Fixed	0.493	0.2511
3	3	4	Fixed	0.366	0.1864
4	4	5	Fixed	0.3811	0.1941
5	5	6	Fixed	0.819	0.707
6	6	7	Fixed	0.1872	0.6188
7	7	8	Fixed	0.7114	0.2351
8	8	9	Fixed	1.03	0.74
9	9	10	Fixed	1.044	0.74
10	10	11	Fixed	0.1966	0.065
11	11	12	Fixed	0.3744	0.1238
12	12	13	Fixed	1.468	1.155
13	13	14	Fixed	0.5416	0.7129
14	14	15	Fixed	0.591	0.526
15	15	16	Fixed	0.7463	0.545
16	16	17	Fixed	1.289	1.721
17	17	18	Fixed	0.732	0.574
18	2	19	Fixed	0.164	0.1565
19	19	20	Fixed	1.5042	1.3554
20	20	21	Fixed	0.4095	0.4784
21	21	22	Fixed	0.7089	0.9373
22	3	23	Fixed	0.4512	0.3083
23	23	24	Fixed	0.898	0.7091
24	24	25	Fixed	0.896	0.7011
25	6	26	Fixed	0.203	0.1034
26	26	27	Fixed	0.2842	0.1447
27	27	28	Fixed	1.059	0.9337
28	28	29	Fixed	0.8042	0.7006
29	29	30	Fixed	0.5075	0.2585
30	30	31	Fixed	0.9744	0.963
31	31	32	Fixed	0.3105	0.3619
32	32	33	Fixed	0.341	0.5302
33	21	8	Switchable	2	2
34	12	22	Switchable	2	2
35	25	29	Switchable	0.5	0.5

Figure 21: Branches data of the enhanced IEEE 33 bus distribution test system [2]

B "Kite Grid" - Python Code

```
    #!/usr/bin/env python
# coding: utf-8
# # Problem 2
import os
import pandas as pd
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt

# <b>Parameters
noiseFactor = 0.0025  # noise
networkFactor = 100  # to change the characteristics of the network (Y)
PtestFactor = 3  # to obtain losses similar to the training data;

# <b>Import data (From Excel file)
direct = os.getcwd()
print(direct)
Info = np.array(pd.read_excel(r'C:\Users\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'C:\Users\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'C:\Users\DASG_Prob2_new.xlsx',
            sheet_name='Load(t,Bus)'))
Power_Info = np.delete(Power_Info, [0], 1)
print("Power consumption information (time, Bus) - (Train)\n", Power_Info, "\n")

# Power Information (test)
Power_Test = np.array(pd.read_excel(r'C:\Users\DASG_Prob2_new.xlsx',
                    sheet_name='Test_Load(t,Bus)'))
Power_Test = np.delete(Power_Test, [0], 1)
print("Power consumption information (time, Bus) - (Test)\n", Power_Test)

time = Power_Info.shape[0]
P = Power_Info
Ptest = Power_Test * PtestFactor

# 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")

# Create the vectors
C = np.zeros((nBus, nLines))
nLine_Aux = 0

# Determine the Incidence Matrix
for i in range(Y.shape[0]):
    for j in range(i + 1, Y.shape[1]):
        if np.absolute(Y[i, j]) != 0:
            C[i, nLine_Aux] = 1
            C[j, nLine_Aux] = -1
            nLine_Aux = nLine_Aux + 1

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

print("The incidence matrix C (nBus,nLines) is:\n", Cl)
```

```
# <b>Definition of Matrix Gij (Diagonal and vector)

# Create the vectors
Gv = np.zeros((1, nLines))
Gd = np.zeros((nLines, nLines))
nLine_Aux = 0

# Determine the Incidence Matrix
for i in range(Y.shape[0]):
    for j in range(i + 1, Y.shape[1]):
        if np.absolute(Y[i, j]) != 0:
            Gv[0, nLine_Aux] = -np.real(Y[i, j])
            Gd[nLine_Aux, nLine_Aux] = -np.real(Y[i, j])
            [Diagonal in matrix]
            nLine_Aux = nLine_Aux + 1

print("Gij_Diag:\n", Gd)

# Matrix creation
teta = np.zeros((nBus - 1, time))
grau = np.zeros((nLines, time))
PL = np.zeros(time)
PL2 = np.zeros(time)
PT = np.zeros(time)
rLoss = np.zeros(time)

# Losses
alfa = np.dot(np.dot(np.dot(np.dot(np.linalg.inv(B), Cl), Gd), np.transpose(Cl)),
np.linalg.inv(B))

for m in range(time):
    PL[m] = np.dot(P[m, :], np.dot(alfa, np.transpose(P[m, :])))

    teta[:, m] = np.dot(np.linalg.inv(B), np.transpose(P[m, :]))

    grau[:, m] = np.dot(np.transpose(Cl), teta[:, m])

    PL2[m] = np.dot(2 * Gv, 1 - np.cos(grau[:, m]))

    PT[m] = np.sum([P[m, :]])  # Total Power

    rLoss[m] = np.divide(PL2[m], PT[m])  # Power Losses (%)

print("Total Power consumption:\n", PT, "\n")
print("Power Losses obtained using the Theta:\n", PL2, "\n")
print("Power Losses obtained without using the Theta:\n", PL, "\n")


##############################################################################
```

```
##########                    My code                              ##########
################################################################################
def somatorio(x):
    if x == 1:
        return 1
    else:
        return x + somatorio(x - 1)


def plot(PLosses_compare, PLosses_):
    plt.plot(np.arange(len(PLosses_compare)), PLosses_compare, drawstyle='steps-post',
             label='Power Loss using equation 16')
    plt.plot(np.arange(len(PLosses_)), PLosses_, drawstyle='steps-post',
    label='Power Loss using equation 13')
    plt.grid(axis='x', color='0.95')
    plt.legend()
    plt.title('Losses Plot')
    plt.xlabel('Time')
    plt.ylabel('Losses [pu]')
    plt.xlim([0, 12])
    plt.show()

    abserror = abs(np.subtract(PLosses_compare, PLosses_))
    relerror = np.divide(abserror, PLosses_) * 100
    plt.plot(np.arange(len(relerror)), relerror, drawstyle='steps-post')

    plt.grid(axis='x', color='0.95')
    plt.legend()
    plt.title('Losses Plot')
    plt.xlabel('Time')
    plt.ylabel('Error [%]')
    plt.xlim([0, 12])
    plt.show()


def plot_bus_size():
    size = 100
    var = np.zeros(size)
    for a in range(size):
        if a == 0:
            var[a] = 0
        var[a] = var[a - 1] + a
        print(var[a])
    plt.plot(np.arange(size), var, color='blue', alpha=0.4)
    plt.grid(axis='x', color='0.95')
    plt.legend()
    plt.title('Power Injection Matrix Size (X)')
    plt.xlabel('Number of Buses')
    plt.ylabel('Matrix size')
```

```
    plt.show()
    exit()


### Training data
sizeX = somatorio(nBus - 1)
X = np.zeros((time, sizeX))

for i in range(time):
    auxSize = nBus - 1
    minimum = 0
    k = 0
    for j in range(sizeX):
        if k == minimum:
            X[i, j] = P[i, k] ** 2
        else:
            X[i, j] = P[i, minimum] * P[i, k] * 2
        k += 1
        if k == auxSize:
            minimum += 1
            k = minimum

Beta = np.dot(np.linalg.inv(np.dot(np.transpose(X), X)), np.dot(np.transpose(X), PL2))

# print("Vetor beta: \n", Beta)
Yaux = np.dot(X, Beta)
mu, sigma = 0, 0.0025
s = np.zeros(len(Yaux))

for i in range(len(Yaux)):
    s[i] = (1 + np.random.normal(mu, sigma, 1))

Plossesfinal_train = Yaux * s
plot(Plossesfinal_train, PL2)

# Testing data
for i in range(time):
    auxSize = nBus - 1
    minimum = 0
    k = 0
    for j in range(sizeX):
        if k == minimum:
            X[i, j] = Ptest[i, k] ** 2
        else:
            X[i, j] = Ptest[i, minimum] * Ptest[i, k] * 2
        k += 1
        if k == auxSize:
            minimum += 1
            k = minimum
```

```
# print("Vetor beta: \n", Beta)
Yaux = np.dot(X, Beta)
Plossesfinal_test = Yaux * s

# Losses
alfa = np.dot(np.dot(np.dot(np.dot(np.linalg.inv(B), Cl), Gd), np.transpose(Cl)),
np.linalg.inv(B))

for m in range(time):
    PL[m] = np.dot(Ptest[m, :], np.dot(alfa, np.transpose(Ptest[m, :])))

    teta[:, m] = np.dot(np.linalg.inv(B), np.transpose(Ptest[m, :]))

    grau[:, m] = np.dot(np.transpose(Cl), teta[:, m])

    PL2[m] = np.dot(2 * Gv, 1 - np.cos(grau[:, m]))

    PT[m] = np.sum([Ptest[m, :]])  # Total Power

    rLoss[m] = np.divide(PL2[m], PT[m])  # Power Losses (%)

print("Total Power consumption:\n", PT, "\n")
print("Power Losses obtained using the Theta:\n", PL2, "\n")
print("Power Losses obtained without using the Theta:\n", PL, "\n")

plot(Plossesfinal_test, PL2)
```

C IEEE 33 bus Grid - Matlab Code

```
%Power information - Training Set
P_train =    [0.332 0.064 0.084 0.12
             0.236 0.164 0.276 0.064
             0.224 0.708 1.572 0.072
             0.36  3.44  1.188 0.18
             1.332 2.176 0.484 1.464
             1.516 3.02  0.316 0.624
             0.92  0.916 0.404 2.772
             0.752 0.64  0.396 1.464
             1.828 0.684 0.576 0.576
             3.568 0.564 0.828 0.428
             0.78  0.356 0.728 0.348
             0.856 0.22  0.308 0.12
             0.684 0.528 0.256 0.44 ];

%Power information - Testing Set
P_test =     [0.18  0.06  0.188 0.132
```

```
            0.1   0.06  0.992 0.132
            0.388 0.1   1.576 0.132
            0.28  0.196 0.08  1.94
            0.5   0.184 0.064 0.572
            1.16  0.108 0.064 0.188
            1.036 0.06  0.068 0.08
            1.036 0.064 0.112 0.064
            1.768 0.064 0.2   0.068
            0.804 0.092 0.184 0.064
            0.824 0.196 0.088 0.096
            0.52  0.188 0.064 0.196
            0.184 0.104 0.068 0.192];


%Generate training data
P_Train_IEEE = zeros(13,32);

for k = 1:13
    for kk = 1:4
        P_Train_IEEE(k,kk)= P_train(k,kk);
        P_Train_IEEE(k,kk+4)= P_train(k,kk)+normrnd(0.3,0.2);  %8
        P_Train_IEEE(k,kk+8)= P_train(k,kk)+normrnd(0.3,0.2);  %12
        P_Train_IEEE(k,kk+12)= P_train(k,kk)+normrnd(0.3,0.2); %16
        P_Train_IEEE(k,kk+16)= P_train(k,kk)+normrnd(0.3,0.2); %20
        P_Train_IEEE(k,kk+20)= P_train(k,kk)+normrnd(0.3,0.2); %24
        P_Train_IEEE(k,kk+24)= P_train(k,kk)+normrnd(0.3,0.2); %28
        P_Train_IEEE(k,kk+28)= P_train(k,kk)+normrnd(0.3,0.2); %32
    end
end

for k = 1:13
    for kk = 1:32
        P_Train_IEEE(k,kk) = 0.0005*P_Train_IEEE(k,kk);
    end
end

%Generate testing data
P_Test_IEEE = zeros(13,32);

for k = 1:13
    for kk = 1:4
        P_Test_IEEE(k,kk)=  P_Test_IEEE(k,kk);
        P_Test_IEEE(k,kk+4)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2);  %8
        P_Test_IEEE(k,kk+8)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2);  %12
        P_Test_IEEE(k,kk+12)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %16
        P_Test_IEEE(k,kk+16)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %20
        P_Test_IEEE(k,kk+20)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %24
        P_Test_IEEE(k,kk+24)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %28
        P_Test_IEEE(k,kk+28)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %32
```

```
    end
end

for k = 1:13
    for kk = 1:32
         P_Test_IEEE(k,kk) = 0.00005* P_Test_IEEE(k,kk);
    end
end

%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;
Z1222 = 2 + 1i*2;

%Grid matrices - Slack Bus removed
Y = zeros(32,32);
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;
Y(8,9) = -1/Z89;

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

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

B = imag(Y);
G = real(Y);


%Incidence Matrix - Slack Bus removed (last line)
C = zeros (32,32);

C(1,1) = 1;
C(2,1) = -1;

C(2,2) = 1;
C(3,2) = -1;

C(3,3) = 1;
```

```
C(4,3) = -1;

C(4,4) = 1;
C(5,4) = -1;

C(5,5) = 1;
C(6,5) = -1;

C(6,6) = 1;
C(7,6) = -1;

C(7,7) = 1;
C(8,7) = -1;

C(8,8) = 1;
C(9,8) = -1;

C(9,9) = 1;
C(10,9) = -1;

C(10,10) = 1;
C(11,10) = -1;

C(11,11) = 1;
C(12,11) = -1;

C(12,12) = 1;
C(13,12) = -1;

C(13,13) = 1;
C(14,13) = -1;

C(14,14) = 1;
C(15,14) = -1;

C(15,15) = 1;
C(16,15) = -1;

C(16,16) = 1;
C(17,16) = -1;

C(17,17) = 1;
C(18,17) = -1;

C(2,18) = 1;
C(19,18) = -1;

C(19,19) = 1;
C(20,19) = -1;
```

```
C(20,20) = 1;
C(21,20) = -1;

C(21,21) = 1;
C(22,21) = -1;

C(3,22) = 1;
C(23,22) = -1;

C(23,23) = 1;
C(24,23) = -1;

C(24,24) = 1;
C(25,24) = -1;

C(6,25) = 1;
C(26,25) = -1;

C(26,26) = 1;
C(27,26) = -1;

C(27,27) = 1;
C(28,27) = -1;

C(28,28) = 1;
C(29,28) = -1;

C(29,29) = 1;
C(30,29) = -1;

C(30,30) = 1;
C(31,30) = -1;

C(31,31) = 1;
C(32,31) = -1;

C(12,32) = 1;
C(22,32) = -1;

C(32,33) = 1;



%Longitudinal conductance
Gij_diag = zeros(32,32);

Gij_diag(1,1) = real(1/Z12);
```

```
Gij_diag(2,2) = real(1/Z23);
Gij_diag(3,3) = real(1/Z34);
Gij_diag(4,4) = real(1/Z45);
Gij_diag(5,5) = real(1/Z56);
Gij_diag(6,6) = real(1/Z67);
Gij_diag(7,7) = real(1/Z78);
Gij_diag(8,8) = real(1/Z89);
Gij_diag(9,9) = real(1/Z910);
Gij_diag(10,10) = real(1/Z1011);
Gij_diag(11,11) = real(1/Z1112);
Gij_diag(12,12) = real(1/Z1213);
Gij_diag(13,13) = real(1/Z1314);
Gij_diag(14,14) = real(1/Z1415);
Gij_diag(15,15) = real(1/Z1516);
Gij_diag(16,16) = real(1/Z1617);
Gij_diag(17,17) = real(1/Z1718);
Gij_diag(18,18) = real(1/Z219);
Gij_diag(19,19) = real(1/Z1920);
Gij_diag(20,20) = real(1/Z2021);
Gij_diag(21,21) = real(1/Z2122);
Gij_diag(22,22) = real(1/Z323);
Gij_diag(23,23) = real(1/Z2324);
Gij_diag(24,24) = real(1/Z2425);
Gij_diag(25,25) = real(1/Z626);
Gij_diag(26,26) = real(1/Z2627);
Gij_diag(27,27) = real(1/Z2728);
Gij_diag(28,28) = real(1/Z2829);
Gij_diag(29,29) = real(1/Z2930);
Gij_diag(30,30) = real(1/Z3031);
Gij_diag(31,31) = real(1/Z3132);
Gij_diag(32,32) = real(1/Z1222);
Gij_diag(33,33) = real(1/Z3233);

%Longitudinal conductance, vector form
for k = 1:32
Gij(k) = Gij_diag(k,k);
end


%Compute Power Losses (Y)
theta = -B^-1*P_Train_IEEE.';
V = G^-1*P_Train_IEEE.';


thetaij = C.'*theta;
Vij = C.'*V;


for k = 1:33
```

```
    for kk = 1:13
        grau(k,kk) = 1-cos(thetaij(k,kk));
        tensao(k,kk) = Vij(k,kk)^2;
    end
end



PL1 = zeros(1,13);
PL2 = zeros(1,13);
for k =1:13
    for kk = 1:32
        PL1(k) = PL1(k) + 2*Gij(kk)*grau(kk,k);
        PL2(k) = PL1(k) + Gij(kk)*tensao(kk,k);
    end
end




%Discovering the loss function
X = zeros(13,10);

for k=1:13
    for kk = 1:32
        X2(k,kk) = P_Train_IEEE(k,kk)^2
    end
end

Y1 = PL1.';
beta1 = (X2.'*X2)^-1*X2.'*Y1;
losses_estimated1 = X2*beta1;

for k =1:13
    compare(k,1)=PL1(k);
    compare(k,2)=losses_estimated1(k);
end

for k =1:13
    compare2(k)=(PL1(k)-losses_estimated1(k))/PL1(k);

end


% %Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare2))
```

```
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses mismatch - %")
% axis([0 11 0 15])
% legend({'Mismatch'},'Location','northwest')
%
% %Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses - pu")
% axis([0 11 0 3.5])
% legend({'Losses exact','Losses predicted'},'Location','northwest')


Y2 = PL2.';
beta2 = (X2.'*X2)^-1*X2.'*Y2;
losses_estimated2 = X2*beta2;

for k =1:13
    compare(k,1)=PL2(k);
    compare(k,2)=losses_estimated2(k);
end

for k =1:13
    compare2(k)=(PL2(k)-losses_estimated2(k))/PL2(k);

end


%Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare2))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses mismatch - %")
% axis([0 11 0 15])
% legend({'Mismatch'},'Location','northwest')
%
% %Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses - pu")
% axis([0 11 0 3.5])
% legend({'Losses exact','Losses predicted'},'Location','northwest')
```

```
%POWER FLOW
%initial estimate
Y(33,32) = -1/Z3233;
Y(33,33) = 1/Z3233;
initial_state = zeros(33,1);

for i =1:33
v(i,1)=1;
end

for i =1:13
for k=1:1000
vb4 = v;
verror = 0;

    for kk=1:32
          v(kk) = conj(P_Train_IEEE(i,kk))/conj(vb4(kk));

        for kkk=1:33
            if(kkk==kk)
            v(kk) = v(kk);
            else
            v(kk) = v(kk) - Y(kk,kkk)*vb4(kkk);
            end
        end
        v(kk) = v(kk)/Y(kk,kk);
    end

  for kk=1:32
      verror = verror + abs(vb4(kk)-v(kk));
  end

  if(verror<=5*10^-4)            %Tolerance
      break
  end

end
initial_state = [initial_state v];
end

mag = abs(initial_state);
arg = angle(initial_state);

cons = zeros(13);
gen = zeros(13);
for k =1:13
    for kk = 1:32
        cons(k) = cons(k) + P_Train_IEEE(k,kk);
```

```
    end
end


for k = 2:14
    for kk = 1:33
        gen(k-1) = gen(k-1) + Y(33,kk)*initial_state(kk,k);
    end
     gen(k-1) = real( conj(initial_state(33,k))*gen(k-1));
end

for k =1:13
    PL_exact(k) = gen(k)-cons(k);
end

for k =1:13
    compare3(k,1)=abs((PL_exact-PL1(k))/PL_exact);
    compare3(k,2)=((PL_exact-PL2(k))/PL_exact);
end



% figure()
% ep=0:12;
% stairs(ep,abs(compare3))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses mismatch - %")
% axis([0 11 0.9 1.1])
% legend({'Argument Diference','Magnitude Diference'},'Location','northwest')




%Test
%model-----------------------------------------------------------------
for k=1:13
    for kk = 1:32
        X3(k,kk) = P_Test_IEEE(k,kk)^2
    end
end

V = G^-1*P_Test_IEEE.';

Vij = C.'*V;


for k = 1:33
    for kk = 1:13
        tensao(k,kk) = Vij(k,kk)^2;
```

```
    end
end



PL3 = zeros(1,13);
for k =1:13
    for kk = 1:32
        PL3(k) = PL1(k) + Gij(kk)*tensao(kk,k);
    end
end

losses_estimated3 = X3*beta2;

for k =1:13
    compare4(k,1)=PL2(k);
    compare4(k,2)=losses_estimated3(k);
end

for k =1:13
    compare5(k)=(PL2(k)-losses_estimated3(k))/PL2(k);

end


% %Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare5))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses mismatch - %")
% axis([0 11 0 15])
% legend({'Mismatch'},'Location','northwest')
%
% %Plot
% figure()
% ep=0:12;
% stairs(ep,abs(compare4))
% xlabel("Epoch - 15 min time stamp")
% ylabel("Power losses - pu")
% axis([0 11 0 3.5])
% legend({'Losses exact','Losses predicted'},'Location','northwest')



%Sensitivity-on--load-mag-------------------------------------------------
av = zeros(1,100);
diff = zeros(1,100);
```

```
for kkk = 1:100

    %Update testing data
    for k = 1:13
        for kk = 1:32
             P_Test_IEEE(k,kk) = 1.1* P_Test_IEEE(k,kk);
        end
    end

    if kkk == 1
        diff(kkk) = 0.1*1.1;
    else
        diff(kkk) = 1.1*diff(kkk-1);
    end


    for k=1:13
        for kk = 1:32
            X3(k,kk) = P_Test_IEEE(k,kk)^2;
        end
    end

    V = G^-1*P_Test_IEEE.';

    Vij = C.'*V;


    for k = 1:33
        for kk = 1:13
            tensao(k,kk) = Vij(k,kk)^2;
        end
    end



    PL3 = zeros(1,13);
    for k =1:13
        for kk = 1:32
            PL3(k) = PL1(k) + Gij(kk)*tensao(kk,k);
        end
    end

    losses_estimated3 = X3*beta2;


    for k =1:13
        av(kkk)=av(kkk) + abs(PL2(k)-losses_estimated3(k))/(13*PL2(k));
    end
```

```
end

% %Plot
% figure()
% ep=1:100;
% plot(diff,abs(av))
% set(gca, 'XScale', 'log')
% set(gca, 'YScale', 'log')
% xlabel("Scaling factor diference")
% ylabel("Average Mismacth - %")
% axis([1 100 0 3.5])




%Sensitivity-on--load-vol-------------------------------------------------
av2 = zeros(1,100);

%Power information - Training Set
P_train2 =    [0.332 0.004 0.084 3.12
             0.236 3.164 0.276 0.064
             2.224 0.708 3.572 0.072
             0.36  3.44  0.188 0.18
             1.332 2.176 0.484 1.464
             0.016 3.02  0.316 0.624
             0.92  0.916 0.404 2.972
             2.752 0.64  0.396 0.464
             0.828 0.684 0.576 2.576
             3.568 0.564 0.828 0.428
             0.78  0.356 0.728 3.348
             0.856 0.22  0.308 0.12
             2.684 0.528 0.256 0.44 ];


%Reset data

%Generate training data
P_Train_IEEE2 = zeros(13,32);

for k = 1:13
    for kk = 1:4
        P_Train_IEEE(k,kk)= P_train2(k,kk);
        P_Train_IEEE(k,kk+4)= P_train2(k,kk)+normrnd(0.3,0.2);  %8
        P_Train_IEEE(k,kk+8)= P_train2(k,kk)+normrnd(0.3,0.2);  %12
        P_Train_IEEE(k,kk+12)= P_train2(k,kk)+normrnd(0.3,0.2); %16
        P_Train_IEEE(k,kk+16)= P_train2(k,kk)+normrnd(0.3,0.2); %20
        P_Train_IEEE(k,kk+20)= P_train2(k,kk)+normrnd(0.3,0.2); %24
        P_Train_IEEE(k,kk+24)= P_train2(k,kk)+normrnd(0.3,0.2); %28
        P_Train_IEEE(k,kk+28)= P_train2(k,kk)+normrnd(0.3,0.2); %32
```

```
    end
end

for k = 1:13
    for kk = 1:32
        P_Train_IEEE2(k,kk) = 0.0005*P_Train_IEEE2(k,kk);
    end
end

%Generate testing data
P_Test_IEEE = zeros(13,32);

for k = 1:13
    for kk = 1:4
        P_Test_IEEE(k,kk)=  P_Test_IEEE(k,kk);
        P_Test_IEEE(k,kk+4)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2);  %8
        P_Test_IEEE(k,kk+8)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2);  %12
        P_Test_IEEE(k,kk+12)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %16
        P_Test_IEEE(k,kk+16)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %20
        P_Test_IEEE(k,kk+20)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %24
        P_Test_IEEE(k,kk+24)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %28
        P_Test_IEEE(k,kk+28)=  P_Test_IEEE(k,kk)+normrnd(0.3,0.2); %32
    end
end

    for k = 1:13
        for kk = 1:32
             P_Test_IEEE(k,kk) = 0.0005* P_Test_IEEE(k,kk);
        end
    end

    for k=1:13
        for kk = 1:32
            X3(k,kk) = P_Test_IEEE(k,kk)^2;
        end
    end

%Average value - for smoothing
    ave = 0;
         for k=1:13
            for kk = 1:32
                ave = ave + P_Test_IEEE(k,kk);
            end
         end
         ave = ave / (k*kk);


for kkk = 1:100
```

```
%Update training data
 if kkk > 1

     for k=1:13
        for kk = 1:32
            if P_Train_IEEE2(k,kk)> ave
               P_Train_IEEE2(k,kk) = P_Train_IEEE2(k,kk) - 0.1*0.0005;
            else
               P_Train_IEEE2(k,kk) = P_Train_IEEE2(k,kk) + 0.1*0.0005;
            end
        end
     end

 end

 for k=1:13
    for kk = 1:32
        X2(k,kk) = P_Train_IEEE2(k,kk)^2;
    end
end

diff(kkk) = kkk;

%Power losses aproxx based on training set
V = G^-1*P_Train_IEEE2.';

Vij = C.'*V;

for k = 1:33
    for kk = 1:13
        tensao(k,kk) = Vij(k,kk)^2;
    end
end

PL3 = zeros(1,13);
for k =1:13
    for kk = 1:32
        PL3(k) = PL3(k) + Gij(kk)*tensao(kk,k);
    end
end


%Train Model
Y3 = PL3.';
beta3 = (X2.'*X2)^-1*X2.'*Y3;

%Test Model
losses_estimated3 = X3*beta3;
```

```
    %Average error in %
    for k =1:13
        av2(kkk)=av2(kkk) + abs(PL3(k)-losses_estimated3(k))/(13*PL2(k));
    end

end

%Plot
figure()
ep=1:100;
plot(diff,abs(av))
% set(gca, 'XScale', 'log')
% set(gca, 'YScale', 'log')
xlabel("Training set smoothness")
ylabel("Average Mismacth - %")
axis([1 100 1 100])
```