

POLITECNICO DI TORINO

MASTER'S DEGREE IN ELECTRICAL ENGINEERING

MASTER'S DEGREE THESIS



**Politecnico  
di Torino**

# Development of Acquisition Codes for the Execution and Post-Elaboration of Standard Tests on Induction Motors

*Supervisors:*

Prof. Silvio VASCHETTO  
Prof. Andrea CAVAGNINO  
Eng. Marco BIASION

*Candidate:*

Federico LUCCA

A.Y. 2020/2021

# Contents

|                                                                                        |           |
|----------------------------------------------------------------------------------------|-----------|
| <b>List of Figures</b>                                                                 | <b>3</b>  |
| <b>List of Tables</b>                                                                  | <b>5</b>  |
| <b>1 Introduction</b>                                                                  | <b>7</b>  |
| 1.1 Objective of the Thesis . . . . .                                                  | 7         |
| 1.2 Python programming language . . . . .                                              | 8         |
| <b>2 Standard Test on Induction Motors</b>                                             | <b>11</b> |
| 2.1 Preliminary tests . . . . .                                                        | 11        |
| 2.2 No Load test . . . . .                                                             | 12        |
| 2.3 Locked Rotor test . . . . .                                                        | 12        |
| <b>3 Power Meter HIOKI PW3337</b>                                                      | <b>14</b> |
| 3.1 Instrument Specifications . . . . .                                                | 15        |
| 3.2 Remote operation . . . . .                                                         | 16        |
| <b>4 Acquisition and Execution Tool</b>                                                | <b>17</b> |
| 4.1 The Python library Tkinter . . . . .                                               | 17        |
| 4.2 Acquisition GUI . . . . .                                                          | 18        |
| 4.2.1 LAN Connection . . . . .                                                         | 19        |
| 4.2.2 File name . . . . .                                                              | 20        |
| 4.2.3 Motor data . . . . .                                                             | 22        |
| 4.2.4 Test settings . . . . .                                                          | 23        |
| 4.2.5 Reference value of the winding resistance at the reference temperature . . . . . | 24        |
| 4.2.6 Winding resistance at the beginning of the test . . . . .                        | 25        |
| 4.2.7 Power meter settings . . . . .                                                   | 25        |
| 4.2.8 Acquisition section . . . . .                                                    | 29        |
| 4.2.9 Resistance at the end of the test . . . . .                                      | 32        |
| 4.3 Multiple acquisition . . . . .                                                     | 32        |
| <b>5 Post-elaboration Tool</b>                                                         | <b>34</b> |
| 5.1 Determination of the parameters of the steady-state equivalent circuit .           | 34        |
| 5.1.1 Calculation of the stator winding resistance . . . . .                           | 35        |
| 5.1.2 Elaboration of the No Load test results . . . . .                                | 41        |
| 5.1.3 Elaboration of the Locked Rotor test results . . . . .                           | 45        |
| 5.1.4 Interpolation strategy . . . . .                                                 | 48        |

---

|          |                                                            |           |
|----------|------------------------------------------------------------|-----------|
| 5.2      | Elaboration GUI . . . . .                                  | 50        |
| 5.2.1    | Interface for the No Load and Locked Rotor tests . . . . . | 51        |
| 5.2.2    | Elaboration of the test results . . . . .                  | 52        |
| <b>6</b> | <b>Experimental validation</b>                             | <b>55</b> |
| 6.1      | Measurement setup . . . . .                                | 55        |
| 6.1.1    | The Induction Motors used for the tests . . . . .          | 55        |
| 6.1.2    | The Three Phase Supply source . . . . .                    | 57        |
| 6.1.3    | The measurement instruments . . . . .                      | 57        |
| 6.2      | Verification methodology . . . . .                         | 59        |
| 6.3      | Experimental Results . . . . .                             | 59        |
| 6.3.1    | Elaboration tool validation . . . . .                      | 60        |
| 6.3.2    | Acquisition tool validation . . . . .                      | 60        |
| <b>7</b> | <b>Conclusion</b>                                          | <b>65</b> |
| <b>8</b> | <b>Appendix</b>                                            | <b>66</b> |
| 8.1      | A. Acquisition tool Python codes . . . . .                 | 66        |
| 8.2      | B. Elaboration tool Python codes . . . . .                 | 82        |

# List of Figures

|      |                                                                                                                                 |    |
|------|---------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | EV market trend[2]. . . . .                                                                                                     | 7  |
| 1.2  | Programming Language trends [3]. . . . .                                                                                        | 8  |
| 1.3  | Comparison between Python and Java. . . . .                                                                                     | 9  |
| 1.4  | IEEE Spectrum Top Programming Languages 2021. . . . .                                                                           | 9  |
| 2.1  | equivalent circuit single-phase of induction machines. . . . .                                                                  | 11 |
| 3.1  | <i>PW337</i> Front panel of the instrument . . . . .                                                                            | 14 |
| 3.2  | <i>PW337</i> Rear panel of the instrument . . . . .                                                                             | 14 |
| 3.3  | Power Meter <i>PW337</i> by Hioki: display commands. . . . .                                                                    | 15 |
| 3.4  | current measure limits. . . . .                                                                                                 | 15 |
| 3.5  | Scheme of principle of the measurement setup. . . . .                                                                           | 16 |
| 4.1  | A simple GUI interface with one button. . . . .                                                                                 | 18 |
| 4.2  | Acquisition GUI. . . . .                                                                                                        | 18 |
| 4.3  | The red lamp on the front panel of the instrument displays after the<br>successful remote connection to the instrument. . . . . | 19 |
| 4.4  | LAN connection section. . . . .                                                                                                 | 19 |
| 4.5  | File name section. . . . .                                                                                                      | 20 |
| 4.6  | Error message for the no test type selection. . . . .                                                                           | 21 |
| 4.7  | The motor data section. . . . .                                                                                                 | 22 |
| 4.8  | Test section. . . . .                                                                                                           | 23 |
| 4.9  | DC resistance measure. . . . .                                                                                                  | 24 |
| 4.10 | winding resistance at the reference temperature section. . . . .                                                                | 24 |
| 4.11 | Winding resistance at the end of the test. . . . .                                                                              | 25 |
| 4.12 | Power meter setting section. . . . .                                                                                            | 25 |
| 4.13 | Aron connection. . . . .                                                                                                        | 26 |
| 4.14 | Acquisition section. . . . .                                                                                                    | 29 |
| 4.15 | Example text file at the end of the Acquisition tool. . . . .                                                                   | 30 |
| 4.16 | The resistance measurements after the test. . . . .                                                                             | 32 |
| 4.17 | Multiple Acquisition settings. . . . .                                                                                          | 33 |
| 5.1  | Electrical single-phase equivalent steady-state circuit. . . . .                                                                | 34 |
| 5.2  | equivalent circuit of the wye-connected stator winding during the DC test. . . . .                                              | 35 |
| 5.3  | Equivalent circuit of the delta-connected stator winding during the DC<br>test. . . . .                                         | 36 |
| 5.4  | Stator windings resistance in case of delta connection. . . . .                                                                 | 36 |
| 5.5  | Variation of the stator winding temperature during the tests. . . . .                                                           | 39 |

|      |                                                                          |    |
|------|--------------------------------------------------------------------------|----|
| 5.6  | Variation of the stator winding resistance during the tests. . . . .     | 39 |
| 5.7  | Linearization of the stator winding resistance during the tests. . . . . | 40 |
| 5.8  | No Load single-phase equivalent circuit. . . . .                         | 41 |
| 5.9  | Vector Diagram. . . . .                                                  | 42 |
| 5.10 | Mechanical and core losses with respect to the supply voltage. . . . .   | 43 |
| 5.11 | Equivalent circuit for the Locked Rotor test. . . . .                    | 46 |
| 5.12 | Reference temperature choice. . . . .                                    | 46 |
| 5.13 | interpolation interface. . . . .                                         | 49 |
| 5.14 | The Post-Elaboration GUI. . . . .                                        | 50 |
| 5.15 | No Load GUI. . . . .                                                     | 51 |
| 5.16 | Locked Rotor GUI. . . . .                                                | 51 |
| 5.17 | Open file option in the No load tool menu bar. . . . .                   | 52 |
| 5.18 | Selection of .VTO text file. . . . .                                     | 52 |
| 5.19 | The No Load interface after the .VTO selection. . . . .                  | 53 |
| 5.20 | Example of Table data for the No Load test. . . . .                      | 53 |
| 5.21 | Example of Table data for the Locked Rotor test. . . . .                 | 54 |
| 5.22 | Example of test results table. . . . .                                   | 54 |
| 6.1  | The induction motors used for the tests. . . . .                         | 55 |
| 6.2  | Laboratory setup. . . . .                                                | 57 |
| 6.3  | The TPS power supplier. . . . .                                          | 57 |
| 6.4  | DC measurements setup. . . . .                                           | 58 |
| 6.5  | Switch with a safety button. . . . .                                     | 58 |
| 6.6  | Verifying methodology scheme. . . . .                                    | 59 |
| 6.7  | Elaboration tool validation methodology scheme. . . . .                  | 60 |
| 6.8  | Acquisition tool validation methodology scheme. . . . .                  | 60 |
| 6.9  | No Load E.m.f comparison. . . . .                                        | 61 |
| 6.10 | No Load current comparison. . . . .                                      | 62 |
| 6.11 | Mechanical and Iron Power losses comparison. . . . .                     | 62 |
| 6.12 | No Load E.m.f comparison. . . . .                                        | 63 |
| 6.13 | No Load current comparison. . . . .                                      | 64 |

# List of Tables

|     |                                                                      |    |
|-----|----------------------------------------------------------------------|----|
| 6.1 | FIMET MA160M4 11 kW Motor data. . . . .                              | 56 |
| 6.2 | FIMET HMA160L4 15 kW Motor data. . . . .                             | 56 |
| 6.3 | FIMET MA180L4 18.5 kW Motor data. . . . .                            | 56 |
| 6.4 | No Load parameters comparison for the 18.5 kW induction motor. . . . | 61 |
| 6.5 | Locked Rotor parameters comparison for the 18.5 kW motor. . . . .    | 63 |

# Abstract

This thesis develops the acquisition codes for the execution and post-elaboration of Standard tests on induction motors. The aim is to create a tool using the open-source programming language Python. This replaces an already existing Visual Basic application which operation is problematic with the most up-to-date Windows operating systems. The Tool is divided in two parts:

- the *Acquisition Tool*: allows the user to remotely control the power meter used for executing the measurements;
- the *Elaboration Tool*: elaborates the tests results and determines the parameters of the steady-state equivalent circuit of induction motors.

Moreover, the Python codes implement new features to improve and simplify the acquisition process.

The Python code is successfully used in the laboratory for performing the Standard tests on three different induction motors. The Elaboration tool is validated comparing the results with those obtained with former Visual Basic application. This thesis demonstrates the excellent capabilities of Python in realizing the acquisition codes for the execution and post-elaboration of standard tests on induction motors.

# Chapter 1

## Introduction

Based on different researches, the electric machine market is increasing continuously every year. The forecast expects growth from an estimated USD 113.3 billion (2020) to USD 169.1 billion in 2026 [1]. The number of electric cars (EV) is on course to increase from 11million vehicles to 145million by the end of the decade. As shown in Figure 1.1, the EV market is increasing drastically in the main region. A report by the International Energy Agency has found that if governments agreed to encourage the production of enough low-carbon vehicles to stay within global climate targets there could be 230million electric vehicles worldwide by 2030 [1].

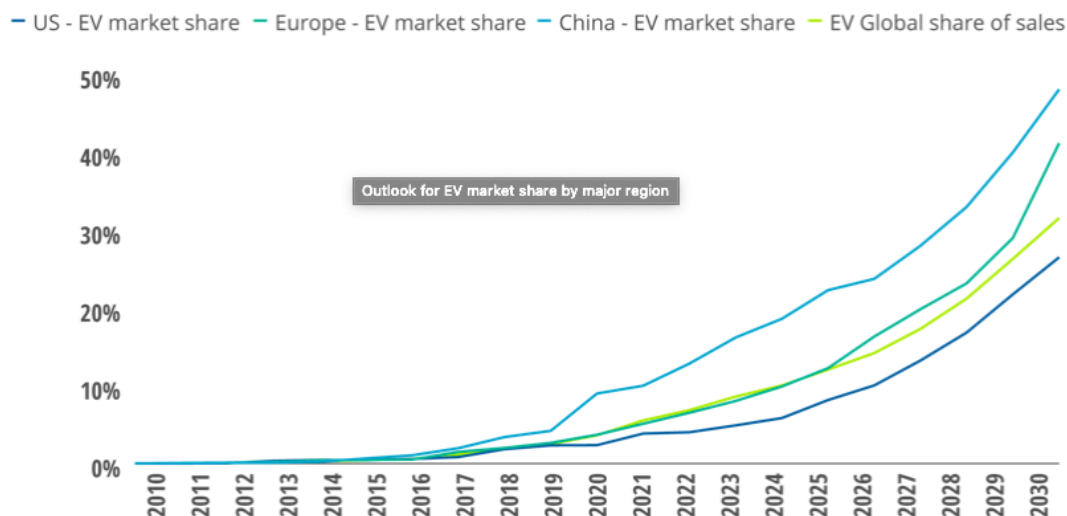


Figure 1.1: EV market trend[2].

### 1.1 Objective of the Thesis

For these reasons, the number of standard tests on the motors will increase drastically and the automation of the standard tests procedure will be necessary. The aim of this master thesis work is to create a Tool, using the open-source programming language Python, to automate the Acquisition and Post-Elaboration of Standard Tests on Induction Motors. This replaces and improves an already existing Visual Basic application which operation is problematic with the most up-to-date Windows operating



systems. Moreover, the Python codes implement new features to improve and simplify the acquisition process.

The Tool is divided in two parts:

- the *Acquisition Tool*: allows the user to remotely control the power meter used for executing the measurements;
- the *Elaboration Tool*: elaborates the tests results and determines the parameters of the steady-state equivalent circuit of induction motors.

These parts of the Tool will be described respectively in the Chapter 4 and Chapter 5.

## 1.2 Python programming language

The choice of Python as the programming language is due to its extremely increase of use in the industrial sector, as shown in Figure 1.2, regarding the web traffic on the *Stack Overflow* [3]. In Figure, the Python trend is compared with the other main programming languages trends in the last decade.

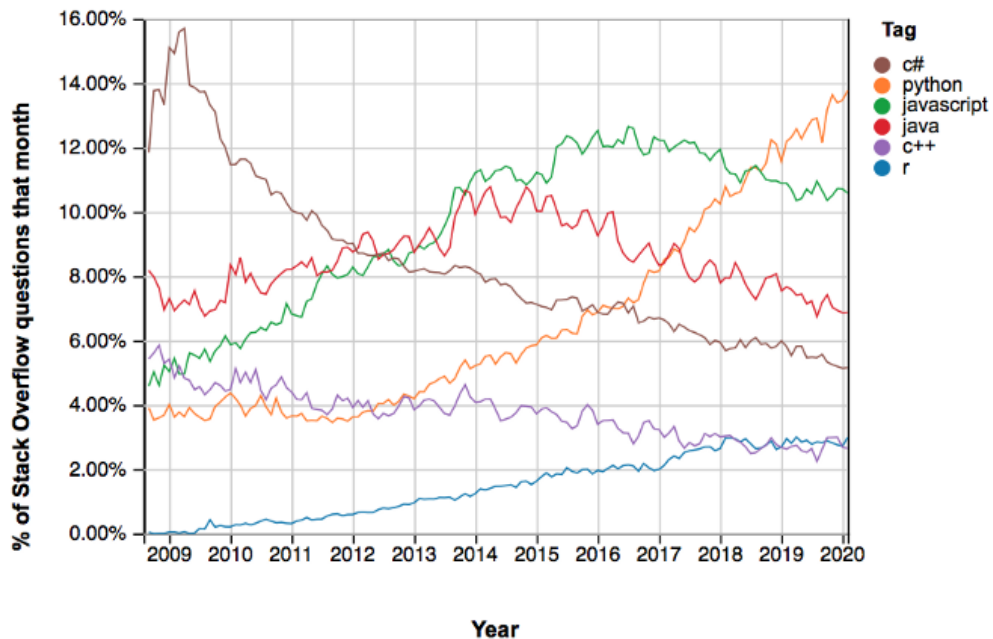


Figure 1.2: Programming Language trends [3].

Based on the PYPL (PopularitY of Programming Language) index, shown in Figure 1.3, Python surpassed the Google research of Java, becoming the most popular programming language in the Google search bar.

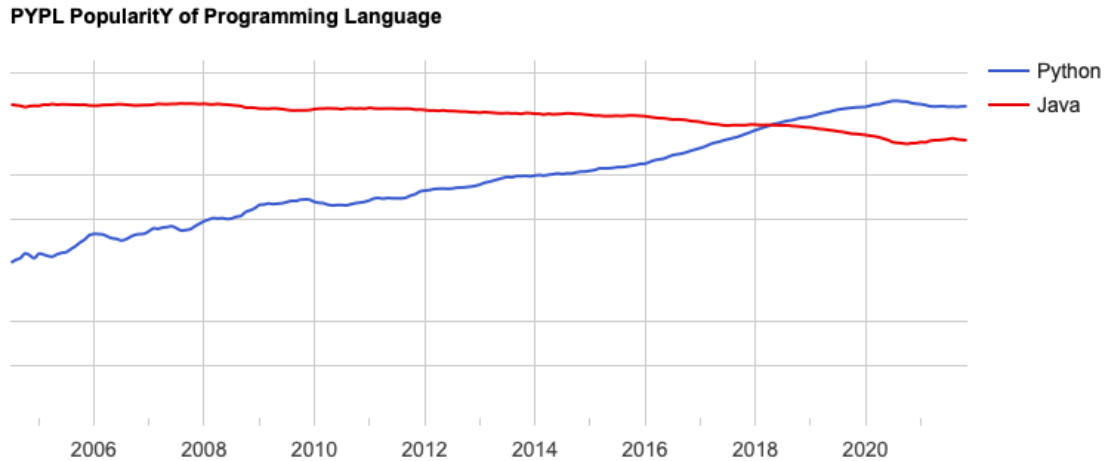


Figure 1.3: Comparison between Python and Java.

Moreover, the IEEE Spectrum places Python in the first position for its "Top Programming Languages 2021"[4]. The IEEE Spectrum used eleven metrics from eight different sources to achieve an overall ranking of language popularity. To understand better the setting metric process refer to the website in the reference [4].

| Language Ranking: IEEE Spectrum |            |      |   |    |       |
|---------------------------------|------------|------|---|----|-------|
| Rank                            | Language   | Type |   |    | Score |
| 1                               | Python     | 🌐    | 💻 | ⚙️ | 100.0 |
| 2                               | Java       | 🌐    | 📱 | 💻  | 95.4  |
| 3                               | C          |      | 📱 | 💻  | 94.7  |
| 4                               | C++        |      | 📱 | 💻  | 92.4  |
| 5                               | JavaScript | 🌐    |   |    | 88.1  |
| 6                               | C#         | 🌐    | 📱 | 💻  | 82.4  |
| 7                               | R          |      |   | 💻  | 81.7  |
| 8                               | Go         | 🌐    |   | 💻  | 77.7  |
| 9                               | HTML       | 🌐    |   |    | 75.4  |
| 10                              | Swift      |      | 📱 | 💻  | 70.4  |

Figure 1.4: IEEE Spectrum Top Programming Languages 2021.

The main Python advantage are:

- **Free and open source:** Python has an open source licence. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behavior and use their version for development.
- **Simple to use:** Code readability and simple user-friendly designs are important aspects of a programming language. Python employs a neat, clean and well-structured design for easy understanding and usage.
- **Vast library support:** The main features of python is the vastity of its standard library, where it's possible to find all the function needed. For example *Numpy* is one of the main OpenSource libraries in Python. This library allows to work to the vectors and matrices and it's the principal library for mathematical applications.
- **Portable across Operating systems:** Python is designed to be highly portable. It is supported by all the operating systems, Windows, Linux, UNIX and macOS. Python code can run on different OS and environments without requiring any modifications.

# Chapter 2

## Standard Test on Induction Motors

This Chapter describes all the fundamentals of the Standard Tests on the induction motors. The methodology follows the procedure written on the "*IEEE Standard Test Procedure for Polyphase Induction Motors and Generators*"[5].

The Standard Tests allow the user to determine the main machine parameters which are useful for the steady-state and dynamic analysis of the machine, for example, determination of the machine performance. The tests request for this scope are:

1. Preliminary tests
2. No Load test
3. Locked Rotor test

The voltages and the currents are expressed to a single winding, while the power is considered for all the induction motor. The single-phase equivalent circuit of the induction machine is shown in the Figure 2.1:

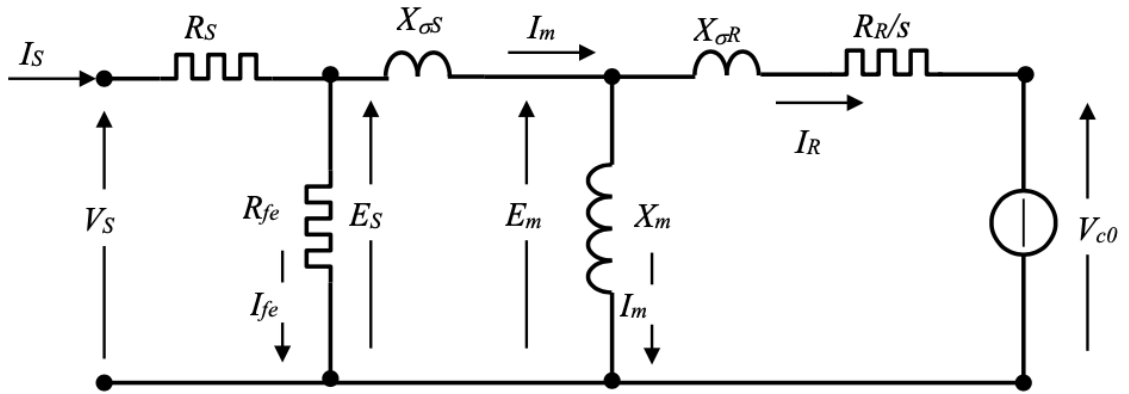


Figure 2.1: equivalent circuit single-phase of induction machines.

### 2.1 Preliminary tests

The winding resistance measurement is commonly the first test performed. It is important to distinguish this type of test based on the power of the motor. With

motors under 10kW of power the volt-amperometric method is sufficient to obtain a precise resistance measure, otherwise it is necessary a better measurement method, as confronting one.

It is important that this type of measurements must be executed with the machine at rest, in order to have the windings at the ambient temperature.

The winding resistance must be measured for each winding. At the end of the test, the resistance parameter  $R_s$  of the model will be the average values of the three resistance windings measurements.

## 2.2 No Load test

The No Load test is performed by running the machine as a motor with no connected load at rated voltage and frequency. When separation of no-load losses is to be accomplished, run this test and record voltage, current, temperature and power input at rated frequency [5]. The speed during the No Load test is close to the synchronous speed because the resistance torque due to the friction and ventilating effects are modest.

The voltage supply must be symmetric, the presence of inverse components can compromise the calculation of the parameter. To obtain all the information from the No Load test, the IEEE document recommends:

- Test at three or more values of voltages between 125% and 75% of the rated voltage, with a point near 100% rated voltage
- Three or more values of voltage between 50% of rated voltage and 20% of rated voltage or to that point where further voltage reduction increases the current.

## 2.3 Locked Rotor test

For this type of Standard Test is necessary to lock the rotor shaft and supply the machine with a reduced voltage than the rated one. The impedance under this testing condition is very low. For these reasons, the IEEE documents recommend effectuating the test with a few measurements:

- $\frac{1}{4}I_{rated}$
- $\frac{1}{2}I_{rated}$
- $\frac{3}{4}I_{rated}$
- $I_{rated}$

It's not necessary to execute the Locked Rotor test with current values major than the rated current of the machine, to avoid excessive windings overheating. In fact, because one of the scopes of the test is to determine the rotor resistance, it's convenient not to alter the temperature of the windings. In this way is possible to refer the

resistance measurement to a temperature with more precision. It should be recognized that the testing of induction machines under locked-rotor conditions with polyphase power involves high mechanical stresses and high rates of heating. Therefore, the following is necessary [5]:

1. Make sure that the machine and locking the rotor are of adequate strength to prevent possible injury to personnel or damage to equipment;
2. Identify the direction of the rotation before starting the test;
3. The machine is at approximately ambient temperature before the test is started.

The current and torque readings shall be taken as quickly as possible, and to obtain representative values, the machine temperature should not exceed the rated temperature rise plus 40°C. The readings for any point shall be taken within 5 s after test voltage is reached.

## Chapter 3

# Power Meter HIOKI PW3337

This Chapter describes the power meter PW3337 by the HIOKI company used during the thesis work. The instrument is used in the acquisition procedure to measure the data from the Standard Test on the induction motors.

The *PW337* is a power meter with power measurement capabilities for the full range of electrical equipment, from single-phase devices such as battery-driven devices and household electronics to industrial use and three-phase electrical equipment[6].

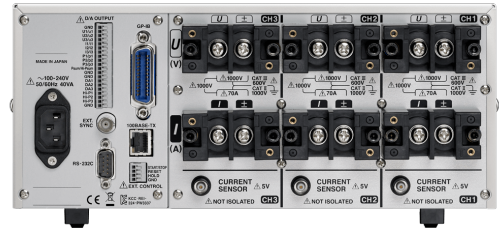
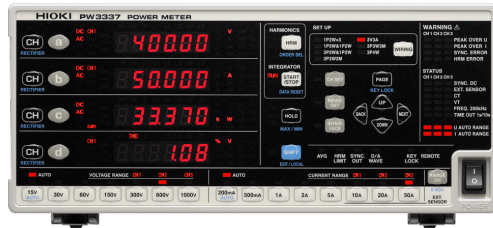
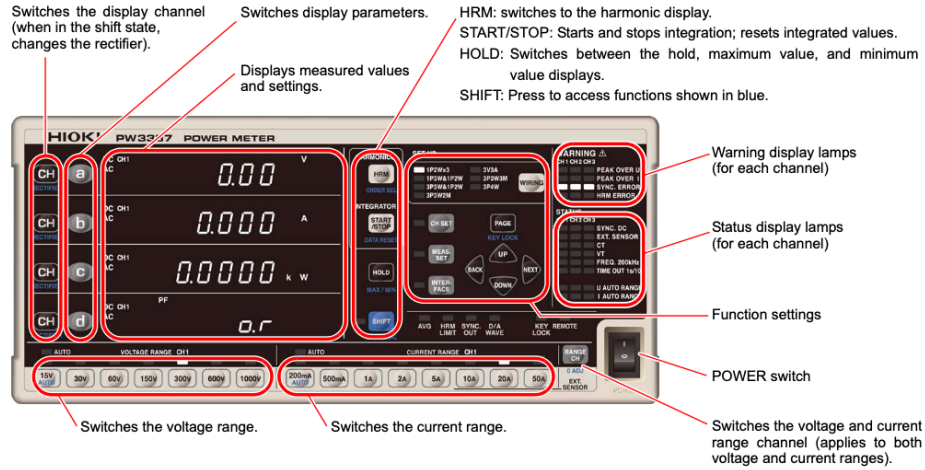


Figure 3.1: *PW337* Front panel of the instru- Figure 3.2: *PW337* Rear panel of the instru-  
ment ment

Figure 3.3 shows the main features of the parts on the front panel of the wattmeter. Further details are reported in the Acquisition tool as described in chapter 4.

Figure 3.3: Power Meter *PW3337* by Hioki: display commands.

### 3.1 Instrument Specifications

The main features of the power meter *PW3337* are[6]:

- **Guaranteed accuracy up to 65 A with direct input**

The current accuracy is guaranteed up to 65 A with a direct input. Over the 65 A, the power meter can also measure the high current but with the use of optional current sensors. Direct-input power meters typically exhibit degraded accuracy when inputting high currents due to shunt resistor self-heating.



Figure 3.4: current measure limits.

- **High accuracy**

The *PW3337* accuracy is 0,1%. For complete details, please refer to the specifications in the website.

- **Wide frequency band**

The *PW3337* can cover not only at inverters fundamental frequency band, but also the carrier frequency band, thanks to a wide-band capability extending from DC and 0.1 Hz to 100 kHz.

- **Integrating fluctuating power values**

Thanks to its broad dynamic range, the *PW3337* can perform integrated power measurement with guaranteed accuracy using a single range, even if the power fluctuates dramatically during integration. Measurements can accommodate waveform peaks of up to 600% of the range rating.



## 3.2 Remote operation

The Power meter *PW337* can be connected to a PC through the LAN connection, to control the instrument remotely. The control occurs thanks to specific instrument's commands, that the user invites remotely from his computer. All the available commands are written in the *Communication manual*[7] of the measurement device. The command can be classified into two different types:

- **Command Messages**

Example: Instruction to set 300 V the voltage range (ch1)

`:VOLTAGE1:RANGE 300`

- **Query Messages**

Example: Request for the current measurement range

`:VOLTAGE1:RANGE?`

The first type is used to control the instrument, such as to change settings or reset. The second one are requests for responses relating to results of operation or measurement, or the state of instrument settings[6]. All these types of instruction are implemented in the Python code to create new acquisition features described in Chapter 4.

The developed Python code used the LAN connection functionality of the PW337 to send and receive data during standard test measurements. The scheme of principle of the measurement setup is shown in the Figure 3.5.

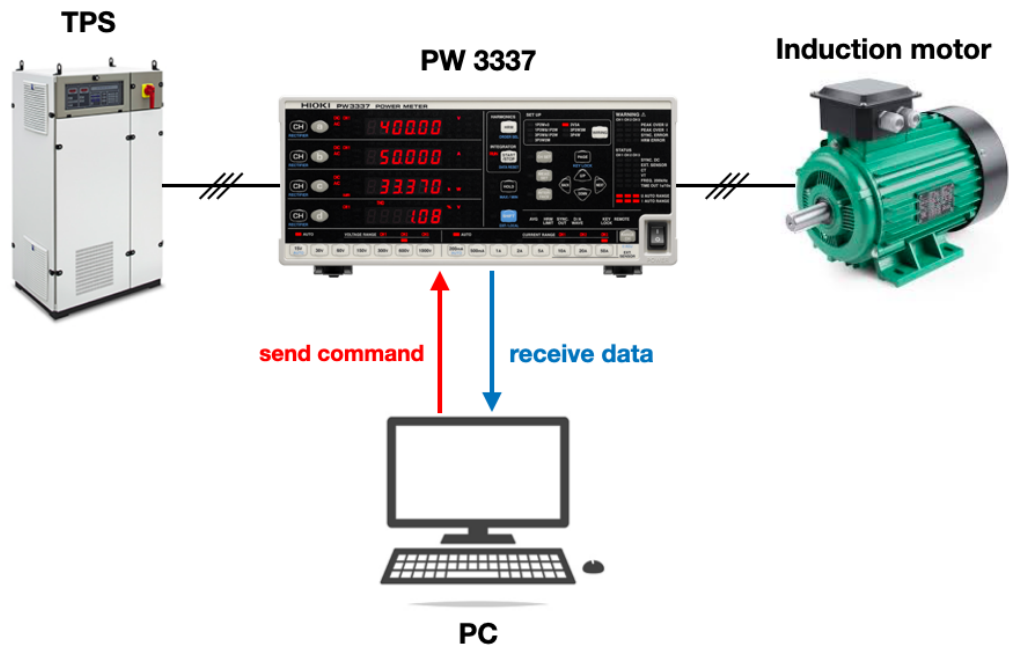


Figure 3.5: Scheme of principle of the measurement setup.

# Chapter 4

## Acquisition and Execution Tool

This Chapter describes the obtained graphical users interface (GUI) for the acquisition program. The GUI collects all the information and data from a user during the main Standard Tests. The standard tests considered in this work refer to "*IEEE Standard Test Procedure for Poly-phase Induction Motors and Generators*"[5] and are:

1. Preliminary test
2. No Load test
3. Locked rotor test

At the end of the acquisition, the program memorizes all the useful information in a dedicated text file. This file will be used in the post-elaboration Tool, to determine all the steady-state induction motor parameters.

### 4.1 The Python library Tkinter

To create a graphical users interface or GUI, python makes available a specif library called TKinter, which has lots of features and instruments to create a professional graphical interface. The simplicity of programming code is the main characteristic of this library. For example, with a few code rows, it is possible to create a window where put some library "gadgets", like buttons or labels:

```
1 from tkinter import *
2
3 # create the main window
4 window = Tk()
5 window.configure(bg='#cccccc')
6 window.title('Motor analysis')
7 window.geometry('1280x800')
8
9 fisrt_button = Button(window, text='text', height=2, width=10)
10 fisrt_button.grid(row=1, column=0)
11
12 window.mainloop()
```

These rows code generate the graphical users interface below.

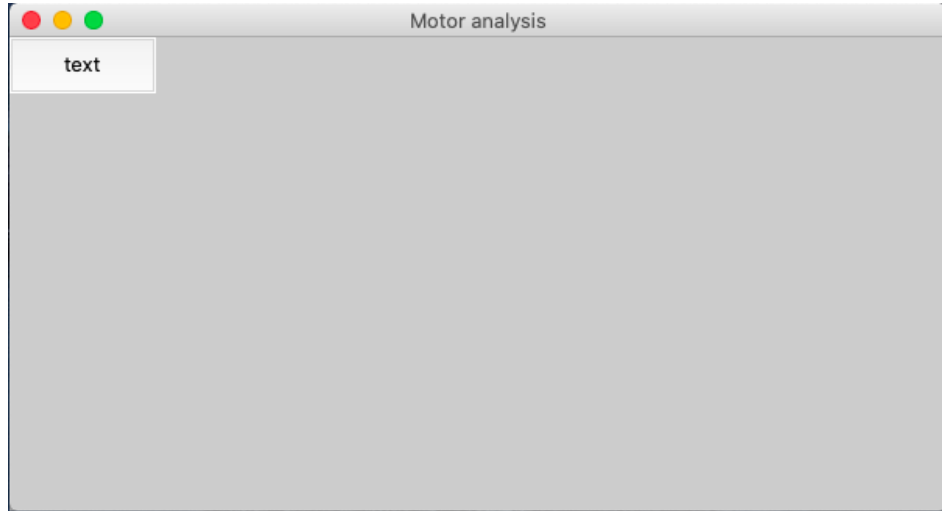


Figure 4.1: A simple GUI interface with one button.

Inside this chapter will not report the code to create the GUI, because will give more importance to the function definite inside this work. All the features and the main gadgets used to create the acquisition GUI are available on the Python website[8]. Moreover all the code developed in this thesis work is presented in the Appendix.

## 4.2 Acquisition GUI

The final GUI created at the end of this master thesis work is shown in the following Figure 4.2.



Figure 4.2: Acquisition GUI.

The GUI is divided into several sections to give a procedure order at the operation

steps. To increase the helping, there are also consecutive red numbers to give the execution order of the filling commands.

### 4.2.1 LAN Connection

The first operation is collocated in the section called "*Connection with LAN*". Here a user can connect the power meter with the PC through a LAN connection (obviously after the cable connection between the computer and the instrument). In the labels are present the default IP and port number of the instrument used in the laboratory, in this way the simple press on the "Connect" button creates the connection between the two devices, and the power meter pass in the remote control setup (a red led should light up).

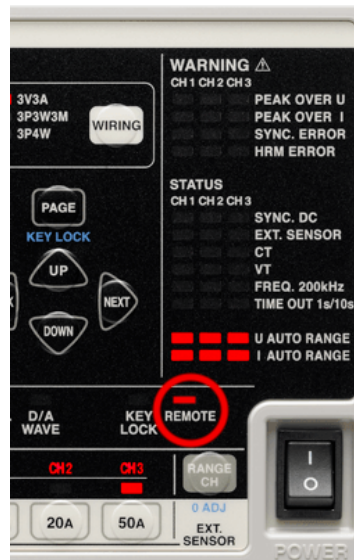


Figure 4.3: The red lamp on the front panel of the instrument displays after the successful remote connection to the instrument.

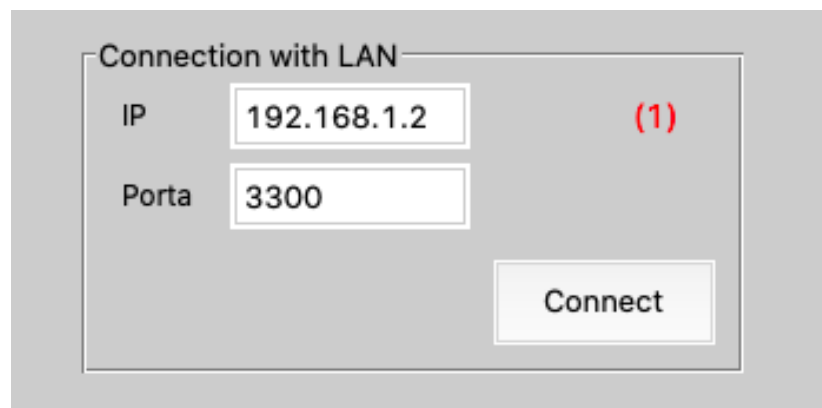


Figure 4.4: LAN connection section.

At the coding level, the connection is allowed thanks to the "lan" python program, which is made available by Hioki company. In this way for the connection with

the instrument is only necessary to recall the provided program in the library of the acquisition code.

```
1 from lan import Lan
```

The Connect button is related to a function defined in the script called "Connection1", which use some features of the Lan program to create the Lan connection.

```
1 def connection1():
2     B_conn['state'] = 'disabled'
3     B_openfile['state'] = 'normal'
4
5     # Instantiation of the LAN communication class
6     global lan
7     lan = Lan(Timeout_default)
8
9     # Connect
10    IP = box_con.get()
11    port = int(box_port.get())
12    #print("IP?")
13    #IP = input()
14    #print("Port?")
15    #port = int(input())
16    if not lan.open(IP, port):
17        return
```

### 4.2.2 File name

Following the number order, the next section is the "File name". This section is important because, in addition to the insert of the file name, the choice of the "Prove type" option determines the type of Standard Test.

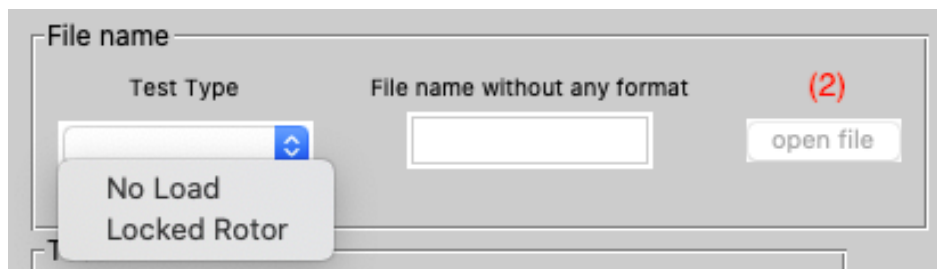


Figure 4.5: File name section.

Every options is correlated at one different text file extension:

- NO load -> .VTO
- Locked -> .CTO

When the "open file" button is selected, the python code opens a file in "write" mode with the name and the extension chosen. This is allowed thanks to a created function call "open file" shown below:

```
1 def open_file():
2     B_openfile['state'] = 'disabled'
```

```
3     save_button['state'] = 'normal'
4
5     filename = casella3.get()
6     if prove_type.get() == 'noload':
7         ext = '.VTO'
8     if prove_type.get() == 'locked':
9         ext = '.CTO'
10    if prove_type.get() == '':
11        messagebox.showinfo('Attention', 'Have to select the test type')
12    )
13    filename1 = filename + ext
```

If no option is selected, the program will show an error message on the screen.

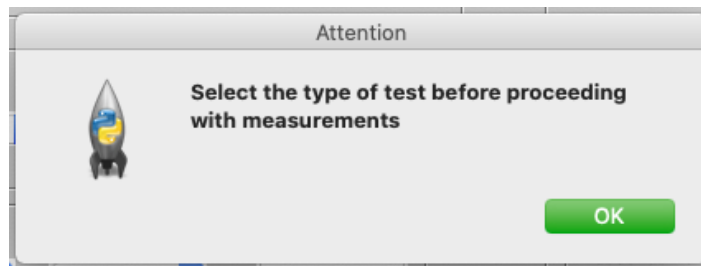


Figure 4.6: Error message for the no test type selection.

### 4.2.3 Motor data

In this section, the user has to fill the labels with the information from the electrical machine data.

Figure 4.7: The motor data section.

After the filling, the button "Save" memorize all the information in the previous text file, opened in the last section. To add the data, the file has to be used in the write method, identified in the python code with the "w" letter.

```

1 def save1():
2
3     B_salve['state'] = 'normal'
4     save_button['state'] = 'disabled'
5     global targa_data
6     global targa
7     targa_data = StringVar()
8     targa_data = box1.get()
9     targa_data = targa_data + ',' + box2.get()
10    targa_data = targa_data + ',' + box3.get()

```

```

11 targa_data = targa_data + ',' + box4.get()
12 targa_data = targa_data + ',' + box5.get()
13 targa_data = targa_data + ',' + box6.get()
14 targa_data = targa_data + ',' + box7.get()
15 targa_data = targa_data + ',' + box8.get()
16 targa_data = targa_data + ',' + box9.get()
17 targa = []
18 for w in targa_data.split(','):
19     targa.append(w)
20 print(targa)
21 file = open(open_file(), 'w')
22 file.write(targa[0] + ',' + targa[1] + '\n' + targa[5] + ',' +
23           targa[2] + ',' + targa[3] + ',' + targa[4] + ',' +
24           targa[8] + ',' + targa[7] + ',' + targa[6])

```

#### 4.2.4 Test settings

In this section, it is possible to specify some characteristics of the prove on the motor, as the supply type, the connection type, and the supply frequency. Moreover in this section is possible to add some notes, or simply the date of the tests.

The screenshot shows a window titled "Test". Inside, there are three dropdown menus: "Supply", "Winding connection", and "Frequency (4)". Below these is a text area labeled "Date and notes" and a "Save" button.

Figure 4.8: Test section.

The option menu under the "supply" label, shown three different choices:

1. SIN, that indicate a pure sinusoidal supply
2. PWM, that indicate a supply under inverter
3. OQ, that indicate a square wave supply

The possible option below the "Connection" voice are:

1. Star connection -> Y
2. Delta connection -> D

All the information in this section is added in the text file in the append method, identified in the python code with the "a" letter. In this way, the new rows will be written in a new row after the previous ones, written in the previous section, and the file will not be overwritten.



```

1 def put_prove():
2
3     B_salve['state'] = 'disabled'
4     B_save2['state'] = 'normal'
5     supp = supply.get()
6     conn = connection.get()
7     freq = casella43.get()
8     dataenote = casella4.get()
9
10    file = open(open_file(), 'a')
11    name1 = open_file()
12    file.write('\n'+supp+',','+freq+',','+conn+',',
13              +name1[-3:len(name1)]+'\n'+dataenote)

```

#### 4.2.5 Reference value of the winding resistance at the reference temperature

From this section begin the acquisition from the instruments. In this program, the section has measured the value of the winding resistance at the reference temperature. To measure the reference temperature is used a digital thermometer was placed close to the motor stator windings. Then the windings are supplied by a DC power source and the measure of voltage and current are read by a voltmeter and amperometer. This measurement is carried out on a single pair of stator windings since in an induction machine all the stator resistances are normally the same. The electrical scheme is shown in Figure 4.9

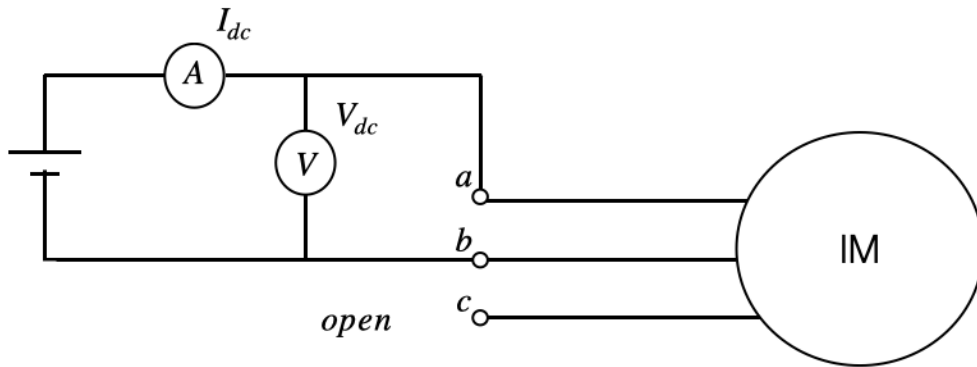


Figure 4.9: DC resistance measure.

**Winding resistance at the reference temperature**

|                                     |                      |                      |     |
|-------------------------------------|----------------------|----------------------|-----|
| DC voltage (V)                      | DC current (A)       | Temperature (°C)     | (5) |
| <input type="text"/>                | <input type="text"/> | <input type="text"/> |     |
| <input type="button" value="Save"/> |                      |                      |     |

Figure 4.10: winding resistance at the reference temperature section.

Like the previous sections, the Save button is defined by a function that allows memorizing all the information on the text file. Also, in this case, the file is open in the "append" writing mode, to not overwrite the document.

#### 4.2.6 Winding resistance at the beginning of the test

In this section, the user reports the voltage and the current measurement in DC supply to measure the stator windings resistance, before starting the AC measure acquisition. This measurement is necessary as the stator resistance at the start of the Standard test may not be the same as the reference resistance determined in the previous section.

Figure 4.11: Winding resistance at the end of the test.

#### 4.2.7 Power meter settings

Before proceeding with the AC measurement of the motor, it's fundamental to verify the instrument settings. Through the section called "setting of the wattmeter" is possible to set different measurement fields of the instrument:

Figure 4.12: Power meter setting section.

1. Wiring
2. Coupling
3. Synchronization source
4. Voltage and current range

## 5. Zero adjustment

All the settings are described in detail in the following paragraphs.

### 1. Wiring

With the Wiring option, it's possible to change the wiring configuration for the measurements. In the Acquisition tool is possible to set only one type of wiring: Aron connection

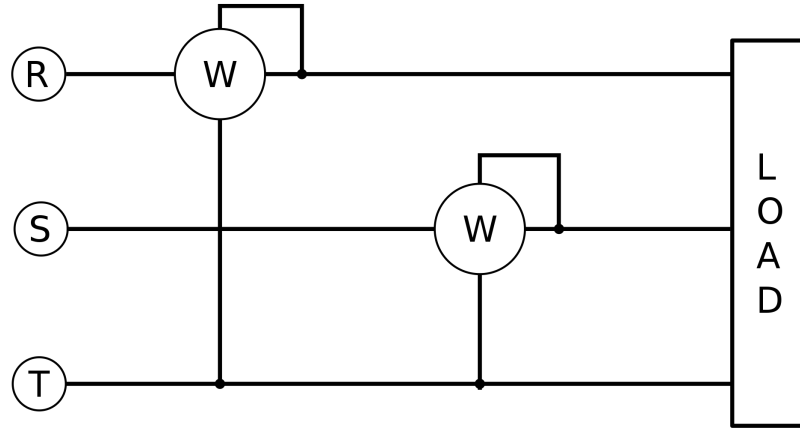


Figure 4.13: Aron connection.

With this configuration is possible to measure the power of a three-phase system with only two-phase power measurements. Connecting for example the amperometric on the 1-2 phases and the voltmetric on the 2-3 phase, the sum of the power from the two wattmeters is the three-phase active power of the system. The connection can be done in any phase, the wattmeters can be connected with the amperometric on any two-phase, while the voltmetric has to be connected with one side on the same amperometric's phase and the other one on a no connected phase. In this way, the possible configurations are three.

### 2. Coupling

The Coupling option give the possibility to select the type of rectifier. The Acquisition tool allows to choose two different options:

- (a) AC
- (b) AC+DC

The AC rectifier calculate the values given by the equation 4.1 as RMS values for the AC component, only for voltage and current[6].

$$\sqrt{(AC + DCvalue)^2 - (DCvalue)^2} \quad (4.1)$$

The "AC+DC" rectifier option displays true RMS values for all frequency bands that can be measured by the instrument for DC only, AC only, and mixed DC and AC voltage and current;

### 3. Synchronization source

enables to choose the type of synchronization source used to determine the cycle that will be applied as the basis for calculations (the voltage and the current are selected respectively for the sinusoidal and PWM supply);

### 4. Voltage and current range

At the bottom of the section there are the possibility to change the voltage and the current range. The default options is the "AUTO" configuration. When the measurement range is set to auto-range operation, the output rate for analog and waveform output will vary with the range. When measuring lines for which measured values fluctuate excessively, exercise care so as not to mistake range conversions. During the auto-range operation, the range is switched as described below:

- Range increased
  - (a) the measured value exceeds 130% of the range
  - (b) the "Pick Over" lamp light up
- Range decreased
  - (a) the measured value is less than 15% of the range (the range will not be decreased when the value would exceed the peak value for the next lower range)

### 5. Zero adjustment

Zero-adjustment consists of performing offset adjustment for the voltage and current internal circuit and degaussing (DEMAG) its internal input current unit. Zero-adjustment or offset adjustment is performed for voltage and current measured values after instrument has warmed up for approximately 30 minutes in order to ensure that its measurements accuracy specifications are satisfied.

The zero-adjustment should always be performed before starting measuring after the instrument has warmed up, though the PW337 automatically active it when the instrument is turned on.

Zero-adjustment adjusts offset in the range of:

- Voltage circuitry:  $\pm 10\%$  of the measurement range

- Current direct input circuitry:  $\pm 10\%$  of the measurement range
- External current sensor input circuitry:  $\pm 10\%$  of the measurement range
- Operating time: Approximately 40 seconds

To performed the Zero-adjustment in the code is simple, it is only necessary send the command to the power meter found in the *Communication Command Instruction Manual* [7]:

```
1 def Zeroadj(): # performing Zero-adjustment (about 40 sec)
2     lan.sendMsg(':DEMAg')
```

With the Set button the code memorize all the information and sent them to the instrument, thanks the setting function create in the code and show below.

```
1 def settings():
2
3
4     if wiring_range.get() == 'Aron':
5         lan.sendMsg('WIR TYPE3')
6
7     if wiring_range.get() == '3 Wattmetri':
8         lan.sendMsg('WIR TYPE5')
9
10
11
12     if coupling_range.get() == 'AC+DC':
13         lan.sendMsg(':DISP U0,I0,P0')
14
15     if coupling_range.get() == 'AC':
16         lan.sendMsg(':DISP UAC0,IAC0,PAC0')
17
18
19
20     if voltage_range.get() != 'AUTO':
21         comm = voltage_range.get()
22         lan.sendMsg(':VOLT:RANG '+comm[0:-1])
23     if voltage_range.get() == 'AUTO':
24         lan.sendMsg(':VOLT:AUTO ON')
25
26     if current_range.get() == '200mA':
27         lan.sendMsg(':CURR:RANG 0.2')
28     if current_range.get() == '500mA':
29         lan.sendMsg(':CURR:RANG 0.5')
30     if current_range.get() == '1A':
31         lan.sendMsg(':CURR:RANG 1.0')
32     if current_range.get() == '2A':
33         lan.sendMsg(':CURR:RANG 2.0')
34     if current_range.get() == '5A':
35         lan.sendMsg(':CURR:RANG 5.0')
36     if current_range.get() == '10A':
37         lan.sendMsg(':CURR:RANG 10.0')
38     if current_range.get() == '20A':
39         lan.sendMsg(':CURR:RANG 20.0')
40     if current_range.get() == '50A':
41         lan.sendMsg(':CURR:RANG 50.0')
42
```

```

43     if current_range.get() == 'AUTO':
44         lan.sendMsg(':CURR:AUTO ON')
45
46     else:
47         pass

```

Take the data from the label thanks to the `.get()` function define for the label python gadget and compare it with some possible cases. At any cases correspond a different command to be sent to the power meter, through a command message.

#### 4.2.8 Acquisition section

This is the main section, where the measurements are taken from the AC motor. This is the first section where the power meter communicate with the computer, sending information through the Lan connection.

Figure 4.14: Acquisition section.

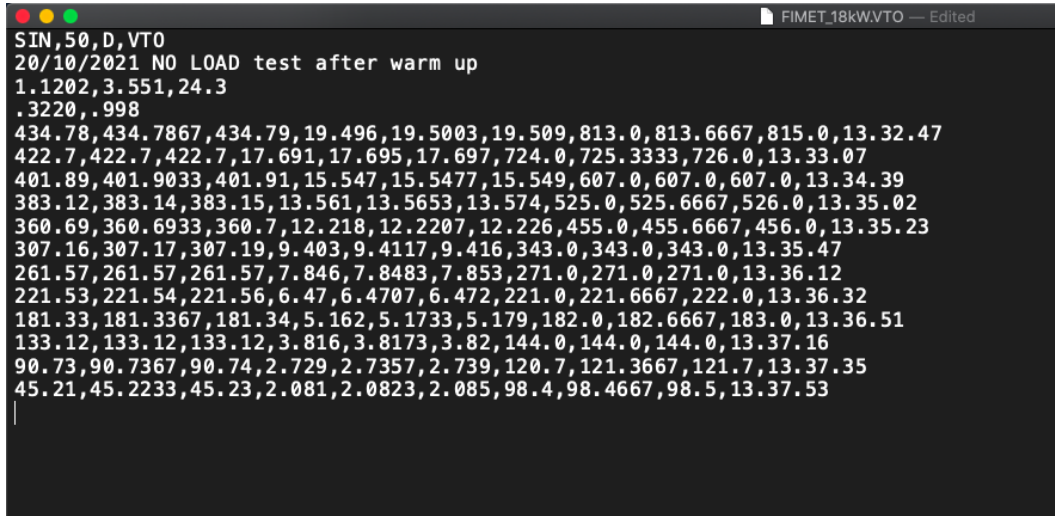
At the beginning it necessary select the number of measurements that the code has to effectuate for each voltage value, chosen for the test. The allow options are 3,5,10,15,20.

When the "Start measurements" button is pushed, the program update the text file created with the voltage, current and power measurements. Every row of the file is written in order to have:

- 3 voltage measurements. The first is the smallest, the second is the average between the all measures and the third one is the max value measured;
- 3 current measurements;
- 3 power measurements;

- the hours, minutes and second at the test moment.

An example text file obtained at the end of the Acquisition tool is shown in the Figure 4.15.



```

SIN,50,D,VTO
20/10/2021 NO LOAD test after warm up
1.1202,3.551,24.3
.3220,.998
434.78,434.7867,434.79,19.496,19.5003,19.509,813.0,813.6667,815.0,13.32.47
422.7,422.7,422.7,17.691,17.695,17.697,724.0,725.3333,726.0,13.33.07
401.89,401.9033,401.91,15.547,15.5477,15.549,607.0,607.0,607.0,13.34.39
383.12,383.14,383.15,13.561,13.5653,13.574,525.0,525.6667,526.0,13.35.02
360.69,360.6933,360.7,12.218,12.2207,12.226,455.0,455.6667,456.0,13.35.23
307.16,307.17,307.19,9.403,9.4117,9.416,343.0,343.0,343.0,13.35.47
261.57,261.57,261.57,7.846,7.8483,7.853,271.0,271.0,271.0,13.36.12
221.53,221.54,221.56,6.47,6.4707,6.472,221.0,221.6667,222.0,13.36.32
181.33,181.3367,181.34,5.162,5.1733,5.179,182.0,182.6667,183.0,13.36.51
133.12,133.12,133.12,3.816,3.8173,3.82,144.0,144.0,144.0,13.37.16
90.73,90.7367,90.74,2.729,2.7357,2.739,120.7,121.3667,121.7,13.37.35
45.21,45.2233,45.23,2.081,2.0823,2.085,98.4,98.4667,98.5,13.37.53

```

Figure 4.15: Example text file at the end of the Acquisition tool.

```

1 def acquisition():
2     print(measure.get())
3     N = int(measure.get())
4     max_V = 0
5     min_V = 10000000000000
6     sum_V = 0
7     max_I = 0
8     min_I = 10000000000000
9     sum_I = 0
10    max_P = 0
11    min_P = 10000000000000
12    sum_P = 0
13    v = []
14    i1 = []
15    p = []
16
17    if wiring_range.get() == 'Aron':
18
19        if coupling_range.get() == 'AC+DC':
20
21            for j in range(0, N):
22                box_meas.delete(0, 3)
23                v.append(only_number(lan.SendQueryMsg('MEAS? U0',
24                Timeout_default)))
25                i1.append(only_number(lan.SendQueryMsg('MEAS? IO',
26                Timeout_default)))
27                p.append(only_number(lan.SendQueryMsg('MEAS? PO',
28                Timeout_default)))
29                print(v[j])
30                box_meas.insert(END, j + 1)
31                if v[j] > max_V:
32                    max_V = v[j]

```

```

30         if v[j] < min_V:
31             min_V = v[j]
32         sum_V = sum_V + v[j]
33         if i1[j] > max_I:
34             max_I = i1[j]
35         if i1[j] < min_I:
36             min_I = i1[j]
37         sum_I = sum_I + i1[j]
38         if p[j] > max_P:
39             max_P = p[j]
40         if p[j] < min_P:
41             min_P = p[j]
42         sum_P = sum_P + p[j]
43         sleep(1) #
44         med_V = round(sum_V / N, 4)
45         med_I = round(sum_I / N, 4)
46         med_P = round(sum_P / N, 4)
47         named_tuple = time.localtime() # get struct_time
48         time_string = time.strftime("%H.%M.%S", named_tuple)
49         file = open(open_file(), 'a')
50         file.write('\n' + str(min_V) + ',' + str(med_V) + ',' +
str(max_V) + ',' + str(min_I) + ',' + str(med_I) + ',' + str(max_I
) + ',' + str(min_P) + ',' + str(med_P) + ',' + str(max_P) + ',' +
time_string)
51
52         if coupling_range.get() == 'AC':
53
54             for j in range(0, N):
55                 box_meas.delete(0, 3)
56                 box_Vrms.delete(0, 40)
57                 box_Irms.delete(0, 40)
58                 box_Prms.delete(0, 40)
59                 v.append(only_number(lan.SendQueryMsg('MEAS? UACO',
Timeout_default)))
60                 i1.append(only_number(lan.SendQueryMsg('MEAS? IACO',
Timeout_default)))
61                 p.append(only_number1(lan.SendQueryMsg('MEAS? PACO',
Timeout_default)))
62                 box_Vrms.insert(END, v[j])
63                 box_Irms.insert(END, i1[j])
64                 box_Prms.insert(END, p[j])
65
66                 box_meas.insert(END, j + 1)
67                 if v[j] > max_V:
68                     max_V = v[j]
69                 if v[j] < min_V:
70                     min_V = v[j]
71                 sum_V = sum_V + v[j]
72                 if i1[j] > max_I:
73                     max_I = i1[j]
74                 if i1[j] < min_I:
75                     min_I = i1[j]
76                 sum_I = sum_I + i1[j]
77                 if p[j] > max_P:
78                     max_P = p[j]
79                 if p[j] < min_P:
80                     min_P = p[j]

```



```

81         sum_P = sum_P + p[j]
82         sleep(1) #
83         med_V = round(sum_V / N, 4)
84         med_I = round(sum_I / N, 4)
85         med_P = round(sum_P / N, 4)
86         named_tuple = time.localtime() # get struct_time
87         time_string = time.strftime("%H.%M.%S", named_tuple)
88         file = open(open_file(), 'a')
89         file.write('\n' + str(min_V) + ',' + str(max_V) + ',' + str(med_V) + ',' +
str(min_I) + ',' + str(max_I) + ',' + str(med_I) + ',' + str(min_P) + ',' + str(max_P) + ',' + str(med_P) + ',' +
time_string)

```

An important observation is the time delay between each measure, that in the code is related to sleep function, present in the row 43 and 81 in above python script. It is set one second. The "Save recording" button write all the data in the file and unblock the next program section.

#### 4.2.9 Resistance at the end of the test

In this last acquisition program section the user reports the voltage and the current with a DC supply at the end of the test and after 30 seconds from the latter. These two measurements are necessary to identify the windings resistance at the end of the test.

Figure 4.16: The resistance measurements after the test.

### 4.3 Multiple acquisition

It's important to make the focus on the multiple acquisitions of the Tool. The possibility to select multiple acquisitions is made to decrease the voltage and current fluctuations errors.

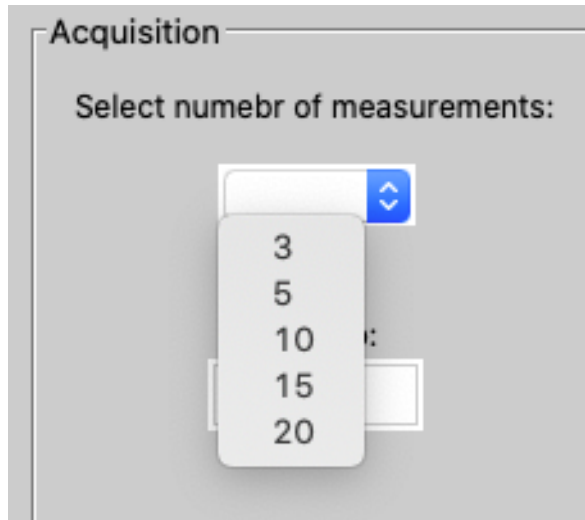


Figure 4.17: Multiple Acquisition settings.

The multiple acquisitions can individuate the presence of measurements in the dynamic state. If the first measurements are different from the last ones, they may be measured in not the steady-state condition. The multiple acquisitions individuate the error by the comparison and also can minimize it by the average calculation. For these reasons, all the elaboration process are made with the average values of the text file.

Moreover, it's possible to set the Wait Time enables to choose the time between consecutive measurement during the multiple acquisitions.

# Post-elaboration Tool

This chapter explains the calculation algorithms to obtain the steady-state parameters of the induction motors from the Standard tests and describes the Elaboration GUI obtained.

### 5.1 Determination of the parameters of the steady-state equivalent circuit

The Elaboration methodology follows the Standard tests on the induction machines described in Chapter 2. For this chapter is useful to remember the single-phase equivalent circuit considered the one shown in the Figure 5.1:

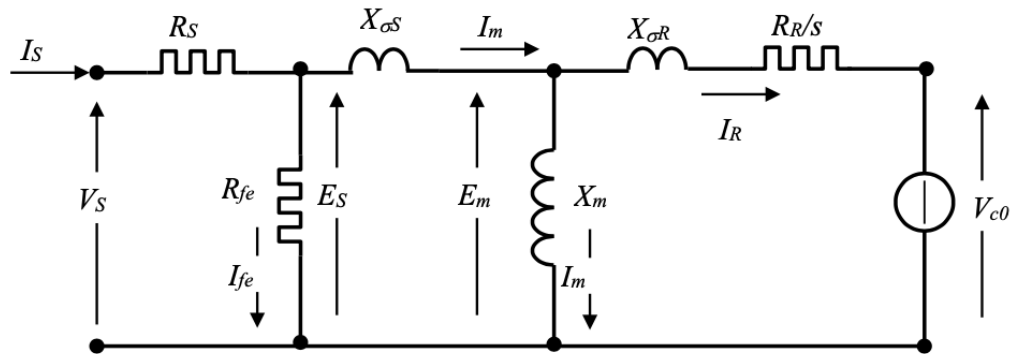


Figure 5.1: Electrical single-phase equivalent steady-state circuit.

where:

- $V_s$  stator phase voltage;
- $I_s$  stator phase current;
- $R_s$  stator winding resistance;
- $R_{fe}$  equivalent iron loss resistance;
- $X_m$  magnetising reactance;
- $X_{\sigma s}$  leakage stator reactance;

- $X_{\sigma r}$  leakage rotor reactance;
- $\frac{R_r}{s}$  rotor resistance;
- $V_{e0}$  e.m.f for the closed stator slots.

### 5.1.1 Calculation of the stator winding resistance

Referring to the "*IEEE Standard Test Procedure for Polyphase Induction Motors and Generators*"[5] nomenclature, the very first measurement concerns the determination of the stator resistance is the preliminary test which allows determining the stator resistance of the windings measurements. All the calculations in this section depend on the stator winding connection:

#### i WYE Connection

#### ii Delta Connection

Consider the different cases:

#### i WYE Connection

If the stator windings are in WYE Connection, the electric circuit considered for the elaboration of the DC test is shown in Figure 5.2.

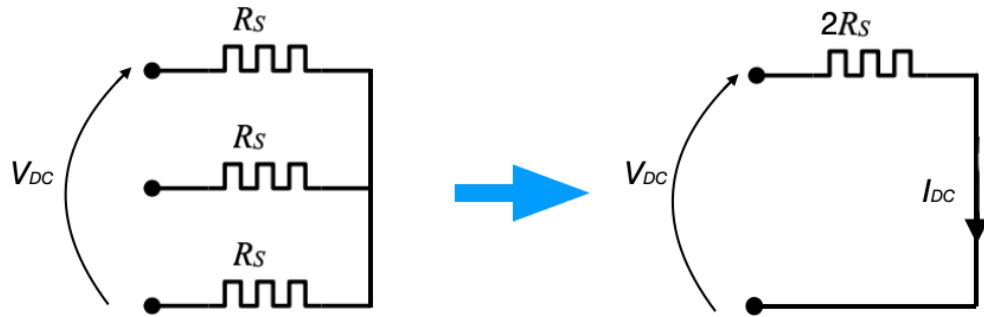


Figure 5.2: equivalent circuit of the wye-connected stator winding during the DC test.

To calculate the stator resistance  $R_s$ , two phase of Stator windings are supplied with a DC voltage source  $V_{DC}$ . Since the two resistance  $R_s$  are in series, with this type of connection the total resistance is the sum of the two.

Therefore, the stator winding resistance  $R_s$  is:

$$R_s = \frac{1}{2} \frac{V_{DC}}{I_{DC}} \quad (5.1)$$

## ii Delta Connection

If the stator windings configuration is the Delta Connection, the electric circuit considered for the elaboration of the DC test is shown in Figure 5.3. From the Delta connection can be obtained the first equivalent circuit.

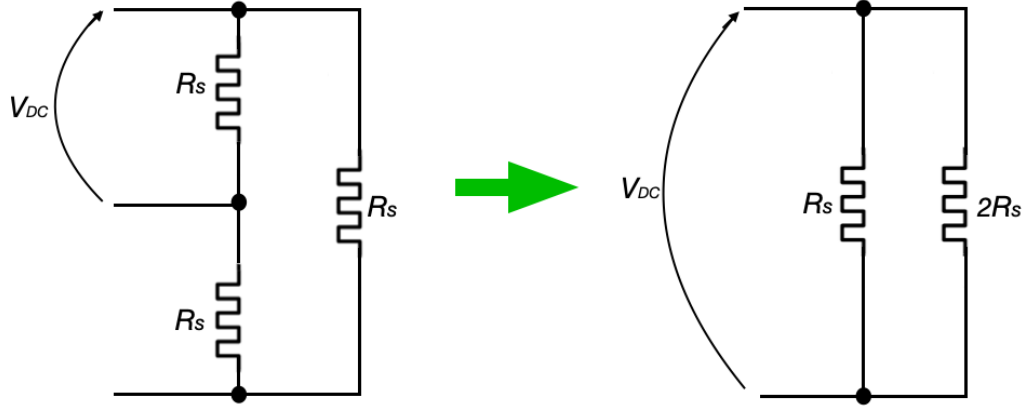


Figure 5.3: Equivalent circuit of the delta-connected stator winding during the DC test.

The final equivalent circuit is given by the Figure 5.4.

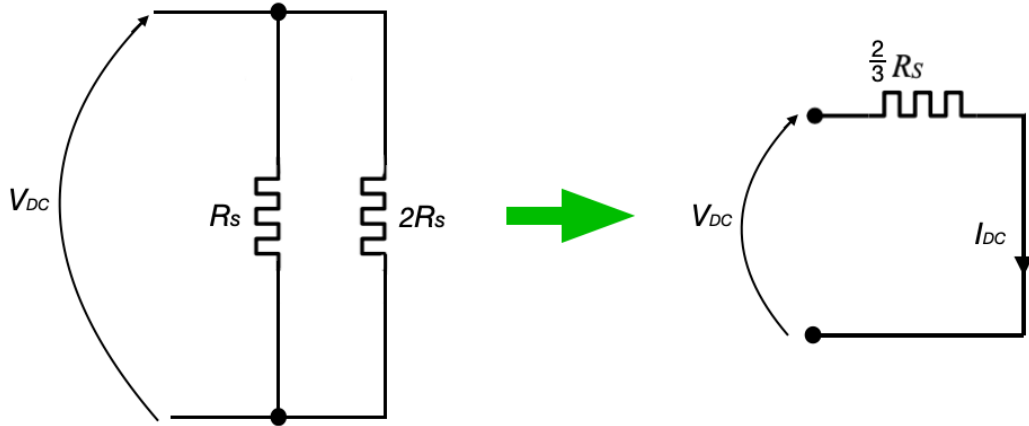


Figure 5.4: Stator windings resistance in case of delta connection.

$$R_{//} = \frac{2R_s^2}{R_s + 2R_s} \quad (5.2)$$

$$R_{//} = \frac{2}{3}R_s \quad (5.3)$$

Considering the final equivalent circuit the  $R_s$  is calculated as shown below

$$R_s = \frac{3}{2} * \frac{V_{DC}}{I_{DC}} \quad (5.4)$$

To simplify the equations, in the Python code developed, it is created a variable to making into account the type connection. The information about the connection is presented on a variable called *con*:

- if Connection is **Y**  $\rightarrow$  the variable *con* will be equal to  $\sqrt{3}$
- if Connection is **D**  $\rightarrow$  the variable *con* will be equal to 1

In this way the two equation became:

$$R_s = \frac{3}{2} * \frac{V_{DC}}{I_{DC}} / con^2 \quad (5.5)$$

During the elaboration algorithm the equation above is used three times to calculate the stator resistance in three different cases:

1. **At the reference temperature**
  2. **At the beginning of the test**
  3. **At the end of the test**
1. **At the reference temperature**

The winding resistance at the reference temperature is calculated in the equation 5.6, where the  $V_{DC_0}$  and  $I_{DC_0}$  are respectively the DC voltage supply on the stator windings and current flow in the circuit.

$$R_{amb} = \frac{3}{2} * \frac{V_{DC_0}}{I_{DC_0}} / con^2 \quad (5.6)$$

This type of test have to be performed with the machine at rest, in order to have the windings at the room temperature, as explain in the Chapter 2.

2. **At the beginning of the test**

The next calculation is the value of the winding resistance at the beginning of the No load test. The resistance is determined with the same equation but with different voltage and current measurements.

$$R_{start} = \frac{3}{2} * \frac{V_{DC_1}}{I_{DC_1}} / con^2 \quad (5.7)$$

In this case,  $V_{DC_1}$  and  $I_{DC_1}$  are respectively the DC voltage and the current measurements before stating of the AC measurements for the No load test and Locked Rotor test.

3. **At the end of the test**

Last resistance calculations is about the resistance at the end of the Standard Test. It is calculated starting from two resistance measurements

- a The resistance at the end of the AC measurements
- b The resistance calculate after 30 seconds from the last DC measurements (a)

The first winding resistance is calculated in the equation 5.8:

$$R_a = \frac{3}{2} * \frac{V_{DC_a}}{I_{DC_a}} / \cos^2 \quad (5.8)$$

In this case the  $V_{DC_a}$  and  $I_{DC_a}$  are the DC voltage and DC current measured at the end of the Standard Test executed, No Load or Locked Rotor.

The second resistance calculation at the end of the test is the one measured after 30 seconds the previous DC measurements, shown in the equation 5.9.

$$R_b = \frac{3}{2} * \frac{V_{DC_b}}{I_{DC_b}} / \cos^2 \quad (5.9)$$

In this case the  $V_{DC_b}$  and  $I_{DC_b}$  are the DC voltage and DC current measured after 30 seconds the previous DC measurements. at the end of the Standard Test executed, No Load or Locked Rotor.

The stator resistances determined allow to calculate the stator windings resistance during the AC measurement of the Standard test.

The algorithm starts from the stator windings temperature consideration. The temperature  $\theta$  is functions of the time  $t$ :

$$\theta = \theta(t) \quad (5.10)$$

The function is proportional to an exponential, where the  $\tau$  is the thermal time constant of the system:

$$\theta(t) \propto (1 - e^{-\frac{t}{\tau}}) \quad (5.11)$$

Each point of the graph in Figure 5.5 represents a different temperature state of the stator winding. The nomenclature of subscripts agrees with that of resistances:

- i the state  $(t_{start}, \theta_{start})$  represents the temperature of the stator windings  $(\theta_{start})$  at the starting of the Standard tests measurements  $(t_{start})$ ;
- ii the state  $(t_{end}, \theta_{end})$  represents the temperature of the stator windings  $(\theta_{end})$  at the end of the Standard tests measurements  $(t_{end})$ ;
- iii the states  $(t_a, \theta_a)$  and  $(t_b, \theta_b)$  represent respectively the temperature of the stator windings  $(\theta_a)$  at the end of the Standard tests measurements  $(t_a)$  and the temperature of the stator windings  $(\theta_b)$  30 seconds after the last measurement  $(t_b)$ .

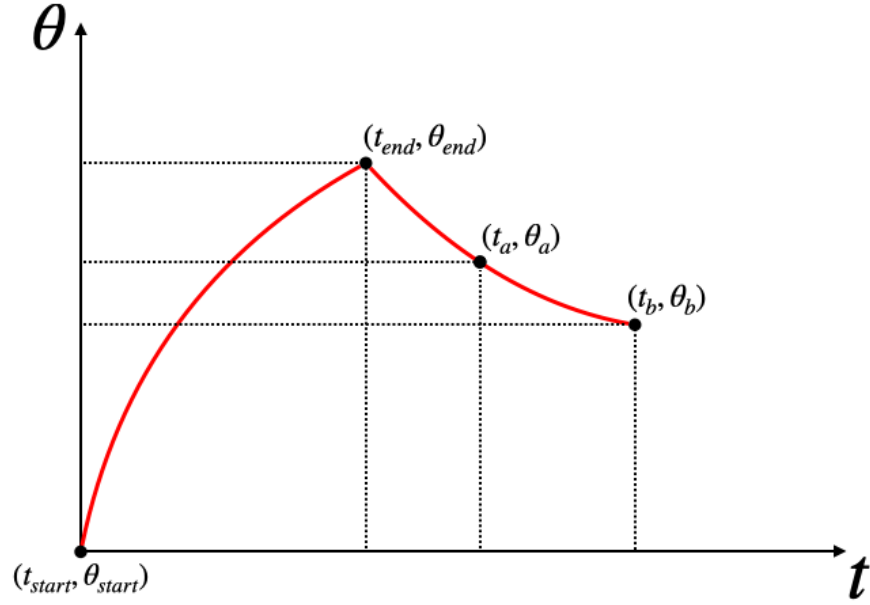


Figure 5.5: Variation of the stator winding temperature during the tests.

The electrical resistance is function of the temperature, for the transitivity propriety the resistance is function of the time:

$$R = R(\theta) \quad (5.12)$$

$$R = R[\theta(t)] \quad (5.13)$$

$$R = R(t) \quad (5.14)$$

The function of the stator resistance of the windings in the time has the same trend of the temperature, as shown the Figure 5.6.

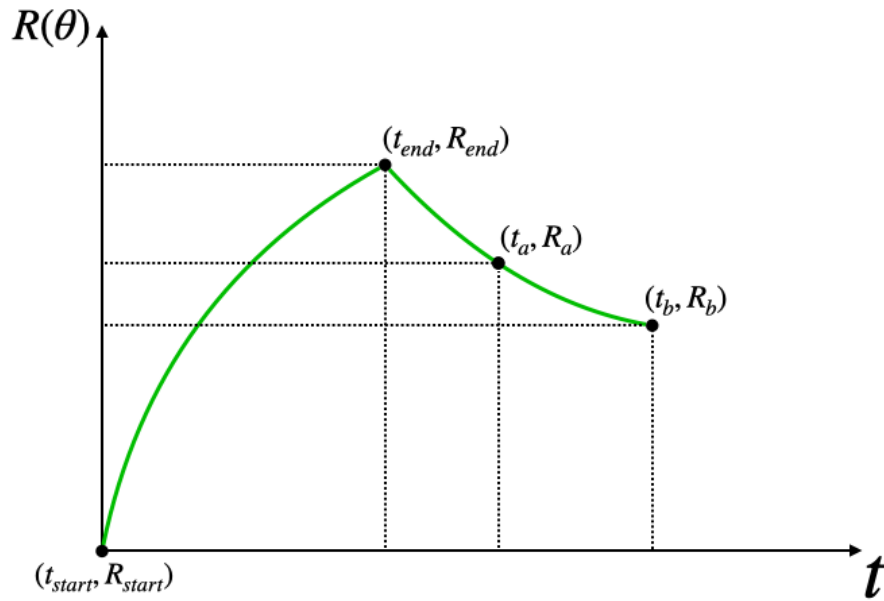


Figure 5.6: Variation of the stator winding resistance during the tests.



Since the  $\Delta T_{test} \ll \tau$  it is possible to consider a liner function instead of an exponential one for calculating the variation of the stator resistance during the tests.

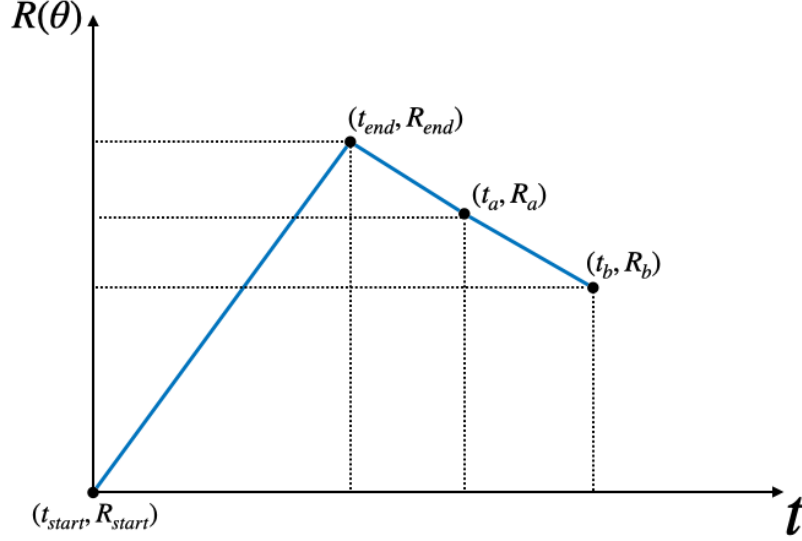


Figure 5.7: Linearization of the stator winding resistance during the tests.

The function of a straight line passing through two points  $a$  and  $b$  is given by the equation 5.15

$$R - R_b = \frac{t - t_b}{t_a - t_b} (R_a - R_b) \quad (5.15)$$

To determine the stator resistance at the end of the measurements, it's sufficient substitute  $R_{end}$  and  $t_{end}$  in the previous equation.

$$R_{end} - R_b = \frac{t_{end} - t_b}{t_a - t_b} (R_a - R_b) \quad (5.16)$$

The fraction in the equation 5.16 will be called  $RappTP$ .

$$R_{end} - R_b = RappTP (R_a - R_b) \quad (5.17)$$

From the last equation the  $R_{end}$  is given by the equation 5.18.

$$R_{end} = RappTP (R_a - R_b) + R_b \quad (5.18)$$

With reference to the Figure 5.6, it is possible to apply the previous algorithm for the points

$$R - R_{start} = \frac{R_{end} - R_{start}}{t_{end} - t_{start}} (t - t_{start}) \quad (5.19)$$

the fraction in the equation 5.19 will be called  $RappRT$

$$R - R_{start} = RappRT (t - t_{start}) \quad (5.20)$$

The equation above allows to calculate the stator windings resistance during the Standard Test measurements.

$$R_i = R_{start} + R_{app}RT(t_i - t_{start}) \quad (5.21)$$

In the equation 5.21, the  $R_i$  and the  $t_i$  are respectively the stator winding resistance and the time expressed in second, for the i-th measure of the Standard tests (No Load and Locked Rotor).

### 5.1.2 Elaboration of the No Load test results

This subsection describes how to obtain the No Load parameters of the induction motors, starting from the voltage, current and active power measurements obtained during the tests.

During the No Load test the equivalent circuit is shown in Figure 5.8. Since the machine rotates at the synchronous speed, the rotor current is practically zero and therefore it is possible to ignore the rotor parameters during the calculations.

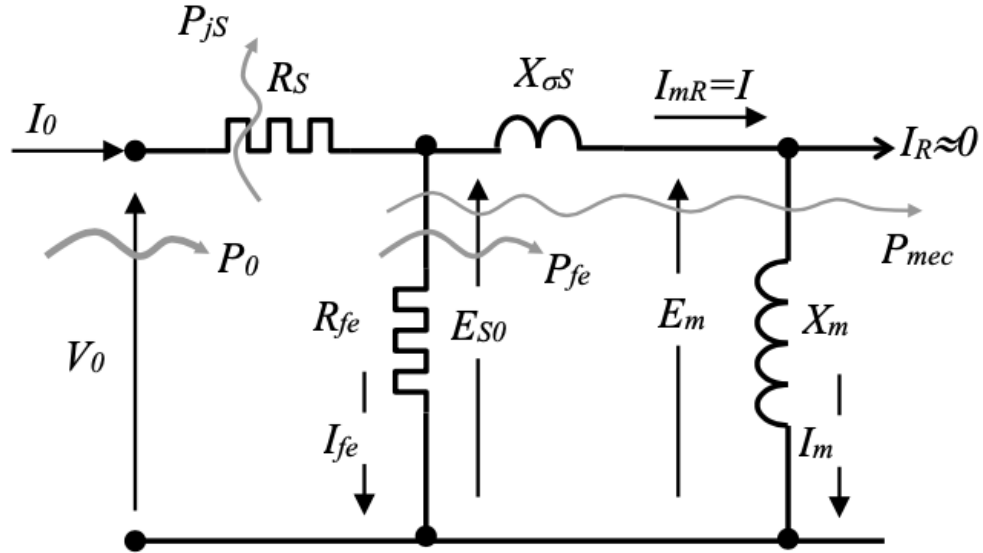


Figure 5.8: No Load single-phase equivalent circuit.

The elaboration Tool works with the phase voltage, so based on the type connection of the test, the voltage is described by the equation 5.22

$$V_0 = \frac{V_M}{\cos} \quad (5.22)$$

$$I_0 = \frac{I_M \cos}{\sqrt{3}} \quad (5.23)$$

The  $V_M$  and  $I_M$  represent respectively the average of the voltages and currents measured during the No Load test, which is presented in the text file obtained at the end of the acquisition tool.

During the Standard Test is important to determine the Power Factor shown in the equation 5.24, where the  $P_0$  is the average power measured during the test.

$$\cos \varphi_0 = \frac{P_0}{3V_0I_0} \quad (5.24)$$

The e.m.f  $E_{s0}$  can be determined thanks to the vector diagram shown in the Figure 5.9

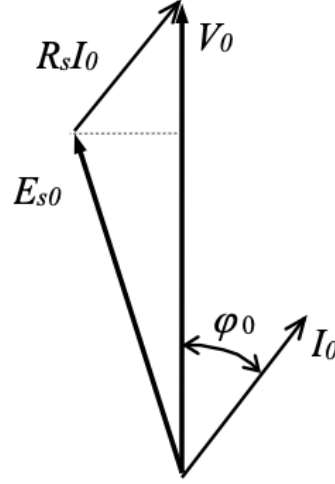


Figure 5.9: Vector Diagram.

$$E_{s0} = \sqrt{(V_0 - R_s I_0 \cos \varphi_0)^2 + (R_s I_0 \sin \varphi_0)^2} \quad (5.25)$$

$$E_{s0} = E_{s0} \cos \varphi_0 \quad (5.26)$$

During the No Load test is important to extrapolate all the power information. The power losses are divided into three different components:

1. Joule losses due to the stator resistance;
2. Iron losses;
3. Mechanical losses.

The relation between all these losses informations and the power injected into the system are described in the equation 5.27

$$P_0 = P_{mec+fe} + 3R_s I_0^2 \quad (5.27)$$

The power losses in the iron and for the mechanical effects are inserted in a single power term:

$$P_{mec+fe} = P_0 - 3R_s I_0^2 \quad (5.28)$$

The  $P_{mec+fe}$  trend is shown in the Figure 5.10, that underline the importance of the measurements at low voltage, which gives the possibility to find the intercept and obtain the mechanical losses information of the system

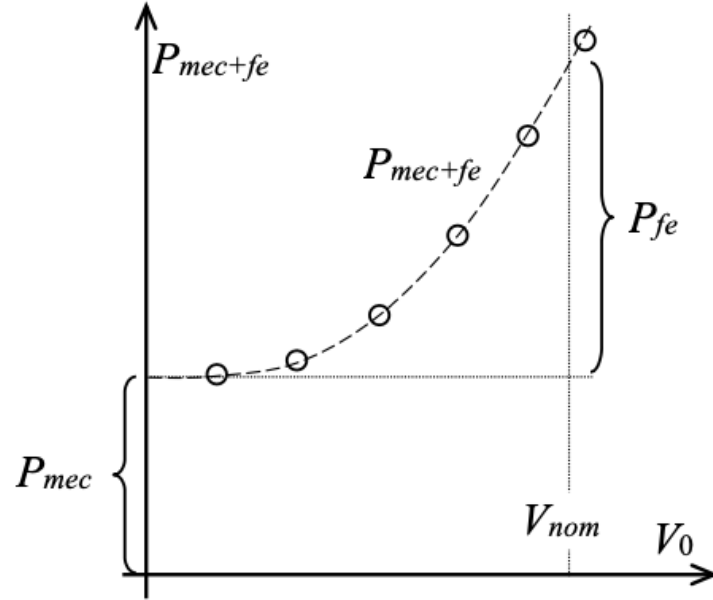


Figure 5.10: Mechanical and core losses with respect to the supply voltage.

All the previous equations are calculated for each measurement, so the results obtained are vectors.

```

1 #completamento della RESISTENZA DI STATORE nelle varie
2     if ColnNom == 'Y':
3         col = sqrt(3)
4     elif ColnNom == 'D':
5         col = 1
6     else:
7         messagebox.showwarning("warning", "Error with the
configuration parameter!")
8     IfaseNom = Inom * col / sqrt(3)
9     RfaseAmb = 1.5 * Vdc0 / Idc0 / pow(col, 2)
10    RfaseInPrv = 1.5 * Vdc1 / Idc1 / pow(col, 2)
11    rappFrq = FreqPrv / FreqNom
12    DeltaV = col * RfaseInPrv * IfaseNom
13    VPrvNom = DeltaV + rappFrq * (Vnom - DeltaV)
14
15    Rad3 = sqrt(3)
16    Rfase2 = 1.5 * float(mylist[-19]) / float(mylist[-16]) / pow(col
, 2)
17    Rfase3 = 1.5 * float(mylist[-9]) / float(mylist[-6]) / pow(col,
2)
18    RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])
19    #RappTP=2
20    print(RappTP)
21    RfaseFiPrv = RappTP * (Rfase2 - Rfase3) + Rfase3
22    RappRT = (RfaseFiPrv - RfaseInPrv) / (tp[-3] - tp[0])
23    print(RfaseFiPrv)

```

```

24     print(RappRT)
25
26     # completamento della terza tabella
27     box4e.insert(END, round(RFaseFiPrv, 7))
28
29     i=0
30     RF=[]
31     IFM=[]
32     PCM=[]
33     PNM=[]
34     CFM=[]
35     EM=[]
36
37     for i in range(Ndati):
38
39         RF.append(RfaseInPrv+(tp[i] - tp[0]) * RappRT)
40         VFM = VM[i] / col
41         IFM = Im[i] * col / Rad3
42         PCM.append(3 * RF[i] * pow(IFM, 2))
43         PNM.append(PTM[i] - PCM[i])
44         CFM.append(PTM[i] / Rad3 / VM[i] / Im[i])
45         SFM = (sqrt(1 - pow(CFM[i], 2)))
46         EFM = sqrt(pow((VFM - RF[i] * IFM * CFM[i]), 2) + pow((RF[i]
47         * IFM * SFM), 2))
         EM.append(EFM * col)

```

From all the vector data obtained in the previous part of the algorithm, the Elaboration Tool extrapolates three functions:

- a  $E.m.f(V)$  function by the interpolation from the  $V_0$  and  $E_{s0}$  calculated for each measurement;
- b  $I_0(E.m.f)$  function by the interpolation from the  $I_0$  and  $E_{s0}$  calculated for each measurement;
- c  $P_0(E.m.f)$  function by the interpolation from the  $P_0$  and  $E_{s0}$  calculated for each measurement.

All these functions are used to determine the No Load parameters, which are:

- $I_0$  No Load current;
- $P_{fe}$  Iron losses;
- $P_{mec}$  Mechanical losses;
- $R_{fe}$  Equivalent iron losses resistance;
- $X_s$  Stator reactance, that is equal to the sum of  $X_m + X_{\sigma s}$ .

From the first interpolation obtain the value of e.m.f at the rated voltage. In fact the  $coef f_1[1]$  is the coefficient of the polynomial interpolation.

$$E' = V_{start_{rated}} coef f_1[1] \quad (5.29)$$

To determine the  $V_{start_{rated}}$  is necessary first to calculate the frequency ratio  $f_{ratio}$  and the voltage variation  $\Delta V$ :

$$f_{ratio} = \frac{f_{start}}{f_{rated}} \quad (5.30)$$

$$\Delta V = R_{start} I_{ph_{rated}} con \quad (5.31)$$

The rated voltage at the starting of the measured is given by the equation 5.32

$$V_{start_{rated}} = \Delta V + f_{ratio}(V_{rated} - \Delta V) \quad (5.32)$$

$$I_0 = (I_0 + coef f_2[j] E'^j) \quad (5.33)$$

$$P_0 = (P_0 + coef f_3[j] E'^j) \quad (5.34)$$

The Iron losses are essentially the contribute of the  $P_{mec+fe}$  without the mechanical losses. The mechanical losses term is represented by the coefficient of zero grade of the third interpolation function.

$$P_{fe} = P_0 - coef f_3[0] \quad (5.35)$$

$$P_{mec} = coef f_3[0] \quad (5.36)$$

The total electric impedance is given by the equation 5.37.

$$Z = \sqrt{3} \frac{E'}{con^2 I_0} \quad (5.37)$$

From the previous equation and with the all data determined is possible to calculate the last two No Load parameters:

$$R_{fe} = 3 \frac{E'^2}{P_{fe} con^2} \quad (5.38)$$

$$X_s = \sqrt{Z^2 - R_{fe}^2} \quad (5.39)$$

From the No Load test is not possible to separate the dispersion inductance term from the magnetization one. This test is only able to determine the total value of the stator inductance  $X_s$ .

### 5.1.3 Elaboration of the Locked Rotor test results

This subsection describes how to determinate the Locked Rotor parameters of the induction motors, starting from the voltage, current and active power measurements obtained during the acquisition procedure.

The equivalent electric circuit used for the Locked Rotor test is shown in Figure 5.11, where the transversal parameters are neglected. Since the rotor speed is zero ( $\omega_r = 0$ ), the  $slip = 1$  and the rotor resistance is defined only with the  $R_r$ . The voltage  $V_{c0}$  is due to the closed stator slots in the induction motors considered.

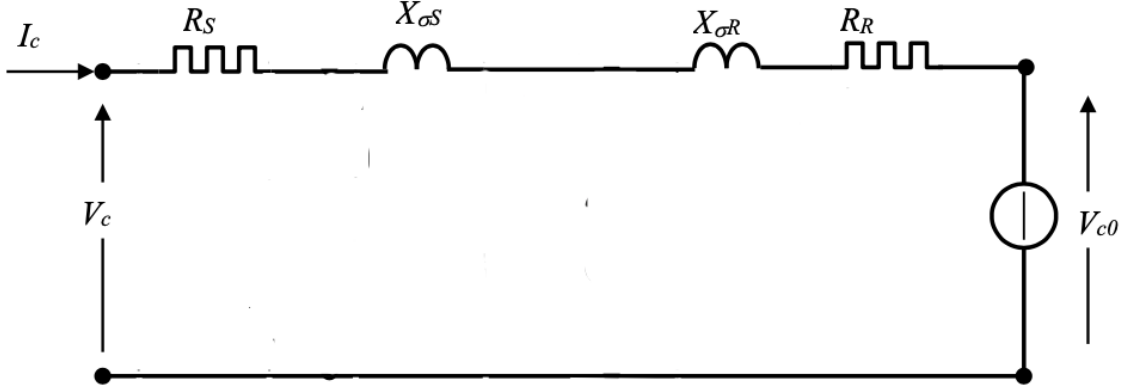


Figure 5.11: Equivalent circuit for the Locked Rotor test.

At the starting of the Tool, it's request the insert of the reference temperature as shown in Figure 5.12.

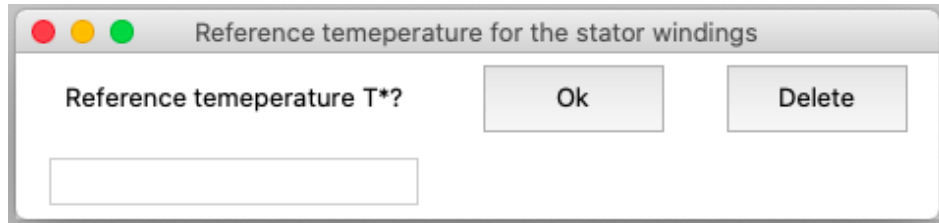


Figure 5.12: Reference temperature choice.

The reference temperature allows to calculate the stator winding resistance for that selected temperature, thanks to the calculation of  $R_{amb}$  executed as shown in the "Preliminary test" section. The measurement results are referred to the

$$R_{ref} = R_{amb} \frac{(T_{ref} + 235)}{(T_{amb} + 235)} \quad (5.40)$$

All the average voltage and current obtained from the Acquisition Tool are converted in the phase magnitude, thanks to the already known  $con$  variable, as shown in the equations 5.41 and 5.42.

$$V_{ccM} = \frac{V_M}{con} \quad (5.41)$$

$$I_{ccM} = I_M \frac{con}{\sqrt{3}} \quad (5.42)$$

From the phase voltage and current data are determined the impedance of the short circuit.

$$Z_{ccM} = \frac{V_{phM}}{I_{phM}} \quad (5.43)$$

$$R_{ccM} = \frac{P_M}{3I_{phM}^2} \quad (5.44)$$

$$X_{ccM} = \sqrt{Z_{ccM}^2 - R_{ccM}^2} \quad (5.45)$$

$$\cos \varphi_M(i) = \frac{R_{ccM}}{Z_{ccM}} \quad (5.46)$$

The previous values are re-calculate in reference temperature, thanks the use of a ratio call  $K_{ref}$  shown in the equation 5.47. The Locked Rotor parameters must be reported to the reference temperature  $\theta_{ref}$ .

$$K_{ref} = \frac{R_{ref}}{R_i} \quad (5.47)$$

$$P_{cM}[i] = K_{ref} P_M[i] \quad (5.48)$$

$$R_{cM} = K_{ref} R_{ccM} \quad (5.49)$$

$$Z_{cM} = \sqrt{R_{cM}^2 + X_{ccM}^2} \quad (5.50)$$

$$V_{phcM} = Z_{cM} I_{phM} \quad (5.51)$$

$$V_{cM}[i] = V_{phcM} \cos \varphi_{cM}[i] \quad (5.52)$$

$$\cos \varphi_{cM}[i] = \frac{R_{cM}}{Z_{cM}} \quad (5.53)$$

From all the vectors data obtained in the previous part of the algorithm, the Elaboration Tool extrapolates two functions:

- a  $V_{cc}(I_{cc})$  function by the interpolation from the  $V_{cM}$  and  $I_{phM}$  calculated for each measurement
- b  $P_{cc}(I_{cc})$  function by the interpolation from the  $P_{cM}$  and  $I_{phM}$  calculated for each measurement

All these interpolation functions are used to determine the Locked Rotor parameters, which are:

- $V_{lr}$  Locked rotor voltage;
- $P_{lr}$  Locked rotor losses;
- $Z_{lr}$  Locked rotor impedance;
- $\cos \varphi$  Locked rotor power factor;
- $R_s$  Stator resistance;
- $R_r$  Rotor resistance;
- $X_s$  Stator reactance;
- $X_r$  Rotor reactance.



The Locked Rotor parameters must be reported to the reference temperature  $\theta_{ref}$ .

$$V_c = V_c + coef f_1[i] I_c^i \quad (5.54)$$

$$P_c = P_c + coef f_2[i] I_c^i \quad (5.55)$$

$$\cos \varphi_c = \frac{P_c}{\sqrt{3} V_c I_c} \quad (5.56)$$

$$Z_{cc} = \frac{\sqrt{3} V_c}{I_c \cos^2} \quad (5.57)$$

$$Z_{aux} = \frac{\sqrt{3} (V_c - coef f_1[0])}{I_c \cos^2} \quad (5.58)$$

$$R_{cc} = Z_{cc} \cos \varphi_c \quad (5.59)$$

$$X_{cc} = \sqrt{Z_{cc}^2 - R_{cc}^2} \quad (5.60)$$

$$R_r = R_{cc} - R_s \quad (5.61)$$

$$X_s = \frac{1}{2} \sqrt{Z_{aux}^2 - R_{cc}^2} \quad (5.62)$$

$$X_r = X_{cc} - X_s \quad (5.63)$$

#### 5.1.4 Interpolation strategy

One of the main features of the Elaboration Tool is the interpolation section. The interpolation developed in the application is a polynomial one, which creates a polynomial function from the points measured. To this propose it is used an already existing function called *Curve Fitting* from the Python library *Scipy*.

Curve fitting is a type of optimization that finds an optimal set of parameters for a defined function that best fits a given set of observations. It requires defining a function form of the mapping function (also called the basis function or objective function), then searching for the parameters to the function that result in the minimum error.

```
1 coefficients, _ = curve_fit(function, x, y)
```

Error is calculated by using the observations from the domain and passing the inputs to the candidate objective function and calculating the output.

Once fit, we can use the mapping function to interpolate or extrapolate new points in the domain. It is common to run a sequence of input values through the mapping function to calculate a sequence of outputs, then create a line plot of the result to show how output varies with input and how well the line fits the observed points.

In the thesis work the mapping function is a polynomial one and it's defined as shown in the equation 5.64

$$p(x) = a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6 + hx^7 + ix^8 + lx^9 \quad (5.64)$$

The only problem of the *Curve Fitting* is that the algorithm inside it has to calculate all coefficients of the polynomial. This is not possible in the case of the Standard Test where some interpolations follow specific trends. In the case of the back e.m.f and

the voltage supply, the function is linear and the only coefficient physically allowed is  $theb$ .

To solve this problem, the function is modified to set some coefficients to zero. In the Execute and Post-elaboration Tool, there is a section where insert the coefficient desired.

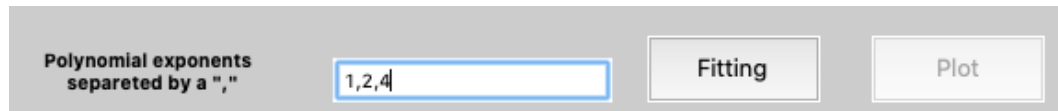


Figure 5.13: interpolation interface.

The code to implement the modify is the following:

```

1      def test3(x, a, b, c, d, e, f, g, h, i, l):
2
3          a1 = 0
4          b1 = 0
5          c1 = 0
6          d1 = 0
7          e1 = 0
8          f1 = 0
9          g1 = 0
10         h1 = 0
11         i1 = 0
12         l1 = 0
13         for indice in range(lung):
14
15             if int(coeff[indice]) == 0:
16                 a1 = 1
17             if int(coeff[indice]) == 1:
18                 b1 = 1
19             if int(coeff[indice]) == 2:
20                 c1 = 1
21             if int(coeff[indice]) == 3:
22                 d1 = 1
23             if int(coeff[indice]) == 4:
24                 e1 = 1
25             if int(coeff[indice]) == 5:
26                 f1 = 1
27             if int(coeff[indice]) == 6:
28                 g1 = 1
29             if int(coeff[indice]) == 7:
30                 h1 = 1
31             if int(coeff[indice]) == 8:
32                 i1 = 1
33             if int(coeff[indice]) == 9:
34                 l1 = 1
35
36         if a1 == 0:
37             a = 0
38         if b1 == 0:
39             b = 0
40         if c1 == 0:

```

```

41         c = 0
42         if d1 == 0:
43             d = 0
44         if e1 == 0:
45             e = 0
46         if f1 == 0:
47             f = 0
48         if g1 == 0:
49             g = 0
50         if h1 == 0:
51             h = 0
52         if i1 == 0:
53             i = 0
54         if l1 == 0:
55             l = 0
56
57         return a + b * x + c * np.power(x, 2) + d * np.power(x, 3)
         + e * np.power(x, 4) + f * np.power(x, 5) + g * np.power(x, 6) +
         h * np.power(x, 7) + i * np.power(x, 8) + l * np.power(x, 9)

```

## 5.2 Elaboration GUI

The Execution and Post-Elaboration interface are shown in the Figure 5.14.

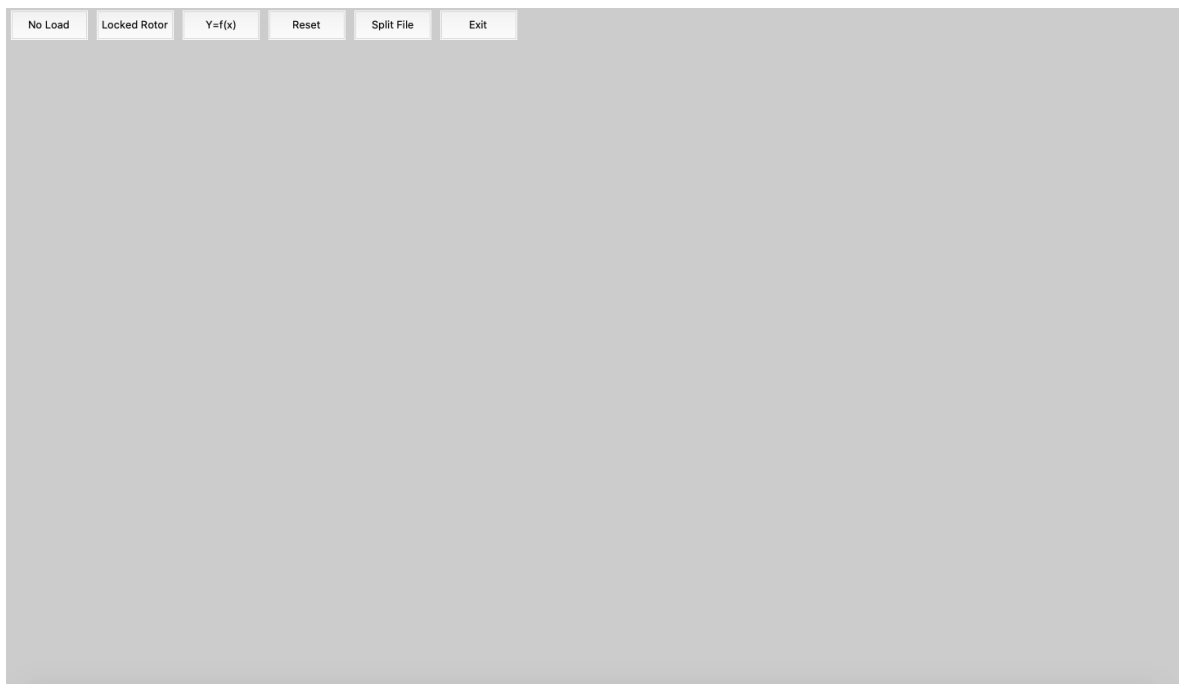


Figure 5.14: The Post-Elaboration GUI.

The interface allows to select four options, corresponding at four different buttons:

1. **No Load buttons:** Open an interface to calculate the No Load steady-state parameters
2. **Locked Rotor buttons:** Open an interface to calculate the Locked Rotor steady-state parameters

3. **Reset:** Reset the Tool, re-starting from the interface shown in Figure 5.14
4. **Exit:** Close the Tool

### 5.2.1 Interface for the No Load and Locked Rotor tests

The No load and the Locked Rotor interface are shown respectively in the Figure 5.15 and Figure 5.16

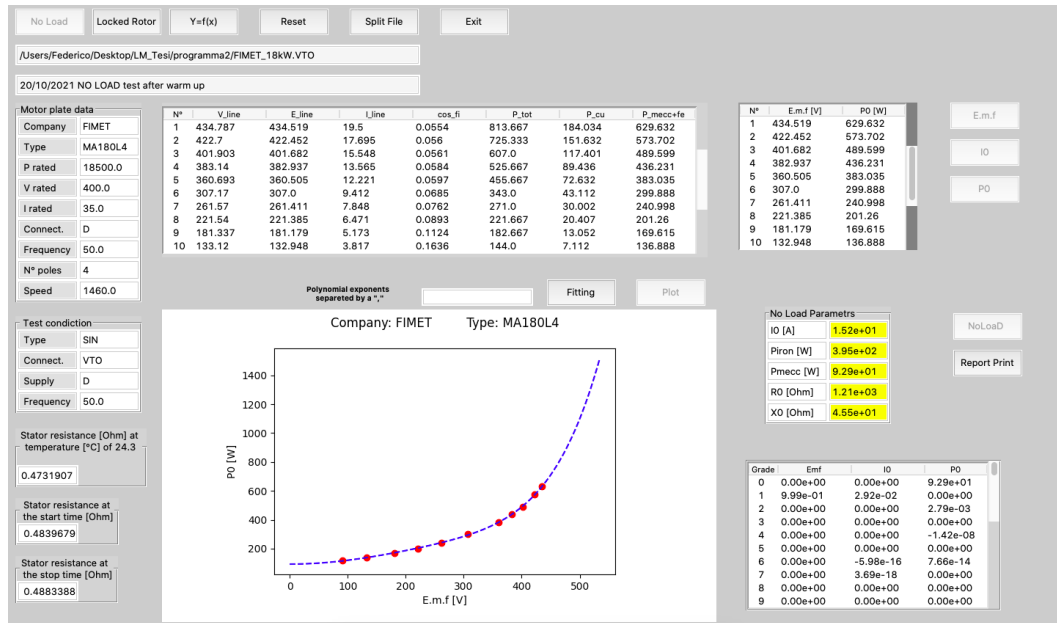


Figure 5.15: No Load GUI.

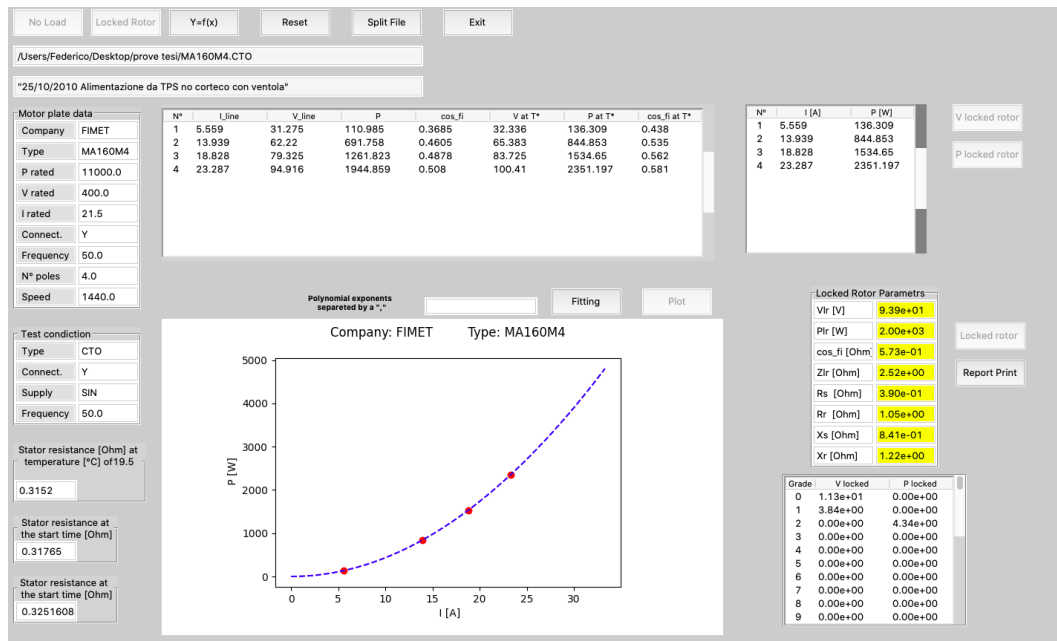


Figure 5.16: Locked Rotor GUI.

### 5.2.2 Elaboration of the test results

To start the elaboration it's necessary open a text file obtained with the Acquisition Tool. The choice of the file is done by selecting the "File" option in the menu bar and then the "Open file" voice, as shown in the Figure 5.17.

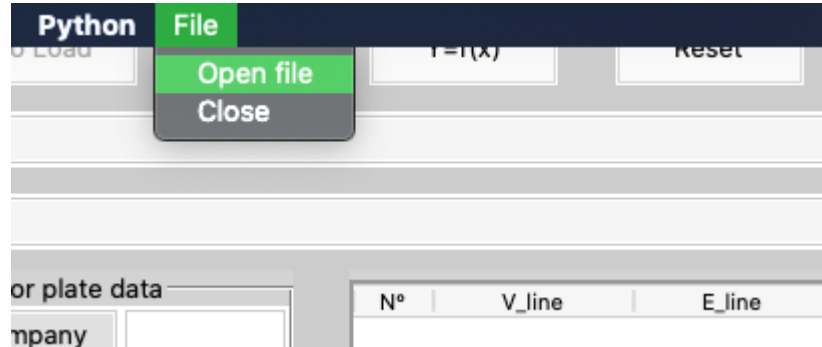


Figure 5.17: Open file option in the No load tool menu bar.

After the selection will open a secondary window to choose the file for the No Load test or the Locked Rotor test elaboration. It's important to notice that only the file with the .VTO extension can be selected for the No Load test, as shown in the Figure 5.18, and only the .CTO extension can be selected for the Locked Rotor.

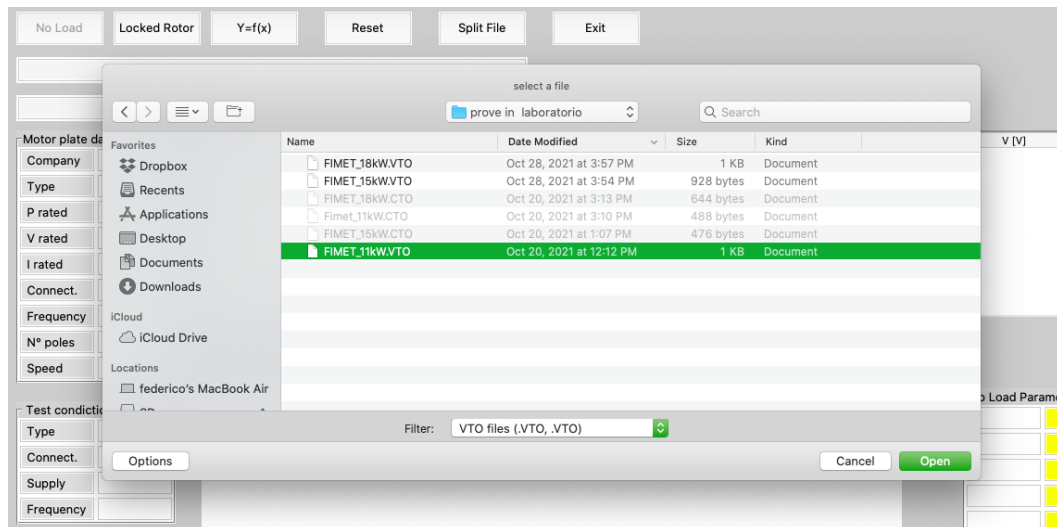


Figure 5.18: Selection of .VTO text file.

After the selection of the text file, all the data are transcribed on the tables interface as shown in the Figure 5.19 below.

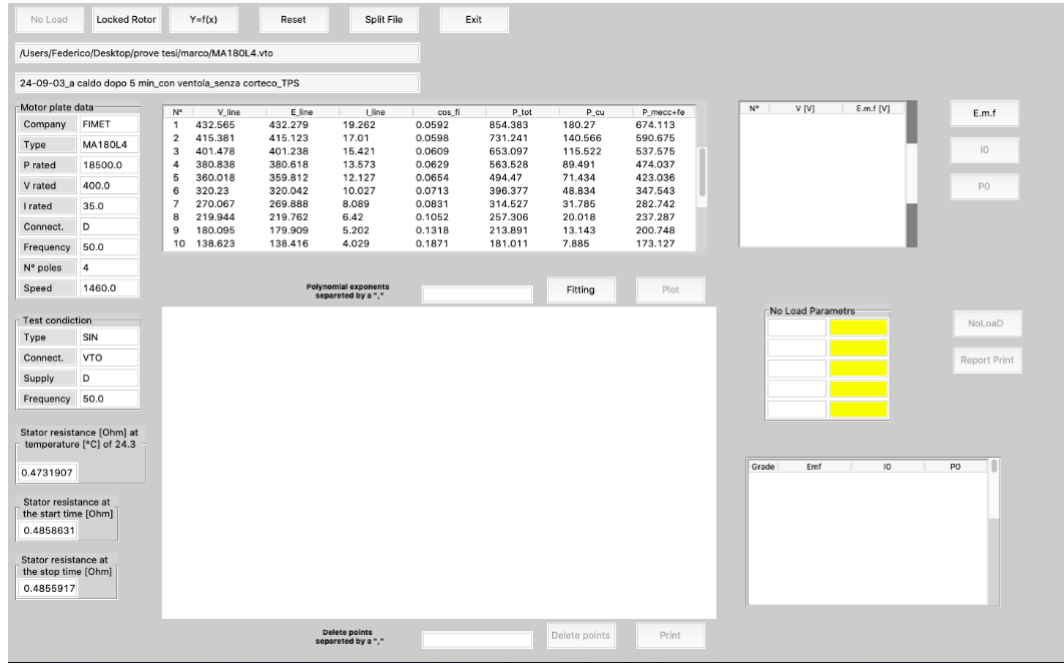


Figure 5.19: The No Load interface after the .VTO selection.

The table data shown in the Figure 5.20 after the selection of the standard test, presents eight different useful data:

| N° | V_line  | E_line  | I_line | cos_phi | P_tot   | P_cu    | P_mecc+fe |
|----|---------|---------|--------|---------|---------|---------|-----------|
| 1  | 432.565 | 432.279 | 19.262 | 0.0592  | 854.383 | 180.27  | 674.113   |
| 2  | 415.381 | 415.123 | 17.01  | 0.0598  | 731.241 | 140.566 | 590.675   |
| 3  | 401.478 | 401.238 | 15.421 | 0.0609  | 653.097 | 115.522 | 537.575   |
| 4  | 380.838 | 380.618 | 13.573 | 0.0629  | 563.528 | 89.491  | 474.037   |
| 5  | 360.018 | 359.812 | 12.127 | 0.0654  | 494.47  | 71.434  | 423.036   |
| 6  | 320.23  | 320.042 | 10.027 | 0.0713  | 396.377 | 48.834  | 347.543   |
| 7  | 270.067 | 269.888 | 8.089  | 0.0831  | 314.527 | 31.785  | 282.742   |
| 8  | 219.944 | 219.762 | 6.42   | 0.1052  | 257.306 | 20.018  | 237.287   |
| 9  | 180.095 | 179.909 | 5.202  | 0.1318  | 213.891 | 13.143  | 200.748   |
| 10 | 138.623 | 138.416 | 4.029  | 0.1871  | 181.011 | 7.885   | 173.127   |

Figure 5.20: Example of Table data for the No Load test.

1.  $N^\circ$ : the numbers of measurements carried out during the No Load test;
2.  $V_{line}$ : The line voltages used during the test;
3.  $E_{line}$ : The back e.m.f generated by the induction motor for each supply voltage;
4.  $I_{line}$ : The line current measured on the No Load test;
5.  $\cos(\phi_i)$ : The power factor;
6.  $P_{tot}$ : The total active power of the induction motor during during the test;
7.  $P_{Cu}$ : The Joule active power losses;
8.  $P_{mecc+fe}$ : The active power losses per mechanical and ferromagnetic effect.

The table data for the Locke Rotor test is different and present others information that are present in the .CTO file and shown in the Figure 5.21.

| N° | I_line | V_line | P        | cos_fi | V at T* | P at T*  | cos_fi at T* |
|----|--------|--------|----------|--------|---------|----------|--------------|
| 1  | 5.387  | 17.525 | 49.894   | 0.3051 | 17.859  | 59.326   | 0.356        |
| 2  | 14.498 | 33.769 | 339.819  | 0.4007 | 34.828  | 401.576  | 0.459        |
| 3  | 24.242 | 50.315 | 933.663  | 0.4419 | 52.185  | 1099.855 | 0.502        |
| 4  | 32.107 | 63.576 | 1629.512 | 0.4609 | 66.089  | 1913.897 | 0.521        |
| 5  | 34.039 | 66.919 | 1834.629 | 0.465  | 69.502  | 2142.58  | 0.523        |
| 6  | 36.887 | 71.798 | 2159.031 | 0.4707 | 74.571  | 2513.563 | 0.528        |

Figure 5.21: Example of Table data for the Locked Rotor test.

1.  $N^\circ$ : the numbers of measurements carried out during the No Load test;
2.  $V_{line}$ : The line voltages used during the test;
3.  $E_{line}$ : The back e.m.f generated by the induction motor for each supply voltage;
4.  $I_{line}$ : The line current measured on the No Load test;
5.  $\cos(fi)$ : The power factor;
6.  $P_{tot}$ : The total active power of the induction motor during during the test;
7.  $P_{Cu}$ : The Joule active power losses;
8.  $P_{mecc+fe}$ : The active power losses per mechanical and ferromagnetic effect.

After the interpolation section described in the dedicated section, the Elaboration tool GUI reports the steady-state parameter results and the polynomial coefficients of the interpolating functions as shown in Figure 5.22.

| No Load Parametrs |          |  |  |
|-------------------|----------|--|--|
| IO [A]            | 1.54e+01 |  |  |
| Piron [W]         | 4.08e+02 |  |  |
| Pmecc [W]         | 1.26e+02 |  |  |
| R0 [Ohm]          | 1.18e+03 |  |  |
| X0 [Ohm]          | 4.51e+01 |  |  |

| Grade | Emf      | IO        | PO        |
|-------|----------|-----------|-----------|
| 0     | 0.00e+00 | 0.00e+00  | 1.26e+02  |
| 1     | 9.99e-01 | 3.10e-02  | 0.00e+00  |
| 2     | 0.00e+00 | -6.19e-06 | 2.75e-03  |
| 3     | 0.00e+00 | 0.00e+00  | 0.00e+00  |
| 4     | 0.00e+00 | -1.07e-10 | -1.39e-08 |
| 5     | 0.00e+00 | 0.00e+00  | 0.00e+00  |
| 6     | 0.00e+00 | 1.64e-15  | 7.92e-14  |
| 7     | 0.00e+00 | 0.00e+00  | 0.00e+00  |
| 8     | 0.00e+00 | 0.00e+00  | 0.00e+00  |
| 9     | 0.00e+00 | 0.00e+00  | 0.00e+00  |

Figure 5.22: Example of test results table.

Moreover, the tool allows to report all the data obtained during the test in a PDF version through the "Report print" button.

# Chapter 6

## Experimental validation

This Chapter describes the experimental validation of the Python program in a real case. The Acquisition and Elaboration Tools were validated in the Polito's laboratory performing the Standard Tests on three different induction motors. The results were compared to those obtained with the Visual Basic application. Two different validation methodologies are investigated, respectively for Acquisition and Elaboration tool.

### 6.1 Measurement setup

The validation test was executed at the Polito electromechanical laboratory, on 20<sup>th</sup> October 2021.

#### 6.1.1 The Induction Motors used for the tests

The test was executed on three different induction motors by the Fimet company, of which all data are known through the use of the Visual Basic application in 2003. The motors were arranged on a wood platform (Figure 6.1) close to the power supply and the power meter.



Figure 6.1: The induction motors used for the tests.



The specific models and motors plate data are shown in the tables below.

1. FIMET MA160M4 11 kW

|              |             |
|--------------|-------------|
| Power        | 11 kW       |
| Voltage      | 400/230 V   |
| Current      | 21,5/37,5 A |
| Power factor | 0.83        |
| Frequency    | 50 Hz       |
| Speed        | 1440 rpm    |

Table 6.1: FIMET MA160M4 11 kW Motor data.

2. FIMET HMA160L4 15 kW

|              |             |
|--------------|-------------|
| Power        | 15 kW       |
| Voltage      | 400/230 V   |
| Current      | 21,5/37,5 A |
| Power factor | 0.83        |
| Frequency    | 50 Hz       |
| Speed        | 1440 rpm    |

Table 6.2: FIMET HMA160L4 15 kW Motor data.

3. FIMET MA180L4 18.5 kW

|              |           |
|--------------|-----------|
| Power        | 18.5 kW   |
| Voltage      | 690/400 V |
| Current      | 20/35 A   |
| Power factor | 0.83      |
| Frequency    | 50 Hz     |
| Speed        | 1460 rpm  |

Table 6.3: FIMET MA180L4 18.5 kW Motor data.

The other setup used for the tests are shown in the Figure 6.2

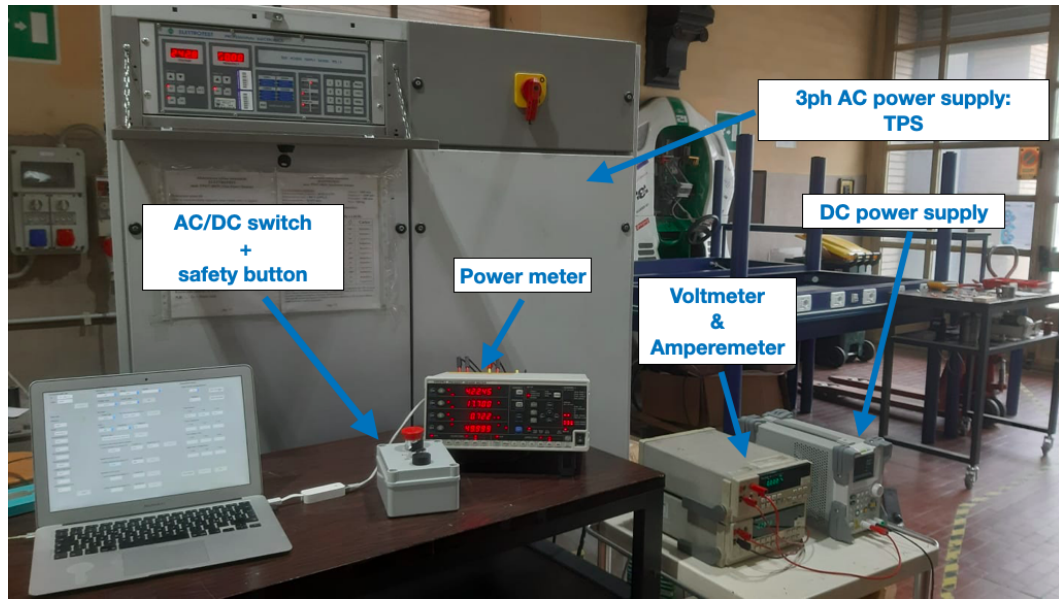


Figure 6.2: Laboratory setup.

### 6.1.2 The Three Phase Supply source

During the tests, the induction motors were supplied by a 3-ph AC power source (TPS) by Electrotest company, which can be programmed in voltage and frequency with power from 1 KVA to 200 KVA. This generator is designed to supply a sinusoidal voltage with elevated purity, stability and precision[9].



Figure 6.3: The TPS power supplier.

### 6.1.3 The measurement instruments

To calculate the winding resistance of the motors, the setup was included:

- DC voltage source to supply the stator windings;

- amperometer;
- voltmeter.

This setup is shown in the Figure 6.4.

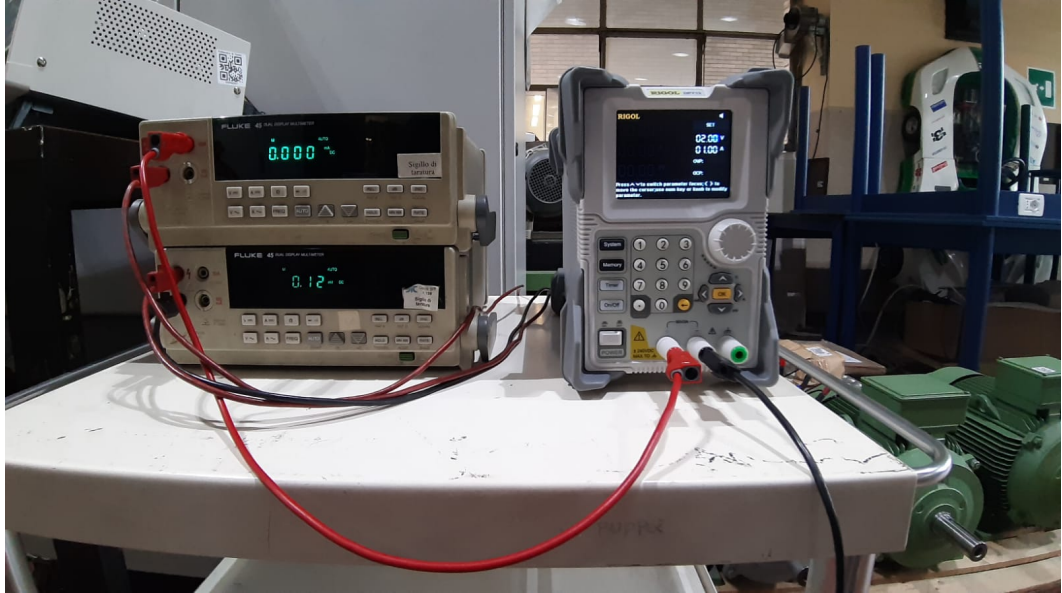


Figure 6.4: DC measurements setup.

A switch shown in the Figure 6.5 allows to choose change the power supply from DC to AC and vice versa. A safety button was included in the switch to interrupt the supply in case of emergency.



Figure 6.5: Switch with a safety button.

## 6.2 Verification methodology

The verification methodology is divided in two parts:

### 1. Elaboration tool validation

This validation compares the steady-state parameters results obtained starting from the same Visual Basic text files and elaborated with both post-elaboration Tools, Python and Visual Basic. The methodology is described by the green and red paths of the Figure 6.6;

### 2. Acquisition tool validation

The Acquisition validation compares the steady-state parameters results obtained with the use of Visual Basic and Python Tools on the same induction motors. It's described with the green and blue paths of the Figure 6.6. This validation is executed both for the No Load test and Locked Rotor test.

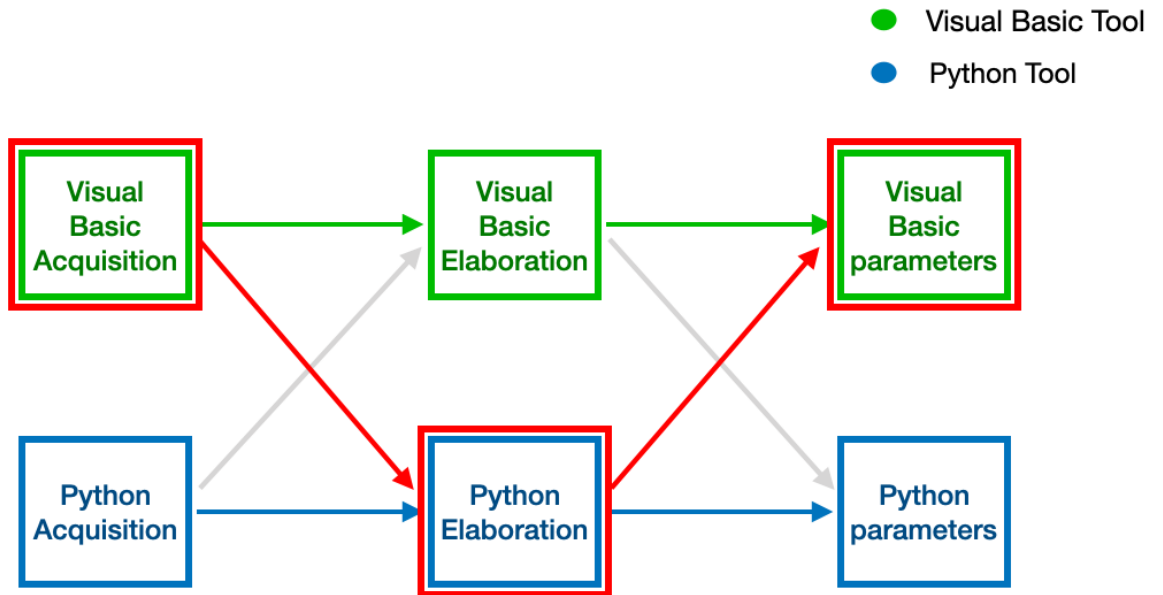


Figure 6.6: Verifying methodology scheme.

## 6.3 Experimental Results

This section shows the results obtained during the validation test, with the Python tool, on 20 October 2021. Furthermore, the comparison is performed with the results obtained with the Visual Basic tool on the same motors in 2003. The experimental results refer to the 18 kW motor only.

### 6.3.1 Elaboration tool validation

The Elaboration tool validation is shown graphically in the Figure 6.7

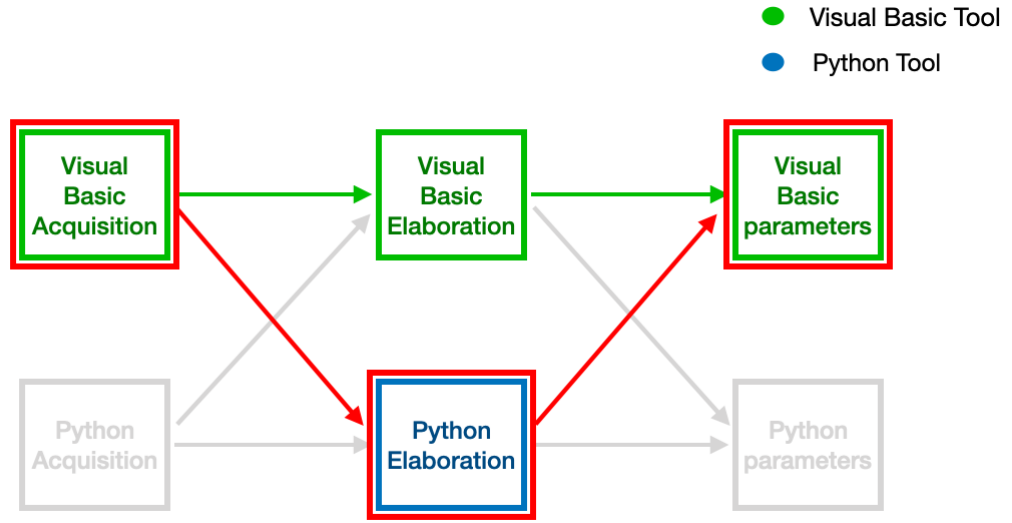


Figure 6.7: Elaboration tool validation methodology scheme.

The steady-state parameters of the induction motors obtained are identically for each validation path. The Python Elaboration Tool is validated since that starting from the same text files as inputs it determines the same results as those obtained with the Visual Basic elaboration tool.

### 6.3.2 Acquisition tool validation

The Acquisition tool validation methodology is shown in the Figure 6.8.

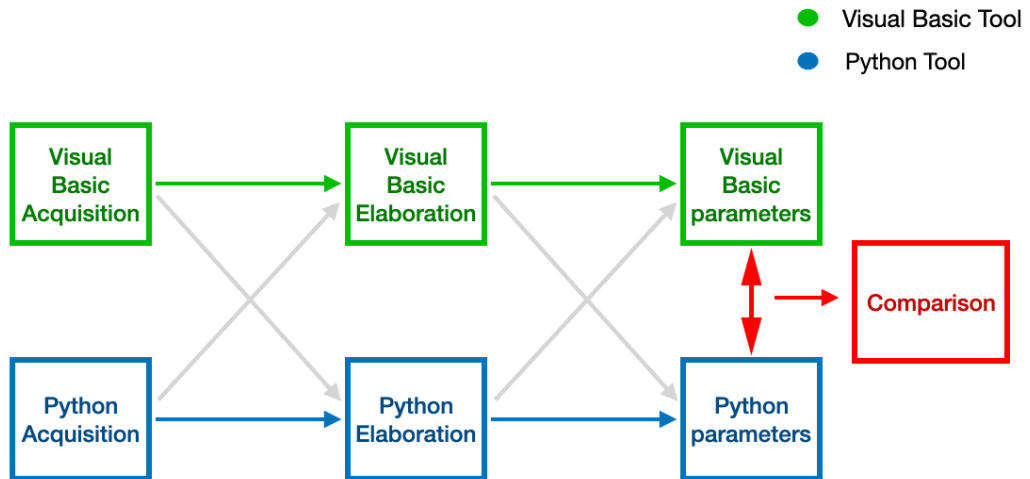


Figure 6.8: Acquisition tool validation methodology scheme.

Comparison of results between two different Python and Visual Basic tools on the same motor is analyzed for both No Load and Locked Rotor tests.

### No Load test results

The comparison between the No Load parameters of the FIMET 18.5 kW induction motor is shown in the table 6.4.

| No Load Parameters                                      | Visual Basic | Python | percentage relative error |
|---------------------------------------------------------|--------------|--------|---------------------------|
| No Load Current $I_0$ (A)                               | 15.4         | 15.3   | 0.65%                     |
| Iron Losses $P_{iron}$ (W)                              | 408          | 397    | 2.69%                     |
| Mechanical Losses $P_{mech}$ (W)                        | 126          | 107    | 15.0%                     |
| Equivalent iron losses resistance $R_{fe}$ ( $\Omega$ ) | 1180         | 1210   | 2.54%                     |
| Stator Reactance $X_0$ ( $\Omega$ )                     | 45.1         | 45.2   | 0.22%                     |

Table 6.4: No Load parameters comparison for the 18.5 kW induction motor.

The mechanical power losses error is due to possible mechanical changes or maintenance carried out on the induction motor during the years (the acquisition data collected with the Visual Basic acquisition tool dates back to 2003).

The No Load interpolation functions comparison obtained with the two tools, Visual Basic and Python, are shown in the figures below.

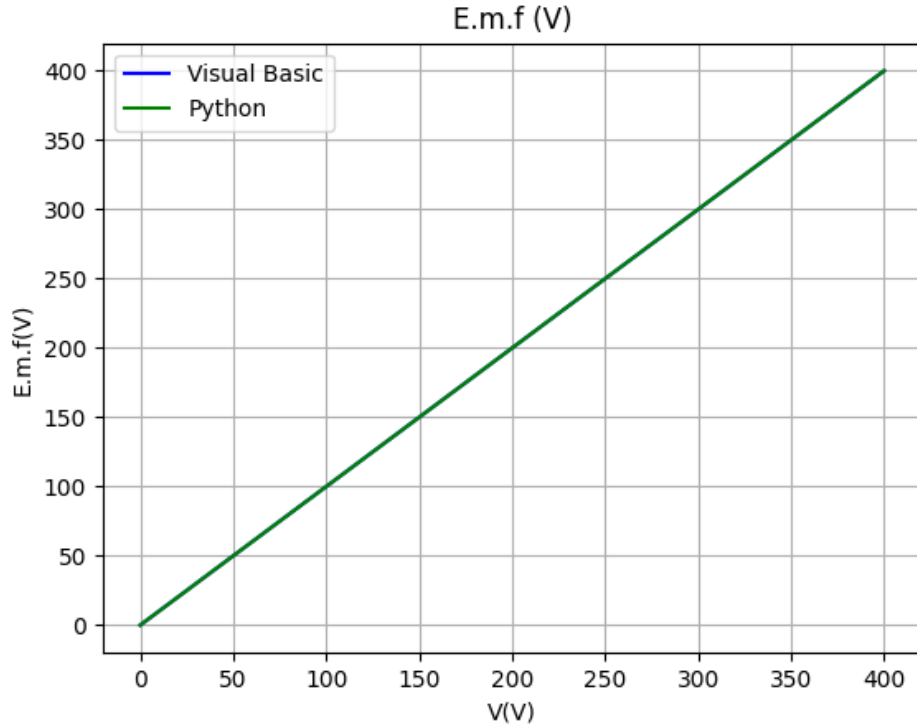


Figure 6.9: No Load E.m.f comparison.

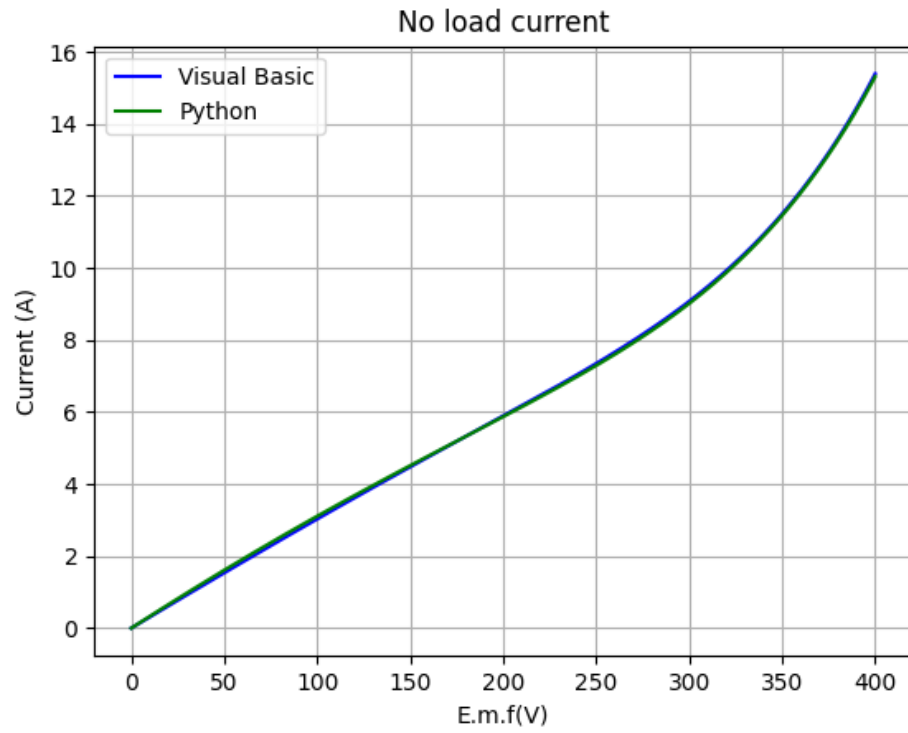


Figure 6.10: No Load current comparison.

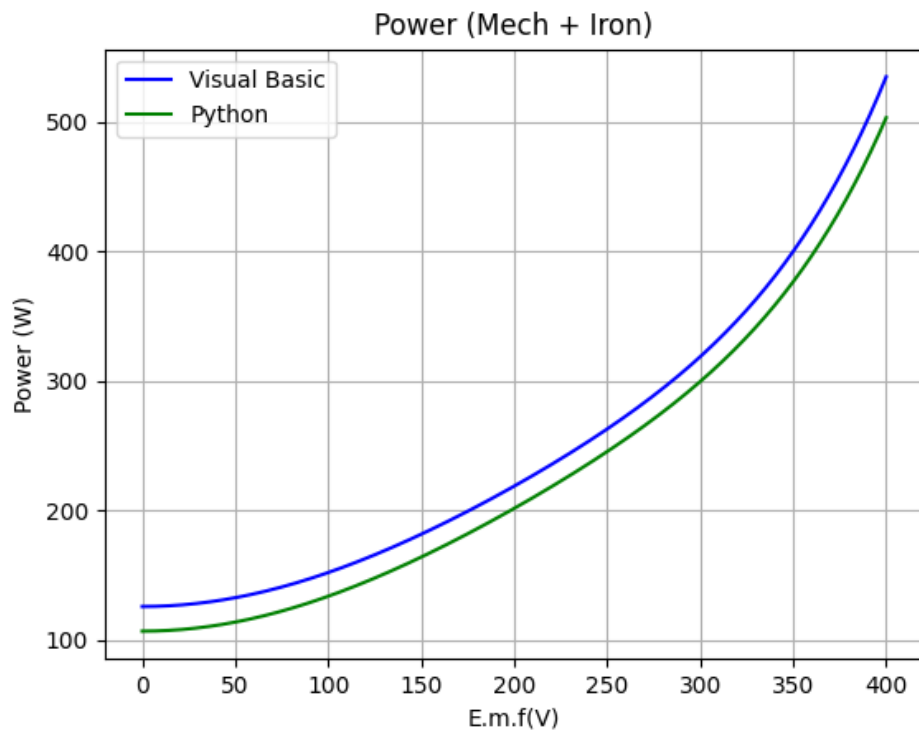


Figure 6.11: Mechanical and Iron Power losses comparison.



### Locked Rotor test results

The comparison between the No Load parameters of the FIMET 18.5 kW induction motor is shown in the table 6.5. The Locked Rotor interpolation functions comparison

| Locked Rotor parameters | Visual Basic | Python | percentage relative error |
|-------------------------|--------------|--------|---------------------------|
| $V_{lr}$ (V)            | 71.2         | 70.9   | 0.42%                     |
| $P_{lr}$ (W)            | 225          | 223    | 0.89%                     |
| $\cos\varphi$           | 0.522        | 0.520  | 0.38%                     |
| $Z_{lr}$ ( $\Omega$ )   | 3.52         | 3.51   | 0.28%                     |
| $R_s$ ( $\Omega$ )      | 0.566        | 0.566  | $\simeq 0\%$              |
| $R_r$ ( $\Omega$ )      | 1.27         | 1.26   | 0.79%                     |
| $X_s$ ( $\Omega$ )      | 1.25         | 1.25   | $\simeq 0\%$              |
| $X_r$ ( $\Omega$ )      | 1.76         | 1.75   | 0.57%                     |

Table 6.5: Locked Rotor parameters comparison for the 18.5 kW motor.

obtained with the two tools, Visual Basic and Python, are shown in the figures below.

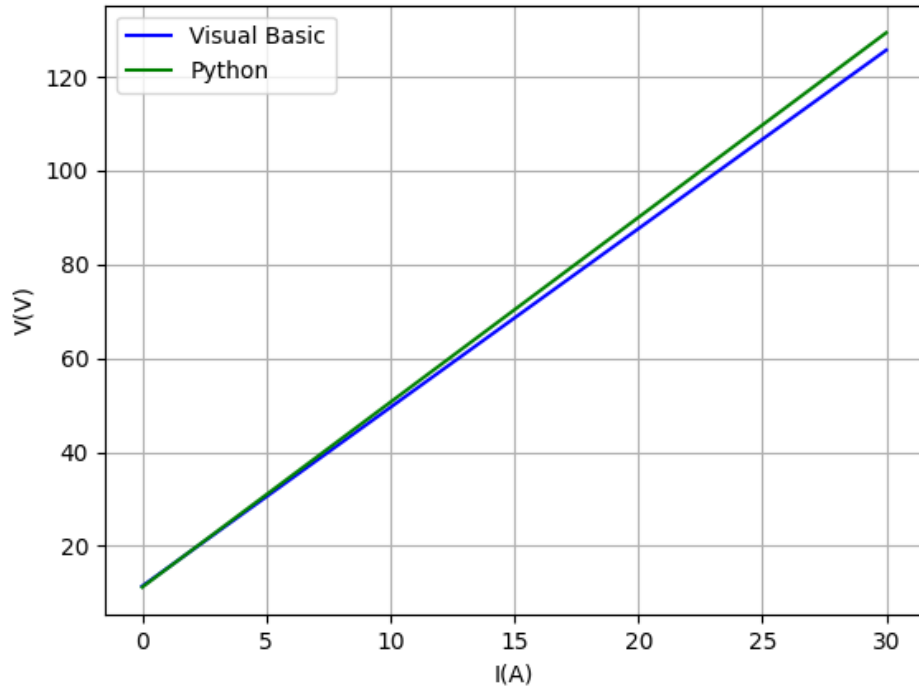


Figure 6.12: No Load E.m.f comparison.



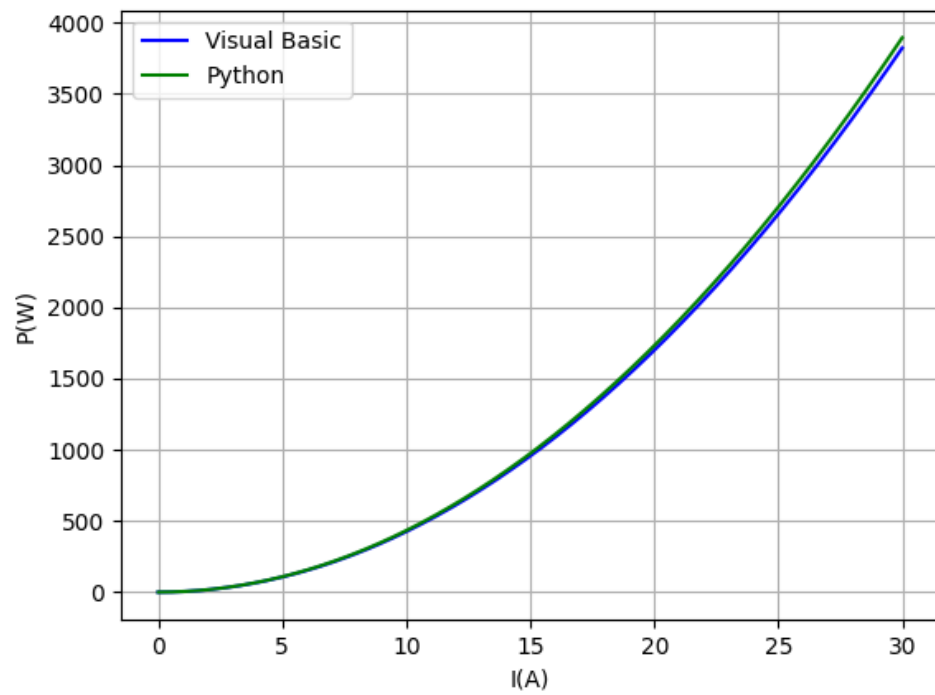


Figure 6.13: No Load current comparison.

Since the Elaboration tool is already validated, the validation of the Acquisition tool allows to validate the entire Python application developed.

# Chapter 7

## Conclusion

The aim of the thesis was the development of the acquisition codes for the execution and post-elaboration of Standard tests on induction motors. This work demonstrates the excellent capabilities of Python in realizing the acquisition codes for the execution and post-elaboration of standard tests on induction motors.

The key contributions of the Thesis are:

- Improvement of the Acquisition tool by implementing new features in the setting of the instrument and in the multiple acquisition sections;
- Simplification of the Elaboration tool by the integration and modification of standard Python library functions in the code;
- Experimental validation of the Python tool in the laboratory of the Standard tests on three different induction motors.

Potential future developments:

- Addition of new tool for the determination of the electromechanical characteristics of the induction motors, starting from the steady-state parameters.

# Chapter 8

## Appendix

### 8.1 A. Acquisition tool Python codes

```
1 from tkinter import *
2 from tkinter import messagebox
3 from lan import Lan
4 import time
5 from threading import Thread
6 from time import sleep
7 from PIL import Image, ImageTk
8
9 # Timeout(1sec)
10 Timeout_default = 1
11
12 indice_grap = 0
13 indice_inter = 0
14 block = 0
15
16 #create the main window
17
18 window = Tk()
19 window.configure(bg='#cccccc')
20 #window.attributes('-fullscreen', True)
21 window.title('Data Acquisition Tool for Standard Tests on Induction
    Motors')
22 window.geometry('1300x800')
23
24
25 def save1():
26
27     B_salve['state'] = 'normal'
28     save_button['state'] = 'disabled'
29     global targa_data
30     global targa
31     targa_data = StringVar()
32     targa_data = box1.get()
33     targa_data = targa_data + ',' + box2.get()
34     targa_data = targa_data + ',' + box3.get()
35     targa_data = targa_data + ',' + box4.get()
36     targa_data = targa_data + ',' + box5.get()
37     targa_data = targa_data + ',' + box6.get()
38     targa_data = targa_data + ',' + box7.get()
```

```

39     targa_data = targa_data + ',' + box8.get()
40     targa_data = targa_data + ',' + box9.get()
41     targa = []
42     for w in targa_data.split(','):
43         targa.append(w)
44     print(targa)
45     file = open(open_file(), 'w')
46     file.write(targa[0] + ',' + targa[1] + '\n' + targa[5] + ',' +
47               targa[2] + ',' + targa[3] + ',' + targa[4] + ',' + targa[8] + ',' +
48               + targa[7] + ',' + targa[6])
49
50
51 def Zeroadj():          # performing Zero-adjustment (about 40 sec)
52     lan.sendMsg(':DEMAg')
53
54 def only_number(value):
55     value = str(value[4:])
56     value = str(value[:-4])
57     value = float(value[2:])
58     return value
59
60 def only_number1(value):
61     #value = str(value[4:])
62     #value = str(value[:-4])
63     value = float(value[5:])
64     return value
65
66 def only_number_PK(value):
67     value = str(value[9:])
68     value = str(value[:-5])
69     value = float(value[2:])
70     return value
71
72 #funzione selected da cancellare probabilmente
73 def selected():
74     if wiring_range.get() == 'Aron':
75         #lan = Lan(Timeout_default)
76         lan.sendMsg('WIR TYPE3')
77         #v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
78         #casella_v1.insert(END, v1)
79
80     if wiring_range.get() == 'tipo 1':
81         #lan = Lan(Timeout_default)
82         lan.sendMsg('WIR TYPE1')
83
84     if wiring_range.get() == '3 Wattmetri':
85         #lan = Lan(Timeout_default)
86         lan.sendMsg('WIR TYPE5')
87         #v2 = lan.SendQueryMsg('MEAS? U3', Timeout_default)
88         #casella_v1.insert(END, v2)
89
90     if coupling_range.get() == 'AC+DC':
91         #lan = Lan(Timeout_default)
92         lan.sendMsg('AOUT:ITEM:U1 RMS') # ??????????
93
94     if coupling_range.get() == 'AC':
95         #lan = Lan(Timeout_default)
96         lan.sendMsg('DISP:ITEM:U1 FND') # ??????????

```

```

94
95     if measure.get() == '1':
96
97         v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
98         print(v1)
99         box_Vrms.delete(0, END)
100        box_Vrms.insert(0, v1)
101
102
103     if measure.get() == '3':
104         global sum_v
105         sum_v = 0.0
106
107         index = 50
108         for i in range(index):
109             v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
110             print("{:.3e}".format(only_number(v1)), v1)
111             sum_v = sum_v + only_number(v1)
112             #Timeout_default = Timeout_default + 0.1
113         v2 = "{:.3e}".format(sum_v/index)
114         box_Vrms.insert(0, v2)
115
116
117     if measure.get() == '5':
118
119         v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
120
121     if measure.get() == '10':
122         v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
123
124     if measure.get() == '15':
125         v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
126
127     if measure.get() == '20':
128         v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
129
130 def settings():
131
132
133     if wiring_range.get() == 'Aron':
134         lan.sendMsg('WIR TYPE3')
135
136     if wiring_range.get() == '3 Wattmetri':
137         lan.sendMsg('WIR TYPE5')
138
139
140
141     if coupling_range.get() == 'AC+DC':
142         lan.sendMsg(':DISP U0,I0,P0')
143
144     if coupling_range.get() == 'AC':
145         lan.sendMsg(':DISP UAC0,IAC0,PAC0')
146
147
148
149     if voltage_range.get() != 'AUTO':
150         comm = voltage_range.get()

```

```

151     lan.sendMsg(':VOLT:RANG '+comm[0:-1])
152     if voltage_range.get() == 'AUTO':
153         lan.sendMsg(':VOLT:AUTO ON')
154
155     if current_range.get() == '200mA':
156         lan.sendMsg(':CURR:RANG 0.2')
157     if current_range.get() == '500mA':
158         lan.sendMsg(':CURR:RANG 0.5')
159     if current_range.get() == '1A':
160         lan.sendMsg(':CURR:RANG 1.0')
161     if current_range.get() == '2A':
162         lan.sendMsg(':CURR:RANG 2.0')
163     if current_range.get() == '5A':
164         lan.sendMsg(':CURR:RANG 5.0')
165     if current_range.get() == '10A':
166         lan.sendMsg(':CURR:RANG 10.0')
167     if current_range.get() == '20A':
168         lan.sendMsg(':CURR:RANG 20.0')
169     if current_range.get() == '50A':
170         lan.sendMsg(':CURR:RANG 50.0')
171
172     if current_range.get() == 'AUTO':
173         lan.sendMsg(':CURR:AUTO ON')
174
175     else:
176
177         pass
178
179 '''
180 def start_thread():
181     # Assign global variable and initialize value
182     global stop
183     stop = 0
184
185     # Create and launch a thread
186     t = Thread(target=acquisition)
187     t.start()
188 '''
189 def acquisition():
190     print(measure.get())
191     N = int(measure.get())
192     max_V = 0
193     min_V = 10000000000000
194     sum_V = 0
195     max_I = 0
196     min_I = 10000000000000
197     sum_I = 0
198     max_P = 0
199     min_P = 10000000000000
200     sum_P = 0
201     v = []
202     i1 = []
203     p = []
204
205     if wiring_range.get() == 'Aron':
206
207         if coupling_range.get() == 'AC+DC':

```

```

208
209         for j in range(0, N):
210             box_meas.delete(0, 3)
211             v.append(only_number(lan.SendQueryMsg('MEAS? U0',
Timeout_default)))
212             i1.append(only_number(lan.SendQueryMsg('MEAS? I0',
Timeout_default)))
213             p.append(only_number(lan.SendQueryMsg('MEAS? P0',
Timeout_default)))
214             print(v[j])
215             box_meas.insert(END, j + 1)
216             if v[j] > max_V:
217                 max_V = v[j]
218             if v[j] < min_V:
219                 min_V = v[j]
220             sum_V = sum_V + v[j]
221             if i1[j] > max_I:
222                 max_I = i1[j]
223             if i1[j] < min_I:
224                 min_I = i1[j]
225             sum_I = sum_I + i1[j]
226             if p[j] > max_P:
227                 max_P = p[j]
228             if p[j] < min_P:
229                 min_P = p[j]
230             sum_P = sum_P + p[j]
231             sleep(1) #
232             med_V = round(sum_V / N, 4)
233             med_I = round(sum_I / N, 4)
234             med_P = round(sum_P / N, 4)
235             named_tuple = time.localtime() # get struct_time
236             time_string = time.strftime("%H.%M.%S", named_tuple)
237             file = open(open_file(), 'a')
238             file.write('\n' + str(min_V) + ',' + str(med_V) + ',' +
str(max_V) + ',' + str(min_I) + ',' + str(med_I) + ',' + str(max_I
) + ',' + str(min_P) + ',' + str(med_P) + ',' + str(max_P) + ',' +
time_string)
239
240         if coupling_range.get() == 'AC':
241
242             for j in range(0, N):
243                 box_meas.delete(0, 3)
244                 box_Vrms.delete(0, 40)
245                 box_Irms.delete(0, 40)
246                 box_Prms.delete(0, 40)
247                 v.append(only_number(lan.SendQueryMsg('MEAS? UACO',
Timeout_default)))
248                 i1.append(only_number(lan.SendQueryMsg('MEAS? IACO',
Timeout_default)))
249                 p.append(only_number1(lan.SendQueryMsg('MEAS? PACO',
Timeout_default)))
250                 box_Vrms.insert(END, v[j])
251                 box_Irms.insert(END, i1[j])
252                 box_Prms.insert(END, p[j])
253
254                 box_meas.insert(END, j + 1)
255                 if v[j] > max_V:

```

```

256         max_V = v[j]
257         if v[j] < min_V:
258             min_V = v[j]
259         sum_V = sum_V + v[j]
260         if i1[j] > max_I:
261             max_I = i1[j]
262         if i1[j] < min_I:
263             min_I = i1[j]
264         sum_I = sum_I + i1[j]
265         if p[j] > max_P:
266             max_P = p[j]
267         if p[j] < min_P:
268             min_P = p[j]
269         sum_P = sum_P + p[j]
270         sleep(1) #
271         med_V = round(sum_V / N, 4)
272         med_I = round(sum_I / N, 4)
273         med_P = round(sum_P / N, 4)
274         named_tuple = time.localtime() # get struct_time
275         time_string = time.strftime("%H.%M.%S", named_tuple)
276         file = open(open_file(), 'a')
277         file.write('\n' + str(min_V) + ',' + str(med_V) + ',' +
str(max_V) + ',' + str(min_I) + ',' + str(med_I) + ',' + str(max_I
) + ',' + str(min_P) + ',' + str(med_P) + ',' + str(max_P) + ',' +
time_string)
278
279 def stop():
280     B_salve3['state'] = 'normal'
281     B_salve4['state'] = 'normal'
282
283
284 #if coupling_range.get() == 'AC+DC':
285
286
287     #if wiring_range.get() == '3 Wattmetri':
288
289     '''
290     if measure.get() == '1':
291         while True:
292             #for i in range(0,30):
293                 v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
294                 #print("{:.4e}".format(only_number(v1))),time_string)
295                 box_meas.delete(0, 3)
296                 box_meas.insert(END, i)
297                 named_tuple = time.localtime() # get struct_time
298                 time_string = time.strftime("%H.%M.%S", named_tuple)
299                 print("{:.4e}".format(only_number(v1))), time_string)
300                 i=i+1
301                 file = open(open_file(), 'a')
302                 file.write('\n'+str(only_number(v1)) + ',' + time_string)
303                 file.close()
304                 if stop==1:
305                     break
306
307
308     if measure.get() == '5':
309         while True:

```



```

310         #for i in range(0,30):
311             v = []
312             v[0] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default))
313             v[1] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default))
314             v[2] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default))
315             v[3] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default))
316             v[4] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default))
317             max=v[0]
318             min=v[0]
319             sum=v[0]
320             for i in range(1, 4):
321                 v[i] = only_number(lan.SendQueryMsg('MEAS? UDC1',
Timeout_default)) # prova senza only number
322                 sum=sum+v[i]
323                 if v[i]>max:
324                     max=v[i]
325                 if v[i]<min:
326                     min=v[i]
327             media=sum/i
328             #print("{:.4e}".format(only_number(v1))),time_string)
329             box_meas.delete(0, 3)
330             box_meas.insert(END, i)
331             named_tuple = time.localtime() # get struct_time
332             time_string = time.strftime("%H.%M.%S", named_tuple)
333             print("{:.4e}".format(only_number(v1))), time_string)
334             i=i+1
335             file = open(open_file(), 'a')
336             file.write('\n'+str(min) + ', ' +str(media)+' ,'+str(max)
+', '+time_string)
337             file.close()
338             if i == 10+1:
339                 break
340     '''
341 def open_file():
342     B_openfile['state'] = 'disabled'
343     save_button['state'] = 'normal'
344
345     filename = casella3.get()
346     if prove_type.get() == 'No load':
347         ext = '.VTO'
348     if prove_type.get() == 'Locked rotor':
349         ext = '.CTO'
350     if prove_type.get() == 'Load':
351         ext = '.CAR'
352     if prove_type.get() == '':
353         messagebox.showinfo('Attention', 'Have to select the prove
type')
354     filename1 = filename + ext
355
356     return filename1
357
358 def create_file():

```

```

359
360     file = open(open_file(), 'w')
361     file.write(targa[0]+',' + targa[1] + '\n' + targa[5]+',' + targa[2]+
362     ',' + targa[3]+',' + targa[4]+',' + targa[8]+',' + targa[7]+',' + targa[6])
363
364 def put_prove():
365
366     B_salve['state'] = 'disabled'
367     B_save2['state'] = 'normal'
368     supp = supply.get()
369     conn = connection.get()
370     freq = casella43.get()
371     dataenote = casella4.get()
372
373     file = open(open_file(), 'a')
374     name1 = open_file()
375     file.write('\n'+supp+',' + freq+',' + conn+',' + name1[-3:len(name1)]+'\\
376     n'+dataenote)
377
378 def resist_amb():
379     B_save2['state'] = 'disabled'
380     B_save3['state'] = 'normal'
381     file = open(open_file(), 'a')
382     file.write('\n'+casella5.get()+',' + casella6.get()+',' + casella7.get
383     ())
384
385 def resist_first():
386     B_save3['state'] = 'disabled'
387     B_stmisure['state'] = 'normal'
388     B_endmisure['state'] = 'normal'
389     file = open(open_file(), 'a')
390     file.write('\n' + casella51.get() + ',' + casella61.get())
391
392 def resit_last():
393     named_tuple = time.localtime() # get struct_time
394     time_string = time.strftime("%H.%M.%S", named_tuple)
395     file = open(open_file(), 'a')
396     file.write('\n' + '0' + ',' + casella53.get() + ',' + '0' + ',' + '0'
397     + ',' + casella63.get()+',' + '0'+',' + '0'+',' + '0'+',' + '0'+',' +
398     time_string)
399
400 def resit_last30():
401     named_tuple = time.localtime() # get struct_time
402     time_string = time.strftime("%H.%M.%S", named_tuple)
403     file = open(open_file(), 'a')
404     file.write('\n' + '0'+',' + casella54.get()+ ',' + '0'+',' + '0'+',' +
405     casella64.get()+',' + '0'+',' + '0'+',' + '0'+',' + '0'+',' + time_string)
406     file.close()
407
408 def connection1():
409     B_conn['state'] = 'disabled'
410     B_openfile['state'] = 'normal'
411
412     # Instantiation of the LAN communication class
413     global lan
414     lan = Lan(Timeout_default)
415
416     # Connect

```

```

410     IP = box_con.get()
411     port = int(box_port.get())
412     #print("IP?")
413     #IP = input()
414     #print("Port?")
415     #port = int(input())
416     if not lan.open(IP, port):
417         return
418
419 def r_conn():
420     global lan
421     lan = Lan(Timeout_default)
422
423     # Connect
424     IP = '192.168.1.2'
425     port = 3300
426     if not lan.open(IP, port):
427         return
428
429
430
431 '''
432     # Send and receive commands
433     while True:
434         # print("Please enter the command (Exit with no input)")      #
commented by Marco 21/07/2021
435         print("Please enter the command:")
436         command = input()
437         # Exit if no input
438         if command == "":
439             break
440         # If the command contains "?"
441         if "?" in command:
442             msgBuf = lan.SendQueryMsg(command, Timeout_default)
443             print(msgBuf)
444             # Send only
445         else:
446             lan.sendMessage(command)
447
448     lan.close()
449 '''
450
451 messagebox.showinfo('welcome', 'Please make sure the LAN cable is
connected before any action.')
452
453 #CONNECTION
454 connect = LabelFrame(window, text="Connection with LAN", font=(
'calibre', 12), bg='#cccccc')
455 connect.grid(row=0, column=0, padx=30, pady=10, rowspan=3, sticky='NW'
)
456 l_ip = Label(connect, text='IP', font=('calibre', 11), bg = '#cccccc')
457 l_ip.grid(row=0, column=0, padx=10, pady=5, sticky='W')
458 l_apath = Label(connect, text='(1)', font=('calibre', 11), bg='#cccccc
', fg='red')
459 l_apath.grid(row=0, column=2, padx=20, pady=5, sticky='E')
460 l_porta = Label(connect, text='Porta', font=('calibre', 11), bg='#
cccccc')

```

```

461 l_porta.grid(row=1, column=0, padx=10, pady=5, sticky='W')
462 box_con = Entry(connect, font=('calibre', 11), width=10)
463 box_con.grid(row=0, column=1)
464 box_con.insert(END, '192.168.1.2')
465 box_port = Entry(connect, font=('calibre', 11), width=10)
466 box_port.grid(row=1, column=1)
467 box_port.insert(END, '3300')
468 B_conn = Button(connect, text='Connect', font=('calibre', 11), height
    =1, width=8, command=connection1)
469 B_conn.grid(row=2, column=2, padx=10, pady=10)
470 #B_conn1= Button(connect,text = 'R_Conn', height=2, width=10, command=
    r_conn)
471 #B_conn1.grid(row=2, column=1, padx=10, pady=10)
472
473 #DATA
474 data = LabelFrame(window, text="Motor plate data ", font=('calibre',
    12), bg='#cccccc')
475 data.grid(row=3, column=0, padx=30, pady=2, rowspan=18, sticky='NW')
476 ditta_label = Label(data, text='Company name', font=('calibre', 10),
    bg = '#cccccc')
477 ditta_label.grid(row=0, column=0, pady=5, sticky='W')
478 num_operation = Label(data, text='(3)', font=('calibre', 10), bg='#
    ccccc', fg='red')
479 num_operation.grid(row=0, column=5, padx=10, pady=10, sticky='SE')
480 box1 = Entry(data, font=('calibre', 11), width=8)
481 box1.grid(row=1, column=0)
482 tipo_label = Label(data, text='Model/Type', font=('calibre', 10), bg =
    '#cccccc')
483 tipo_label.grid(row=2, column=0, pady=5, sticky='W')
484 box2 = Entry(data, font=('calibre', 11), width=8)
485 box2.grid(row=3, column=0)
486 tensione_label = Label(data, text='Rated voltage(V)', font=('calibre',
    10), bg = '#cccccc')
487 tensione_label.grid(row=4, column=0, pady=5, sticky='W')
488 box3 = Entry(data, font=('calibre', 11), width=8)
489 box3.grid(row=5, column=0)
490 corrente_label = Label(data, text='Rated current (A)', font=('calibre'
    , 10), bg = '#cccccc')
491 corrente_label.grid(row=6, column=0, pady=5, sticky='W')
492 box4 = Entry(data, font=('calibre', 11), width=8)
493 box4.grid(row=7, column=0)
494 coll_label = Label(data, text='Winding connection [Y/D]', font=(
    'calibre', 10), bg = '#cccccc')
495 coll_label.grid(row=8, column=0, pady=5, sticky='W')
496 box5 = Entry(data, font=('calibre', 11), width=8)
497 box5.grid(row=9, column=0)
498 power_label = Label(data, text='Rated power (W)', font=('calibre', 10)
    , bg = '#cccccc')
499 power_label.grid(row=10, column=0, pady=5, sticky='W')
500 box6 = Entry(data, font=('calibre', 11), width=8)
501 box6.grid(row=11, column=0)
502 tensione_label = Label(data, text='Rated speed (rpm)', font=('calibre'
    , 10), bg = '#cccccc')
503 tensione_label.grid(row=12, column=0, pady=5, sticky='W')
504 box7 = Entry(data, font=('calibre', 11), width=8)
505 box7.grid(row=13, column=0)
506 corrente_label = Label(data, text='Pole number', font=('calibre', 10),

```

```

        bg = '#cccccc')
507 corrente_label.grid(row=14, column=0, pady=5, sticky='W')
508 box8 = Entry(data, font=('calibre', 11), width=8)
509 box8.grid(row=15, column=0)
510 coll_label = Label(data, text='Rated frequency (Hz)', font=('calibre',
        10), bg = '#cccccc')
511 coll_label.grid(row=16, column=0, pady=5, sticky='W')
512 box9 = Entry(data, width=8)
513 box9.grid(row=17, column=0)
514 save_button = Button(data, text='Save', font=('calibre', 11), height
        =1, width=8, command=save1, state='disabled')
515 save_button.grid(row=18, column=5, padx=10, pady=10, sticky='SE')
516 #casella_v1=Entry(window, width=8) mostrava il valore delle tensioni,
        per capire se il pc comunicava correttamnte col wattmetro
517 #casella_v1.grid(row=2,column=4)
518 #salve_button['state'] = 'disabled'
519
520 # CREATE SETTING OF THE WATTMETER
521 setting = LabelFrame(window, text="Setting of the wattmeter", font=(
        'calibre', 12), bg='#cccccc')
522 setting.grid(row=0, column=1, padx=10, pady=10, rowspan=4, sticky='NW'
        )
523 lab_coupling = Label(setting, text='Rectifier setting', font=('calibre
        ', 10), bg='#cccccc')
524 lab_coupling.grid(row=0, column=0, padx=10, sticky='NW')
525 lab_wiring = Label(setting, text='Wiring type', font=('calibre', 10),
        bg='#cccccc')
526 lab_wiring.grid(row=0, column=1, padx=10, sticky='NW')
527 lab_display = Label(setting, text='Synchronization source', font=(
        'calibre', 10), bg='#cccccc')
528 lab_display.grid(row=0, column=2, padx=10, sticky='NW')
529 coupling_range = StringVar()
530 coupling_range.set('')
531 wiring_range = StringVar()
532 wiring_range.set('')
533 display_range = StringVar()
534 display_range.set('')
535 voltage_range = StringVar()
536 voltage_range.set('AUTO')
537 current_range = StringVar()
538 current_range.set('AUTO')
539 menu_coupling_range = OptionMenu(setting, coupling_range, "AC+DC", "AC
        ")
540 menu_coupling_range.grid(row=0, column=0, padx=10, pady=20, sticky='SW
        ')
541 menu_coupling_range.config(font=('calibre', 11), width=8)
542 menu_wiring_range = OptionMenu(setting, wiring_range, "Aron", "3
        Wattmetri")
543 menu_wiring_range.grid(row=0, column=1, padx=10, pady=20, sticky='SW')
544 menu_wiring_range.config( font=('calibre', 11), width=8)
545 menu_display_range = OptionMenu(setting, display_range, "Voltage", "
        Current")
546 menu_display_range.grid(row=0, column=2, padx=10, pady=20, sticky='SW'
        )
547 menu_display_range.config( font=('calibre', 11), width=8)
548 lab_Vport = Label(setting, text='Voltage range', font=('calibre', 10),
        bg='#cccccc')

```

```

549 lab_Vport.grid(row=1, column=0, padx=10, sticky='NW')
550 lab_Iport = Label(setting, text='Current range', font=('calibre', 10),
    bg='#cccccc')
551 lab_Iport.grid(row=1, column=1, padx=10, sticky='NW')
552 menu_voltage_range = OptionMenu(setting, voltage_range, 'AUTO', '15V',
    '30V', '60V', '150V', '300V', '600V', '1000V')
553 menu_voltage_range.grid(row=1, column=0, padx=10, pady=20, sticky='SW'
    )
554 menu_voltage_range.config( font=('calibre', 11), width=8)
555 menu_current_range = OptionMenu(setting, current_range, 'AUTO', '200mA
    ', '500mA', '1A', '2A', '5A', '10A', '20A', '50A')
556 menu_current_range.grid(row=1, column=1, padx=10, pady=20, sticky='SW'
    )
557 menu_current_range.config( font=('calibre', 11), width=8)
558 #casella1 = Entry(setting, width=8)
559 #casella1.grid(row=1, column=0, pady=20, padx=10, sticky='SW')
560 #casella1.insert(END, 1.0)
561 #casella22 = Entry(setting, width=8)
562 #casella22.grid(row=1, column=1, pady=20, padx=10, sticky='SW')
563 #casella22.insert(END, 1.0)
564 create_button = Button(setting, text='Zero-Adjust.', font=('calibre',
    11), height=1, width=8, command=Zeroadj)
565 create_button.grid(row=1, column=2, pady=20, padx=10, sticky='NW')
566 B_SETTT = Button(setting, text='Set', font=('calibre', 11), height=1,
    width=8, command=settings)
567 B_SETTT.grid(row=2, column=2, pady=20, padx=10, sticky='SW')
568
569 # FILE NAME
570 LB_filename = LabelFrame(window, text="File name", font=('calibre',
    12), bg='#cccccc')
571 LB_filename.grid(row=4, column=1, padx=10, columnspan=3, sticky='W',
    rowspan=1)
572 B_openfile = Button(LB_filename, text='open file', font=('calibre',
    11), height=1, width=8, command=open_file, state='disabled')
573 B_openfile.grid(row=0, column=6, padx=10, pady=5)
574 lab_type = Label(LB_filename, text='Test Type', font=('calibre', 10),
    bg='#cccccc')
575 lab_type.grid(row=0, column=0, padx=10, pady=5, sticky='N')
576 lab_insert = Label(LB_filename, text='File name without any format',
    font=('calibre', 10), bg='#cccccc')
577 lab_insert.grid(row=0, column=1, padx=10, pady=5, sticky='N')
578 prove_type = StringVar()
579 prove_type.set('')
580 menu_prove_type = OptionMenu(LB_filename, prove_type, "No Load", '
    Locked Rotor')
581 menu_prove_type.grid(row=0, column=0, padx=10, pady=25, sticky='S')
582 menu_prove_type.config(font=('calibre', 11), width=10)
583 casella3 = Entry(LB_filename, font=('calibre', 11), width=12)
584 casella3.grid(row=0, column=1, padx=10, pady=25, sticky='S')
585
586 # PROVE
587 LB_prove = LabelFrame(window, text="Test", font=('calibre', 12), bg='#
    ccccc')
588 LB_prove.grid(row=5, column=1, padx=10, columnspan=4, sticky='W')
589 B_salve = Button(LB_prove, text='Save', font=('calibre', 11), height
    =1, width=8, command=put_prove, state='disabled')
590 B_salve.grid(row=1, column=2, padx=10, pady=10, sticky='W')

```

```

591 lab_supply = Label(LB_prove, text='Supply', font=('calibre', 10), bg =
    '#cccccc')
592 lab_supply.grid(row=0, column=0, padx=10, sticky='WN')
593 lab_connection = Label(LB_prove, text='Winding connection', font=(
    'calibre', 10), bg = '#cccccc')
594 lab_connection.grid(row=0, column=1, padx=10, sticky='WN')
595 supply = StringVar()
596 supply.set('')
597 menu_supply = OptionMenu(LB_prove, supply, "SIN", 'PWM', "OQ")
598 menu_supply.grid(row=0, column=0, padx=10, pady=20, sticky='WS')
599 menu_supply.config(font=('calibre', 11), width=8)
600 connection = StringVar()
601 connection.set('')
602 menu_conn = OptionMenu(LB_prove, connection, "Y", 'D')
603 menu_conn.grid(row=0, column=1, padx=10, pady=20, sticky='WS')
604 menu_conn.config(font=('calibre', 11), width=8)
605 lab_frequency = Label(LB_prove, text='Frequency', font=('calibre', 10)
    , bg='#cccccc')
606 lab_frequency.grid(row=0, column=2, padx=10, sticky='N')
607 lab_notes = Label(LB_prove, text='Date and notes', font=('calibre',
    10), bg='#cccccc')
608 lab_notes.grid(row=1, column=0, padx=10, sticky='N')
609 casella4 = Entry(LB_prove, font=('calibre', 11), width=20)
610 casella4.grid(row=1, column=0, padx=10, pady=20, columnspan=2, sticky='
    S')
611 casella43 = Entry(LB_prove, font=('calibre', 11), width=10)
612 casella43.grid(row=0, column=2, padx=10, pady=20, sticky='S')
613
614 # RESISTENZA ALLA TEMPERATURA AMBIENTE
615 LB_resamb = LabelFrame(window, text="Winding resistance at the
    reference temperature", font=('calibre', 12), bg='#cccccc')
616 LB_resamb.grid(row=6, column=1, padx=10, columnspan=3, sticky='W')
617 B_save2 = Button(LB_resamb, text='Save', font=('calibre', 11), height
    =1, width=8, command=resist_amb, state='disabled')
618 B_save2.grid(row=0, column=4, padx=10, pady=10, sticky='S')
619 lab_tension = Label(LB_resamb, text='DC voltage (V)', font=('calibre',
    10), bg='#cccccc')
620 lab_tension.grid(row=0, column=0, padx=10, pady=5, sticky='N')
621 number_4 = Label(LB_resamb, text='(5)', font=('calibre', 10), bg='#
    cccccc', fg='red')
622 number_4.grid(row=0, column=4, padx=10, pady=10, sticky='NE')
623 lab_current = Label(LB_resamb, text='DC current (A)', font=('calibre',
    10), bg='#cccccc')
624 lab_current.grid(row=0, column=1, padx=10, pady=5, sticky='N')
625 lab_temperature = Label(LB_resamb, text='Temperature (°C)', font=(
    'calibre', 10), bg='#cccccc')
626 lab_temperature.grid(row=0, column=2, padx=10, pady=5, sticky='N')
627 casella5 = Entry(LB_resamb, font=('calibre', 11), width=8)
628 casella5.grid(row=0, column=0, padx=10, pady=30, sticky='S')
629 casella6 = Entry(LB_resamb, font=('calibre', 11), width=8)
630 casella6.grid(row=0, column=1, padx=10, pady=30, sticky='S')
631 casella7 = Entry(LB_resamb, font=('calibre', 11), width=8)
632 casella7.grid(row=0, column=2, padx=10, pady=30, sticky='S')
633
634 # RESISTENZA PRIMA PROVA
635 LB_resfirst = LabelFrame(window, text="Stator resistance at the
    beginning of the test", font=('calibre', 12), bg='#cccccc')

```



```

636 LB_resfirst.grid(row=7, column=1, padx=10, columnspan=2, sticky='W')
637 B_save3 = Button(LB_resfirst, text='Save', font=('calibre', 11),
        height=1, width=8, command=resist_first, state='disabled')
638 B_save3.grid(row=0, column=5, padx=10, pady=10)
639 lab_tension1 = Label(LB_resfirst, text='DC voltage (V)', font=(
        'calibre', 10), bg='#cccccc')
640 lab_tension1.grid(row=0, column=0, padx=10, pady=10, sticky='N')
641 lab_current1 = Label(LB_resfirst, text='DC current (A)', font=(
        'calibre', 10), bg='#cccccc')
642 lab_current1.grid(row=0, column=1, padx=10, pady=10, sticky='N')
643 casella51 = Entry(LB_resfirst, font=('calibre', 11), width=8)
644 casella51.grid(row=0, column=0, padx=10, pady=30, sticky='S')
645 casella61 = Entry(LB_resfirst, font=('calibre', 11), width=8)
646 casella61.grid(row=0, column=1, padx=10, pady=30, sticky='S')
647
648
649 #ACQUISIZIONE MISURE
650 LB_acq = LabelFrame(window, text="Acquisition", font=('calibre', 11),
        bg='#cccccc')
651 LB_acq.grid(row=0, column=4, padx=10, pady=10, rowspan=10, columnspan
        =4, sticky='NE')
652 lab_numsur = Label(LB_acq, text='Select numebr of consecutive
        measurements:', font=('calibre', 10), bg = '#cccccc')
653 lab_numsur.grid(row=0, column=0, padx=10, pady=10)
654 measure = StringVar()
655 measure.set('')
656 menu_measure = OptionMenu(LB_acq, measure, '3', "5", '10', "15", '20')
657 menu_measure.grid(row=1, column=0, padx=10, sticky='N')
658 menu_measure.config(font=('calibre', 11), width=5)
659 lab_num = Label(LB_acq, text='Meas.Nr:', font=('calibre', 10), bg='#
        ccccc')
660 lab_num.grid(row=2, column=0, padx=10, pady=10, sticky='N')
661 box_meas = Entry(LB_acq, font=('calibre', 11), width=8)
662 box_meas.grid(row=2, column=0, padx=10, pady=30, sticky='S')
663 #B_start = Button(LB_acq, text='Starttt', command=acquisition, height
        =2, width=10)
664 #B_start.grid(row = 0, column = 1, padx=1, pady=5, sticky='W')
665 B_stmisure = Button(LB_acq, text='Start recording', font=('calibre',
        11), command=acquisition, height=1, width=10, state='disabled')
666 B_stmisure.grid(row=1, column=1, padx=1, pady=5, sticky='W')
667 B_endmisure = Button(LB_acq, text='Save recording', font=('calibre',
        11), command=stop, height=1, width=10, state='disabled')
668 B_endmisure.grid(row=2, column=1, padx=1, pady=5, sticky='W')
669 lab_waitmeas = Label(LB_acq, text='Select wait time:', font=('calibre'
        , 10), bg='#cccccc')
670 lab_waitmeas.grid(row=3, column=0, padx=10, pady=10)
671 wait = StringVar()
672 wait.set('')
673 menu_wait = OptionMenu(LB_acq, measure, '0.1sec', "0.25sec", '0.5sec',
        "1sec")
674 menu_wait.grid(row=4, column=0, padx=10, sticky='N')
675 menu_wait.config(font=('calibre', 11), width=5)
676
677 # VALORI EFFICACI
678 LB_rms = LabelFrame(LB_acq, text="Measured values: Total", font=(
        'calibre', 10), bg='#cccccc')
679 LB_rms.grid(row=5, column=0, padx=10, pady=8, rowspan=1, columnspan=4,

```



```

        sticky='W')
680 lab_Vrms= Label(LB_rms, text='V rms (V):', font=('calibre', 10), bg =
        '#cccccc')
681 lab_Vrms.grid(row=0, column=0, padx=10, pady=10)
682 lab_Irms= Label(LB_rms, text='I rms (A):', font=('calibre', 10), bg =
        '#cccccc')
683 lab_Irms.grid(row=0, column=1, padx=10, pady=10)
684 lab_Prms= Label(LB_rms, text='P rms (kW or W):', font=('calibre', 10),
        bg = '#cccccc')
685 lab_Prms.grid(row=0, column=2, padx=10, pady=10)
686 box_Vrms= Entry(LB_rms, font=('calibre', 11), width=8)
687 box_Vrms.grid(row=1, column=0, padx=10, sticky='W')
688 box_Irms= Entry(LB_rms, font=('calibre', 11), width=8)
689 box_Irms.grid(row=1, column=1, padx=10, sticky='W')
690 box_Prms= Entry(LB_rms, font=('calibre', 11), width=8)
691 box_Prms.grid(row=1, column=2, padx=10, sticky='W')
692
693 # VALORI FONDAMENTALI
694 LB_fond = LabelFrame(LB_acq, text="Measured values: Fundamental (1st
        harm.)", font=('calibre', 12), bg='#cccccc')
695 LB_fond.grid(row=6, column=0, padx=10, pady=8, rowspan=1, columnspan
        =4, sticky='W')
696 lab_Vfond= Label(LB_fond, text='V rms (V):', font=('calibre', 10), bg=
        '#cccccc')
697 lab_Vfond.grid(row=0, column=0, padx=10, pady=10)
698 lab_Ifond= Label(LB_fond, text='I rms (A):', font=('calibre', 10), bg=
        '#cccccc')
699 lab_Ifond.grid(row=0, column=1, padx=10, pady=10)
700 lab_Pfond= Label(LB_fond, text='P rms (kW or W):', font=('calibre',
        10), bg='#cccccc')
701 lab_Pfond.grid(row=0, column=2, padx=10, pady=10)
702 box_Vfond= Entry(LB_fond, font=('calibre', 11), width=8)
703 box_Vfond.grid(row=1, column=0, padx=10, sticky='W')
704 box_Ifond= Entry(LB_fond, font=('calibre', 11), width=8)
705 box_Ifond.grid(row=1, column=1, padx=10, sticky='W')
706 box_Pfond= Entry(LB_fond, font=('calibre', 11), width=8)
707 box_Pfond.grid(row=1, column=2, padx=10, sticky='W')
708
709 # RESISTENZA LAST PROVA
710 LB_reslast = LabelFrame(window, text="Stator resistance at the end of
        the test and 30sec. after", font=('calibre', 12), bg='#cccccc')
711 LB_reslast.grid(row=6, column=4, padx=10, columnspan=3, sticky='W')
712 B_salve3 = Button(LB_reslast, text='Save', font=('calibre', 11),
        height=1, width=8, command=resit_last, state='disabled')
713 B_salve3.grid(row=0, column=5, padx=10, pady=10)
714 B_salve4 = Button(LB_reslast, text='Save', font=('calibre', 11),
        height=1, width=8, command=resit_last30, state='disabled')
715 B_salve4.grid(row=1, column=5, padx=10, pady=10)
716 lab_tension2 = Label(LB_reslast, text='DC voltage (V)', font=('calibre
        ', 10), bg='#cccccc')
717 lab_tension2.grid(row=0, column=0, padx=10, pady=10, sticky='N')
718 lab_current2 = Label(LB_reslast, text='DC current (A)', font=('calibre
        ', 10), bg='#cccccc')
719 lab_current2.grid(row=0, column=1, padx=10, pady=10, sticky='N')
720 casella53 = Entry(LB_reslast, width=8)
721 casella53.grid(row=0, column=0, padx=10, pady=30, sticky='S')
722 casella63 = Entry(LB_reslast, width=8)

```

```

723 casella63.grid(row=0, column=1, padx=10, pady=30, sticky='S')
724 lab_tension3 = Label(LB_reslast, text='DC voltage (V)', font=('calibre',
    , 10), bg='#cccccc')
725 lab_tension3.grid(row=1, column=0, padx=10, pady=10, sticky='N')
726 lab_current3 = Label(LB_reslast, text='DC current (A)', font=('calibre',
    , 10), bg='#cccccc')
727 lab_current3.grid(row=1, column=1, padx=10, pady=10, sticky='N')
728 casella54 = Entry(LB_reslast, font=('calibre', 11), width=8)
729 casella54.grid(row=1, column=0, padx=10, pady=30, sticky='S')
730 casella64 = Entry(LB_reslast, font=('calibre', 11), width=8)
731 casella64.grid(row=1, column=1, padx=10, pady=30, sticky='S')
732
733 '''
734 # VALORI RMS MEDI
735 LB_rmsm = LabelFrame(LB_acq, text=" Valori efficaci medi ", bg='#
    ccccc')
736 LB_rmsm.grid(row=7, column=0, padx=10, pady=8, rowspan=1, columnspan
    =4, sticky='W')
737 lab_Vrmsm= Label(LB_rmsm, text='V rms:', font=('calibre', 12), bg = '#
    ccccc')
738 lab_Vrmsm.grid(row=0, column=0, padx=10, pady=10)
739 lab_Irmsm= Label(LB_rmsm, text='I rms:', font=('calibre', 12), bg = '#
    ccccc')
740 lab_Irmsm.grid(row=0, column=1, padx=10, pady=10)
741 lab_Prmsm= Label(LB_rmsm, text='P rms:', font=('calibre', 12), bg = '#
    ccccc')
742 lab_Prmsm.grid(row=0, column=2, padx=10, pady=10)
743 box_Vrmsm= Entry(LB_rmsm, width=8)
744 box_Vrmsm.grid(row=1, column=0, padx=10, sticky='W')
745 box_Irmsm= Entry(LB_rmsm, width=8)
746 box_Irmsm.grid(row=1, column=1, padx=10, sticky='W')
747 box_Prmsm= Entry(LB_rmsm, width=8)
748 box_Prmsm.grid(row=1, column=2, padx=10, sticky='W')
749
750 # VALORI FONDAMENTALI MEDI
751 LB_fondm= LabelFrame(LB_acq, text="Valori medi di fondamentle", bg='#
    ccccc')
752 LB_fondm.grid(row=8, column=0, padx=10, pady=8, rowspan=1, columnspan
    =4, sticky='W')
753 lab_Vfondm= Label(LB_fondm, text='V rms:', font=('calibre', 12), bg =
    '#cccccc')
754 lab_Vfondm.grid(row=0, column=0, padx=10, pady=10)
755 lab_Ifondm= Label(LB_fondm, text='I rms:', font=('calibre', 12), bg =
    '#cccccc')
756 lab_Ifondm.grid(row=0, column=1, padx=10, pady=10)
757 lab_Pfondm= Label(LB_fondm, text='P rms:', font=('calibre', 12), bg =
    '#cccccc')
758 lab_Pfondm.grid(row=0, column=2, padx=10, pady=10)
759 box_Vfondm= Entry(LB_fondm, width=8)
760 box_Vfondm.grid(row=1, column=0, padx=10, sticky='W')
761 box_Ifondm= Entry(LB_fondm, width=8)
762 box_Ifondm.grid(row=1, column=1, padx=10, sticky='W')
763 box_Pfondm= Entry(LB_fondm, width=8)
764 box_Pfondm.grid(row=1, column=2, padx=10, sticky='W')
765 '''
766 create_button = Button(text='Save file', height=2, width=10, state='
    disabled') #aggiungere un comando per decidere dove salvare il

```

```

    tutto
767 create_button.grid(row=7, column=5, padx=1, pady=5, sticky='W')
768
769 quit_button = Button(text='Exit', command = window.quit, height=2,
    width=10)
770 quit_button.grid(row=7, column=6, padx=1, pady=5, sticky='W')
771
772 '''
773 #Create a canvas
774 canvas= Canvas(window, width= 100, height= 50,bg='#cccccc')
775 canvas.grid(row=0, column=6, padx=1, pady=5, sticky='W',rowspan=10)
776
777 #Load an image in the script
778 img= (Image.open("marchio_e_logotipo_politecnico_di_torino_full.png"))
779
780 #Resize the Image using resize method
781 resized_image = img.resize((200, 100), Image.ANTIALIAS)
782 new_image = ImageTk.PhotoImage(resized_image)
783
784 #Add image to the Canvas Items
785 canvas.create_image(10,10, anchor=NW, image=new_image)
786
787 '''
788 window.mainloop()

```

## 8.2 B. Elaboration tool Python codes

```

1 from tkinter import *
2 from tkinter import messagebox
3 from tkinter import filedialog
4 from tkinter import ttk
5 from math import *
6 from matplotlib.figure import Figure
7 from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
    NavigationToolbar2Tk)
8 import matplotlib
9 import matplotlib.pyplot as plt
10 import numpy as np
11 from scipy.optimize import curve_fit
12 from matplotlib import pyplot as plt
13 from fpdf import FPDF
14 import cmath
15
16 global indice_grap,Vnom,Inom,ColnNom,FreqNom,Npoli
17 indice_grap = 0 #indice che ci permetterà di dividere usare lo
    stesso tasto con differenti funzioni per plottare i dati
18 indice_inter = 0 #indice che ci permetterà di dividere usare lo
    stesso tasto con differenti funzioni per usare l'interpolazione
19
20 #create the main window
21
22 window = Tk() # create the main window
23 window.configure(bg = '#cccccc') # background color
24 window.attributes('-fullscreen', True)
25 window.title('Motor analysis')

```

```

26 #window.geometry('1280x800')
27 filename1='ciao'
28
29
30 '''
31 filewin = Toplevel(window)
32 button = Button(filewin, text="Do nothing button")
33 button.pack()
34 '''
35
36 def reset():      # create a function to reset the main program
37     global window, noload_button, lockrotor_button
38
39     window.destroy() # destroy the current window to create another
40     one
41     window = Tk()
42     window.configure(bg='#cccccc')
43     window.attributes('-fullscreen', True)
44     window.title('Motor analysis')
45
46     # NO LOAD BUTTON
47     #noload_button['state'] = 'normal'
48     noload_button = Button(text='No Load', command=noload, height=2,
49                             width=10)
50     noload_button.grid(row=0, column=0, padx=10, pady=5, sticky='W')
51
52     # LOCKED ROTOR BUTTON
53     lockrotor_button = Button(text='Locked Rotor', height=2, width=10,
54                               command=locked)
55     lockrotor_button.grid(row=0, column=1, padx=1, pady=5, sticky='W')
56
57     # Y=f(x) BUTTON
58     y_button = Button(text='Y=f(x)', height=2, width=10, command=
59                       caratteristic)
60     y_button.grid(row=0, column=2, padx=10, pady=5, sticky='W')
61
62     # RESET BUTTON
63     reset_button = Button(text='Reset', height=2, width=10, command=
64                           reset)
65     reset_button.grid(row=0, column=3, padx=1, pady=5, sticky='W')
66
67     # SPLIT FILE BUTTON
68     splitfile_button = Button(text='Split File', height=2, width=10,
69                               command=donothing)
70     splitfile_button.grid(row=0, column=4, padx=10, pady=5, sticky='W')
71
72     # EXIT BUTTON
73     quit_button = Button(text='Exit', command=window.quit, height=2,
74                          width=10)
75     quit_button.grid(row=0, column=5, padx=1, pady=5, sticky='W')
76
77 def noload():
78     noload_button['state'] = 'disabled'
79     global indice_grap, Vnom, Inom, ColnNom, FreqNom, Npoli, Pnom, Ngiri,
80     FreqPrv, Alim, CollPrv, VM, Im, PTM, RfaseInPrv, RFaseFiPrv, IO, Pferro, R0,
81     X0 #serve per far riuscire il reset, perch prima dava problemi
82     con il plot

```

```

73 indice_grap=0
74 indice_inter=0
75 global filename1
76
77 def PDF():
78     pdf = FPDF()
79
80     # Add a page
81     pdf.add_page()
82
83     # set style and size of font
84     # that you want in the pdf
85     pdf.set_font("helvetica", 'B', size=20)
86     pdf.image('polito_logo_2021_blu.jpg', 10, 1, 40)
87
88     # create a cell
89     pdf.cell(200, 10, txt="NO LOAD TEST", ln=1, align='C')
90
91     # add another cell
92     pdf.set_font("helvetica", '', size=10)
93     # pdf.set_text_color(169,169,169)
94     pdf.cell(200, 10, txt="Path and file name: " + window.filename
95 , ln=1, align='C')
96     pdf.cell(100, 10, txt='Motor type: ' + model)
97     pdf.cell(100, 10, txt='Company: ' + company, ln=1)
98     pdf.set_font("helvetica", 'B', size=10)
99     pdf.cell(100, 8, txt='Motor plate data', ln=1)
100    pdf.set_font("helvetica", '', size=10)
101    # pdf.set_text_color(169, 169, 169)
102    pdf.cell(50, 5, txt='Power [kW]: ' + str(Pnom))
103    pdf.cell(50, 5, txt='Voltage [V]: ' + str(Vnom))
104    pdf.cell(50, 5, txt='Current [A]: ' + str(Inom), ln=1)
105    pdf.cell(50, 5, txt='Connection: ' + ColnNom)
106    pdf.cell(50, 5, txt='Frequency [Hz]: ' + str(FreqNom))
107    pdf.cell(50, 5, txt='Speed [rpm]: ' + str(Ngiri), ln=1)
108    pdf.cell(50, 5, txt='Poles number: ' + str(Npoli), ln=1)
109    pdf.set_font("helvetica", 'B', size=10)
110    pdf.cell(100, 8, txt='Test conditions', ln=1)
111    pdf.set_font("helvetica", '', size=10)
112    pdf.cell(50, 5, txt='Test connection: ' + CollPrv)
113    pdf.cell(50, 5, txt='Test supply: ' + Alim, ln=1)
114    pdf.cell(50, 5, txt='Test frequency [Hz]: ' + str(FreqPrv))
115    pdf.cell(50, 5, txt='Reference voltage [V] at 50 [Hz]: ' + str(
Vnom), ln=1)
116    pdf.cell(50, 5, txt='Test data and notes: ' + note, ln=1)
117    pdf.set_font("helvetica", 'B', size=10)
118    pdf.cell(100, 8, txt='Test results', ln=1)
119    pdf.set_font("helvetica", '', size=10)
120    pdf.cell(25, 8, txt='Vline [V]', border=1, align='C')
121    pdf.cell(25, 8, txt='Eline [V]', border=1, align='C')
122    pdf.cell(25, 8, txt='Iline [A]', border=1, align='C')
123    pdf.cell(25, 8, txt='cosfi', border=1, align='C')
124    pdf.cell(25, 8, txt='Ptot [W]', border=1, align='C')
125    pdf.cell(25, 8, txt='Pcu [W]', border=1, align='C')
126    pdf.cell(25, 8, txt='Piron+mec [W]', border=1, align='C', ln=1)
127    for i in range(0, 10):
        pdf.cell(25, 5, txt=f'{round(VM[i],3)}', border=1, align='C')

```

```

128     pdf.cell(25, 5, txt=f'{round(EM[i],3)}', border=1, align='C')
129     pdf.cell(25, 5, txt=f'{round(Im[i],3)}', border=1, align='C')
130     pdf.cell(25, 5, txt=f'{round(CFM[i],3)}', border=1, align='C'
131 )
132     pdf.cell(25, 5, txt=f'{round(PTM[i],3)}', border=1, align='C'
133 )
134     pdf.cell(25, 5, txt=f'{round(PCM[i],3)}', border=1, align='C'
135 )
136     pdf.cell(25, 5, txt=f'{round(PNM[i],3)}', border=1, align='C'
137 , ln=1)
138     pdf.cell(0, 5, txt='Stator resistance at the reference
139 temperature of ' + temp + ' = ' + str(round((RfaseAmb), 3)) + ' [
140 Ohm]', ln=1)
141     pdf.cell(0, 5, txt='Stator resistance at start time = ' + str(
142 round((RfaseInPrv),3)) + ' [Ohm]', ln=1)
143     pdf.cell(0, 5, txt='Stator resistance at stop time = ' + f'
144 {"{: .2e}".format(RfaseFiPrv)}' + ' [Ohm]', ln=1)
145     pdf.set_font("helvetica", 'B', size=10)
146     pdf.cell(0, 8, txt='No load parameters at the test frequency and
147 at the reference voltage:', ln=1)
148     pdf.set_font("helvetica", '', size=10)
149     pdf.cell(0, 5, txt='No load current:' + f'{round(IO, 3)} ' + '[A]
150 ', ln=1)
151     pdf.cell(0, 5, txt='Iron losses:' + f'{round(Pferro, 3)} ' + '[W]
152 ', ln=1)
153     pdf.cell(0, 5, txt='Mechanical losses:' + f'{round((param2[0])
154 ,3)} ' + '[W]', ln=1)
155     pdf.cell(0, 5, txt='Equivalent iron losses resistance:' + f'{
156 round(R0, 3)} ' + '[Ohm]', ln=1)
157     pdf.cell(0, 5, txt='Stator reactance:' + f'{round(X0, 3)} ' + '[
158 Ohm]', ln=1)
159     pdf.set_font("helvetica", 'B', size=10)
160     pdf.cell(0, 8, txt='Coefficients of the fitting polinomial
161 equation:', ln=1)
162     pdf.set_font("helvetica", '', size=10)
163     pdf.cell(25, 8, txt='Grade', border=1, align='C')
164     pdf.cell(25, 8, txt='E = f(V)', border=1, align='C')
165     pdf.cell(25, 8, txt='IO = f(E)', border=1, align='C')
166     pdf.cell(25, 8, txt='P0 = f(E)', border=1, align='C', ln=1)
167     for i in range(0, 10):
168         pdf.cell(25, 5, txt=f'{i}', border=1, align='C')
169         pdf.cell(25, 5, txt=f'{"{: .3e}".format(pino[i])}', border=1,
170 align='C')
171         pdf.cell(25, 5, txt=f'{"{: .3e}".format(param1[i])}', border
172 =1, align='C')
173         pdf.cell(25, 5, txt=f'{"{: .3e}".format(param2[i])}', border
174 =1, align='C', ln=1)
175
176     # save the pdf with name .pdf
177     pdf.output("NO_LOAD.pdf")
178
179 def file():
180     window.filename = filedialog.askopenfilename( initialdir='/Users
181 /Federico/Desktop/LM_Tesi/programma2', title='select a file',
182 filetype=((('VTO files', '*.VTO'), ('VTO files', '*.VTO')))) # nel
183 primo VTO c'era CTO
184     # tex = open(window.filename, 'r')

```

```

164     mytext.insert(END, window.filename)
165     # tex.close()
166     global company, model, P_target, V_targa, I_targa, con_targa,
frequency_targa, poles_targa, speed_n, tipologia, frequency,
typology, mesure, note, VFM, EM, curre_mes, CFM, power_mes, PCM, PNM, temp,
RfaseAmb, Rsfase_first, pino, param1, EFM, RFaseFinePrv

167
168
169     mylist = [] # define an empty list
170
171     with open(window.filename, 'r') as file: # usare 'as file' per
rendere il programma globale per qualunque file aperto!
172
173         for line in file:
174
175             for word in line.split(','):
176                 mylist.append(word)
177
178     file.close()
179
180     # definiamo il numero delle misure
181
182     # define all the parametres based on the txt file
183
184     company = mylist[0]
185     model = mylist[1]
186     Pnom = float(mylist[2])
187     Vnom = float(mylist[3])
188     Inom = float(mylist[4])
189     ColnNom = mylist[5]
190     FreqNom = float(mylist[6])
191     Npoli = int(mylist[7])
192     Ngiri = float(mylist[8])
193     Alim = mylist[9]
194     FreqPrv = float(mylist[10])
195     CollPrv = mylist[11]
196     Prv = mylist[12]
197     note = mylist[13]
198     Vdc0 = float(mylist[14])
199     Idc0 = float(mylist[15])
200     Tamb = mylist[16]
201     Vdc1 = float(mylist[17])
202     Idc1 = float(mylist[18])
203     box1a.insert(END, company)
204     box2a.insert(END, model)
205     box3a.insert(END, Pnom)
206     box4a.insert(END, Vnom) # non posso convertire numerico se
voglio aggiungere la stringa dell'unità di misura
207     box5a.insert(END, Inom)
208     box6a.insert(END, CollPrv)
209     box7a.insert(END, FreqNom)
210     box8a.insert(END, Npoli)
211     box9a.insert(END, Ngiri)
212
213     # inserimento delle note
214     mynote.insert(END, note)
215

```



```

216 # completamento della secodna tabella
217 box1b.insert(END, Alim)
218 box2b.insert(END, Prv)
219 box3b.insert(END, CollPrv)
220 box4b.insert(END, FreqPrv)
221
222 #sresistance_temp = LabelFrame(window, bg='#d3d3d3', text='
Stator resistance at' + temp + '°C [Ohm]')
223 #sresistance_temp.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
224
225 sresistance_temp1 = LabelFrame(window, bg='#d3d3d3', text="
Stator resistance [Ohm] at \ntemperature [°C] of " + Tamb)
226 sresistance_temp1.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
227 box_t=Entry(sresistance_temp1, width=8, bg='#FFFFFF')
228 box_t.grid(row=0, column=0)
229
230 col = 1
231 #completamento della RESISTENZA DI STATORE nelle varie
232 if ColnNom == 'Y':
233     col = sqrt(3)
234 elif ColnNom == 'D':
235     col = 1
236 else:
237     messagebox.showwarning("warning", "Error with the
configuration parameter!")
238 IfaseNom = Inom * col / sqrt(3)
239 RfaseAmb = 1.5 * Vdc0 / Idc0 / pow(col, 2)
240 RfaseInPrv = 1.5 * Vdc1 / Idc1 / pow(col, 2)
241 rappFrq = FreqPrv / FreqNom
242 DeltaV = col * RfaseInPrv * IfaseNom
243 VPrvNom = DeltaV + rappFrq * (Vnom - DeltaV)
244
245 #completamento della terza tabella
246 #box4c.insert(END, RfaseAmb)
247 box_t.insert(END, round(RfaseAmb, 7))
248
249 # completamento della terza tabella
250 box4d.insert(END, round(RfaseInPrv, 7))
251
252 # completamento tabella misure
253 i=1
254 j=0
255 VM=[]
256 for tensione in mylist[20:-20:10]:
257     VM.append(float(tensione))
258     i=i+1
259 Ndati=int(i-1)
260 i=1
261 Im=[]
262 for current in mylist[23:-20:10]:
263     Im.append(float(current))
264     i=i+1
265 i = 1
266 PTM=[]
267 for power in mylist[26:-20:10]:

```



```

268     PTM.append(float(power))
269     i=i+1
270     i = 1
271     temp_mes = []
272     for tempo in mylist[28:-1:10]:
273         temp_mes.append(tempo)
274         i = i + 1
275     temp_mes.append(mylist[-1])
276
277
278     tp = []
279     k = 0
280     for tempo in temp_mes:
281         k = 0
282
283         for a in tempo.split('.'):
284
285             # print(tp)
286             if k == 0:
287                 hour = int(a)
288             elif k == 1:
289                 minut = int(a)
290             else:
291                 sec = int(a)
292                 k = k + 1
293             som = sec + 60 * minut + 3600 * hour
294             tp.append(som) # tipo int
295
296
297     #print(Ndati)
298
299     # completamento della tabella delle misurazioni con i calcoli
300     Rad3 = sqrt(3)
301     Rfase2 = 1.5 * float(mylist[-19]) / float(mylist[-16]) / pow(col
, 2)
302     Rfase3 = 1.5 * float(mylist[-9]) / float(mylist[-6]) / pow(col,
2)
303     RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])
304     #RappTP=2
305     print(RappTP)
306     RFaseFiPrv = RappTP * (Rfase2 - Rfase3) + Rfase3
307     RappRT = (RFaseFiPrv - RfaseInPrv) / (tp[-3] - tp[0])
308     print(RFaseFiPrv)
309     print(RappRT)
310
311     # completamento della terza tabella
312     box4e.insert(END, round(RFaseFiPrv, 7))
313
314     i=0
315     RF=[]
316     IFM=[]
317     PCM=[]
318     PNM=[]
319     CFM=[]
320     EM=[]
321
322     for i in range(Ndati):

```

```

323
324     RF.append(RfaseInPrv+(tp[i] - tp[0]) * RappRT)
325     VFM = VM[i] / col
326     IFM = Im[i] * col / Rad3
327     PCM.append(3 * RF[i] * pow(IFM, 2))
328     PNM.append(PTM[i] - PCM[i])
329     CFM.append(PTM[i] / Rad3 / VM[i] / Im[i])
330     SFM = (sqrt(1 - pow(CFM[i], 2)))
331     EFM = sqrt(pow((VFM - RF[i] * IFM * CFM[i]), 2) + pow((RF[i]
* IFM * SFM), 2))
332     #EFM = (sqrt(pow((VFM[i] - Rfase[i] * IFM[i] * CFM[i]), 2) +
pow((Rfase[i] * IFM[i] * SFM), 2)))
333     EM.append(EFM * col)
334
335
336
337     tree.insert(parent='',index='end',iid=i,text=i+1,values=(
round(VM[i],3),round(EM[i],3),round(Im[i],3),round(CFM[i],4),round
(PTM[i],3),round(PCM[i],3),round(PNM[i],3),))
338     scroll.configure(command=tree.yview)
339
340     # abilitare il comando emf
341     def emf():
342         plot_button['state'] = 'normal'
343         #fitting_button['state']='disabled'
344         q = 1
345         i = 0
346         for i in range(Ndati):
347             tree1.insert(parent='', index='end', iid=i, text=i + 1,
values=(round(VM[i], 3), round(EM[i], 3)))
348             io['state']='normal'
349             alfio['state'] = 'disabled'
350
351     alfio.configure(command=emf)
352
353     def i0():
354         plot_button['state'] = 'normal'
355         #fitting_button['state'] = 'disabled'
356         q = 2
357         i = 0
358         tree2 = ttk.Treeview(analysis)
359
360         tree2['columns'] = ('E.m.f [V]', 'IO [A]')
361
362         # Formate our columns
363         tree2.column('#0', width=25, minwidth=40)
364         tree2.column('E.m.f [V]', width=100, minwidth=40)
365         tree2.column('IO [A]', width=100, minwidth=40)
366         tree2.grid(row=0, column=0)
367
368         # Create Headings
369         tree2.heading('#0', text='N°')
370         tree2.heading('E.m.f [V]', text='E.m.f [V]')
371         tree2.heading('IO [A]', text='IO [A]')
372         for i in range(Ndati):
373             tree2.insert(parent='', index='end', iid=i, text=i + 1,
values=(round(EM[i], 3), round(Im[i], 3)))

```

```

374     po['state'] = 'normal'
375     io['state'] = 'disabled'
376
377     io.configure(command=io)
378
379     def P0():
380         plot_button['state'] = 'normal'
381         #fitting_button['state'] = 'disabled'
382         q = 3
383         i = 0
384         tree3 = ttk.Treeview(analysis)
385
386         tree3['columns'] = ('E.m.f [V]', 'PO [W]')
387
388         # Formate our columns
389         tree3.column('#0', width=25, minwidth=40)
390         tree3.column('E.m.f [V]', width=100, minwidth=40)
391         tree3.column('PO [W]', width=100, minwidth=40)
392         tree3.grid(row=0, column=0)
393
394         # Create Headings
395         tree3.heading('#0', text='N°')
396         tree3.heading('E.m.f [V]', text='E.m.f [V]')
397         tree3.heading('PO [W]', text='PO [W]')
398         for i in range(Ndati):
399             tree3.insert(parent='', index='end', iid=i, text=i + 1,
400 values=(round(EM[i], 3), round(PNM[i], 3)))
401             po['state'] = 'disabled'
402
403         po.configure(command=P0)
404
405     def grap():
406
407         # Data for plotting
408         #t = np.arange(0.0, 2.0, 0.01)
409         #s = 1 + np.sin(2 * np.pi * t)
410         global indice_grap
411         global indice_inter
412
413         if indice_grap == 0:
414             plt.grid(True)
415             foto = Figure(figsize=(6, 4))
416             anna = foto.add_subplot(111)
417             anna.plot(VM, EM, 'o', color='red')
418             # setting x and y axis range
419             plt.ylim(0, 450)
420             plt.xlim(0, 450)
421             anna.set(xlabel='V [V]', ylabel='E.m.f [V]', title='
Company: ' + company + ' Type: ' + model)
422             matteo = FigureCanvasTkAgg(foto, master=window)
423             matteo.draw()
424             matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
425             indice_inter = 0
426
427         if indice_grap == 1:

```

```

428     foto1 = Figure(figsize=(6, 4))
429     anna1 = foto1.add_subplot(111)
430     anna1.plot(EM, Im, 'o', color='red')
431     # setting x and y axis range
432     plt.ylim(0, 25)
433     plt.xlim(0, 450)
434     anna1.set(xlabel='E.m.f [V]', ylabel='IO [A]', title='
Company: ' + company + '                                Type: ' + model)
435     matteo = FigureCanvasTkAgg(foto1, master=window)
436     matteo.draw()
437     matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
438     indice_inter = 1
439
440     if indice_grap == 2:
441         f = Figure(figsize=(6, 4))
442         a = f.add_subplot(111)
443         a.plot(EM, PNM, 'o', color='red')
444         # setting x and y axis range
445         plt.ylim(0, 750)
446         plt.xlim(0, 450)
447         a.set(xlabel='E.m.f [V] ', ylabel='PO [W]', title='Company
: ' + company + '                                Type: ' + model)
448         matteo = FigureCanvasTkAgg(f, master=window)
449         matteo.draw()
450         matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
451         indice_inter = 2
452
453         indice_grap = indice_grap+1
454         #indice_inter = indice_inter + 1
455         plot_button['state'] = 'disabled'
456         #fitting_button['state'] = 'normal'
457
458
459
460     plot_button.configure(command=grap)
461     #indice_inter=0
462     def interpolation():
463         global param2, param1, pino
464
465         inserimento = grades.get()
466         print(inserimento)
467         coeff = []
468         for w in inserimento.split(','):
469             coeff.append(w)
470
471         lung = len(coeff)
472         grades_input.delete(0, 20) #cancella i gradi in ingresso ogni
volta che effettuiamo il fitting
473
474         global indice_inter
475
476         def test(x, a):
477             return a * x
478
479         def test3(x, a, b, c, d, e, f, g, h, i, l):

```

```

480
481     a1 = 0
482     b1 = 0
483     c1 = 0
484     d1 = 0
485     e1 = 0
486     f1 = 0
487     g1 = 0
488     h1 = 0
489     i1 = 0
490     l1 = 0
491     for indice in range(lung):
492
493         if int(coeff[indice]) == 0:
494             a1 = 1
495         if int(coeff[indice]) == 1:
496             b1 = 1
497         if int(coeff[indice]) == 2:
498             c1 = 1
499         if int(coeff[indice]) == 3:
500             d1 = 1
501         if int(coeff[indice]) == 4:
502             e1 = 1
503         if int(coeff[indice]) == 5:
504             f1 = 1
505         if int(coeff[indice]) == 6:
506             g1 = 1
507         if int(coeff[indice]) == 7:
508             h1 = 1
509         if int(coeff[indice]) == 8:
510             i1 = 1
511         if int(coeff[indice]) == 9:
512             l1 = 1
513
514     if a1 == 0:
515         a = 0
516     if b1 == 0:
517         b = 0
518     if c1 == 0:
519         c = 0
520     if d1 == 0:
521         d = 0
522     if e1 == 0:
523         e = 0
524     if f1 == 0:
525         f = 0
526     if g1 == 0:
527         g = 0
528     if h1 == 0:
529         h = 0
530     if i1 == 0:
531         i = 0
532     if l1 == 0:
533         l = 0
534
535     return a + b * x + c * np.power(x, 2) + d * np.power(x, 3)
+ e * np.power(x, 4) + f * np.power(x, 5) + g * np.power(x, 6) +

```

```

h * np.power(x, 7) + i * np.power(x, 8) + l * np.power(x, 9)
536
537     if indice_inter == 0:
538         x = np.array(VM)
539         y = np.array(EM)
540         #param, param_cov = curve_fit(test, x, y)
541         #ans = param[0] + x
542         param, _ = curve_fit(test, x, y)
543         ans = param[0] * x
544         #ans = param[0] + param[1] * x + param[2] * np.power(x, 2)
+ param[3] * np.power(x, 3) + param[4] * np.power(x, 4) + param
[5] * np.power(x, 5) + param[6] * np.power(x, 6) + param[7] * np.
power(x, 7) + param[8] * np.power(x, 8) + param[9] * np.power(x,
9)

545         foto = Figure(figsize=(6, 4))
546         anna = foto.add_subplot(111)
547         anna.plot(x, y, 'o', color='red', label="data")
548         anna.plot(x, ans, '--', color='blue', label="optimized
data")

549         plt.ylim(0, 450)
550         plt.xlim(0, 450)
551         anna.set(xlabel='V [V]', ylabel='E.m.f [V]', title='
Company: ' + company + '                                Type: ' + model)
552         intr = FigureCanvasTkAgg(foto, master=window)
553         intr.draw()
554         intr.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
555         pino = []
556         #fitting_button['state'] = 'disabled'
557         for i in range(0, 10):
558             if i==1:
559                 pino.append(param[0])
560             else:
561                 pino.append(0.0)
562
563                 treep.insert(parent='', index='end', iid=i, text=i,
values="{:.2e}".format(pino[i]))
564
565
566         if indice_inter == 1:
567             x = np.array(EM)
568             y = np.array(Im)
569             param1, _ = curve_fit(test3, x, y)
570             ans5 = param1[0] + param1[1] * x + param1[2] * np.power(x,
2) + param1[3] * np.power(x, 3) + param1[4] * np.power(x, 4) +
param1[5] * np.power(x, 5) + param1[6] * np.power(x, 6) + param1
[7] * np.power(x, 7) + param1[8] * np.power(x, 8) + param1[9] * np
.power(x, 9)
571             print("funcion coefficients:")
572             print(param1)
573
574             for index in range(0, 10):
575                 if param1[index] == 1.00000000e+00:
576                     param1[index] = 0
577
578             print('coefficienti con inserimento dei dati FILTRATI:')
579             print(param1)

```

```

580         x1 = np.linspace(0, max(x) + 100, 500)
581         y1 = param1[0] + param1[1] * x1 + param1[2] * np.power(x1,
2) + param1[3] * np.power(x1, 3) + param1[4] * np.power(x1, 4) +
param1[5] * np.power(x1, 5) + param1[6] * np.power(x1, 6) + param1
[7] * np.power(x1, 7) + param1[8] * np.power(x1, 8) + param1[9] *
np.power(x1, 9)
582         foto6 = Figure(figsize=(6, 4))
583         aaa = foto6.add_subplot(111)
584         aaa.plot(x, y, 'o', color='red', label="data")
585         aaa.plot(x1, y1, '--', color='blue', label="optimized data
")
586         plt.xlim([0, max(x) + 100])
587         plt.ylim([0, max(y) + 10])
588         #plt.ylim(0, 25)
589         #plt.xlim(0, 450)
590         aaa.set(xlabel='E.m.f [V]', ylabel='IO [A]', title = '
Company: ' + company + '                                Type: ' + model)
591         intr = FigureCanvasTkAgg(foto6, master=window)
592         intr.draw()
593         intr.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
594
595         #for record in treep.get.children():
596         #    treep.delete(record)
597
598         for i in range(0, 10):
599             treep.set(i, column='#2', value="{:.2e}".format(param1[
i]))
600
601         if indice_inter == 2:
602             x = np.array(EM)
603             y = np.array(PNM)
604             param2, _ = curve_fit(test3, x, y)
605             ans5 = param2[0] + param2[1] * x + param2[2] * np.power(x,
2) + param2[3] * np.power(x, 3) + param2[4] * np.power(x, 4) +
param2[5] * np.power(x, 5) + param2[6] * np.power(x, 6) + param2
[7] * np.power(x, 7) + param2[8] * np.power(x, 8) + param2[9] * np
.power(x, 9)
606             print("funcion coefficients:")
607             print(param2)
608
609             for index in range(0, 10):
610                 if param2[index] == 1.00000000e+00:
611                     param2[index] = 0
612
613             print('coefficienti con inserimento dei dati FILTRATI:')
614             print(param2)
615             x1 = np.linspace(0, max(x) + 100, 500)
616             y1 = param2[0] + param2[1] * x1 + param2[2] * np.power(x1,
2) + param2[3] * np.power(x1, 3) + param2[4] * np.power(x1, 4) +
param2[5] * np.power(x1, 5) + param2[6] * np.power(x1, 6) + param2
[7] * np.power(x1, 7) + param2[8] * np.power(x1, 8) + param2[9] *
np.power(x1, 9)
617             foto6 = Figure(figsize=(6, 4))
618             aaa = foto6.add_subplot(111)
619             aaa.plot(x, y, 'o', color='red', label="data")
620             aaa.plot(x1, y1, '--', color='blue', label="optimized data

```

```

")
621     plt.xlim([0, max(x) + 100])
622     plt.ylim([0, max(y) + 10])
623     # plt.ylim(0, 25)
624     # plt.xlim(0, 450)
625     aaa.set(xlabel='E.m.f [V]', ylabel='P0 [W]', title='
Company: ' + company + '                                Type: ' + model)
626     intr = FigureCanvasTkAgg(foto6, master=window)
627     intr.draw()
628     intr.get_tk_widget().grid(row=4, column=2, pady=10, padx
=10, columnspan=7, rowspan=9, sticky='N')
629     i=0
630     for inde in range(0, 10):
631         treep.set(inde, column='#3', value="{:.2e}".format(
param2[inde]))
632         #fitting_button['state'] = 'disabled'
633         Nload_button['state'] = 'normal'
634
635         #indice_inter=indice_inter+1
636
637     fitting_button.configure(command=interpolation)
638
639     def NLparameters():
640         global param2
641         casella1.insert(END, 'IO [A]')                # No load current
642         casella3.insert(END, 'Piron [W]')              # Iron losses
643         casella5.insert(END, 'Pmecc [W]')              # Mechanical losses
644         casella7.insert(END, 'R0 [Ohm]')               # Equivalent iron
losses resistance
645         casella9.insert(END, 'X0 [Ohm]')              # Stator reactance
646
647         E0 = VPrvNom * pino[1]
648         IO = 0
649         P0 = 0
650         j=0
651         for j in range(0, 10):
652
653             IO = (IO + param1[j] * np.power(E0, j))
654             print(param1[j])
655             print(IO)
656             P0 = (P0 + param2[j] * np.power(E0, j))
657
658         Pferro = P0 - param2[0]
659         Z = E0 * sqrt(3) / (col * col * IO)
660         g = Pferro / 3 / pow(E0 / col, 2)
661         y = 1 / Z
662         B = sqrt(np.power(y, 2) - np.power(g, 2))
663         R0 = 1 / g
664         X0 = 1 / B
665
666         casella2.insert(END, '{:.2e}'.format(IO))
667         casella4.insert(END, '{:.2e}'.format(Pferro))
668         casella6.insert(END, '{:.2e}'.format(param2[0]))
669         casella8.insert(END, '{:.2e}'.format(R0))
670         casella11.insert(END, '{:.2e}'.format(X0))
671         Nload_button['state'] = 'disabled'
672         report['state'] = 'normal'

```



```

673
674     # Create the .par file
675     filename1 = company + '.PAR'
676     file = open(filename1, 'w')
677     file.write('VT0' + '\n' + company + ',' + model + str(Vnom) +
        ',' + str(Inom) + ',' + str(FreqNom) + ',' + str(Npoli) + '\n' +
        ColnNom + '\n' + str(param1[0]))
678     for i in range(1, 10):
679         file.write(',') + str(param1[i]))
680     file.write('\n' + str(param2[0]))
681     for i in range(1,10):
682         file.write(',') + str(param2[i]))
683     file.write('\n' + CollPrv)
684     file.close()
685
686
687     Nload_button.configure(command=NLparameters)
688
689
690
691     #Create the MENU BAR
692
693     menubar = Menu(window)
694     filemenu = Menu(menubar, tearoff=0)
695     filemenu.add_command(label="Open file", command=file)
696     filemenu.add_command(label="Close", command=window.quit)
697     menubar.add_cascade(label="File", menu=filemenu)
698     window.config(menu=menubar)
699
700     # CREATE MOTOR PLATE DATA
701     data = LabelFrame(window, text="Motor plate data", bg='#d3d3d3')
702     data.grid(row=3, column=0, padx=10, pady=5, columnspan=6, sticky='W')
703     # INSIDE THE LABEL
704     box1 = Entry(data, width=8, bg='#E0E0E0')
705     box1.grid(row=0, column=0)
706     box1.insert(END, ' Company')
707     box2 = Entry(data, width=8, bg='#E0E0E0')
708     box2.grid(row=1, column=0)
709     box2.insert(END, ' Type')
710     box3 = Entry(data, width=8, bg='#E0E0E0')
711     box3.grid(row=2, column=0)
712     box3.insert(END, ' P rated')
713     box4 = Entry(data, width=8, bg='#E0E0E0')
714     box4.grid(row=3, column=0)
715     box4.insert(END, ' V rated')
716     box5 = Entry(data, width=8, bg='#E0E0E0')
717     box5.grid(row=4, column=0)
718     box5.insert(END, ' I rated')
719     box6 = Entry(data, width=8, bg='#E0E0E0')
720     box6.grid(row=5, column=0)
721     box6.insert(END, ' Connect.')
722     box7 = Entry(data, width=8, bg='#E0E0E0')
723     box7.grid(row=6, column=0)
724     box7.insert(END, ' Frequency')
725     box8 = Entry(data, width=8, bg='#E0E0E0')
726     box8.grid(row=7, column=0)
727     box8.insert(END, ' N° poles')

```

```

728 box9 = Entry(data, width=8, bg='#E0E0E0')
729 box9.grid(row=8, column=0)
730 box9.insert(END, ' Speed')
731 box1a = Entry(data, width=8, bg='FFFFFF')
732 box1a.grid(row=0, column=1)
733 box2a = Entry(data, width=8, bg='FFFFFF')
734 box2a.grid(row=1, column=1)
735 box3a = Entry(data, width=8, bg='FFFFFF')
736 box3a.grid(row=2, column=1)
737 box4a = Entry(data, width=8, bg='FFFFFF')
738 box4a.grid(row=3, column=1)
739 box5a = Entry(data, width=8, bg='FFFFFF')
740 box5a.grid(row=4, column=1)
741 box6a = Entry(data, width=8, bg='FFFFFF')
742 box6a.grid(row=5, column=1)
743 box7a = Entry(data, width=8, bg='FFFFFF')
744 box7a.grid(row=6, column=1)
745 box8a = Entry(data, width=8, bg='FFFFFF')
746 box8a.grid(row=7, column=1)
747 box9a = Entry(data, width=8, bg='FFFFFF')
748 box9a.grid(row=8, column=1)
749
750 # Create TEST CONDICTION label frame
751 condition = LabelFrame(window, text=" Test condiction", bg='#
d3d3d3')
752 condition.grid(row=4, column=0, padx=10, pady=5, columnspan=6,
sticky='W')
753
754 # Create the tab for the TEST CONDICTION
755 box1 = Entry(condition, width=8, bg='#E0E0E0')
756 box1.grid(row=0, column=0)
757 box1.insert(END, ' Type')
758 box2 = Entry(condition, width=8, bg='#E0E0E0')
759 box2.grid(row=1, column=0)
760 box2.insert(END, ' Connect.')
761 box3 = Entry(condition, width=8, bg='#E0E0E0')
762 box3.grid(row=2, column=0)
763 box3.insert(END, ' Supply')
764 box4 = Entry(condition, width=8, bg='#E0E0E0')
765 box4.grid(row=3, column=0)
766 box4.insert(END, ' Frequency')
767 box1b = Entry(condition, width=8, bg='FFFFFF')
768 box1b.grid(row=0, column=1)
769 box2b = Entry(condition, width=8, bg='FFFFFF')
770 box2b.grid(row=1, column=1)
771 box3b = Entry(condition, width=8, bg='FFFFFF')
772 box3b.grid(row=2, column=1)
773 box4b = Entry(condition, width=8, bg='FFFFFF')
774 box4b.grid(row=3, column=1)
775
776 sresistance_temp = LabelFrame(window, text="Stator resistance [0hm]
at \ntemperature [°C] of ", bg='#d3d3d3')
777 sresistance_temp.grid(row=5, column=0, padx=10, pady=5, columnspan
=6, sticky='W')
778
779 box4c = Entry(sresistance_temp, width=8, bg='FFFFFF')
780 box4c.grid(row=0, column=0)

```

```

781 box4c.insert(END, ' ')
782
783 sresistance_starttime = LabelFrame(window, text="Stator resistance
784 at\n the start time [Ohm]", bg='#d3d3d3')
785 sresistance_starttime.grid(row=6, column=0, padx=10, pady=5,
786 columnspan=6, sticky='W')
787
788 box4d = Entry(sresistance_starttime, width=8, bg='FFFFFF')
789 box4d.grid(row=0, column=0)
790 box4d.insert(END, ' ')
791
792 sresistance_stoptime = LabelFrame(window, text="Stator resistance
793 at \n the stop time [Ohm]", bg='#d3d3d3')
794 sresistance_stoptime.grid(row=7, column=0, padx=10, pady=5,
795 columnspan=6, sticky='W')
796
797 box4e = Entry(sresistance_stoptime, width=8, bg='FFFFFF')
798 box4e.grid(row=0, column=0)
799 box4e.insert(END, ' ')
800
801 #testo = Entry(text = 'text 1')
802 #testo.grid(row=12, column=0, padx=20, pady=5, columnspan=6, sticky
803 = 'W')
804
805 mynote = Entry(window, bg='F5F5F5', width=60, )
806 mynote.grid(row=2, column=0, padx=10, pady=5, columnspan=6, sticky=
807 'W')
808
809 # CASELLA PER LA DIRECTORY
810 mytext = Entry(window, bg='F5F5F5', width=60)
811 mytext.grid(row=1, column=0, padx=10, pady=8, columnspan=6, sticky=
812 'W')
813
814 # DATA
815
816 tabledata = Frame(window, bg='#d3d3d3', width=200, height=300)
817 tabledata.grid(row=3, column=2, pady=5, columnspan=10, rowspan=9,
818 sticky='WN')
819
820 # Scroll bar
821 scrollbar= Scrollbar(tabledata)
822 scrollbar.grid(row=0, column=9, rowspan=9, sticky='E')
823
824 tree= ttk.Treeview(tabledata, yscrollcommand=scrollbar.set)
825
826 # Define columns
827 tree['columns'] = ('V_line', 'E_line', 'I_line', 'cos-fi', 'P_tot', '
828 P_cu', 'P_mecc+fe')
829
830 # Formate our columns
831 tree.column('#0', width=25, minwidth=40)
832 tree.column('V_line', width=100, minwidth=40)
833 tree.column('E_line', width=100, minwidth=40)
834 tree.column('I_line', width=100, minwidth=40)
835 tree.column('cos-fi', width=100, minwidth=40)

```

```

829 tree.column('P_tot', width=100, minwidth=40)
830 tree.column('P_cu', width=100, minwidth=40)
831 tree.column('P_mecc+fe', width=100, minwidth=40)
832
833 # Create Headings
834 tree.heading('#0', text='N°')
835 tree.heading('V_line', text='V_line')
836 tree.heading('E_line', text='E_line')
837 tree.heading('I_line', text='I_line')
838 tree.heading('cos_fi', text='cos_fi')
839 tree.heading('P_tot', text='P_tot')
840 tree.heading('P_cu', text='P_cu')
841 tree.heading('P_mecc+fe', text='P_mecc+fe')
842
843 tree.grid(row=0, column=0, pady=5)
844
845 # CREATE THE CANVAS FOR THE GRAH
846 graph = Canvas(window, width=750, height=420)
847 graph.grid(row=4, column=2, pady=5, columnspan=7, rowspan=9, sticky
848           ='NW')
849
850 # Crete the Space for the analysis
851 analysis = Frame(window, bg='grey', width=200, height=200)
852 analysis.grid(row=3, column=14, padx=30, pady=5, columnspan=2,
853               rowspan=9, sticky='N')
854
855 # Scrrill bar
856 scrollbar = Scrollbar(analysis)
857 scrollbar.grid(row=0, column=9, sticky='E')
858
859 tree1 = ttk.Treeview(analysis, yscrollcommand=scrollbar.set)
860
861 # Define columns
862 tree1['columns'] = ('V [V]', 'E.m.f [V]')
863
864 # Formate our columns
865 tree1.column('#0', width=25, minwidth=40)
866 tree1.column('V [V]', width=100, minwidth=40)
867 tree1.column('E.m.f [V]', width=100, minwidth=40)
868
869 # Create Headings
870 tree1.heading('#0', text='N°')
871 tree1.heading('V [V]', text='V [V]')
872 tree1.heading('E.m.f [V]', text='E.m.f [V]')
873
874 tree1.grid(row=0, column=0, sticky='NW')
875
876 alfio = Button(text='E.m.f', height=2, width=10)
877 alfio.grid(row=3, column=16, padx=5, pady=5, sticky='NE')
878
879 io = Button(text='IO', command=window.quit, height=2, width=10,
880            state='disabled')
881 io.grid(row=3, column=16, padx=5, pady=55, sticky='NE')
882
883 po = Button(text='PO', command=window.quit, height=2, width=10,
884            state='disabled')

```

```

882 po.grid(row=3, column=16, padx=5, pady=105, sticky='NE')
883
884 plot_button = Button(text='Plot', height=2, width=10, state='
disabled')
885 plot_button.grid(row=3, column=8, padx=5, sticky='S')
886
887 fitting_button = Button(text='Fitting', height=2, width=10)
888 fitting_button.grid(row=3, column=7, padx=5, sticky='S')
889
890 print_button = Button(text='Print', height=2, width=10, state='
disabled')
891 print_button.grid(row=14, column=8, padx=5, sticky='S')
892
893 delete_button = Button(text='Delete points', height=2, width=10,
state='disabled')
894 delete_button.grid(row=14, column=7, padx=5, sticky='S')
895
896 grades = StringVar()
897 grades_input = Entry(window, textvariable = grades, font=('calibre',
, 10, 'normal'))
898 grades_input.grid(row=3, column=4, padx=5, columnspan=3, sticky='ES
')
899
900 grades_label = Label(window, text='Polynomial exponents \n
separeted by a ",", font=('calibre', 10, 'bold'), bg = '#cccccc')
901 grades_label.grid(row=3, column=3, columnspan=2, sticky='S')
902
903 delete_points = StringVar()
904 delete_input = Entry(window, textvariable=delete_points, font=('
calibre', 10, 'normal'))
905 delete_input.grid(row=14, column=4, padx=5, columnspan=3, sticky='
ES')
906
907 delete_label = Label(window, text='Delete points \n separeted by a
",", font=('calibre', 10, 'bold'),bg='#cccccc')
908 delete_label.grid(row=14, column=3, columnspan=2, sticky='S')
909
910 NL = LabelFrame(window, bg='#d3d3d3', text='No Load Parametr's')
911 NL.grid(row=4, column=14, padx=5, pady=1, columnspan=2, rowspan=9,
sticky='N')
912
913 casella1 = Entry(NL, width=8)
914 casella1.grid(row=0, column=0)
915 casella2 = Entry(NL, width=8, bg='yellow')
916 casella2.grid(row=0, column=1)
917 casella3 = Entry(NL, width=8)
918 casella3.grid(row=1, column=0)
919 casella4 = Entry(NL, width=8, bg='yellow')
920 casella4.grid(row=1, column=1)
921 casella5 = Entry(NL, width=8)
922 casella5.grid(row=2, column=0)
923 casella6 = Entry(NL, width=8, bg='yellow')
924 casella6.grid(row=2, column=1)
925 casella7 = Entry(NL, width=8)
926 casella7.grid(row=3, column=0)
927 casella8 = Entry(NL, width=8, bg='yellow')
928 casella8.grid(row=3, column=1)

```

```

929 casella9 = Entry(NL, width=8)
930 casella9.grid(row=4, column=0)
931 casella11 = Entry(NL, width=8, bg='yellow')
932 casella11.grid(row=4, column=1)
933
934
935 Nload_button = Button(text='NoLoad ', height=2, width=10, state =
    'disabled')
936 Nload_button.grid(row=4, column=16, padx=1, pady=10, sticky='NE')
937 report = Button( text='Report Print', height=2, width=10, command=
    PDF, state = 'disabled')
938 report.grid(row=4, column=16, padx=1, pady=60, sticky='NE')
939
940 polynomial_frame = LabelFrame(window, bg='#d3d3d3')
941 polynomial_frame.grid(row=5, column=14, padx=40, pady=50,
    columnspan=4, rowspan=10, sticky='N')
942
943 # Scrill bar
944 scrollbar = Scrollbar(polynomial_frame)
945 scrollbar.grid(row=0, column=1, rowspan=9, sticky='NE')
946
947 treep = ttk.Treeview(polynomial_frame, yscrollcommand=scrollbar.set)
948
949 # Define columns
950 treep['columns'] = ('Emf', 'IO', 'PO')
951
952 # Formate our columns
953 treep.column('#0', width=25, minwidth=40)
954 treep.column('Emf', width=100, minwidth=40)
955 treep.column('IO', width=100, minwidth=40)
956 treep.column('PO', width=100, minwidth=40)
957
958
959 # Create Headings
960 treep.heading('#0', text='Grade')
961 treep.heading('Emf', text='Emf')
962 treep.heading('IO', text='IO')
963 treep.heading('PO', text='PO')
964
965 treep.grid(row=0, column=0)
966
967 def locked():
968     global window, noload_button, lockrotor_button
969     global indice_grap, Vnom, Inom, ColnNom, FreqNom, Npoli
970
971     window.destroy() # destroy the current window to create another
        one
972     window = Tk()
973     window.configure(bg='#cccccc')
974     window.attributes('-fullscreen', True)
975     window.title('Motor analysis')
976
977     # NO LOAD BUTTON
978     # noload_button['state'] = 'normal'
979     noload_button = Button(text='No Load', command=noload, height=2,
        width=10, state='disabled')
980     noload_button.grid(row=0, column=0, padx=10, pady=5, sticky='W')

```

```

981
982 # LOCKED ROTOR BUTTON
983 lockrotor_button = Button(text='Locked Rotor', height=2, width=10,
984                             command=locked)
985 lockrotor_button.grid(row=0, column=1, padx=1, pady=5, sticky='W')
986
987 # Y=f(x) BUTTON
988 y_button = Button(text='Y=f(x)', height=2, width=10, command=
989                     caratteristic)
990 y_button.grid(row=0, column=2, padx=10, pady=5, sticky='W')
991
992 # RESET BUTTON
993 reset_button = Button(text='Reset', height=2, width=10, command=
994                       reset)
995 reset_button.grid(row=0, column=3, padx=1, pady=5, sticky='W')
996
997 # SPLIT FILE BUTTON
998 splitfile_button = Button(text='Split File', height=2, width=10,
999                             command=donothing)
1000 splitfile_button.grid(row=0, column=4, padx=10, pady=5, sticky='W')
1001
1002 # EXIT BUTTON
1003 quit_button = Button(text='Exit', command=window.quit, height=2,
1004                      width=10)
1005 quit_button.grid(row=0, column=5, padx=1, pady=5, sticky='W')
1006
1007 lockrotor_button['state'] = 'disabled'
1008 global indice_grap      #serve per far riuscire il reset, perch
1009 prima dava problemi con il plot
1010 indice_grap=0
1011
1012 def PDF():
1013     pdf = FPDF()
1014
1015     # Add a page
1016     pdf.add_page()
1017
1018     # set style and size of font
1019     # that you want in the pdf
1020     pdf.set_font("helvetica", 'B', size=20)
1021     pdf.image('polito_logo_2021_blu.jpg', 10, 1, 40)
1022
1023     # create a cell
1024     pdf.cell(200, 10, txt="LOCKED ROTOR TEST", ln=1, align='C')
1025
1026     # add another cell
1027     pdf.set_font("helvetica", '', size=10)
1028     # pdf.set_text_color(169,169,169)
1029     pdf.cell(200, 10, txt="Path and file name:  " + window.filename
1030 , ln=1, align='C')
1031     pdf.cell(100, 10, txt='Motor type: ' + model)
1032     pdf.cell(100, 10, txt='Company: ' + company, ln=1)
1033     pdf.set_font("helvetica", 'B', size=10)
1034
1035     pdf.cell(100, 8, txt='Motor plate data', ln=1)
1036     pdf.set_font("helvetica", '', size=10)
1037     # pdf.set_text_color(169, 169, 169)

```



```

1031 pdf.cell(50, 5, txt='Power [kW]: ' + str(P_target))
1032 pdf.cell(50, 5, txt='Voltage [V]: ' + str(V_targa))
1033 pdf.cell(50, 5, txt='Current [A]: ' + str(I_targa), ln=1)
1034 pdf.cell(50, 5, txt='Connection: ' + con_targa)
1035 pdf.cell(50, 5, txt='Frequency [Hz]: ' + str(frequency_targa))
1036 pdf.cell(50, 5, txt='Speed [rpm]: ' + str(speed_n), ln=1)
1037 pdf.cell(50, 5, txt='Poles number: ' + str(poles_targa), ln=1)
1038 pdf.set_font("helvetica", 'B', size=10)
1039
1040 pdf.cell(100, 8, txt='Test conditions', ln=1)
1041 pdf.set_font("helvetica", '', size=10)
1042 pdf.cell(50, 5, txt='Test connection: ' + typology)
1043 pdf.cell(50, 5, txt='Test supply: ' + tipologia, ln=1)
1044 pdf.cell(50, 5, txt='Test frequency [Hz]: ' + str(frequency))
1045 pdf.cell(50, 5, txt='Reference voltage [V] at 50 [Hz]: ' + str(
V_targa), ln=1)
1046 pdf.cell(50, 5, txt='Test data and notes: ' + note, ln=1)
1047
1048 pdf.set_font("helvetica", 'B', size=10)
1049 pdf.cell(100, 8, txt='Test results', ln=1)
1050 pdf.set_font("helvetica", '', size=10)
1051 pdf.cell(25, 8, txt='Vline [V]', border=1, align='C')
1052 pdf.cell(25, 8, txt='Eline [V]', border=1, align='C')
1053 pdf.cell(25, 8, txt='Iline [A]', border=1, align='C')
1054 pdf.cell(25, 8, txt='cosfi', border=1, align='C')
1055 pdf.cell(25, 8, txt='V at T°[V]', border=1, align='C')
1056 pdf.cell(25, 8, txt='P at T°[W]', border=1, align='C')
1057 pdf.cell(25, 8, txt='Cosfi at T°', border=1, align='C', ln=1)
1058 for i in range(0, 10):
1059     pdf.cell(25, 5, txt=f'{round(VFM[i],3)}', border=1, align='C'
)
1060     pdf.cell(25, 5, txt=f'{round(EM[i],3)}', border=1, align='C')
1061     pdf.cell(25, 5, txt=f'{round(curre_mes[i],3)}', border=1,
align='C')
1062     pdf.cell(25, 5, txt=f'{round(CFM[i],3)}', border=1, align='C'
)
1063     pdf.cell(25, 5, txt=f'{round(power_mes[i],3)}', border=1,
align='C')
1064     pdf.cell(25, 5, txt=f'{round(PCM[i],3)}', border=1, align='C'
)
1065     pdf.cell(25, 5, txt=f'{round(PNM[i],3)}', border=1, align='C'
, ln=1)
1066
1067 pdf.cell(0, 5, txt='Stator resistance at the reference
temperature of ' + temp + ' = ' + f'"{:.2e}".format(RFaseFinePrv)
}' + ' [Ohm]', ln=1)
1068 pdf.cell(0, 5, txt='Stator resistance at start time = ' + f'
"{:.2e}".format(RFaseFinePrv)}' + ' [Ohm]', ln=1)
1069 pdf.cell(0, 5, txt='Stator resistance at stop time = ' + f'
"{:.2e}".format(RFaseFinePrv)}' + ' [Ohm]', ln=1)
1070
1071 pdf.set_font("helvetica", 'B', size=10)
1072 pdf.cell(0, 8, txt='Locked rotor parameters at the reference
temperature T = ' + T_reference + ' °C', ln=1)
1073 pdf.set_font("helvetica", '', size=10)
1074 pdf.cell(0, 5, txt='No load current:', ln=1)
1075 pdf.cell(0, 5, txt='Iron losses:', ln=1)

```



```

1076 pdf.cell(0, 5, txt='Mechanical losses: ' + f'{round((param2[0]),3)} ' + '[W]', ln=1)
1077 pdf.cell(0, 5, txt='Equivalent iron losses resistance:', ln=1)
1078 pdf.cell(0, 5, txt='Stator reactance:', ln=1)
1079
1080 pdf.set_font("helvetica", 'B', size=10)
1081 pdf.cell(0, 8, txt='Coefficients of the fitting polynomial equation:', ln=1)
1082 pdf.set_font("helvetica", '', size=10)
1083 pdf.cell(25, 8, txt='Grade', border=1, align='C')
1084 pdf.cell(25, 8, txt='V = f(I)', border=1, align='C')
1085 pdf.cell(25, 8, txt='P = f(I)', border=1, align='C')
1086 for i in range(0, 10):
1087     pdf.cell(25, 5, txt=f'{i}', border=1, align='C')
1088     pdf.cell(25, 5, txt=f'{"{: .3e}".format(pino[i])}', border=1, align='C')
1089     pdf.cell(25, 5, txt=f'{"{: .3e}".format(param1[i])}', border=1, align='C')
1090
1091 # save the pdf with name .pdf
1092 pdf.output("Locked_rotor.pdf")
1093
1094
1095 def file():
1096     window.filename = filedialog.askopenfilename( initialdir='/Users/Federico/Desktop/LM_Tesi/programma2', title='select a file',
1097     filetypes=((('CT0 files', '*.CT0'), ('CT0 files', '*.CT0'))))
1098     # tex = open(window.filename, 'r')
1099     mytext.insert(END, window.filename)
1100     # tex.close()
1101     global T_reference, model
1102
1103     mylist = [] # define an empty list
1104
1105     with open(window.filename, 'r') as file: # usare 'as file' per rendere il programma globale per qualunque file aperto!
1106
1107         for line in file:
1108
1109             for word in line.split(','):
1110                 mylist.append(word)
1111
1112         file.close()
1113
1114     # definiamo il numero delle misure
1115
1116     # define all the parameters based on the txt file
1117
1118     company = mylist[0]
1119     model = mylist[1]
1120     P_target = float(mylist[2])
1121     V_targa = float(mylist[3])
1122     I_targa = float(mylist[4])
1123     con_targa = mylist[5]
1124     frequency_targa = float(mylist[6])
1125     poles_targa = float(mylist[7])

```

```

1126     speed_n = float(mylist[8])
1127     tipologia = mylist[9]
1128     frequency = float(mylist[10])
1129     typology = mylist[11]
1130     mesure = mylist[12]
1131     note = mylist[13]
1132     Vdc_temp = float(mylist[14])
1133     Idc_temp = float(mylist[15])
1134     Tamb = float(mylist[16])
1135     Vdc_first = float(mylist[17])
1136     Idc_first = float(mylist[18])
1137     box1a.insert(END, company)
1138     box2a.insert(END, model)
1139     box3a.insert(END, P_target)
1140     box4a.insert(END, V_targa)      # non posso convertire numerico se
voglio aggiungere la stringa dell'unità di misura
1141     box5a.insert(END, I_targa)
1142     box6a.insert(END, con_targa)
1143     box7a.insert(END, frequency_targa)
1144     box8a.insert(END, poles_targa)
1145     box9a.insert(END, speed_n)
1146
1147     # inserimento delle note
1148     mynote.insert(END, note)
1149
1150     # completamento della secodna tabella
1151     box1b.insert(END, mesure)
1152     box2b.insert(END, typology)
1153     box3b.insert(END, tipologia)
1154     box4b.insert(END, frequency)
1155
1156     temp=mylist[16]
1157     #sresistance_temp = LabelFrame(window, bg='#d3d3d3', text='
Stator resistance at\n' + temp + '°C [0hm]')
1158     #sresistance_temp.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
1159
1160     sresistance_temp1 = LabelFrame(window, bg='#d3d3d3', text='
Stator resistance [0hm] at \ntemperature [°C] of' + temp)
1161     sresistance_temp1.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
1162     box_t = Entry(sresistance_temp1, width=8, bg='#FFFFFF')
1163     box_t.grid(row=0, column=0)
1164     #print(temp)
1165
1166     #temperature window creation
1167     def get_temperature():
1168         T_reference = float(E_temperature.get()) # get the reference
temperature
1169         print(T_reference)
1170         W_temperature.destroy() # destroy the secondary window
1171
1172         col = 1
1173         # completamento della RESISTENZA DI STATORE nelle varie
1174         if con_targa == 'Y':
1175             col = sqrt(3)
1176         elif con_targa == 'D':

```

```

1177         col = 1
1178     else:
1179         messagebox.showwarning("warning", "Error with the
configuration parameter!")
1180     IfaseNom = I_targa * col / sqrt(3)
1181     RfaseAmb = 1.5 * Vdc_temp / ( Idc_temp) / pow(col, 2)
1182     Rsfase_first = 3 * Vdc_first / (2 * Idc_first) / pow(col, 2)
1183     rappFrq = frequency / frequency_targa
1184     DeltaV = col * Rsfase_first * IfaseNom
1185     VPrvNom = DeltaV + rappFrq * (V_targa - DeltaV)
1186
1187     # completamento della terza tabella
1188     #box4c.insert(END, RfaseAmb)
1189     box_t.insert(END, round(RfaseAmb, 7))
1190
1191     # completamento della terza tabella
1192     box4d.insert(END, round(Rsfase_first, 7))
1193
1194     # completamento della terza tabella
1195     #box4e.insert(END, mylist[17])
1196
1197     # completamento tabella misure
1198     i = 1
1199     tens_mes = []
1200     for tensione in mylist[20:-20:10]:
1201         tens_mes.append(float(tensione))
1202         i = i + 1
1203     Ndati = int(i - 1)
1204     i = 1
1205     V1 = []
1206     for tensione1 in mylist[19:-20:10]:
1207         V1.append(float(tensione1))
1208
1209     V2 = []
1210     for tensione in mylist[21:-20:10]:
1211         V2.append(float(tensione))
1212
1213     I1 = []
1214     for tensione1 in mylist[22:-20:10]:
1215         I1.append(float(tensione1))
1216
1217     I2 = []
1218     for tensione1 in mylist[24:-20:10]:
1219         I2.append(float(tensione1))
1220
1221     i = 1
1222     curre_mes = []
1223     for current in mylist[23:-20:10]:
1224         curre_mes.append(float(current))
1225         i = i + 1
1226     i = 1
1227     PM = []
1228     for power in mylist[26:-20:10]:
1229         PM.append(float(power))
1230
1231     P1 = []
1232     for power in mylist[25:-20:10]:

```

```

1233         P1.append(float(power))
1234
1235     P2 = []
1236     for power in mylist[27:-20:10]:
1237         P2.append(float(power))
1238
1239     i = 1
1240     temp_mes = []
1241     for tempo in mylist[28:-1:10]:
1242         temp_mes.append(tempo)
1243         i = i + 1
1244     temp_mes.append(mylist[-1])
1245
1246     tp = []
1247     k = 0
1248     for tempo in temp_mes:
1249         k = 0
1250
1251         for a in tempo.split('.'):
1252
1253             # print(tp)
1254             if k == 0:
1255                 hour = int(a)
1256             elif k == 1:
1257                 minut = int(a)
1258             else:
1259                 sec = int(a)
1260                 k = k + 1
1261             som = sec + 60 * minut + 3600 * hour
1262             tp.append(som) # tipo int
1263         # print(tp)
1264         # print(Ndati)
1265
1266         # completamento della tabella delle misurazioni con i calcoli
1267
1268         Rad3 = sqrt(3)
1269         Rfase_end = 1.5 * float(mylist[-19]) / float(mylist[-16]) /
1270         pow(col, 2)
1271         Rfase_end30 = 1.5 * float(mylist[-9]) / float(mylist[-6]) /
1272         pow(col, 2)
1273         RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])
1274
1275         RFaseFinePrv = RappTP * (Rfase_end - Rfase_end30) +
1276         Rfase_end30
1277         RappRT = (RFaseFinePrv - Rsfase_first) / (tp[Ndati] - tp[0])
1278         RFaseTrif = RfaseAmb * (T_reference + 235) / (Tamb + 235)
1279
1280         box4e.insert(END, round(RFaseFinePrv, 7))
1281
1282     i = 0
1283     Rfase = []
1284     PCM = []
1285     PNM = []
1286     CFM = []
1287     EM = []
1288     RappTemp = []
1289     PC1 = []

```

```

1287     PC2 = []
1288     VCM = []
1289     VC1 = []
1290     VC2 = []
1291     CFCM = []
1292
1293     for i in range(Ndati):
1294         Rfase.append(Rsfase_first + (tp[i] - tp[0]) * RappRT)
1295         RappTemp = RFaseTrif / Rfase[i]
1296         VFM = (tens_mes[i] / col)
1297         IFM = (curre_mes[i] * col / Rad3)
1298         VF1 = V1[i] / col
1299         IF1 = I1[i] * col / Rad3
1300         VF2 = V2[i] / col
1301         IF2 = I2[i] * col / Rad3
1302         ZCCM = (VFM / IFM)
1303         RCCM = PM[i] / (3 * pow(IFM, 2))
1304         XCCM = sqrt(pow(ZCCM, 2) - pow(RCCM, 2))
1305         CFM.append(RCCM / ZCCM)
1306         PCM.append(RappTemp * PM[i])
1307         #PCM.append(3 * Rfase[i] * pow(IFM[i], 2))
1308         RCM = RappTemp * RCCM
1309         ZCM = sqrt(pow(RCM, 2) + pow(XCCM, 2))
1310         PC1.append(RappTemp * P1[i])
1311         PC2.append(RappTemp * P2[i])
1312         VFCM = ZCM * IFM
1313         VCM.append(VFCM * col)
1314         VFC1 = ZCM * IF1
1315         VC1.append(VFC1 * col)
1316         VFC2 = ZCM * IF2
1317         VC2.append(VFC2 * col)
1318         CFCM.append(RCM / ZCM)
1319         #PNM.append(PM[i] - PCM[i])
1320         #CFM.append(PM[i] / Rad3 / tens_mes[i] / curre_mes[i])
1321         #SFM = (sqrt(1 - pow(CFM[i], 2)))
1322         #EFM = (sqrt(pow((VFM[i] - Rfase[i] * IFM[i] * CFM[i]), 2)
+ pow((Rfase[i] * IFM[i] * SFM), 2)))
1323         #EM.append(EFM * col)
1324
1325         tree.insert(parent='', index='end', iid=i, text=i + 1,
values=(round(curre_mes[i], 3), round(tens_mes[i], 3), round(PM[i]
], 3), round(CFM[i], 4), round(VCM[i], 3), round(PCM[i], 3), round
(CFCM[i], 3),))
1326         scroll.configure(command=tree.yview)
1327
1328     # print(IFM)
1329
1330     # abilitare il comando emf
1331     def Vlocked():
1332         plot_button['state'] = 'normal'
1333         fitting_button['state'] = 'disabled'
1334         indice_inter = 0
1335         q = 1
1336         i = 0
1337         for i in range(Ndati):
1338             tree1.insert(parent='', index='end', iid=i, text=i + 1,
values=(round(curre_mes[i], 3), round(VCM[i], 3)))

```

```

1339         io['state'] = 'normal'
1340         alfio['state'] = 'disabled'
1341
1342     alfio.configure(command=Vlocked)
1343
1344     def Plocked():
1345         plot_button['state'] = 'normal'
1346         fitting_button['state'] = 'disabled'
1347         q = 2
1348         i = 0
1349         tree2 = ttk.Treeview(analysis)
1350
1351         tree2['columns'] = ('I [A]', 'P [W]')
1352
1353         # Formate our columns
1354         tree2.column('#0', width=25, minwidth=40)
1355         tree2.column('I [A]', width=100, minwidth=40)
1356         tree2.column('P [W]', width=100, minwidth=40)
1357         tree2.grid(row=0, column=0)
1358
1359         # Create Headings
1360         tree2.heading('#0', text='N°')
1361         tree2.heading('I [A]', text='I [A]')
1362         tree2.heading('P [W]', text='P [W]')
1363         for i in range(Ndati):
1364             tree2.insert(parent='', index='end', iid=i, text=i + 1,
values=(round(curre_mes[i], 3), round(PCM[i], 3)))
1365             io['state'] = 'disabled'
1366
1367     io.configure(command=Plocked)
1368
1369     def grap():
1370
1371         # Data for plotting
1372         # t = np.arange(0.0, 2.0, 0.01)
1373         # s = 1 + np.sin(2 * np.pi * t)
1374         global indice_grap, indice_inter1
1375
1376         if indice_grap == 0:
1377             f = Figure(figsize=(6, 4))
1378             a = f.add_subplot(111)
1379             a.plot(curre_mes, VCM, 'o', color='red')
1380             # setting x and y axis range
1381             plt.ylim(0, 450)
1382             plt.xlim(0, 450)
1383             a.set(xlabel=' I (A)', ylabel=' V (V)',
1384                 title='Company: ' + company + ' Type: ' +
model)
1385             matteo = FigureCanvasTkAgg(f, master=window)
1386             matteo.draw()
1387             matteo.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')
1388             indice_inter1 = 0
1389
1390         if indice_grap == 1:
1391             f = Figure(figsize=(6, 4))
1392             a = f.add_subplot(111)

```

```

1393         a.plot(curre_mes, PCM, 'o', color='red')
1394         # setting x and y axis range
1395         plt.ylim(0, 25)
1396         plt.xlim(0, 450)
1397         a.set(xlabel=' I (A)', ylabel=' P (W)',
1398              title='Company: ' + company + '                                Type: ' +
model)
1399         matteo = FigureCanvasTkAgg(f, master=window)
1400         matteo.draw()
1401         matteo.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')
1402         indice_inter1 = 1
1403
1404         indice_grap = indice_grap + 1
1405         plot_button['state'] = 'disabled'
1406         fitting_button['state'] = 'normal'
1407
1408         plot_button.configure(command=grap)
1409
1410         def interpolation():
1411             global param2, parametri, parametri1, indice_inter,
parametri2
1412
1413             inserimento = grades.get()
1414             print(inserimento)
1415             coeff = []
1416             for w in inserimento.split(','):
1417                 coeff.append(w)
1418
1419             lung = len(coeff)
1420
1421             global param2,Vcc,Pcc,CFcc,Zcc,Zaux,Rcc,Xcc,Rs,Rr,Xs,Xr
1422
1423             def test(x, a, b):
1424                 return a + b * x
1425
1426             def test2(x, a):
1427                 return a * np.power(x, 2)
1428
1429             def test3(x, a, b, c, d, e, f, g, h, i, l):
1430
1431                 a1 = 0
1432                 b1 = 0
1433                 c1 = 0
1434                 d1 = 0
1435                 e1 = 0
1436                 f1 = 0
1437                 g1 = 0
1438                 h1 = 0
1439                 i1 = 0
1440                 l1 = 0
1441                 for indice in range(lung):
1442
1443                     if int(coeff[indice]) == 0:
1444                         a1 = 1
1445                     if int(coeff[indice]) == 1:
1446                         b1 = 1

```

```

1447         if int(coeff[indice]) == 2:
1448             c1 = 1
1449         if int(coeff[indice]) == 3:
1450             d1 = 1
1451         if int(coeff[indice]) == 4:
1452             e1 = 1
1453         if int(coeff[indice]) == 5:
1454             f1 = 1
1455         if int(coeff[indice]) == 6:
1456             g1 = 1
1457         if int(coeff[indice]) == 7:
1458             h1 = 1
1459         if int(coeff[indice]) == 8:
1460             i1 = 1
1461         if int(coeff[indice]) == 9:
1462             l1 = 1
1463
1464         if a1 == 0:
1465             a = 0
1466         if b1 == 0:
1467             b = 0
1468         if c1 == 0:
1469             c = 0
1470         if d1 == 0:
1471             d = 0
1472         if e1 == 0:
1473             e = 0
1474         if f1 == 0:
1475             f = 0
1476         if g1 == 0:
1477             g = 0
1478         if h1 == 0:
1479             h = 0
1480         if i1 == 0:
1481             i = 0
1482         if l1 == 0:
1483             l = 0
1484
1485         return a + b * x + c * np.power(x, 2) + d * np.power(x,
1486             3) + e * np.power(x, 4) + f * np.power(x, 5) + g * np.power(x, 6)
1487             + h * np.power(x, 7) + i * np.power(x, 8) + l * np.power(x, 9)
1488
1489         if indice_inter1 == 0:
1490             x = np.array(curre_mes)
1491             y = np.array(VCM)
1492             # param, param_cov = curve_fit(test, x, y)
1493             # ans = param[0] + x
1494             param, _ = curve_fit(test, x, y)
1495             ans = param[0] + param[1] * x
1496             # ans = param[0] + param[1] * x + param[2] * np.power(x
1497             , 2) + param[3] * np.power(x, 3) + param[4] * np.power(x, 4) +
1498             param[5] * np.power(x, 5) + param[6] * np.power(x, 6) + param[7] *
1499             np.power(x, 7) + param[8] * np.power(x, 8) + param[9] * np.power(
1500             x, 9)
1501
1502         foto = Figure(figsize=(6, 4))
1503         anna = foto.add_subplot(111)
1504         anna.plot(x, y, 'o', color='red', label="data")

```



```

1498         anna.plot(x, ans, '--', color='blue', label="optimized
data")
1499         plt.ylim(0, 450)
1500         plt.xlim(0, 450)
1501         anna.set(xlabel=' I [A]', ylabel=' V [V]', title='
Company: ' + company + ' Type: ' + model)
1502         intr = FigureCanvasTkAgg(foto, master=window)
1503         intr.draw()
1504         intr.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')
1505         parametri = []
1506         # fitting_button['state'] = 'disabled'
1507         for i in range(0, 10):
1508             print(i)
1509             if i == 0:
1510                 parametri.append(param[0])
1511
1512             if i == 1:
1513                 parametri.append(param[1])
1514
1515             if i > 1:
1516                 parametri.append(0.0)
1517
1518             treep.insert(parent='', index='end', iid=i, text=i,
values="{:.2e}".format(parametri[i]))
1519             print(parametri)
1520             print(Ndati)
1521
1522         parametri1 = []
1523
1524         if indice_inter1 == 1:
1525             x = np.array(curre_mes)
1526             y = np.array(PCM)
1527             parametri1, _ = curve_fit(test2, x, y)
1528             #ans5 = parametri1[0] + parametri1[1] * x + parametri1
[2] * np.power(x, 2) + parametri1[3] * np.power(x, 3) + parametri1
[4] * np.power(x, 4) + parametri1[5] * np.power(x, 5) + parametri1
[6] * np.power(x, 6) + parametri1[7] * np.power(x, 7) +
parametri1[8] * np.power(x, 8) + parametri1[9] * np.power(x, 9)
1529             ans7 = parametri1[0] * np.power(x, 2)
1530             x1 = np.linspace(0, max(x) + 10, 500)
1531             y1 = parametri1[0] * np.power(x1, 2)
1532             foto8 = Figure(figsize=(6, 4))
1533             aaa = foto8.add_subplot(111)
1534             aaa.plot(x, y, 'o', color='red', label="data")
1535             aaa.plot(x1, y1, '--', color='blue', label="optimized
data")
1536             plt.xlim([0, max(x) + 10])
1537             plt.ylim([0, max(y) + 10])
1538             # plt.ylim(0, 25)
1539             # plt.xlim(0, 450)
1540             aaa.set(xlabel=' I [A]', ylabel=' P [W]', title='
Company: ' + company + ' Type: ' + model)
1541             intr = FigureCanvasTkAgg(foto8, master=window)
1542             intr.draw()
1543             intr.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')

```

```

1544     parametri2 = []
1545     # fitting_button['state'] = 'disabled'
1546     for i in range(0, 10):
1547         if i == 2:
1548             parametri2.append(parametri1[0])
1549         else:
1550             parametri2.append(0.0)
1551
1552         #treep.insert(parent='', index='end', iid=i, text=i,
values="{: .3e}".format(parametri2[i]))
1553         treep.set(i, column='#2', value="{: .2e}".format(
parametri2[i]))
1554
1555     if indice_inter1 == 5:
1556         x = np.array(curre_mes)
1557         y = np.array(PCM)
1558         parametri1, _ = curve_fit(test2, x, y)
1559         #ans5 = parametri1[0] + parametri1[1] * x + parametri1
[2] * np.power(x, 2) + parametri1[3] * np.power(x, 3) + parametri1
[4] * np.power(x, 4) + parametri1[5] * np.power(x, 5) + parametri1
[6] * np.power(x, 6) + parametri1[7] * np.power(x, 7) +
parametri1[8] * np.power(x, 8) + parametri1[9] * np.power(x, 9)
1560         ans5 = parametri1[0] * np.power(x, 2)
1561         print("funzione coefficients zio:")
1562         print(parametri1)
1563
1564         for index in range(0, 10):
1565             print(index)
1566             if parametri1[index] == 1.00000000e+00:
1567                 parametri1[index] = 0
1568
1569         print('coefficienti con inserimento dei dati FILTRATI:
')
1570         print(parametri1)
1571         x1 = np.linspace(0, max(x) + 100, 500)
1572         y1 = parametri1[0] + parametri1[1] * x1 + parametri1[2]
* np.power(x1, 2) + parametri1[3] * np.power(x1, 3) + parametri1
[4] * np.power(x1, 4) + parametri1[5] * np.power(x1, 5) +
parametri1[6] * np.power(x1, 6) + parametri1[7] * np.power(x1, 7)
+ parametri1[8] * np.power(x1, 8) + parametri1[9] * np.power(x1,
9)
1573         foto6 = Figure(figsize=(6, 4))
1574         aaa = foto6.add_subplot(111)
1575         aaa.plot(x, y, 'o', color='red', label="data")
1576         aaa.plot(x1, y1, '--', color='blue', label="optimized
data")
1577         plt.xlim([0, max(x) + 100])
1578         plt.ylim([0, max(y) + 10])
1579         # plt.ylim(0, 25)
1580         # plt.xlim(0, 450)
1581         aaa.set(xlabel=' I [A]', ylabel=' P [W]', title='
Company: ' + company + ' Type: ' + model)
1582         intr = FigureCanvasTkAgg(foto6, master=window)
1583         intr.draw()
1584         intr.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')
1585

```

```

1586
1587         for i in range(0, 10):
1588             treep.set(i, column='#2', value="{:.3e}".format(
1589                 parametri1[i]))
1590
1591     fitting_button.configure(command=interpolation)
1592
1593     def LRparameters():
1594
1595         casella1.insert(END, 'Vlr [V]')
1596         casella3.insert(END, 'Plr [W]')
1597         casella5.insert(END, 'cos_fi [Ohm]')
1598         casella7.insert(END, 'Zlr [Ohm]')
1599         casella9.insert(END, 'Rs [Ohm]')
1600         casella12.insert(END, 'Rr [Ohm]')
1601         casella14.insert(END, 'Xs [Ohm]')
1602         casella16.insert(END, 'Xr [Ohm]')
1603
1604         Vcc = 0
1605         Pcc = 0
1606         Icc = I_targa
1607
1608         for i in range(0, 10):
1609             Vcc = Vcc + parametri[i] * pow(Icc, i)
1610             Pcc = Pcc + parametri2[i] * pow(Icc, i)
1611
1612         CFcc = Pcc / sqrt(3) / Vcc / Icc
1613         Zcc = Vcc * sqrt(3) / Icc / col / col
1614         Zaux = (Vcc - parametri[0]) * sqrt(3) / Icc / col / col
1615         Rcc = Zcc * CFcc
1616         Xcc = sqrt(pow(Zcc, 2) - pow(Rcc, 2))
1617         Rs = RFaseTrif
1618         print(Rs)
1619         Rr = Rcc - Rs
1620         Xs = sqrt(pow(Zaux, 2) - pow(Rcc, 2)) / 2
1621         Xr = Xcc - Xs
1622
1623         casella2.insert(END, '{:.2e}'.format(Vcc))
1624         casella4.insert(END, '{:.2e}'.format(Pcc))
1625         casella6.insert(END, '{:.2e}'.format(CFcc))
1626         casella8.insert(END, '{:.2e}'.format(Zcc))
1627         casella11.insert(END, '{:.2e}'.format(Rs))
1628         casella13.insert(END, '{:.2e}'.format(Rr))
1629         casella15.insert(END, '{:.2e}'.format(Xs))
1630         casella17.insert(END, '{:.2e}'.format(Xr))
1631
1632
1633         LR_button['state'] = 'disabled'
1634         report['state'] = 'normal'
1635
1636         # Create the .par file
1637         filename2 = company + '.PAR'
1638         file = open(filename2, 'a')
1639         file.write('\n' + 'CT0' + '\n' + str(RFaseTrif) + ',' +
1640             str(T_reference) + '\n' + str(parametri[0]))
1641         for i in range(1, 10):

```

```

1641         file.write(',') + str(parametri[i]))
1642     file.write('\n' + str(parametri2[0]))
1643     for i in range(1, 10):
1644         file.write(',') + str(parametri2[i]))
1645     file.write('\n'+ con_targa)
1646     file.close()
1647
1648
1649
1650     LR_button.configure(command=LRparameters)
1651
1652     W_temperature=Toplevel()
1653     W_temperature.title('Reference temeperature for the stator
1654     windings')
1655     L_temperaturewind=Label(W_temperature, text='Reference
1656     temeperature T*?')
1657     L_temperaturewind.grid(row=0, column=0, padx=15, pady=5)
1658     B_temperatureok=Button(W_temperature, text='Ok', height=2, width
1659     =10, command=get_temperature)
1660     B_temperatureok.grid(row=0, column=1, padx=15, pady=5)
1661     B_temperaturedelete = Button(W_temperature, text='Delete',
1662     height=2, width=10, command=W_temperature.destroy)
1663     B_temperaturedelete.grid(row=0, column=2, padx=15, pady=5)
1664     E_temperature=Entry(W_temperature)
1665     E_temperature.grid(row=1, column=0, padx=15, pady=5)
1666
1667
1668
1669     #Crete the MENU BAR
1670
1671     menubar = Menu(window)
1672     filemenu = Menu(menubar, tearoff=0)
1673     filemenu.add_command(label="Open file", command=file)
1674     filemenu.add_command(label="Close", command=window.quit)
1675     menubar.add_cascade(label="File", menu=filemenu)
1676     window.config(menu=menubar)
1677
1678
1679     # CREATE MOTOR PLATE DATA
1680     data = LabelFrame(window, text="Motor plate data", bg='#d3d3d3')
1681     data.grid(row=3, column=0, padx=10, pady=5, columnspan=6, sticky='W'
1682     )
1683
1684     # INSIDE THE LABEL
1685     box1 = Entry(data, width=8, bg='#E0E0E0')
1686     box1.grid(row=0, column=0)
1687     box1.insert(END, ' Company')
1688     box2 = Entry(data, width=8, bg='#E0E0E0')
1689     box2.grid(row=1, column=0)
1690     box2.insert(END, ' Type')
1691     box3 = Entry(data, width=8, bg='#E0E0E0')
1692     box3.grid(row=2, column=0)
1693     box3.insert(END, ' P rated')
1694     box4 = Entry(data, width=8, bg='#E0E0E0')
1695     box4.grid(row=3, column=0)
1696     box4.insert(END, ' V rated')
1697     box5 = Entry(data, width=8, bg='#E0E0E0')
1698     box5.grid(row=4, column=0)

```

```

1693 box5.insert(END, ' I rated')
1694 box6 = Entry(data, width=8, bg='#E0E0E0')
1695 box6.grid(row=5, column=0)
1696 box6.insert(END, ' Connect.')
```

```

1697 box7 = Entry(data, width=8, bg='#E0E0E0')
1698 box7.grid(row=6, column=0)
1699 box7.insert(END, ' Frequency')
```

```

1700 box8 = Entry(data, width=8, bg='#E0E0E0')
1701 box8.grid(row=7, column=0)
1702 box8.insert(END, ' N° poles')
```

```

1703 box9 = Entry(data, width=8, bg='#E0E0E0')
1704 box9.grid(row=8, column=0)
1705 box9.insert(END, ' Speed')
```

```

1706 box1a = Entry(data, width=8, bg='#FFFFFF')
```

```

1707 box1a.grid(row=0, column=1)
```

```

1708 box2a = Entry(data, width=8, bg='#FFFFFF')
```

```

1709 box2a.grid(row=1, column=1)
```

```

1710 box3a = Entry(data, width=8, bg='#FFFFFF')
```

```

1711 box3a.grid(row=2, column=1)
```

```

1712 box4a = Entry(data, width=8, bg='#FFFFFF')
```

```

1713 box4a.grid(row=3, column=1)
```

```

1714 box5a = Entry(data, width=8, bg='#FFFFFF')
```

```

1715 box5a.grid(row=4, column=1)
```

```

1716 box6a = Entry(data, width=8, bg='#FFFFFF')
```

```

1717 box6a.grid(row=5, column=1)
```

```

1718 box7a = Entry(data, width=8, bg='#FFFFFF')
```

```

1719 box7a.grid(row=6, column=1)
```

```

1720 box8a = Entry(data, width=8, bg='#FFFFFF')
```

```

1721 box8a.grid(row=7, column=1)
```

```

1722 box9a = Entry(data, width=8, bg='#FFFFFF')
```

```

1723 box9a.grid(row=8, column=1)
```

```

1724
```

```

1725 # Create TEST CONDICTION label frame
```

```

1726 condition = LabelFrame(window, text=" Test condiction", bg='#
d3d3d3')
```

```

1727 condition.grid(row=4, column=0, padx=10, pady=5, columnspan=6,
sticky='W')
```

```

1728
```

```

1729 # Create the tab for the TEST CONDICTION
```

```

1730 box1 = Entry(condition, width=8, bg='#E0E0E0')
```

```

1731 box1.grid(row=0, column=0)
```

```

1732 box1.insert(END, ' Type')
```

```

1733 box2 = Entry(condition, width=8, bg='#E0E0E0')
```

```

1734 box2.grid(row=1, column=0)
```

```

1735 box2.insert(END, ' Connect.')
```

```

1736 box3 = Entry(condition, width=8, bg='#E0E0E0')
```

```

1737 box3.grid(row=2, column=0)
```

```

1738 box3.insert(END, ' Supply')
```

```

1739 box4 = Entry(condition, width=8, bg='#E0E0E0')
```

```

1740 box4.grid(row=3, column=0)
```

```

1741 box4.insert(END, ' Frequency')
```

```

1742 box1b = Entry(condition, width=8, bg='#FFFFFF')
```

```

1743 box1b.grid(row=0, column=1)
```

```

1744 box2b = Entry(condition, width=8, bg='#FFFFFF')
```

```

1745 box2b.grid(row=1, column=1)
```

```

1746 box3b = Entry(condition, width=8, bg='#FFFFFF')
```

```

1747 box3b.grid(row=2, column=1)
```

```

1748 box4b = Entry(condition, width=8, bg='FFFFFF')
1749 box4b.grid(row=3, column=1)
1750
1751 sresistance_temp = LabelFrame(window, bg='d3d3d3', text='Stator
resistance [Ohm] at \ntemperature [°C] of ')
1752 sresistance_temp.grid(row=5, column=0, padx=10, pady=5, columnspan
=6, sticky='W')
1753
1754 box4c = Entry(sresistance_temp, width=8, bg='FFFFFF')
1755 box4c.grid(row=0, column=0)
1756 box4c.insert(END, ' ')
1757
1758 sresistance_starttime = LabelFrame(window, text="Stator resistance
at\n the start time [Ohm]", bg='d3d3d3')
1759 sresistance_starttime.grid(row=6, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
1760
1761 box4d = Entry(sresistance_starttime, width=8, bg='FFFFFF')
1762 box4d.grid(row=0, column=0)
1763 box4d.insert(END, ' ')
1764
1765 sresistance_stoptime = LabelFrame(window, text="Stator resistance
at \n the start time [Ohm]", bg='d3d3d3')
1766 sresistance_stoptime.grid(row=7, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
1767
1768 box4e = Entry(sresistance_stoptime, width=8, bg='FFFFFF')
1769 box4e.grid(row=0, column=0)
1770 box4e.insert(END, ' ')
1771
1772 #testo = Entry(text='text 1')
1773 #testo.grid(row=12, column=0, padx=20, pady=5, columnspan=6, sticky
='W')
1774
1775 mynote = Entry(window, bg='F5F5F5', width=60, )
1776 mynote.grid(row=2, column=0, padx=10, pady=5, columnspan=6, sticky=
'W')
1777
1778 # CASELLA PER LA DIRECTORY
1779 mytext = Entry(window, bg='F5F5F5', width=60)
1780 mytext.grid(row=1, column=0, padx=10, pady=8, columnspan=6, sticky=
'W')
1781
1782
1783 # DATA
1784
1785 tabledata = Frame(window, bg='d3d3d3', width=200, height=300)
1786 tabledata.grid(row=3, column=2, pady=5, columnspan=10, rowspan=9,
sticky='WN')
1787
1788 # Scroll bar
1789 scrollbar = Scrollbar(tabledata)
1790 scrollbar.grid(row=0, column=9, rowspan=9, sticky='E')
1791
1792 tree = ttk.Treeview(tabledata, yscrollcommand=scrollbar.set)
1793
1794

```

```

1795 # Define columns
1796 tree['columns'] = ('I_line', 'V_line', 'P', 'cos-fi', 'V at T*', 'P
    at T*', 'cos-fi at T*')
1797
1798 # Formate our columns
1799 tree.column('#0', width=25, minwidth=40)
1800 tree.column('I_line', width=100, minwidth=40)
1801 tree.column('V_line', width=100, minwidth=40)
1802 tree.column('P', width=100, minwidth=40)
1803 tree.column('cos-fi', width=100, minwidth=40)
1804 tree.column('V at T*', width=100, minwidth=40)
1805 tree.column('P at T*', width=100, minwidth=40)
1806 tree.column('cos-fi at T*', width=100, minwidth=40)
1807
1808 # Create Headings
1809 tree.heading('#0', text='N°')
1810 tree.heading('I_line', text='I_line')
1811 tree.heading('V_line', text='V_line')
1812 tree.heading('P', text='P')
1813 tree.heading('cos-fi', text='cos-fi')
1814 tree.heading('V at T*', text='V at T*')
1815 tree.heading('P at T*', text='P at T*')
1816 tree.heading('cos-fi at T*', text='cos-fi at T*')
1817
1818 tree.grid(row=0, column=0, pady=5)
1819
1820 # CREATE THE CANVAS FOR THE GRAH
1821 graph = Canvas(window, width=750, height=420)
1822 graph.grid(row=4, column=2, pady=5, columnspan=7, rowspan=9, sticky
    ='NW')
1823
1824 # Crete the Space for the analysis
1825 analysis = Frame(window, bg='grey', width=200, height=200)
1826 analysis.grid(row=3, column=14, padx=30, pady=5, columnspan=2,
    rowspan=9, sticky='N')
1827
1828 # Scroll bar
1829 scrollbar = Scrollbar(analysis)
1830 scrollbar.grid(row=0, column=9, sticky='E')
1831
1832 tree1 = ttk.Treeview(analysis, yscrollcommand=scrollbar.set)
1833
1834 # Define columns
1835 tree1['columns'] = ('I [A]', 'V [V]')
1836
1837 # Formate our columns
1838 tree1.column('#0', width=25, minwidth=40)
1839 tree1.column('I [A]', width=100, minwidth=40)
1840 tree1.column('V [V]', width=100, minwidth=40)
1841
1842
1843 # Create Headings
1844 tree1.heading('#0', text='N°')
1845 tree1.heading('I [A]', text='I [A]')
1846 tree1.heading('V [V]', text='V [V]')
1847
1848 tree1.grid(row=0, column=0, sticky='NW')

```

```

1849
1850 alfio = Button(text='V locked rotor', height=2, width=10)
1851 alfio.grid(row=3, column=16, padx=5, pady=5, sticky='NE')
1852
1853 io = Button(text='P locked rotor', command=window.quit, height=2,
1854 width=10, state='disabled')
1855 io.grid(row=3, column=16, padx=5, pady=55, sticky='NE')
1856
1857 plot_button = Button(text='Plot', height=2, width=10, state='
disabled')
1858 plot_button.grid(row=3, column=8, padx=5, sticky='S')
1859
1860 fitting_button = Button(text='Fitting', height=2, width=10, state='
disabled')
1861 fitting_button.grid(row=3, column=7, padx=5, sticky='S')
1862
1863 print_button = Button(text='Print', height=2, width=10, state='
disabled')
1864 print_button.grid(row=14, column=8, padx=5, sticky='S')
1865
1866 delete_button = Button(text='Delete points', height=2, width=10,
state='disabled')
1867 delete_button.grid(row=14, column=7, padx=5, sticky='S')
1868
1869 grades = StringVar()
1870 grades_input = Entry(window, textvariable = grades, font=('calibre',
,10,'normal'))
1871 grades_input.grid(row=3, column=4, padx=5, columnspan=3, sticky='ES
')
1872
1873 grades_label = Label(window, text='Polynomial exponents \n
separeted by a ",", font=('calibre', 10, 'bold'), bg = '#cccccc')
1874 grades_label.grid(row=3, column=3, columnspan=2, sticky='S')
1875
1876 delete_points = StringVar()
1877 delete_input = Entry(window, textvariable=grades, font=('calibre',
10, 'normal'))
1878 delete_input.grid(row=14, column=4, padx=5, columnspan=3, sticky='
ES')
1879
1880 delete_label = Label(window, text='Delete points \n separeted by a
",", font=('calibre', 10, 'bold'), bg='#cccccc')
1881 delete_label.grid(row=14, column=3, columnspan=2, sticky='S')
1882
1883 NL = LabelFrame(window, bg='#d3d3d3', text='Locked Rotor Parametr's
')
1884 NL.grid(row=3, column=14, padx=5, pady=250, columnspan=8, rowspan
=9, sticky='N')
1885 casella1 = Entry(NL, width=8)
1886 casella1.grid(row=0, column=0)
1887 casella2 = Entry(NL, width=8, bg='yellow')
1888 casella2.grid(row=0, column=1)
1889 casella3 = Entry(NL, width=8)
1890 casella3.grid(row=1, column=0)
1891 casella4 = Entry(NL, width=8, bg='yellow')
1892 casella4.grid(row=1, column=1)

```



```

1893 casella5 = Entry(NL, width=8)
1894 casella5.grid(row=2, column=0)
1895 casella6 = Entry(NL, width=8, bg='yellow')
1896 casella6.grid(row=2, column=1)
1897 casella7 = Entry(NL, width=8)
1898 casella7.grid(row=3, column=0)
1899 casella8 = Entry(NL, width=8, bg='yellow')
1900 casella8.grid(row=3, column=1)
1901 casella9 = Entry(NL, width=8)
1902 casella9.grid(row=4, column=0)
1903 casella11 = Entry(NL, width=8, bg='yellow')
1904 casella11.grid(row=4, column=1)
1905 casella12 = Entry(NL, width=8)
1906 casella12.grid(row=5, column=0)
1907 casella13 = Entry(NL, width=8, bg='yellow')
1908 casella13.grid(row=5, column=1)
1909 casella14 = Entry(NL, width=8)
1910 casella14.grid(row=6, column=0)
1911 casella15 = Entry(NL, width=8, bg='yellow')
1912 casella15.grid(row=6, column=1)
1913 casella16 = Entry(NL, width=8)
1914 casella16.grid(row=7, column=0)
1915 casella17 = Entry(NL, width=8, bg='yellow')
1916 casella17.grid(row=7, column=1)
1917
1918
1919 LR_button = Button(text='Locked rotor ', height=2, width=10)
1920 LR_button.grid(row=4, column=16, padx=1, pady=10, sticky='NE')
1921 report = Button( text='Report Print', command=PDF, height=2, width
1922 =10, state='normal')
1923 report.grid(row=4, column=16, padx=1, pady=60, sticky='NE')
1924
1925 polynomial_frame = LabelFrame(window, bg='#d3d3d3')
1926 polynomial_frame.grid(row=5, column=14, padx=40, pady=50,
1927 columnspan=4, rowspan=10, sticky='N')
1928
1929 # Scroll bar
1930 scrollbar = Scrollbar(polynomial_frame)
1931 scrollbar.grid(row=0, column=1, rowspan=9, sticky='NE')
1932
1933 treep = ttk.Treeview(polynomial_frame, yscrollcommand=scroll.set)
1934
1935 # Define columns
1936 treep['columns'] = ('V locked', 'P locked')
1937
1938 # Formate our columns
1939 treep.column('#0', width=25, minwidth=40)
1940 treep.column('V locked', width=100, minwidth=40)
1941 treep.column('P locked', width=100, minwidth=40)
1942
1943 # Create Headings
1944 treep.heading('#0', text='Grade')
1945 treep.heading('V locked', text='V locked')
1946 treep.heading('P locked', text='P locked')
1947
1948 treep.grid(row=0, column=0)

```

# Bibliography

- [1] Jillian Ambrose. <https://www.theguardian.com/environment/2021/apr/29/electric-vehicles-on-worlds-roads-expected-to-increase-to-145m-by-2030>. [Online; accessed 05-October-2021].
- [2] Deloitte. <https://www2.deloitte.com/us/en/insights/focus/future-of-mobility/electric-vehicle-trends-2030.html>. [Online; accessed 29-October-2021].
- [3] <https://towardsdatascience.com/why-python-is-not-the-programming-language-of-the-future-30ddc5339b66>. [Online; accessed 4-November-2021].
- [4] <https://spectrum.ieee.org/top-programming-languages-2021>. [Online; accessed 27-October-2021].
- [5] “IEEE Standard Test Procedure for Polyphase Induction Motors and Generators”. In: *IEEE Std 112-2017 (Revision of IEEE Std 112-2004)* (2018), pp. 1–115. DOI: 10.1109/IEEESTD.2018.8291810.
- [6] Hioki. *PW 3337 Instruction Manual*.
- [7] Hioki. *PW 3337 Communication Manual*. [https://www.hioki.com/global/support/download/software/versionup/detail/id\\_196](https://www.hioki.com/global/support/download/software/versionup/detail/id_196). [Online; accessed 27-October-2021].
- [8] Python. *Tkinter—Python interface to Tcl/Tk*. <https://docs.python.org/3/library/tkinter.html>. [Online; accessed 12-September-2021].
- [9] Electrotest. <http://www.elettrotestspa.it/en/prodotti.php>. [Online; accessed 27-October-2021].