Politecnico di Torino

Master's Degree in Electrical Engineering

Master's Degree Thesis

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

List of Figures

List of Tables

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].

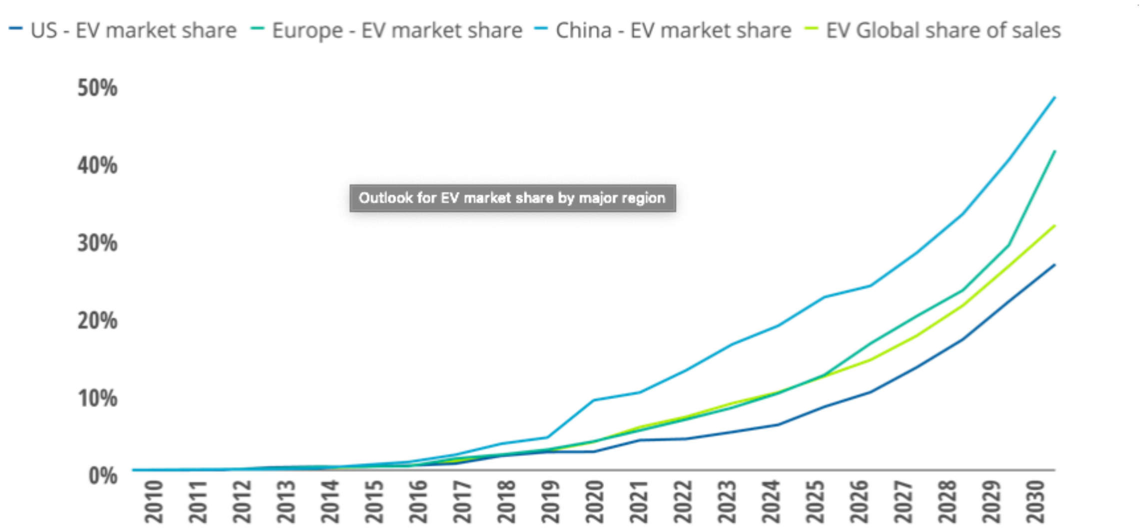


Figure 1.1: EV market trend[2].

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.

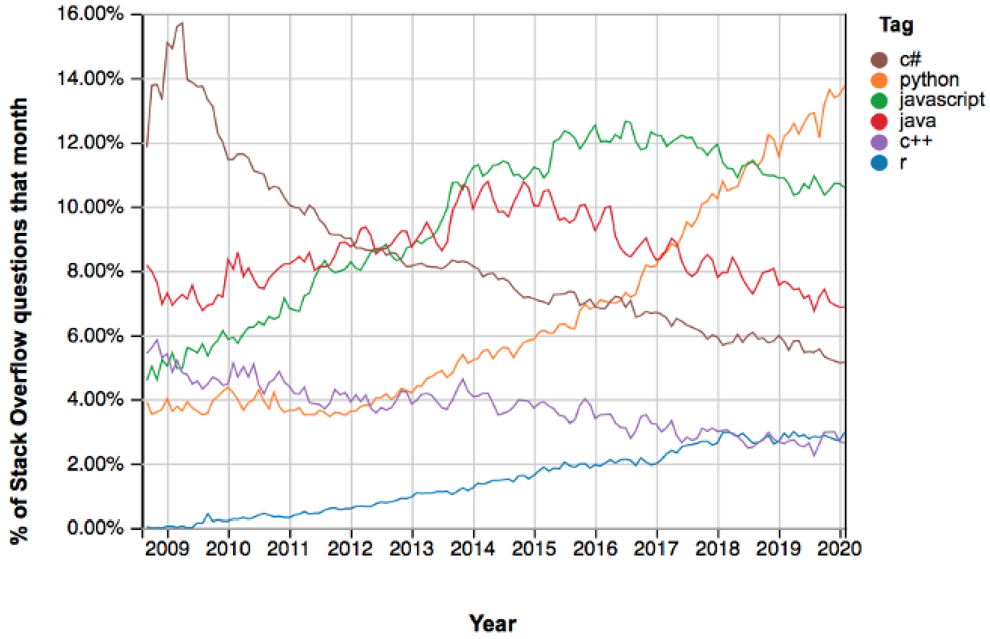


Figure 1.2: Programming Language trends [3].

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.

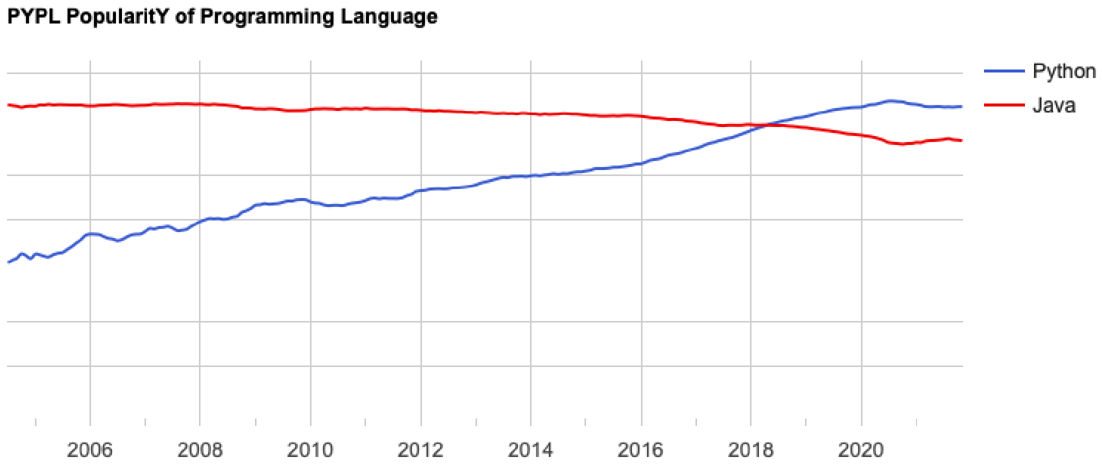


Figure 1.3: Comparison between Python and Java.

Moreover, the IEEE Spectrum places Python in the fist 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:

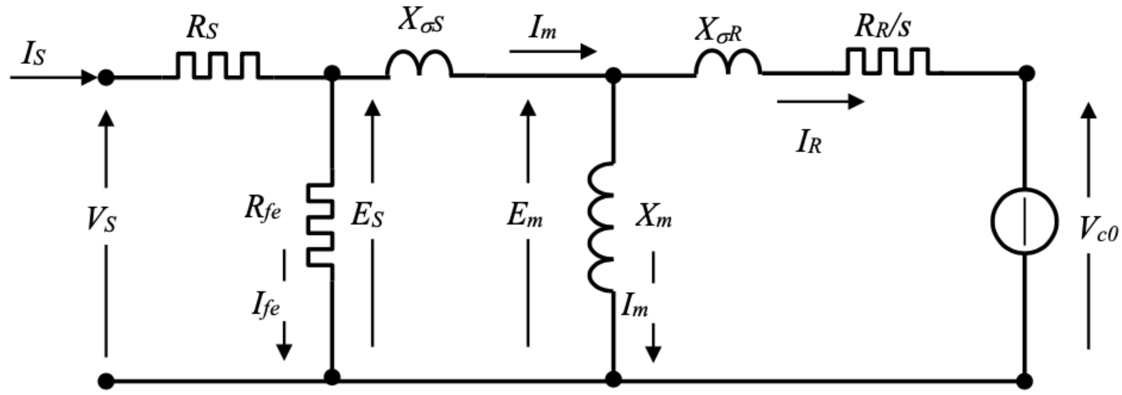


Figure 2.1: equivalent circuit single-phase of induction machines.

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 *PW*337 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].

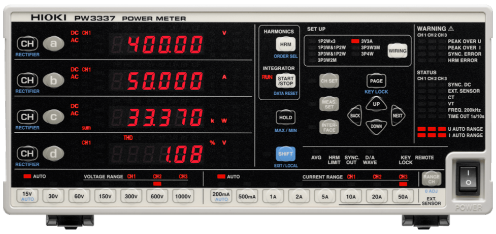


Figure 3.1: *PW*337 Front panel of the instrument

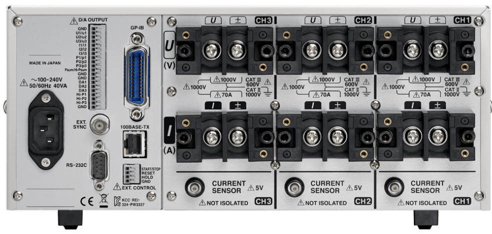


Figure 3.2: *PW*337 Rear panel of the instrument

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.

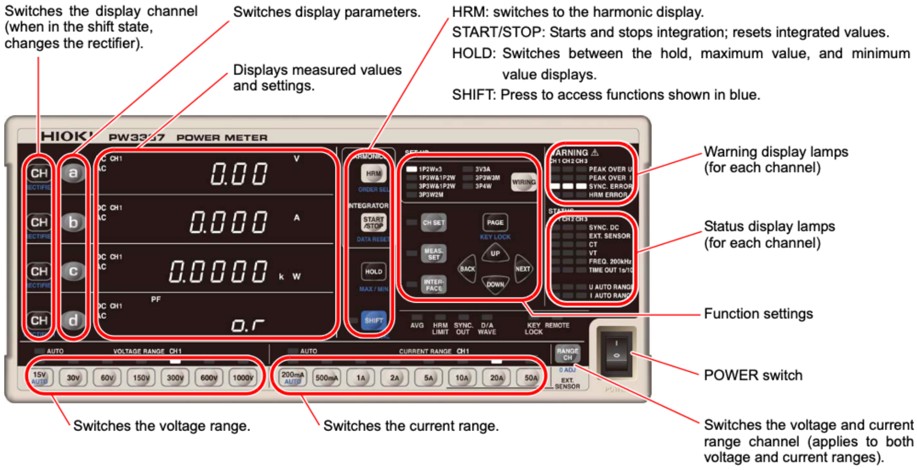


Figure 3.3: Power Meter *PW*337 by Hioki: display commands.

3.1 Instrument Specifications

The main features of the power meter *PW*337 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 *PW*337 accuracy is 0,1%. For complete details, please refer to the specifications in the website.

- **Wide frequency band**

  The *PW*337 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 *PW*337 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 *PW*337 can be connected to a PC through the LAN connection, to control the instrument remotely. The control occur thanks a specif instrument's commands, that the user invite 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.

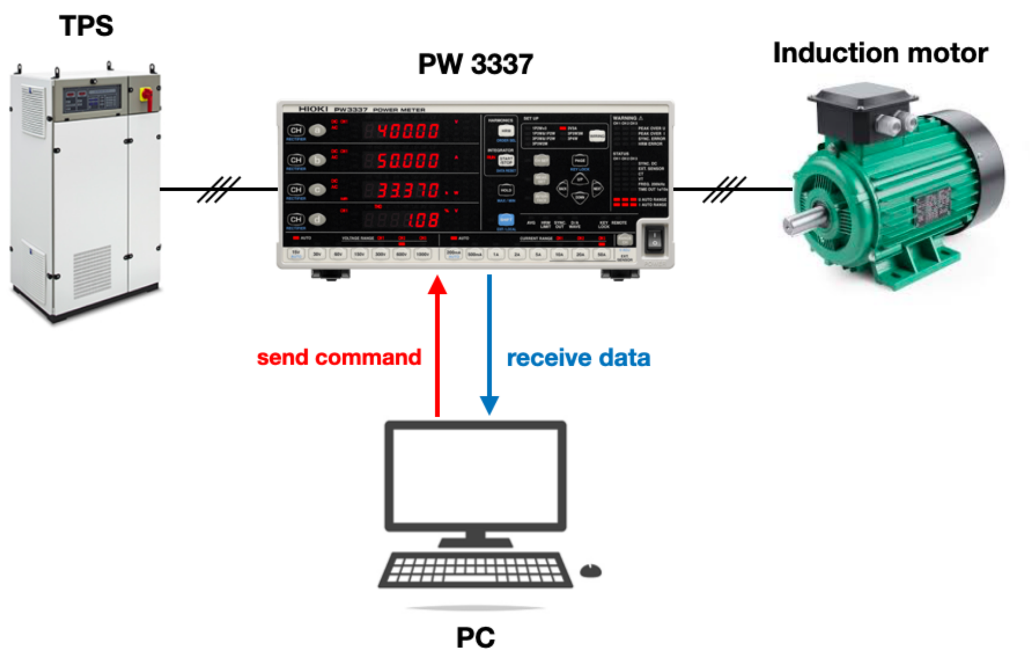


Figure 3.5: Scheme of principle of the measurement setup.

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:

```
from tkinter import *

# create the main window
window = Tk()
window.configure(bg='#cccccc')
window.title('Motor analysis')
window.geometry('1280x800')

fisrt_button = Button(window, text='text', height=2, width=10)
fisrt_button.grid(row=1, column=0)

window.mainloop()
```

These rows code generate the graphical users interface below.

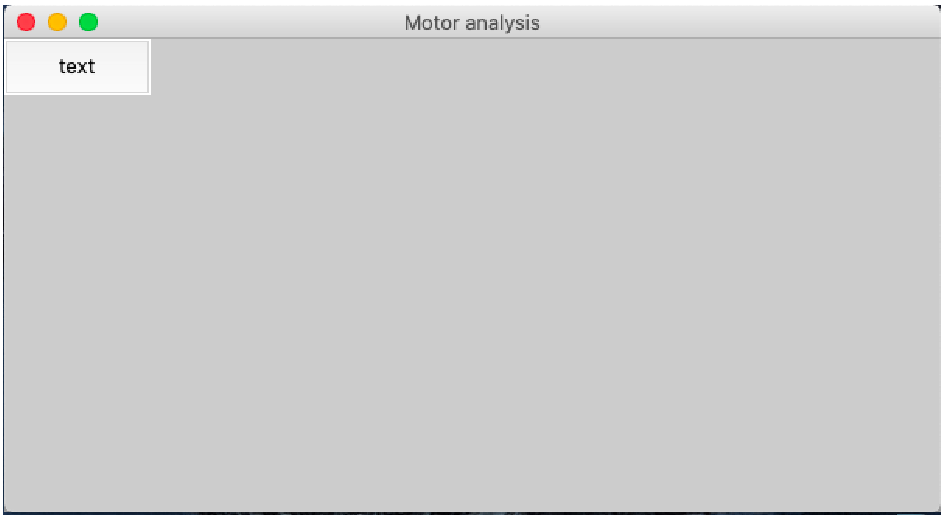


Figure 4.1: A simple GUI interface with one button.

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).

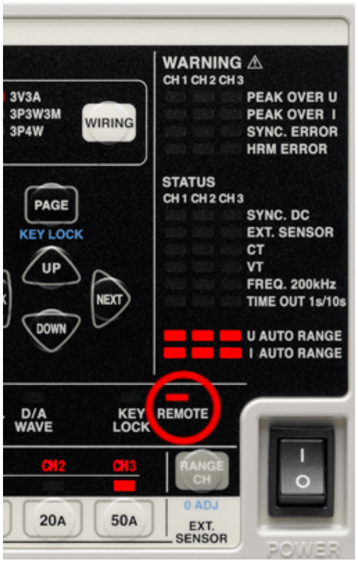


Figure 4.3: The red lamp on the front panel of the instrument displays after the successful remote connection to the instrument.

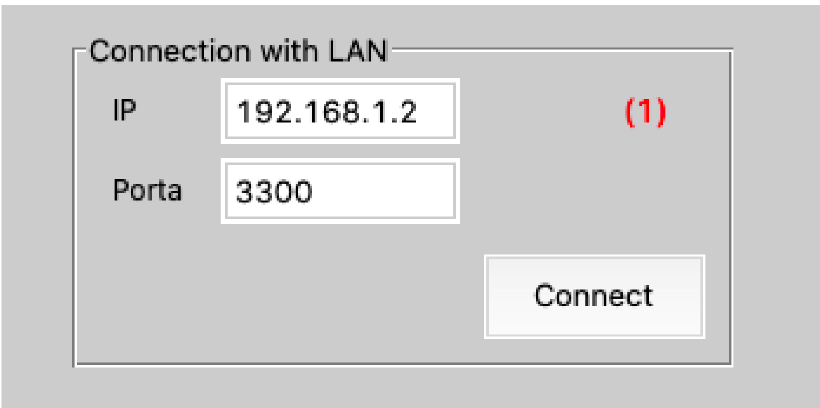


Figure 4.4: LAN connection section.

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.

```
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.

```
def connection1():
    B_conn['state'] = 'disabled'
    B_openfile['state'] = 'normal'

    # Instantiation of the LAN communication class
    global lan
    lan = Lan(Timeout_default)

    # Connect
    IP = box_con.get()
    port = int(box_port.get())
    #print("IP?")
    #IP = input()
    #print("Port?")
    #port = int(input())
    if not lan.open(IP, port):
        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.

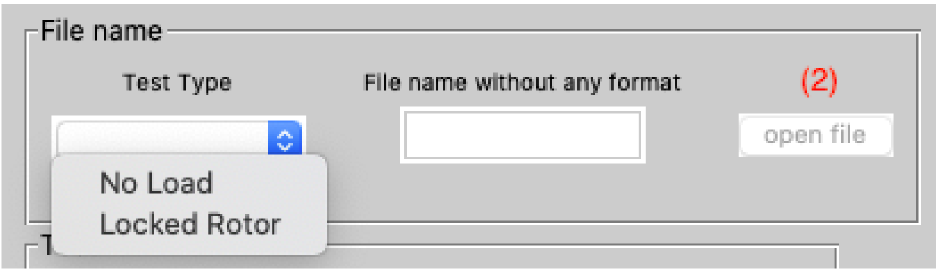


Figure 4.5: File name section.

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:

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

```
    save_button['state'] = 'normal'

    filename = casella3.get()
    if prove_type.get() == 'noload':
        ext ='.VTO'
    if prove_type.get() == 'locked':
        ext ='.CTO'
    if prove_type.get() == '                    ':
        messagebox.showinfo('Attention','Have to select the test type')
    filename1 = filename + ext
```

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

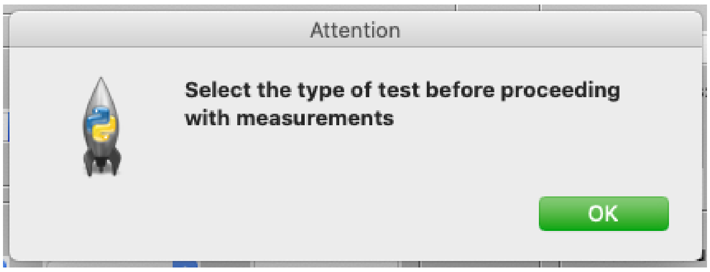


Figure 4.6: Error message for the no test type selection.

4.2.3 Motor data

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

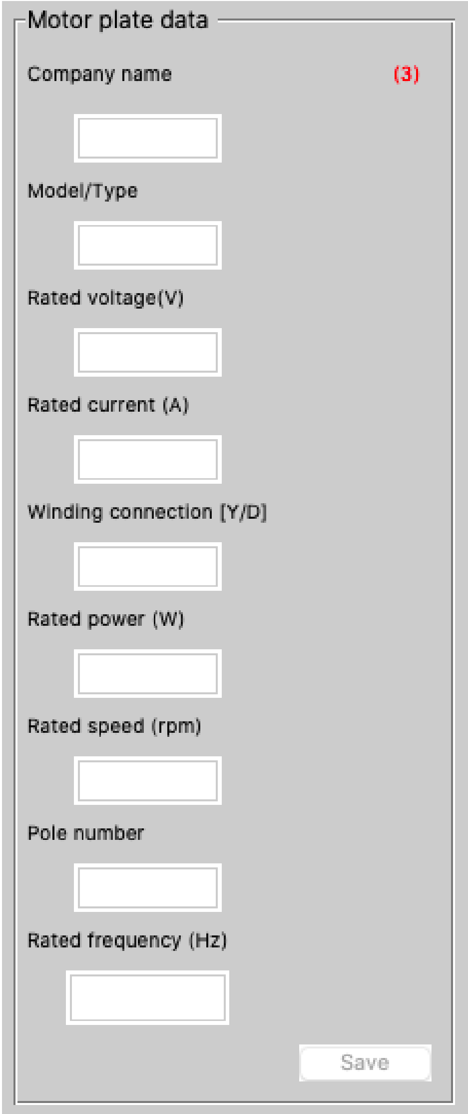


Figure 4.7: The motor data section.

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.

```python
def save1():

    B_salve['state'] = 'normal'
    save_button['state'] = 'disabled'
    global targa_data
    global targa
    targa_data = StringVar()
    targa_data = box1.get()
    targa_data = targa_data + ',' + box2.get()
    targa_data = targa_data + ',' + box3.get()
```

```
    targa_data = targa_data + ’,’ + box4.get()
    targa_data = targa_data + ’,’ + box5.get()
    targa_data = targa_data + ’,’ + box6.get()
    targa_data = targa_data + ’,’ + box7.get()
    targa_data = targa_data + ’,’ + box8.get()
    targa_data = targa_data + ’,’ + box9.get()
    targa = []
    for w in targa_data.split(’,’):
        targa.append(w)
    print(targa)
    file = open(open_file(), ’w’)
    file.write(targa[0] + ’,’ + targa[1] + ’\n’ + targa[5] + ’,’ +
                targa[2] + ’,’ + targa[3] + ’,’ + targa[4] + ’,’ +
                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.

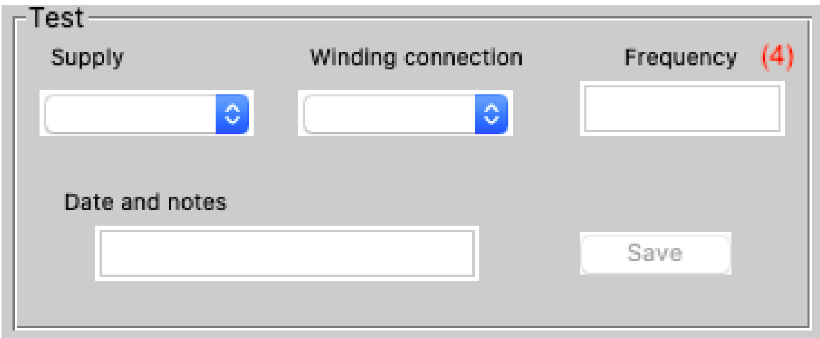


Figure 4.8: Test section.

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.

```python
def put_prove():

    B_salve['state'] = 'disabled'
    B_save2['state'] = 'normal'
    supp = supply.get()
    conn = connection.get()
    freq = casella43.get()
    dataenote = casella4.get()

    file = open(open_file(), 'a')
    name1 = open_file()
    file.write('\n'+supp+','+freq+','+conn+','
            +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

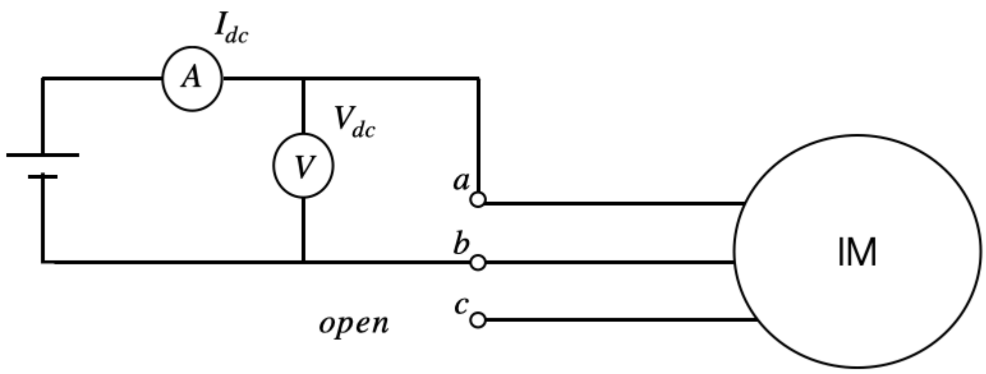


Figure 4.9: DC resistance measure.

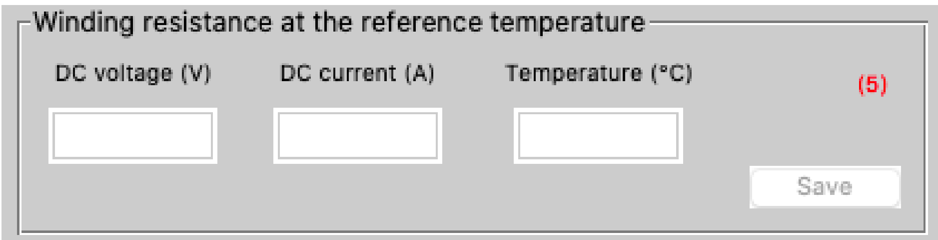


Figure 4.10: winding resistance at the reference temperature section.

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.

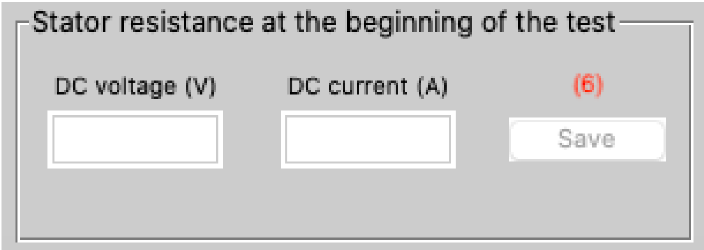


Figure 4.11: Winding resistance at the end of the test.

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:

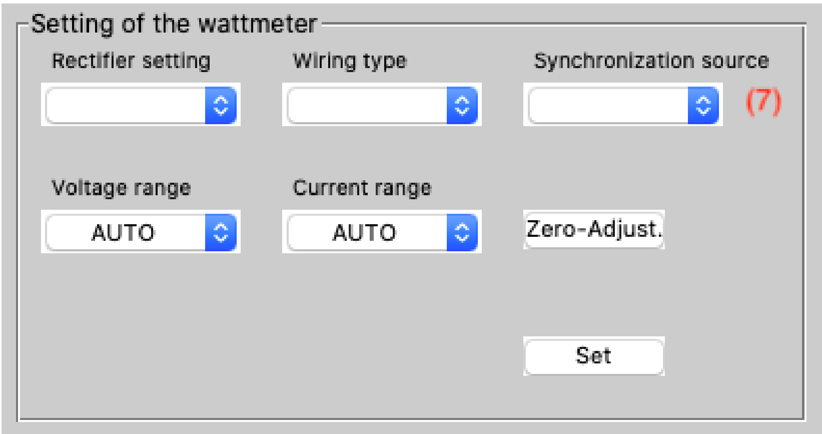


Figure 4.12: Power meter setting section.

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

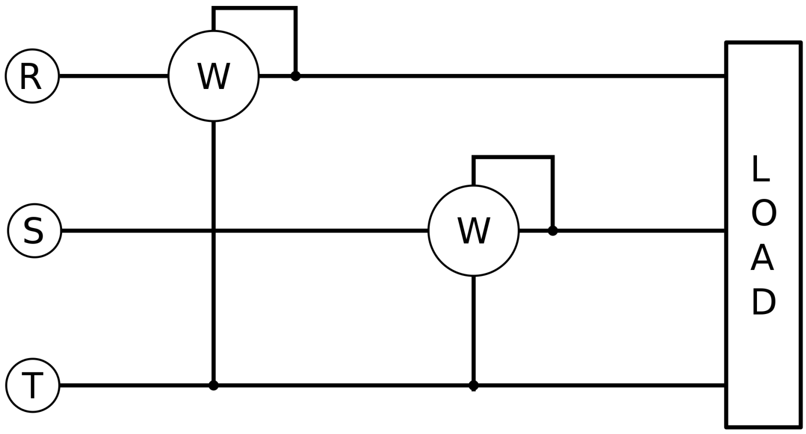
   

   Figure 4.13: Aron connection.

   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].

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

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 cur- rent 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-rage 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: ±10 % of the measurement range

- Current direct input circuitry: ±10 % of the measurement range
- External current sensor input circuitry: ±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]:

```
def Zeroadj():      # performing Zero-adjustment (about 40 sec)
    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.

```
def settings():


    if wiring_range.get() == 'Aron':
        lan.sendMsg('WIR TYPE3')

    if wiring_range.get() == '3 Wattmetri':
        lan.sendMsg('WIR TYPE5')



    if coupling_range.get() == 'AC+DC':
        lan.sendMsg(':DISP U0,I0,P0')

    if coupling_range.get() == 'AC':
        lan.sendMsg(':DISP UAC0,IAC0,PAC0')



    if voltage_range.get() != 'AUTO':
        comm = voltage_range.get()
        lan.sendMsg(':VOLT:RANG '+comm[0:-1])
    if voltage_range.get() == 'AUTO':
        lan.sendMsg(':VOLT:AUTO ON')

    if current_range.get() == '200mA':
        lan.sendMsg(':CURR:RANG 0.2')
    if current_range.get() == '500mA':
        lan.sendMsg(':CURR:RANG 0.5')
    if current_range.get() == '1A':
        lan.sendMsg(':CURR:RANG 1.0')
    if current_range.get() == '2A':
        lan.sendMsg(':CURR:RANG 2.0')
    if current_range.get() == '5A':
        lan.sendMsg(':CURR:RANG 5.0')
    if current_range.get() == '10A':
        lan.sendMsg(':CURR:RANG 10.0')
    if current_range.get() == '20A':
        lan.sendMsg(':CURR:RANG 20.0')
    if current_range.get() == '50A':
        lan.sendMsg(':CURR:RANG 50.0')

```

```
    if current_range.get() == 'AUTO':
        lan.sendMsg(':CURR:AUTO ON')

    else:
        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.

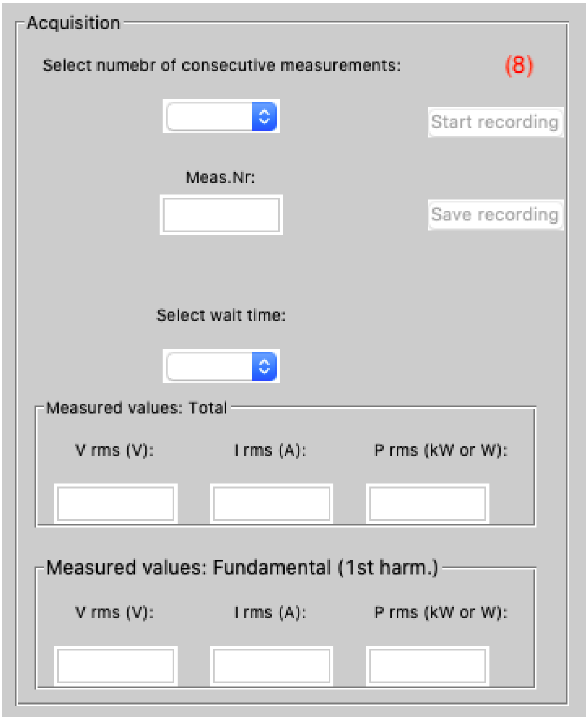


Figure 4.14: Acquisition section.

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.

```
FIMET_18kW.VTO — Edited
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.

```python
def acquisition():
    print(measure.get())
    N = int(measure.get())
    max_V = 0
    min_V = 1000000000000
    sum_V = 0
    max_I = 0
    min_I = 1000000000000
    sum_I = 0
    max_P = 0
    min_P = 1000000000000
    sum_P = 0
    v = []
    i1 = []
    p = []

    if wiring_range.get() == 'Aron':

        if coupling_range.get() == 'AC+DC':

            for j in range(0, N):
                box_meas.delete(0, 3)
                v.append(only_number(lan.SendQueryMsg('MEAS? U0', Timeout_default)))
                i1.append(only_number(lan.SendQueryMsg('MEAS? I0', Timeout_default)))
                p.append(only_number(lan.SendQueryMsg('MEAS? P0', Timeout_default)))
                print(v[j])
                box_meas.insert(END, j + 1)
                if v[j] > max_V:
                    max_V = v[j]
```

```
                if v[j] < min_V:
                    min_V = v[j]
                sum_V = sum_V + v[j]
                if i1[j] > max_I:
                    max_I = i1[j]
                if i1[j] < min_I:
                    min_I = i1[j]
                sum_I = sum_I + i1[j]
                if p[j] > max_P:
                    max_P = p[j]
                if p[j] < min_P:
                    min_P = p[j]
                sum_P = sum_P + p[j]
                sleep(1)  #
            med_V = round(sum_V / N, 4)
            med_I = round(sum_I / N, 4)
            med_P = round(sum_P / N, 4)
            named_tuple = time.localtime()  # get struct_time
            time_string = time.strftime("%H.%M.%S", named_tuple)
            file = open(open_file(), 'a')
            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)

        if coupling_range.get() == 'AC':

            for j in range(0, N):
                box_meas.delete(0, 3)
                box_Vrms.delete(0, 40)
                box_Irms.delete(0, 40)
                box_Prms.delete(0, 40)
                v.append(only_number(lan.SendQueryMsg('MEAS? UAC0', Timeout_default)))
                i1.append(only_number(lan.SendQueryMsg('MEAS? IAC0', Timeout_default)))
                p.append(only_number1(lan.SendQueryMsg('MEAS? PAC0', Timeout_default)))
                box_Vrms.insert(END, v[j])
                box_Irms.insert(END, i1[j])
                box_Prms.insert(END, p[j])

                box_meas.insert(END, j + 1)
                if v[j] > max_V:
                    max_V = v[j]
                if v[j] < min_V:
                    min_V = v[j]
                sum_V = sum_V + v[j]
                if i1[j] > max_I:
                    max_I = i1[j]
                if i1[j] < min_I:
                    min_I = i1[j]
                sum_I = sum_I + i1[j]
                if p[j] > max_P:
                    max_P = p[j]
                if p[j] < min_P:
                    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(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)
```

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.

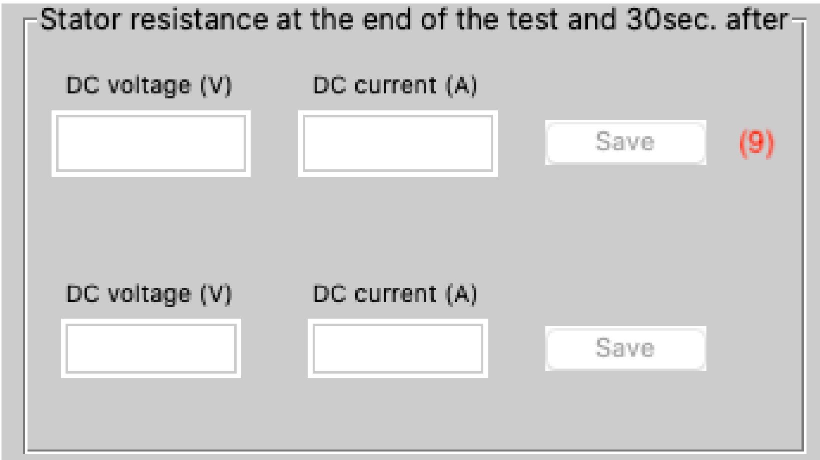


Figure 4.16: The resistance measurements after the test.

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.

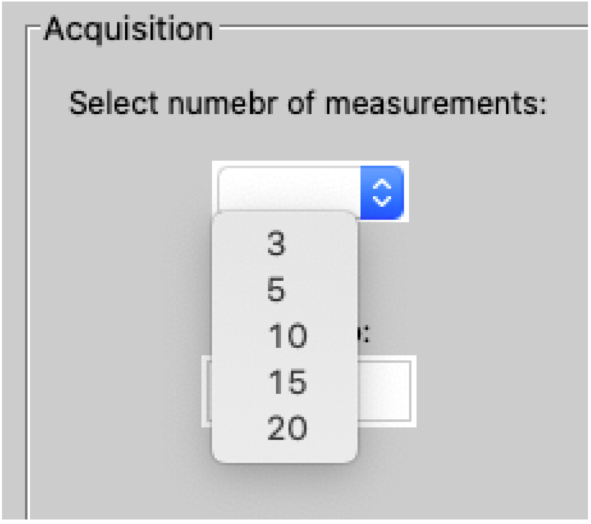


Figure 4.17: Multiple Acquisition settings.

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.

Chapter 5

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:

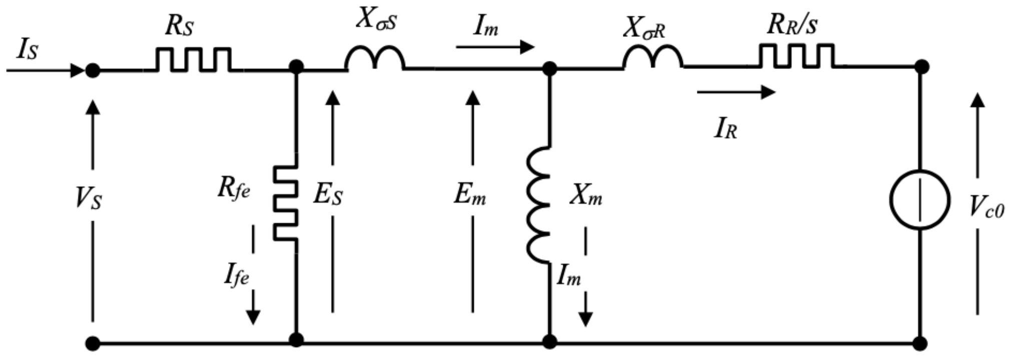


Figure 5.1: Electrical single-phase equivalent steady-state circuit.

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_{c0} 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.

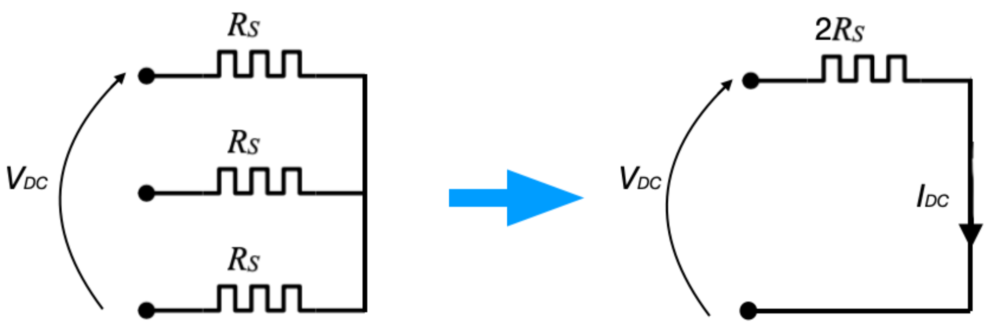


Figure 5.2: equivalent circuit of the wye-connected stator winding during the DC test.

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:

$$R_s = \frac{1}{2}\frac{V_{DC}}{I_{DC}} \tag{5.1}$$

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.

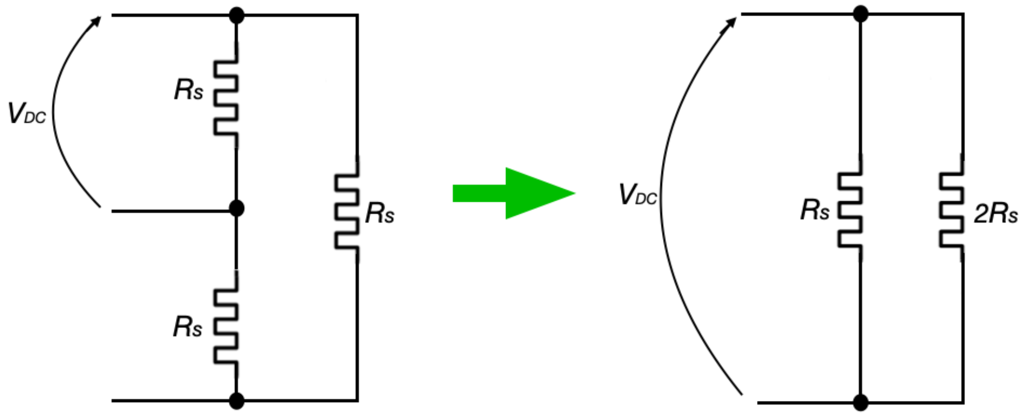


Figure 5.3: Equivalent circuit of the delta-connected stator winding during the DC test.

The final equivalent circuit is given by the Figure 5.4.

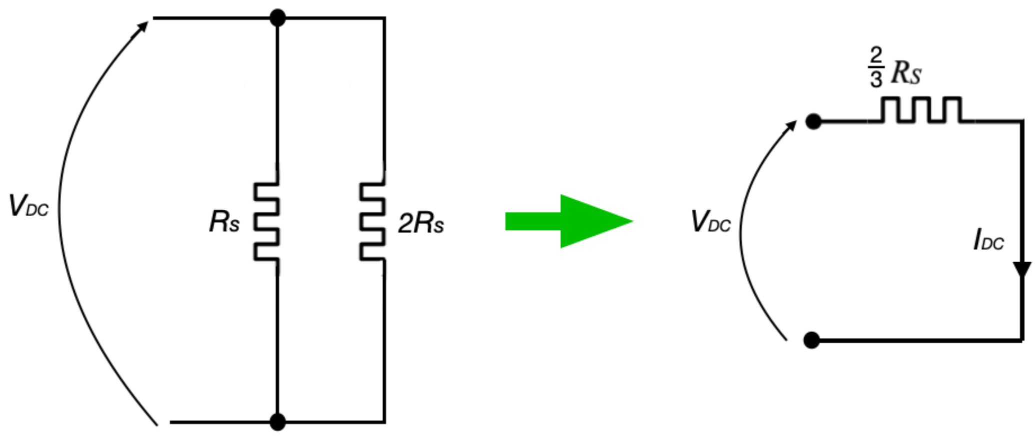


Figure 5.4: Stator windings resistance in case of delta connection.

$$R_{//} = \frac{2R_s^2}{R_s + 2R_s} \tag{5.2}$$

$$R_{//} = \frac{2}{3}R_s \tag{5.3}$$

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

$$R_s = \frac{3}{2} * \frac{V_{DC}}{I_{DC}} \tag{5.4}$$

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

In this way the two equation became:

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

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.

   $$R_{amb} = \frac{3}{2} * \frac{V_{DC_0}}{I_{DC_0}} / col^2 \tag{5.6}$$

   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.

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

   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:

$$R_a = \frac{3}{2} * \frac{V_{DC_a}}{I_{DC_a}} / con^2 \tag{5.8}$$

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.

$$R_b = \frac{3}{2} * \frac{V_{DC_b}}{I_{DC_b}} / con^2 \tag{5.9}$$

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 θ is functions of the time t:

$$\theta = \theta(t) \tag{5.10}$$

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

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

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 (θ_{start}) at the starting of the Standard tests measurements(t_{start});

ii the state (t_{end}, θ_{end}) represents the temperature of the stator windings (θ_{end}) at the end of the Standard tests measurements(t_{end});

iii the states (t_a, θ_a) and (t_b, θ_b) represent respectively the temperature of the stator windings (θ_a) at the end of the Standard tests measurements(t_a) and the temperature of the stator windings (θ_b) 30 seconds after the last measurement (t_b).

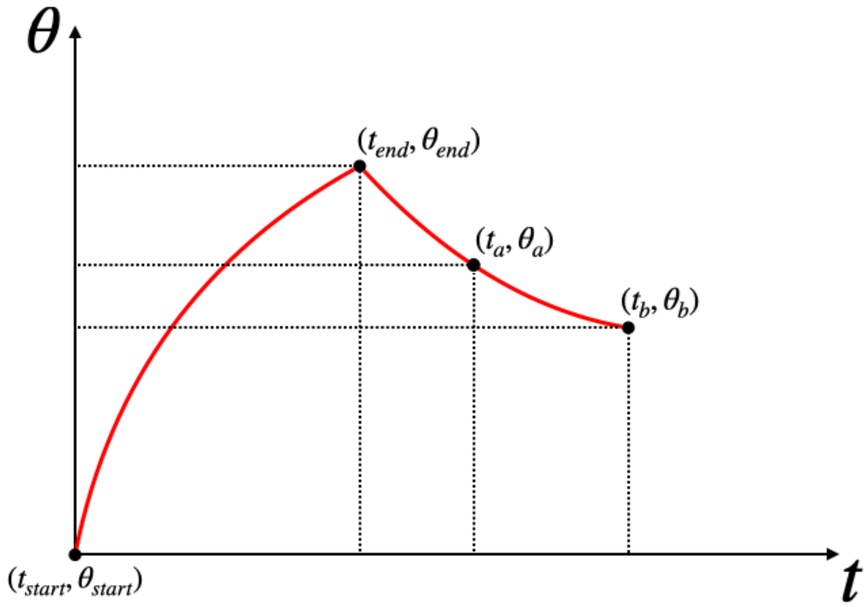


Figure 5.5: Variation of the stator winding temperature during the tests.

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

$$R = R(\theta) \tag{5.12}$$

$$R = R[\theta(t)] \tag{5.13}$$

$$R = R(t) \tag{5.14}$$

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.

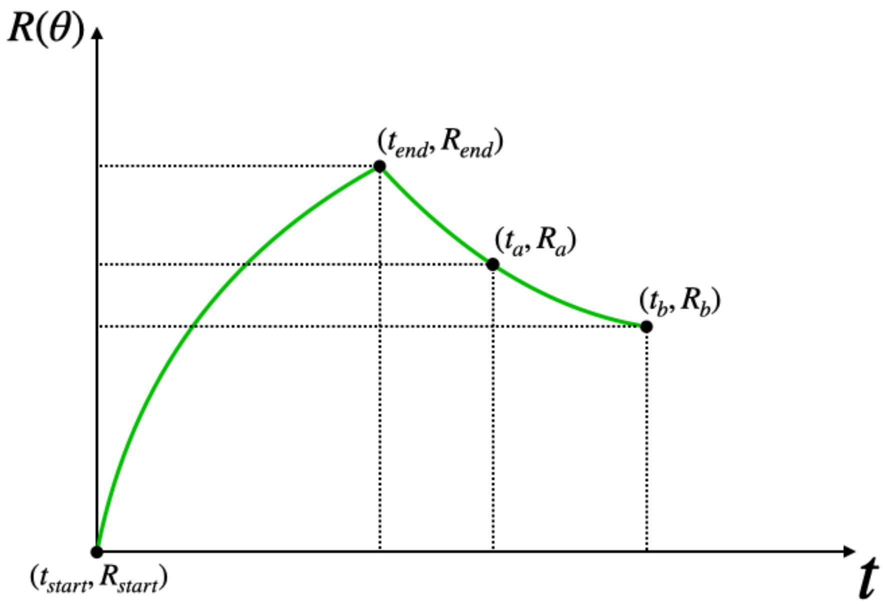


Figure 5.6: Variation of the stator winding resistance during the tests.

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.

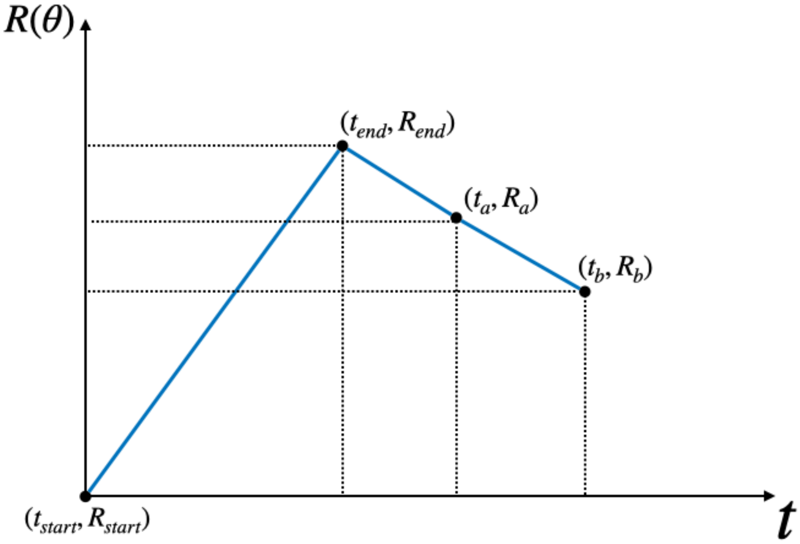


Figure 5.7: Linearization of the stator winding resistance during the tests.

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

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

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

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

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

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

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

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

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

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

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

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

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

$$R_i = R_{start} + RappRT(t_i - t_{start}) \tag{5.21}$$

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.

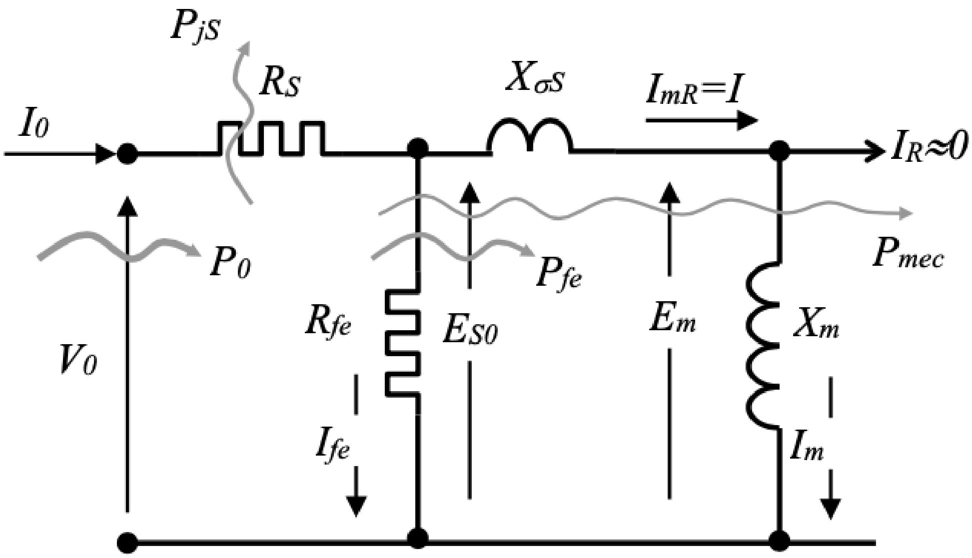


Figure 5.8: No Load single-phase equivalent circuit.

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

$$V_0 = \frac{V_M}{con} \tag{5.22}$$

$$I_0 = \frac{I_M con}{\sqrt{3}} \tag{5.23}$$

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.

$$\cos\varphi_0 = \frac{P_0}{3V_0I_0} \tag{5.24}$$

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

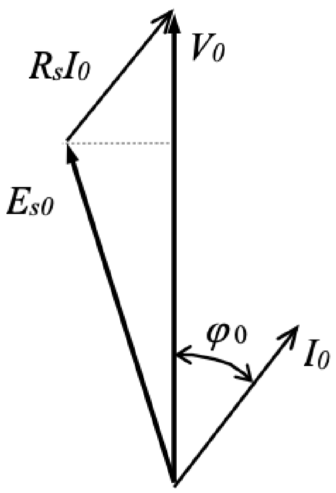


Figure 5.9: Vector Diagram.

$$E_{s0} = \sqrt{(V_0 - R_sI_0\cos\varphi_0)^2 + (R_sI_0\sin\varphi_0)^2} \tag{5.25}$$

$$E_{s0} = E_{s0}con \tag{5.26}$$

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

$$P_0 = P_{mec+fe} + 3R_sI_0^2 \tag{5.27}$$

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

$$P_{mec+fe} = P_0 - 3R_sI_0^2 \tag{5.28}$$

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

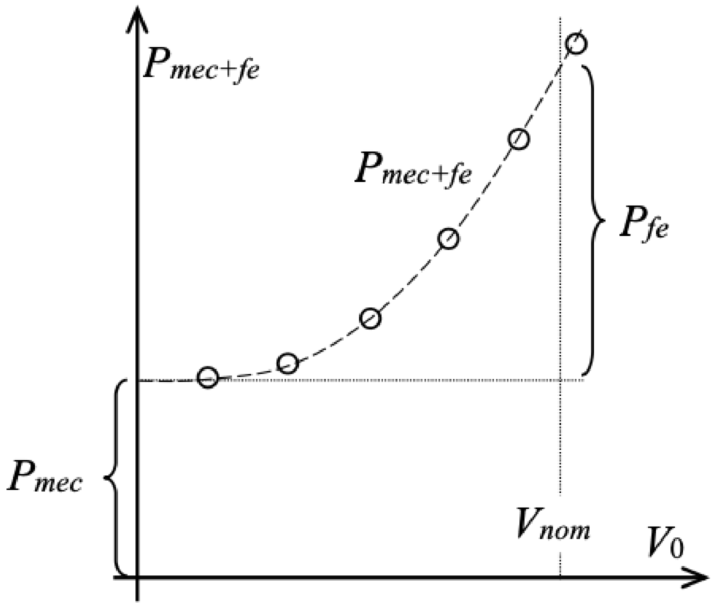


Figure 5.10: Mechanical and core losses with respect to the supply voltage.

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

```
#completamento della RESISTENZA DI STATORE nelle varie
      if ColnNom == 'Y':
         col = sqrt(3)
      elif ColnNom == 'D':
         col = 1
      else:
         messagebox.showwarning("warning", "Error with the
   configuration parameter!")
      IfaseNom = Inom * col / sqrt(3)
      RfaseAmb = 1.5 * Vdc0 / Idc0 / pow(col, 2)
      RfaseInPrv = 1.5 * Vdc1 / Idc1 / pow(col, 2)
      rappFrq = FreqPrv / FreqNom
      DeltaV = col * RfaseInPrv * IfaseNom
      VPrvNom = DeltaV + rappFrq * (Vnom - DeltaV)

           Rad3 = sqrt(3)
      Rfase2 = 1.5 * float(mylist[-19]) / float(mylist[-16]) / pow(col
   , 2)
      Rfase3 = 1.5 * float(mylist[-9]) / float(mylist[-6]) / pow(col,
   2)
      RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])
      #RappTP=2
      print(RappTP)
      RFaseFiPrv = RappTP * (Rfase2 - Rfase3) + Rfase3
      RappRT = (RFaseFiPrv - RfaseInPrv) / (tp[-3] - tp[0])
      print(RFaseFiPrv)
```

```
    print(RappRT)

    # completamento della terza tabella
    box4e.insert(END, round(RFaseFiPrv, 7))

    i=0
    RF=[]
    IFM=[]
    PCM=[]
    PNM=[]
    CFM=[]
    EM=[]

    for i in range(Ndati):

        RF.append(RfaseInPrv+(tp[i] - tp[0]) * RappRT)
        VFM = VM[i] / col
        IFM = Im[i] * col / Rad3
        PCM.append(3 * RF[i] * pow(IFM, 2))
        PNM.append(PTM[i] - PCM[i])
        CFM.append(PTM[i] / Rad3 / VM[i] / Im[i])
        SFM = (sqrt(1 - pow(CFM[i], 2)))
        EFM = sqrt(pow((VFM - RF[i] * IFM * CFM[i]), 2) + pow((RF[i] * 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 eqaul 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 $coeff_1[1]$ is the coefficient of the polynomial interpolation.

$$E' = V_{start_{rated}} coeff_1[1] \quad (5.29)$$

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

$$f_{ratio} = \frac{f_{start}}{f_{rated}} \tag{5.30}$$

$$\Delta V = R_{start} I_{ph_{rated}} con \tag{5.31}$$

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

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

$$I_0 = (I_0 + coeff_2[j]E'^j) \tag{5.33}$$

$$P_0 = (P_0 + coeff_3[j]E'^j) \tag{5.34}$$

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.

$$P_{fe} = P_0 - coeff_3[0] \tag{5.35}$$

$$P_{mec} = coeff_3[0] \tag{5.36}$$

The total electric impedance is given by the equation 5.37.

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

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

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

$$X_s = \sqrt{Z^2 - R_{fe}^2} \tag{5.39}$$

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.

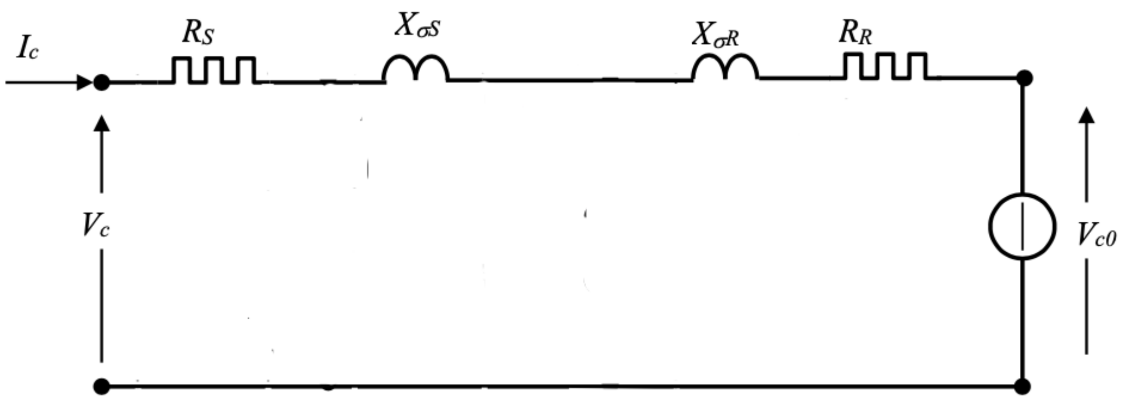


Figure 5.11: Equivalent circuit for the Locked Rotor test.

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

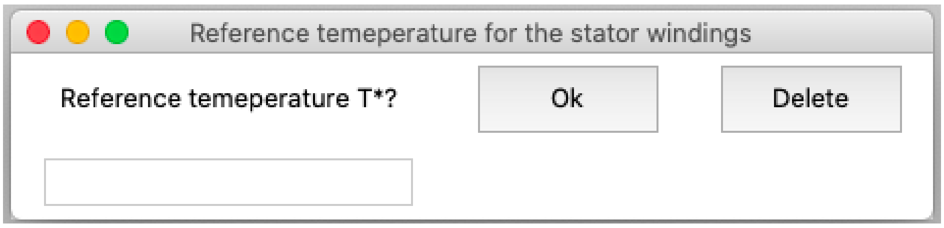


Figure 5.12: Reference temperature choice.

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

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

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.

$$V_{cc_M} = \frac{V_M}{con} \tag{5.41}$$

$$I_{cc_M} = I_M\frac{con}{\sqrt{3}} \tag{5.42}$$

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

$$Z_{cc_M} = \frac{V_{ph_M}}{I_{ph_M}} \tag{5.43}$$

$$R_{cc_M} = \frac{P_M}{3I^2_{ph_M}} \tag{5.44}$$

$$X_{cc_M} = \sqrt{Z_{cc_M}^2 - R_{cc_M}^2} \tag{5.45}$$

$$\cos\varphi_M(i) = \frac{R_{cc_M}}{Z_{cc_M}} \tag{5.46}$$

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 θ_{ref}.

$$K_{ref} = \frac{R_{ref}}{R_i} \tag{5.47}$$

$$P_{c_M}[i] = K_{ref} P_M[i] \tag{5.48}$$

$$R_{c_M} = K_{ref} R_{cc_M} \tag{5.49}$$

$$Z_{c_M} = \sqrt{R_{c_M}^2 + X_{cc_M}^2} \tag{5.50}$$

$$Vphc_M = Z_{c_M} I_{ph_M} \tag{5.51}$$

$$V_{c_M}[i] = Vphc_m con \tag{5.52}$$

$$\cos\varphi_{c_m}[i] = \frac{R_{c_m}}{Z_{c_m}} \tag{5.53}$$

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_{c_M} and I_{ph_M} calculated for each measurement

b $P_{cc}(I_{cc})$ function by the interpolation from the P_{c_M} and I_{ph_M} calculated for each measurement

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

- Vlr Locked rotor voltage;
- Plr Locked rotor losses;
- Zlr Locked rotor impedance;
- $cos\varphi$ Locked rotor power factor;
- Rs Stator resistance;
- Rr Rotor resistance;
- Xs Stator reactance;
- Xr Rotor reactance.

The Locked Rotor parameters must be reported to the reference temperature θ_{ref}.

$$V_c = V_c + coeff_1[i]I_c^i \tag{5.54}$$

$$P_c = P_c + coeff_2[i]I_c^i \tag{5.55}$$

$$cos\varphi_c = \frac{P_c}{\sqrt{3}V_c I_c} \tag{5.56}$$

$$Z_{cc} = \frac{\sqrt{3}V_c}{I_c con^2} \tag{5.57}$$

$$Z_{aux} = \frac{\sqrt{3}(Vc - coeff_1[0])}{I_c con^2} \tag{5.58}$$

$$R_{cc} = Z_{cc} cos\varphi_c \tag{5.59}$$

$$X_{cc} = \sqrt{Z_{cc}^2 - R_{cc}^2} \tag{5.60}$$

$$R_r = R_{cc} - R_s \tag{5.61}$$

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

$$X_r = X_{cc} - X_s \tag{5.63}$$

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.

```
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

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

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 the*b*.

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.

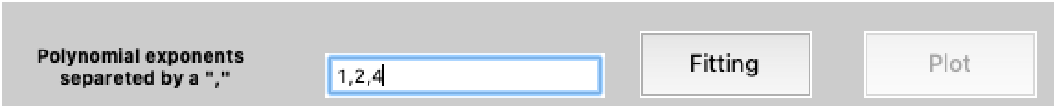


Figure 5.13: interpolation interface.

The code to implement the modify is the following:

```
def test3(x, a, b, c, d, e, f, g, h, i, l):

    a1 = 0
    b1 = 0
    c1 = 0
    d1 = 0
    e1 = 0
    f1 = 0
    g1 = 0
    h1 = 0
    i1 = 0
    l1 = 0
    for indice in range(lung):

        if int(coeff[indice]) == 0:
            a1 = 1
        if int(coeff[indice]) == 1:
            b1 = 1
        if int(coeff[indice]) == 2:
            c1 = 1
        if int(coeff[indice]) == 3:
            d1 = 1
        if int(coeff[indice]) == 4:
            e1 = 1
        if int(coeff[indice]) == 5:
            f1 = 1
        if int(coeff[indice]) == 6:
            g1 = 1
        if int(coeff[indice]) == 7:
            h1 = 1
        if int(coeff[indice]) == 8:
            i1 = 1
        if int(coeff[indice]) == 9:
            l1 = 1

    if a1 == 0:
        a = 0
    if b1 == 0:
        b = 0
    if c1 == 0:
```

```
                c = 0
            if d1 == 0:
                d = 0
            if e1 == 0:
                e = 0
            if f1 == 0:
                f = 0
            if g1 == 0:
                g = 0
            if h1 == 0:
                h = 0
            if i1 == 0:
                i = 0
            if l1 == 0:
                l = 0

            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.

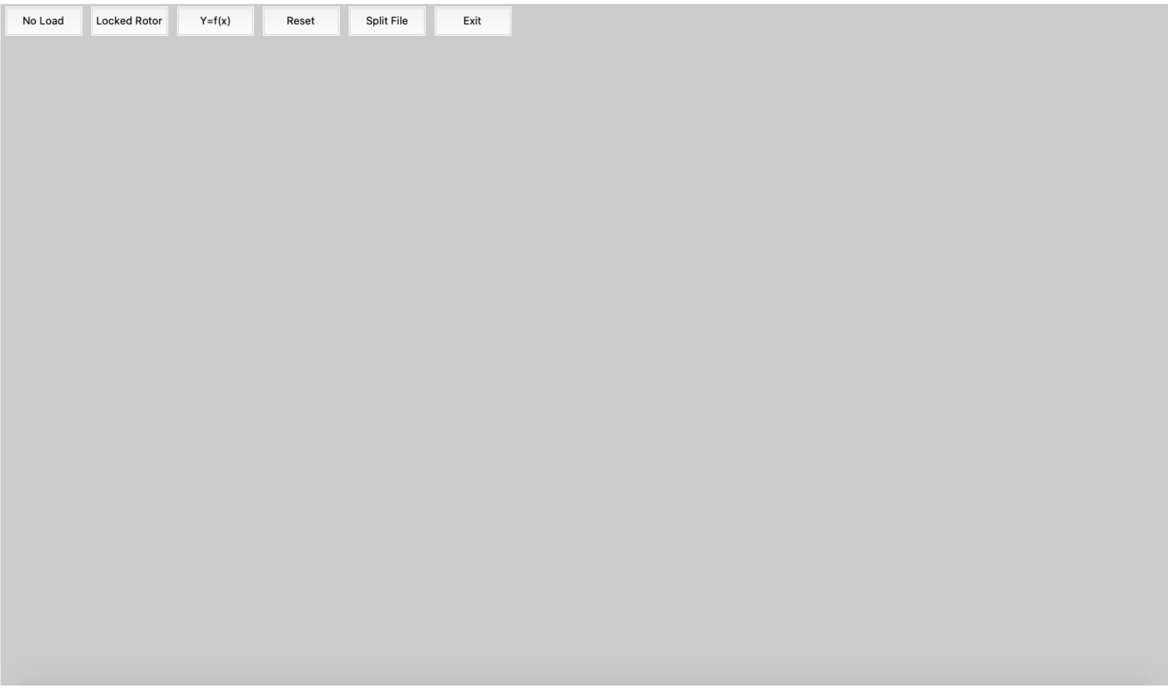


Figure 5.14: The Post-Elaboration GUI.

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

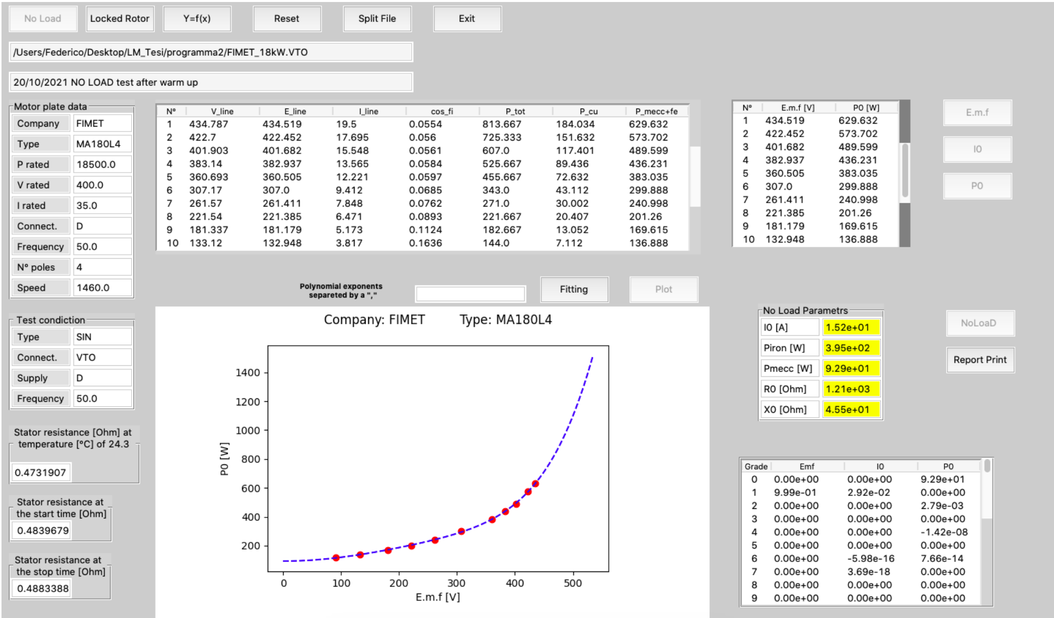


Figure 5.15: No Load GUI.

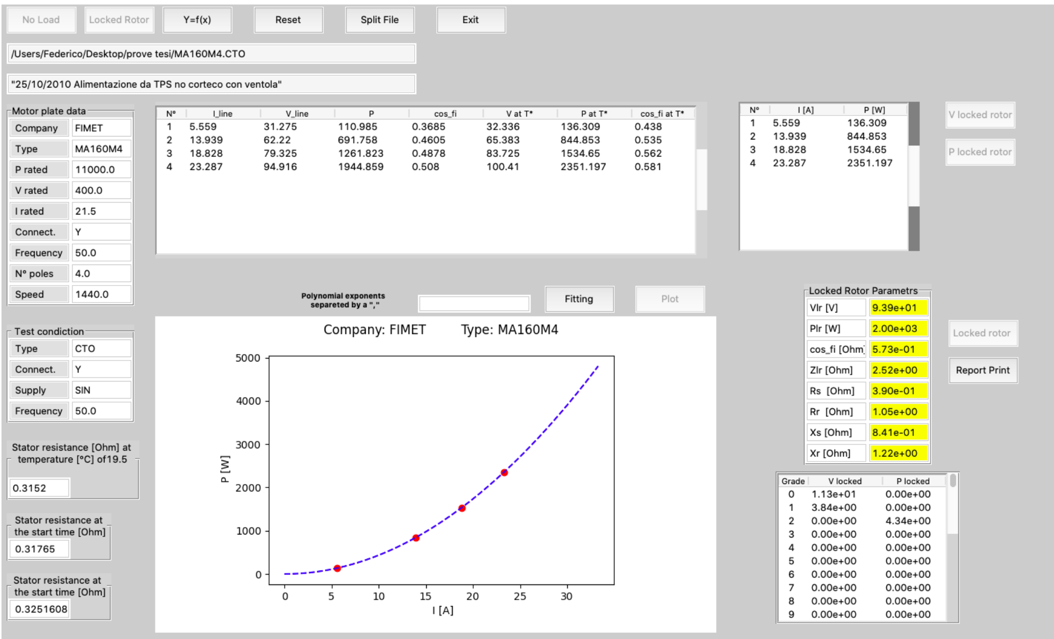


Figure 5.16: Locked Rotor GUI.

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.

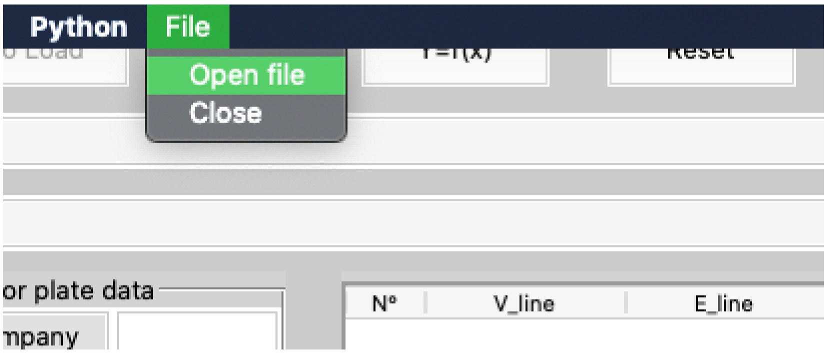


Figure 5.17: Open file option in the No load tool menu bar.

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.

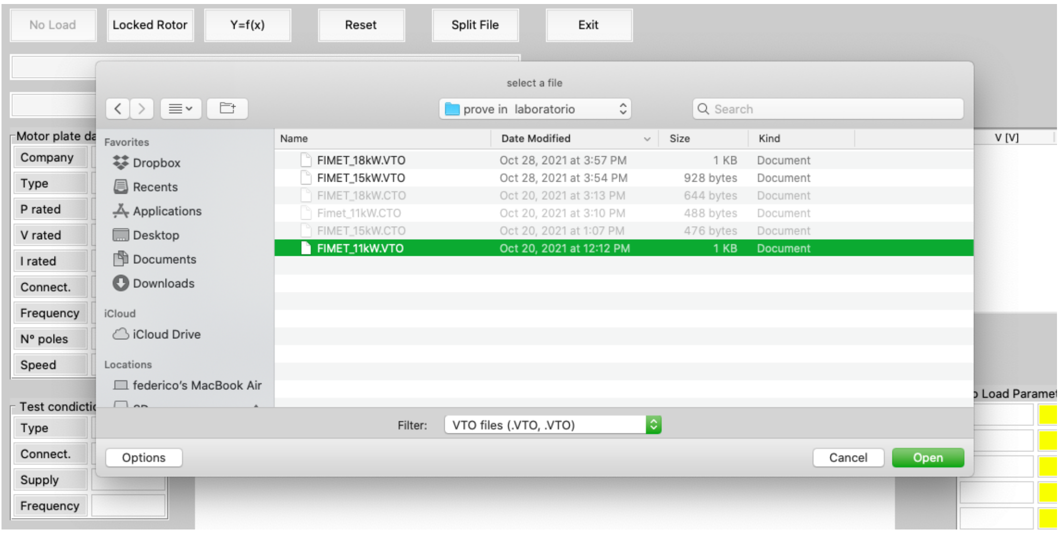


Figure 5.18: Selection of .VTO text file.

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

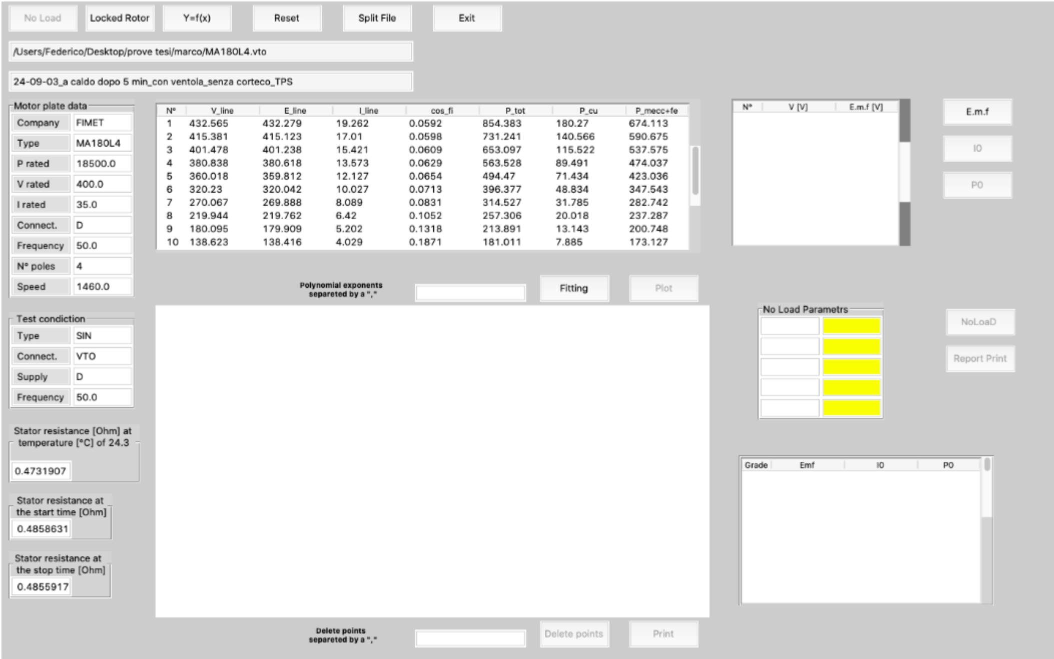


Figure 5.19: The No Load interface after the .VTO selection.

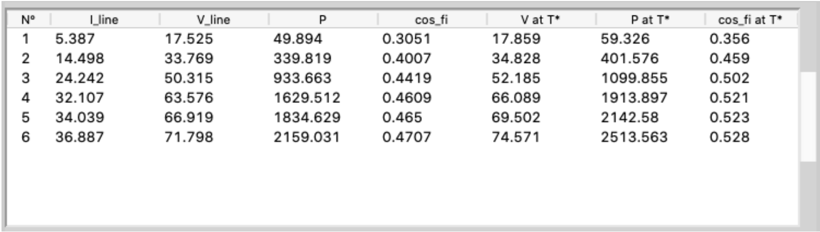
The table data shown in the Figure 5.20 after the selection of the standart test, presents eight different useful data:

N°	V_line	E_line	I_line	cos_fi	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°: 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.

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.

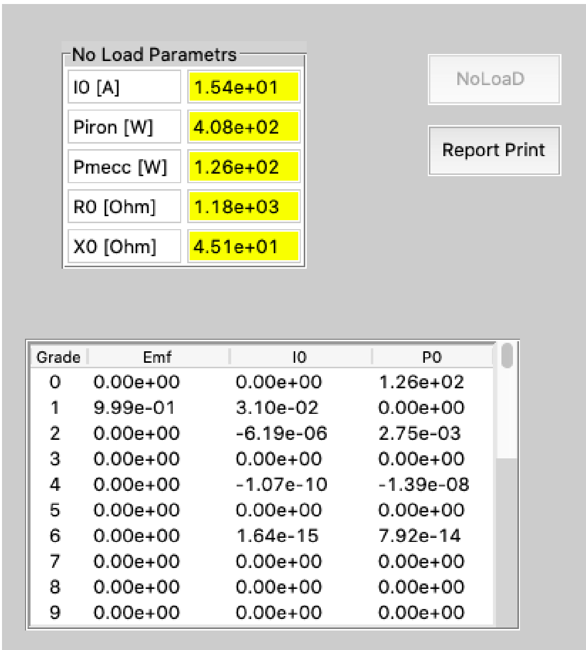


N°	I_line	V_line	P	cos_fi	V at T*	P at T*	cos_fi at T*
1	5.387	17.525	49.894	0.3051	17.859	59.326	0.356
2	14.498	33.769	339.819	0.4007	34.828	401.576	0.459
3	24.242	50.315	933.663	0.4419	52.185	1099.855	0.502
4	32.107	63.576	1629.512	0.4609	66.089	1913.897	0.521
5	34.039	66.919	1834.629	0.465	69.502	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°: 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

I0 [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	I0	P0
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^{th} 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

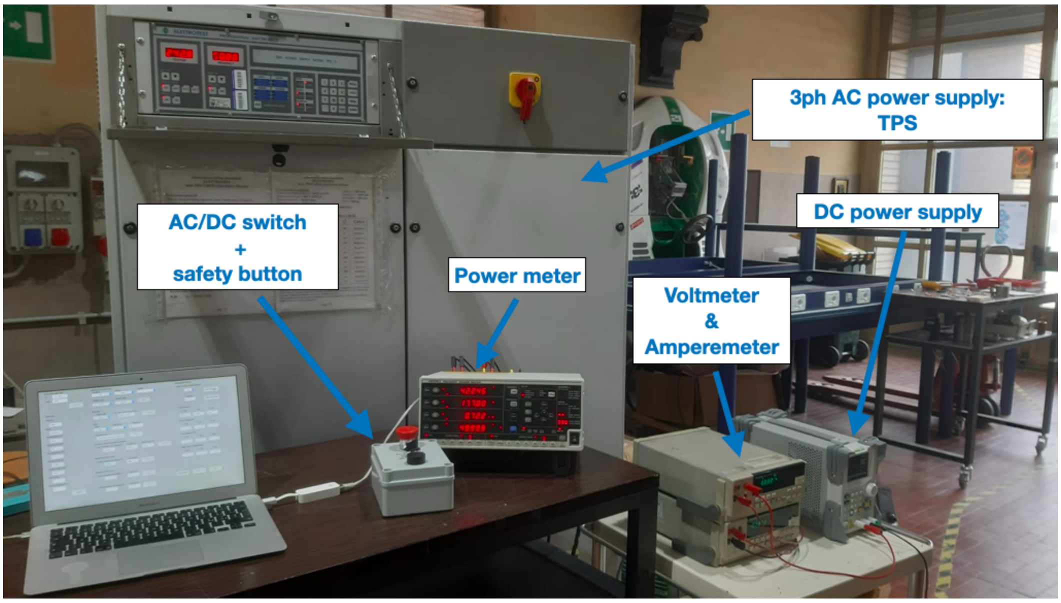


Figure 6.2: Laboratory setup.

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.

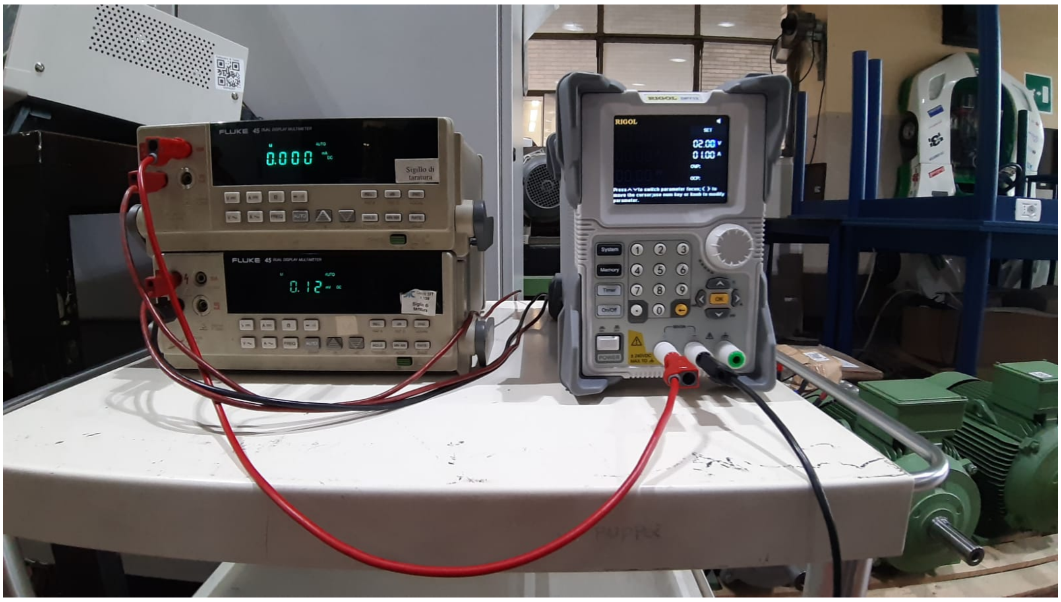


Figure 6.4: DC measurements setup.

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.

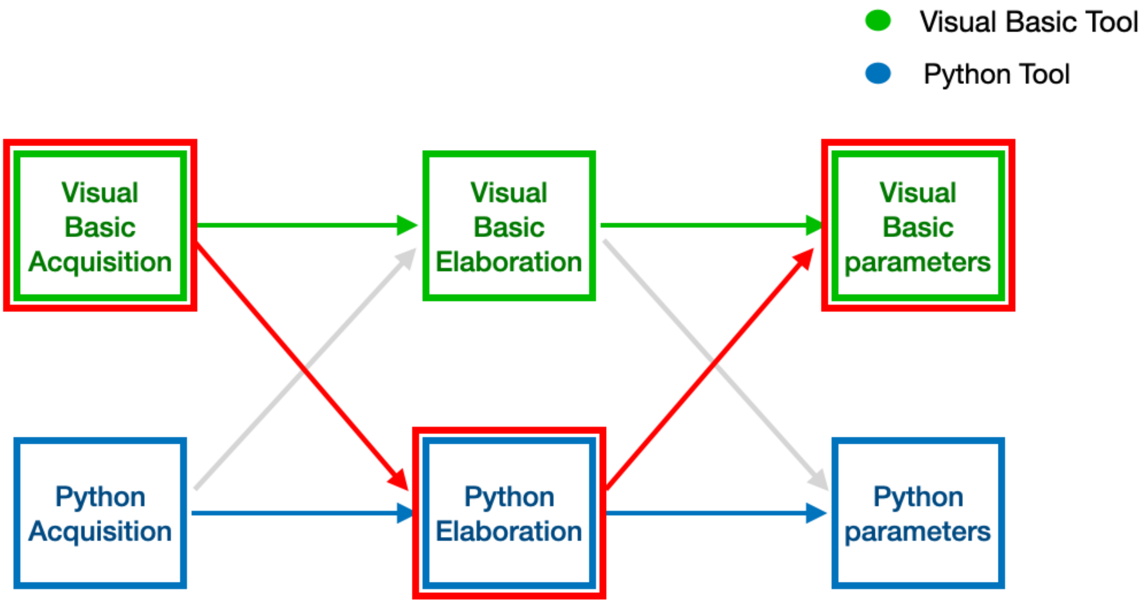


Figure 6.6: Verifying methodology scheme.

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

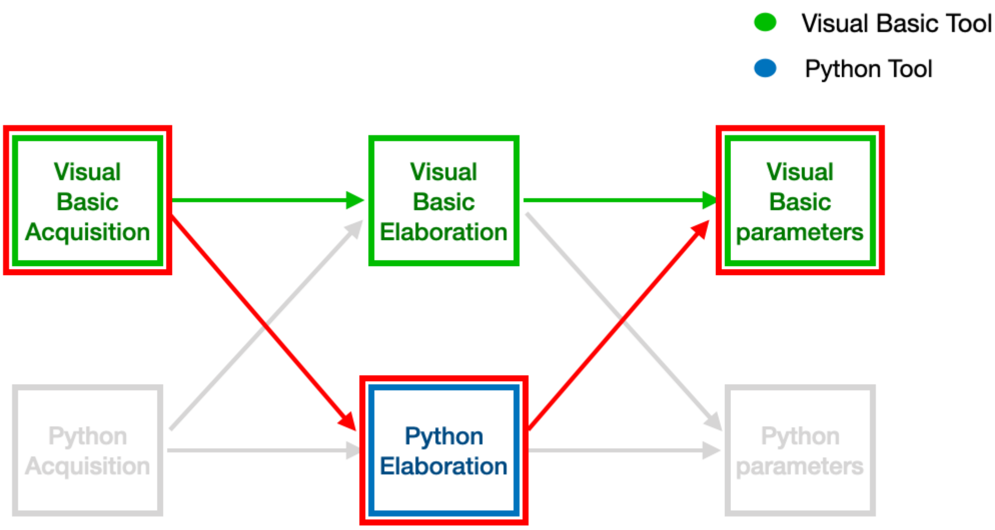


Figure 6.7: Elaboration tool validation methodology scheme.

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.

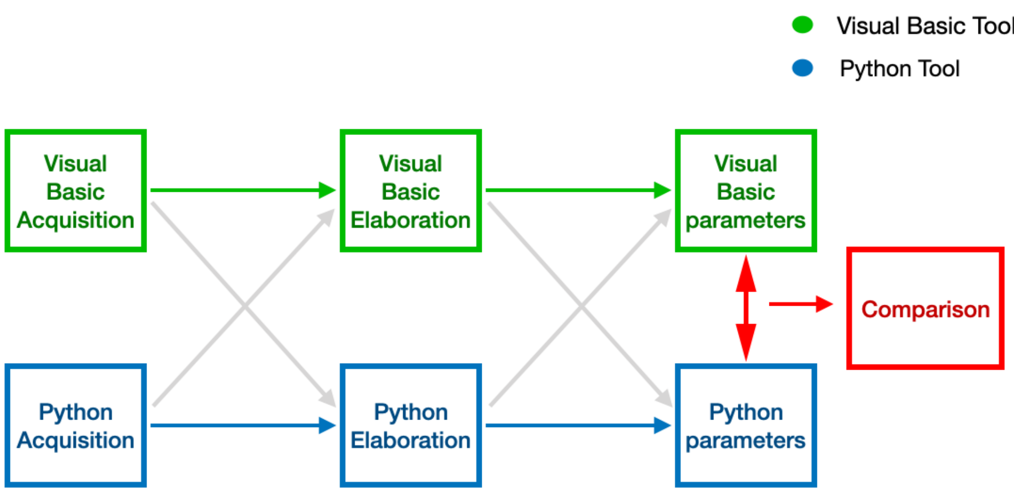


Figure 6.8: Acquisition tool validation methodology scheme.

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} (Ω)	1180	1210	2.54%
Stator Reactance X_0 (Ω)	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.

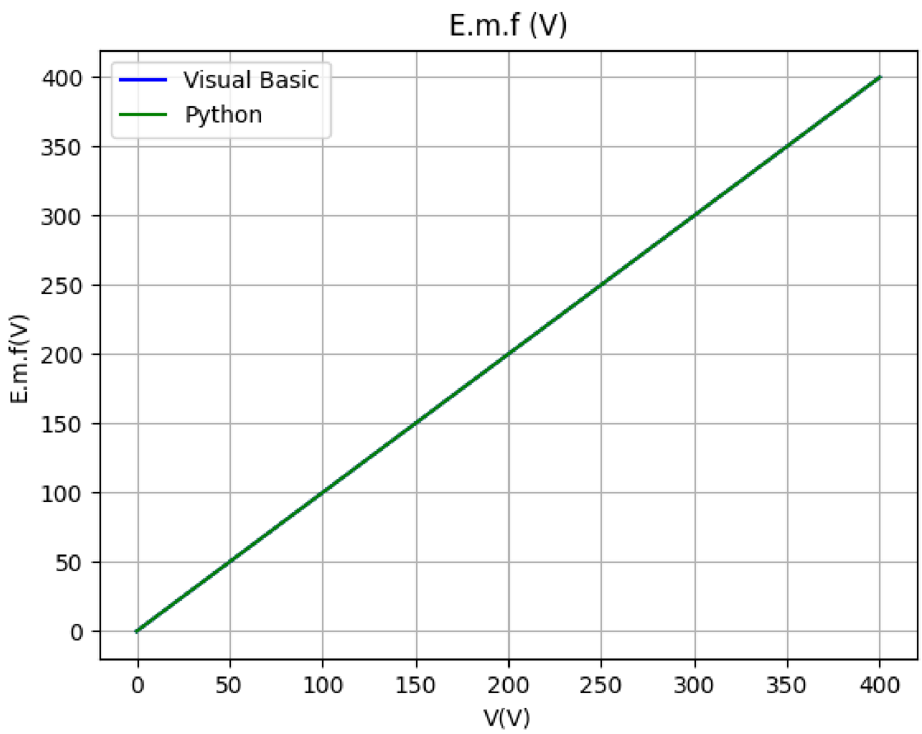


Figure 6.9: No Load E.m.f comparison.

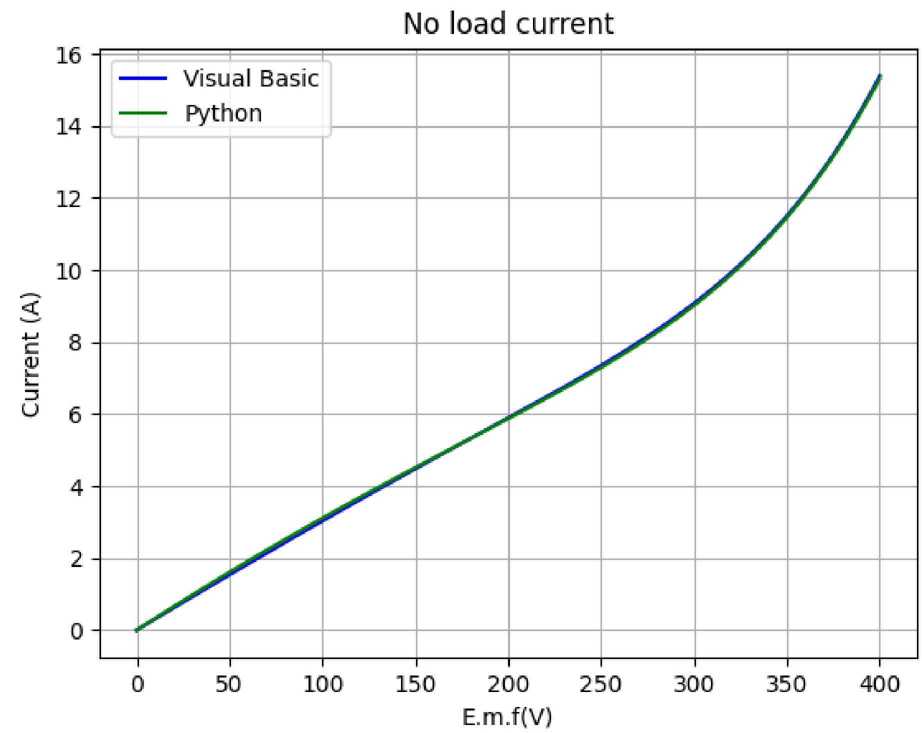


Figure 6.10: No Load current comparison.

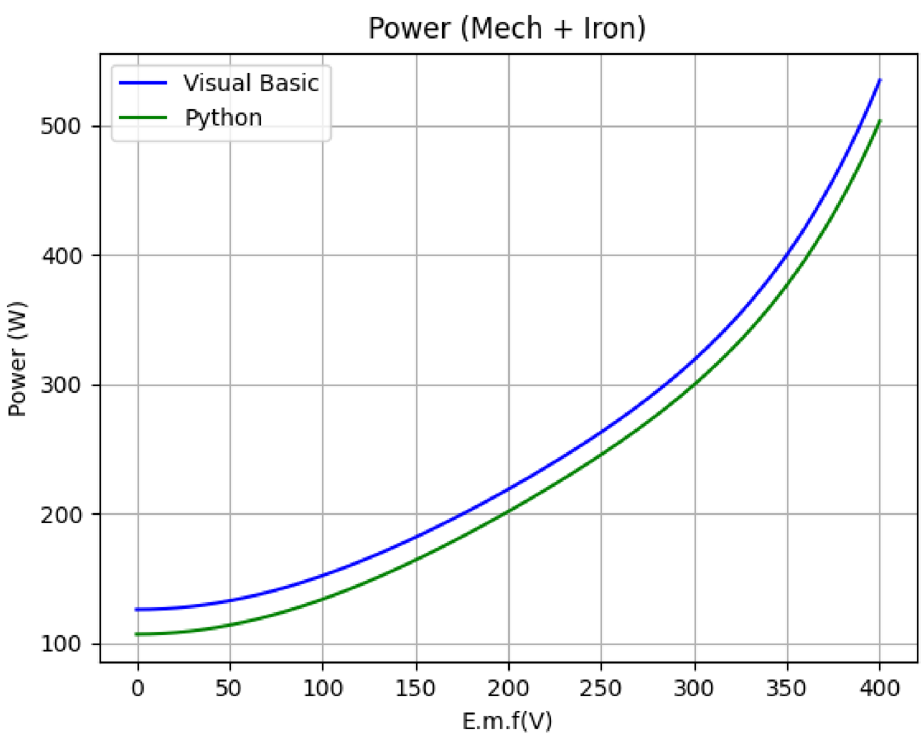


Figure 6.11: Mechanical and Iron Power losses comparison.

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} (Ω)	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.

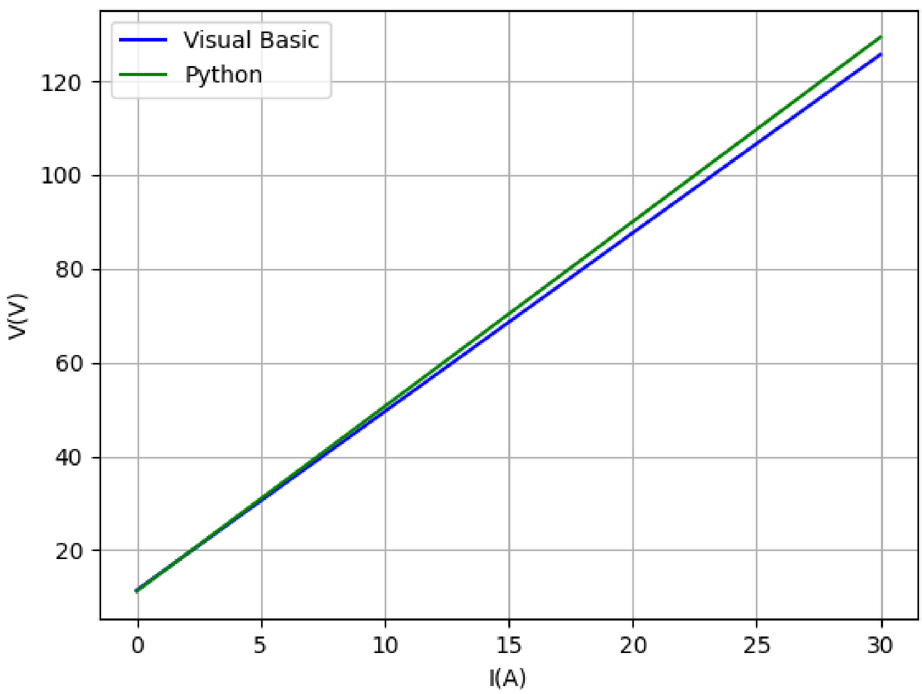


Figure 6.12: No Load E.m.f comparison.

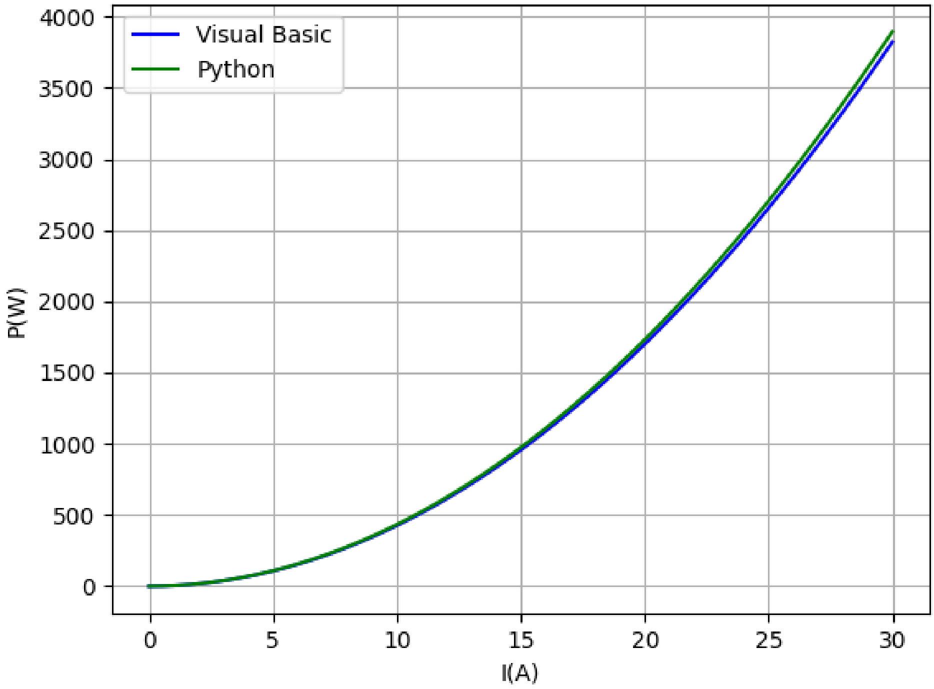


Figure 6.13: No Load current comparison.

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

```python
from tkinter import *
from tkinter import messagebox
from lan import Lan
import time
from threading import Thread
from time import sleep
from PIL import Image,ImageTk

# Timeout(1sec)
Timeout_default = 1

indice_grap = 0
indice_inter = 0
block = 0

#create the main window

window = Tk()
window.configure(bg='#cccccc')
#window.attributes('-fullscreen', True)
window.title('Data Acquisition Tool for Standard Tests on Induction
    Motors')
window.geometry('1300x800')


def save1():

    B_salve['state'] = 'normal'
    save_button['state'] = 'disabled'
    global targa_data
    global targa
    targa_data = StringVar()
    targa_data = box1.get()
    targa_data = targa_data + ',' + box2.get()
    targa_data = targa_data + ',' + box3.get()
    targa_data = targa_data + ',' + box4.get()
    targa_data = targa_data + ',' + box5.get()
    targa_data = targa_data + ',' + box6.get()
    targa_data = targa_data + ',' + box7.get()
```

```
    targa_data = targa_data + ',' + box8.get()
    targa_data = targa_data + ',' + box9.get()
    targa = []
    for w in targa_data.split(','):
        targa.append(w)
    print(targa)
    file = open(open_file(), 'w')
    file.write(targa[0] + ',' + targa[1] + '\n' + targa[5] + ',' +
   targa[2] + ',' + targa[3] + ',' + targa[4] + ',' + targa[8] + ','
   + targa[7] + ',' + targa[6])

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

def only_number(value):
    value = str(value[4:])
    value = str(value[:-4])
    value = float(value[2:])
    return value

def only_number1(value):
    #value = str(value[4:])
    #value = str(value[:-4])
    value = float(value[5:])
    return value

def only_number_PK(value):
    value = str(value[9:])
    value = str(value[:-5])
    value = float(value[2:])
    return value

#funzione selected da cancellare probabilmente
def selected():
    if wiring_range.get() == 'Aron':
        #lan = Lan(Timeout_default)
        lan.sendMsg('WIR TYPE3')
        #v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)
        #casella_v1.insert(END, v1)

    if wiring_range.get() == 'tipo 1':
        #lan = Lan(Timeout_default)
        lan.sendMsg('WIR TYPE1')

    if wiring_range.get() == '3 Wattmetri':
        #lan = Lan(Timeout_default)
        lan.sendMsg('WIR TYPE5')
        #v2 = lan.SendQueryMsg('MEAS? U3', Timeout_default)
        #casella_v1.insert(END, v2)

    if coupling_range.get() == 'AC+DC':
        #lan = Lan(Timeout_default)
        lan.sendMsg('AOUT:ITEM:U1 RMS')  # ????????2

    if coupling_range.get() == 'AC':
        #lan = Lan(Timeout_default)
        lan.sendMsg('DISP:ITEM:U1 FND')  #??????????
```

```
    if measure.get() == '1':

        v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
        print(v1)
        box_Vrms.delete(0, END)
        box_Vrms.insert(0, v1)


    if measure.get() == '3':
        global sum_v
        sum_v = 0.0

        index = 50
        for i in range(index):
            v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
            print("{:.3e}".format(only_number(v1)), v1)
            sum_v = sum_v + only_number(v1)
            #Timeout_default = Timeout_default + 0.1
        v2 = "{:.3e}".format(sum_v/index)
        box_Vrms.insert(0, v2)


    if measure.get() == '5':

        v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)

    if measure.get() == '10':
        v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)

    if measure.get() == '15':
        v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)

    if measure.get() == '20':
        v1 = lan.SendQueryMsg('MEAS? U1', Timeout_default)

def settings():


    if wiring_range.get() == 'Aron':
        lan.sendMsg('WIR TYPE3')

    if wiring_range.get() == '3 Wattmetri':
        lan.sendMsg('WIR TYPE5')



    if coupling_range.get() == 'AC+DC':
        lan.sendMsg(':DISP U0,I0,P0')

    if coupling_range.get() == 'AC':
        lan.sendMsg(':DISP UAC0,IAC0,PAC0')



    if voltage_range.get() != 'AUTO':
        comm = voltage_range.get()
```

```
        lan.sendMsg(':VOLT:RANG '+comm[0:-1])
    if voltage_range.get() == 'AUTO':
        lan.sendMsg(':VOLT:AUTO ON')

    if current_range.get() == '200mA':
        lan.sendMsg(':CURR:RANG 0.2')
    if current_range.get() == '500mA':
        lan.sendMsg(':CURR:RANG 0.5')
    if current_range.get() == '1A':
        lan.sendMsg(':CURR:RANG 1.0')
    if current_range.get() == '2A':
        lan.sendMsg(':CURR:RANG 2.0')
    if current_range.get() == '5A':
        lan.sendMsg(':CURR:RANG 5.0')
    if current_range.get() == '10A':
        lan.sendMsg(':CURR:RANG 10.0')
    if current_range.get() == '20A':
        lan.sendMsg(':CURR:RANG 20.0')
    if current_range.get() == '50A':
        lan.sendMsg(':CURR:RANG 50.0')

    if current_range.get() == 'AUTO':
        lan.sendMsg(':CURR:AUTO ON')

    else:

        pass

'''
def start_thread():
        # Assign global variable and initialize value
        global stop
        stop = 0

        # Create and launch a thread
        t = Thread(target=acquisition)
        t.start()
'''
def acquisition():
    print(measure.get())
    N = int(measure.get())
    max_V = 0
    min_V = 1000000000000
    sum_V = 0
    max_I = 0
    min_I = 1000000000000
    sum_I = 0
    max_P = 0
    min_P = 1000000000000
    sum_P = 0
    v = []
    i1 = []
    p = []

    if wiring_range.get() == 'Aron':

        if coupling_range.get() == 'AC+DC':
```

```

            for j in range(0, N):
                box_meas.delete(0, 3)
                v.append(only_number(lan.SendQueryMsg('MEAS? U0',
Timeout_default)))
                i1.append(only_number(lan.SendQueryMsg('MEAS? I0',
Timeout_default)))
                p.append(only_number(lan.SendQueryMsg('MEAS? P0',
Timeout_default)))
                print(v[j])
                box_meas.insert(END, j + 1)
                if v[j] > max_V:
                    max_V = v[j]
                if v[j] < min_V:
                    min_V = v[j]
                sum_V = sum_V + v[j]
                if i1[j] > max_I:
                    max_I = i1[j]
                if i1[j] < min_I:
                    min_I = i1[j]
                sum_I = sum_I + i1[j]
                if p[j] > max_P:
                    max_P = p[j]
                if p[j] < min_P:
                    min_P = p[j]
                sum_P = sum_P + p[j]
                sleep(1)  #
            med_V = round(sum_V / N, 4)
            med_I = round(sum_I / N, 4)
            med_P = round(sum_P / N, 4)
            named_tuple = time.localtime()  # get struct_time
            time_string = time.strftime("%H.%M.%S", named_tuple)
            file = open(open_file(), 'a')
            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)

        if coupling_range.get() == 'AC':

            for j in range(0, N):
                box_meas.delete(0, 3)
                box_Vrms.delete(0, 40)
                box_Irms.delete(0, 40)
                box_Prms.delete(0, 40)
                v.append(only_number(lan.SendQueryMsg('MEAS? UAC0',
Timeout_default)))
                i1.append(only_number(lan.SendQueryMsg('MEAS? IAC0',
Timeout_default)))
                p.append(only_number1(lan.SendQueryMsg('MEAS? PAC0',
Timeout_default)))
                box_Vrms.insert(END, v[j])
                box_Irms.insert(END, i1[j])
                box_Prms.insert(END, p[j])

                box_meas.insert(END, j + 1)
                if v[j] > max_V:
```

```
                    max_V = v[j]
                if v[j] < min_V:
                    min_V = v[j]
                sum_V = sum_V + v[j]
                if i1[j] > max_I:
                    max_I = i1[j]
                if i1[j] < min_I:
                    min_I = i1[j]
                sum_I = sum_I + i1[j]
                if p[j] > max_P:
                    max_P = p[j]
                if p[j] < min_P:
                    min_P = p[j]
                sum_P = sum_P + p[j]
                sleep(1)  #
            med_V = round(sum_V / N, 4)
            med_I = round(sum_I / N, 4)
            med_P = round(sum_P / N, 4)
            named_tuple = time.localtime()  # get struct_time
            time_string = time.strftime("%H.%M.%S", named_tuple)
            file = open(open_file(), 'a')
            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)

def stop():
    B_salve3['state'] = 'normal'
    B_salve4['state'] = 'normal'


#if coupling_range.get() = 'AC+DC':


    #if wiring_range.get() == '3 Wattmetri':

'''
    if measure.get() == '1':
        while True:
        #for i in range(0,30):
            v1 = lan.SendQueryMsg('MEAS? UDC1', Timeout_default)
            #print(("{:.4e}".format(only_number(v1))),time_string)
            box_meas.delete(0, 3)
            box_meas.insert(END, i)
            named_tuple = time.localtime()  # get struct_time
            time_string = time.strftime("%H.%M.%S", named_tuple)
            print(("{:.4e}".format(only_number(v1))), time_string)
            i=i+1
            file = open(open_file(), 'a')
            file.write('\n'+str(only_number(v1)) + ',' + time_string)
            file.close()
            if stop==1:
                break


    if measure.get() == '5':
        while True:
```

```
        #for i in range(0,30):
            v = []
            v[0] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default))
            v[1] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default))
            v[2] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default))
            v[3] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default))
            v[4] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default))
            max=v[0]
            min=v[0]
            sum=v[0]
            for i in range(1, 4):
                v[i] = only_number(lan.SendQueryMsg('MEAS? UDC1', Timeout_default)) # prova senza only number
                sum=sum+v[i]
                if v[i]>max:
                    max=v[i]
                if v[i]<min:
                    min=v[i]
            media=sum/i
            #print(("{:.4e}".format(only_number(v1))),time_string)
            box_meas.delete(0, 3)
            box_meas.insert(END, i)
            named_tuple = time.localtime()  # get struct_time
            time_string = time.strftime("%H.%M.%S", named_tuple)
            print(("{:.4e}".format(only_number(v1))), time_string)
            i=i+1
            file = open(open_file(), 'a')
            file.write('\n'+str(min) + ',' +str(media)+','+str(max)+','+time_string)
            file.close()
            if i == 10+1:
                break
'''
def open_file():
    B_openfile['state'] = 'disabled'
    save_button['state'] = 'normal'

    filename = casella3.get()
    if prove_type.get() == 'No load':
        ext ='.VTO'
    if prove_type.get() == 'Locked rotor':
        ext ='.CTO'
    if prove_type.get() == 'Load':
        ext ='.CAR'
    if prove_type.get() == '                  ':
        messagebox.showinfo('Attention', 'Have to select the prove type')
    filename1 = filename + ext

    return filename1

def create_file():
```

```

    file = open(open_file(), 'w')
    file.write(targa[0]+',' + targa[1] + '\n' + targa[5]+','+targa[2]+
   ','+targa[3]+','+targa[4]+','+targa[8]+','+targa[7]+','+targa[6])
def put_prove():

    B_salve['state'] = 'disabled'
    B_save2['state'] = 'normal'
    supp = supply.get()
    conn = connection.get()
    freq = casella43.get()
    dataenote = casella4.get()

    file = open(open_file(), 'a')
    name1 = open_file()
    file.write('\n'+supp+','+freq+','+conn+','+name1[-3:len(name1)]+'\
   n'+dataenote)

def resist_amb():
    B_save2['state'] = 'disabled'
    B_save3['state'] = 'normal'
    file = open(open_file(), 'a')
    file.write('\n'+casella5.get()+','+casella6.get()+','+casella7.get
   ())

def resist_first():
    B_save3['state'] = 'disabled'
    B_stmisure['state'] = 'normal'
    B_endmisure['state'] = 'normal'
    file = open(open_file(), 'a')
    file.write('\n' + casella51.get() + ',' + casella61.get())

def resit_last():
    named_tuple = time.localtime()  # get struct_time
    time_string = time.strftime("%H.%M.%S", named_tuple)
    file = open(open_file(), 'a')
    file.write('\n' + '0' +','+ casella53.get() +',' + '0' + ',' + '0'
    + ','+ casella63.get()+','+'0'+','+'0'+','+'0'+','+'0'+','+
   time_string)

def resit_last30():
    named_tuple = time.localtime()  # get struct_time
    time_string = time.strftime("%H.%M.%S", named_tuple)
    file = open(open_file(), 'a')
    file.write('\n' +'0'+','+ casella54.get()+ ','+'0'+','+'0'+','+
   casella64.get()+','+'0'+','+'0'+','+'0'+','+'0'+','+ time_string)
    file.close()

def connection1():
    B_conn['state'] = 'disabled'
    B_openfile['state'] = 'normal'

    # Instantiation of the LAN communication class
    global lan
    lan = Lan(Timeout_default)

    # Connect
```

```
    IP = box_con.get()
    port = int(box_port.get())
    #print("IP?")
    #IP = input()
    #print("Port?")
    #port = int(input())
    if not lan.open(IP, port):
        return

def r_conn():
    global lan
    lan = Lan(Timeout_default)

    # Connect
    IP = '192.168.1.2'
    port = 3300
    if not lan.open(IP, port):
        return



'''
    # Send and receive commands
    while True:
        # print("Please enter the command (Exit with no input)")    # commented by Marco 21/07/2021
        print("Please enter the command:")
        command = input()
        # Exit if no input
        if command == "":
            break
        # If the command contains "?"
        if "?" in command:
            msgBuf = lan.SendQueryMsg(command, Timeout_default)
            print(msgBuf)
            # Send only
        else:
            lan.sendMsg(command)

    lan.close()
'''

messagebox.showinfo('welcome', 'Please make sure the LAN cable is connected before any action.')

#CONNECTION
connect = LabelFrame(window, text="Connection with LAN", font=('calibre', 12), bg='#cccccc')
connect.grid(row=0, column=0, padx=30, pady=10, rowspan=3, sticky='NW')
l_ip = Label(connect, text='IP', font=('calibre', 11), bg = '#cccccc')
l_ip.grid(row=0, column=0, padx=10, pady=5, sticky='W')
l_apath = Label(connect, text='(1)', font=('calibre', 11), bg='#cccccc', fg='red')
l_apath.grid(row=0, column=2, padx=20, pady=5, sticky='E')
l_porta = Label(connect, text='Porta', font=('calibre', 11), bg='#cccccc')
```

```
l_porta.grid(row=1, column=0, padx=10, pady=5, sticky='W')
box_con = Entry(connect, font=('calibre', 11), width=10)
box_con.grid(row=0, column=1)
box_con.insert(END, '192.168.1.2')
box_port = Entry(connect, font=('calibre', 11), width=10)
box_port.grid(row=1, column=1)
box_port.insert(END, '3300')
B_conn = Button(connect, text='Connect', font=('calibre', 11), height=1, width=8, command=connection1)
B_conn.grid(row=2, column=2, padx=10, pady=10)
#B_conn1= Button(connect,text = 'R_Conn', height=2, width=10, command=r_conn)
#B_conn1.grid(row=2, column=1, padx=10, pady=10)

#DATA
data = LabelFrame(window, text="Motor plate data ", font=('calibre', 12), bg='#cccccc')
data.grid(row=3, column=0, padx=30, pady=2, rowspan=18, sticky='NW')
ditta_label = Label(data, text='Company name', font=('calibre', 10), bg = '#cccccc')
ditta_label.grid(row=0, column=0, pady=5, sticky='W')
num_operation = Label(data, text='(3)', font=('calibre', 10), bg='#cccccc', fg='red')
num_operation.grid(row=0, column=5, padx=10, pady=10, sticky='SE')
box1 = Entry(data, font=('calibre', 11), width=8)
box1.grid(row=1, column=0)
tipo_label = Label(data, text='Model/Type', font=('calibre', 10), bg = '#cccccc')
tipo_label.grid(row=2, column=0, pady=5, sticky='W')
box2 = Entry(data, font=('calibre', 11), width=8)
box2.grid(row=3, column=0)
tensione_label = Label(data, text='Rated voltage(V)', font=('calibre', 10), bg = '#cccccc')
tensione_label.grid(row=4, column=0, pady=5, sticky='W')
box3 = Entry(data, font=('calibre', 11), width=8)
box3.grid(row=5, column=0)
corrente_label = Label(data, text='Rated current (A)', font=('calibre', 10), bg = '#cccccc')
corrente_label.grid(row=6, column=0, pady=5, sticky='W')
box4 = Entry(data, font=('calibre', 11), width=8)
box4.grid(row=7, column=0)
coll_label = Label(data, text='Winding connection [Y/D]', font=('calibre', 10), bg = '#cccccc')
coll_label.grid(row=8, column=0, pady=5, sticky='W')
box5 = Entry(data, font=('calibre', 11), width=8)
box5.grid(row=9, column=0)
power_label = Label(data, text='Rated power (W)', font=('calibre', 10), bg = '#cccccc')
power_label.grid(row=10, column=0, pady=5, sticky='W')
box6 = Entry(data, font=('calibre', 11), width=8)
box6.grid(row=11, column=0)
tensione_label = Label(data, text='Rated speed (rpm)', font=('calibre', 10), bg = '#cccccc')
tensione_label.grid(row=12, column=0, pady=5, sticky='W')
box7 = Entry(data, font=('calibre', 11), width=8)
box7.grid(row=13, column=0)
corrente_label = Label(data, text='Pole number', font=('calibre', 10),
```

```
    bg = '#cccccc')
corrente_label.grid(row=14, column=0, pady=5, sticky='W')
box8 = Entry(data, font=('calibre', 11), width=8)
box8.grid(row=15, column=0)
coll_label = Label(data, text='Rated frequency (Hz)', font=('calibre', 10), bg = '#cccccc')
coll_label.grid(row=16, column=0, pady=5, sticky='W')
box9 = Entry(data, width=8)
box9.grid(row=17, column=0)
save_button = Button(data, text='Save', font=('calibre', 11), height=1, width=8, command=save1, state='disabled')
save_button.grid(row=18, column=5, padx=10, pady=10, sticky='SE')
#casella_v1=Entry(window, width=8) mostrava il valore delle tensioni, per capire se il pc comunicava correttamnte col wattmetro
#casella_v1.grid(row=2,column=4)
#salve_button['state'] = 'disabled'

# CREATE SETTING OF THE WATTMETER
setting = LabelFrame(window, text="Setting of the wattmeter", font=('calibre', 12), bg='#cccccc')
setting.grid(row=0, column=1, padx=10, pady=10, rowspan=4, sticky='NW')
lab_coupling = Label(setting, text='Rectifier setting', font=('calibre', 10), bg='#cccccc')
lab_coupling.grid(row=0, column=0, padx=10, sticky='NW')
lab_wiring = Label(setting, text='Wiring type', font=('calibre', 10), bg='#cccccc')
lab_wiring.grid(row=0, column=1, padx=10, sticky='NW')
lab_display = Label(setting, text='Synchronization source', font=('calibre', 10), bg='#cccccc')
lab_display.grid(row=0, column=2, padx=10, sticky='NW')
coupling_range = StringVar()
coupling_range.set('                  ')
wiring_range = StringVar()
wiring_range.set('                    ')
display_range = StringVar()
display_range.set('                   ')
voltage_range = StringVar()
voltage_range.set('AUTO')
current_range = StringVar()
current_range.set('AUTO')
menu_coupling_range = OptionMenu(setting, coupling_range, "AC+DC", "AC")
menu_coupling_range.grid(row=0, column=0, padx=10, pady=20, sticky='SW')
menu_coupling_range.config(font=('calibre', 11), width=8)
menu_wiring_range = OptionMenu(setting, wiring_range, "Aron", "3 Wattmetri")
menu_wiring_range.grid(row=0, column=1, padx=10, pady=20, sticky='SW')
menu_wiring_range.config( font=('calibre', 11), width=8)
menu_display_range = OptionMenu(setting, display_range, "Voltage", "Current")
menu_display_range.grid(row=0, column=2, padx=10, pady=20, sticky='SW')
menu_display_range.config( font=('calibre', 11), width=8)
lab_Vport = Label(setting, text='Voltage range', font=('calibre', 10), bg='#cccccc')
```

```
lab_Vport.grid(row=1, column=0, padx=10, sticky='NW')
lab_Iport = Label(setting, text='Current range', font=('calibre', 10), bg='#cccccc')
lab_Iport.grid(row=1, column=1, padx=10, sticky='NW')
menu_voltage_range = OptionMenu(setting, voltage_range, 'AUTO', '15V', '30V', '60V', '150V', '300V', '600V', '1000V')
menu_voltage_range.grid(row=1, column=0, padx=10, pady=20, sticky='SW')
menu_voltage_range.config( font=('calibre', 11), width=8)
menu_current_range = OptionMenu(setting, current_range, 'AUTO', '200mA', '500mA', '1A', '2A', '5A', '10A', '20A', '50A')
menu_current_range.grid(row=1, column=1, padx=10, pady=20, sticky='SW')
menu_current_range.config( font=('calibre', 11), width=8)
#casella1 = Entry(setting, width=8)
#casella1.grid(row=1, column=0, pady=20, padx=10, sticky='SW')
#casella1.insert(END, 1.0)
#casella22 = Entry(setting, width=8)
#casella22.grid(row=1, column=1, pady=20, padx=10, sticky='SW')
#casella22.insert(END, 1.0)
create_button = Button(setting, text='Zero-Adjust.', font=('calibre', 11), height=1, width=8, command=Zeroadj)
create_button.grid(row=1, column=2, pady=20, padx=10, sticky='NW')
B_SETTT = Button(setting, text='Set', font=('calibre', 11), height=1, width=8, command=settings)
B_SETTT.grid(row=2, column=2, pady=20, padx=10, sticky='SW')

# FILE NAME
LB_filename = LabelFrame(window, text="File name", font=('calibre', 12), bg='#cccccc')
LB_filename.grid(row=4, column=1, padx=10, columnspan=3, sticky='W', rowspan=1)
B_openfile = Button(LB_filename, text='open file', font=('calibre', 11), height=1, width=8, command=open_file, state='disabled')
B_openfile.grid(row=0, column=6, padx=10, pady=5)
lab_type = Label(LB_filename, text='Test Type', font=('calibre', 10), bg='#cccccc')
lab_type.grid(row=0, column=0, padx=10, pady=5, sticky='N')
lab_insert = Label(LB_filename, text='File name without any format', font=('calibre', 10), bg='#cccccc')
lab_insert.grid(row=0, column=1, padx=10, pady=5, sticky='N')
prove_type = StringVar()
prove_type.set('                  ')
menu_prove_type = OptionMenu(LB_filename, prove_type, "No Load", 'Locked Rotor')
menu_prove_type.grid(row=0, column=0, padx=10, pady=25, sticky='S')
menu_prove_type.config(font=('calibre', 11), width=10)
casella3 = Entry(LB_filename, font=('calibre', 11), width=12)
casella3.grid(row=0, column=1, padx=10, pady=25, sticky='S')

# PROVE
LB_prove = LabelFrame(window, text="Test", font=('calibre', 12), bg='#cccccc')
LB_prove.grid(row=5, column=1, padx=10, columnspan=4, sticky='W')
B_salve = Button(LB_prove, text='Save', font=('calibre', 11), height=1, width=8, command=put_prove, state='disabled')
B_salve.grid(row=1, column=2, padx=10, pady=10, sticky='W')
```

```
lab_supply = Label(LB_prove, text='Supply', font=('calibre', 10), bg = '#cccccc')
lab_supply.grid(row=0, column=0, padx=10, sticky='WN')
lab_connection = Label(LB_prove, text='Winding connection', font=('calibre', 10), bg = '#cccccc')
lab_connection.grid(row=0, column=1, padx=10, sticky='WN')
supply = StringVar()
supply.set('                  ')
menu_supply = OptionMenu(LB_prove, supply, "SIN", 'PWM', "OQ")
menu_supply.grid(row=0, column=0, padx=10, pady=20, sticky='WS')
menu_supply.config(font=('calibre', 11), width=8)
connection = StringVar()
connection.set('                  ')
menu_conn = OptionMenu(LB_prove, connection, "Y", 'D')
menu_conn.grid(row=0, column=1, padx=10, pady=20, sticky='WS')
menu_conn.config(font=('calibre', 11), width=8)
lab_frequency = Label(LB_prove, text='Frequency', font=('calibre', 10), bg='#cccccc')
lab_frequency.grid(row=0, column=2, padx=10, sticky='N')
lab_notes = Label(LB_prove, text='Date and notes', font=('calibre', 10), bg='#cccccc')
lab_notes.grid(row=1, column=0, padx=10, sticky='N')
casella4 = Entry(LB_prove, font=('calibre', 11), width=20)
casella4.grid(row=1, column=0, padx=10, pady=20,columnspan=2, sticky='S')
casella43 = Entry(LB_prove, font=('calibre', 11), width=10)
casella43.grid(row=0, column=2, padx=10, pady=20, sticky='S')

# RESISTENZA ALLA TEMPERATURA AMBIENTE
LB_resamb = LabelFrame(window, text="Winding resistance at the reference temperature", font=('calibre', 12), bg='#cccccc')
LB_resamb.grid(row=6, column=1, padx=10, columnspan=3, sticky='W')
B_save2 = Button(LB_resamb, text='Save', font=('calibre', 11), height=1, width=8, command=resist_amb, state='disabled')
B_save2.grid(row=0, column=4, padx=10, pady=10, sticky='S')
lab_tension = Label(LB_resamb, text='DC voltage (V)', font=('calibre', 10), bg='#cccccc')
lab_tension.grid(row=0, column=0, padx=10, pady=5, sticky='N')
number_4 = Label(LB_resamb, text='(5)', font=('calibre', 10), bg='#cccccc', fg='red')
number_4.grid(row=0, column=4, padx=10, pady=10, sticky='NE')
lab_current = Label(LB_resamb, text='DC current (A)', font=('calibre', 10), bg='#cccccc')
lab_current.grid(row=0, column=1, padx=10, pady=5, sticky='N')
lab_temperature = Label(LB_resamb, text='Temperature (°C)', font=('calibre', 10), bg='#cccccc')
lab_temperature.grid(row=0, column=2, padx=10, pady=5, sticky='N')
casella5 = Entry(LB_resamb, font=('calibre', 11), width=8)
casella5.grid(row=0, column=0, padx=10, pady=30, sticky='S')
casella6 = Entry(LB_resamb, font=('calibre', 11), width=8)
casella6.grid(row=0, column=1, padx=10, pady=30, sticky='S')
casella7 = Entry(LB_resamb, font=('calibre', 11), width=8)
casella7.grid(row=0, column=2, padx=10, pady=30, sticky='S')

# RESISTENZA PRIMA PROVA
LB_resfirst = LabelFrame(window, text="Stator resistance at the beginning of the test", font=('calibre', 12), bg='#cccccc')
```

```
LB_resfirst.grid(row=7, column=1, padx=10, columnspan=2, sticky='W')
B_save3 = Button(LB_resfirst, text='Save', font=('calibre', 11), height=1, width=8, command=resist_first, state='disabled')
B_save3.grid(row=0, column=5, padx=10, pady=10)
lab_tension1 = Label(LB_resfirst, text='DC voltage (V)', font=('calibre', 10), bg='#cccccc')
lab_tension1.grid(row=0, column=0, padx=10, pady=10, sticky='N')
lab_current1 = Label(LB_resfirst, text='DC current (A)', font=('calibre', 10), bg='#cccccc')
lab_current1.grid(row=0, column=1, padx=10, pady=10, sticky='N')
casella51 = Entry(LB_resfirst, font=('calibre', 11), width=8)
casella51.grid(row=0, column=0, padx=10, pady=30, sticky='S')
casella61 = Entry(LB_resfirst, font=('calibre', 11), width=8)
casella61.grid(row=0, column=1, padx=10, pady=30, sticky='S')


#ACQUISIZIONE MISURE
LB_acq = LabelFrame(window, text="Acquisition", font=('calibre', 11), bg='#cccccc')
LB_acq.grid(row=0, column=4, padx=10, pady=10, rowspan=10, columnspan=4, sticky='NE')
lab_numesur = Label(LB_acq, text='Select numebr of consecutive measurements:', font=('calibre', 10), bg = '#cccccc')
lab_numesur.grid(row=0, column=0, padx=10, pady=10)
measure = StringVar()
measure.set('                  ')
menu_measure = OptionMenu(LB_acq, measure, '3', "5", '10', "15", '20')
menu_measure.grid(row=1, column=0, padx=10, sticky='N')
menu_measure.config(font=('calibre', 11), width=5)
lab_nume = Label(LB_acq, text='Meas.Nr:', font=('calibre', 10), bg='#cccccc')
lab_nume.grid(row=2, column=0, padx=10, pady=10, sticky='N')
box_meas = Entry(LB_acq, font=('calibre', 11), width=8)
box_meas.grid(row=2, column=0, padx=10, pady=30, sticky='S')
#B_start = Button(LB_acq, text='Starttt', command=acquisition, height=2, width=10)
#B_start.grid(row = 0, column = 1, padx=1, pady=5, sticky='W')
B_stmisure = Button(LB_acq, text='Start recording', font=('calibre', 11), command=acquisition, height=1, width=10, state='disabled')
B_stmisure.grid(row=1, column=1, padx=1, pady=5, sticky='W')
B_endmisure = Button(LB_acq, text='Save recording', font=('calibre', 11), command=stop, height=1, width=10, state='disabled')
B_endmisure.grid(row=2, column=1, padx=1, pady=5, sticky='W')
lab_waitmeas = Label(LB_acq, text='Select wait time:', font=('calibre', 10), bg='#cccccc')
lab_waitmeas.grid(row=3, column=0, padx=10, pady=10)
wait = StringVar()
wait.set('                  ')
menu_wait = OptionMenu(LB_acq, measure, '0.1sec', "0.25sec", '0.5sec', "1sec")
menu_wait.grid(row=4, column=0, padx=10, sticky='N')
menu_wait.config(font=('calibre', 11), width=5)

# VALORI EFFICACI
LB_rms = LabelFrame(LB_acq, text="Measured values: Total", font=('calibre', 10), bg='#cccccc')
LB_rms.grid(row=5, column=0, padx=10, pady=8, rowspan=1, columnspan=4,
```

```
    sticky='W')
lab_Vrms= Label(LB_rms, text='V rms (V):', font=('calibre', 10), bg = '#cccccc')
lab_Vrms.grid(row=0, column=0, padx=10, pady=10)
lab_Irms= Label(LB_rms, text='I rms (A):', font=('calibre', 10), bg = '#cccccc')
lab_Irms.grid(row=0, column=1, padx=10, pady=10)
lab_Prms= Label(LB_rms, text='P rms (kW or W):', font=('calibre', 10), bg = '#cccccc')
lab_Prms.grid(row=0, column=2, padx=10, pady=10)
box_Vrms= Entry(LB_rms, font=('calibre', 11), width=8)
box_Vrms.grid(row=1, column=0, padx=10, sticky='W')
box_Irms= Entry(LB_rms, font=('calibre', 11), width=8)
box_Irms.grid(row=1, column=1, padx=10, sticky='W')
box_Prms= Entry(LB_rms, font=('calibre', 11), width=8)
box_Prms.grid(row=1, column=2, padx=10, sticky='W')

# VALORI FONDAMENTALI
LB_fond = LabelFrame(LB_acq, text="Measured values: Fundamental (1st harm.)", font=('calibre', 12), bg='#cccccc')
LB_fond.grid(row=6, column=0, padx=10, pady=8, rowspan=1, columnspan=4, sticky='W')
lab_Vfond= Label(LB_fond, text='V rms (V):', font=('calibre', 10), bg='#cccccc')
lab_Vfond.grid(row=0, column=0, padx=10, pady=10)
lab_Ifond= Label(LB_fond, text='I rms (A):', font=('calibre', 10), bg='#cccccc')
lab_Ifond.grid(row=0, column=1, padx=10, pady=10)
lab_Pfond= Label(LB_fond, text='P rms (kW or W):', font=('calibre', 10), bg='#cccccc')
lab_Pfond.grid(row=0, column=2, padx=10, pady=10)
box_Vfond= Entry(LB_fond, font=('calibre', 11), width=8)
box_Vfond.grid(row=1, column=0, padx=10, sticky='W')
box_Ifond= Entry(LB_fond, font=('calibre', 11), width=8)
box_Ifond.grid(row=1, column=1, padx=10, sticky='W')
box_Pfond= Entry(LB_fond, font=('calibre', 11), width=8)
box_Pfond.grid(row=1, column=2, padx=10, sticky='W')

# RESISTENZA LAST PROVA
LB_reslast = LabelFrame(window, text="Stator resistance at the end of the test and 30sec. after", font=('calibre', 12), bg='#cccccc')
LB_reslast.grid(row=6, column=4, padx=10, columnspan=3, sticky='W')
B_salve3 = Button(LB_reslast, text='Save', font=('calibre', 11), height=1, width=8, command=resit_last, state='disabled')
B_salve3.grid(row=0, column=5, padx=10, pady=10)
B_salve4 = Button(LB_reslast, text='Save', font=('calibre', 11), height=1, width=8, command=resit_last30, state='disabled')
B_salve4.grid(row=1, column=5, padx=10, pady=10)
lab_tension2 = Label(LB_reslast, text='DC voltage (V)', font=('calibre', 10), bg='#cccccc')
lab_tension2.grid(row=0, column=0, padx=10, pady=10, sticky='N')
lab_current2 = Label(LB_reslast, text='DC current (A)', font=('calibre', 10), bg='#cccccc')
lab_current2.grid(row=0, column=1, padx=10, pady=10, sticky='N')
casella53 = Entry(LB_reslast, width=8)
casella53.grid(row=0, column=0, padx=10, pady=30, sticky='S')
casella63 = Entry(LB_reslast, width=8)
```

```
casella63.grid(row=0, column=1, padx=10, pady=30, sticky='S')
lab_tension3 = Label(LB_reslast, text='DC voltage (V)', font=('calibre', 10), bg='#cccccc')
lab_tension3.grid(row=1, column=0, padx=10, pady=10, sticky='N')
lab_current3 = Label(LB_reslast, text='DC current (A)', font=('calibre', 10), bg='#cccccc')
lab_current3.grid(row=1, column=1, padx=10, pady=10, sticky='N')
casella54 = Entry(LB_reslast, font=('calibre', 11), width=8)
casella54.grid(row=1, column=0, padx=10, pady=30, sticky='S')
casella64 = Entry(LB_reslast, font=('calibre', 11), width=8)
casella64.grid(row=1, column=1, padx=10, pady=30, sticky='S')

'''
# VALORI RMS MEDI
LB_rmsm = LabelFrame(LB_acq, text=" Valori efficaci medi ", bg='#cccccc')
LB_rmsm.grid(row=7, column=0, padx=10, pady=8, rowspan=1, columnspan=4, sticky='W')
lab_Vrmsm= Label(LB_rmsm, text='V rms:', font=('calibre', 12), bg = '#cccccc')
lab_Vrmsm.grid(row=0, column=0, padx=10, pady=10)
lab_Irmsm= Label(LB_rmsm, text='I rms:', font=('calibre', 12), bg = '#cccccc')
lab_Irmsm.grid(row=0, column=1, padx=10, pady=10)
lab_Prmsm= Label(LB_rmsm, text='P rms:', font=('calibre', 12), bg = '#cccccc')
lab_Prmsm.grid(row=0, column=2, padx=10, pady=10)
box_Vrmsm= Entry(LB_rmsm, width=8)
box_Vrmsm.grid(row=1, column=0, padx=10, sticky='W')
box_Irmsm= Entry(LB_rmsm, width=8)
box_Irmsm.grid(row=1, column=1, padx=10, sticky='W')
box_Prmsm= Entry(LB_rmsm, width=8)
box_Prmsm.grid(row=1, column=2, padx=10, sticky='W')

# VALORI FONDAMENTALI MEDI
LB_fondm= LabelFrame(LB_acq, text="Valori medi di fondamentle", bg='#cccccc')
LB_fondm.grid(row=8, column=0, padx=10, pady=8, rowspan=1, columnspan=4, sticky='W')
lab_Vfondm= Label(LB_fondm, text='V rms:', font=('calibre', 12), bg = '#cccccc')
lab_Vfondm.grid(row=0, column=0, padx=10, pady=10)
lab_Ifondm= Label(LB_fondm, text='I rms:', font=('calibre', 12), bg = '#cccccc')
lab_Ifondm.grid(row=0, column=1, padx=10, pady=10)
lab_Pfondm= Label(LB_fondm, text='P rms:', font=('calibre', 12), bg = '#cccccc')
lab_Pfondm.grid(row=0, column=2, padx=10, pady=10)
box_Vfondm= Entry(LB_fondm, width=8)
box_Vfondm.grid(row=1, column=0, padx=10, sticky='W')
box_Ifondm= Entry(LB_fondm, width=8)
box_Ifondm.grid(row=1, column=1, padx=10, sticky='W')
box_Pfondm= Entry(LB_fondm, width=8)
box_Pfondm.grid(row=1, column=2, padx=10, sticky='W')
'''
create_button = Button(text='Save file', height=2, width=10, state='disabled')  #aggiungere un comando per decidere dove salvare il
```

```
    tutto
create_button.grid(row=7, column=5, padx=1, pady=5, sticky='W')

quit_button = Button(text='Exit', command = window.quit, height=2, width=10)
quit_button.grid(row=7, column=6, padx=1, pady=5, sticky='W')

'''
#Create a canvas
canvas= Canvas(window, width= 100, height= 50,bg='#cccccc')
canvas.grid(row=0, column=6, padx=1, pady=5, sticky='W',rowspan=10)

#Load an image in the script
img= (Image.open("marchio_e_logotipo_politecnico_di_torino_full.png"))

#Resize the Image using resize method
resized_image = img.resize((200, 100), Image.ANTIALIAS)
new_image = ImageTk.PhotoImage(resized_image)

#Add image to the Canvas Items
canvas.create_image(10,10, anchor=NW, image=new_image)

'''
window.mainloop()
```

8.2 B. Elaboration tool Python codes

```
from tkinter import *
from tkinter import messagebox
from tkinter import filedialog
from tkinter import ttk
from math import *
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg, NavigationToolbar2Tk)
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from matplotlib import pyplot as plt
from fpdf import FPDF
import cmath

global indice_grap,Vnom,Inom,ColnNom,FreqNom,Npoli
indice_grap = 0   #indice che ci permeterrà di dividere usare lo stesso tasto con differenti funzioni per plottare i dati
indice_inter = 0  #indice che ci permeterrà di dividere usare lo stesso tasto con differenti funzioni per usare l'interpolazione

#create the main window

window = Tk()  # create the main window
window.configure(bg = '#cccccc') # background color
window.attributes('-fullscreen', True)
window.title('Motor analysis')
```

```
#window.geometry('1280x800')
filename1='ciao'


   '''
   filewin = Toplevel(window)
   button = Button(filewin, text="Do nothing button")
   button.pack()
   '''

def reset():    # create a function to reset the main program
   global window, noload_button, lockrotor_button

   window.destroy()  # destroy the current window to create another
    one
   window = Tk()
   window.configure(bg='#cccccc')
   window.attributes('-fullscreen', True)
   window.title('Motor analysis')

   # NO LOAD BUTTON
   #noload_button['state'] = 'normal'
   noload_button = Button(text='No Load', command=noload, height=2,
    width=10)
   noload_button.grid(row=0, column=0, padx=10, pady=5, sticky='W')

   # LOCKED ROTOR BUTTON
   lockrotor_button = Button(text='Locked Rotor', height=2, width=10,
    command=locked)
   lockrotor_button.grid(row=0, column=1, padx=1, pady=5, sticky='W')

   # Y=f(x) BUTTON
   y_button = Button(text='Y=f(x)', height=2, width=10, command=
    caratteristic)
   y_button.grid(row=0, column=2, padx=10, pady=5, sticky='W')

   # RESET BUTTON
   reset_button = Button(text='Reset', height=2, width=10, command=
    reset)
   reset_button.grid(row=0, column=3, padx=1, pady=5, sticky='W')

   # SPLIT FILE BUTTON
   splitfile_button = Button(text='Split File', height=2, width=10,
    command=donothing)
   splitfile_button.grid(row=0, column=4, padx=10, pady=5, sticky='W')

   # EXIT BUTTON
   quit_button = Button(text='Exit', command=window.quit, height=2,
    width=10)
   quit_button.grid(row=0, column=5, padx=1, pady=5, sticky='W')

def noload():
   noload_button['state'] = 'disabled'
   global indice_grap,Vnom,Inom,ColnNom,FreqNom,Npoli,Pnom,Ngiri,
    FreqPrv,Alim,CollPrv,VM,Im,PTM,RfaseInPrv,RFaseFiPrv,I0,Pferro,R0,
    X0  #serve per far riuscire il reset, perch  prima dava problemi
    con il plot
```

```
indice_grap=0
indice_inter=0
global filename1

def PDF():
    pdf = FPDF()

    # Add a page
    pdf.add_page()

    # set style and size of font
    # that you want in the pdf
    pdf.set_font("helvetica", 'B', size=20)
    pdf.image('polito_logo_2021_blu.jpg', 10, 1, 40)

    # create a cell
    pdf.cell(200, 10, txt="NO LOAD TEST", ln=1, align='C')

    # add another cell
    pdf.set_font("helvetica", '', size=10)
    # pdf.set_text_color(169,169,169)
    pdf.cell(200, 10, txt="Path and file name:   " + window.filename, ln=1, align='C')
    pdf.cell(100, 10, txt='Motor type: ' + model)
    pdf.cell(100, 10, txt='Company: ' + company, ln=1)
    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(100, 8, txt='Motor plate data', ln=1)
    pdf.set_font("helvetica", '', size=10)
    # pdf.set_text_color(169, 169, 169)
    pdf.cell(50, 5, txt='Power [kW]:  ' + str(Pnom))
    pdf.cell(50, 5, txt='Voltage [V]:  ' + str(Vnom))
    pdf.cell(50, 5, txt='Current [A]:  ' + str(Inom), ln=1)
    pdf.cell(50, 5, txt='Connection:  ' + ColnNom)
    pdf.cell(50, 5, txt='Frequency [Hz]:  ' + str(FreqNom))
    pdf.cell(50, 5, txt='Speed [rpm]:  ' + str(Ngiri), ln=1)
    pdf.cell(50, 5, txt='Poles number:  ' + str(Npoli), ln=1)
    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(100, 8, txt='Test conditions', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(50, 5, txt='Test connection:  ' + CollPrv)
    pdf.cell(50, 5, txt='Test supply:  ' + Alim, ln=1)
    pdf.cell(50, 5, txt='Test frequency [Hz]:  ' + str(FreqPrv))
    pdf.cell(50, 5, txt='Reference voltage [V] at 50 [Hz]:  ' + str(Vnom), ln=1)
    pdf.cell(50, 5, txt='Test data and notes:  '+ note, ln=1)
    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(100, 8, txt='Test results', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(25, 8, txt='Vline [V]', border=1, align='C')
    pdf.cell(25, 8, txt='Eline [V]', border=1, align='C')
    pdf.cell(25, 8, txt='Iline [A]', border=1, align='C')
    pdf.cell(25, 8, txt='cosfi', border=1, align='C')
    pdf.cell(25, 8, txt='Ptot [W]', border=1, align='C')
    pdf.cell(25, 8, txt='Pcu [W]', border=1, align='C')
    pdf.cell(25, 8, txt='Piron+mec [W]', border=1, align='C', ln=1)
    for i in range(0, 10):
        pdf.cell(25, 5, txt=f'{round(VM[i],3)}', border=1, align='C')
```

```
            pdf.cell(25, 5, txt=f'{round(EM[i],3)}', border=1, align='C')
            pdf.cell(25, 5, txt=f'{round(Im[i],3)}', border=1, align='C')
            pdf.cell(25, 5, txt=f'{round(CFM[i],3)}', border=1, align='C')
            pdf.cell(25, 5, txt=f'{round(PTM[i],3)}', border=1, align='C')
            pdf.cell(25, 5, txt=f'{round(PCM[i],3)}', border=1, align='C')
            pdf.cell(25, 5, txt=f'{round(PNM[i],3)}', border=1, align='C', ln=1)
    pdf.cell(0, 5, txt='Stator resistance at the reference temperature of ' + temp + ' = ' + str(round((RfaseAmb), 3)) + ' [Ohm]', ln=1)
    pdf.cell(0, 5, txt='Stator resistance at start time = ' + str(round((RfaseInPrv),3)) + ' [Ohm]', ln=1)
    pdf.cell(0, 5, txt='Stator resistance at stop time = ' + f'{"{:.2e}".format(RFaseFiPrv)}' + ' [Ohm]', ln=1)
    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(0, 8, txt='No load parameters at the test frequency and at the reference voltage:', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(0, 5, txt='No load current:' + f'{round(I0, 3)} '+ '[A]', ln=1)
    pdf.cell(0, 5, txt='Iron losses:' + f'{round(Pferro, 3)} '+ '[W]', ln=1)
    pdf.cell(0, 5, txt='Mechanical losses: '+ f'{round((param2[0]),3)} '+ '[W]', ln=1)
    pdf.cell(0, 5, txt='Equivalent iron losses resistance:' + f'{round(R0, 3)} '+ '[Ohm]', ln=1)
    pdf.cell(0, 5, txt='Stator reactance:' + f'{round(X0, 3)} '+ '[Ohm]', ln=1)
    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(0, 8, txt='Coeffcients of the fitting polinomial equation:', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(25, 8, txt='Grade', border=1, align='C')
    pdf.cell(25, 8, txt='E = f(V)', border=1, align='C')
    pdf.cell(25, 8, txt='I0 = f(E)', border=1, align='C')
    pdf.cell(25, 8, txt='P0 = f(E)', border=1, align='C', ln=1)
    for i in range(0, 10):
        pdf.cell(25, 5, txt=f'{i}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{"{:.3e}".format(pino[i])}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{"{:.3e}".format(param1[i])}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{"{:.3e}".format(param2[i])}', border=1, align='C', ln=1)

    # save the pdf with name .pdf
    pdf.output("NO_LOAD.pdf")

def file():
    window.filename = filedialog.askopenfilename( initialdir='/Users/Federico/Desktop/LM_Tesi/programma2', title='select a file', filetypes=(('VTO files', '*.VTO'), ('VTO files', '*.VTO'))) # nel primo VTO c'era CTO
    # tex = open(window.filename, 'r')
```

```
    mytext.insert(END, window.filename)
    # tex.close()
    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


    mylist = []  # define an empty list

    with open(window.filename, 'r') as file:  # usare 'as file' per rendere il programma globale per qualunque file aperto!

        for line in file:

            for word in line.split(','):
                mylist.append(word)

    file.close()

    # definiamo il numero delle misure

    # define all the parametres based on the txt file

    company = mylist[0]
    model = mylist[1]
    Pnom = float(mylist[2])
    Vnom = float(mylist[3])
    Inom = float(mylist[4])
    ColnNom = mylist[5]
    FreqNom = float(mylist[6])
    Npoli = int(mylist[7])
    Ngiri = float(mylist[8])
    Alim = mylist[9]
    FreqPrv = float(mylist[10])
    CollPrv = mylist[11]
    Prv = mylist[12]
    note = mylist[13]
    Vdc0 = float(mylist[14])
    Idc0 = float(mylist[15])
    Tamb = mylist[16]
    Vdc1 = float(mylist[17])
    Idc1 = float(mylist[18])
    box1a.insert(END, company)
    box2a.insert(END, model)
    box3a.insert(END, Pnom)
    box4a.insert(END, Vnom)     # non posso convertire numerico se voglio aggiugere la stringa dell'unità di misura
    box5a.insert(END, Inom)
    box6a.insert(END, CollPrv)
    box7a.insert(END, FreqNom)
    box8a.insert(END, Npoli)
    box9a.insert(END, Ngiri)

    # inserimento delle note
    mynote.insert(END, note)

```

```
    # completamento della secodna tabella
    box1b.insert(END, Alim)
    box2b.insert(END, Prv)
    box3b.insert(END, CollPrv)
    box4b.insert(END, FreqPrv)

    #sresistance_temp = LabelFrame(window, bg='#d3d3d3', text='
Stator resistance at' + temp + '°C [Ohm]')
    #sresistance_temp.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')

    sresistance_temp1 = LabelFrame(window, bg='#d3d3d3', text="
Stator resistance [Ohm] at \ntemperature [°C] of " + Tamb)
    sresistance_temp1.grid(row=5, column=0, padx=10, pady=5,
columnspan=6, sticky='W')
    box_t=Entry(sresistance_temp1, width=8, bg='#FFFFFF')
    box_t.grid(row=0, column=0)

    col = 1
    #completamento della RESISTENZA DI STATORE nelle varie
    if ColnNom == 'Y':
        col = sqrt(3)
    elif ColnNom == 'D':
        col = 1
    else:
        messagebox.showwarning("warning", "Error with the
configuration parameter!")
    IfaseNom = Inom * col / sqrt(3)
    RfaseAmb = 1.5 * Vdc0 / Idc0 / pow(col, 2)
    RfaseInPrv = 1.5 * Vdc1 / Idc1 / pow(col, 2)
    rappFrq = FreqPrv / FreqNom
    DeltaV = col * RfaseInPrv * IfaseNom
    VPrvNom = DeltaV + rappFrq * (Vnom - DeltaV)

    #completamento della terza tabella
    #box4c.insert(END, RfaseAmb)
    box_t.insert(END, round(RfaseAmb, 7))

    # completamento della terza tabella
    box4d.insert(END, round(RfaseInPrv, 7))

    # completamento tabella misure
    i=1
    j=0
    VM=[]
    for tensione in mylist[20:-20:10]:
        VM.append(float(tensione))
        i=i+1
    Ndati=int(i-1)
    i=1
    Im=[]
    for current in mylist[23:-20:10]:
        Im.append(float(current))
        i=i+1
    i = 1
    PTM=[]
    for power in mylist[26:-20:10]:
```

```
        PTM.append(float(power))
        i=i+1
    i = 1
    temp_mes = []
    for tempo in mylist[28:-1:10]:
        temp_mes.append(tempo)
        i = i + 1
    temp_mes.append(mylist[-1])


    tp = []
    k = 0
    for tempo in temp_mes:
        k = 0

        for a in tempo.split('.'):

            # print(tp)
            if k == 0:
                hour = int(a)
            elif k == 1:
                minut = int(a)
            else:
                sec = int(a)
            k = k + 1
        som = sec + 60 * minut + 3600 * hour
        tp.append(som)  #   tipo int


    #print(Ndati)

    # completamento della tabella delle misurazioni con i calcoli
    Rad3 = sqrt(3)
    Rfase2 = 1.5 * float(mylist[-19]) / float(mylist[-16]) / pow(col, 2)
    Rfase3 = 1.5 * float(mylist[-9]) / float(mylist[-6]) / pow(col, 2)
    RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])
    #RappTP=2
    print(RappTP)
    RFaseFiPrv = RappTP * (Rfase2 - Rfase3) + Rfase3
    RappRT = (RFaseFiPrv - RfaseInPrv) / (tp[-3] - tp[0])
    print(RFaseFiPrv)
    print(RappRT)

    # completamento della terza tabella
    box4e.insert(END, round(RFaseFiPrv, 7))

    i=0
    RF=[]
    IFM=[]
    PCM=[]
    PNM=[]
    CFM=[]
    EM=[]

    for i in range(Ndati):
```

```

        RF.append(RfaseInPrv+(tp[i] - tp[0]) * RappRT)
        VFM = VM[i] / col
        IFM = Im[i] * col / Rad3
        PCM.append(3 * RF[i] * pow(IFM, 2))
        PNM.append(PTM[i] - PCM[i])
        CFM.append(PTM[i] / Rad3 / VM[i] / Im[i])
        SFM = (sqrt(1 - pow(CFM[i], 2)))
        EFM = sqrt(pow((VFM - RF[i] * IFM * CFM[i]), 2) + pow((RF[i] * IFM * SFM), 2))
        #EFM = (sqrt(pow((VFM[i] - Rfase[i] * IFM[i] * CFM[i]), 2) + pow((Rfase[i] * IFM[i] * SFM), 2)))
        EM.append(EFM * col)



        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),))
        scroll.configure(command=tree.yview)

    # abilitare il comando emf
    def emf():
        plot_button['state'] = 'normal'
        #fitting_button['state']='disabled'
        q = 1
        i = 0
        for i in range(Ndati):
            tree1.insert(parent='', index='end', iid=i, text=i + 1, values=(round(VM[i], 3), round(EM[i], 3)))
        io['state']='normal'
        alfio['state'] = 'disabled'

    alfio.configure(command=emf)

    def i0():
        plot_button['state'] = 'normal'
        #fitting_button['state'] = 'disabled'
        q = 2
        i = 0
        tree2 = ttk.Treeview(analysis)

        tree2['columns'] = ('E.m.f [V]', 'I0 [A]')

        # Formate our columns
        tree2.column('#0', width=25, minwidth=40)
        tree2.column('E.m.f [V]', width=100, minwidth=40)
        tree2.column('I0 [A]', width=100, minwidth=40)
        tree2.grid(row=0, column=0)

        # Create Headings
        tree2.heading('#0', text='N°')
        tree2.heading('E.m.f [V]', text='E.m.f [V]')
        tree2.heading('I0 [A]', text='I0 [A]')
        for i in range(Ndati):
            tree2.insert(parent='', index='end', iid=i, text=i + 1, values=(round(EM[i], 3), round(Im[i], 3)))
```

```
        po['state'] = 'normal'
        io['state'] = 'disabled'

    io.configure(command=i0)

    def P0():
        plot_button['state'] = 'normal'
        #fitting_button['state'] = 'disabled'
        q = 3
        i = 0
        tree3 = ttk.Treeview(analysis)

        tree3['columns'] = ('E.m.f [V]', 'P0 [W]')

        # Formate our columns
        tree3.column('#0', width=25, minwidth=40)
        tree3.column('E.m.f [V]', width=100, minwidth=40)
        tree3.column('P0 [W]', width=100, minwidth=40)
        tree3.grid(row=0, column=0)

        # Create Headings
        tree3.heading('#0', text='N°')
        tree3.heading('E.m.f [V]', text='E.m.f [V]')
        tree3.heading('P0 [W]', text='P0 [W]')
        for i in range(Ndati):
            tree3.insert(parent='', index='end', iid=i, text=i + 1, values=(round(EM[i], 3), round(PNM[i], 3)))
        po['state'] = 'disabled'

    po.configure(command=P0)


    def grap():

        # Data for plotting
        #t = np.arange(0.0, 2.0, 0.01)
        #s = 1 + np.sin(2 * np.pi * t)
        global indice_grap
        global indice_inter

        if indice_grap == 0:
            plt.grid(True)
            foto = Figure(figsize=(6, 4))
            anna = foto.add_subplot(111)
            anna.plot(VM, EM, 'o', color='red')
            # setting x and y axis range
            plt.ylim(0, 450)
            plt.xlim(0, 450)
            anna.set(xlabel='V [V]', ylabel= 'E.m.f [V]', title='Company: ' + company + '        Type: ' + model)
            matteo = FigureCanvasTkAgg(foto, master=window)
            matteo.draw()
            matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            indice_inter = 0

        if indice_grap == 1:
```

```
            foto1 = Figure(figsize=(6, 4))
            anna1 = foto1.add_subplot(111)
            anna1.plot(EM, Im, 'o', color='red')
            # setting x and y axis range
            plt.ylim(0, 25)
            plt.xlim(0, 450)
            anna1.set(xlabel='E.m.f [V]', ylabel='I0 [A]', title='Company: ' + company + '           Type: ' + model)
            matteo = FigureCanvasTkAgg(foto1, master=window)
            matteo.draw()
            matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            indice_inter = 1

        if indice_grap == 2:
            f = Figure(figsize=(6, 4))
            a = f.add_subplot(111)
            a.plot(EM, PNM, 'o', color='red')
            # setting x and y axis range
            plt.ylim(0, 750)
            plt.xlim(0, 450)
            a.set(xlabel='E.m.f [V] ', ylabel='P0 [W]', title='Company: ' + company + '           Type: ' + model)
            matteo = FigureCanvasTkAgg(f, master=window)
            matteo.draw()
            matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            indice_inter = 2

        indice_grap = indice_grap+1
        #indice_inter = indice_inter + 1
        plot_button['state'] = 'disabled'
        #fitting_button['state'] = 'normal'



    plot_button.configure(command=grap)
    #indice_inter=0
    def interpolation():
        global param2, param1, pino

        inserimento = grades.get()
        print(inserimento)
        coeff = []
        for w in inserimento.split(','):
            coeff.append(w)

        lung = len(coeff)
        grades_input.delete(0, 20) #cancella i gradi in ingresso ogni volta che effettuimo il fitting

        global indice_inter

        def test(x, a):
            return a * x

        def test3(x, a, b, c, d, e, f, g, h, i, l):
```

```python

            a1 = 0
            b1 = 0
            c1 = 0
            d1 = 0
            e1 = 0
            f1 = 0
            g1 = 0
            h1 = 0
            i1 = 0
            l1 = 0
            for indice in range(lung):

                if int(coeff[indice]) == 0:
                    a1 = 1
                if int(coeff[indice]) == 1:
                    b1 = 1
                if int(coeff[indice]) == 2:
                    c1 = 1
                if int(coeff[indice]) == 3:
                    d1 = 1
                if int(coeff[indice]) == 4:
                    e1 = 1
                if int(coeff[indice]) == 5:
                    f1 = 1
                if int(coeff[indice]) == 6:
                    g1 = 1
                if int(coeff[indice]) == 7:
                    h1 = 1
                if int(coeff[indice]) == 8:
                    i1 = 1
                if int(coeff[indice]) == 9:
                    l1 = 1

            if a1 == 0:
                a = 0
            if b1 == 0:
                b = 0
            if c1 == 0:
                c = 0
            if d1 == 0:
                d = 0
            if e1 == 0:
                e = 0
            if f1 == 0:
                f = 0
            if g1 == 0:
                g = 0
            if h1 == 0:
                h = 0
            if i1 == 0:
                i = 0
            if l1 == 0:
                l = 0

            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)

        if indice_inter == 0:
            x = np.array(VM)
            y = np.array(EM)
            #param, param_cov = curve_fit(test, x, y)
            #ans = param[0] + x
            param, _ = curve_fit(test, x, y)
            ans = param[0] * x
            #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)
            foto = Figure(figsize=(6, 4))
            anna = foto.add_subplot(111)
            anna.plot(x, y, 'o', color='red', label="data")
            anna.plot(x, ans, '--', color='blue', label="optimized data")
            plt.ylim(0, 450)
            plt.xlim(0, 450)
            anna.set(xlabel='V [V]', ylabel='E.m.f [V]', title='Company: ' + company + '          Type: ' + model)
            intr = FigureCanvasTkAgg(foto, master=window)
            intr.draw()
            intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            pino = []
            #fitting_button['state'] = 'disabled'
            for i in range(0, 10):
                if i==1:
                    pino.append(param[0])
                else:
                    pino.append(0.0)

                treep.insert(parent='', index='end', iid=i, text=i, values="{:.2e}".format(pino[i]))


        if indice_inter == 1:
            x = np.array(EM)
            y = np.array(Im)
            param1, _ = curve_fit(test3, x, y)
            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)
            print("funcion coefficients:")
            print(param1)

            for index in range(0, 10):
                if param1[index] == 1.00000000e+00:
                    param1[index] = 0

            print('coefficienti con inserivmento dei dati FILTRATI:')
            print(param1)
```

```
            x1 = np.linspace(0, max(x) + 100, 500)
            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)
            foto6 = Figure(figsize=(6, 4))
            aaa = foto6.add_subplot(111)
            aaa.plot(x, y, 'o', color='red', label="data")
            aaa.plot(x1, y1, '--', color='blue', label="optimized data")
            plt.xlim([0, max(x) + 100])
            plt.ylim([0, max(y) + 10])
            #plt.ylim(0, 25)
            #plt.xlim(0, 450)
            aaa.set(xlabel='E.m.f [V]', ylabel='I0 [A]', title ='Company: ' + company + '          Type: ' + model)
            intr = FigureCanvasTkAgg(foto6, master=window)
            intr.draw()
            intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')

            #for record in treep.get_children():
             #  treep.delete(record)

            for i in range(0, 10):
                treep.set(i, column='#2', value="{:.2e}".format(param1[i]))

        if indice_inter == 2:
            x = np.array(EM)
            y = np.array(PNM)
            param2, _ = curve_fit(test3, x, y)
            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)
            print("funcion coefficients:")
            print(param2)

            for index in range(0, 10):
                if param2[index] == 1.00000000e+00:
                    param2[index] = 0

            print('coefficienti con inserivmento dei dati FILTRATI:')
            print(param2)
            x1 = np.linspace(0, max(x) + 100, 500)
            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)
            foto6 = Figure(figsize=(6, 4))
            aaa = foto6.add_subplot(111)
            aaa.plot(x, y, 'o', color='red', label="data")
            aaa.plot(x1, y1, '--', color='blue', label="optimized data
```

```
")
            plt.xlim([0, max(x) + 100])
            plt.ylim([0, max(y) + 10])
            # plt.ylim(0, 25)
            # plt.xlim(0, 450)
            aaa.set(xlabel='E.m.f [V]', ylabel='P0 [W]', title='Company: ' + company + '          Type: ' + model)
            intr = FigureCanvasTkAgg(foto6, master=window)
            intr.draw()
            intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            i=0
            for inde in range(0, 10):
                treep.set(inde, column='#3', value="{:.2e}".format(param2[inde]))
            #fitting_button['state'] = 'disabled'
            Nload_button['state'] = 'normal'

        #indice_inter=indice_inter+1

    fitting_button.configure(command=interpolation)

    def NLparameters():
        global param2
        casella1.insert(END, 'I0 [A]')          # No load current
        casella3.insert(END, 'Piron [W]')       # Iron losses
        casella5.insert(END, 'Pmecc [W]')       # Mechanical losses
        casella7.insert(END, 'R0 [Ohm]')        # Equivalent iron losses resistance
        casella9.insert(END, 'X0 [Ohm]')        # Stator reactance

        E0 = VPrvNom * pino[1]
        I0 = 0
        P0 = 0
        j=0
        for j in range(0, 10):

            I0 = (I0 + param1[j] * np.power(E0, j))
            print(param1[j])
            print(I0)
            P0 = (P0 + param2[j] * np.power(E0, j))

        Pferro = P0 - param2[0]
        Z = E0 * sqrt(3) / (col * col * I0)
        g = Pferro / 3 / pow(E0 / col, 2)
        y = 1 / Z
        B = sqrt(np.power(y, 2) - np.power(g, 2))
        R0 = 1 / g
        X0 = 1 / B

        casella2.insert(END, '{:.2e}'.format(I0))
        casella4.insert(END, '{:.2e}'.format(Pferro))
        casella6.insert(END, '{:.2e}'.format(param2[0]))
        casella8.insert(END, '{:.2e}'.format(R0))
        casella11.insert(END, '{:.2e}'.format(X0))
        Nload_button['state'] = 'disabled'
        report['state'] = 'normal'
```

```

      # Create the .par file
      filename1 = company + '.PAR'
      file = open(filename1, 'w')
      file.write('VTO' + '\n' + company + ',' + model + str(Vnom) + ',' + str(Inom) + ',' + str(FreqNom) + ',' + str(Npoli) + '\n' + ColnNom + '\n' + str(param1[0]))
      for i in range(1, 10):
         file.write(',' + str(param1[i]))
      file.write('\n' + str(param2[0]))
      for i in range(1,10):
         file.write(',' + str(param2[i]))
      file.write('\n' + CollPrv)
      file.close()


   Nload_button.configure(command=NLparameters)



#Crete the MENU BAR

menubar = Menu(window)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Open file", command=file)
filemenu.add_command(label="Close", command=window.quit)
menubar.add_cascade(label="File", menu=filemenu)
window.config(menu=menubar)

# CREATE MOTOR PLATE DATA
data = LabelFrame(window, text="Motor plate data", bg='#d3d3d3')
data.grid(row=3, column=0,padx=10,pady=5, columnspan=6,sticky='W')
# INSIDE THE LABEL
box1 = Entry(data, width=8, bg='#E0E0E0')
box1.grid(row=0, column=0)
box1.insert(END, ' Company')
box2 = Entry(data, width=8, bg='#E0E0E0')
box2.grid(row=1, column=0)
box2.insert(END, ' Type')
box3 = Entry(data, width=8, bg='#E0E0E0')
box3.grid(row=2, column=0)
box3.insert(END, ' P rated')
box4 = Entry(data, width=8, bg='#E0E0E0')
box4.grid(row=3, column=0)
box4.insert(END, ' V rated')
box5 = Entry(data, width=8, bg='#E0E0E0')
box5.grid(row=4, column=0)
box5.insert(END, ' I rated')
box6 = Entry(data, width=8, bg='#E0E0E0')
box6.grid(row=5, column=0)
box6.insert(END, ' Connect.')
box7 = Entry(data, width=8, bg='#E0E0E0')
box7.grid(row=6, column=0)
box7.insert(END, ' Frequency')
box8 = Entry(data, width=8, bg='#E0E0E0')
box8.grid(row=7, column=0)
box8.insert(END, ' N° poles')
```

```
    box9 = Entry(data, width=8, bg='#E0E0E0')
    box9.grid(row=8, column=0)
    box9.insert(END, ' Speed')
    box1a = Entry(data, width=8, bg='#FFFFFF')
    box1a.grid(row=0, column=1)
    box2a = Entry(data, width=8, bg='#FFFFFF')
    box2a.grid(row=1, column=1)
    box3a = Entry(data, width=8, bg='#FFFFFF')
    box3a.grid(row=2, column=1)
    box4a = Entry(data, width=8, bg='#FFFFFF')
    box4a.grid(row=3, column=1)
    box5a = Entry(data, width=8, bg='#FFFFFF')
    box5a.grid(row=4, column=1)
    box6a = Entry(data, width=8, bg='#FFFFFF')
    box6a.grid(row=5, column=1)
    box7a = Entry(data, width=8, bg='#FFFFFF')
    box7a.grid(row=6, column=1)
    box8a = Entry(data, width=8, bg='#FFFFFF')
    box8a.grid(row=7, column=1)
    box9a = Entry(data, width=8, bg='#FFFFFF')
    box9a.grid(row=8, column=1)

    # Create TEST CONDICTION label frame
    condiction = LabelFrame(window, text=" Test condiction", bg='#
d3d3d3')
    condiction.grid(row=4, column=0, padx=10, pady=5, columnspan=6,
sticky='W')

    # Create the tab for the TEST CONDICTION
    box1 = Entry(condiction, width=8, bg='#E0E0E0')
    box1.grid(row=0, column=0)
    box1.insert(END, ' Type')
    box2 = Entry(condiction, width=8, bg='#E0E0E0')
    box2.grid(row=1, column=0)
    box2.insert(END, ' Connect.')
    box3 = Entry(condiction, width=8, bg='#E0E0E0')
    box3.grid(row=2, column=0)
    box3.insert(END, ' Supply')
    box4 = Entry(condiction, width=8, bg='#E0E0E0')
    box4.grid(row=3, column=0)
    box4.insert(END, ' Frequency')
    box1b = Entry(condiction, width=8, bg='#FFFFFF')
    box1b.grid(row=0, column=1)
    box2b = Entry(condiction, width=8, bg='#FFFFFF')
    box2b.grid(row=1, column=1)
    box3b = Entry(condiction, width=8, bg='#FFFFFF')
    box3b.grid(row=2, column=1)
    box4b = Entry(condiction, width=8, bg='#FFFFFF')
    box4b.grid(row=3, column=1)

    sresistance_temp = LabelFrame(window, text="Stator resistance [Ohm]
 at \ntemperature [°C] of  ", bg='#d3d3d3')
    sresistance_temp.grid(row=5, column=0, padx=10, pady=5, columnspan
=6, sticky='W')

    box4c = Entry(sresistance_temp, width=8, bg='#FFFFFF')
    box4c.grid(row=0, column=0)
```

```
    box4c.insert(END, ' ')

    sresistance_startime = LabelFrame(window, text="Stator resistance
    at\n the start time [Ohm]", bg='#d3d3d3')
    sresistance_startime.grid(row=6, column=0, padx=10, pady=5,
    columnspan=6, sticky='W')

    box4d = Entry(sresistance_startime, width=8, bg='#FFFFFF')
    box4d.grid(row=0, column=0)
    box4d.insert(END, ' ')

    sresistance_stoptime = LabelFrame(window, text="Stator resistance
    at \n the stop time [Ohm]", bg='#d3d3d3')
    sresistance_stoptime.grid(row=7, column=0, padx=10, pady=5,
    columnspan=6, sticky='W')

    box4e = Entry(sresistance_stoptime, width=8, bg='#FFFFFF')
    box4e.grid(row=0, column=0)
    box4e.insert(END, ' ')

    #testo = Entry(text ='text 1')
    #testo.grid(row=12, column=0,padx=20, pady=5, columnspan=6, sticky
    ='W')

    mynote = Entry(window, bg='#F5F5F5', width=60, )
    mynote.grid(row=2, column=0, padx=10, pady=5, columnspan=6, sticky=
    'W')

    # CASELLA PER LA DIRECTORY
    mytext = Entry(window, bg='#F5F5F5', width=60)
    mytext.grid(row=1, column=0, padx=10, pady=8, columnspan=6, sticky=
    'W')


    # DATA

    tabledata = Frame(window, bg='#d3d3d3', width=200, height=300)
    tabledata.grid(row=3, column=2, pady=5, columnspan=10,rowspan=9,
    sticky='WN')

    # Scrill bar
    scroll= Scrollbar(tabledata)
    scroll.grid(row=0,column=9,rowspan=9,sticky='E')

    tree= ttk.Treeview(tabledata,yscrollcommand=scroll.set)


    # Define colums
    tree['columns'] = ('V_line', 'E_line', 'I_line','cos_fi','P_tot','
    P_cu','P_mecc+fe')

    # Formate our columns
    tree.column('#0', width=25, minwidth=40)
    tree.column('V_line', width=100, minwidth=40)
    tree.column('E_line', width=100, minwidth=40)
    tree.column('I_line', width=100, minwidth=40)
    tree.column('cos_fi', width=100, minwidth=40)
```

```
    tree.column('P_tot', width=100, minwidth=40)
    tree.column('P_cu', width=100, minwidth=40)
    tree.column('P_mecc+fe', width=100, minwidth=40)

    # Create Headings
    tree.heading('#0', text='N°')
    tree.heading('V_line', text='V_line')
    tree.heading('E_line', text='E_line')
    tree.heading('I_line', text='I_line')
    tree.heading('cos_fi', text='cos_fi')
    tree.heading('P_tot', text='P_tot')
    tree.heading('P_cu', text='P_cu')
    tree.heading('P_mecc+fe', text='P_mecc+fe')

    tree.grid(row=0,column=0,pady=5)

    # CREATE THE CANVAS FOR THE GRAH
    graph = Canvas(window,width=750, height=420)
    graph.grid(row=4, column=2, pady=5, columnspan=7, rowspan=9, sticky
    ='NW')

    # Crete the Space for the analysis
    analysis = Frame(window, bg='grey',width=200, height=200)
    analysis.grid(row=3, column=14, padx=30,pady=5, columnspan=2,
    rowspan=9, sticky='N')

    # Scrill bar
    scroll1 = Scrollbar(analysis)
    scroll1.grid(row=0, column=9, sticky='E')

    tree1 = ttk.Treeview(analysis, yscrollcommand=scroll1.set)

    # Define colums
    tree1['columns'] = ('V [V]', 'E.m.f [V]')

    # Formate our columns
    tree1.column('#0', width=25, minwidth=40)
    tree1.column('V [V]', width=100, minwidth=40)
    tree1.column('E.m.f [V]', width=100, minwidth=40)


    # Create Headings
    tree1.heading('#0', text='N°')
    tree1.heading('V [V]', text='V [V]')
    tree1.heading('E.m.f [V]', text='E.m.f [V]')

    tree1.grid(row=0, column=0, sticky='NW')

    alfio = Button(text='E.m.f', height=2, width=10)
    alfio.grid(row=3, column=16, padx=5, pady=5, sticky='NE')

    io = Button(text='IO', command=window.quit, height=2, width=10,
    state='disabled')
    io.grid(row=3, column=16, padx=5, pady=55, sticky='NE')

    po = Button(text='PO', command=window.quit, height=2, width=10,
    state='disabled')
```

```
    po.grid(row=3, column=16, padx=5, pady=105, sticky='NE')

    plot_button = Button(text='Plot', height=2, width=10, state='disabled')
    plot_button.grid(row=3, column=8, padx=5, sticky='S')

    fitting_button = Button(text='Fitting', height=2, width=10)
    fitting_button.grid(row=3, column=7, padx=5, sticky='S')

    print_button = Button(text='Print', height=2, width=10, state='disabled')
    print_button.grid(row=14, column=8, padx=5, sticky='S')

    delete_button = Button(text='Delete points', height=2, width=10, state='disabled')
    delete_button.grid(row=14, column=7, padx=5, sticky='S')

    grades = StringVar()
    grades_input = Entry(window, textvariable = grades, font=('calibre', 10, 'normal'))
    grades_input.grid(row=3, column=4, padx=5, columnspan=3, sticky='ES')

    grades_label = Label(window, text='Polynomial exponents \n separeted by a ","', font=('calibre', 10, 'bold'), bg = '#cccccc')
    grades_label.grid(row=3, column=3, columnspan=2, sticky='S')

    delete_points = StringVar()
    delete_input = Entry(window, textvariable=delete_points, font=('calibre', 10, 'normal'))
    delete_input.grid(row=14, column=4, padx=5, columnspan=3, sticky='ES')

    delete_label = Label(window, text='Delete points \n separeted by a ","', font=('calibre', 10, 'bold'),bg='#cccccc')
    delete_label.grid(row=14, column=3, columnspan=2, sticky='S')

    NL = LabelFrame(window, bg='#d3d3d3', text='No Load Parametrs')
    NL.grid(row=4, column=14, padx=5, pady=1, columnspan=2, rowspan=9, sticky='N')

    casella1 = Entry(NL, width=8)
    casella1.grid(row=0, column=0)
    casella2 = Entry(NL, width=8, bg='yellow')
    casella2.grid(row=0, column=1)
    casella3 = Entry(NL, width=8)
    casella3.grid(row=1, column=0)
    casella4 = Entry(NL, width=8, bg='yellow')
    casella4.grid(row=1, column=1)
    casella5 = Entry(NL, width=8)
    casella5.grid(row=2, column=0)
    casella6 = Entry(NL, width=8, bg='yellow')
    casella6.grid(row=2, column=1)
    casella7 = Entry(NL, width=8)
    casella7.grid(row=3, column=0)
    casella8 = Entry(NL, width=8, bg='yellow')
    casella8.grid(row=3, column=1)
```

```
    casella9 = Entry(NL, width=8)
    casella9.grid(row=4, column=0)
    casella11 = Entry(NL, width=8, bg='yellow')
    casella11.grid(row=4, column=1)


    Nload_button = Button(text='NoLoaD ', height=2, width=10, state ='disabled')
    Nload_button.grid(row=4, column=16, padx=1, pady=10,sticky='NE')
    report = Button( text='Report Print', height=2, width=10, command=PDF, state ='disabled')
    report.grid(row=4, column=16, padx=1, pady=60, sticky='NE')

    polinomial_frame = LabelFrame(window, bg='#d3d3d3')
    polinomial_frame.grid(row=5, column=14, padx=40, pady=50, columnspan=4, rowspan=10, sticky='N')

    # Scrill bar
    scrollp = Scrollbar(polinomial_frame)
    scrollp.grid(row=0, column=1, rowspan=9, sticky='NE')

    treep = ttk.Treeview(polinomial_frame, yscrollcommand=scroll.set)

    # Define colums
    treep['columns'] = ('Emf', 'I0', 'P0')

    # Formate our columns
    treep.column('#0', width=25, minwidth=40)
    treep.column('Emf', width=100, minwidth=40)
    treep.column('I0', width=100, minwidth=40)
    treep.column('P0', width=100, minwidth=40)


    # Create Headings
    treep.heading('#0', text='Grade')
    treep.heading('Emf', text='Emf')
    treep.heading('I0', text='I0')
    treep.heading('P0', text='P0')

    treep.grid(row=0, column=0)

def locked():
    global window, noload_button, lockrotor_button
    global indice_grap,Vnom,Inom,ColnNom,FreqNom,Npoli

    window.destroy()  # destroy the current window to create another one
    window = Tk()
    window.configure(bg='#cccccc')
    window.attributes('-fullscreen', True)
    window.title('Motor analysis')

    # NO LOAD BUTTON
    # noload_button['state'] = 'normal'
    noload_button = Button(text='No Load', command=noload, height=2, width=10, state='disabled')
    noload_button.grid(row=0, column=0, padx=10, pady=5, sticky='W')
```

```

# LOCKED ROTOR BUTTON
lockrotor_button = Button(text='Locked Rotor', height=2, width=10,
command=locked)
lockrotor_button.grid(row=0, column=1, padx=1, pady=5, sticky='W')

# Y=f(x) BUTTON
y_button = Button(text='Y=f(x)', height=2, width=10, command=
caratteristic)
y_button.grid(row=0, column=2, padx=10, pady=5, sticky='W')

# RESET BUTTON
reset_button = Button(text='Reset', height=2, width=10, command=
reset)
reset_button.grid(row=0, column=3, padx=1, pady=5, sticky='W')

# SPLIT FILE BUTTON
splitfile_button = Button(text='Split File', height=2, width=10,
command=donothing)
splitfile_button.grid(row=0, column=4, padx=10, pady=5, sticky='W')

# EXIT BUTTON
quit_button = Button(text='Exit', command=window.quit, height=2,
width=10)
quit_button.grid(row=0, column=5, padx=1, pady=5, sticky='W')

lockrotor_button['state'] = 'disabled'
global indice_grap     #serve per far riuscire il reset, perch
prima dava problemi con il plot
indice_grap=0

def PDF():
    pdf = FPDF()

    # Add a page
    pdf.add_page()

    # set style and size of font
    # that you want in the pdf
    pdf.set_font("helvetica", 'B', size=20)
    pdf.image('polito_logo_2021_blu.jpg', 10, 1, 40)

    # create a cell
    pdf.cell(200, 10, txt="LOCKED ROTOR TEST", ln=1, align='C')

    # add another cell
    pdf.set_font("helvetica", '', size=10)
    # pdf.set_text_color(169,169,169)
    pdf.cell(200, 10, txt="Path and file name:   " + window.filename
, ln=1, align='C')
    pdf.cell(100, 10, txt='Motor type: ' + model)
    pdf.cell(100, 10, txt='Company: ' + company, ln=1)
    pdf.set_font("helvetica", 'B', size=10)

    pdf.cell(100, 8, txt='Motor plate data', ln=1)
    pdf.set_font("helvetica", '', size=10)
    # pdf.set_text_color(169, 169, 169)
```

```
    pdf.cell(50, 5, txt='Power [kW]:  ' + str(P_target))
    pdf.cell(50, 5, txt='Voltage [V]:  ' + str(V_targa))
    pdf.cell(50, 5, txt='Current [A]:  ' + str(I_targa), ln=1)
    pdf.cell(50, 5, txt='Connection:  ' + con_targa)
    pdf.cell(50, 5, txt='Frequency [Hz]:  ' + str(frequency_targa))
    pdf.cell(50, 5, txt='Speed [rpm]:  ' + str(speed_n), ln=1)
    pdf.cell(50, 5, txt='Poles number:  ' + str(poles_targa), ln=1)
    pdf.set_font("helvetica", 'B', size=10)

    pdf.cell(100, 8, txt='Test conditions', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(50, 5, txt='Test connection:  ' + typology)
    pdf.cell(50, 5, txt='Test supply:  ' + tipologia, ln=1)
    pdf.cell(50, 5, txt='Test frequency [Hz]:  ' + str(frequency))
    pdf.cell(50, 5, txt='Reference voltage [V] at 50 [Hz]:  ' + str(V_targa), ln=1)
    pdf.cell(50, 5, txt='Test data and notes:  '+ note, ln=1)

    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(100, 8, txt='Test results', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(25, 8, txt='Vline [V]', border=1, align='C')
    pdf.cell(25, 8, txt='Eline [V]', border=1, align='C')
    pdf.cell(25, 8, txt='Iline [A]', border=1, align='C')
    pdf.cell(25, 8, txt='cosfi', border=1, align='C')
    pdf.cell(25, 8, txt='V at T°[V]', border=1, align='C')
    pdf.cell(25, 8, txt='P at T°[W]', border=1, align='C')
    pdf.cell(25, 8, txt='Cosfi at T°', border=1, align='C', ln=1)
    for i in range(0, 10):
        pdf.cell(25, 5, txt=f'{round(VFM[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(EM[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(curre_mes[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(CFM[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(power_mes[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(PCM[i],3)}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{round(PNM[i],3)}', border=1, align='C', ln=1)

    pdf.cell(0, 5, txt='Stator resistance at the reference temperature of ' + temp + ' = ' + f'{"{:.2e}".format(RFaseFinePrv)}' + ' [Ohm]', ln=1)
    pdf.cell(0, 5, txt='Stator resistance at start time = ' + f'{"{:.2e}".format(RFaseFinePrv)}' + ' [Ohm]', ln=1)
    pdf.cell(0, 5, txt='Stator resistance at stop time = ' + f'{"{:.2e}".format(RFaseFinePrv)}' + ' [Ohm]', ln=1)

    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(0, 8, txt='Locked rotor parameters at the reference temperature T =' + T_reference + ' °C', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(0, 5, txt='No load current:', ln=1)
    pdf.cell(0, 5, txt='Iron losses:', ln=1)
```

```
    pdf.cell(0, 5, txt='Mechanical losses: '+ f'{round((param2[0]),3)} '+ '[W]', ln=1)
    pdf.cell(0, 5, txt='Equivalent iron losses resistance:', ln=1)
    pdf.cell(0, 5, txt='Stator reactance:', ln=1)

    pdf.set_font("helvetica", 'B', size=10)
    pdf.cell(0, 8, txt='Coeffcients of the fitting polinomial equation:', ln=1)
    pdf.set_font("helvetica", '', size=10)
    pdf.cell(25, 8, txt='Grade', border=1, align='C')
    pdf.cell(25, 8, txt='V = f(I)', border=1, align='C')
    pdf.cell(25, 8, txt='P = f(I)', border=1, align='C')
    for i in range(0, 10):
        pdf.cell(25, 5, txt=f'{i}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{"{:.3e}".format(pino[i])}', border=1, align='C')
        pdf.cell(25, 5, txt=f'{"{:.3e}".format(param1[i])}', border=1, align='C')

    # save the pdf with name .pdf
    pdf.output("Locked_rotor.pdf")


def file():
    window.filename = filedialog.askopenfilename( initialdir='/Users/Federico/Desktop/LM_Tesi/programma2', title='select a file', filetypes=(('CTO files', '*.CTO'), ('CTO files', '*.CTO')))
    # tex = open(window.filename, 'r')
    mytext.insert(END, window.filename)
    # tex.close()
    global T_reference,model

    mylist = []  # define an empty list

    with open(window.filename,'r') as file:  # usare 'as file' per rendere il programma globale per qualunque file aperto!

        for line in file:

            for word in line.split(','):
                mylist.append(word)

    file.close()


    # definiamo il numero delle misure

    # define all the parametres based on the txt file

    company = mylist[0]
    model = mylist[1]
    P_target = float(mylist[2])
    V_targa = float(mylist[3])
    I_targa = float(mylist[4])
    con_targa = mylist[5]
    frequency_targa = float(mylist[6])
    poles_targa = float(mylist[7])
```

```
        speed_n = float(mylist[8])
        tipologia = mylist[9]
        frequency = float(mylist[10])
        typology = mylist[11]
        mesure = mylist[12]
        note = mylist[13]
        Vdc_temp = float(mylist[14])
        Idc_temp = float(mylist[15])
        Tamb = float(mylist[16])
        Vdc_first = float(mylist[17])
        Idc_first = float(mylist[18])
        box1a.insert(END, company)
        box2a.insert(END, model)
        box3a.insert(END, P_target)
        box4a.insert(END, V_targa)    # non posso convertire numerico se
 voglio aggiugere la stringa dell'unità di misura
        box5a.insert(END, I_targa)
        box6a.insert(END, con_targa)
        box7a.insert(END, frequency_targa)
        box8a.insert(END, poles_targa)
        box9a.insert(END, speed_n)

        # inserimento delle note
        mynote.insert(END, note)

        # completamento della secodna tabella
        box1b.insert(END, mesure)
        box2b.insert(END, typology)
        box3b.insert(END, tipologia)
        box4b.insert(END, frequency)

        temp=mylist[16]
        #sresistance_temp = LabelFrame(window, bg='#d3d3d3', text='
    Stator resistance at\n' + temp + '°C [Ohm]')
        #sresistance_temp.grid(row=5, column=0, padx=10, pady=5,
    columnspan=6, sticky='W')

        sresistance_temp1 = LabelFrame(window, bg='#d3d3d3', text='
    Stator resistance [Ohm] at \ntemperature [°C] of' + temp)
        sresistance_temp1.grid(row=5, column=0, padx=10, pady=5,
    columnspan=6, sticky='W')
        box_t = Entry(sresistance_temp1, width=8, bg='#FFFFFF')
        box_t.grid(row=0, column=0)
        #print(temp)

        #temperature window creation
        def get_temperature():
            T_reference = float(E_temperature.get()) # get the reference
    temperature
            print(T_reference)
            W_temperature.destroy() # destroy the secondary window

            col = 1
            # completamento della RESISTENZA DI STATORE nelle varie
            if con_targa == 'Y':
                col = sqrt(3)
            elif con_targa == 'D':
```

```
            col = 1
        else:
            messagebox.showwarning("warning", "Error with the configuration parameter!")
        IfaseNom = I_targa * col / sqrt(3)
        RfaseAmb = 1.5 * Vdc_temp / ( Idc_temp) / pow(col, 2)
        Rsfase_first = 3 * Vdc_first / (2 * Idc_first) / pow(col, 2)
        rappFrq = frequency / frequency_targa
        DeltaV = col * Rsfase_first * IfaseNom
        VPrvNom = DeltaV + rappFrq * (V_targa - DeltaV)

        # completamento della terza tabella
        #box4c.insert(END, RfaseAmb)
        box_t.insert(END, round(RfaseAmb, 7))

        # completamento della terza tabella
        box4d.insert(END, round(Rsfase_first, 7))

        # completamento della terza tabella
        #box4e.insert(END, mylist[17])

        # completamento tabella misure
        i = 1
        tens_mes = []
        for tensione in mylist[20:-20:10]:
            tens_mes.append(float(tensione))
            i = i + 1
        Ndati = int(i - 1)
        i = 1
        V1 = []
        for tensione1 in mylist[19:-20:10]:
            V1.append(float(tensione1))

        V2 = []
        for tensione in mylist[21:-20:10]:
            V2.append(float(tensione))

        I1 = []
        for tensione1 in mylist[22:-20:10]:
            I1.append(float(tensione1))

        I2 = []
        for tensione1 in mylist[24:-20:10]:
            I2.append(float(tensione1))

        i = 1
        curre_mes = []
        for current in mylist[23:-20:10]:
            curre_mes.append(float(current))
            i = i + 1
        i = 1
        PM = []
        for power in mylist[26:-20:10]:
            PM.append(float(power))

        P1 = []
        for power in mylist[25:-20:10]:
```

```
            P1.append(float(power))

        P2 = []
        for power in mylist[27:-20:10]:
            P2.append(float(power))

        i = 1
        temp_mes = []
        for tempo in mylist[28:-1:10]:
            temp_mes.append(tempo)
            i = i + 1
        temp_mes.append(mylist[-1])

        tp = []
        k = 0
        for tempo in temp_mes:
            k = 0

            for a in tempo.split('.'):

                # print(tp)
                if k == 0:
                    hour = int(a)
                elif k == 1:
                    minut = int(a)
                else:
                    sec = int(a)
                k = k + 1
            som = sec + 60 * minut + 3600 * hour
            tp.append(som)  #     tipo int
        # print(tp)
        # print(Ndati)

        # completamento della tabella delle misurazioni con i calcoli

        Rad3 = sqrt(3)
        Rfase_end = 1.5 * float(mylist[-19]) / float(mylist[-16]) / pow(col, 2)
        Rfase_end30 = 1.5 * float(mylist[-9]) / float(mylist[-6]) / pow(col, 2)
        RappTP = (tp[-1] - tp[-3]) / (tp[-1] - tp[-2])

        RFaseFinePrv = RappTP * (Rfase_end - Rfase_end30) + Rfase_end30
        RappRT = (RFaseFinePrv - Rsfase_first) / (tp[Ndati] - tp[0])
        RFaseTrif = RfaseAmb * (T_reference + 235) / (Tamb + 235)

        box4e.insert(END, round(RFaseFinePrv, 7))

        i = 0
        Rfase = []
        PCM = []
        PNM = []
        CFM = []
        EM = []
        RappTemp = []
        PC1 = []
```

```
        PC2 = []
        VCM = []
        VC1 = []
        VC2 = []
        CFCM = []

        for i in range(Ndati):
            Rfase.append(Rsfase_first + (tp[i] - tp[0]) * RappRT)
            RappTemp = RFaseTrif / Rfase[i]
            VFM = (tens_mes[i] / col)
            IFM = (curre_mes[i] * col / Rad3)
            VF1 = V1[i] / col
            IF1 = I1[i] * col / Rad3
            VF2 = V2[i] / col
            IF2 = I2[i] * col / Rad3
            ZCCM = (VFM / IFM)
            RCCM = PM[i] / (3 * pow(IFM, 2))
            XCCM = sqrt(pow(ZCCM, 2) - pow(RCCM, 2))
            CFM.append(RCCM / ZCCM)
            PCM.append(RappTemp * PM[i])
            #PCM.append(3 * Rfase[i] * pow(IFM[i], 2))
            RCM = RappTemp * RCCM
            ZCM = sqrt(pow(RCM, 2) + pow(XCCM, 2))
            PC1.append(RappTemp * P1[i])
            PC2.append(RappTemp * P2[i])
            VFCM = ZCM * IFM
            VCM.append(VFCM * col)
            VFC1 = ZCM * IF1
            VC1.append(VFC1 * col)
            VFC2 = ZCM * IF2
            VC2.append(VFC2 * col)
            CFCM.append(RCM / ZCM)
            #PNM.append(PM[i] - PCM[i])
            #CFM.append(PM[i] / Rad3 / tens_mes[i] / curre_mes[i])
            #SFM = (sqrt(1 - pow(CFM[i], 2)))
            #EFM = (sqrt(pow((VFM[i] - Rfase[i] * IFM[i] * CFM[i]), 2) + pow((Rfase[i] * IFM[i] * SFM), 2)))
            #EM.append(EFM * col)

            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),))
            scroll.configure(command=tree.yview)

        # print(IFM)

        # abilitare il comando emf
        def Vlocked():
            plot_button['state'] = 'normal'
            fitting_button['state'] = 'disabled'
            indice_inter = 0
            q = 1
            i = 0
            for i in range(Ndati):
                tree1.insert(parent='', index='end', iid=i, text=i + 1, values=(round(curre_mes[i], 3), round(VCM[i], 3)))
```

```
            io['state'] = 'normal'
            alfio['state'] = 'disabled'

        alfio.configure(command=Vlocked)

        def Plocked():
            plot_button['state'] = 'normal'
            fitting_button['state'] = 'disabled'
            q = 2
            i = 0
            tree2 = ttk.Treeview(analysis)

            tree2['columns'] = ('I [A]', 'P [W]')

            # Formate our columns
            tree2.column('#0', width=25, minwidth=40)
            tree2.column('I [A]', width=100, minwidth=40)
            tree2.column('P [W]', width=100, minwidth=40)
            tree2.grid(row=0, column=0)

            # Create Headings
            tree2.heading('#0', text='N°')
            tree2.heading('I [A]', text='I [A]')
            tree2.heading('P [W]', text='P [W]')
            for i in range(Ndati):
                tree2.insert(parent='', index='end', iid=i, text=i + 1,
 values=(round(curre_mes[i], 3), round(PCM[i], 3)))
            io['state'] = 'disabled'

        io.configure(command=Plocked)

        def grap():

            # Data for plotting
            # t = np.arange(0.0, 2.0, 0.01)
            # s = 1 + np.sin(2 * np.pi * t)
            global indice_grap, indice_inter1

            if indice_grap == 0:
                f = Figure(figsize=(6, 4))
                a = f.add_subplot(111)
                a.plot(curre_mes, VCM, 'o', color='red')
                # setting x and y axis range
                plt.ylim(0, 450)
                plt.xlim(0, 450)
                a.set(xlabel=' I (A)', ylabel=' V (V)',
                      title='Company: ' + company + '          Type: ' +
 model)
                matteo = FigureCanvasTkAgg(f, master=window)
                matteo.draw()
                matteo.get_tk_widget().grid(row=4, column=2, pady=10,
padx=10, columnspan=7, rowspan=9, sticky='N')
                indice_inter1 = 0

            if indice_grap == 1:
                f = Figure(figsize=(6, 4))
                a = f.add_subplot(111)
```

```
                a.plot(curre_mes, PCM, 'o', color='red')
                # setting x and y axis range
                plt.ylim(0, 25)
                plt.xlim(0, 450)
                a.set(xlabel=' I (A)', ylabel=' P (W)',
                      title='Company: ' + company + '          Type: ' + model)
                matteo = FigureCanvasTkAgg(f, master=window)
                matteo.draw()
                matteo.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
                indice_inter1 = 1

            indice_grap = indice_grap + 1
            plot_button['state'] = 'disabled'
            fitting_button['state'] = 'normal'

        plot_button.configure(command=grap)

        def interpolation():
            global param2, parametri, parametri1, indice_inter, parametri2

            inserimento = grades.get()
            print(inserimento)
            coeff = []
            for w in inserimento.split(','):
                coeff.append(w)

            lung = len(coeff)

            global param2,Vcc,Pcc,CFcc,Zcc,Zaux,Rcc,Xcc,Rs,Rr,Xs,Xr

            def test(x, a, b):
                return a + b * x

            def test2(x, a):
                return a * np.power(x, 2)

            def test3(x, a, b, c, d, e, f, g, h, i, l):

                a1 = 0
                b1 = 0
                c1 = 0
                d1 = 0
                e1 = 0
                f1 = 0
                g1 = 0
                h1 = 0
                i1 = 0
                l1 = 0
                for indice in range(lung):

                    if int(coeff[indice]) == 0:
                        a1 = 1
                    if int(coeff[indice]) == 1:
                        b1 = 1
```

```
                        if int(coeff[indice]) == 2:
                            c1 = 1
                        if int(coeff[indice]) == 3:
                            d1 = 1
                        if int(coeff[indice]) == 4:
                            e1 = 1
                        if int(coeff[indice]) == 5:
                            f1 = 1
                        if int(coeff[indice]) == 6:
                            g1 = 1
                        if int(coeff[indice]) == 7:
                            h1 = 1
                        if int(coeff[indice]) == 8:
                            i1 = 1
                        if int(coeff[indice]) == 9:
                            l1 = 1

                    if a1 == 0:
                        a = 0
                    if b1 == 0:
                        b = 0
                    if c1 == 0:
                        c = 0
                    if d1 == 0:
                        d = 0
                    if e1 == 0:
                        e = 0
                    if f1 == 0:
                        f = 0
                    if g1 == 0:
                        g = 0
                    if h1 == 0:
                        h = 0
                    if i1 == 0:
                        i = 0
                    if l1 == 0:
                        l = 0

                    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)

                if indice_inter1 == 0:
                    x = np.array(curre_mes)
                    y = np.array(VCM)
                    # param, param_cov = curve_fit(test, x, y)
                    # ans = param[0] + x
                    param, _ = curve_fit(test, x, y)
                    ans = param[0] + param[1] * x
                    # 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)
                    foto = Figure(figsize=(6, 4))
                    anna = foto.add_subplot(111)
                    anna.plot(x, y, 'o', color='red', label="data")
```

```
            anna.plot(x, ans, '--', color='blue', label="optimized data")
            plt.ylim(0, 450)
            plt.xlim(0, 450)
            anna.set(xlabel=' I [A]', ylabel=' V [V]', title=' Company: ' + company + '         Type: ' + model)
            intr = FigureCanvasTkAgg(foto, master=window)
            intr.draw()
            intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
            parametri = []
            # fitting_button['state'] = 'disabled'
            for i in range(0, 10):
                print(i)
                if i == 0:
                    parametri.append(param[0])

                if i == 1:
                    parametri.append(param[1])

                if i > 1:
                    parametri.append(0.0)

                treep.insert(parent='', index='end', iid=i, text=i, values="{:.2e}".format(parametri[i]))
            print(parametri)
            print(Ndati)

        parametri1 = []

        if indice_inter1 == 1:
            x = np.array(curre_mes)
            y = np.array(PCM)
            parametri1, _ = curve_fit(test2, x, y)
            #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)
            ans7 = parametri1[0] * np.power(x, 2)
            x1 = np.linspace(0, max(x) + 10, 500)
            y1 = parametri1[0] * np.power(x1, 2)
            foto8 = Figure(figsize=(6, 4))
            aaa = foto8.add_subplot(111)
            aaa.plot(x, y, 'o', color='red', label="data")
            aaa.plot(x1, y1, '--', color='blue', label="optimized data")
            plt.xlim([0, max(x) + 10])
            plt.ylim([0, max(y) + 10])
            # plt.ylim(0, 25)
            # plt.xlim(0, 450)
            aaa.set(xlabel=' I [A]', ylabel=' P [W]', title=' Company: ' + company + '         Type: ' + model)
            intr = FigureCanvasTkAgg(foto8, master=window)
            intr.draw()
            intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')
```

```
                parametri2 = []
                # fitting_button['state'] = 'disabled'
                for i in range(0, 10):
                    if i == 2:
                        parametri2.append(parametri1[0])
                    else:
                        parametri2.append(0.0)

                    #treep.insert(parent='', index='end', iid=i, text=i, values="{:.3e}".format(parametri2[i]))
                    treep.set(i, column='#2', value="{:.2e}".format(parametri2[i]))

            if indice_inter1 == 5:
                x = np.array(curre_mes)
                y = np.array(PCM)
                parametri1, _ = curve_fit(test2, x, y)
                #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)
                ans5 = parametri1[0] * np.power(x, 2)
                print("funcion coefficients zio:")
                print(parametri1)

                for index in range(0, 10):
                    print(index)
                    if parametri1[index] == 1.00000000e+00:
                        parametri1[index] = 0

                print('coefficienti con inserivmento dei dati FILTRATI:')
                print(parametri1)
                x1 = np.linspace(0, max(x) + 100, 500)
                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)
                foto6 = Figure(figsize=(6, 4))
                aaa = foto6.add_subplot(111)
                aaa.plot(x, y, 'o', color='red', label="data")
                aaa.plot(x1, y1, '--', color='blue', label="optimized data")
                plt.xlim([0, max(x) + 100])
                plt.ylim([0, max(y) + 10])
                # plt.ylim(0, 25)
                # plt.xlim(0, 450)
                aaa.set(xlabel=' I [A]', ylabel=' P [W]', title=' Company: ' + company + '          Type: ' + model)
                intr = FigureCanvasTkAgg(foto6, master=window)
                intr.draw()
                intr.get_tk_widget().grid(row=4, column=2, pady=10, padx=10, columnspan=7, rowspan=9, sticky='N')

```

```

            for i in range(0, 10):
                treep.set(i, column='#2', value="{:.3e}".format(parametri1[i]))


    fitting_button.configure(command=interpolation)

    def LRparameters():

        casella1.insert(END, 'Vlr [V]')
        casella3.insert(END, 'Plr [W]')
        casella5.insert(END, 'cos_fi [Ohm]')
        casella7.insert(END, 'Zlr [Ohm]')
        casella9.insert(END, 'Rs  [Ohm]')
        casella12.insert(END, 'Rr  [Ohm]')
        casella14.insert(END, 'Xs [Ohm]')
        casella16.insert(END, 'Xr [Ohm]')

        Vcc = 0
        Pcc = 0
        Icc = I_targa

        for i in range(0, 10):
            Vcc = Vcc + parametri[i] * pow(Icc, i)
            Pcc = Pcc + parametri2[i] * pow(Icc, i)

        CFcc = Pcc / sqrt(3) / Vcc / Icc
        Zcc = Vcc * sqrt(3) / Icc / col / col
        Zaux = (Vcc - parametri[0]) * sqrt(3) / Icc / col / col
        Rcc = Zcc * CFcc
        Xcc = sqrt(pow(Zcc, 2) - pow(Rcc, 2))
        Rs = RFaseTrif
        print(Rs)
        Rr = Rcc - Rs
        Xs = sqrt(pow(Zaux, 2) - pow(Rcc, 2)) / 2
        Xr = Xcc - Xs

        casella2.insert(END, '{:.2e}'.format(Vcc))
        casella4.insert(END, '{:.2e}'.format(Pcc))
        casella6.insert(END, '{:.2e}'.format(CFcc))
        casella8.insert(END, '{:.2e}'.format(Zcc))
        casella11.insert(END, '{:.2e}'.format(Rs))
        casella13.insert(END, '{:.2e}'.format(Rr))
        casella15.insert(END, '{:.2e}'.format(Xs))
        casella17.insert(END, '{:.2e}'.format(Xr))


        LR_button['state'] = 'disabled'
        report['state'] = 'normal'

        # Create the .par file
        filename2 = company + '.PAR'
        file = open(filename2, 'a')
        file.write('\n' + 'CTO' + '\n' + str(RFaseTrif) + ',' + str(T_reference) + '\n' + str(parametri[0]))
        for i in range(1, 10):
```

```
            file.write(',' + str(parametri[i]))
         file.write('\n' + str(parametri2[0]))
         for i in range(1, 10):
            file.write(',' + str(parametri2[i]))
         file.write('\n'+ con_targa)
         file.close()



      LR_button.configure(command=LRparameters)

   W_temperature=Toplevel()
   W_temperature.title('Reference temeperature for the stator windings')
   L_temperaturewind=Label(W_temperature, text='Reference temeperature T*?')
   L_temperaturewind.grid(row=0, column=0, padx=15, pady=5)
   B_temperatureok=Button(W_temperature, text='Ok', height=2, width=10, command=get_temperature)
   B_temperatureok.grid(row=0, column=1, padx=15, pady=5)
   B_temperaturedelete = Button(W_temperature, text='Delete', height=2, width=10, command=W_temperature.destroy)
   B_temperaturedelete.grid(row=0, column=2, padx=15, pady=5)
   E_temperature=Entry(W_temperature)
   E_temperature.grid(row=1, column=0, padx=15, pady=5)




#Crete the MENU BAR

menubar = Menu(window)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Open file", command=file)
filemenu.add_command(label="Close", command=window.quit)
menubar.add_cascade(label="File", menu=filemenu)
window.config(menu=menubar)

# CREATE MOTOR PLATE DATA
data = LabelFrame(window, text="Motor plate data", bg='#d3d3d3')
data.grid(row=3, column=0, padx=10, pady=5, columnspan=6,sticky='W')
# INSIDE THE LABEL
box1 = Entry(data, width=8, bg='#E0E0E0')
box1.grid(row=0, column=0)
box1.insert(END, ' Company')
box2 = Entry(data, width=8, bg='#E0E0E0')
box2.grid(row=1, column=0)
box2.insert(END, ' Type')
box3 = Entry(data, width=8, bg='#E0E0E0')
box3.grid(row=2, column=0)
box3.insert(END, ' P rated')
box4 = Entry(data, width=8, bg='#E0E0E0')
box4.grid(row=3, column=0)
box4.insert(END, ' V rated')
box5 = Entry(data, width=8, bg='#E0E0E0')
box5.grid(row=4, column=0)
```

```
    box5.insert(END, ' I rated')
    box6 = Entry(data, width=8, bg='#E0E0E0')
    box6.grid(row=5, column=0)
    box6.insert(END, ' Connect.')
    box7 = Entry(data, width=8, bg='#E0E0E0')
    box7.grid(row=6, column=0)
    box7.insert(END, ' Frequency')
    box8 = Entry(data, width=8, bg='#E0E0E0')
    box8.grid(row=7, column=0)
    box8.insert(END, ' N° poles')
    box9 = Entry(data, width=8, bg='#E0E0E0')
    box9.grid(row=8, column=0)
    box9.insert(END, ' Speed')
    box1a = Entry(data, width=8, bg='#FFFFFF')
    box1a.grid(row=0, column=1)
    box2a = Entry(data, width=8, bg='#FFFFFF')
    box2a.grid(row=1, column=1)
    box3a = Entry(data, width=8, bg='#FFFFFF')
    box3a.grid(row=2, column=1)
    box4a = Entry(data, width=8, bg='#FFFFFF')
    box4a.grid(row=3, column=1)
    box5a = Entry(data, width=8, bg='#FFFFFF')
    box5a.grid(row=4, column=1)
    box6a = Entry(data, width=8, bg='#FFFFFF')
    box6a.grid(row=5, column=1)
    box7a = Entry(data, width=8, bg='#FFFFFF')
    box7a.grid(row=6, column=1)
    box8a = Entry(data, width=8, bg='#FFFFFF')
    box8a.grid(row=7, column=1)
    box9a = Entry(data, width=8, bg='#FFFFFF')
    box9a.grid(row=8, column=1)

    # Create TEST CONDICTION label frame
    condiction = LabelFrame(window, text=" Test condiction", bg='#d3d3d3')
    condiction.grid(row=4, column=0, padx=10, pady=5, columnspan=6, sticky='W')

    # Create the tab for the TEST CONDICTION
    box1 = Entry(condiction, width=8, bg='#E0E0E0')
    box1.grid(row=0, column=0)
    box1.insert(END, ' Type')
    box2 = Entry(condiction, width=8, bg='#E0E0E0')
    box2.grid(row=1, column=0)
    box2.insert(END, ' Connect.')
    box3 = Entry(condiction, width=8, bg='#E0E0E0')
    box3.grid(row=2, column=0)
    box3.insert(END, ' Supply')
    box4 = Entry(condiction, width=8, bg='#E0E0E0')
    box4.grid(row=3, column=0)
    box4.insert(END, ' Frequency')
    box1b = Entry(condiction, width=8, bg='#FFFFFF')
    box1b.grid(row=0, column=1)
    box2b = Entry(condiction, width=8, bg='#FFFFFF')
    box2b.grid(row=1, column=1)
    box3b = Entry(condiction, width=8, bg='#FFFFFF')
    box3b.grid(row=2, column=1)
```

```
box4b = Entry(condition, width=8, bg='#FFFFFF')
box4b.grid(row=3, column=1)

sresistance_temp = LabelFrame(window, bg='#d3d3d3', text='Stator resistance [Ohm] at \ntemperature [°C] of  ')
sresistance_temp.grid(row=5, column=0, padx=10, pady=5, columnspan=6, sticky='W')

box4c = Entry(sresistance_temp, width=8, bg='#FFFFFF')
box4c.grid(row=0, column=0)
box4c.insert(END, ' ')

sresistance_startime = LabelFrame(window, text="Stator resistance at\n the start time [Ohm]", bg='#d3d3d3')
sresistance_startime.grid(row=6, column=0, padx=10, pady=5, columnspan=6, sticky='W')

box4d = Entry(sresistance_startime, width=8, bg='#FFFFFF')
box4d.grid(row=0, column=0)
box4d.insert(END, ' ')

sresistance_stoptime = LabelFrame(window, text="Stator resistance at \n the start time [Ohm]", bg='#d3d3d3')
sresistance_stoptime.grid(row=7, column=0, padx=10, pady=5, columnspan=6, sticky='W')

box4e = Entry(sresistance_stoptime, width=8, bg='#FFFFFF')
box4e.grid(row=0, column=0)
box4e.insert(END, ' ')

#testo = Entry(text ='text 1')
#testo.grid(row=12, column=0,padx=20, pady=5, columnspan=6, sticky='W')

mynote = Entry(window, bg='#F5F5F5', width=60, )
mynote.grid(row=2, column=0, padx=10, pady=5, columnspan=6, sticky='W')

# CASELLA PER LA DIRECTORY
mytext = Entry(window, bg='#F5F5F5', width=60)
mytext.grid(row=1, column=0, padx=10, pady=8, columnspan=6, sticky='W')


# DATA

tabledata = Frame(window, bg='#d3d3d3', width=200, height=300)
tabledata.grid(row=3, column=2, pady=5, columnspan=10, rowspan=9, sticky='WN')

# Scrill bar
scroll= Scrollbar(tabledata)
scroll.grid(row=0, column=9, rowspan=9, sticky='E')

tree= ttk.Treeview(tabledata, yscrollcommand=scroll.set)
```

```
    # Define colums
    tree['columns'] = ('I_line', 'V_line', 'P', 'cos_fi', 'V at T*', 'P
     at T*', 'cos_fi at T*')

    # Formate our columns
    tree.column('#0', width=25, minwidth=40)
    tree.column('I_line', width=100, minwidth=40)
    tree.column('V_line', width=100, minwidth=40)
    tree.column('P', width=100, minwidth=40)
    tree.column('cos_fi', width=100, minwidth=40)
    tree.column('V at T*', width=100, minwidth=40)
    tree.column('P at T*', width=100, minwidth=40)
    tree.column('cos_fi at T*', width=100, minwidth=40)

    # Create Headings
    tree.heading('#0', text='N°')
    tree.heading('I_line', text='I_line')
    tree.heading('V_line', text='V_line')
    tree.heading('P', text='P')
    tree.heading('cos_fi', text='cos_fi')
    tree.heading('V at T*', text='V at T*')
    tree.heading('P at T*', text='P at T*')
    tree.heading('cos_fi at T*', text='cos_fi at T*')

    tree.grid(row=0, column=0, pady=5)

    # CREATE THE CANVAS FOR THE GRAH
    graph = Canvas(window, width=750, height=420)
    graph.grid(row=4, column=2, pady=5, columnspan=7, rowspan=9, sticky
    ='NW')

    # Crete the Space for the analysis
    analysis = Frame(window, bg='grey',width=200, height=200)
    analysis.grid(row=3, column=14, padx=30,pady=5, columnspan=2,
    rowspan=9, sticky='N')

    # Scrill bar
    scroll1 = Scrollbar(analysis)
    scroll1.grid(row=0, column=9, sticky='E')

    tree1 = ttk.Treeview(analysis, yscrollcommand=scroll1.set)

    # Define colums
    tree1['columns'] = ('I [A]', 'V [V]')

    # Formate our columns
    tree1.column('#0', width=25, minwidth=40)
    tree1.column('I [A]', width=100, minwidth=40)
    tree1.column('V [V]', width=100, minwidth=40)


    # Create Headings
    tree1.heading('#0', text='N°')
    tree1.heading('I [A]', text='I [A]')
    tree1.heading('V [V]', text='V [V]')

    tree1.grid(row=0, column=0, sticky='NW')
```

```python

alfio = Button(text='V locked rotor', height=2, width=10)
alfio.grid(row=3, column=16, padx=5, pady=5, sticky='NE')

io = Button(text='P locked rotor', command=window.quit, height=2, width=10, state='disabled')
io.grid(row=3, column=16, padx=5, pady=55, sticky='NE')

plot_button = Button(text='Plot', height=2, width=10, state='disabled')
plot_button.grid(row=3, column=8, padx=5, sticky='S')

fitting_button = Button(text='Fitting', height=2, width=10, state='disabled')
fitting_button.grid(row=3, column=7, padx=5, sticky='S')

print_button = Button(text='Print', height=2, width=10, state='disabled')
print_button.grid(row=14, column=8, padx=5, sticky='S')

delete_button = Button(text='Delete points', height=2, width=10, state='disabled')
delete_button.grid(row=14, column=7, padx=5, sticky='S')

grades = StringVar()
grades_input = Entry(window, textvariable = grades, font=('calibre',10,'normal'))
grades_input.grid(row=3, column=4, padx=5, columnspan=3, sticky='ES')

grades_label = Label(window, text='Polynomial exponents \n separeted by a ","', font=('calibre', 10, 'bold'), bg = '#cccccc')
grades_label.grid(row=3, column=3, columnspan=2, sticky='S')

delete_points = StringVar()
delete_input = Entry(window, textvariable=grades, font=('calibre', 10, 'normal'))
delete_input.grid(row=14, column=4, padx=5, columnspan=3, sticky='ES')

delete_label = Label(window, text='Delete points \n separeted by a ","', font=('calibre', 10, 'bold'), bg='#cccccc')
delete_label.grid(row=14, column=3, columnspan=2, sticky='S')


NL = LabelFrame(window, bg='#d3d3d3', text='Locked Rotor Parametrs')
NL.grid(row=3, column=14, padx=5, pady=250, columnspan=8, rowspan=9, sticky='N')
casella1 = Entry(NL, width=8)
casella1.grid(row=0, column=0)
casella2 = Entry(NL, width=8, bg='yellow')
casella2.grid(row=0, column=1)
casella3 = Entry(NL, width=8)
casella3.grid(row=1, column=0)
casella4 = Entry(NL, width=8, bg='yellow')
casella4.grid(row=1, column=1)
```

```
    casella5 = Entry(NL, width=8)
    casella5.grid(row=2, column=0)
    casella6 = Entry(NL, width=8, bg='yellow')
    casella6.grid(row=2, column=1)
    casella7 = Entry(NL, width=8)
    casella7.grid(row=3, column=0)
    casella8 = Entry(NL, width=8, bg='yellow')
    casella8.grid(row=3, column=1)
    casella9 = Entry(NL, width=8)
    casella9.grid(row=4, column=0)
    casella11 = Entry(NL, width=8, bg='yellow')
    casella11.grid(row=4, column=1)
    casella12 = Entry(NL, width=8)
    casella12.grid(row=5, column=0)
    casella13 = Entry(NL, width=8, bg='yellow')
    casella13.grid(row=5, column=1)
    casella14 = Entry(NL, width=8)
    casella14.grid(row=6, column=0)
    casella15 = Entry(NL, width=8, bg='yellow')
    casella15.grid(row=6, column=1)
    casella16 = Entry(NL, width=8)
    casella16.grid(row=7, column=0)
    casella17 = Entry(NL, width=8, bg='yellow')
    casella17.grid(row=7, column=1)


    LR_button = Button(text='Locked rotor ', height=2, width=10)
    LR_button.grid(row=4, column=16, padx=1,pady=10,sticky='NE')
    report = Button( text='Report Print', command=PDF, height=2, width
    =10, state ='normal')
    report.grid(row=4, column=16, padx=1, pady=60,sticky='NE')

    polinomial_frame = LabelFrame(window, bg='#d3d3d3')
    polinomial_frame.grid(row=5, column=14, padx=40, pady=50,
     columnspan=4, rowspan=10, sticky='N')

    # Scrill bar
    scrollp = Scrollbar(polinomial_frame)
    scrollp.grid(row=0, column=1, rowspan=9, sticky='NE')

    treep = ttk.Treeview(polinomial_frame, yscrollcommand=scroll.set)

    # Define colums
    treep['columns'] = ('V locked', 'P locked')

    # Formate our columns
    treep.column('#0', width=25, minwidth=40)
    treep.column('V locked', width=100, minwidth=40)
    treep.column('P locked', width=100, minwidth=40)


    # Create Headings
    treep.heading('#0', text='Grade')
    treep.heading('V locked', text='V locked')
    treep.heading('P locked', text='P locked')

    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. [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].