An Application for Basic Curve Fitting of Experimental Data

Nabil Nasreldeen Mohamed

4th year student,

Computer science Department

Supervised By

Dr / Khaled Mohammed Hussien

Physics Department, Faculty of Science, South Valley University

2022 / 2023

Contents

Introduction:

In the field of physics, data analysis plays a crucial role in understanding and interpreting experimental results. When conducting experiments, scientists collect data to test hypotheses, validate theories, and make predictions. However, raw data alone is often insufficient to draw meaningful conclusions. That's where the process of fitting datasets comes into play.

The process typically involves selecting an appropriate mathematical model or equation that is expected to represent the underlying physics of the system being studied.

Depending on the nature of the data and the phenomena being investigated this model can be **linear equation** or **nonlinear function** like a curve for example.

Linear fitting(Linear regression):

Introduction:

Linear regression is a type of statistical analysis used to predict the relationship between two variables. It assumes a linear relationship between the independent variable and the dependent variable, and aims to find the best-fitting line that describes the relationship. The line is determined by minimizing the sum of the squared differences between the predicted values and the actual values.

The general form for the linear function:

$$f(x)=mx+c$$

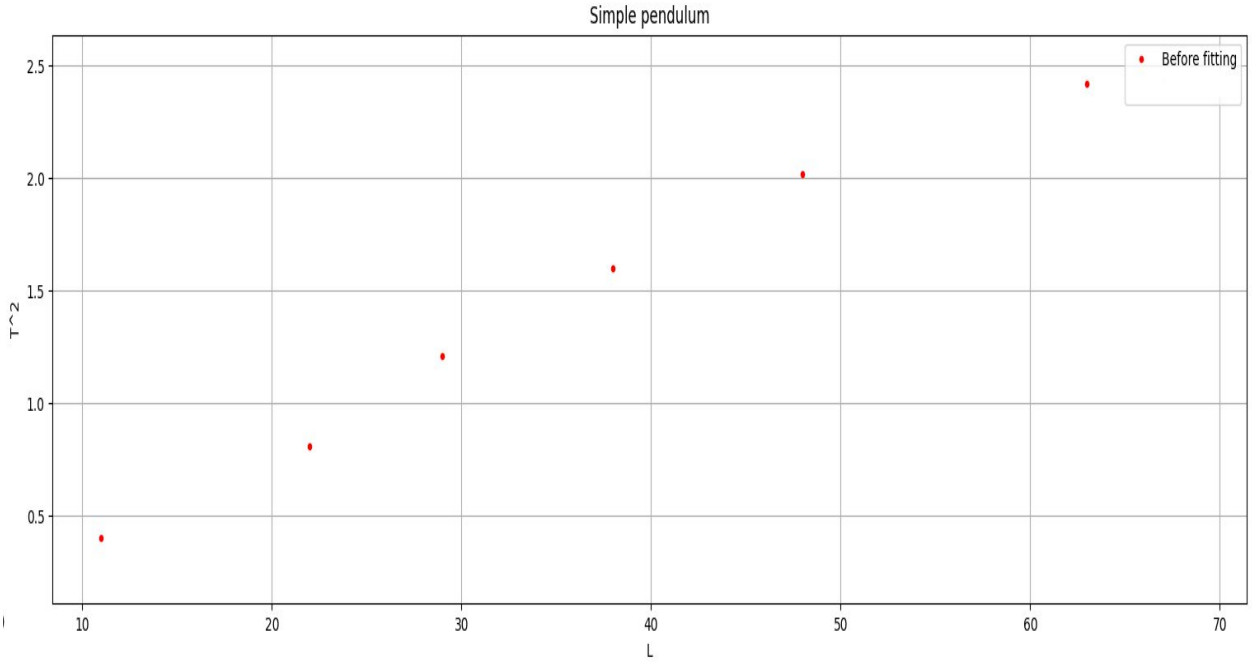
Where m corresponds to slope and c corresponds to intercept.

Consider the following points for simple pendulum experiment:

X=[11,22,29,38,48,63]

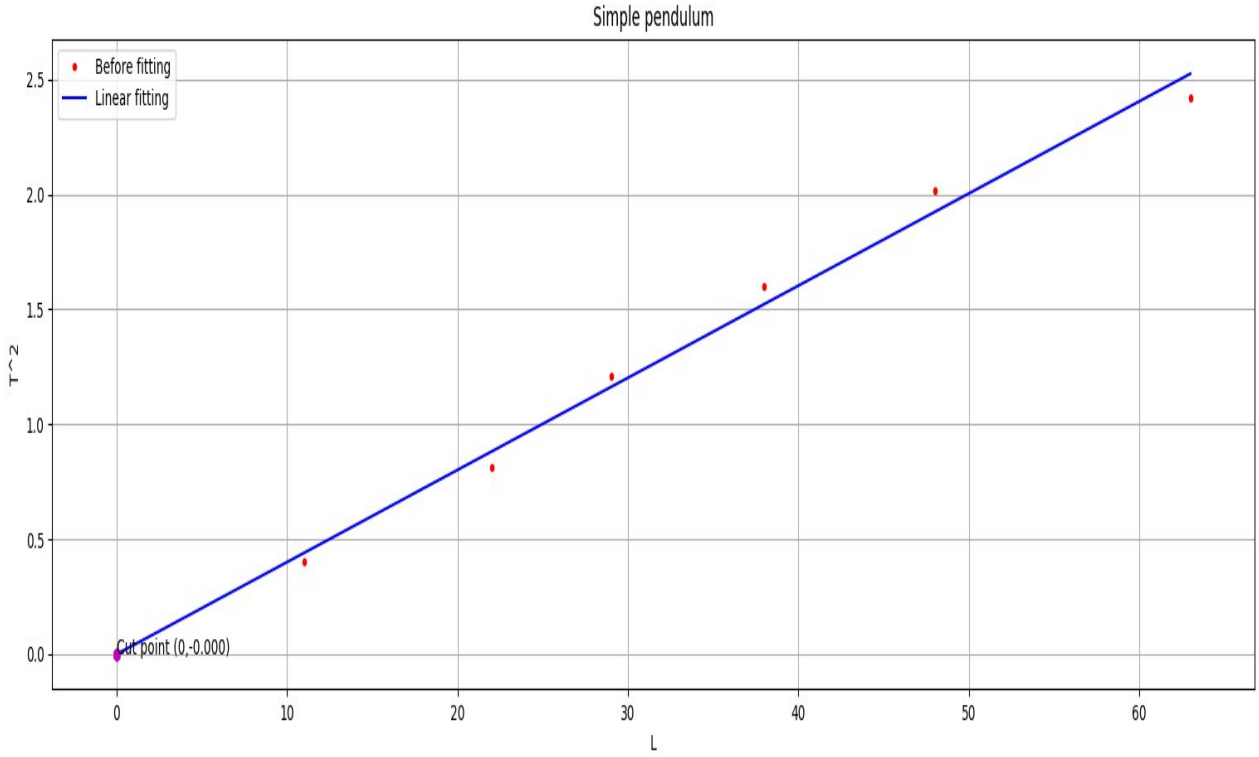
Y=[0.403,0.81,1.21,1.6,2.015,2.42]

Can be plotted as the following:



We want to find the optimal values of the slope and intercept that fit the data above.

By using linear fitting on the points, that gives us the following result:



And the fitting result is :

Slope : 0.04009754 s^2/cm

Intercept = 0 s^2

Earth gravitational acceleration = 984.55 cm /s^2

What about if we want interpolation??

Or extrapolation??

Interpolation:

Interpolation is a mathematical technique used to estimate values between two known data points. It involves constructing a function or curve that passes through the given data points and can be used to approximate values at intermediate points.

The basic idea behind interpolation is to assume that the values between the known data points follow a certain pattern or trend.

By analyzing the available data, interpolation methods aim to find a function or equation that best fits the given data points and can be used to predict or estimate values at other points within the range.

There are various interpolation methods, including linear interpolation, polynomial interpolation, spline interpolation, and more.

The choice of method depends on the nature of the data and the desired accuracy of the estimation.

Linear interpolation is the simplest form of interpolation and involves drawing a straight line between two adjacent data points and estimating values based on the position along that line.

Polynomial interpolation uses a polynomial function to approximate the data points.

Spline interpolation divides the data range into smaller segments and fits separate curves within each segment to provide a smoother estimate.

It's important to note that interpolation assumes a continuous and smooth relationship between data points, which may not always be accurate.

Extrapolation:

Extrapolation is a process of estimating or predicting values beyond a given set of data points by extending or projecting existing trends or patterns.

It involves making assumptions about the continuation of a particular trend or relationship and using those assumptions to make predictions for future values.

Extrapolation is commonly used in various fields, including statistics, economics, physics, and engineering.

It assumes that the underlying factors affecting a particular phenomenon will remain relatively constant or continue to evolve in a predictable manner.

However, it's important to note that extrapolation carries inherent risks and limitations.

The accuracy of extrapolated predictions depends on several factors, such as the quality and representativeness of the data used, the stability of underlying trends, and the validity of the assumptions made.

Extrapolating too far beyond the range of available data or in the presence of complex, nonlinear relationships can lead to unreliable or misleading results.

Therefore, while extrapolation can be a useful tool for making rough estimates or short-term predictions, it should be approached with caution and supplemented with other analytical methods and validation techniques whenever possible.

To **interpolate** or **extrapolate** a point for a linear data sets we will use the first degree newton approximation formula:

$$f_1(x) = f_0(x) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

Note that $\frac{f(x_1) - f(x_0)}{x_1 - x_0}$ is a finite divided-difference approximation of the first derivative, which also represents the slope of the line of the first-degree polynomial approximation.

We can test interpolation or extrapolation as follows :

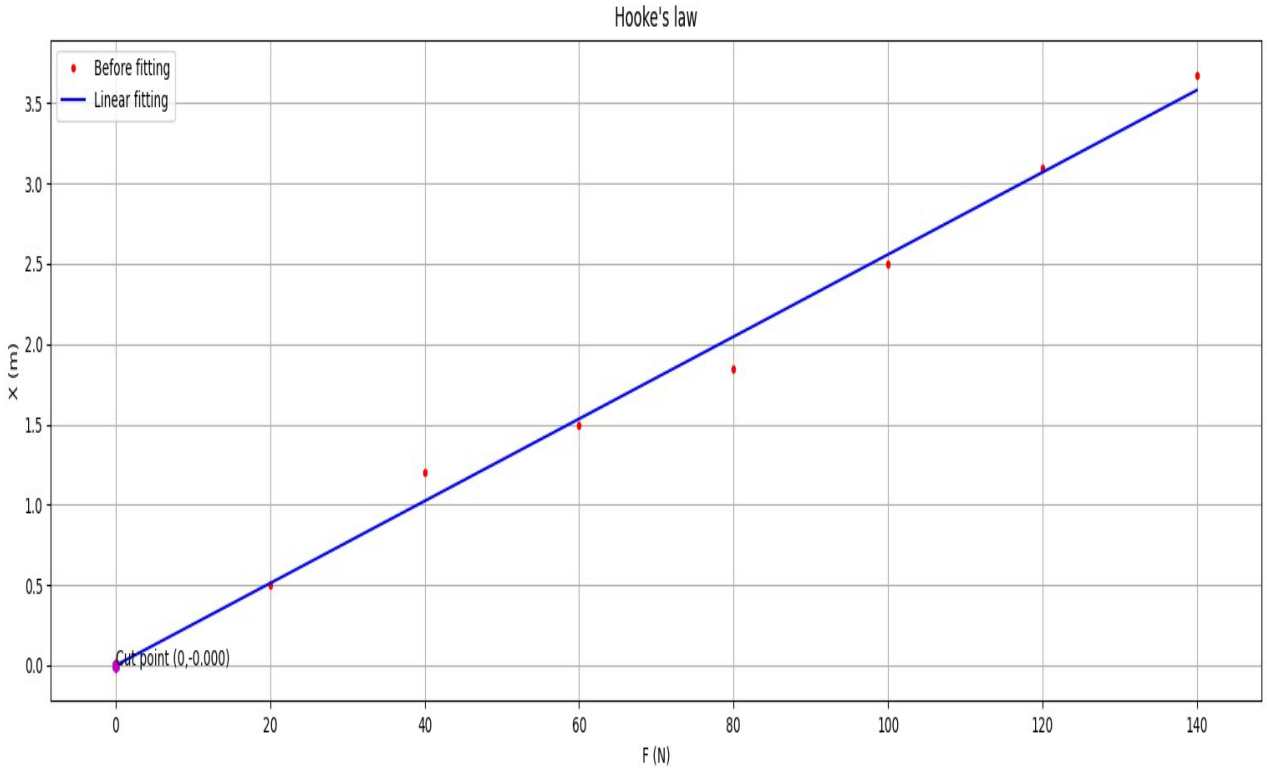
Lets take the extrapolation as an example

With the following hooke’s law expereiment points:

X = [20,40,60,80,100,120,140]

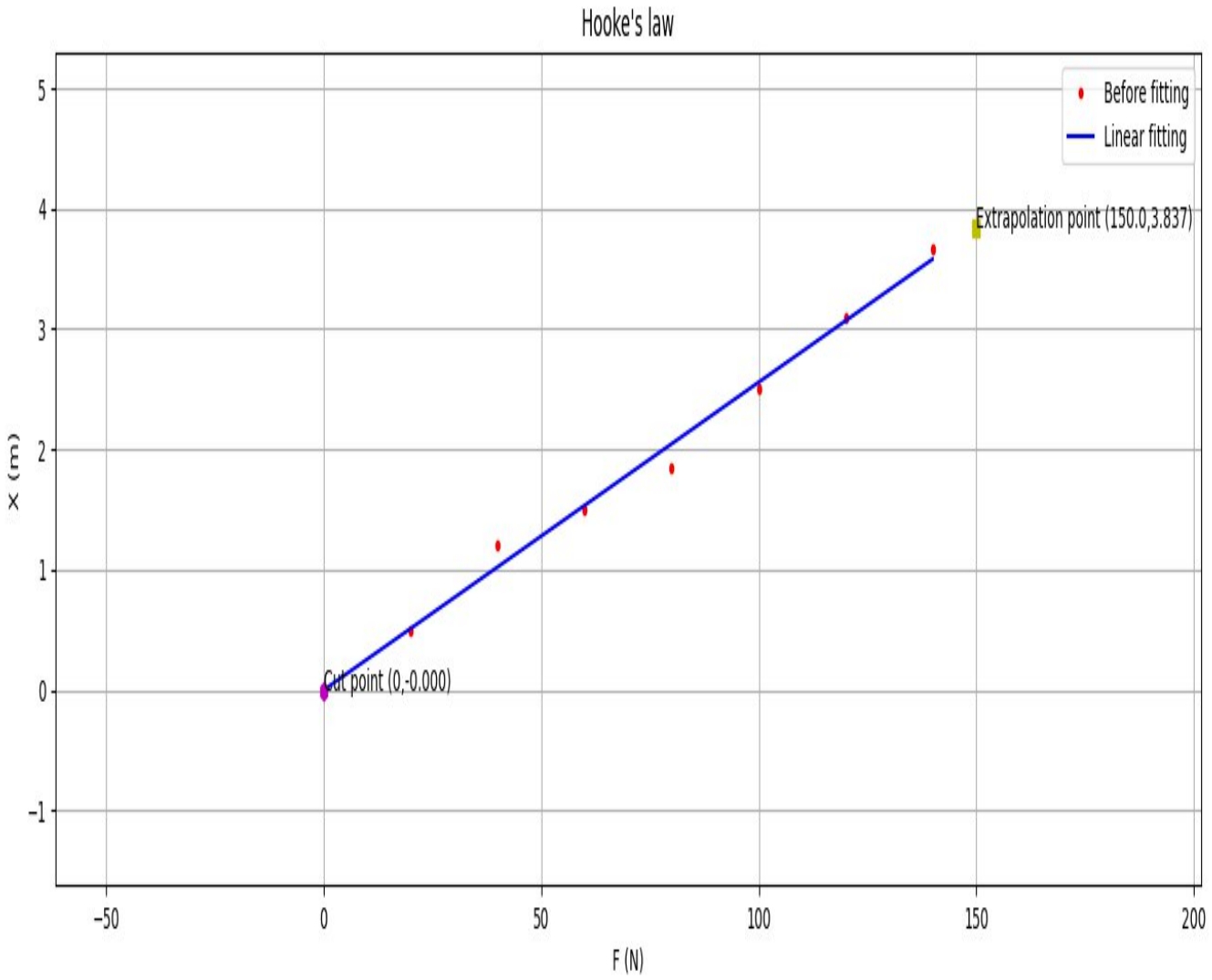
Y = [0.5,1.2,1.5,1.85,2.5,3.1,3.675]

Which gives us after fitting :



If we choose x=150 for extrapolation

By using newton formula, we can plot the result with extrapolation:



And the fitting result with extrapolation is :

Slope : 0.0255803 s^2/gm

Intercept = 0 cm

The extrapolated point = (150,3.837)

The interpolation method will be the same as the extrapolation one except that it will belong to the fitting interval.

Curve fitting:

Introduction:

Curve fitting is the process of finding a mathematical function in an analytic form that best fits this set of data.

The first question that may arise is **why do we need that ?**

There are many cases that curve fitting can prove useful:

- quantify a general trend of the measured data.
- remove noise from a function.
- extract meaningful parameters from the learning curve.
- summarize the relationships among two or more variables.

- In spectroscopy, data may be fitted with:

- Gaussian
- Lorentzian
- Voigt
- Other related functions

The choice of the fitting function depends on the specific characteristics of the data and the physical or chemical processes being studied.

Gaussian distribution :

In statistics, a normal distribution or Gaussian distribution is a type of continuous probability distribution for a real-valued random variable.

When we plot a dataset such as a histogram, the shape of that charted plot is what we call its distribution.

The most commonly observed shape of continuous values is the bell curve, also called the Gaussian or normal distribution.

It is named after the German mathematician Carl Friedrich Gauss.

Some common example datasets that follow Gaussian distribution are X-ray diffraction and photoluminescence in order to determine line widths and other properties.

The general form of its probability density function is :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Where $-\infty \leq x \leq \infty, -\infty \leq \mu \leq \infty, \sigma > 0$. σ^2 is the variance of the distribution and μ is the mean or average of the distribution.

The part:

$$\frac{1}{\sigma\sqrt{2\pi}}$$

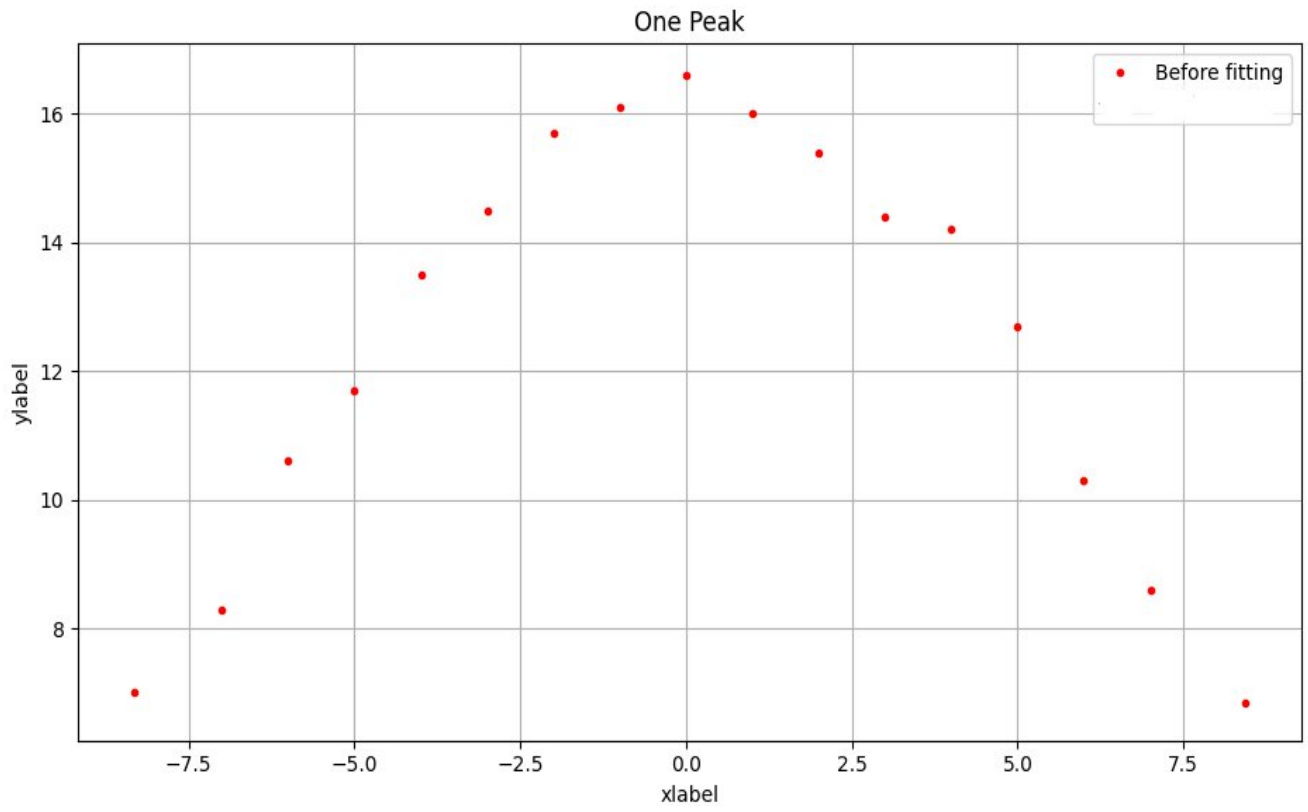
can be considered as the amplitude.

Consider the following points:

X=[-8.33, -7.0, -6.0, -5.0, -4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.44]

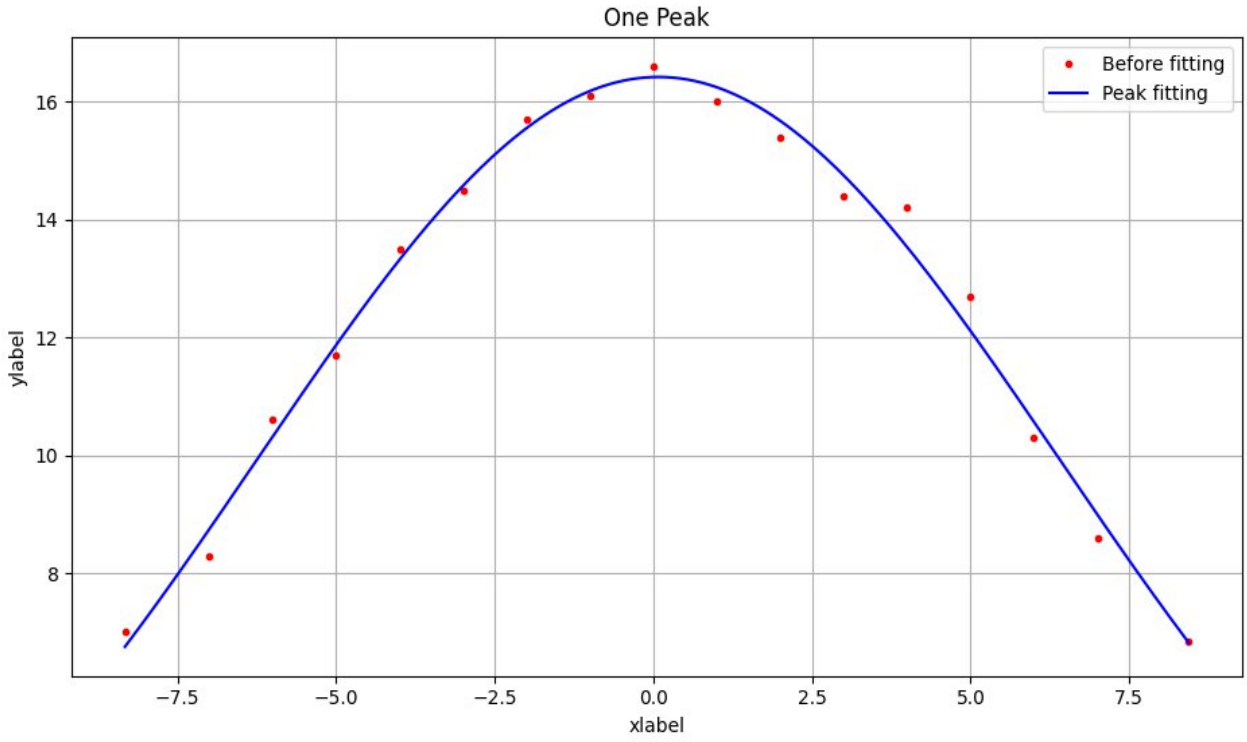
Y=[7.0, 8.3, 10.6, 11.7, 13.5, 14.5, 15.7, 16.1, 16.6, 16.0, 15.4, 14.4, 14.2, 12.7, 10.3, 8.6, 6.84]

Can be plotted as the following:



We want to find the optimal values of the amplitude ,mean and sigma that fit the data above.

By applying gaussian function on the points, that gives us the following result:



And the fitting result is :

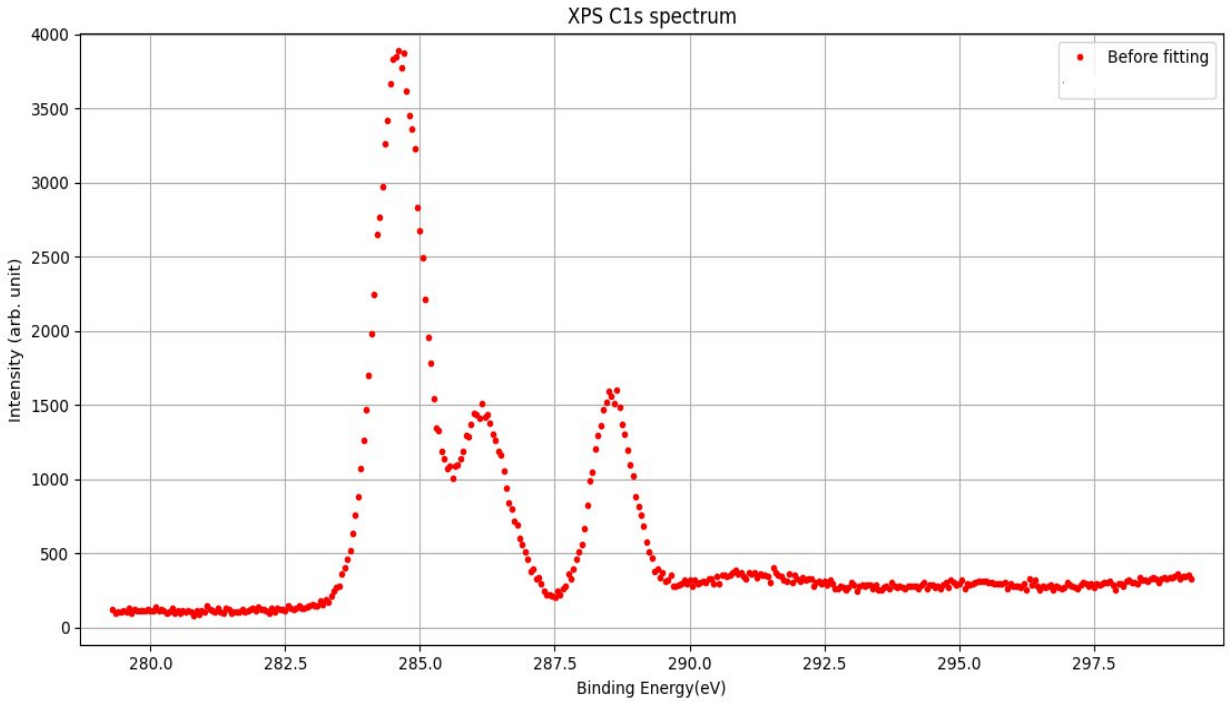
center: 0.07867061 +/- 0.06905517

fwhm: 14.8517187 +/- 0.22931927

height: 16.4218349 +/- 0.14335344

sigma: 6.30694434 +/- 0.09738293

If we have a more complex example like the image below:



That’s need a more complex model

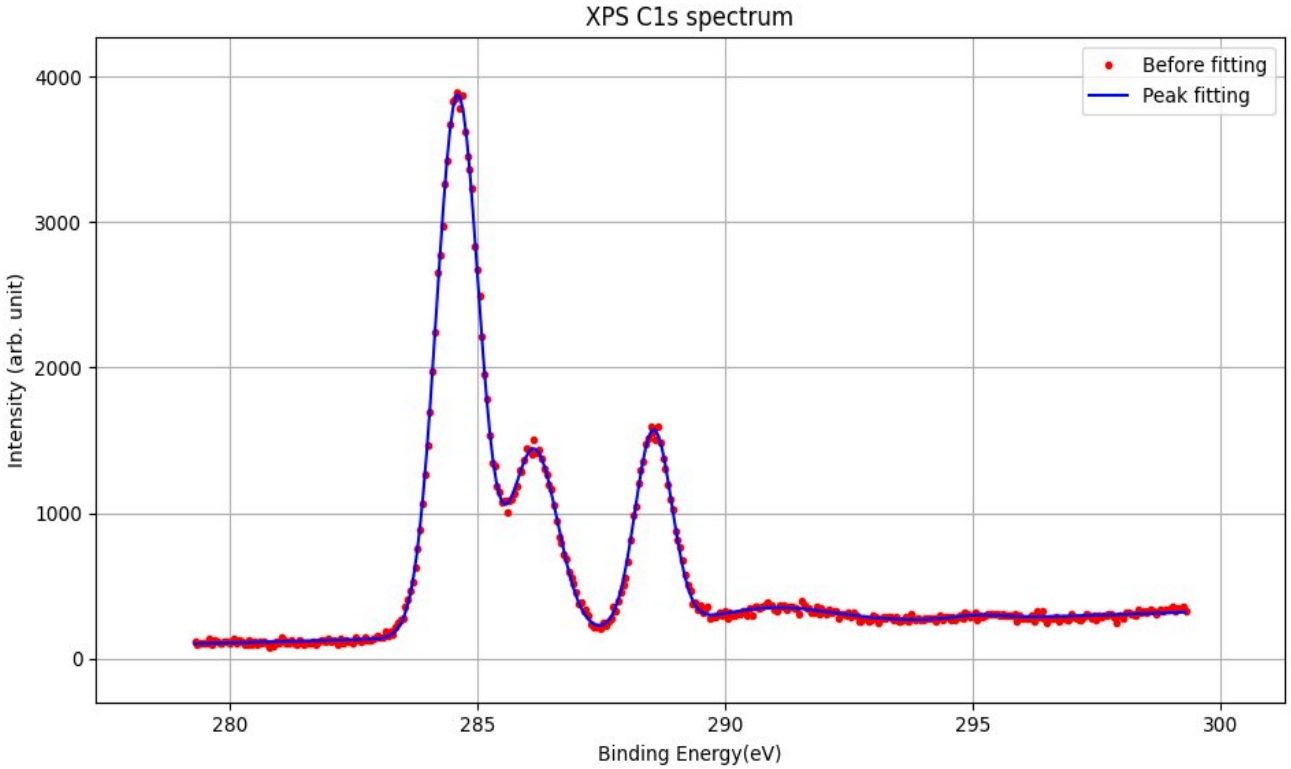
So we will use (Composite gaussian model) to apply gaussian model for every peak with the following initial guesses for the parameters (the last three guesses are background points):

Amplitude = [3870,1490,1590,350,316,356]

Mean = [284.56,286.11,288.57,291.33, 295.39 ,298.98]

Sigma = [1,1,1,3,10,10]

and the image below will be the result:



And the fitting result is:

peak1_center: 284.608380 +/- 0.00132443

peak1_fwhm: 1.00666023 +/- 0.00332189

peak1_height: 3723.60375 +/- 8.90207580

peak1_sigma: 0.42748925 +/- 0.00141067

peak2_center: 286.148459 +/- 0.00406750

peak2_fwhm: 1.11348405 +/- 0.01110725

peak2_height: 1271.12562 +/- 9.05365056

peak2_sigma: 0.47285315 +/- 0.00471681

peak3_center: 288.560908 +/- 0.00305425

peak3_fwhm: 0.86876038 +/- 0.00801579

peak3_height: 1356.59893 +/- 10.0283242

peak3_sigma: 0.36892857 +/- 0.00340399

For more accuracy for the composite model we give it initial guesses at the background points too.

That gives us the following results:

peak4_center: 295.125096 +/- 0.16164702

peak4_fwhm: 1.24749048 +/- 0.47057135

peak4_height: 28.1504220 +/- 8.71267418

peak4_sigma: 0.52976044 +/- 0.19983326

peak5_center: 290.921913 +/- 0.07113044

peak5_fwhm: 3.05715470 +/- 0.26354108

peak5_height: 128.518442 +/- 8.80025054

peak5_sigma: 1.29825409 +/- 0.11191560

peak6_center: 313.929000 +/- 1.99742797

peak6_fwhm: 48.8683479 +/- 11.8489739

peak6_height: 408.126497 +/- 118.156141

peak6_sigma: 20.7524770 +/- 5.03179601

Lorentzian distribution:

The Lorentzian distribution, also known as the Cauchy distribution, is a continuous probability distribution named after the French mathematician Augustin-Louis Cauchy. It is often used to model certain phenomena in physics and statistics.

The Lorentzian distribution has several notable characteristics:

- Heavy tails: Unlike many other distributions, the Lorentzian distribution has infinite variance, meaning it has heavy tails.

  This property indicates that extreme values are more likely to occur compared to other distributions with finite variance.

- No moments: The moments of the Lorentzian distribution do not exist because the integral of x^n multiplied by the Lorentzian

PDF does not converge for any positive integer n.

- Symmetry: The Lorentzian distribution is symmetric around its peak (x_0). The probability of observing a value on the left side of x_0 is equal to the probability of observing the same value on the right side.

The Lorentzian distribution arises in various fields, such as physics, specifically in quantum mechanics and spectroscopy.

It is used to model resonance phenomena, such as energy levels in atoms and the spectral lines of particles.

In statistics, the Lorentzian distribution is sometimes used as a theoretical model for certain types of data, although it is less commonly used compared to other distributions like the Gaussian (normal) distribution.

The general form of lorentzian function is:

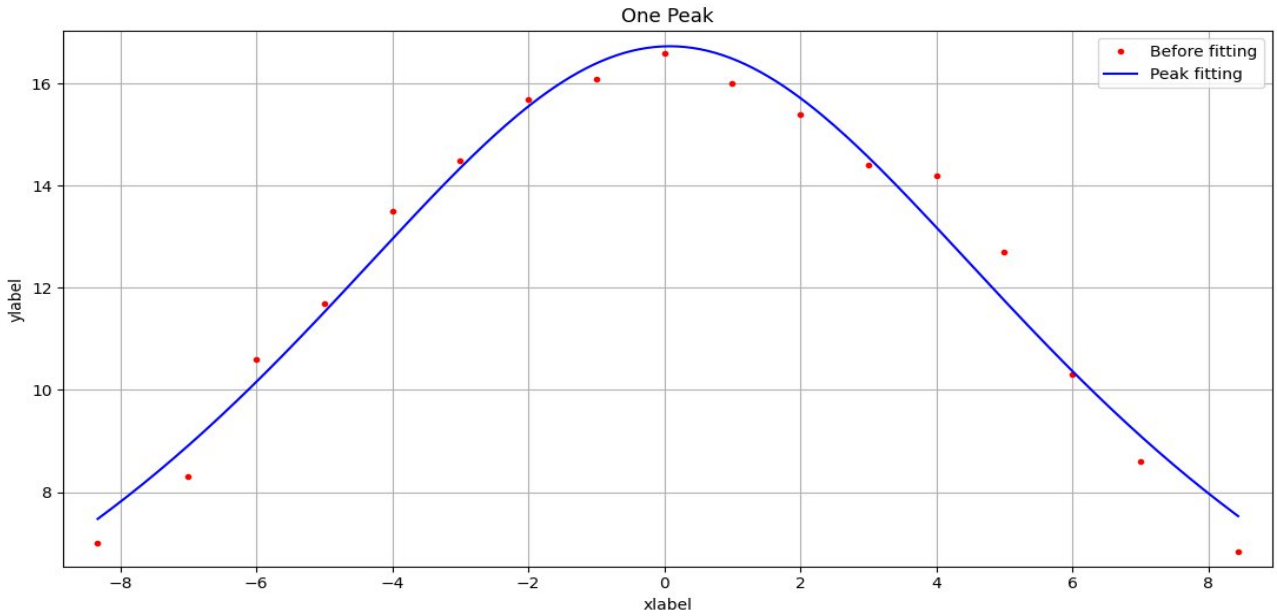
$$f(x; A, \mu, \sigma) = \frac{A}{\pi}\left[\frac{\sigma}{(x-\mu)^2 + \sigma^2}\right]$$

Where $-\infty \leq x \leq \infty, -\infty \leq \mu \leq \infty, \sigma > 0$. σ^2 is the variance of the distribution and μ is the mean or average of the distribution.

And A is the amplitude.

If we used the one peak example that we used with gaussian distribution previous example but with lorentzian function instead.

That gives us the following result:



And the fitting result is :

center: 0.08267810 +/- 0.11651662

fwhm: 15.1123113 +/- 0.49164474

height: 16.7372339 +/- 0.25243151

sigma: 7.55615564 +/- 0.24582237

We can see that the result is slightly changed from the result of gaussian function because of

Lorentzian function has much wider tails than Gaussian distribution.

…We can use composite model with lorentzian model like what we did with gaussian model.

Voigt distribution:

The Voigt distribution, also known as the Voigt profile or the Voigt function, is a probability distribution that arises in the study of spectroscopy and signal processing.

It is named after the German physicist Woldemar Voigt.

The Voigt distribution is a convolution of a Gaussian distribution (or Gaussian profile) and a Lorentzian distribution (or Lorentzian profile).

The Gaussian distribution represents the effects of random noise or instrumental broadening, while the Lorentzian distribution represents the natural line broadening or resonance of the system.

The Voigt distribution provides a more accurate model for the observed line shapes in many

physical systems, especially when both Gaussian and Lorentzian broadening mechanisms are present.

Mathematically, the probability density function (PDF) of the Voigt distribution is given by the convolution of the PDFs of the Gaussian and Lorentzian distributions.

However, the Voigt function does not have a simple closed-form expression and is typically calculated using numerical methods or specialized algorithms.

The Voigt distribution finds applications in various fields, including spectroscopy, astronomy, crystallography, and signal processing.

It is commonly used to analyze and interpret experimental data, particularly in situations

where the observed line profiles are influenced by both Gaussian and Lorentzian broadening.

The general form of lorentzian function is:

$$f(x; A, \mu, \sigma, \gamma) = \frac{A\mathrm{Re}[w(z)]}{\sigma\sqrt{2\pi}}$$

Where

$$z = \frac{x - \mu + i\gamma}{\sigma\sqrt{2}}$$
$$w(z) = e^{-z^2}\mathrm{erfc}(-iz)$$

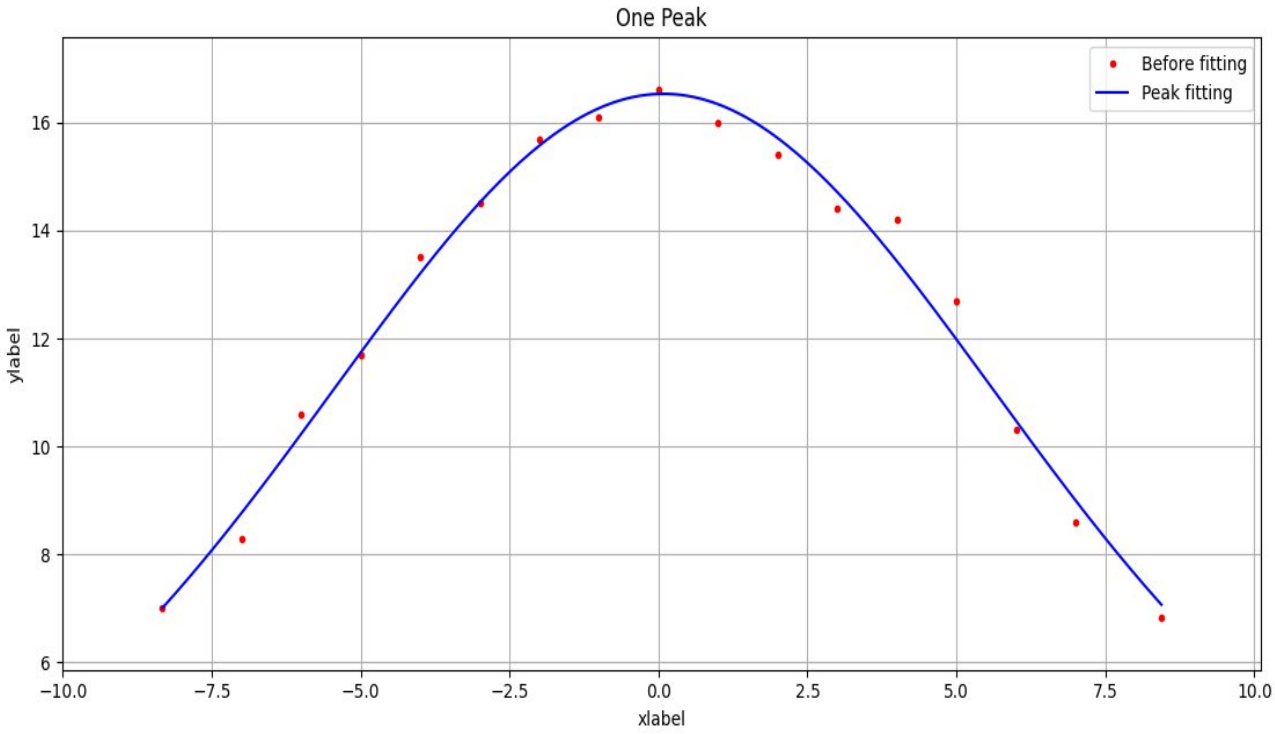
and erfc() is the complementary error function. As above, amplitude corresponds to A, center to μ, and sigma to σ.

The parameter gamma corresponds to ɣ.

If gamma is kept at the default value (constrained to sigma), the full width at half maximum is approximately 3.6013σ .

If we used the one peak example that we used with gaussian distribution before but with voigt function instead.

That gives us the following result:



And the fitting result is :

center: 0.08138378 +/- 0.07887461

fwhm: 14.8821118 +/- 0.28397769

gamma: 4.13243843 +/- 0.07885442

height: 16.5324930 +/- 0.16565426

sigma: 4.13243843 +/- 0.07885442

We can see that the result is slightly changed from the result of gaussian and lorentzian functions and we have a new parameter in the result which is **gamma.**

…We can use composite model with voigt model like what we did with gaussian model.

Python Desktop application:

Python langauage:

Python is a high-level, interpreted programming language that is widely used for general-purpose programming. It was created by Guido van Rossum and released in 1991.
Python is known for its simplicity, readability, and a large ecosystem of libraries and frameworks that make it suitable for a wide range of applications.

Here are some key features and aspects of Python:

- Readability: Python emphasizes code readability with its clean and straightforward syntax.
  It uses indentation to define code blocks, making it easy to understand and maintain.
- Object-Oriented: Python supports object-oriented programming (OOP) paradigms.

It allows you to define classes, create objects, and implement inheritance, encapsulation, and polymorphism.

- Large Ecosystem: Python has a vast ecosystem of third-party libraries and frameworks that extend its capabilities. Popular libraries like NumPy, Pandas, Matplotlib, and TensorFlow enable efficient scientific computing, data analysis, and machine learning.
- Cross-Platform Compatibility: Python is available on various operating systems, including Windows, macOS, and Linux.
  This allows you to write code once and run it on different platforms without major modifications.
- Physics ,Data Analysis and Machine Learning: Python has become a popular language for data analysis and machine

learning due to its libraries like NumPy, Pandas, and scikit-learn.
These libraries offer efficient data structures, data manipulation tools, and machine learning algorithms.

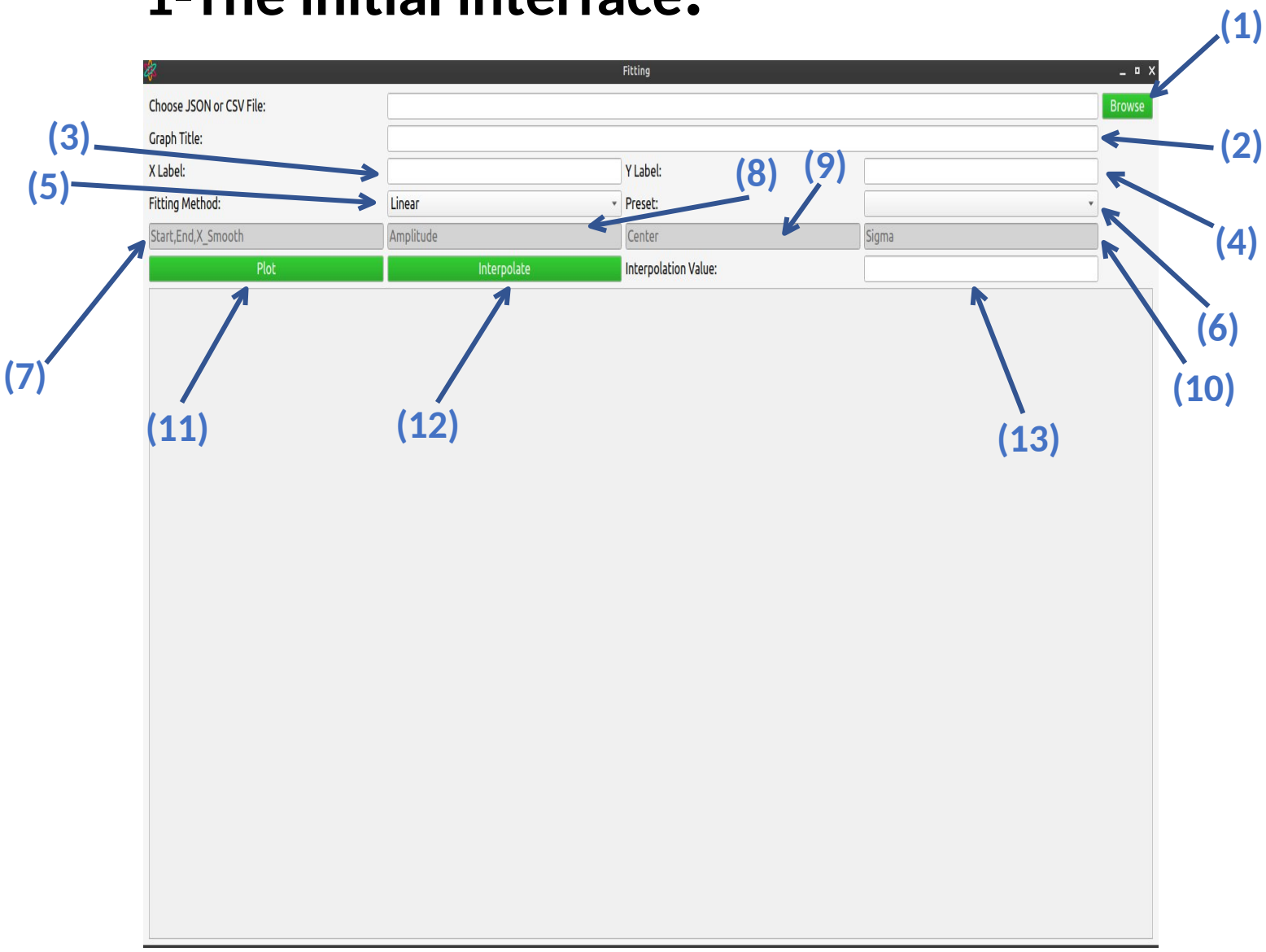
- Community and Support: Python has a large and active community of developers who contribute to its growth. The community provides extensive documentation, tutorials, and forums, making it easy to find help and resources when needed.

Python's versatility and simplicity have made it a favorite language for beginners and experienced programmers alike.
Its usage spans across various domains, including web development, scientific computing, data analysis, artificial intelligence, and automation.

Desktop application:

An application for linear and curve fitting with the ability to use linear interpolation or extrapolation.

1-The initial interface:



Notes:-

A) The fields (7),(8),(9) and (10) are enabled only when using curve fitting methods.

B) The button (12) and the field (13) are enabled only when using linear method.

(1) The first step to start the application by choosing the dataset file(CSV :a generated file by XPS device for example and JSON: a generated file by modern programming languages).

(2) Graph title (experiment title).

(3)X-axis label.

(4)Y-axis label.

(5) Choosing fitting method depending on the dataset (the available methods are :[linear,gaussian,lorentzian,voigt]).

(6) Add Preset with the experiment result (the available presets are:[Hooke's law,Simple Pendulum] for adding the measuring units and the experiment requirements).

(7) limit x interval with start and end and you can choose the number of fitting points (x_smooth) for more curve smoothness with the big datasets (start ,end and x_smooth all separated by comma).

(8) Add the initial guesses for the amplitude parameter separated by comma.

(9) Add the initial guesses for the center(mean) parameter separated by comma.

(10) Add the initial guesses for sigma parameter separated by comma.

(11) Plot the points when all the fields are filled except the interpolation value field.

(12) Plot the points with interpolation when all the fields are filled with the interpolation value field.

(13) Add the interpolation or extrapolation value before pressing button (12) (when pressing enter in this field interpolate button will be pressed too).

2-The dataset file structure(partially optional):

A)CSV file:

By opening the file with excel and choosing the comma delimiter then the structure of the file should be like this:

	A	B	C	D	E	F
1	x	y	xlabel	ylabel	title	
2	11	0.403	L	T^2	Simple pendulum	
3	22	0.81				
4	29	1.21				
5	38	1.6				
6	48	2.015				
7	63	2.42				
8						

Where A1=x and B1=y are required for the program to read the file.

And the columns C,D and E every one of them is optional (but will be good for program to do autofill the fields of the experiment).

B) JSON file:

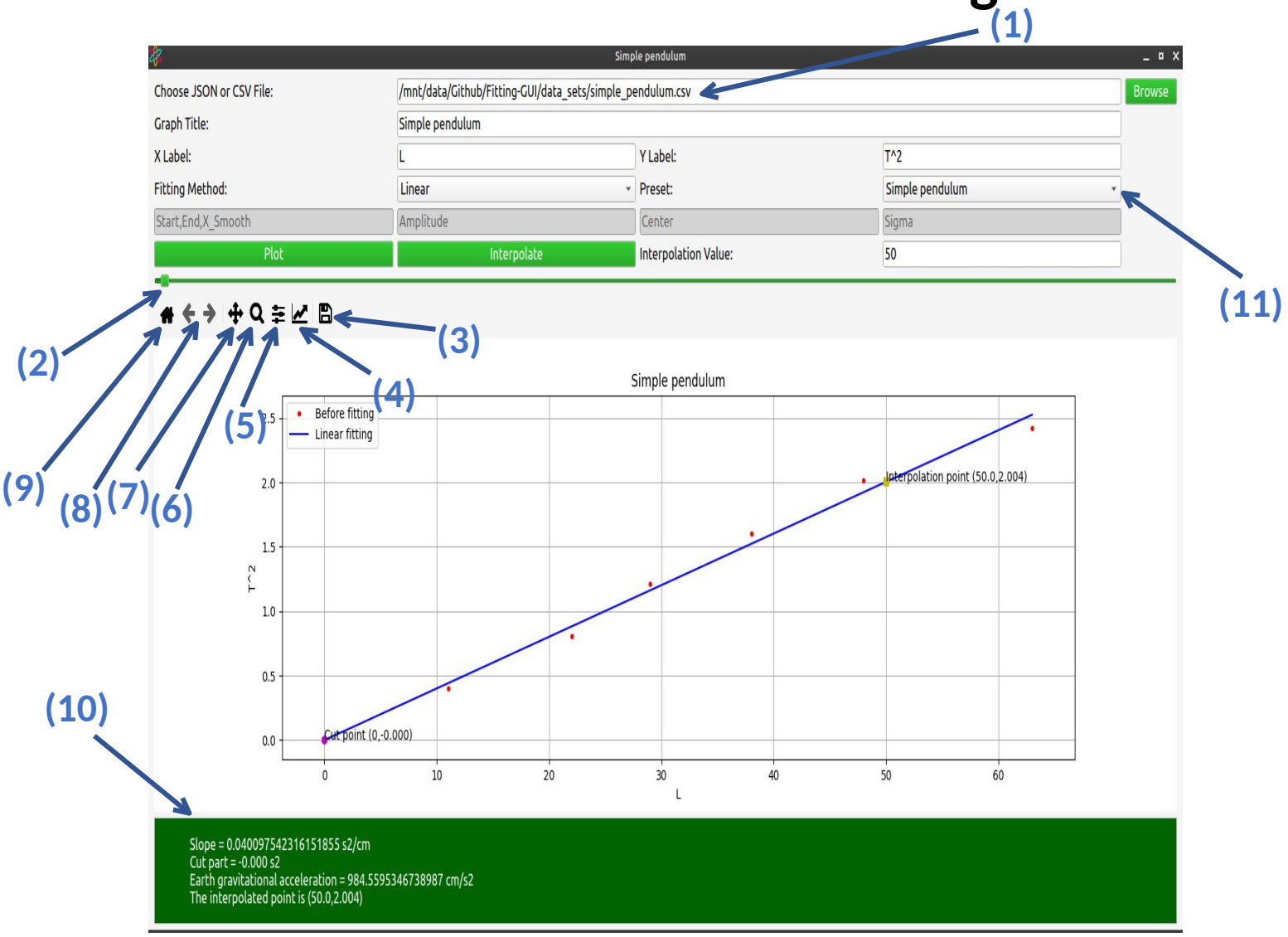
By opening the file with any text editor then the structure of the file should be like this if we compared with previous CSV file constraints:

```
{
  "title":"Simple pendulum",
  "xlabel":"L",
  "ylabel":"T^2",
  "x": [11,22,29,38,48,63],
  "y": [0.403,0.81,1.21,1.6,2.015,2.42]
}
```

Where x and y keys are required for the program to read the file

And the keys title,xlabel and ylabel every one of them is optional (but will be good for program to do autofill the fields of the experiment).

3-The interface with linear fitting:



(1) Autofilled dataset file path.

(2) Zoom-in and zoom-out for the figure.

(3) Save png image for the figure.

(4) Edit the figure like changing the line or the points color and shape and more.

(5) configure subplots.

(6) zoom for any custom section in the figure.

(7) move the figure to any direction with mouse left click or zoom in or out for one axis like x or y with mouse right click.

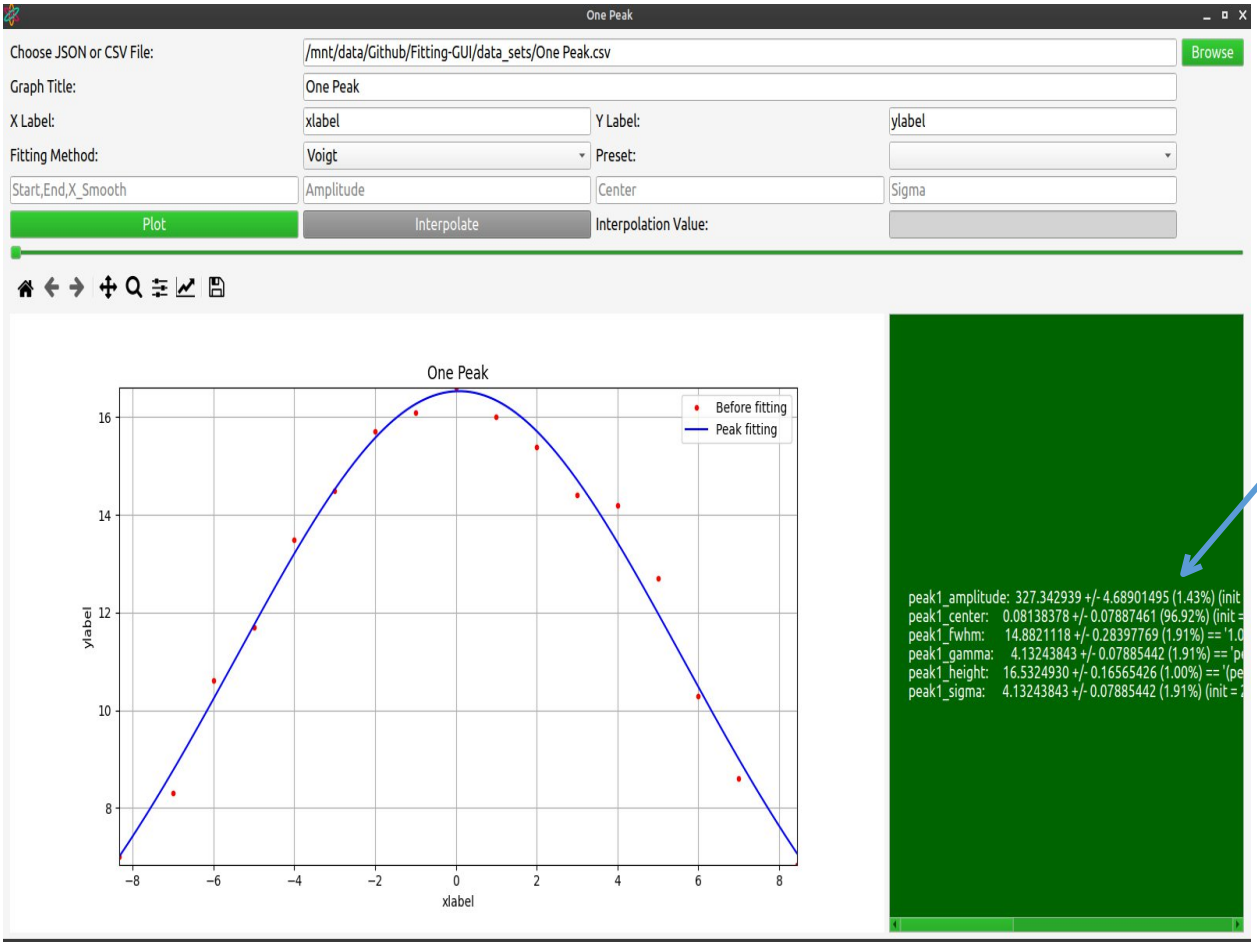
(8)back or forward to a view that changed by the buttons (6) and (7).

(9)reset to original view.

(10) The linear experiment result.

(11) Autofilled preset because it matches the title in the dataset file.

4-The interface with curve fitting:

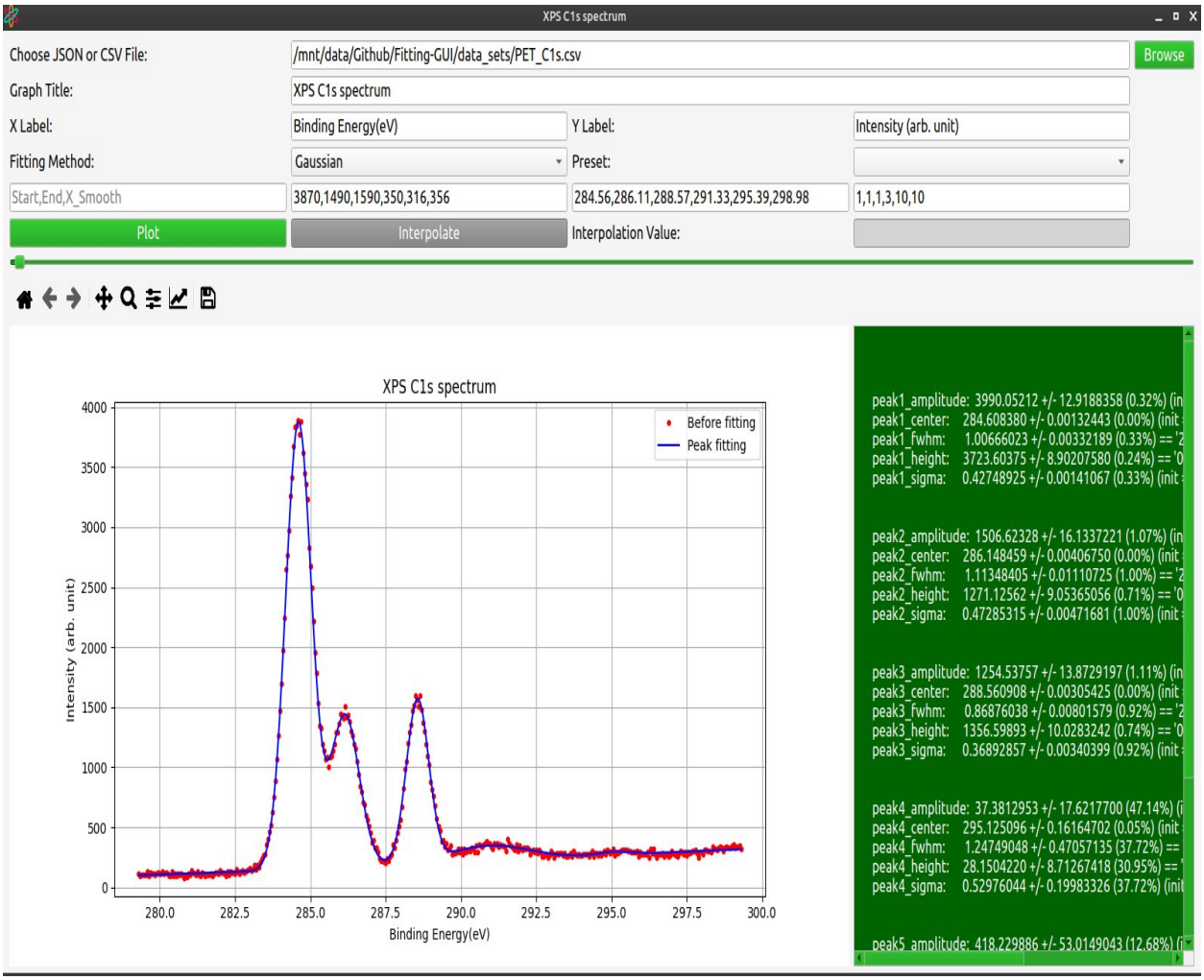


Note:-

One peak doesn't need initial guesses in the fields Amplitude,Center and Sigma.

(1) Peak fitting result.

A complex curve with initial guesses:



5-The application full code:

The main structure of the project is the following files:

(main.py , fitting_functions.py and interpolation_functions.py)

Where main.py is the execution file of the project.

1)main.py

Importing the used libraries in the file:

```
#!/usr/bin/python3


import sys
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg
as FigureCanvas
from matplotlib.backends.backend_qt5agg import
NavigationToolbar2QT as NavigationToolbar
from PyQt5.QtWidgets import QApplication, QMainWindow, QLabel,
QGridLayout, QWidget, QPushButton, QFileDialog, QLineEdit,
QMessageBox,QComboBox,QSlider,QScrollArea
from PyQt5.QtGui import QIcon, QPalette, QColor,QCursor
from PyQt5.QtCore import Qt
from fitting_functions import *
import interpolation_functions
import os.path as path
import pandas as pd
```

Constructing the GUI:

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()


        self.app_dir=path.dirname(__file__)


        MainWindow.linear_fit=linear_fit
        MainWindow.gaussian_fit=gaussian_fit
        MainWindow.lorentzian_fit=lorentzian_fit
        MainWindow.voigt_fit=voigt_fit
        MainWindow.def1=interpolation_functions.def1
        MainWindow.newton1=interpolation_functions.newton1
        MainWindow.newton2=interpolation_functions.newton2


        # Set window properties
        self.setWindowTitle("Fitting")
        self.setStyleSheet("""
        *{
            font-size:16px;
        }
        QPushButton:disabled{
            background-color:grey;
        }
        QLineEdit:disabled{
            background-color:lightgrey
        }
        QMainWindow{
            background:rgb(245,245,245);
        }
        QPushButton::hover{
        background:green
```

```
        }


        """)


self.setWindowIcon(QIcon(path.join(self.app_dir,'assets','icon.pn
g')))
        self.setGeometry(100, 100, 800, 720)


        # Create central widget and grid layout
        central_widget = QWidget(self)
        self.setCentralWidget(central_widget)
        self.grid_layout = QGridLayout(central_widget)

        # Create labels and line edits for file input
        file_label = QLabel("Choose JSON or CSV File:")
        self.file_edit = QLineEdit()
        self.file_edit.setReadOnly(True)
        file_button = QPushButton("Browse")
        file_button.clicked.connect(self.browse_file)
        file_button.setCursor(QCursor(Qt.PointingHandCursor))


        # Create labels and line edits for plot input
        title_label = QLabel("Graph Title:")
        self.title_edit = QLineEdit()
        xlabel_label = QLabel("X Label:")
        self.xlabel_edit = QLineEdit()
        ylabel_label = QLabel("Y Label:")
        self.ylabel_edit = QLineEdit()


        fitting_method_label=QLabel("Fitting Method:")
        self.fitting_method_edit=QComboBox()
```

```
self.fitting_method_edit.addItems(["Linear","Gaussian","Lorentzia
n","Voigt"])

self.fitting_method_edit.currentIndexChanged.connect(self.disable
_enable_interpolation)
        experiment_label=QLabel("Preset:")
        self.experiment_edit=QComboBox()
        self.experiment_edit.addItems(["","Simple
pendulum","Hooke's law"])


        # will be used if we used custom fitting functions
instead of the models
        self.start_end_xsmooth_edit = QLineEdit()

self.start_end_xsmooth_edit.setPlaceholderText('Start,End,X_Smoot
h')
        self.amplitude_edit = QLineEdit()
        self.amplitude_edit.setPlaceholderText('Amplitude')
        self.center_edit = QLineEdit()
        self.center_edit.setPlaceholderText('Center')
        self.sigma_edit = QLineEdit()
        self.sigma_edit.setPlaceholderText('Sigma')


        # Create buttons for plot and interpolation
        plot_button = QPushButton("Plot")
        plot_button.clicked.connect(self.plot)
        plot_button.setCursor(QCursor(Qt.PointingHandCursor))
        self.interp_button = QPushButton("Interpolate")
        self.interp_button.clicked.connect(self.interpolate)

self.interp_button.setCursor(QCursor(Qt.PointingHandCursor))
        # Create label for interpolation input
        self.interp_label = QLabel("Interpolation Value:")
        self.interp_edit = QLineEdit()
        # When pressing enter button
```

```
self.interp_edit.returnPressed.connect(self.interp_button.click)


        # Create slider for graph zoom
        self.slider=QSlider(Qt.Horizontal)
        self.slider.setRange(0,1000)
        self.slider.setValue(0)
        self.slider.valueChanged.connect(self.graph_draw_zoom)


        # Create figure canvas for plot output
        self.canvas = FigureCanvas(plt.Figure())


        # Create label for results
        self.result_label = QLabel()
        self.scroll_area = QScrollArea(self)
        self.scroll_area.setWidgetResizable(True)
        self.scroll_area.setWidget(self.result_label)


        # Add widgets to grid layout
        self.grid_layout.addWidget(file_label, 0, 0)
        self.grid_layout.addWidget(self.file_edit, 0, 1, 1, 3)
        self.grid_layout.addWidget(file_button, 0, 4)


        self.grid_layout.addWidget(title_label, 1, 0)
        self.grid_layout.addWidget(self.title_edit, 1, 1, 1, 3)


        self.grid_layout.addWidget(xlabel_label, 2, 0)
        self.grid_layout.addWidget(self.xlabel_edit, 2, 1)
        self.grid_layout.addWidget(ylabel_label, 2, 2)
        self.grid_layout.addWidget(self.ylabel_edit, 2, 3)


        self.grid_layout.addWidget(fitting_method_label, 3, 0)
```

```
        self.grid_layout.addWidget(self.fitting_method_edit, 3,
1)
        self.grid_layout.addWidget(experiment_label, 3, 2)
        self.grid_layout.addWidget(self.experiment_edit, 3, 3)



        # will be used if we used custom fitting functions
instead of the models

self.grid_layout.addWidget(self.start_end_xsmooth_edit,4,0)
        self.grid_layout.addWidget(self.amplitude_edit,4,1)
        self.grid_layout.addWidget(self.center_edit,4,2)
        self.grid_layout.addWidget(self.sigma_edit,4,3)
        self.start_end_xsmooth_edit.setEnabled(False)
        self.amplitude_edit.setEnabled(False)
        self.center_edit.setEnabled(False)
        self.sigma_edit.setEnabled(False)



        self.grid_layout.addWidget(plot_button, 5, 0)
        self.grid_layout.addWidget(self.interp_button, 5, 1)
        self.grid_layout.addWidget(self.interp_label, 5, 2)
        self.grid_layout.addWidget(self.interp_edit, 5, 3)



        self.grid_layout.addWidget(self.scroll_area, 9, 0, 1, 5)
```

The method of reading the dataset file and autofilling fields :

```
    def browse_file(self):
        # Open file dialog and get selected file path
        file_path,_= QFileDialog.getOpenFileName(self, "Open JSON
or CSV File", path.join(self.app_dir,'data_sets'), "JSON or CSV
Files (*.json  *.csv)")


        if file_path:
            file_ext = path.splitext(file_path)[1]
            # Read  file and set default input values
            if(file_ext==".csv"):
                file= pd.read_csv(file_path)
            elif(file_ext==".json"):
                file= pd.read_json(file_path)


            try:
                title=file['title'][0]
                self.title_edit.setText(title)
                self.experiment_edit.setCurrentIndex(0)
                for i in range(self.experiment_edit.count()):
                    if(self.experiment_edit.itemText(i)==title):
                        self.experiment_edit.setCurrentIndex(i)
            except KeyError:
                self.title_edit.setText("")
                self.experiment_edit.setCurrentIndex(0)

            try:
                xlabel=file['xlabel'][0]
                self.xlabel_edit.setText(xlabel)
            except KeyError:
                self.xlabel_edit.setText("")
```

```
            try:
                ylabel=file['ylabel'][0]
                self.ylabel_edit.setText(ylabel)
            except KeyError:
                self.ylabel_edit.setText("")


            try:
                self.x = file['x']
                self.y = file['y']
                indexes = list(range(len(self.x)))
                indexes.sort(key=self.x.__getitem__)

self.x=np.array(list(map(self.x.__getitem__,indexes)))

self.y=np.array(list(map(self.y.__getitem__,indexes)))
                # those value values to restore x and y when
errors happens
                self.x_temp=self.x
                self.y_temp=self.y
            except KeyError as error:
                self.title_edit.setText("")
                self.experiment_edit.setCurrentIndex(0)
                self.xlabel_edit.setText("")
                self.ylabel_edit.setText("")
                QMessageBox.warning(self, "Data error", f"Please
provide a valid {error} column name in the chosen file .")
                return


            self.n = len(self.x)
            self.file_edit.setText(file_path)
```

The shared method between the regular plotting and the plotting with interpolation:

```
def shared_plot(self):
        # Get input values
        filename = self.file_edit.text()
        xlabel = self.xlabel_edit.text()
        ylabel = self.ylabel_edit.text()
        title = self.title_edit.text()
        self.setWindowTitle(title)



        # Check if all input values are provided
        if not all([filename, xlabel, ylabel, title]):
            QMessageBox.warning(self, "Error", "Please provide
all input values.")
            return
        experiment=self.select_fitting_method()
        self.result = f"{self.add_experiment_result()}"
        self.x=self.x_temp
        self.y=self.y_temp

        # Clear previous plot and draw new plot on canvas
        self.canvas.figure.clear()
        self.ax = self.canvas.figure.add_subplot(111)
        self.ax.plot(experiment.x,
experiment.y,'ro',markersize=3,label="Before fitting")
        self.ax.plot(experiment.x_smooth,
experiment.y_fit,'b',label=experiment.after_fitting_label)



        # Plot cut part point
        if(self.fitting_method_edit.currentText()=="Linear"):
            self.ax.plot(0, experiment.c, "h m")
            self.ax.text(0, experiment.c, f"Cut point
(0,{experiment.c:.3f})")
            self.result_label.setText(self.result)
```

```
self.result_label.setStyleSheet("background:darkgreen;color:white
;padding:5px 50px;font-size:16px;")
        else:
            self.result_label.setText(self.peak_result)

self.result_label.setStyleSheet("background:darkgreen;color:white
;padding:5px;font-size:16px;")


        self.min_x=min(*experiment.x,*experiment.x_smooth)
        self.max_x=max(*experiment.x,*experiment.x_smooth)
        self.min_y=min(*experiment.y,*experiment.y_fit)
        self.max_y=max(*experiment.y,*experiment.y_fit)


        self.ax.legend(loc="best")
        self.ax.set_xlabel(xlabel)
        self.ax.set_ylabel(ylabel)
        self.ax.set_title(title)
        self.ax.grid()
```

The method for plotting without interpolation:

```
    def plot(self):
        self.shared_plot()
        self.graph_draw_zoom()
```

The method for plotting with interpolation:

```
 # plot with interpolation
    def interpolate(self):
        interp_value = self.interp_edit.text()
        try:
```

```
            interp_value = float(interp_value)
        except ValueError:
            QMessageBox.warning(self, "Error", "Please provide a
valid interpolation value.")
            return
        self.shared_plot()

        experiment=self
        y_interp= experiment.newton1(interp_value)
        # Plot interpolated/extrapolated point
        self.ax.plot(interp_value, y_interp, "s y")


        # Determine if interpolated/extrapolated point is within
plot range
        interp_point=f"({interp_value},{y_interp:.3f})"
        if min(experiment.x) ≤interp_value ≤ max(experiment.x):
            interpolation_text=f"The interpolated point is
{interp_point}"
            self.ax.text(interp_value, y_interp, f"Interpolation
point {interp_point}")
        else:
            interpolation_text=f"The extrapolated point is
{interp_point}"
            self.ax.text(interp_value, y_interp, f"Extrapolation
point {interp_point}")
        self.result = f"{self.result}\n{interpolation_text}"
        self.result_label.setText(self.result)


        self.graph_draw_zoom()
```

The method for disabling or enabling the parameters fields and the interpolation field depending on the selected fitting method:

```
    def disable_enable_interpolation(self):
        if(self.fitting_method_edit.currentText()=="Linear"):
            self.interp_button.setEnabled(True)
            self.interp_edit.setEnabled(True)
            self.start_end_xsmooth_edit.setEnabled(False)
            self.amplitude_edit.setEnabled(False)
            self.center_edit.setEnabled(False)
            self.sigma_edit.setEnabled(False)
        else:
            self.interp_button.setEnabled(False)
            self.interp_edit.setEnabled(False)
            self.start_end_xsmooth_edit.setEnabled(True)
            self.amplitude_edit.setEnabled(True)
            self.center_edit.setEnabled(True)
            self.sigma_edit.setEnabled(True)
```

The method for changing the place of experiment result label depending on the fitting method:

```
    def select_fitting_method(self):
        current_method = self.fitting_method_edit.currentText()


        if current_method == "Linear":
            self.linear_fit()
            self.after_fitting_label="Linear fitting"
```

```
            self.grid_layout.removeWidget(self.scroll_area)
            self.grid_layout.addWidget(self.canvas, 8, 0, 1, 5)
            self.grid_layout.addWidget(self.scroll_area, 9, 0, 1,
5)
        else:
            self.grid_layout.removeWidget(self.scroll_area)
            self.grid_layout.addWidget(self.canvas, 8, 0, 2, 3)
            self.grid_layout.addWidget(self.scroll_area, 8, 3, 2,
2)
            if current_method == "Gaussian":
                self.gaussian_fit()
            elif current_method == "Lorentzian":
                self.lorentzian_fit()
            elif current_method == "Voigt":
                self.voigt_fit()
            self.after_fitting_label="Peak fitting"


        return self
```

The method for adding the linear experiment result depending on The selected preset:

```
    def add_experiment_result(self):
        current_experiment=self.experiment_edit.currentText()
        result=""
        if(self.fitting_method_edit.currentText()=="Linear"):
            if(current_experiment=="Simple pendulum"):
                g = 4*(np.pi**2)/self.m
                result=f"Slope = {self.m} s2/cm\nCut part =
{self.c:.3f} s2\nEarth gravitational acceleration = {g} cm/s2"
            elif(current_experiment=="Hooke's law"):
                result=f"Slope = {self.m} s2/gm\nCut part =
{self.c:.3f} cm"
```

```
            else:
                result=f"Slope = {self.m}\nCut part =
{self.c:.3f}"
        return result
```

The method for drawing the graph when changing the value of the slider:

```
    def graph_draw_zoom(self):
        lim_percentage=self.slider.value()/100
        self.ax.set_xlim(self.min_x-(self.max_x-
self.min_x)*lim_percentage, self.max_x+(self.max_x-
self.min_x)*lim_percentage)
        self.ax.set_ylim(self.min_y-(self.max_y-
self.min_y)*lim_percentage, self.max_y+(self.max_y-
self.min_y)*lim_percentage)



        self.grid_layout.addWidget(self.slider, 6, 0, 1, 5)
        self.toolbar = NavigationToolbar(self.canvas, self)
        self.toolbar.locLabel.setStyleSheet("color:initial")
        self.grid_layout.addWidget(self.toolbar, 7, 0, 1, 5)
        self.canvas.draw()
```

Some Global styles and the execution of the application:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)

    # Set application style to light mode
    palette = QPalette()
    palette.setColor(QPalette.Window, QColor(240, 240, 240))
    palette.setColor(QPalette.WindowText, Qt.black)
    palette.setColor(QPalette.Base, QColor(255, 255, 255))
    palette.setColor(QPalette.AlternateBase, QColor(240, 240,
240))
    palette.setColor(QPalette.ToolTipBase, Qt.black)
    palette.setColor(QPalette.ToolTipText, Qt.white)
    palette.setColor(QPalette.Text, Qt.black)
    palette.setColor(QPalette.Button, QColor(0,150,0))
    palette.setColor(QPalette.ButtonText, QColor("white"))
    palette.setColor(QPalette.BrightText, Qt.black)
    palette.setColor(QPalette.Link, QColor(0, 0, 255))
    palette.setColor(QPalette.Highlight, QColor("green"))
    palette.setColor(QPalette.HighlightedText, Qt.white)
    app.setPalette(palette)


    window = MainWindow()
    combo_palette = QPalette()
    combo_palette.setColor(QPalette.Button, QColor(240,240,240))
    combo_palette.setColor(QPalette.ButtonText, QColor("black"))
    combo_palette.setColor(QPalette.Highlight,
QColor(0,255,0,100))
    window.fitting_method_edit.setPalette(combo_palette)
    window.experiment_edit.setPalette(combo_palette)


    window.show()
    sys.exit(app.exec_())
```

B)interpolation_functions.py

```
# first derivative function
def def1(self, i):
    return (self.y_fit[i+1]-self.y_fit[i])/(self.x_smooth[i+1]-
self.x_smooth[i])



# first degree newton function
def newton1(self, X):
    return self.y_fit[0]+self.def1(0)*(X-self.x_smooth[0])



# second degree newton function(temporary)
def newton2(self, X):
    return self.newton1(X)+((self.def1(1)-
self.def1(0))/(self.x_smooth[2]-self.x_smooth[0]))*(X-
self.x_smooth[0])*(X-self.x_smooth[1])
```

C)fitting_functions.py

Importing the used libraries:

```
import numpy as np
from lmfit.models import
Model,VoigtModel,LorentzianModel,GaussianModel
from scipy.optimize import curve_fit
from scipy.special import wofz
import re
import copy
```

Linear fitting function:

```
def linear_fit(self):
    # Least squares function
    self.m = (self.n*np.sum(self.x*self.y)-
np.sum(self.x)*np.sum(self.y))/(self.n*np.sum(self.x**2)-
np.sum(self.x)**2)
    self.c = (1/self.n)*(np.sum(self.y)-self.m*np.sum(self.x))
    self.x_smooth = np.linspace(min(self.x.min(),0),
self.x.max(), 300)
    self.y_fit = self.m*self.x_smooth+self.c
```

The shared function between the curve fitting functions that can create one model and composite model:

```
def __private_shared_fitting_body(self,model:Model):
    # params = model.make_params(amplitude=self.amplitude,
mean=self.mean, sigma=self.sigma,gamma=self.gamma)



# ======= creating interval start end x_smooth started
=================================
    start_end_xsmooth_arr=[self.x[0],self.x[-1],300]
    if(self.start_end_xsmooth_edit.text()≠''):

start_end_xsmooth=self.start_end_xsmooth_edit.text().split(',')
        n=len(start_end_xsmooth)
        for i in range(n):
            if(start_end_xsmooth[i]≠''):

start_end_xsmooth_arr[i]=float(start_end_xsmooth[i])
# ======= creating interval start end x_smooth ended
=================================



# ======= creating amplitude center sigma started
=================================
    params_length=0
    params_list=[[],[],[]]
    if(self.amplitude_edit.text()≠'' and
self.center_edit.text()≠'' and self.sigma_edit.text()≠ ''):
        amplitude=self.amplitude_edit.text().split(',')
        center=self.center_edit.text().split(',')
        sigma=self.sigma_edit.text().split(',')
        params_length=len(amplitude)
```

```
        for i in range(params_length):
            params_list[0].append(float(amplitude[i]))
            params_list[1].append(float(center[i]))
            params_list[2].append(float(sigma[i]))



# ======= creating amplitude center sigma ended
=====================================
    model_temp = copy.deepcopy(model)
    model.prefix="peak1_"
    for i in range(params_length):
        amplitude=params_list[0][i]
        center=params_list[1][i]
        sigma=params_list[2][i]
        if(i==0):

model.set_param_hint('amplitude',value=amplitude,min=0)

model.set_param_hint('center',value=center,min=center*0.95,max=ce
nter*1.05)
            model.set_param_hint('sigma',value=sigma,min=0)
            continue
        new_model = copy.deepcopy(model_temp)
        new_model.prefix=f"peak{i+1}_"



new_model.set_param_hint('amplitude',value=amplitude,min=0)

new_model.set_param_hint('center',value=center,min=center*0.95,ma
x=center*1.05)
        new_model.set_param_hint('sigma',value=sigma,min=0)
        model+=new_model



self.x_smooth=np.linspace(start_end_xsmooth_arr[0],start_end_xsmo
```

```
oth_arr[1],int(start_end_xsmooth_arr[2]))


    params=model.make_params()
    if(params_length==0):
        params=model.guess(self.y,x=self.x)
    result = model.fit(self.y,params,x=self.x)
    self.y_fit = result.eval(x=self.x_smooth)
    self.peak_result=re.sub(r" (?=peak\d+_amplitude)","\n\n
",result.fit_report(show_correl=False,sort_pars=True).split('[[Va
riables]]')[-1])
```

The non shared function for adding the models of the curve fitting:

```
def gaussian(x, amplitude, mean, sigma):
    return amplitude * np.exp(-(x - mean) ** 2 / (2 * sigma **
2))


def gaussian_fit(self):
    # model = Model(gaussian)
    # GaussianModel used function as the function above
    model= GaussianModel()
    __private_shared_fitting_body(self,model)



# def lorentzian(x, amplitude, mean, sigma):
#     return (amplitude * sigma**2) / ((x - mean)**2 + sigma**2)
```

```
def lorentzian_fit(self):
    # Fit Lorentzian distribution
    # model = Model(lorentzian)
    # LorentzianModel used function as the function above
    model=LorentzianModel()
    __private_shared_fitting_body(self,model)



def voigt(x, amplitude, mean, sigma, gamma):
    z = ((x - mean) + 1j*gamma) / (sigma * np.sqrt(2))
    return amplitude * np.real(wofz(z))


def voigt_fit(self):
    # Fit Voigt distribution
    # VoigtModel used function as the function above
    model = VoigtModel()
    __private_shared_fitting_body(self,model)
```

When using

```
from fitting_functions import *
```

the variables and functions that can be imported are below:

```
__all__=["linear_fit","gaussian_fit","lorentzian_fit","voigt_fit"
,"np"]
```