UFR - Sciences et Techniques, Besancon

# Smoothing spectra using least-squares fits

## Fortran implementation - Practical report

M1 CompuPhys 2023/2024 - Léo BECHET

# Introduction

The least squares fit technique dates back to the 1800s, when discovered by Legendre that was trying to compute the orbits of bodies in astronomy. It was later improved upon by Gauss who independently rediscovered it while trying to achieve similar purposes. It has a wide variety of applications, namely machine processing, image processing, and in our case, curve fitting.

Here we will focus on fitting a gaussian curve representing the spectral lines in order to improve the precision of the measure. The idea is to eliminate background noise by fitting as closely as possible a new function that would eliminate background noise.

# Hypothesis

We will first transform or wavenumber to a more usable variable X using the following hypothesis :

$$X = \frac{\omega - \overline{\omega}}{\sqrt{\overline{\omega^2} - \overline{\omega}^2}} = \frac{\omega - \overline{\omega}}{\sigma} \quad (1)$$

Where ω is the wavenumber and a bar signifies the mean of the values. We can also see the apparition of the standard deviation σ. Spectral lines are usually represented using a Lorentz function, thus we can define :

$$F(\omega) = \frac{S}{\pi} \cdot \frac{\gamma}{(\omega - \omega_m)^2 + \gamma^2} \quad (2)$$

Where S is the integral of F, $\omega_m$ is the wavenumber of the maximum, and γ the half width at half height. Let's reverse this function.

$$F^{-1}(X) = \frac{\pi}{S \cdot \gamma} \cdot X^2 + \frac{2\pi\sigma}{S\gamma}(\overline{\omega} - \omega_m)X + \frac{\pi}{S\gamma}\{\gamma^2 + (\overline{\omega} - \omega_m)^2\}^2 \quad (3)$$

$$a_0 = \frac{\pi}{S\gamma}\{\gamma^2 + (\overline{\omega} - \omega_m)^2\}^2 \quad (4)$$

$$a_1 = \frac{2\pi\sigma}{S\gamma}(\overline{\omega} - \omega_m)X \quad (5)$$

$$a_2 = \frac{\pi}{S \cdot \gamma} \cdot X^2 \quad (6)$$

Here we identified part of the equation to coefficients for readability. Upon further computations that we will not develop here for the sake of simplicity (cf. Seminar), we can deduce S, γ and $\omega_m$ using those coefficients as well as the expression for them :

$$S = \frac{\pi\sigma}{\sqrt{a_0 a_2 - \frac{{a_1}^2}{4}}} \quad (7)$$

$$\gamma = \sigma\sqrt{\frac{a_0}{a_2} - \frac{{a_1}^2}{4 \cdot {a_2}^2}} \quad (8)$$

$$\omega_m = \overline{\omega} - \left(\frac{\sigma a_1}{2a_2}\right)^2 \quad (9)$$

Calculations for the a coefficients will be described later in the code.

# Fit integration

This section is dedicated to the construction of the codebase. We do not take into account the weights, i.e. the mean of weights will always be 1 in this case. We will not talk about file handling here and only relevant parts of declared coroutines will be shown. Please refer to the codebase to have access to the code as a whole.

## Defining data and variables

We would like to be able to reuse the code for multiple spectrums and not have hardcoded values that would require recompilation at each step. In order to do so we declare our arrays as allocatable.

```
implicit none
real :: pi = acos(-1.0)
  integer :: i
  integer :: max_elements
  integer, parameter :: start = 2280
  real, parameter :: step = 0.01
  real, allocatable :: data(:)
     ...
  real :: omegam, gamma, S
  !final theoretical curve
  real, allocatable :: theoretical_f(:)
```

Arrays are told to be allocatable in order to be dynamically set using the number of lines, i.e. data points, in the file used.

## Console arguments

We would like to pass arguments from the command line to the program. Luckily for us, the gfortran compiler has a coroutine that can be used to do just that. Here we set the weight type we wish to use, the input file and where to save our computed data, and

```
call get_command_argument(1, weight_type)
call get_command_argument(2, filename)
call get_command_argument(3, filename_output)
```

A coroutine is called to get the number of lines of data in a file, to allocate arrays:

```
call CountLinesInFile(filename, max_elements)
allocate(data(max_elements))
   ...
allocate(theoretical_f(max_elements))
```

## Least-squares fit computations

Once the arrays are allocated we can populate them using our data and the relevant computations..

```
call ReadFileFixed(filename, max_elements, data)
omegas = [(start + real(i - 1) * step, i = 1,max_elements)]
call ComputeStandardDeviation(omegas, max_elements, sigma)
```

```
call ComputeAverage(omegas, max_elements, m_omega)
X = (omegas - m_omega) / sigma
```

Omegas are computed using the given information, starting at 2280, with a step of 0.01 and a number of points equal to the number of data points in the file. We then construct X using the formula specified in the Hypothesis section. In order to compute the weights, we need to check the arguments passed to the program by the command line. Here we already gathered it above and saved it in weights_type. In case the type is “square”, then we use the square of the value as weight, instead of 1.

```
if (weight_type == "square") then
  weights = (data)**2
endif
```

We need to compute averages for the calculations of the a coefficients. Those will be used later for the curve parameters. We will refer to the formula exposed in the seminar to determine which values will be used.

```
call ComputeAverage(weights, max_elements, m_w)
call weighted_average(1/data, weights, max_elements, m_y)! m_y
call weighted_average(X, weights, max_elements, m_x)! m_x
call weighted_average(X**2, weights, max_elements, m_x2)! m_x2
call weighted_average(X**3, weights, max_elements, m_x3)! m_x3
call weighted_average(X**4, weights, max_elements, m_x4)! m_x4
call weighted_average(X*(1/data), weights, max_elements, m_xy)! m_xy
call weighted_average(X**2*(1/data), weights, max_elements, m_x2y)!m_x2y
```

The weighted_average() coroutine is misleading. It computes a variation of the weighted mean, where the denominator is not the sum of the weights but the number of items being weighted and average. One can argue that this is not a weighted average, it did however fit well in the codebase and wasn’t changed for that reason. We will now focus on computing the a coefficients.

```
under = m_w*m_x2*m_x4 + 2*m_x*m_x2*m_x3 - m_x2**3 - m_w*m_x3**2 - m_x**2*m_x4
```

```
a0 = (m_y*m_x2*m_x4 + m_xy*m_x3*m_x2 + m_x*m_x3*m_x2y - m_x2y*m_x2**2 - m_y*m_x3**2 - m_x*m_xy*m_x4)/under
```

```
a1 = (m_w*m_xy*m_x4 + m_x2*m_x3*m_y + m_x*m_x2*m_x2y - m_x2**2*m_xy - m_x*m_x4*m_y - m_w*m_x3*m_x2y  )/under
```

```
a2 = (m_w*m_x2*m_x2y + m_x*m_x3*m_y + m_x*m_x2*m_xy - m_x2**2*m_y - m_x**2*m_x2y - m_w*m_x3*m_xy)/under
```

In order to limit calculations, we've separated the denominator from the formulas to avoid having it computed 3 times. We then used the previously computed means and said denominator to get our coefficients.

```
omegam = m_omega - ((sigma*a1) / (2*a2))
gamma = sigma * sqrt((a0/a2) - (a1**2)/(4*a2**2))
S = (pi*sigma) / sqrt(a0*a2 - (a1**2)/4)
```

Using the formulas defined in the hypothesis we can come back to the three parameters needed for the Lorentz equation. They are computed using the code described above.

## Fitted curve computation and graphs

The least squares fit has now been completed, we wish to plot the function. Sadly there is no graphical display that we know of in Fortran. For that matter we will compute the fitted function in Fortran but plot it using python. The calculation for the function is as follows.

```
theoretical_f = (S/pi) * gamma/((omegas-omegam)**2 + gamma**2)
```

That line saves the values of F(omega) to the theoretical_f variable. That will then be saved in a file to later be imported in python. The plotting part of the code goes as follows.

```
def GetAndPlot(fname, label="", style=""):
    # with open(input("file for plot 1\n > "), "r")as file:
    with open(fname, "r")as file:
        print(len(data))
        # data = [float(i) for i in file.read().split("\n") if i != '']
        data = [float(val) for val in data[:-1]]
        plt.plot([i*graph_step for i in range(len(data))], data , style,
label=label, alpha=0.7 )

GetAndPlot("data/spectre_1.txt", label="experimental", style="bo")
GetAndPlot("output/sp1_1", label="weight: 1", style="r--")
GetAndPlot("output/sp1_sq", label="weight: **2", style="r-")
plt.title("data/spectre_1.txt")
plt.legend()
plt.show()
```

Note that the experimental data is plotted first as to not overlap on the fitted curves. Experimental points are blue disks, fitted weight 1 is red dashed and fitted weight squared is red solid. Also note that no names have been given to the axis. We are not looking to

express any scientific conclusions, but rather to visually confirm the fit. The resulting graph is, for the first spectra, the following.

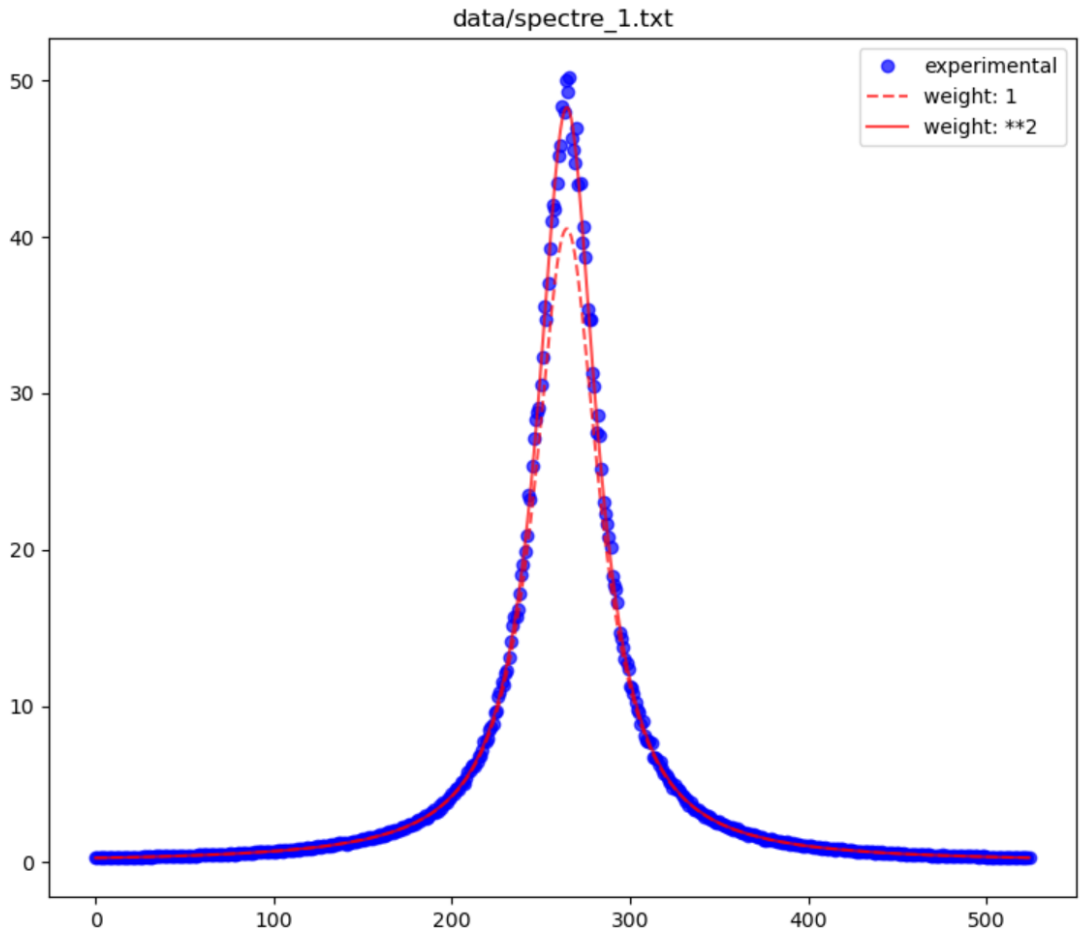


*Fig 1. Plot of the experimental data (blue) with the 1 weight fit (dashed red) and squared weight fit (solid red), shows the correction made using squared weights.*

We can see that the weight squared function does correct the error when using the weights of 1. We can repeat that process for the 4 other spectrums. In reality, a wrapper was made in python, that creates all necessary files, runs the fortran binary with the right parameters, and plots the graphs. We recommend you take a look if you wish to have more information on how the data is processed.

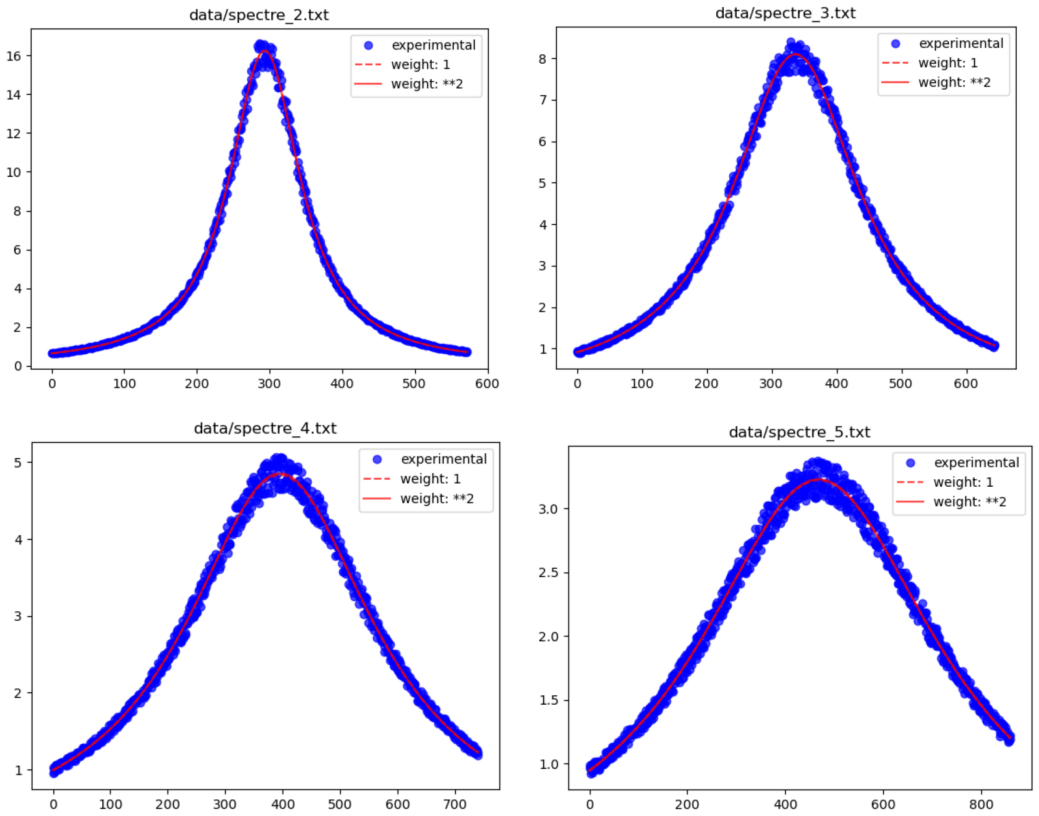


*Fig 2. Plot of data and fits for the other 4 graphs.*

As we can see, the fitted curves do fit pretty well on the other 4 spectrums as well.

## Linear regression

From the command line output of the Fortran program, we get the values for a linear regression on γ and ω. We can build the table below:

| | Weights 1 | | | Weights squared | | |
|---|---|---|---|---|---|---|
| Atm \ var | $\omega_m$ | γ | S | $\omega_m$ | γ | S |
| 1 Atm | 2282.65 | 0.21904 | 27.8919 | 22.8265 | 0.20038 | 30.4026 |
| 3 Atm | 2282.93 | 0.59753 | 30.4650 | 2282.83 | 0.59804 | 30.5968 |
| 6 Atm | 2283.37 | 1.19619 | 30.4236 | 2283.36 | 1.19851 | 30.4579 |
| 10 Atm | 2283.95 | 2.00545 | 30.4506 | 2283.94 | 2.00033 | 30.4984 |
| 15 Atm | 2284.68 | 3.01125 | 30.4684 | 2284.68 | 3.00943 | 30.5077 |

*Fig 3. Table of obtained results for each fits.*

Only the data from the squared weights is interesting as it is the most precise values we can obtain. We will do a linear regression on γ and ω to obtain the zero pressure and the broadening and shift coefficients. Due to the lightweight aspect of the task, we chose to use Python as the computation time difference with fortran will be more than negligible, and it will allow us to graph plots very easily. We implement a very simple regression function.

```
def linear_regression(x, y):
    n = len(x)
    # means of x and y
    m_x = sum(x) / n
    m_y = sum(y) / n
    # slope (m) intercept (b)
    n = sum((xi - m_x) * (yi - m_y) for xi, yi in zip(x, y))
    d = sum((xi - m_x)**2 for xi in x)

    m = n / d
    b = m_y - m * m_x
    return m, b

print(f'linear regression for gamma (slope/intercept):
{linear_regression(P, gammas)}')
print(f'linear regression for omega (slope/intercept):
{linear_regression(P, omegas)}')

Console out :
linear regression for gamma (slope/intercept): (0.2006625396825397,
-0.0032997777777779014)
linear regression for omega (slope/intercept): (0.14825396825395612,
2282.4542222222226)
```

We know that the zero pressure is the intersection between $\omega_m$ and the Y axis, thus we can conclude that the zero pressure is of 2282.454 Pa; we know that the slope of $\omega_m$ is also equal to the shift coefficient, which we can conclude that its value is 0.1482 and finally, we can find 0.201 for γ as it is the slope of the γ regression. In order to visually verify our regression, we plot it. The resulting graphs are as follows.

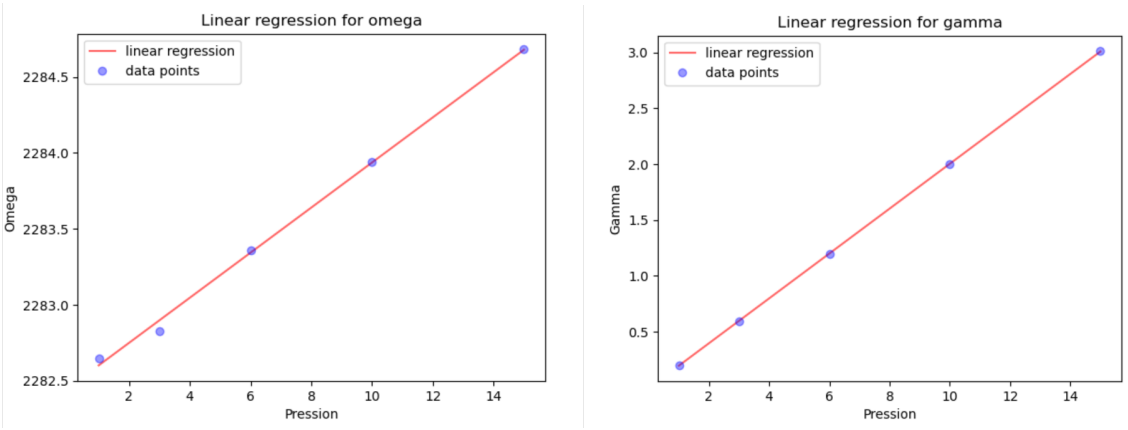


*Fig 4. Linear regression performed on ω (left) and γ (right)*

Both regression seems to fit quite well and are well aligned with the computed data points.

# Conclusion

The least squares fit algorithm was successfully implemented in Fortran, allowing us to get more precise informations on experimental data. Though the main part of the practical was focused on implementing the codebase of the algorithm, it did allow us to do a deep dive on the inner workings of the technique. Not only have our skills in Fortran been put to the test, we also learned valuable resources and discovered new techniques that we can now add to our own personal toolkits for future experimental data exploitation.

Fitting the spectrums of multiple pressures made it possible to compute linear regressions on the data that gave us access to the gas physical properties. Such techniques can be used in astrophysics, analyzing the atmosphere being one of the many examples.