Trabajo Fin de Máster

Máster en Física

Characterization of nanowires for qubit fabrication based on Silicon technology using Raman spectroscopy

Autor: Ginés González Guirado

Tutor/es: Jorge Serrano Gutiérrez, Vanessa Giselle Hinojosa Chasiquiza y Óscar Martínez Sacristán

Abstract

The advancement of quantum computing facilities necessitates the scalable development of reliable qubit platforms, which is currently at the forefront of research efforts by major industrial players such as Google, Amazon, and Intel. One of the most promising strategies to address this challenge involves adapting current silicon nanoelectronics chip technology to meet the requirements for semiconductor qubit fabrication. This adaptation utilizes either electron or hole spin-orbit coupling in semiconductors, along with gates to control, read, and manipulate spin states electrically.

This approach demands state-of-the-art fabrication processes that meticulously control the roughness, strain, doping, and positioning of semiconductor nanowires, two-dimensional semiconductor and oxide layers, and metallic gates.

In this Master's thesis, we report a micro-Raman investigation of silicon nanowires and chips within the framework of a Spanish national research program aimed at developing the first Spanish scalable qubit fabrication platform based on silicon technologies, in collaboration with the Barcelona Microelectronics Institute - CSIC.

To achieve this, hyperspectral maps and spectral profiles were obtained using two different laser wavelengths on both polycrystalline and monocrystalline silicon nanowire layers deposited on a single-crystal silicon substrate covered with thermal oxide. In all cases, the Raman spectrum is dominated by the signal from the silicon substrate, presenting a challenge in identifying the characteristics of the silicon nanowire, which are of interest for this project.

A careful combination of polarization studies and intentional laser heating of the uppermost layer, along with a unique fitting strategy developed in Python during this Master's thesis, provided a method to characterize the main features of the nanowire layer with submicron resolution. This method represents the primary result of the current thesis and will be applied in the future to analyze further steps in the quest for the fabrication of scalable semiconductor qubit platforms.

Resumen

El avance de las instalaciones de computación cuántica requiere el desarrollo escalable de plataformas de qubits fiables, que actualmente está en la vanguardia de los esfuerzos de investigación de los principales actores industriales como Google, Amazon e Intel. Una de las estrategias más prometedoras para afrontar este reto consiste en adaptar la actual tecnología de chips nanoelectrónicos de silicio para cumplir los requisitos de fabricación de qubits semiconductores. Esta adaptación utiliza el acoplamiento espín-órbita de electrones o huecos en semiconductores, junto con puertas para controlar, leer y manipular eléctricamente los estados de espín. Este enfoque exige rigurosos procesos de fabricación de última generación que controlen meticulosamente la rugosidad, la deformación, el dopaje y la posición de nanohilos semiconductores, las capas bidimentsionales semiconductoras y de diversos óxidos y las puertas metálicas.

En este trabajo de fin de máster, presentamos una investigación micro-Raman de nanohilos y chips de silicio en el marco de un programa nacional español de investigación destinado a desarrollar la primera plataforma española de fabricación escalable de qubits basada en tecnologías de silicio, en colaboración con el Instituto de Microelectrónica de Barcelona - CSIC.

Para ello, se obtuvieron mapas hiperespectrales y perfiles espectrales utilizando dos longitudes de onda láser diferentes sobre capas de nanohilos de silicio policristalino y monocristalino depositadas sobre un sustrato de silicio monocristalino recubierto de óxido térmico. En todos los casos, el espectro Raman está dominado por la señal procedente del sustrato de silicio, lo que supone un desafío a la hora de identificar las características del nanohilo de silicio, que son de interés para este proyecto.

Una cuidadosa combinación de estudios de polarización y calentamiento láser intencionado de la capa superior, junto con una estrategia de ajuste singular desarrollada en Python durante esta tesis de máster, proporcionó un método para obtener las principales características de la capa de nanohilos con una resolución submicrométrica. Este método representa el principal resultado de la presente tesis y se aplicará en el futuro para analizar nuevos pasos en la búsqueda de la fabricación de plataformas de qubits semiconductores escalables.

Contents

1 Introduction **1**

2 Methodology **2**
2.1 Raman spectroscopy 2
2.2 Sample description and preparation 9
2.2.1 Polycristalline Si samples 9
2.2.2 Monocrystalline Si samples 10
2.3 Experimental method 11
2.4 Data treatment 12

3 Experimental results and discussion **15**
3.1 Polycristalline Si samples 15
3.2 Monocrystalline Si samples 21
3.2.1 Analysis using a red laser: Hyperspectral maps 21
3.2.2 Analysis using a green laser: Spectral profiles (linescans) 23
3.2.3 Analysis using a green laser: Hyperspectral maps 30
3.2.4 Influence of the green laser polarization on edge spectral profiles 35

4 Conclusion and further work **37**

Bibliography **39**

Appendix **42**
Polycrystalline Si sample maps with red laser 42
Python Code for Linescans 42
Python Code for Maps 87

1 Introduction

The field of quantum computing has been extensively studied due to its potential applications, since the initial proposals by Feynman [1] and others in the 1980s [2]. A universal quantum computer would enable much faster resolution of certain types of calculations, such as integer factorization and quantum simulation, compared to classical computers [3]. In recent years, devices capable of performing specific calculations, which are difficult or even impossible for classical computers, have been developed to achieve this goal. However, manufacturing these devices on a large scale presents significant challenges, such as the production of qubits at near-room temperatures and the use of materials and fabrication processes that are easy to industrialize. Therefore, one of the current research targets is to achieve the mass production of qubits in a systematic, cost-effective manner while minimizing defects that may arise during the manufacturing process.

Various proposals have been developed to create qubits. Some of the proposed and implemented approaches include superconducting qubits [4], trapped ion qubits [5], quantum dots and spin-based qubits in semiconductors [6], qubits of neutral atoms [7], defects or color centers in semiconductors [8], qubits based on nuclear magnetic resonance [9] and photonic qubits [10].

The idea of using nanowires to manufacture qubits starts from the pioneering work of Nadj-Perge et al. [11], showing that through spin-orbit interaction the spins can be controlled even at the level of individual electrons. In that work, a spin-orbit qubit implemented on an InAs nanowire is presented, where the spin-orbit interaction is so strong that spin and motion can no longer be separated. So, in that situation, they performed fast qubit rotations and universal control of single qubits using only electric fields. Additionally, they improved coherence by dynamically decoupling the qubit from the environment. Furthermore, the qubits were individually addressable, as they were housed in single-electron quantum dots. An advantage of nanowires over the rest of the options for qubit fabrication, from the point of view of scalability, is that they allow multiple metallic contacts or local superconductors and electrostatic gates on the top, bottom and side of the wire [12].

Since silicon is the fundamental semiconductor in modern electronics, producing Si nanowires is easier compared to other materials, as techniques for working with and doping silicon are well-established. Therefore, the large-scale implementation of these qubits would not require significant efforts in terms of the equipment needed for their production. Given the rigor and precision required to manufacture semiconductor qubits with proper functionality, it is essential to employ characterization techniques capable of detecting submicron structural defects. Such techniques are crucial for selecting the optimal procedures for qubit fabrication. Consequently, micro-Raman spectroscopy emerges as an ideal technique for this purpose.

The use of Raman spectroscopy for studying nanowires is preferred due to its non-destructive nature and its ability to detect defects, disorder, and stress, either compression or elongation, in the structure. These defects can be precisely identified by fitting different types of functions to the experimentally obtained peaks, depending on the material's doping. If the material is highly doped, the Raman spectrum is best fitted with a Fano profile. However, the most common function used for Raman peak analysis is a Voigt profile, as it accounts for both the natural width of a Raman peak (Lorentzian profile) and the broadening caused by the measuring instruments (Gaussian profile). By observing variations in the peak positions and their full-width-at-half-maximum (FWHM), we can identify the defects produced in the devices during the manufacturing process. This allows us to determine the best procedures for fabricating the

nanowires and the contact shapes that generate the least stress, which is crucial for the device's proper functioning and durability. To fit the experimental Raman peaks in this work, a Python code was developed.

In this work, the characterization of Silicon (Si) nanowires for qubit fabrication has been carried out using Raman spectroscopy. Crystalline Si and polycrystalline Si nanowires have been studied, as well as different lengths, widths, production methods, and different contact shapes, to observe the differences in defects that are generated in the devices when manufactured in one way or another.

2 Methodology

2.1 Raman spectroscopy

Raman scattering can be described from classical electromagnetism, according to Long (2002) [13]. Simplifying and without losing generality, we can consider that the beam irradiating the sample is a linearly polarized monochromatic plane wave:

$$\vec{E} = \vec{E}_0 e^{i(\vec{k}_i \cdot \vec{x} - \omega_i t)} \tag{2.1}$$

Where $\vec{E}_0$ represents the amplitude of the incident electric field, $\vec{k}_i$ denotes the wave vector and ω_i is the angular frequency of the incident radiation. When a dielectric material like silicon is illuminated, the incident electric field induces a dipole moment within it:

$$\vec{P} = \boldsymbol{\chi} \cdot \vec{E} \tag{2.2}$$

Here, $\boldsymbol{\chi}$ represents the material's susceptibility tensor. The susceptibility $\boldsymbol{\chi}$ is modulated by the fluctuations of the atoms around their equilibrium positions in the lattice. These fluctuations are characterized by the lattice vibrations and, therefore, they are related to the phonons of the lattice. Phonons can be described in their normal coordinates through the Bloch function as [14]:

$$Q_j = Q_{j0} e^{\pm(\vec{q}_j \cdot \vec{r} - \omega_j t)} \tag{2.3}$$

Where $\vec{q}_j$ and ω_j are the wave vector and angular frequency of the normal mode j, respectively. Then, the material susceptibility can be expanded in Taylor series with respect to the normal coordinates of the phonons around their equilibrium position $\vec{\chi_0}$ as:

$$\boldsymbol{\chi} = \boldsymbol{\chi}_0 + \sum_j \left(\frac{\partial \boldsymbol{\chi}}{\partial Q_j} \right)_0 Q_j + \frac{1}{2} \sum_{j,i} \left(\frac{\partial^2 \boldsymbol{\chi}}{\partial Q_j \partial Q_i} \right)_0 Q_j Q_i + ... \tag{2.4}$$

Where the subscript 0 denotes the value of the magnitude in the equilibrium configuration. For small atomic vibrations around their equilibrium positions, and thus within the harmonic approximation, the susceptibility can be expressed as:

$$\boldsymbol{\chi} = \boldsymbol{\chi}_0 + \sum_j \left(\frac{\partial \boldsymbol{\chi}}{\partial Q_j} \right)_0 Q_j \tag{2.5}$$

The normal coordinate Q_j, whose expression is given by the equation (2.3), can be simplified,

under the harmonic electrical approximation, as:

$$Q_j = Q_{j0} \cos(\omega_j t) \tag{2.6}$$

Thus, with these approximations, the polarization induced in the irradiated sample, which is described in the equation (2.2), can be approximated as:

$$\vec{P} = \boldsymbol{\chi}_0 \cdot \vec{E}_0 \cos(\omega_i t) + \sum_j \left(\frac{\partial \boldsymbol{\chi}}{\partial Q_j} \right)_0 Q_{j0} \vec{E}_0 \cos(\omega_j t) \cos(\omega_i t) \tag{2.7}$$

From this expression, using a trigonometric relationship for the cosines according to [13], we obtain:

$$\vec{P} = \underbrace{\boldsymbol{\chi}_0 \cdot \vec{E}_0 \cos(\omega_i t)}_{\text{Rayleigh scattering}} + \underbrace{\sum_j \left(\frac{\partial \boldsymbol{\chi}}{\partial Q_j} \right)_0 Q_{j0} \vec{E}_0 \frac{1}{2} [\cos((\omega_i - \omega_j)t) + \cos((\omega_i + \omega_j)t)]}_{\text{Raman scattering}} \tag{2.8}$$

In this equation, three distinct frequencies contributing to the polarization of the sample are clearly identified. Assuming that the illuminated sample emits energy as a dipole, the light emitted by the sample is primarily characterized by these frequencies. The first term, characterized by a frequency ω_i corresponds to elastic scattering or Rayleigh scattering, where the scattered light has the same frequency as the incident light. The second term encompasses inelastic events or Raman scattering, characterized by frequencies $\omega_i - \omega_j$ and $\omega_i + \omega_j$, contributing to Raman Stokes and Raman anti-Stokes scattering, respectively of phonons of $\hbar\omega_j$ energy, i.e., corresponds to the j-th mode. Typically, Raman spectra are represented by the *Raman shift* on the x-axis, which denotes the frequency differences $\omega_i \pm \omega_j$, and usually expressed in units of cm^{-1}, as can be seen in Fig. 1.

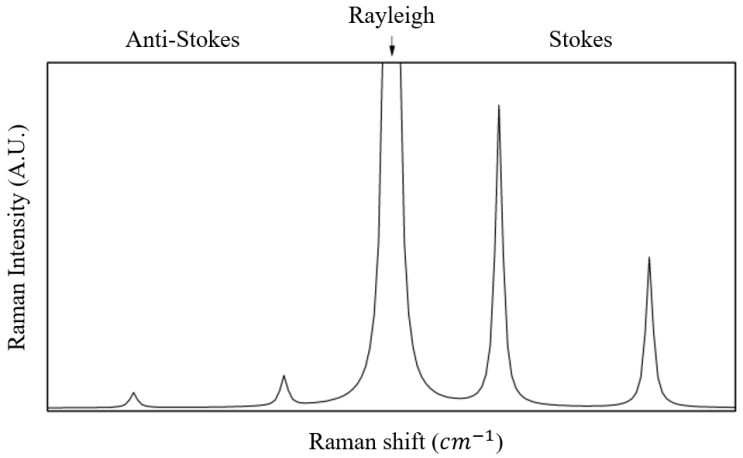

Figure 1: Example of a Raman spectrum, illustrating the Rayleigh radiation, Raman Stokes, and Raman Anti-Stokes radiations for two Raman modes, along with their varying relative intensities. Illustration adapted from [15].

Drawing upon the formalism established by M. Cardona et al. in [16], the intensities of Stokes and anti-Stokes emission from a sample illuminated by an electromagnetic field can be derived from a classical perspective, where the energy radiated per unit time by the sample is expressed

as:

$$\frac{dI}{d\Omega} = \frac{\omega^4}{(4\pi)^2 \epsilon_0 c^3} |\hat{e}_s \cdot \vec{P}|^2 \tag{2.9}$$

Where I is the intensity irradiated by the sample, Ω the element of solid angle, ϵ_0 the permittivity of the medium in free propagation, c the speed of light in the medium in free propagation, ω the frequency of the dipole moment $\vec{P}$ induced in the material and $\hat{e}_s$ the unit vector that represents the polarization of the light scattered by the system measured at the observation point.

In this classical treatment, the medium has dimensions larger than the wavelength of the incident radiation, and the induced dipole moment is determined by the incident electric field through $\vec{P} = \epsilon_0 \boldsymbol{\chi} \hat{e}_i E_i$ [14]. Because the wavelength of the incident light is much larger for nanowires (NWs), the intensity radiated by bulk material under illumination is not valid for NW structures [17]. In this approximation for the bulk, it is assumed that the electric field inside the material, which causes the dipole moment in the material, is equal to that of the incident radiation. However, due to differences in the dielectric constants of the medium, substrate, and NW, and because of the dimensions of the NW relative to the wavelength of the incident electromagnetic field, the electric field inside the NW can differ significantly from the incident electric field.

When nanowires are illuminated with a laser beam, the incident electromagnetic field induces polarization in the molecular bonds of the NWs. This polarization can be separated into two components as in equation (2.7), that is, a term that oscillates with the incident electromagnetic field (Rayleigh scattering) and a term of polarization induced by the displacements of the atoms in the lattice around their equilibrium positions. The latter term includes Raman Stokes and anti-Stokes scattering. The intensity of the photons radiated from this Raman scattering is determined by the expression [14]:

$$I_R = \left(\frac{\omega_i^2 \mu_0 |E|}{4\pi d} \right)^2 \langle |\hat{e}_i \cdot \mathbf{R} \cdot \hat{e}_s|^2 \rangle \tag{2.10}$$

Where ω_i is the frequency of the incident laser, μ_0 the magnetic permeability of the NW, $|E|$ the modulus of the electric field inside the NW, d is the distance between the detector and the sample, $\hat{e}_i$ is the vector of the incident polarization, $\hat{e}_s$ is the vector of the scattered polarization and, finally, $\mathbf{R}$ is the characteristic Raman tensor of the material. The Raman tensor depends on the derivatives of the polarizability tensor ($\boldsymbol{\alpha}$) with respect to the normal coordinates of the atoms in the semiconductor, as follows [13]:

$$\mathbf{R} = \left(\frac{\partial \boldsymbol{\alpha}}{\partial Q_j} \right)_0 Q_j \tag{2.11}$$

When analyzing the expression 2.10, it is noticeable that the signal coming from the NWs is strongly influenced by the distribution of the electric field inside them when the NWs are illuminated with a laser beam.

The cross section for Raman events is independent of the electric field inside the sample, so the expression for the bulk is also valid for NWs, so that the differential cross section can be calculated, according to [16], as:

$$\frac{d\sigma}{d\Omega} = \frac{\omega^4 V^2}{(4\pi)^2 c^4} |\hat{e}_s \cdot \boldsymbol{\chi} \cdot \hat{e}_i|^2 \tag{2.12}$$

Where $\boldsymbol{\chi}$ is the susceptibility independent of the volume of the material illuminated by the

incident beam or scattering volume V. By introducing the approximation given in equation (2.5) for $\boldsymbol{\chi}$, and considering the phonon population distributions, the cross sections for Raman Stokes and anti-Stokes scattering can be obtained, as described in [16], as follows:

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{Stokes}} = \frac{\hbar}{2\omega_j}\left(\frac{1}{e^{\frac{\hbar\omega_j}{k_B T}}-1}+1\right)\frac{(\omega_i-\omega_j)^4 V^2}{(4\pi)^2 c^4}|\hat{e}_s\cdot\frac{d\boldsymbol{\chi}}{dQ_j}\cdot\hat{e}_i|^2 \tag{2.13}$$

$$\left(\frac{d\sigma}{d\Omega}\right)_{\text{anti-Stokes}} = \left(\frac{\hbar}{2\omega_j}\frac{1}{e^{\frac{\hbar\omega_j}{k_B T}}-1}\right)\frac{(\omega_i+\omega_j)^4 V^2}{(4\pi)^2 c^4}|\hat{e}_s\cdot\frac{d\boldsymbol{\chi}}{dQ_j}\cdot\hat{e}_i|^2 \tag{2.14}$$

From these two equations, the relationship between the cross-sections for Stokes and anti-Stokes scattering can be established, which is equivalent to the ratio of intensities between them:

$$\frac{I_{\text{anti-Stokes}}}{I_{\text{Stokes}}} = \left(\frac{\omega_i+\omega_j}{\omega_i-\omega_j}\right)^4 e^{-\frac{\hbar\omega_j}{k_B T}} \approx e^{-\frac{\hbar\omega_j}{k_B T}} \tag{2.15}$$

Applying the approximation since $\omega_i \gg \omega_j$. This relationship shows that at room temperature or lower, the Raman Stokes intensity significantly exceeds the anti-Stokes Raman intensity. Hence, it is typically preferable to use Stokes radiation for sample characterization. Furthermore, this relationship allows a direct determination of the sample temperature, making it widely utilized for in situ thermography of devices during operation [15]. However, conducting such measurements requires equipment capable of effectively filtering out Rayleigh scattering and covering a sufficiently broad frequency range to capture both peaks, only available for a limited number of high-resolution spectrometers.

Raman Tensor and selection rules

The Raman Tensor, defined in equation (2.11), determines which vibrational modes are active in the material in Raman spectroscopy. In 1964, Loudon [18] derived the Raman tensors for each of the 32 point symmetry groups, so that three different Raman tensors were obtained for silicon:

$$\left(\frac{d\boldsymbol{\chi}}{dQ_j}\right)_x = \begin{pmatrix}0&0&0\\0&0&d\\0&d&0\end{pmatrix} \quad ; \quad \left(\frac{d\boldsymbol{\chi}}{dQ_j}\right)_y = \begin{pmatrix}0&0&d\\0&0&0\\d&0&0\end{pmatrix} \quad ; \quad \left(\frac{d\boldsymbol{\chi}}{dQ_j}\right)_z = \begin{pmatrix}0&d&0\\d&0&0\\0&0&0\end{pmatrix} \tag{2.16}$$

In the absence of stresses, the three vibration modes active in silicon are degenerate, so their Raman shifts are the same, specifically $\omega_j = 520.7\ \text{cm}^{-1}$ for crystalline silicon at room temperature and ambient pressure. This value is routinely used for calibration of Raman spectra.

Furthermore, as can be seen in equation (2.13), the Raman tensor is multiplied by the unit vectors representing the polarization of the incident and scattered light. Thus, in silicon samples with surfaces oriented in a crystalline direction, the combination of these vectors with the corresponding Raman tensors leads to a modulation of the Raman intensity emitted by the sample as a function of the polarization of the incident light [19].

Quantum description

There are certain properties of Raman scattering that need quantum treatment to be explained, such as the resonant Raman scattering that occurs in NWs.

In each Raman scattering event, according to [20] three particles participate: an incident photon with frequency ω_i, a phonon with frequency ω_j and a photon with frequency ω_s. First, the material, in an initial state $|i\rangle$, is excited by the incident photon. Then, the material passes to an intermediate state $|a\rangle$, when an electron-hole pair is created. This process can be described with the electron-radiation interaction Hamiltonian H_{e-R}. From the intermediate state, the material relaxes by emitting (Stokes) or absorbing (anti-Stokes) a phonon of frequency ω_j, passing to another intermediate state $|b\rangle$. This process is described with the electron-phonon interaction Hamiltonian (H_{e-ph}), defined in [20]. Finally, the material relaxes from the second intermediate state, emitting a photon of frequency ω_s, through radiative recombination.

Figure 2 illustrates the transitions that give rise to Rayleigh, Stokes Raman and anti-Stokes Raman radiation.

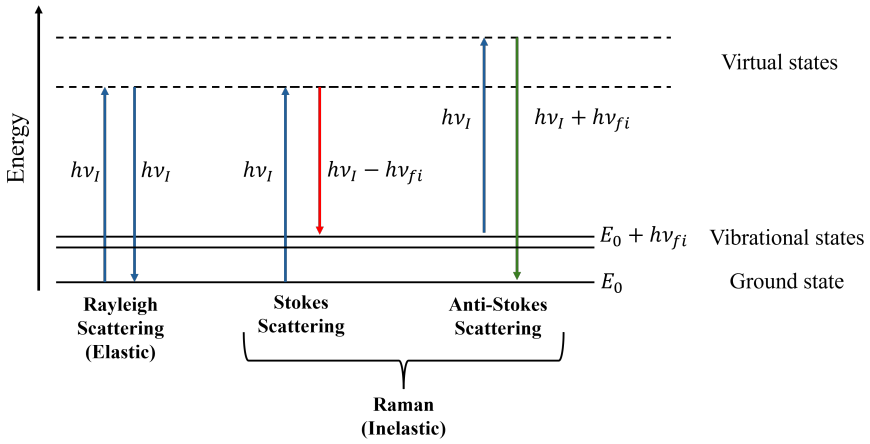

Figure 2: Energy transition diagram for Rayleigh and Raman scattering.

As can be seen in Figure 2, in the complete process there is conservation of energy, that is:

$$\hbar\omega_s = \hbar\omega_i \pm \hbar\omega_j \tag{2.17}$$

Moreover, in first-order Raman scattering the total momentum is conserved:

$$\hbar\vec{k}_s = \hbar\vec{k}_i \pm \hbar\vec{q}_j \tag{2.18}$$

Where $\vec{k}_s$ is the wavevector of the scattered photon, $\vec{k}_i$ the wavevector of the incident radiation and $\vec{q}_j$ the wavevector of the phonon. The minus and plus signs are for Stokes and anti-Stokes radiation, as described previously. The maximum value of $\vec{q}_j$ is reached in the limit in which the wavevector of the Raman photon is equal in magnitude and opposite in direction to the wavevector of the incident photon (backscattering), so, $\vec{q}_j < \frac{4\pi}{\lambda_i}$ according to [17]. Therefore, for $\lambda > 300$ nm, the value of $\vec{q}_j$ is significantly smaller than the width of the Brillouin zone, with a maximum width of π/a being a the lattice parameter. As a result, $\vec{q}_j$ is restricted to the central region of the first Brillouin zone and the selection rule for Raman-active phonons can be established as:

$$\vec{q}_j \simeq 0 \tag{2.19}$$

Finally, quantum mechanics offers an alternative method to compute the probability of Raman scattering, employing Fermi's golden rule and third-order perturbation theory. The probability

of a Raman scattering event is determined by the following equation [20]:

$$P(\omega_j) = \frac{2\pi}{\hbar} \left| \sum_{a,b} \frac{\langle i|H_{e-R}|b\rangle\langle b|H_{e-ph}|a\rangle\langle a|H_{e-R}|i\rangle}{[\hbar\omega_i - (E_a - E_i)][\hbar\omega_i - \hbar\omega_j(E_b - E_i)]} \right|^2 \times \delta(\hbar\omega_i - \hbar\omega_j - \hbar\omega_s) \tag{2.20}$$

In this equation the states that appear are those that have been previously explained. From this expression, it can be deduced that if the frequency of the incident photons is similar to the transition energy of the electronic states of the material, characterized by $(E_a - E_i)$ and $(E_b - E_i)$, the probability of these Raman scattering events dramatically increases, since the denominator of (2.20) tends to zero. This situation is known as resonant Raman scattering [20].

Shape of the peaks in a Raman spectrum

In an ideal scenario, the excitation-relaxation process can be described as:

$$\int_{-\infty}^{\infty} \delta(\omega - \omega_0) d\omega = \omega_0 \tag{2.21}$$

This scenario implies infinite phonon lifetimes with well-defined and unique frequency. However, various factors disrupt the ideal monochromatic behavior, influencing the actual form of peaks in a Raman/Photoluminiscence (PL) spectrum [21]:

- Quantum-mechanical uncertainty in energy levels (phonon/vibration lifetime), which determines the natural width of a Raman/PL peak.
- Collisional broadening, where phonon scattering events reduce the lifetime and broadens the width of the peak.
- Inhomogeneous broadening, disorder and Doppler/Thermal broadening.
- Instrumental Broadening.

As $\Delta\epsilon\Delta t \sim \hbar$, indicating a fundamental limitation imposed by quantum mechanics, the relaxation of excited states inherently possesses a finite lifetime, precluding instantaneous transitions. Consequently, the relaxation process can be formally characterized as a first-order phenomenon, according to Demtröder [21]:

$$\frac{\partial \epsilon_n}{\partial t} = -\frac{\epsilon_n}{\tau_n} \tag{2.22}$$

The lifetime (τ_n) will determine the spread of energies at which the photons will be emitted in the relaxation process.

When we translate this into the frequency domain using Fourier transform, the outcome is:

$$\int_0^{\infty} e^{-t/\tau_n} e^{-i\omega t} dt = \frac{1/\tau_n}{(1/\tau_n)^2 + \omega^2} + \frac{i\omega}{(1/\tau_n)^2 + \omega^2} \tag{2.23}$$

which real part exhibits the generalized Lorentzian line shape:

$$I = \frac{I_0}{\pi\gamma} \left(\frac{\gamma^2}{\gamma^2 + (\omega - \omega_0)^2} \right) \tag{2.24}$$

with

$$\gamma = \frac{\text{FWHM}}{2} \qquad \Longleftrightarrow \qquad \tau_n = \frac{2}{\text{FWHM}} \tag{2.25}$$

Phonon scattering events reduce the lifetime, consequently leading to increased uncertainty in the frequency of emitted photons during the relaxation process. Despite the reduction in lifetime, the overall shape of the observed line remains Lorentzian; however, it results in broader peaks. So, wider Lorentzian peaks indicates more scattering [21]:

$$\text{FWHM} = \frac{2}{\tau_{\text{collision}}} \tag{2.26}$$

with $\tau_{\text{collision}} < \tau_n$.

Other factors influencing peak shape include temperature and pressure. Higher temperatures yield more phonons and collisions, resulting in broader peaks [22]. Similarly, increased pressure leads to more collisions, also contributing to broader peaks.

The compression and stretching of a sample affects the Raman shift as well [23]. When a sample is compressed, its Raman frequency increases, whereas stretching leads to a decrease in its Raman frequency. This can be understood through the spring model analogy applied to atomic interactions [24], where the vibrational frequency of atoms, denoted by ω, is proportional to the spring constant k according to the formula $\omega = \sqrt{\frac{k}{m}}$. A compressed material results in closer atomic interactions, leading to a higher spring constant k and consequently a higher frequency ω. Conversely, stretching the material results in further apart atoms and a lower spring constant, thus yielding a lower frequency.

Inhomogeneous broadening arises when the relaxation process extends across numerous vibrational levels. This can be caused by disorder in the semiconductor lattice, e.g., due to defects, as well as by thermal fluctuations in the atoms, leading to a Doppler broadening effect, which we are not currently focusing on. So, this broadening leads to a randomly distributed perturbation of the phonon lifetime, resulting in a Gaussian dispersion of frequencies for the photons emitted during the relaxation process:

$$G(\omega) = \frac{1}{\sqrt{\pi}\Gamma} e^{-(\omega-\omega_0)^2/\Gamma^2} \tag{2.27}$$

Hence, what we observe is the combination of the Lorentzian shape outlined earlier with the Gaussian spread caused by this randomly distributed effect. This corresponds to a convolution of both functions in the frequency domain, that is referred to as a Voigt profile [21]:

$$I_D(\omega) \propto \int G(\omega'; \sigma) L((\omega - \omega"); \gamma) d\omega' \tag{2.28}$$

Instruments used to measure spectral lines, such as spectrometers and Charge-Coupled Devices (CCDs), significantly influence the shape of Raman/PL lines. Spectrometers utilize gratings with angular dispersion proportional to the groove spacing, where gratings with more lines/mm provide better accuracy in FWHM but less lateral angular dispersion. Diffraction at apertures also contributes to broadening. CCDs are affected by the point spread function, which dictates how accurately frequencies map to specific pixels, causing photon leakage to neighboring pixels and broadening the peak signal. These effects result in inhomogeneous broadening of the Raman peaks, introducing also Gaussian perturbations. Consequently, the final line shape is often modeled as a single Voigt profile, representing the convolution of the initial Voigt profile (Lorentzian profile convoluted with inhomogeneous broadening) with the Gaussian broadening from experimental dispersion and is expressed as shown in Equation (2.28).

2.2 Sample description and preparation

The fabrication of the samples was conducted by our partners in a clean-room facility located at the Barcelona Institute of Microelectronics (IMB, CSIC), through a lengthy process involving the sequential addition and removal of layers using various techniques, such as photolithography, electron beam litography and Atomic Layer Deposition (ALD). This process was designed to achieve the desired final result. Unlike the vertical growth method used in [25, 26], the manufacturing process for these silicon nanowires (SiNWs) with contacts is similar to that of an integrated circuit.

Initially, a silicon ingot was grown. If a monocrystalline structure is desired, the Czochralski (CZ) technique [27] is employed. The ingot was then sliced into 1 mm thick wafers, which were mechanically polished to remove roughness and damage caused by the slicing blade [28]. From this point, doping was performed, and the specific manufacturing process for the studied samples begins.

In this work, two different samples have been studied. In one of them the NWs were made of polycrystalline Si, while in the other they were made of crystalline Si. Also, the NWs can be tailored with specific widths and lengths. The composition of each of the samples is detailed below.

2.2.1 Polycristalline Si samples

In this case, the fabrication process starts with a silicon wafer oriented with its surface perpendicular to the $\langle 001 \rangle$ crystalline direction, doped with boron to achieve resistivities of 0.1-1.4 Ωcm. Then, a silicon oxide layer of about 400 nm was added. Next, 32 nm of polycrystalline silicon was deposited on top. Conventional optical lithography seems to be unsuitable for defining NWs due to its inability to create patterns with nanometric resolution, due to the diffraction-limited optical resolution. Therefore, Electron Beam Lithography (EBL) was used instead. EBL, with its nanometer-scale precision [29], enables the fabrication of complex designs, thereby facilitating the creation of nanostructures [30].

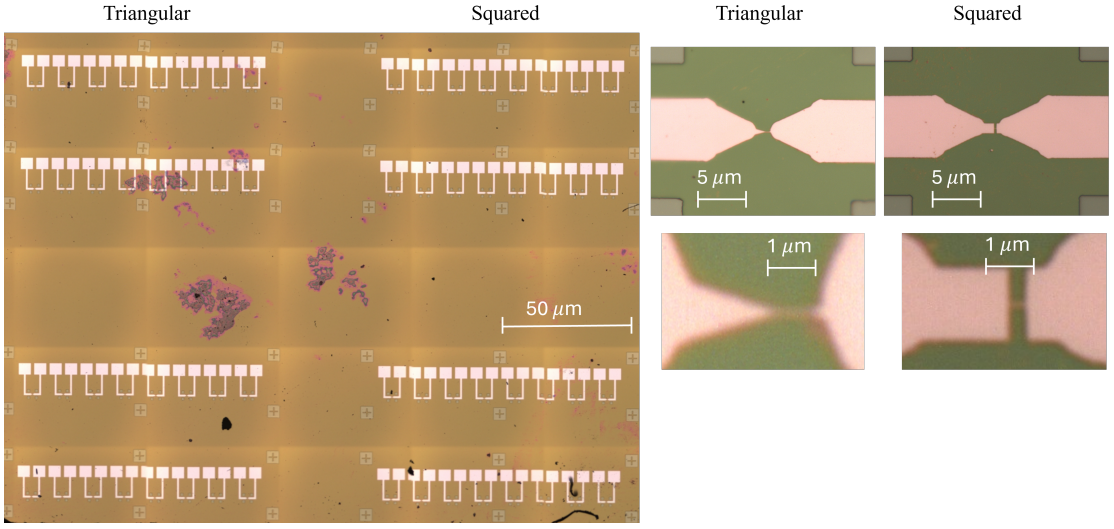

Figure 3: Polycrystalline Si sample with NWs featuring triangular contacts on the left side and square contacts on the right side. On the right side of the image a triangular contact and a square contact with the 100X objective are shown and, in addition, they have been zoomed in in the NW area.

After defining the nanowires, a 44 nm layer of SiO_2 was deposited using Plasma Enhanced Chemical Vapour Deposition (PECVD), and the entire structure was implanted with boron at a dose of $2 \cdot 10^{14}$ atoms/cm^2. The PECVD oxide was removed, so it should not be present in the sample, although some may have remained. Additionally, a layer of HfO_2 was added by using ALD and then removed. At the end of the process, chips of 250×300 µm^2 with NWs with different widths, lengths and contact shape (triangular on the left side and rectangular on the right side) were obtained.

Figure 3 shows a 5X microscopy image of the chip that has been studied. A 100X image of a NW with a triangular contact and another one with a squared contact and a zoom of both images on the NW are also presented. The NWs with triangular contacts are arranged on the left side of the chip, while those with square contacts are located on the right side. The width of the NWs varies by row, while the length varies by columns.

In this work, we focus on NWs with triangular contact, since the square ones were discarded in the fabrication process.

2.2.2 Monocrystalline Si samples

In this sample, the substrate is monocrystalline silicon oriented with its surface perpendicular to the ⟨001⟩ crystalline direction, with a thickness of 725 μm and boron doping. Resistivity ranges from 8 to 12 Ωcm. A 400 nm layer of thermal SiO_2 is then applied, followed by an upper layer of 50 nm monocrystalline silicon, also oriented along the ⟨001⟩ crystalline direction with similar resistivity values. The thickness of this layer may vary slightly at different points on the wafer.

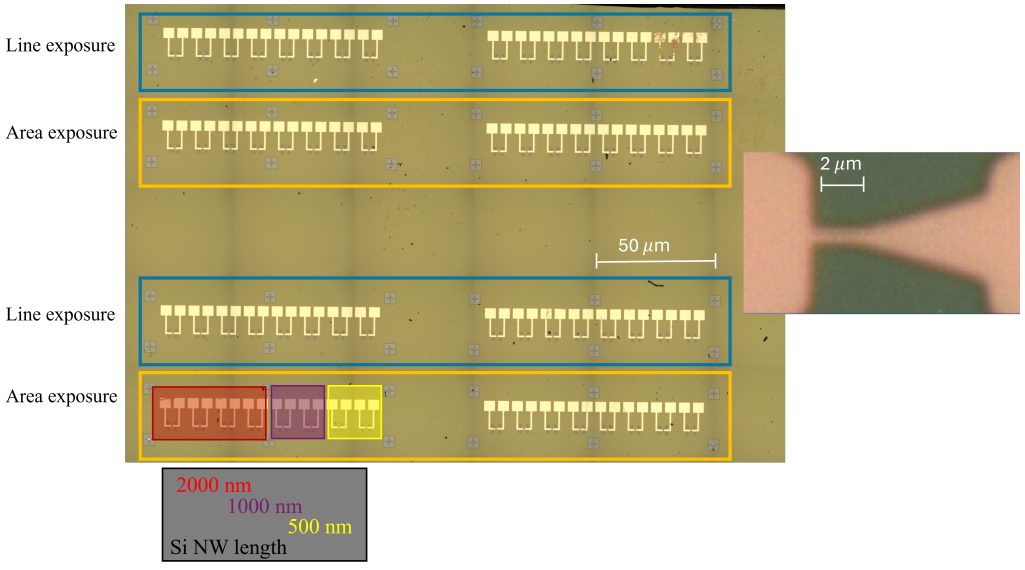

Figure 4: Sample of monocrystalline Si oriented along the ⟨001⟩ crystalline direction with NWs produced using line exposure in odd rows and area exposure in even rows. The NWs length varies by column: The NWs in the first four columns of each block are 2000 nm long, those in columns 5 and 6 are 1000 nm long, and those in columns 7 and 8 are 500 nm long. On the right side of the image a NW with the 100X objective is shown.

To achieve long narrow shapes, such as NWs, there are several methods of exposition available, with line exposure and area exposure being two key approaches in EBL. These methods differ in how the electron beam traces the pattern. In line exposure, the beam is positioned at one end

of the desired NW and gradually moved to the other end in a single continuous pass. In area exposure, the NW is treated as a rectangle, and the beam draws several parallel lines across it, which may not be parallel to the NW's main axis. These lines begin and end at different points along the structure, effectively covering the entire area.

Figure 4 shows the sample, where the odd rows contain NWs created using line exposure, and the even rows contain NWs made using area exposure. Additionally, the length of the NWs varies by column. Dividing the sample into right and left blocks, the NWs in the first four columns of each block are 2000 nm long, those in columns 5 and 6 are 1000 nm long, and those in columns 7 and 8 are 500 nm long. It is also presented a optical image of a NW with the 100X objective. In this sample all the contacts are triangular.

2.3 Experimental method

The measurements were conducted using a HORIBA Soleil System. Two lasers, emitting at wavelengths of 532 nm and 638 nm respectively, served as the light sources. The emitted laser light is linearly polarized. As previously mentioned, the polarization of light affects the measurement by interacting with the Raman tensor [19]. Additionally, a typical Raman setup includes a microscope, a filter to eliminate Rayleigh radiation, and a spectrometer coupled with a CCD. In Figure 5, the HORIBA Soleil instrument is presented, while in Figure 6, we provide a schematic representation of the HORIBA Soleil components along with the optical path traversed by the laser-produced light and the scattered light from the sample.

Figure 5: HORIBA Soleil coupled with a SuperHead.

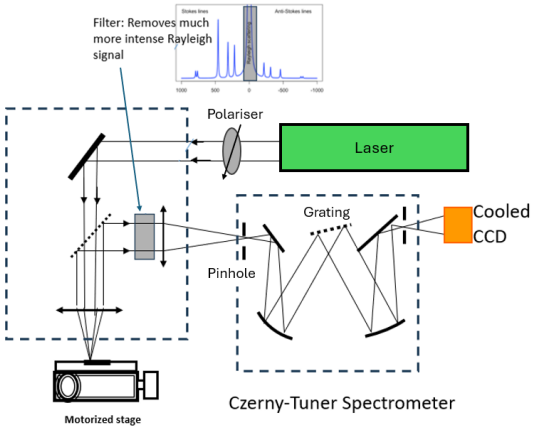

Figure 6: Scheme of the HORIBA Soleil components.

The microscope is employed to focus the laser light on the sample while simultaneously collecting the Raman emission from the sample. A 100X objective with a numerical aperture (NA) of 0.95 was employed for this purpose. This objective enables visualization of the sample and precise selection of the laser focus area using a camera. The laser light is focused on an illumination area of approximately 1 μm diameter in size for both lasers, in accordance with the diffraction resolution-limit $\frac{1.22\ \lambda}{\mathrm{NA}}$ [31]. The lateral resolution of the confocal microscope (LR) with this objective falls within the range defined by the Rayleigh criterion and the optimal resolution for confocal microscopy [32], which, for the lasers used, is between slightly less than 300 nm and slightly more than 400 nm, for $\lambda = 532$ nm and $\lambda = 638$ nm, respectively.

Rayleigh radiation can be up to 10^7 times greater than that of Raman Stokes radiation. That is why notch filters or bandpass filters [14] are required to effectively discriminate against Rayleigh radiation. To control the depth from which scattered light is collected and achieve confocality [33], a pinhole is employed. The size of the pinhole determines the planes of the sample from which scattered photons reach the CCD, in addition to the focal plane. Given the thinness of the first layer of the samples studied, of 30-50 nm thickness, it was crucial to minimize the pinhole size to maximize information from the nanowires (NW) and contacts while minimizing substrate signal.

Filtered light from the sample is scattered off a diffraction grating before reaching the CCD. The frequency resolution of the Raman spectrum depends on the number of lines per millimeter of this grating. For these measurements, gratings with 1800 lines/mm and 2400 lines/mm were used, providing the highest possible frequency resolution and sufficient Raman intensity, minimizing signal-to-noise ratio (SNR). Finally, the light from the diffraction grating is focused onto a CCD array.

The parameters used to take the spectra of both samples are detailed below in Table 1:

Sample	Measurement Type	Laser wavelength (nm)	Power (mW)	Grating (lines/mm)	Pinhole (μm)
Polycrystalline Si	Maps	638	20	1800	100
Monocrystalline Si	Maps	638	20	1800	100
		532	18	2400	10
	Z linescans	532	5.7	2400	10
		532	11	2400	10
		532	14	2400	10
		532	18	2400	10
	X-Y linescans	532	18	2400	10
	Edge linescan	532	18	2400	10

Table 1: Parameters used to take the spectra of the polycrystalline and monocrystalline Si samples.

The acquisition time for the spectra varies depending on the type of measurement and the other parameters involved. However, before fitting the Voigt profile to the experimental spectra, the intensity values were normalized to 1 second acquisition time. For instance, if a spectrum was taken for 1.5 seconds, the obtained intensity was divided by a factor of 1.5. This normalization allows for a fair comparison of intensities across different measurements, accounting for the varying parameters that influence Raman intensity. Additionally, it helps maintain a low signal-to-noise ratio to minimize fitting errors due to noise. Each point spectrum was taken only once, meaning the number of accumulations is 1. The objective used in all measurements is 100X, as mentioned in the previous section.

2.4 Data treatment

Both the substrate of the polycrystalline Si sample and that of the monocrystalline Si sample, as well as the NW layer in the latter, are oriented in the $\langle 001 \rangle$ crystalline direction. For backscattering from a $\langle 001 \rangle$ surface, the Raman tensors R_x and R_y correspond to scattering by transverse optical phonons (TO) polarized along the x and y axes, respectively. Meanwhile, R_z corresponds to scattering by longitudinal optical phonons (LO) polarized along the z axis [19]. In the absence of stress, the three corresponding Raman optical modes of silicon have the same Raman shift of $\omega_{j0} = 520.7$ cm^{-1} $(j = 1, 2, 3)$ at ambient temperature (25°C), that is, they are degenerate. So the adjustment we have to make will only be to one peak, which corresponds to

the peak due to one-phonon processes, which are significantly more probable than two-phonon processes.

To analyze the experimentally obtained Raman spectra, a Python code was developed to fit the Voigt function to the experimental Raman peaks using the `LMFIT` library [34]. The `PANDAS` library [35] was employed to read tables containing experimental data for point spectra, profiles and maps, while the `NUMPY` library [36] was used for various numerical operations. For the analytical convolution [37] of the Lorentzian and Gaussian functions to obtain the Voigt function, the Fadeeva function [38], implemented as *wofz* in the `SCIPY` library [39], was utilized. The expression used to calculate the Voigt profile is the following [37]:

$$V = A\frac{2\sqrt{\ln(2)}}{\Delta_G\sqrt{\pi}}\Re|w(z)| \tag{2.29}$$

Where A is the amplitude of the function, Δ_G is the FWHM of the Gaussian, $w(z)$ is the Fadeeva function and $z = x + iy$, where x and z have been defined in the following way:

$$x = \frac{2\sqrt{\ln(2)}(\omega - \omega_0)}{\Delta_G} \quad ; \quad y = \frac{\Delta_L\sqrt{\ln(2)}}{\Delta_G}$$

Where ω is the Raman shift, ω_0 is the center of the Voigt function and Δ_L is the FWHM of the Lorentzian.

The `MATPLOTLIB` library [40] was used for the graphical representation of the parameters obtained from the fit, including spectra, profiles and maps. This code allows fitting a Voigt profile with a single peak or two peaks. To distinguish the top contact, where the NW is located, from the substrate, the top part had to be laser heated. Without heating, it was not possible to differentiate between the layer of the NW and the Si substrate based on peak position and FWHM, as most of the received signal originated from the substrate.

One challenge when automatically fitting a hyper-spectral map, for samples with various thin layers such as the studied in this work, is determining the best fit for each spectrum. Some spectra may be best fitted with a single peak, while others, particularly in regions where heating occurs, may require a two-peak fit.

When two peaks appear due to heating, one is due to the NW layer and the other to the substrate. Therefore, it is essential to use a suitable criterion to differentiate between spectra that need a two-peak fit and those that do not.

To address this, a statistical criterion based on the coefficient of determination (r^2) has been implemented in the developed Python code, enabling the distinction between spectra requiring a two-peak fit and those that do not. The coefficient of determination is a statistical metric that indicates the proportion of variation in a dependent variable that can be explained by an independent variable. In this context, it is a value between 0 and 1 that measures how well the Voigt function fits the Raman spectra.

First, a single-peak fit is performed. If the r^2 value for any spectrum is less than 0.99, a threshold is calculated. This threshold is determined by calculating the difference between the maximum and minimum r^2 values among all spectra in the map. Then, this difference is multiplied by a factor (0.9) and added to the minimum r^2 value to obtain the threshold. Then, a two-peak fit is applied to spectra with a r^2 value lower than this threshold. If the calculated threshold exceeds 0.99, the two-peak fit is performed on spectra with a r^2 value less than 0.99. The criterion

applied is as follows:

$$r^2_{\text{Single peak}} < r^2_{\text{threshold}} \quad \text{ó} \quad 0.99 \quad \Longrightarrow \quad \text{Double peak} \tag{2.30}$$

This approach ensures that spectra with an r^2 value lower than the threshold or 0.99 are fitted with two peaks, which is more accurate in these cases. However, applying this criterion introduces new challenges that require additional constraints. Specifically, the parameters for peak 1, henceforth referred as *substrate peak*, and peak 2, named hereafter *NW peak*, in the two-peak fit must be constrained to ensure accurate fitting.

A constraint applied to the substrate peak is that its Lorentzian FWHM (Full Width at Half Maximum) should fall within the range observed in the substrate area where no contact layer is present. This means the width should average between the maximum and minimum widths observed in spectra taken from the substrate-only region of the map. To achieve this, the Lorentzian FWHM of a spectrum and its neighboring spectra in the substrate-only area are averaged. The allowable range for the Lorentzian FWHM of peak 1, henceforth referred as substrate peak, is set to be within 15% above or below this average. Regarding the peak position (Raman shift), the minimum value for the substrate peak Raman shift should exceed that of the single-peak fit to account for potential heating-induced shifts. Furthermore, a minimum intensity threshold is imposed for this peak, set at the average noise level.

Regarding constraints for the NW peak, its Raman shift must be lower than the minimum Raman shift of the single-peak fit, with its upper limit set by the single-peak fit's minimum position. This adjustment allows us to differentiate the heating effect on the upper layer, which causes a decrease in Raman shift. The underlying assumption is that the NW layer, due to size constraints, dissipates heat less effectively, thus showing a higher temperature.

For the Lorentzian FWHM, the minimum width of the NW peak should be at least as wide as the minimum averaged substrate width, since due to heating effects the width of this peak should be broader. Additionally, the intensity of this peak must be at least equal to the average noise level and at most equal to the minimum intensity of the single-peak fit, as signals from this layer are typically very weak.

To avoid the NW peak from being fitted to very low-intensity peaks that may arise from noise, a filter has been imposed. This filter excludes peaks fitted with a two-peak model if the NW peak intensity is comparable to noise. Thus, a minimum intensity threshold has been established based on the noise level plus one standard deviation, considering the Gaussian nature of noise. Using the standard deviation of a Gaussian distribution is appropriate as noise is a natural phenomenon often exhibiting Gaussian behavior.

When fitting spectra of linescans instead of maps, additional constraints must be considered depending on the linescan type, in addition to those mentioned for mapping. The following constraint was added: the r^2 value of the peaks, besides meeting previous conditions, had to exceed 0.83. This was to avoid poorly fitting spectra dominated by noise, where a two-peak fit would be inappropriate. This mainly happens when a depth profile (Z) is made. Since the laser is not focused on the sample at the beginning and end of the profile, the Raman intensity received from these spectra is slightly higher than that of the noise. Thus, to avoid unnecessary two-peak fits in cases where no genuine second peak exists, this condition had to be imposed.

Before beginning the analysis of the maps and profiles made in section 3, two spectra are presented below, in Figures 7 and 8 obtained from a map taken for the monocrystalline Si sample.

To illustrate the difference that occurs when there is heating, due to bad heat dissipation, the spectrum in Figure 7 has been obtained from a region where there was clear heating and, therefore, two peaks well differentiated. In contrast, the spectrum in Figure 8 has been obtained from a section with good heat dissipation. Both spectra are presented with the single peak fit (upper graph) and with the double peak fit (lower graph).

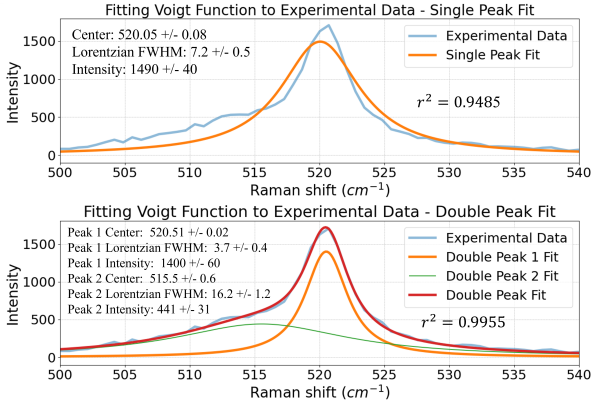

Figure 7: Raman spectrum of the triangular contact with the NW, where heating has caused the NW layer's peak to separate and appear shifted to lower Raman shifts and broader.

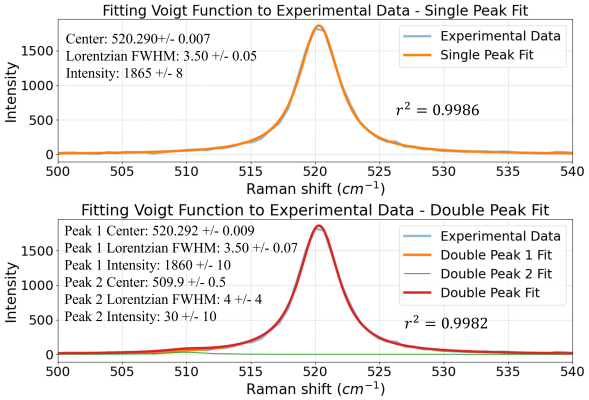

Figure 8: Raman spectrum of the non-triangular contact region, where the region has been able to dissipate heat and the substrate cannot be differentiated from the NW layer.

It can be clearly observed how the double peak adjustment is necessary in Figure 7 where there is heating. Furthermore, following the criterion explained for fitting with two peaks in both maps and linescans, the spectrum of Figure 8, did not require two peaks because the coefficient of determination r^2 is more than 0.99.

The value of the Gaussian FWHM has been adjusted to the error value of the experimental equipment for all spectra adjustments performed. The Python code used is presented in the appendix (Python Code for Linescans), and available upon request.

3 Experimental results and discussion

In this section, the experimental results obtained will be presented, and a detailed discussion of these results will be conducted. Firstly, the polycrystalline Si sample will be analyzed using the 638 nm laser. Next, the analysis will be performed on the monocrystalline Si sample with its surface perpendicular to the crystalline direction $\langle 001 \rangle$ with both lasers: 638 nm and 532 nm. Finally, the influence of polarization on the edges of the sample will be studied with the 532 nm laser, focusing on specific position marks and a contact edge.

3.1 Polycristalline Si samples

This sample has been studied using a 638 nm wavelength laser. Maps were obtained with the NW oriented at 0° relative to the laser polarization, meaning parallel to the laser polarization. In this configuration, due to the interaction between the laser light and the NW structure, the Raman signal emitted by the NW is enhanced compared to the signal emitted at other angles with respect to the polarization [14].

Column 7 of the NWs with triangular contact, located on the left side of Figure 3, has been

analyzed. The NWs in this column have a length of 1000 nm. This analysis focuses on the variations in stress or energy dissipation that occur as the width of the NW changes. Additionally, NWs with the maximum available length were chosen, as this size is required for fabricating devices that include the necessary gates and connections for qubit operation.

The size of all the maps taken for this sample is 7×5 μm and the step, that is, the distance traveled in the sample between the volume of two point spectra, is 0.1 μm. Figure 9 (a) displays the optical image of the sample region where the map was acquired, specifically where the NW that is 30 nm wide and 1000 nm long is located. In Figure 9 (b), the intensity map of the Voigt profiles used to fit the experimental Raman spectra with a single peak is represented. Figures (c) and (d) present the values of the coefficient of determination (r^2) for the least squares fit with a single peak and with two peaks, respectively. The two-peak fitting is performed exclusively for spectra that meet the criteria described in section 2.4.

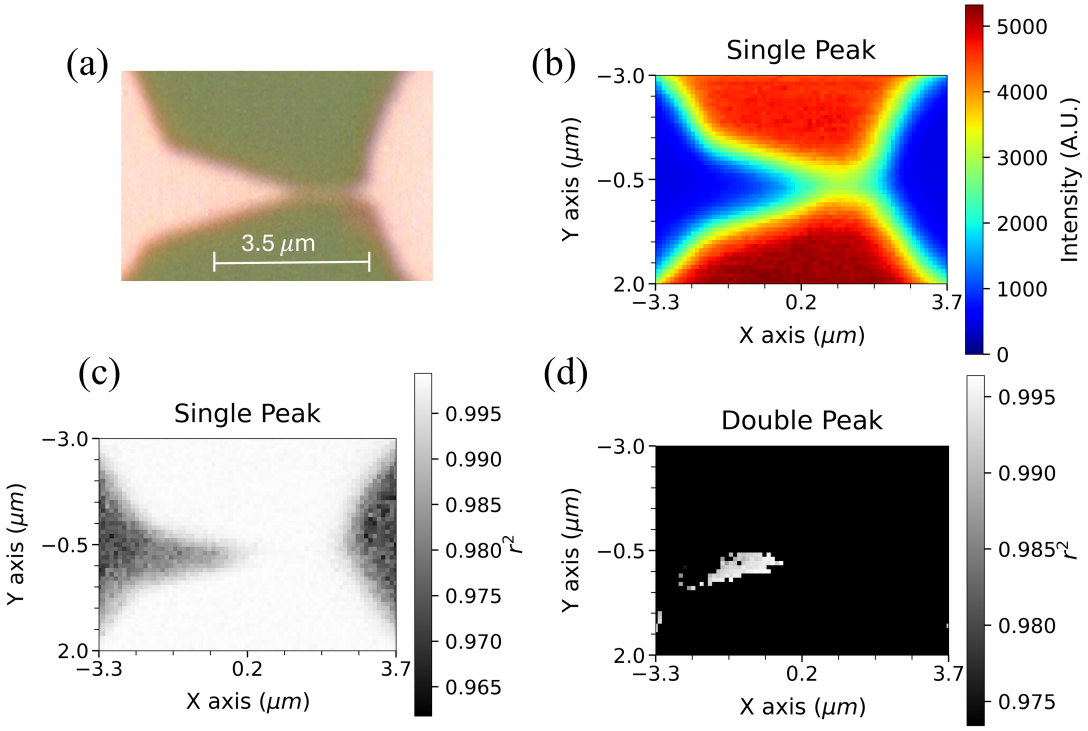

Figure 9: (a) Optical image of the sample region where the map is created. (b) Representation of the intensities of the Voigt profiles fitted to the experimental spectra using a single peak. (c) Coefficient of determination for the single peak fits. (d) Coefficient of determination for the two-peak fits.

Before analyzing the information provided by the intensity map, it is crucial to estimate which layer most of the Raman intensity originates from, in order to accurately interpret the intensity distribution of the Raman spectra. So, to estimate how much signal we receive from each layer, we must consider the penetration depth of the laser in Si, which is determined by the intensity of the electric field and the wavelength dependent absorption by the material. The penetration depth for a laser of a certain wavelength in a material can be determined using the Beer-Lambert law [41]:

$$I \sim e^{-\alpha d} \tag{3.1}$$

where I is the intensity of the electric field, α is the absorption coefficient of the material, and

d is the penetration depth.

The absorption coefficient can be obtained through the following expression [41]:

$$\alpha = \alpha(\lambda) = \frac{4\pi k(\lambda)}{\lambda} \tag{3.2}$$

where λ is the wavelength and $k(\lambda)$ is the imaginary part of the refractive index, which depends on the wavelength.

Thus, for $\lambda = 638$ nm, the corresponding absorption coefficient is:

$$\alpha_{638\text{nm}} = \frac{4\pi k(\lambda)}{\lambda} = \frac{4\pi \cdot 0.015008}{638 \cdot 10^{-7}\ \text{cm}} = 2956.1\ \text{cm}^{-1} \tag{3.3}$$

where the imaginary part of the refractive index of silicon, $k(\lambda)$, has been obtained from the website [42], using data from the paper [41]. Now, by substituting equation (3.3) into equation (3.1) and knowing that the thickness of the NW layer is 32 nm, while assuming the initial laser intensity is 1, we can estimate the absorption of this layer:

$$I \sim e^{-\alpha d} = e^{-2956.1(\text{cm}^{-1})32\cdot 10^{-7}(\text{cm})} \sim 0.99 \tag{3.4}$$

This means the NW layer absorbs only 1% of the power of the emitted radiation by the 638 nm wavelength laser. Consequently, most of the detected Raman intensity originates from the substrate.

In the intensity map of Figure 9 (b), the shape of the NW layer is visible due to a loss of Raman intensity in the region containing the NW layer. As explained earlier, nearly all the detected signal originates from the substrate layer. However, when light passes through the NW layer, the laser's emitted light is scattered. This scattering reduces the light intensity reaching the substrate, resulting in a lower Raman signal.

The NW layer's contribution is observed when it is heated, because its peak shifts to lower Raman shifts. Therefore, the intensity of the peak corresponding to the substrate layer and the NW layer can be determined with the two-peak fit.

The intensity maps of the two peak fits are not included here, although they can be seen in the annex, because the intensity values of the NW peak and the substrate peak do not provide really relevant information beyond the relative intensity of one peak with respect to the other.

The intensity maps do not reveal observable differences between NWs with different widths (see appendix). However, variations in intensity are noticeable between the substrate regions at the top and bottom of the images. This difference, as seen in Figure 9 (b), arises because of accidental temperature increase inside the sample chamber, that causes the silicon to expand, thus altering the focal point from its initial position and affecting Raman intensities.

In Figures 9 (c) and 9 (d), it can be observed that the spectra fitted with two peaks show a better fit compared to those fitted with a single peak. This is indicated by a higher r^2 coefficient of determination in the latter case.

The Lorentzian FWHM maps of the Voigt profiles fitted to the experimental Raman spectra for the same 30 nm wide NW are now plotted in Figure 10.

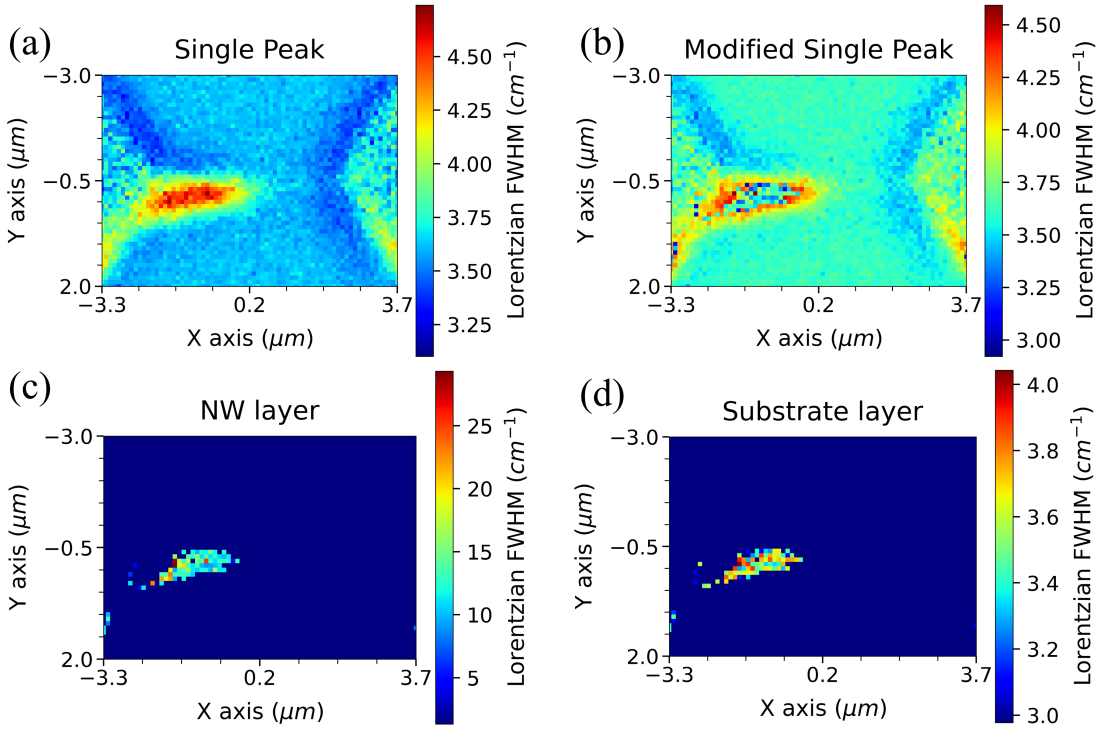

Figure 10: Representation of the Lorentzian FWHM for single-peak fitting (a), single-peak fitting, with the values of substrate peaks from point spectra where two-peak fitting has been performed (b), the NW peak from two-peak fitting (c) and the Lorentzian FWHM of the substrate peak from two-peak fitting (d).

In Figure 10 (a), the Lorentzian FWHM is observed to increase primarily in the triangular contact area (left contact). This increase in width can be attributed to the NW layer. Due to the small surface area at the triangular contact, the absorbed laser energy cannot be fully dissipated. The NW layer is situated over silicon oxide, a poor thermal conductor, making it difficult for the energy to dissipate within the small surface area and volume of the triangular contact of the silicon layer, leading to heating and broadening of the Raman peaks.

In the lower part of the image, the Lorentzian FWHM is also slightly higher than the average value because the laser transmits energy to a very small region for an extended period. The NW layer struggles to dissipate this energy compared to the Si substrate due to its small thickness and area and its placement on silicon oxide, so the NW layer is heated, producing this broadening of the Lorentzian FWHM.

The increase in the Lorentzian FWHM is mainly attributed to the NW layer's peak, as visualized by the two-peak fitting in Figure 10 (c). The Lorentzian FWHM of the substrate peak remains relatively constant or increases slightly, as the Si substrate also expands due to the laser energy and the temperature increase in the sample chamber, which can rise by up to 2^{o}C during mapping. This variation in temperature is known because the equipment incorporates a thermometer that allows to read the temperature instantly. For the point Raman spectra, where the two-peak fit is not applied in Figures 10 (c) and 10 (d), the background is determined by taking the minimum value and subtracting 10% of that value.

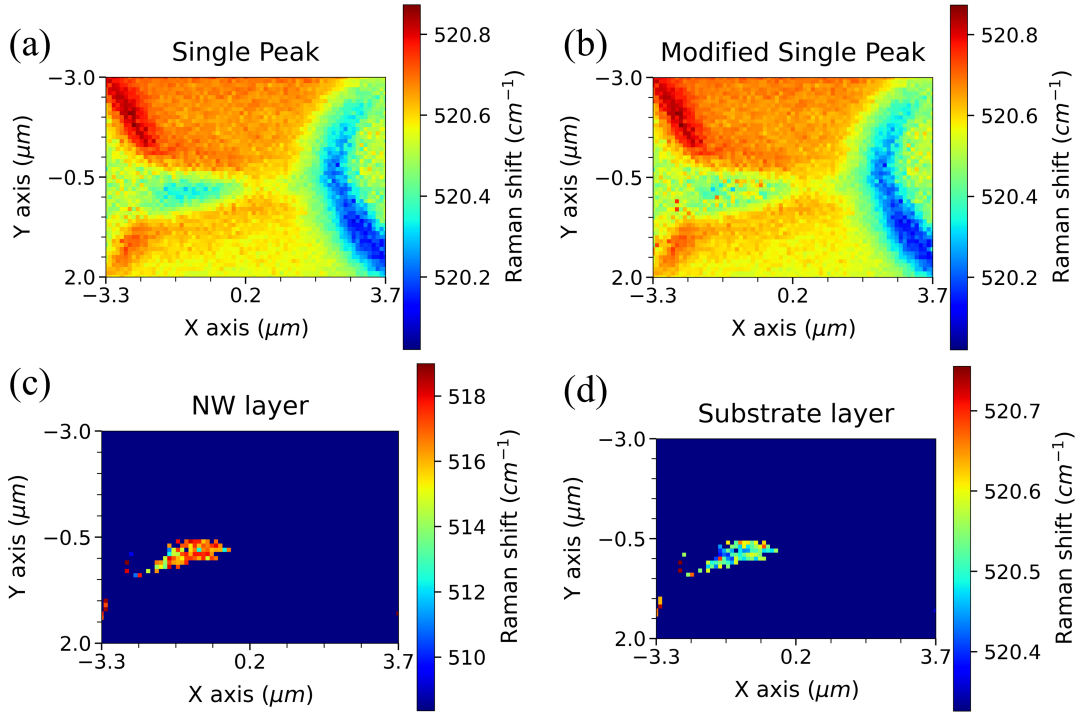

Figure 11: (a) Representation of the Raman shift for single-peak fitting. (b) Representation of the Raman shift for single-peak fitting, with the values of substrate peaks from point spectra where two-peak fitting has been performed. (c) Representation of the Raman shift of the NW peak from two-peak fitting. (d) Representation of the Raman shift of the substrate peak from two-peak fitting.

In Figure 11 (a), the Raman shift is observed to be higher at the edges of the triangular contact. This shift towards higher Raman shifts is likely due to compression [43] generated in the Si crystal lattice at the NW layer during etching. Conversely, the Raman shift is lower at the right contact, indicating a relaxation of the Si crystalline structure at this edge.

When analyzing the Raman shift, temperature must be considered as it influences the Raman shift by shifting it to lower values due to the relaxation of the crystalline structure as the material expands. As mentioned before, the temperature of the sample chamber increases during the spectrum acquisition for mapping, resulting in lower Raman shifts in the lower part of the image compared to the upper part. Additionally, the poor heat dissipation in the triangular contact area caused the Raman shift in this region to be lower than it would be if the area could dissipate energy effectively and avoid heating.

Figure 11 (b) shows the Raman shift map for the single-peak fit with the substrate peak values from the two-peak fit. This approach helps to partially eliminate the influence of temperature on the phonon frequency across the rest of the sample.

Figures 11 (c) and 11 (d) show the Raman shift values of the NW peak and the substrate peak from the two-peak fitting, respectively. The NW peak is shifted to frequencies significantly lower than the average value observed in the single-peak fit, whereas the substrate peak remains around the average value (520 - 521 cm^{-1}).

By analyzing the phonon frequencies (Raman shift), it is possible to determine both the temperature and the stress at the specific region where the Raman spectrum was acquired. The

temperature, expressed in °C, can be calculated based on the Raman shift values using an equation derived from Tsu et al. [44], expressed as follows:

$$T = T_0 + \frac{1}{-5.4 \cdot 10^{-5}(^{\circ}\mathrm{C}^{-1})} \ln\left(\frac{\omega}{\omega_0}\right) \tag{3.5}$$

where T_0 is the ambient temperature (25°C) and ω_0 is the silicon phonon frequency at that temperature.

The uniaxial stress (σ), expressed in GPa, for monocrystalline silicon oriented along the $\langle 001 \rangle$ crystalline direction can be determined from the Raman shift values (ω), expressed in cm^{-1}, as described by Li et al. (2022) [45], using the following equation:

$$\sigma_{\langle 001 \rangle} = -0.434 \left(\frac{\mathrm{GPa}}{\mathrm{cm}^{-1}}\right) (\omega - \omega_0) \tag{3.6}$$

where ω_0 is the Raman shift of the Si reference peak, ω is the Raman shift of the observed peak, and $\sigma_{\langle 001 \rangle}$ is the uniaxial stress for Si when the surface emitting the Raman signal is oriented along the $\langle 001 \rangle$ crystal direction. In this orientation, only the third mode of the Raman tensor is active, according to De Wolf et al. (1996) [19]. For this sample, the Si is polycrystalline in the NW layer and monocrystalline, oriented with its surface perpendicular to the $\langle 001 \rangle$ crystalline direction, in the substrate layer. Thus, for the NW layer, this equation does not accurately reflect the stress associated with the variation in phonon frequency. But it is adequate for the substrate layer, which provided almost all of the Raman signal, and for the NW layer and the substrate layer of the monocrystalline Si sample.

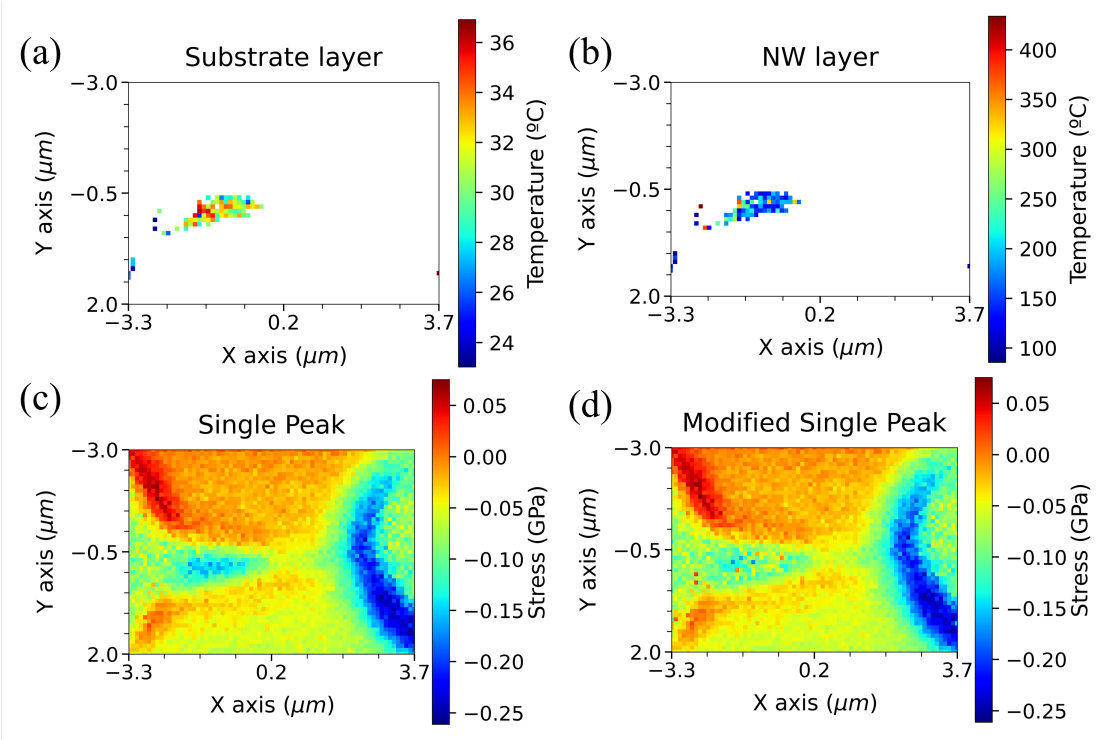

Figure 12: (a) Substrate peak temperature representation of the two-peak fit. (b) Representation of the NW peak temperature from the two-peak fit. (c) Stress representation for Si from the single-peak fit. (d) Representation of the Si stress for the single-peak fit and the single-peak fit with the substrate peak values substituting the single-peak values in the regions where a two-peak fit was applied.

Now, the temperature maps for the substrate and NW peaks using the two-peak fit, as well as the stress maps associated with variations in Raman shifts, are presented in Figure 12. The stress maps include both the single-peak fit and the single-peak fit with the substrate peak values substituting the single-peak values in the regions where a two-peak fit was applied. This approach helps to partially eliminate the temperature effect, allowing a clearer visualization of regions under stress, whether due to compression or elongation.

In Figure 12 (a), the temperature of the substrate peaks, obtained from Eq. (3.5), was around room temperature or slightly higher. Meanwhile, in Figure 12 (b), the temperature of the NW layer exhibited a minimum around 100°C. This means that the NW layer heated up considerably. This significant heating of the NW layer was attributed to the laser energy absorbed by the material, exacerbated by its poor heat dissipation capabilities.

Analyzing Figure 12 (d), positive stress (compression) occurs in the triangular contact, mainly in the upper part. As the temperature of the chamber raised during spectrum acquisition for mapping, the material underwent expansion or relaxation in the mapped areas. This phenomenon caused a decrease in Raman shifts, which was reflected in the transformation of Raman shift to temperature. Additionally, Si strain was observed in the left contact (triangular contact), while Si relaxation was visualized in the right contact. These effects can be attributed to the etching process during device manufacturing.

3.2 Monocrystalline Si samples

3.2.1 Analysis using a red laser: Hyperspectral maps

This sample was initially analyzed using a 638 nm wavelength laser, consistent with the methodology applied to the preceding sample. In addition to conducting an analysis with the sample oriented exclusively parallel to the direction of laser polarization, it was also positioned at 90°, 180°, and 270° relative to the polarization. This was done to study the effect of laser polarization in relation to the sample's position.

In Figure 13, the optical image is displayed showing the sample oriented with the NW length parallel to the laser polarization direction and then inverting it, i.e., rotating it by 180° (Fig. 13 (c)). Additionally, the Raman shift of the peaks is presented for each orientation of the sample. It was observed that the Raman shifts vary depending on the sample's orientation within the equipment. This was surprising as it seems to be something related to the equipment and not the sample. In principle, if the stresses are consistent, the difference in the Raman shift should remain the same regardless of whether the sample was rotated or not. Thus, it was hypothesized that a variation in the Raman peak positions might occur due to a surface topology change, such as a step on the sample's surface.

To investigate this, mapping was performed by sweeping from left to right (the usual method) and also from right to left, resulting in the maps shown in Figures 13 (e) and 13 (f) for each case. With these experiments, there were no changes in the Raman shifts. Therefore, we concluded that the scanning direction did not influence the results; rather, the orientation of the sample did. Considering this appeared to be a systematic error rather than an inherent sample characteristic, the decision was made to switch to a 532 nm wavelength laser to determine if the phenomenon persisted. As discussed in section 3.2.3, this phenomenon did not occur with the 532 nm laser. However, due to time constraints in completing the Master's Thesis, further investigation of this phenomenon remains to be undertaken.

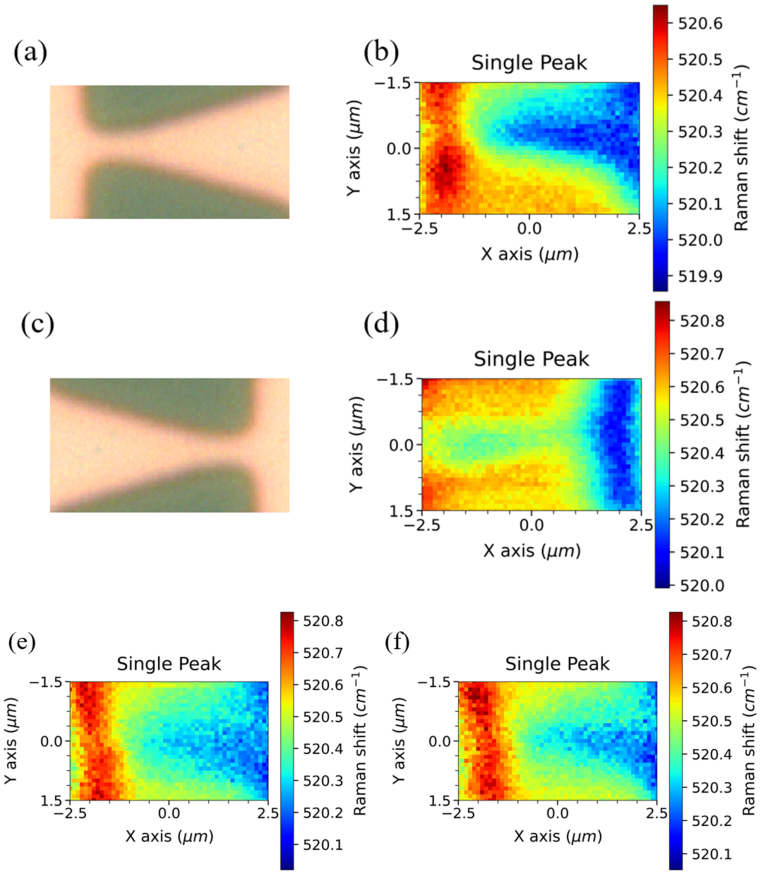

Figure 13: (a) Optical image of the sample region where the map was taken with the NW parallel to the laser polarization direction. (b) Representation of the Raman shift for single-peak fitting. (c) Optical image of the sample rotated 180° with respect to (a). (d) Representation of the Raman shift for single-peak fitting with the sample inverted. (e) Representation of the Raman shift for single-peak fitting for a right sweep. (f) Representation of the Raman shift for single-peak fitting for a left sweep.

So, this sample was analyzed using both a 532 nm wavelength laser and a 638 nm wavelength laser and both lasers experienced power losses before reaching the sample, due to allingment issues in the equipment, though the most relevant results have been achieved with the 532 nm wavelength laser because in this case the losses were smaller. Moreover, the penetration depth of the 532 nm wavelength laser in Si is smaller compared to that of the 638 nm laser, as it absorbs more efficiently, as demonstrated below. This characteristic enables gathering more detailed information from the NW layer, which is the focus of this study.

The penetration depth of the 532 nm wavelength laser can be estimated similarly to that of the 638 nm wavelength laser using the Beer-Lambert law (Eq. 3.1). This estimation requires knowledge of the absorption coefficient, which can be obtained from Schinke et al. [41] using the expression (Eq. 3.2). Utilizing the data from [42], based on the findings of [41], we determine the imaginary part of the refractive index of silicon, $k(\lambda)$. Substituting $k(\lambda)$ into Eq. 3.2, we derive the absorption coefficient of silicon for the 532 nm wavelength as follows:

$$\alpha_{532\text{nm}} = \frac{4\pi k(\lambda)}{\lambda} = \frac{4\pi \cdot 0.033822}{532 \cdot 10^{-7}\ \text{cm}} = 7989.2\ \text{cm}^{-1} \tag{3.7}$$

Now, by substituting Equation 3.7 into Equation (3.1) and considering the depth of the NW layer as 50 nm, with an initial laser intensity assumed to be 1, we can estimate the absorption of this layer:

$$I \sim e^{-\alpha d} = e^{-7989.2(\mathrm{cm}^{-1})50 \cdot 10^{-7}(\mathrm{cm})} \sim 0.96 \tag{3.8}$$

This means that the NW layer absorbed 4% of the power of the emitted radiation by the 532 nm wavelength laser. Therefore, the 532 nm laser yielded a larger $I_{\mathrm{NW}}/I_{\mathrm{Substrate}}$ ratio compared to that produced by the 638 nm wavelength laser. This is because silicon absorbs more efficiently at 532 nm, resulting in increased Raman scattering, while also reducing the intensity that reaches the substrate.

3.2.2 Analysis using a green laser: Spectral profiles (linescans)

In order to differentiate the NW layer from the substrate, since both are silicon with the same doping and thus exhibit nearly identical properties, and considering that the NW layer emits very little (only 4% of the light emitted by the laser is absorbed by the NW layer). We decided to heat the NW layer by applying a power high enough to shift the peak of the NW without causing damage. This involves providing an amount of energy that the NW can adequately dissipate. The dissipation of heat in the NWs depends on the material on which they are located [14]. In this case, the NWs lay on silicon oxide, which has low thermal conductivity. Therefore, the difference in dissipation between the bulk and the NWs arises because silicon is a better thermal conductor than silicon oxide. Energy dissipates more efficiently in the bulk silicon since it is conducted through the silicon layer, whereas dissipation in the NWs is less efficient due to the small volume of silicon and the need for dissipation through silicon oxide. The power that can be applied to each NW to heat it without damaging it may vary. For example, if an NW at some point is not in contact with its substrate, it will be unable to dissipate the energy and may evaporate.

Therefore, to conduct measurements with the spectral focus on both the NW layer and the silicon substrate layer, for comparison and to gather information from the NW layer, several in-depth linescans were performed at different powers on the NW aligned parallel to the laser polarization. This alignment maximizes the NW signal due to the interaction between light and the NW [14]. The goal was to determine the power levels at which the NW heats up and the depth at which the NW is located relative to the autofocus performed by the equipment. The autofocus is carried out automatically by the equipment, which conducts a depth linescan and then positions the sample at the height where the maximum intensity of the Raman peaks is detected, which is the substrate.

In Figure 14, the intensities (lines) and Lorentzian FWHM (dots) of the Voigt profile fit to the experimental spectra are shown, for depth profiles obtained with several values of laser powers. These are presented for both the single-peak fit and the two-peak fit, applied to the spectra that meet the criterion for performing the two-peak fit.

In Figure 14 (a), an increase in intensity is observed during the linescan as the laser beam focuses on the substrate, reaching a maximum and then gradually decreasing as the focus moves away from the substrate. This gradual decrease occurs because the NW layer contributes to the Raman signal as the focus aligns with it. The variation in intensities becomes more pronounced with higher laser power, causing a noticeable change in intensity as the focus moves away from the substrate.

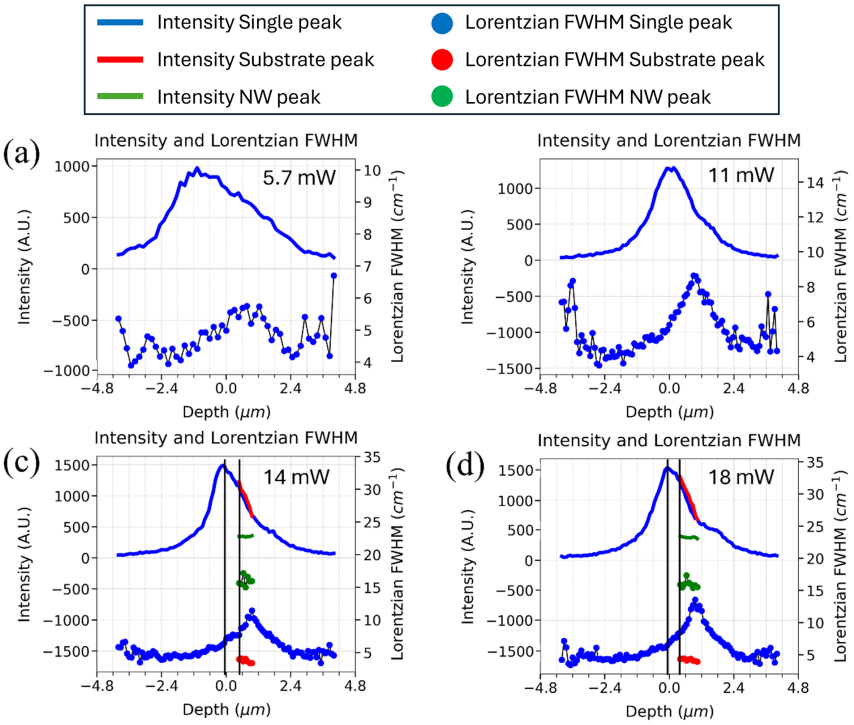

Figure 14: Representation of intensity (lines) and Lorentzian FWHM (dots) during a depth linescan across the NW at varying power levels: 5.7 mW (a), 11 mW (b), 14 mW (c), and 18 mW (d).

Figures 14 (b), (c) and (d) reveal a "shoulder" in the intensities representation, attributed to enhanced emission from the NW layer. By applying criteria for a double peak fitting, we determined the distance of the NW layer from the focal point, which ideally aligns with the substrate where the strongest signal originates. Depending on the distance at which the double peak fitting occurred relative to the intensity maximum, we estimated the NW layer was between 0.4 and 0.9 μm away from the substrate. This range is in agreement with known sample growth parameters, as the silicon oxide layer separating the NW layer from the Si substrate measures 400 nm, falling within the estimated distance range.

Upon conducting the double peak fit, it can be observed, in Figures 14 (c) and (d), that the intensity of the substrate peak (red line) closely matches that obtained from the single peak fit, while the intensity of the NW peak (green line) is notably lower.

It can also be observed that the Lorentzian FWHM increases as the laser power increases and comes into focus with the NW. This phenomenon occurs because the NW cannot effectively dissipate the energy from the laser, increasing its temperature considerably. Consequently, there is a significant variation in the Lorentzian FWHM of the peaks. Upon fitting with two peaks, it can be observed that the Lorentzian FWHM of the substrate peak remains similar to its average width in the single-peak fit, around 4 cm^{-1}, when the sample was not sufficiently heated due to low laser power. In contrast, the Lorentzian FWHM of the peak associated with the NW layer is much larger, approximately 16 cm^{-1}.

Figure 15 shows the values obtained for the Raman shift, during the same linescan in depth over the NW, and on the right y axis these values are transformed from the Raman shift to temperature, according to equation (3.5).

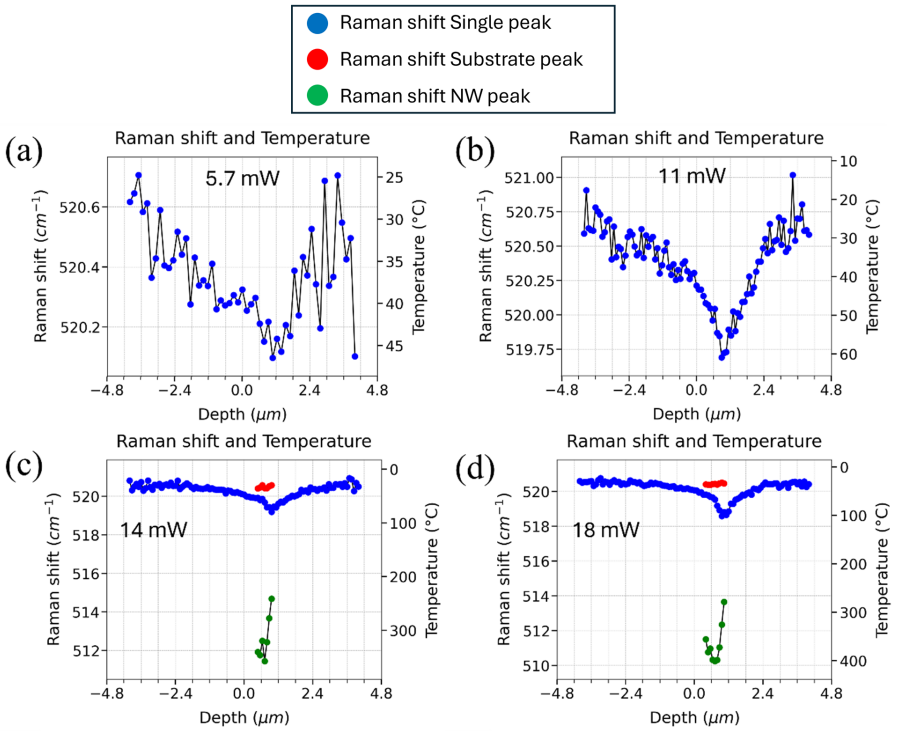

Figure 15: Representation of Raman shift and temperature during a depth linescan across the NW at varying power levels: 5.7 mW (a), 11 mW (b), 14 mW (c), and 18 mW (d).

In Figures 15 (c) and (d), it can be observed that the Raman shift of the peaks decreases in the NW region as the power increases, which is attributed to the heating of the NW due to its poor energy dissipation. When fitting to two peaks, the Raman shift of the substrate peak remains around 520.5 cm^{-1}, consistent with the single-peak fit when the sample is not heated. In contrast, the NW peak position shifts to lower frequencies, reaching 510 cm^{-1} at a power of 18 mW.

Using the variation of the Raman shift, the temperature can be estimated with Equation (3.5). This calculation yields temperatures exceeding 300°C for the NW, while the substrate temperature remains around 30°C or slightly higher. Temperature is a crucial factor to consider for the proper functioning of qubits and electronic devices in general.

It is also noticeable that the fit at the extremes of the linescan is poor, due to the low signal-to-noise ratio. In these cases, the condition that the intensity of the NW peak exceeds the average intensity of the noise plus its variance is not achieved, and, thus, the fit to two peaks is not performed.

The same linescan was performed at a power of 18 mW, but with the NW oriented perpendicular to the direction of the laser polarization. In this case, the NW should not heat up as much, making it more challenging to determine its Raman shift compared to the previous case. Figure 16 presents the values obtained for intensity (lines), Lorentzian FWHM (dots), Raman shift, and temperature of the experimental peaks after fitting to the Voigt profile.

Two local maxima can be observed in the intensity, in Figure 16 (a), located at approximately 0.9 μm, which roughly coincides with the distance of the NW layer from the Si substrate layer. Furthermore, there is a slight increase in the Lorentzian FWHM at the second intensity maximum, where the NW layer contributes more significantly. Therefore, this measurement also

allows us to identify the distance of the NW from the substrate, despite the much lower heating compared to the previous figures where the NW was parallel to the laser polarization direction. Consequently, even though the criterion for performing a two-peak fit is not met, this method still enables the detection of the NW layer.

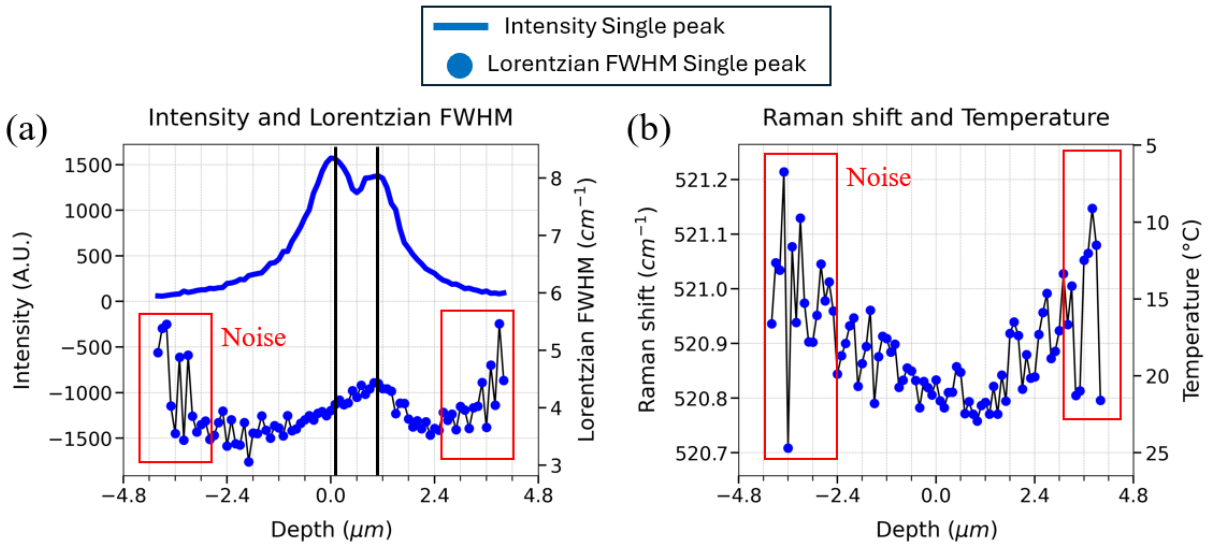

Figure 16: Representation of (a) the intensity (lines) and Lorentzian FWHM (dots) and (b) the Raman shift and temperature during a linescan at depth over the NW, perpendicular to the laser polarization direction, using 18 mW power.

If we analyze Figure 16 (b), we can observe that the variation in the peak position and temperature is small. Therefore, unlike the previous case, it is not as clear from the Raman shift where the NW is located. However, there is a minimum in the peak position that can be associated to the NW. This allows us to determine the position of the NW relative to the substrate, albeit with less clarity than before.

The results of some linescans performed at 18 mW power will be presented in the following, both in the direction parallel and perpendicular to the NW, with the NW length parallel to the laser polarization. Since the position of the NW had already been determined spectrally, the linescans were conducted at both the NW depth and the substrate depth to compare the variation in peak fitting parameters when focusing on each layer. In Figure 17, the intensity and Lorentzian FWHM of the profiles parallel to the NW are shown in Figures 17 (a) and (b), with the focus on the NW and on the substrate, respectively. Figures 17 (c) and (d) display the intensity values (lines) and Lorentzian FWHM (dots) of the profiles perpendicular to the NW.

We observed in Figure 17 (a), that the intensity is significantly higher in the area where the NW is located for a linescan parallel to the NW with the focus on it. The NW is approximately 2 μm in length, which corresponds to the region of highest intensity. This region is slightly larger than the actual length of the NW due to the laser's finite diameter and the contribution from the narrowest part of the triangular contact.

Moreover, the intensity of the single-peak fit is lower than the intensity of the substrate peak in the two-peak fit, indicating a poor single-peak fit in that region. The FWHM also increases significantly in this region, and the FWHM of the single-peak fit is nearly the same as that of the NW peak in the two-peak fit. This suggests that the decrease in intensity in the single-peak fit is due to a better width fit to achieve the highest possible r^2.

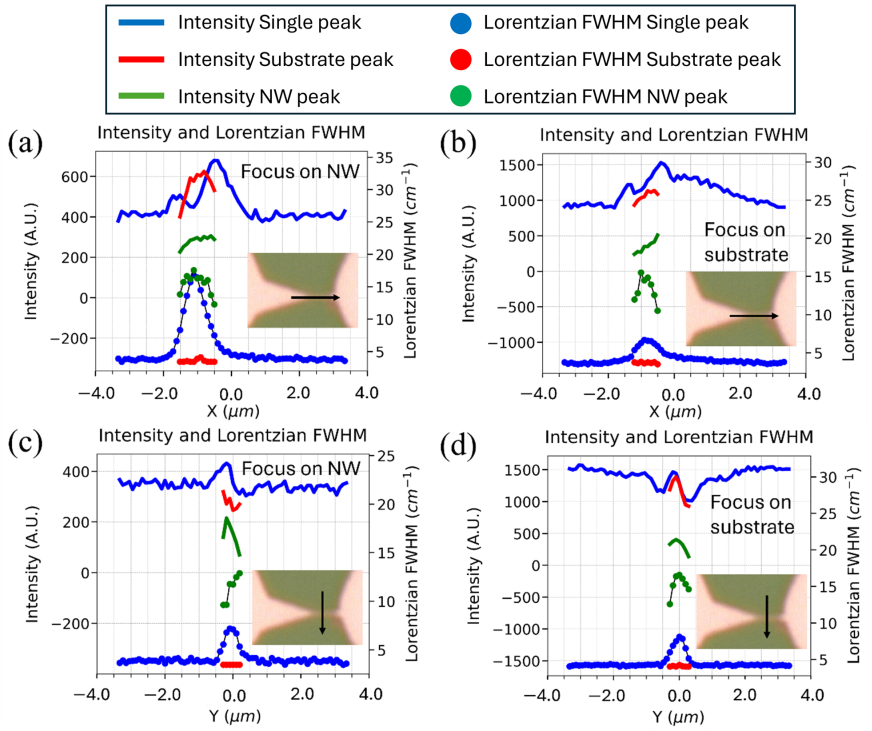

Figure 17: Representation of the intensity (lines) and the Lorentzian FWHM (dots) when making linescans parallel and perpendicular to the NW at a power of 18 mW, with the NW length parallel to the laser polarization direction. In Figures (a) and (b) the linescan has been made parallel to the NW on it and in the first the focus is on the NW layer, while in the second the focus is on the substrate layer. In Figures (c) and (d) the linescan has been made perpendicular to the NW, focusing on the NW layer and the substrate layer, respectively.

Comparing the same linescan in Figure 17 (b), with the focus on the substrate instead, it is clear that the overall intensity is higher because the substrate emits more Raman signal. The Lorentzian FWHM is lower in this case for both the single-peak fit and the NW peak of the two-peak fit, indicating that the NW layer is less heated, which is expected since the focus is on the substrate layer.

In Figure 17 (c), where a linescan was performed perpendicular to the NW with the focus on the NW layer, the highest intensity is observed when passing over the NW. The intensity when we are in the NW is the maximum that is reached, since we are focused on its layer. To accurately determine the width of the NW, the diameter of the laser beam would need to be deconvoluted, since the beam excites the NW if the distance from the NW to the center of the beam is less than the beam's diameter. The Lorentzian FWHM increases where the NW is located and continues to increase as the linescan progresses until the laser is no longer in contact with the NW. This is because each point spectrum taken with the beam exciting the NW results in heating, and due to the NW's poor energy dissipation, the last spectra in which the beam illuminates the NW show it to be hotter.

Comparing the linescan taken with the focus on the NW and the focus on the substrate, as shown in Figure 17 (d), it can be observed that when the laser is focused on the substrate, there is a drop in intensity just before and after the NW. This drop is not observed when it is focused on the NW, where there is only an increase in intensity due to the excitation of the NW. This intensity drop on both sides of the NW can be attributed to some scattered light at the NW

edges, preventing it from reaching the substrate and it emits less Raman signal. Additionally, since the NW is not in focus and not excited by the center of the laser, it scatters very little, resulting in an overall loss of Raman intensity. When the laser beam is focused on the NW, the combined Raman scattering from the NW and the substrate results in a Raman intensity similar to that of the substrate alone. In this case, the Lorentzian FWHM associated with the NW is even greater than that observed in the previous scenario.

Now, in Figure 18 the values obtained for the Raman shift of the peaks, as well as for the temperature, are presented.

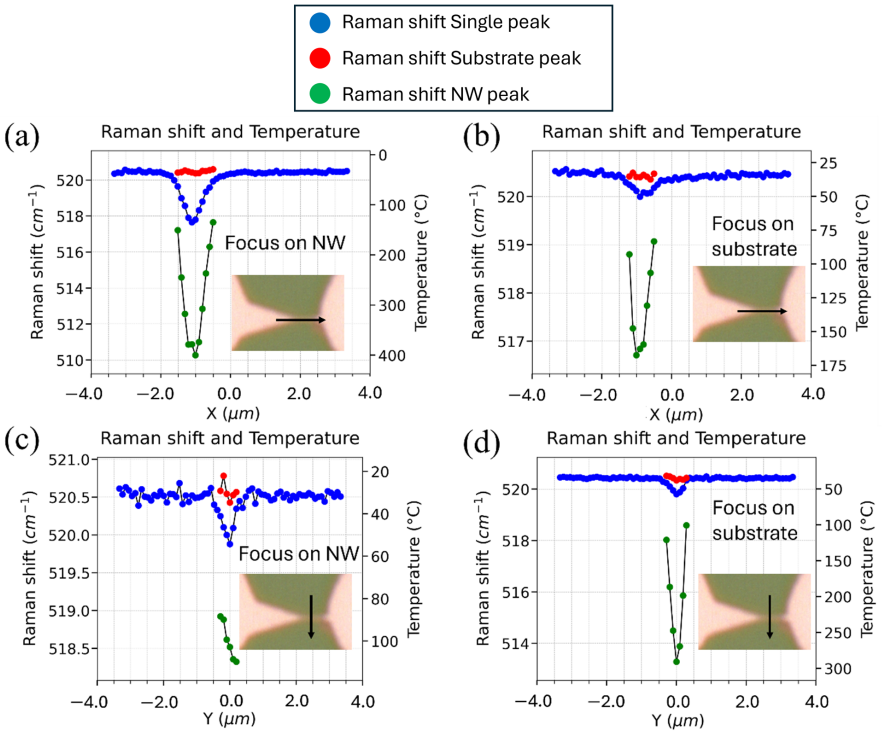

Figure 18: Representation of Raman shift and temperature by making linescans (a, b) parallel and (c, d) perpendicular (y-direction) to the NW at a power of 18 mW, with the NW parallel to the laser polarization direction. In Figures (a) and (c) the linescan has been made focusing on the NW layer, while in (b) and (d) the focus is on the substrate.

In Figure 18 (a), which shows the linescan parallel to the NW with the focus on that layer, there is a significant decrease in the Raman shift. When fitting to two peaks, the peak position of the substrate remains at the average position of the single-peak fit when the sample is not heated. However, the position of the peak associated with the NW decreases to 510 cm^{-1}, indicating that the NW reaches temperatures up to 400°C. This coincides with very large Lorentzian FWHMs, which is a heating effect. In this case, the variation of the Raman shift of the peak associated with the NW, with respect to the fit to a single peak, is greater than the relative variation between both in width. In contrast, when focusing on the substrate layer, Figure 18 (b), the minimum frequency is much higher, just below 517 cm^{-1}, resulting in an NW temperature of about 160°C. This abruptly change in temperature with slightly change in focus measurement, shows the highly potential of Raman microscopy to analyse nanostructures. Nonetheless, this consideration must be taken into account for further analysis.

In the linescans made perpendicular to the NW, shown in Figures 18 (c) and 18 (d), there is

also a noticeable decrease in Raman shift due to heating when reaching the NW. The decrease in Raman shift is slightly greater for the single-peak fit when the focus is on the substrate. However, when fitting with two peaks, the Raman shift associated with the NW peak reaches values lower than 514 cm^{-1}, corresponding to a temperature close to 300°C.

Now, let us compare the previous results with those obtained when the NW is perpendicular to the laser polarization direction. In Figure 19, the intensity values (lines) and the Lorentzian FWHM (dots) of the Voigt profiles fitted to the experimental spectra for linescans both parallel and perpendicular to the NW are presented, at the depths of both the substrate and the NW layers.

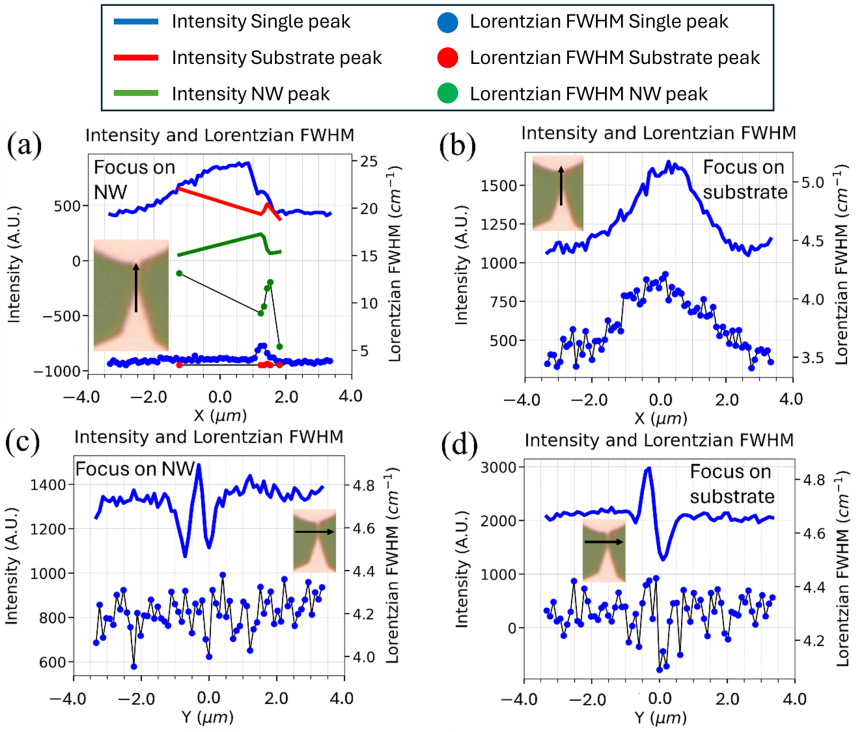

Figure 19: Representation of the intensity and the Lorentzian FWHM when making linescans (a, b) parallel and (c, d) perpendicular to the NW at a power of 18 mW, with the NW perpendicular to the laser polarization direction. In Figures (a) and (c) the linescan has been made on the NW layer, while in (b) and (d) the focus is on the substrate layer.

In Figure 19 (a), the intensity (lines) and Lorentzian FWHM (dots) values are presented for a linescan parallel to the NW, with the NW perpendicular to the laser polarization and the focus on the NW layer. It can be observed that although the effect is much smaller compared to when the NW is parallel to the polarization, in this case the NW layer still shows some heating, resulting in a slight increase in the FWHM in a localized region. Comparing this with the representation when the focus is on the substrate layer, Figure 19 (b), in the latter case, the Lorentzian FWHM exhibits smaller variations, and the conditions required for fitting with two peaks are not achieved.

The variation in intensity is different when performing a linescan perpendicular to the NW with the focus on the NW layer, Figure 19 (c), compared to when the focus is on the substrate layer, Figure 19 (d). When focusing on the NW layer, there is a noticeable decrease in intensity just before and after the NW. In contrast, when the laser is focused on the substrate layer, the

intensity fluctuates as it passes through the NW, with a peak and dip around the NW, before returning to the intensity level of the substrate layer. The origin of this different behavior requires further investigation.

Figure 20 shows the values obtained for the Raman shift of the peaks, as well as for the temperature.

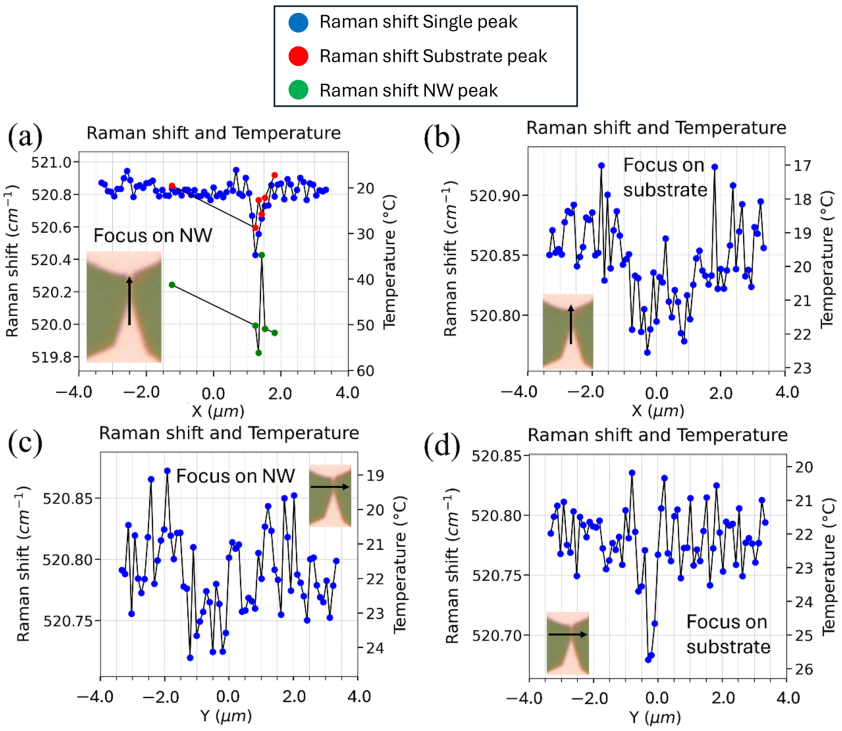

Figure 20: Representation of the Raman shift and temperature when making linescans (a, b) parallel and (c, d) perpendicular to the NW at a power of 18 mW, with the NW perpendicular to the laser polarization direction. In Figures (a) and (c) the linescan has been made on the NW layer, while in (b) and (d) the focus is on the substrate layer.

The Raman shift, in Figure 20 (a), decreases in the region where heating occurs, corresponding to the area where the double-peak fit has been applied. The temperature of the NW peak, obtained from the fit to two peaks in that region, reaches values around 50°C.

Figures 20 (b), (c) and (d) show minimal changes in the Raman shift. However, there is a slight decrease in the Raman shifts in Figure 20 (d), particularly in the region where higher intensity was observed in Figure 19 (d), which corresponds to the area around the NW. Nevertheless, this variation is so slight that it could potentially be attributed to equipment error. The variations in Raman shifts correlate with changes in the Lorentzian FWHM, suggesting they are linked to temperature variations within the sample.

3.2.3 Analysis using a green laser: Hyperspectral maps

In this section, we present maps obtained using a 532 nm wavelength laser, where we varied the orientation of the sample relatively to the laser polarization direction.

Firstly, Figure 21 depicts maps showing the intensity values derived from Voigt profile fits to experimental data. These maps were obtained with the NW length aligned parallel to the laser

polarization direction. Additionally, an optical image of the sample region where point spectra were collected to create the maps is presented in Figure 21 (a).

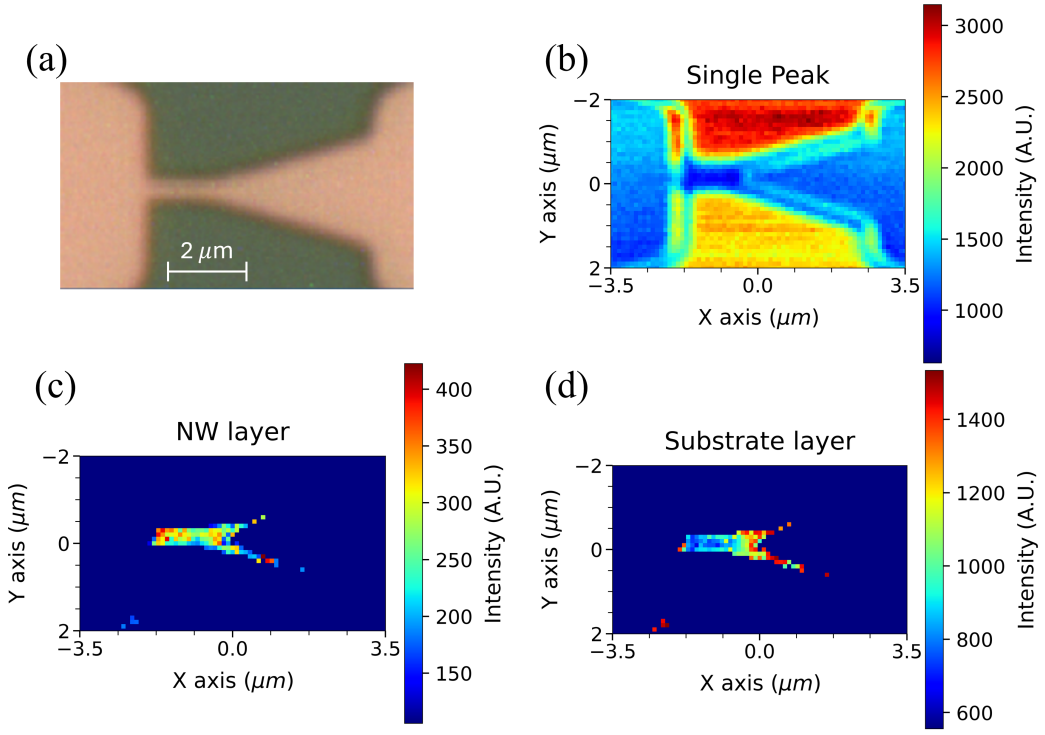

Figure 21: Optical image of the region of the sample where the map has been made (a), along with the maps with the values of the intensities when performing the fit of the Voigt function both for the fit to a single peak (b) and for a two-peak fit displaying data for the NW layer (c) and the substrate layer (d).

In Figure 21 (b), the intensity of the Voigt profiles of the fit to a single peak reveal lower intensities in the NW layer, as previously noted in Section 3.1. Moreover, using the 532 nm wavelength laser provides finer details compared to the 638 nm wavelength, attributed to increased absorption by the NW layer and enhanced Raman signal emission of this layer. There is a distinct pattern where intensity increases along the edges and decreases sharply in a narrow region, most prominently perpendicular to the laser polarization direction.

The intensities of the NW and substrate layers are discerned from spectra meeting the criterion for two-peak fitting, depicted in Figures 21 (c) and 21 (d). The intensity of the peaks of the NW layer, Figure 21 (c), are greater in the NW region, owing to its enhanced interaction with the incident beam's electromagnetic field when the NW is aligned parallel to the laser polarization [14]. Consequently, the substrate peaks in that region have a lower intensity with respect to other regions of the sample. In addition, the region of the NW is the area with lower Raman intensity as can be seen in Figure 21 (b) of the intensities from fitting to a single peak.

In Figure 22 (a), from the single-peak fit, the Lorentzian FWHM is noticeably larger in the NW and along the edges of the triangular contact, as well as along edges of the sample parallel to the laser polarization direction. In Figure 22 (b), by substituting the substrate peaks in the fit map to a single peak, the increase in the Lorentzian FWHM can be better seen on the edges that are parallel to the direction of laser polarization. Figures 22 (c) and 22 (d) show a significant increase in Lorentzian FWHM primarily within the NW and its layer, while the substrate layer

maintains a relatively constant value. This increase in FWHM within the NW layer is attributed to heating caused by the laser energy, which emits with a power of 18 mW, coupled with its poor energy dissipation due to its small volume and placement on silicon oxide.

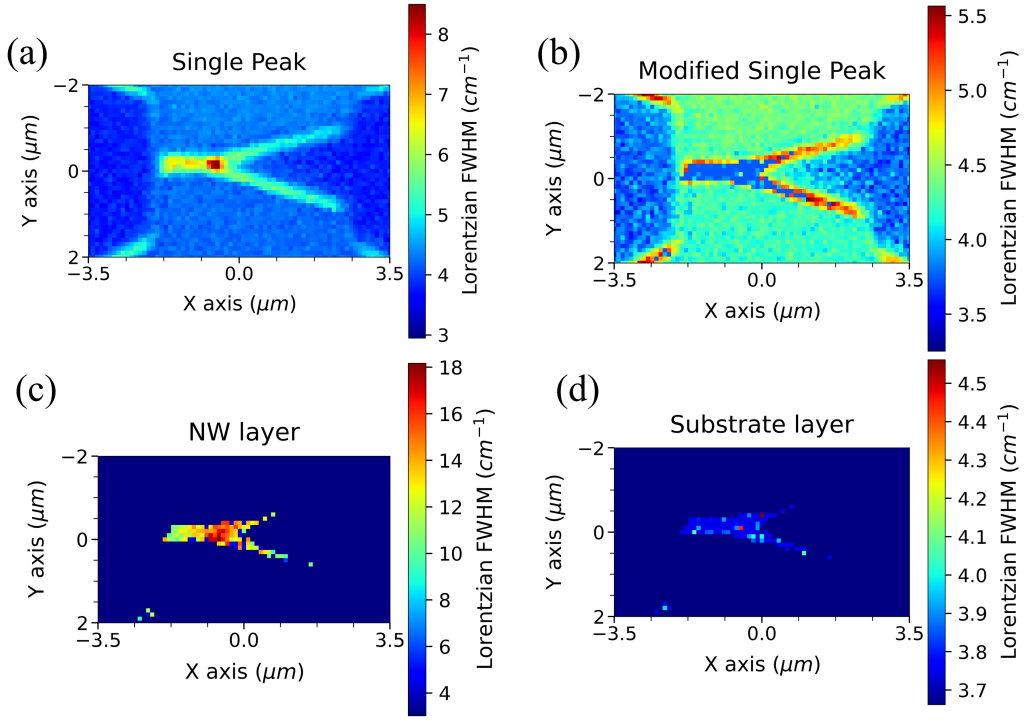

Figure 22: Representation of the Lorentzian FWHM (a) for single-peak fitting, (b) for single-peak fitting, with the values of substrate peaks from point spectra where two-peak fitting was performed, (c) for the NW peak and (d) for the substrate peak from the two-peak fitting.

Figure 23 (a) shows the values of the Raman shift for the fit to a single peak and Figure 23 (b) for the fit to a single peak by replacing the values of it with the values of the substrate peak in the spectra where the fit to two peaks was made. Furthermore, the Raman shift values of the two-peak fit for the NW layer and the substrate layer are shown in Figures 23 (c) and 23 (d), respectively.

For the single-peak fit, it is noticeable that the Raman shift is lower in the area of the NW and along the edges parallel to the laser polarization direction. This indicates a decrease in Raman shift in regions where the Lorentzian FWHM increases, as observed in Figure 22 (a).

As can be seen in Figures 23 (c) and 23 (d), the main decrease Raman shift occurs in the NW and in its layer, while the substrate Raman shift remains relatively constant. This behavior mirrors the trends observed in the Lorentzian FWHM maps, reflecting the heating effects within the NW layer due to its poor ability to dissipate energy absorbed from the laser.

In order to compare the effects of sample orientation relative to the laser polarization direction, Figure 24 presents the sample optical image, intensity (b), Lorentzian FWHM (c), and Raman shift (d) maps, with the NW length perpendicular to the laser polarization direction.

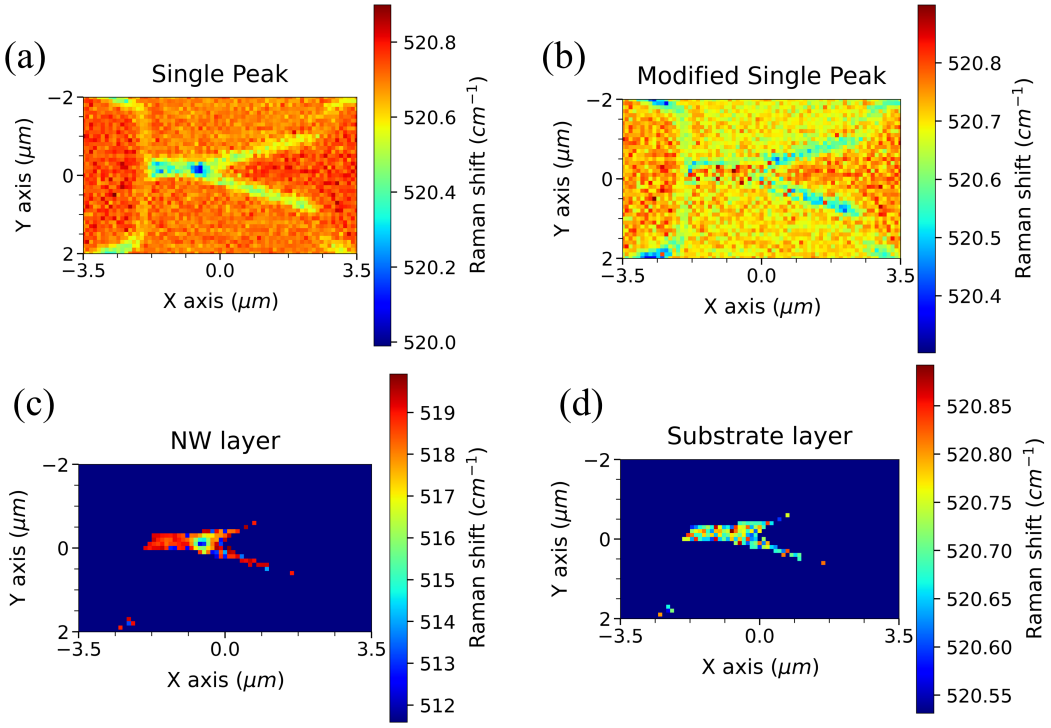

Figure 23: Representation of the Raman shift for (a) single-peak fitting, (b) single-peak fitting, with the values of substrate peaks from point spectra where two-peak fitting has been performed, (c) the NW peak and (d) the substrate peak from the two-peak fitting.

In this case, there is also a noticeable pattern in the intensity map, such as in Figure 21 (a). There is a slight increase in intensity at the edges, followed by a decrease until it reaches the substrate in a small region, where the intensity increases again. This effect is more pronounced along the edges perpendicular to the laser polarization direction, just like when the NW is parallel to the laser polarization direction. Since the NW is not aligned parallel to the laser polarization, it does not heat up or it does it minimally. So, the NW layer peaks do not shift significantly, thereby not meeting the criterion for a two-peak fit.

On the other hand, Figure 24 (c) shows that the Lorentzian FWHM is larger at the edges parallel to the laser polarization. Similarly, in the Raman shift map, Figure 24 (d), the Raman shift is lower at the edges aligned with the direction of laser polarization.

Thus, the increase of the FWHM and the decrease in Raman shift observed at these edges could be attributed to edge effects during the etching process in device manufacturing, such as compression or strain phenomena.

Stress and temperature maps derived from Raman shifts are presented in Figure 25. These maps were generated using fits where the single peak fit were replaced by the substrate layer peaks, where the two-peak fit was realized, for samples oriented at both 0° and 90° relative to the laser polarization direction. This maps aims to exclude regions of the sample affected by heating, enabling a clearer observation of the polarization effect on the edges. This analysis provides insights into the stress induced during the etching process to achieve the desired geometry.

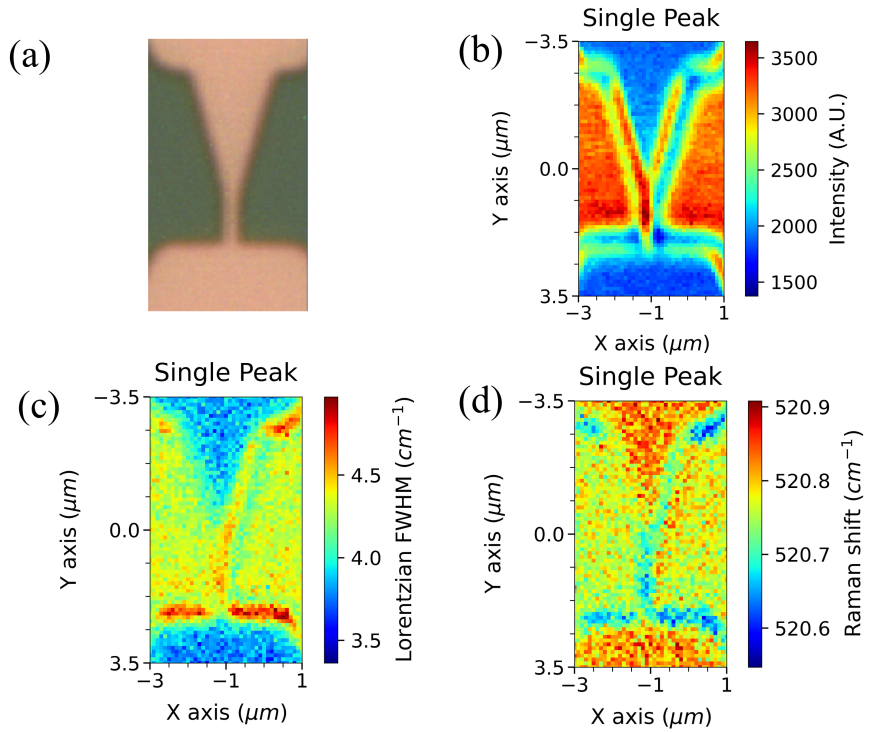

Figure 24: (a) Optical image of the sample's region where the map has been obtained. (b) Representation of the Raman intensity, (c) Lorentzian FWHM and (d) Raman shift for single-peak fitting.

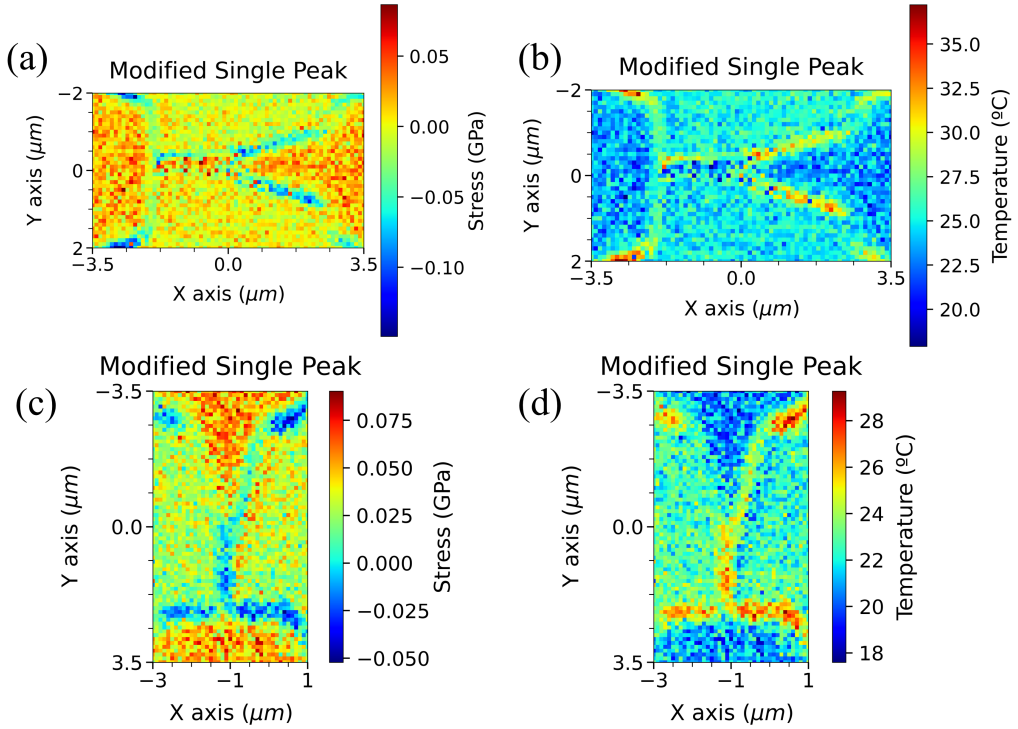

Figure 25: Representation of the stress for the adjustment to a modified peak with the peaks of the substrate where the adjustment to two peaks has been carried out for both the sample at 0° (a) and for the sample at 90° (c). In addition, the temperature maps are represented in the same way for both the sample at 0° (b) and the sample at 90° (d).

The temperature maps show a relatively small temperature range, ±7°C, which is in our limit resolution. However, it may signal as well a reduced thermal conductivity in the contact edges together with some polarization effects. Furthermore, since the temperature represented on the map is lower than the actual chamber temperature during the mapping process, this indicates that the observed polarization-dependent edge effect is attributed to stress and microstructures formed due to etching during the manufacturing process.

3.2.4 Influence of the green laser polarization on edge spectral profiles

In this section, we delve into the influence of laser polarization on the edges through linescans to further investigate the effects observed in the previous maps. The goal is to gain insight into the stress induced during the etching process used in device manufacturing. Figure 26 presents intensity and Lorentzian FWHM values obtained from profiles perpendicular to an edge positioned at various angles relative to the laser polarization direction. The angles of orientation with respect to the laser polarization direction are (a) 0°, (b) 30°, (c) 60°, and (d) 90°. The nominal power of the laser used was 18 mW.

In Figure 26 (a), when the edge aligns parallel to the laser polarization direction, the intensity increases noticeably as it transitions from the NW layer (where intensity is lower) to the substrate. When the intensity is increasing, that is, at the edge, there is an increase in the Lorentzian FWHM, indicating a broader spectral response at the edge.

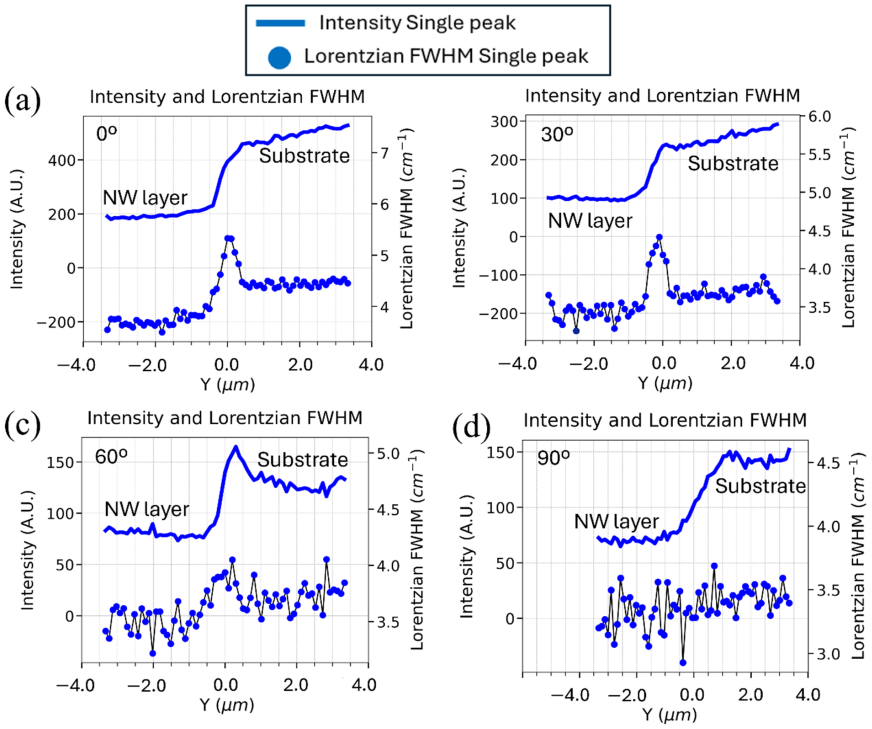

Figure 26: Representation of the intensity and the Lorentzian FWHM when making linescans perpendicular to an edge, forming different angles with the polarization axis: (a) 0°, (b) 30°, (c) 60°, and (d) 90°.

At 30° orientation with respect to the laser polarization (Figure 26 (b)), the intensity decreases both in the NW layer and on the substrate with respect to the previous case. This reduction in intensity is attributed to the dependence of Raman scattering on the polarization direction of

both incident and scattered light [46, 47, 48]. Besides, the Lorentzian FWHM shows a smaller variation compared to the previous orientation.

When the edge is set at 60° (Figure 26 (c)) relative to the incident beam's polarization direction, the intensity variation diminishes compared to previous angles. The Lorentzian FWHM increases slightly at the edge, remaining relatively constant at 3.5 cm^{-1} when the laser is on the substrate.

Finally, in Figure 26 (d), where the edge is perpendicular to the laser polarization direction, the intensity variation from the NW layer to the substrate resembles that at 60°. However, the Lorentzian FWHM remains constant throughout the linescan.

In conclusion, the polarization direction of the incident beam significantly influences the Lorentzian FWHM of Raman peaks at the edges of the sample, indicating a correlation between polarization and spectral response characteristics.

Figure 27 displays the Raman shift values along with the corresponding stress calculations derived from these shifts, for the same scattering geometries as employed in Fig. 26.

In Figure 27 (a), when the edge is parallel to the beam polarization direction, a noticeable decrease in the Raman shift is observed right at the edge, corresponding to the increased Lorentzian FWHM seen in Figure 26 (a). This effect, although less pronounced in the Raman shift compared to the Lorentzian FWHM, also occurs when the edge is oriented at 30° relative to the polarization direction. For the edge at 60°, there is minimal variation in the Raman shift compared to the rest of the sample. Lastly, when the edge is perpendicular to the polarization direction of the beam, the Raman shift remains nearly constant throughout the measurement.

We conclude, then, that both the Lorentzian FWHM and the Raman shift of the Raman peaks are influenced by the angle formed between the edge and the polarization direction of the laser beam.

Furthermore, using the position of the Raman peaks, it is possible to calculate the stress and temperature of the sample region where the spectra are obtained. The stress and temperature values associated with the Raman shift are depicted on the right vertical axes of Figure 27 and Figure 28, respectively.

In Figure 27 (a), a Raman shift difference between the NW layer and the substrate layer can be observed. This difference can be associated with strain generated in the NW layer due to the difference in the lattice constants between silicon and silicon oxide [49]. Although the difference in the Raman shift is very small and could fall within the measurement error of the equipment. It is consistently observed that the Raman shift in the NW layer and the substrate layer remains around 520.4 cm^{-1} and 520.3 cm^{-1}, respectively. This consistent difference can be attributed to the strain generated by the lattice constant mismatch between the NW layer and the substrate layer. According to the conversion from Raman shift to stress in Eq. (3.6), this difference corresponds to approximately 0.05 GPa.

This compression due to the lattice parameter difference between silicon and silicon oxide is illustrated in Figure 28 (b). Another effect to consider is temperature, which mainly affects the edge of the Si layer, Figure 28 (a), where the Raman shift of the peak decreases significantly compared to other regions. The thermal and stress effects are intertwined, so there may be some influence of stress at the edge. However, the temperature effect is not seen in the NW layer, as the temperature in the NW layer should be higher than that of the substrate, but the opposite is observed in this case.

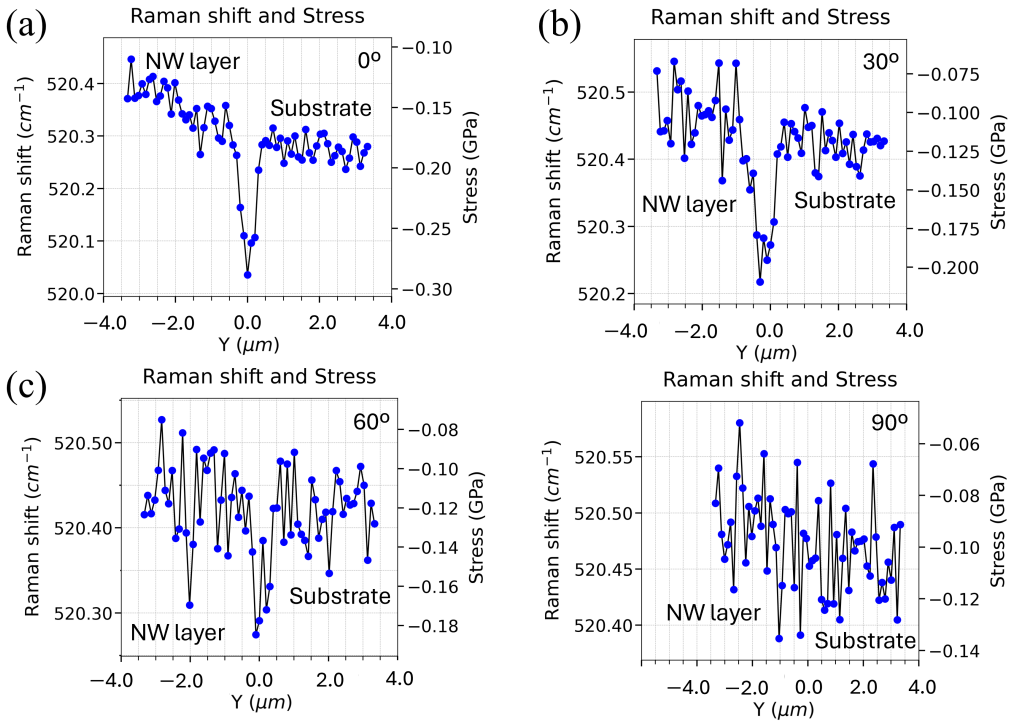

Figure 27: Representation of the Raman shift and stress by making linescans perpendicular to an edge, forming different angles with the polarization axis: (a) 0°, (b) 30°, (c) 60°, and (d) 90°.

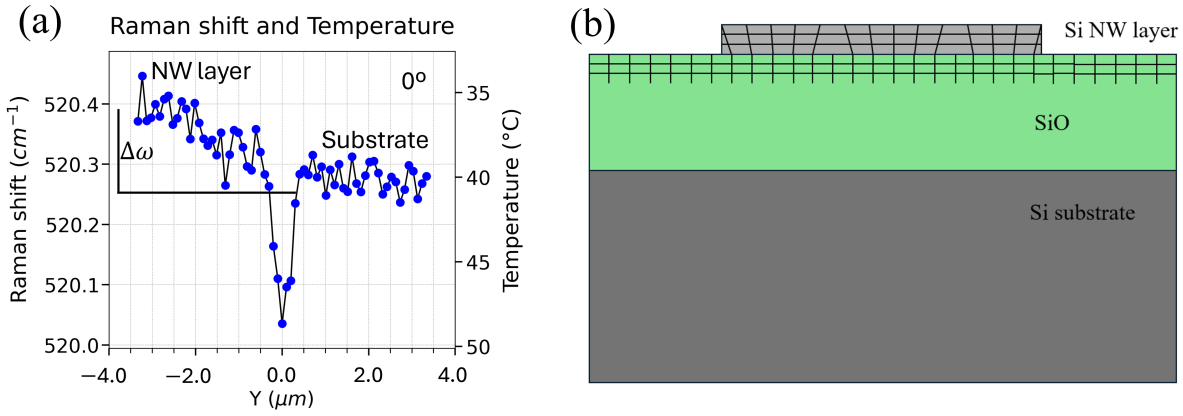

Figure 28: (a)Representation of the temperature converted, spectral profile made perpendicular to an edge forming (a) 0° with the polarization axis. (b) Sketch illustrating the strain caused in the NW layer, due to the lattice mismatch between the NW and thermal silicon oxide layers.

4 Conclusion and further work

In conclusion, this study employed Raman spectroscopy, pushing the technique to its resolution limits, to characterize silicon nanowires (NWs) for qubit fabrication. This non-destructive technique enables quality control of nanoscale devices without causing damage.

To manage the extensive datasets generated by hyperspectral maps, a Python code was developed. This code employs a statistical criterion with specific constraints to determine when a

double-peak fitting is necessary, enhancing the accuracy of the fit compared to a single-peak adjustment and avoiding overfitting.

The analysis of polycrystalline and monocrystalline silicon samples using a 638 nm laser revealed an unusual effect in the Raman shift when the sample was positioned at different angles relative to the measuring instrument. Initially, it seemed that stress was obtained from the Raman shift, but rotating the sample caused the Raman shift to change, indicating that the effect is not inherent to the sample but due to an unidentified external factor. This effect did not occur with the 532 nm laser, suggesting that further investigation is needed.

In the analysis of the monocrystalline silicon sample using the 532 nm laser, the NW layer was differentiated from the substrate layer by heating. We developed a method to focus on the NW layer before automatically identifying the maximum intensity, with the autofocus of the equipment, which is in the substrate layer. Then heating the NW layer, which occurs when it is parallel to the laser polarization, we dentified the NW layer due to the two-peak fitting criteria. Additionally, when the NW is perpendicular to the laser polarization, two intensity peaks were observed, corresponding to the distance between the substrate and the NW layer. This spectral differentiation enabled the acquisition of linescans parallel and perpendicular to the NW with the NW parallel and perpendicular to the laser polarization direction, revealing differences in the information obtained based on the focus layer.

Furthermore, the mapping allowed us to study the heating of the NW as a function of the laser polarization, due to its poor energy dissipation, which had already been previously studied. However, we also observed variations in the Lorentzian FWHM and the Raman shift at the sample edges, dependent on polarization. This was analyzed in detail by varying the position of an edge relative to the laser polarization. Through this, we examined the influence of polarization at the edges to identify regions of heating or stress generation, which will be explored further in future studies.

This work revealed a Raman shift difference between the NW layer and the substrate, indicating that the stress in the NW layer is due to the lattice parameter mismatch between the Si layer of the NW and the SiO layer of the substrate, ruling out heating effects that primarily affect the edges.

In the future, the development of this type of device analysis may be crucial for the advancement and production of high-quality qubits and nanoscale electronic devices.

References

[1] R. P. Feynman, "Simulating physics with computers," in *Feynman and computation*, pp. 133–153, CRC Press, 2018.

[2] P. Benioff, "The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines," *Journal of Statistical Physics*, vol. 22, pp. 563–591, 1980.

[3] V. Lordi and J. M. Nichol, "Advances and opportunities in materials science for scalable quantum computing," *MRS Bulletin*, vol. 46, pp. 589–595, 2021.

[4] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, pp. 369–395, 2020.

[5] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, 2019.

[6] L. M. Vandersypen and M. A. Eriksson, "Quantum computing with semiconductor spins," *Physics Today*, vol. 72, no. 8, pp. 38–45, 2019.

[7] M. Saffman, "Quantum computing with atomic qubits and Rydberg interactions: Progress and challenges," *Journal of Physics B: Atomic, Molecular and Optical Physics*, vol. 49, no. 20, p. 202001, 2016.

[8] V. Dobrovitski, G. Fuchs, A. Falk, C. Santori, and D. Awschalom, "Quantum control over single spins in diamond," *Annu. Rev. Condens. Matter Phys.*, vol. 4, no. 1, pp. 23–50, 2013.

[9] L. M. Vandersypen and I. L. Chuang, "NMR techniques for quantum control and computation," *Reviews of Modern Physics*, vol. 76, no. 4, pp. 1037–1069, 2004.

[10] F. Flamini, N. Spagnolo, and F. Sciarrino, "Photonic quantum information processing: A review," *Reports on Progress in Physics*, vol. 82, no. 1, p. 016001, 2018.

[11] S. Nadj-Perge, S. Frolov, E. Bakkers, and L. P. Kouwenhoven, "Spin–orbit qubit in a semiconductor nanowire," *Nature*, vol. 468, no. 7327, pp. 1084–1087, 2010.

[12] S. M. Frolov, S. R. Plissard, S. Nadj-Perge, L. P. Kouwenhoven, and E. P. Bakkers, "Quantum computing based on semiconductor nanowires," *MRS bulletin*, vol. 38, no. 10, pp. 809–815, 2013.

[13] D. Long, *"A unified treatment of the theory of Raman scattering by molecules"*. Chichester John Wiley & Sons, 2002.

[14] J. A. Calvo, *"Transporte térmico y caracterización Raman de nanohilos semiconductores de silicio y silicio-germanio"*. PhD thesis, Universidad de Valladolid, 2013.

[15] J. L. Pura Ruiz, *"Interaction between coherent light and axially heterostructured semiconductor nanowires"*. PhD thesis, Universidad de Valladolid, 2019.

[16] M. Cardona and G. Güthertodt, *"Light Scattering in Solids II"*, vol. 50 of *Topics in Applied Physics*. Springer, Berlin, 1982.

[17] G. S. Doerk, C. Carraro, and R. Maboudian, "Raman spectroscopy for characterization of semiconducting nanowires," *Springer*, pp. 477–506, 2012.

[18] R. Loudon, "The raman effect in crystals," *Advances in Physics*, vol. 13, no. 52, pp. 423–482, 1964.

[19] I. De Wolf, "Micro-Raman spectroscopy to study local mechanical stress in silicon integrated circuits," *Semiconductor science and technology*, vol. 11, no. 2, p. 139, 1996.

[20] Y. Peter and M. Cardona, *"Fundamentals of semiconductors: Physics and materials properties"*. Springer Science & Business Media, 2010.

[21] W. Demtröder, *Laser Spectroscopy*, vol. 2. Springer, 1973.

[22] T. Hart, R. Aggarwal, and B. Lax, "Temperature dependence of Raman scattering in silicon," *Physical Review B*, vol. 1, no. 2, p. 638, 1970.

[23] I. D. Wolf, "Stress measurements in Si microelectronics devices using Raman spectroscopy," *Journal of Raman spectroscopy*, vol. 30, no. 10, pp. 877–883, 1999.

[24] C. Kittel and P. McEuen, *Introduction to Solid State Physics*. John Wiley & Sons, 2018.

[25] P. Yu, J. Wu, S. Liu, J. Xiong, C. Jagadish, and Z. M. Wang, "Design and fabrication of silicon nanowires towards efficient solar cells," *Nano Today*, vol. 11, no. 6, pp. 704–737, 2016.

[26] J. M. Weisse, C. H. Lee, D. R. Kim, and X. Zheng, "Fabrication of flexible and vertical silicon nanowire electronics," *Nano letters*, vol. 12, no. 6, pp. 3339–3343, 2012.

[27] G. Mueller, "The Czochralski method-where we are 90 years after Jan Czochralski's invention," *Crystal Research and Technology: Journal of Experimental and Industrial Crystallography*, vol. 42, no. 12, pp. 1150–1161, 2007.

[28] F. O. Sequeda, "Integrated circuit fabrication—a process overview," *JOM*, vol. 37, no. 5, pp. 43–50, 1985.

[29] A. N. Broers, A. Hoole, and J. M. Ryan, "Electron beam lithography—resolution limits," *Microelectronic Engineering*, vol. 32, no. 1-4, pp. 131–142, 1996.

[30] Y. Zheng, P.-p. Gao, X. Tang, J.-z. Liu, and J.-a. Duan, "Effects of electron beam lithography process parameters on structure of silicon optical waveguide based on SOI," *Journal of Central South University*, vol. 29, no. 10, pp. 3335–3345, 2022.

[31] J. Casas Pelaez, *Óptica*. Zaragoza: Libreria Pons, 7ª ed., 2ª reimp. ed., 2008.

[32] K. R. Spring, T. J. Fellers, and M. W. Davidson, "Resolution and contrast in confocal microscopy," *National High Magnetic Field Laboratory*, 2006.

[33] G. Cox and C. J. Sheppard, "Practical limits of resolution in confocal and non-linear microscopy," *Microscopy Research and Technique*, vol. 63, no. 1, pp. 18–22, 2004.

[34] M. Newville, T. Stensitzki, D. B. Allen, M. Rawlik, A. Ingargiola, and A. Nelson, "Lmfit: Non-linear least-square minimization and curve-fitting for python," *Astrophysics Source Code Library*, pp. ascl–1606, 2016.

[35] T. pandas development team, "pandas-dev/pandas: Pandas," Dec. 2023.

[36] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[37] S. Schippers, "Analytical expression for the convolution of a Fano line profile with a gaussian," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 219, pp. 33–36, 2018.

[38] M. R. Zaghloul and A. N. Ali, "Algorithm 916: computing the Faddeyeva and Voigt functions," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 2, pp. 1–22, 2012.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[40] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 03, pp. 90–95, 2007.

[41] C. Schinke, P. Christian Peest, J. Schmidt, R. Brendel, K. Bothe, M. R. Vogt, I. Kröger, S. Winter, A. Schirmacher, S. Lim, *et al.*, "Uncertainty analysis for the coefficient of band-to-band absorption of crystalline silicon," *AIP Advances*, vol. 5, no. 6, 2015.

[42] M. N. Polyanskiy, "Refractiveindex. info database of optical constants," *Scientific Data*, vol. 11, no. 1, p. 94, 2024.

[43] I. De Wolf, H. Maes, and S. K. Jones, "Stress measurements in silicon devices through raman spectroscopy: Bridging the gap between theory and experiment," *Journal of Applied Physics*, vol. 79, no. 9, pp. 7148–7156, 1996.

[44] R. Tsu and J. G. Hernandez, "Temperature dependence of silicon Raman lines," *Applied Physics Letters*, vol. 41, no. 11, pp. 1016–1018, 1982.

[45] X. Li, S. Jin, R. Zhang, Y. Gao, Z. Liu, Y. Yao, Y. Wang, X. Wang, Y. Zhang, and X. Tao, "The resolution and repeatability of stress measurement by raman and EBSD in silicon," *Vacuum*, vol. 203, p. 111276, 2022.

[46] S. Nakashima and M. Hangyo, "Characterization of semiconductor materials by raman microprobe," *IEEE Journal of Quantum Electronics*, vol. 25, no. 5, pp. 965–975, 1989.

[47] K. Mizoguchi and S.-i. Nakashima, "Determination of crystallographic orientations in silicon films by raman-microprobe polarization measurements," *Journal of Applied Physics*, vol. 65, no. 7, pp. 2583–2590, 1989.

[48] Z. Lu, T. Quinn, and H. Reehal, "Polarization-dependent Raman spectra of thin crystalline silicon films," *Journal of Applied Physics*, vol. 97, no. 3, 2005.

[49] N. Daldosso, M. Luppi, S. Ossicini, E. Degoli, R. Magri, G. Dalba, P. Fornasini, R. Grisenti, F. Rocca, L. Pavesi, *et al.*, "Role of the interface region on the optoelectronic properties of silicon nanocrystals embedded in SiO 2," *Physical Review B*, vol. 68, no. 8, p. 085327, 2003.

Appendix

Polycrystalline Si sample maps with red laser

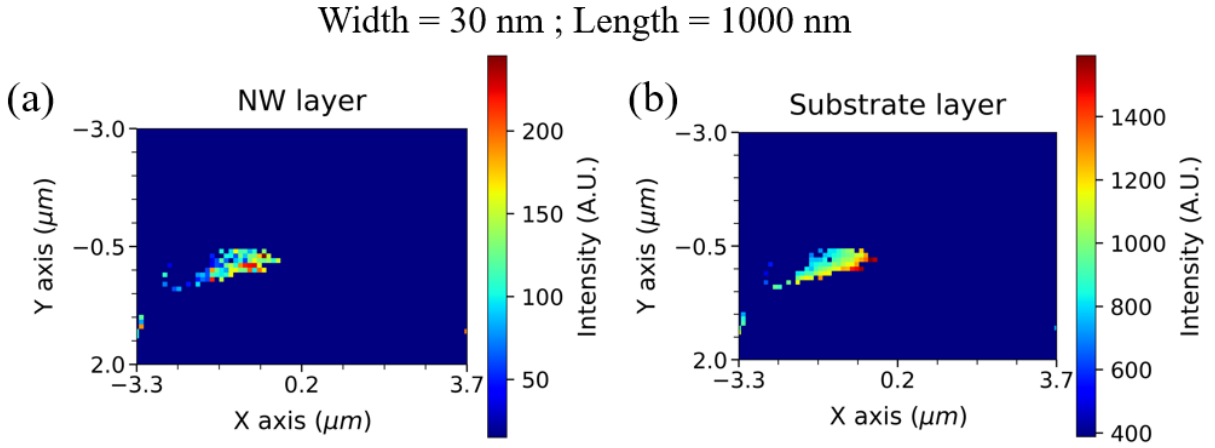

Figure 29: Representation of the intensity of the NW layer peak (a) and the substrate layer peak (b) for the two peak fitting.

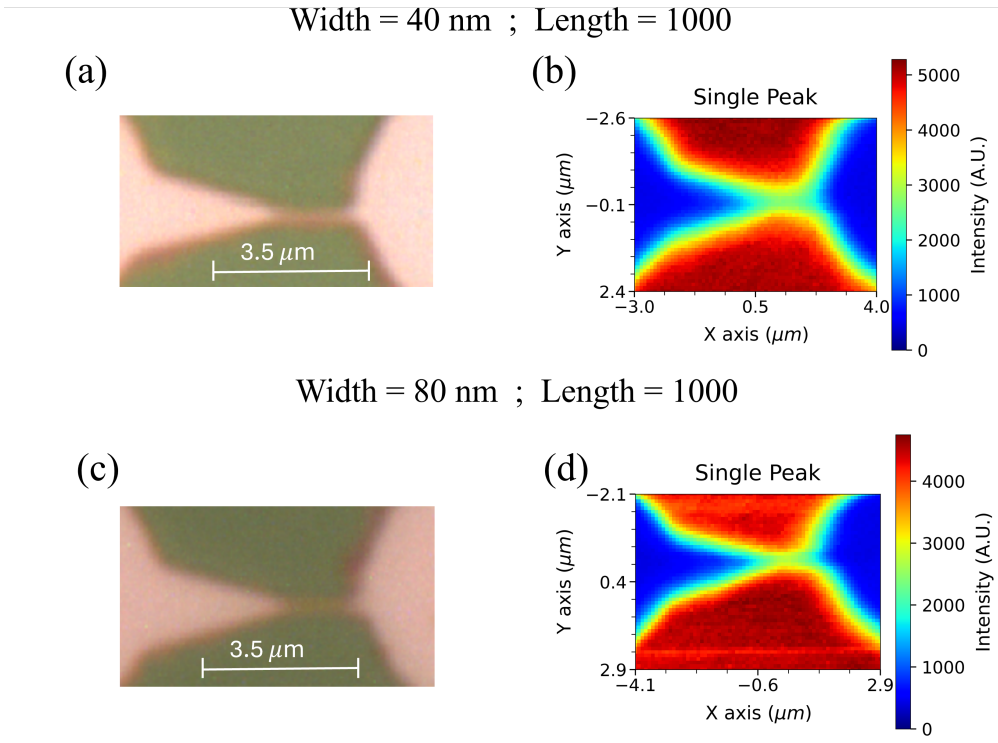

Figure 30: Optical images of NWs with widths of 40 nm (a) and 80 nm (c), alongside intensity maps represented with a single peak fit for NWs of 40 nm (b) and 80 nm (d)..

Python Code for Linescans

Listing 1: Python Code for Linescans

```
# -*- coding: utf-8 -*-
"""
@author: Gin s Gonz lez Guirado
```

```
"""

import os
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lmfit import Parameters, Minimizer, fit_report
from scipy.special import wofz
from matplotlib.ticker import AutoMinorLocator, MultipleLocator
from scipy.ndimage import gaussian_filter
from scipy.optimize import fsolve

# Function to open the file
def open_file(txt_file):
    try:
        data = pd.read_table(txt_file, delim_whitespace=True, header=None)
        print(data)
        return data
    except Exception as e:
        print(f"Error opening file: {e}")
        return None

def extract_number_before_s(filename):
    # Usar regex para encontrar el n mero antes de 's'
    match = re.search(r'(\d*\.?\d+)s', filename)
    if match:
        return float(match.group(1))
    else:
        return 1  # Valor por defecto si no se encuentra n mero


# Define Voigt function using Faddeeva function
def voigt_f_not_normalised(x, x0, g_FWHM, l_FWHM):
    """
    Calculate the Voigt function.

    The Voigt function represents the convolution of a Gaussian and a Lorentzian
        function.

    Parameters:
        x (array-like): The input variable.
        x0 (float): The center of the Voigt function.
        g_FWHM (float): The full width at half maximum (FWHM) of the Gaussian
            component.
        l_FWHM (float): The full width at half maximum (FWHM) of the Lorentzian
            component.

    Returns:
        array-like: The Voigt function evaluated at x.

    Notes:
        The Voigt function is computed using the Faddeeva function (wofz) from
            the scipy library.

    References:
        - Mofreh R. Zaghloul and Ahmed N. Ali. (2011). Algorithm 916: Computing
            the Faddeyeva and Voigt Functions. ACM Transactions on Mathematical
```

```
            Software, Volume 38, Issue 2 Article No.: 15 pp 1  22  https://doi.
            org/10.1145/2049673.2049679
        - S. Schippers Analytical expression for the convolution of a Fano line
            profile with a gaussian,/ Journal of Quantitative Spectroscopy &
            Radiative Transfer 219 (2018) 33  36  , https://doi.org/10.1016/j.
            jqsrt.2018.08.003

    """

    x_i = x - x0
    alpha = np.maximum(g_FWHM / 2,1e-4)
    gamma =np.maximum(l_FWHM / 2,1e-4)
    sigma = alpha / np.sqrt(2 * np.log(2))

    # Calculate the complex argument for the Faddeeva function
    z = (x_i + 1j * gamma) / sigma / np.sqrt(2)

    # Evaluate the Faddeeva function to compute the Voigt function
    faddeeva = wofz(z)

    # Compute the real part of the Faddeeva function and normalize by the
        standard deviation
    voigt = np.real(faddeeva) / sigma / np.sqrt(2 * np.pi)

    return voigt

def voigt_f(x, x0, g_FWHM, l_FWHM, A):
    """
    Calculate the normalized Voigt function by normalizing the peak.

    Parameters:
        x (array-like): The input variable.
        x0 (float): The center of the Voigt function.
        g_FWHM (float): The full width at half maximum (FWHM) of the Gaussian
            component.
        l_FWHM (float): The full width at half maximum (FWHM) of the Lorentzian
            component.
        amplitude (float): The amplitude of the lineshape.
    Returns:
        array-like: The normalized Voigt function evaluated at x.

    """

    # Calculate the unnormalized Voigt function
    voigt = voigt_f_not_normalised(x, x0, g_FWHM, l_FWHM)

    # Find the maximum value of the Voigt function at x0
    max_value = np.maximum(voigt_f_not_normalised(x0, x0, g_FWHM, l_FWHM),1e-20)

    # Normalize the Voigt function by dividing by the maximum value
    voigt_normalized = voigt / max_value

    return voigt_normalized*A


# Function to create the intensities map
def intensities_linescan(data_to_fit, filename):
    data = data_to_fit
```

```
107
108     # Calculate the maximum value of the intensity for each pixel (excluding NaN
            values)
109     max_values = np.nanmax(data.iloc[1:, 1:].values, axis=1)
110
111     # Extract x coordinates
112     x = [float(value) for value in data.iloc[1:, 0].values]
113
114     # Create a figure and axis
115     fig, ax = plt.subplots(figsize=(10, 8))
116
117     # Set the limits of the axes based on data range
118     x_min, x_max = np.min(x), np.max(x)
119     intensity_min, intensity_max = np.min(max_values), np.max(max_values)
120
121     # To establish the major locators
122     x_locator = abs(x_max - x_min)/2
123     intensity_locator = abs(intensity_max - intensity_min)/2
124
125     # Limits of the axes
126     ax.set_xlim(x_min, x_max)
127     ax.set_ylim(intensity_min, intensity_max)
128
129     # Set the size of major ticks
130     ax.tick_params(axis="x", labelsize=10, which="major", length=5, width=1)
131     ax.tick_params(axis="y", labelsize=10, which="major", length=5, width=1)
132     # Set the size of minor ticks
133     ax.tick_params(axis="x", labelsize=8, which="minor", length=2.5, width=0.5)
134     ax.tick_params(axis="y", labelsize=8, which="minor", length=2.5, width=0.5)
135
136     # Set major and minor tick locators based on subplot
137     ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major ticks
138     ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
139     ax.yaxis.set_major_locator(MultipleLocator(intensity_locator))  # Set major
            ticks
140     ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
141
142     # Set the minimum and maximum major tick locations for x and y axes
143     ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=False)
144     ax.set_yticks(np.arange(intensity_min, intensity_max + intensity_locator,
            intensity_locator), minor=False)
145
146     # Plot the intensity values using a line plot
147     ax.plot(x, max_values, color='blue', linestyle='-')
148
149     # Add labels based on the filename
150     if 'Z_linescan' in filename:
151         ax.set_xlabel('Depth ($\mu m$)')
152     elif 'Y_linescan' in filename:
153         ax.set_xlabel('Y ($\mu m$)')
154     elif 'X_linescan' in filename:
155         ax.set_xlabel('X ($\mu m$)')
156     else:
157         ax.set_xlabel('X axis ($\mu m$)')
158
159     ax.set_ylabel('Intensity (counts)')
```

```
    ax.set_title('Profile of the Maximum Intensity of the Raman spectra')

    # Show the plot
    plt.show()


# Function to ask for the number of peaks and the model type to adjust them
def ask_for_peaks():
    num_peaks = int(input("Enter the number of peaks: "))

    peaks = []

    for i in range(num_peaks):
        print(f"\nEntering parameters for peak {i+1}:")

        x0_initial = float(input("Enter initial value for peak center: "))
        Gauss_FWHM_initial = float(input("Enter initial value for FWHM Gaussian:
             "))
        Lorentz_FWHM_initial = float(input("Enter initial value for FWHM
            Lorentzian: "))
        I_initial = float(input("Enter initial value for intensity: "))
        peaks.append([x0_initial, I_initial, Gauss_FWHM_initial,
            Lorentz_FWHM_initial])

    return peaks


def model_f(params, x, peaks):
    """
    Calculate the simple model function consisting of set of peak functions
        without baseline.

    Parameters:
        params (lmfit.Parameters): The parameters object containing the fitting
            parameters.
        x (array-like): The input variable.
        peaks (list): The list of peak positions, and intensities.
        model_type (list, optional): The type of peak model to use for each peak
            , the array must have the same size as peaks.
                                    Options are 'Gaussian', 'Lorentz', 'Gauss-
                                        Lorentz', 'Voigt',
                                    'Fano-Simply', and 'Fano-Full'. Default is '
                                        all Gaussian'.

    Returns:
        array-like: The calculated model function evaluated at x.
    """


    function_composed=[]

    for  item in range(len(peaks)):
        # Load the parameters for the peaks
        function_composed.append(voigt_f(x,
                                    params['Peak_'+str(item+1)+'_Center'],
                                    params['Peak_'+str(item+1)+'_Gauss_FWHM'],
                                    params['Peak_'+str(item+1)+'_Lorentz_FWHM'
                                        ],
```

```
                                        params['Peak_'+str(item+1)+'_Intensity'])
                                        )

    peak_term=np.sum(function_composed, axis=0)

    return peak_term


# Function to define the parameters for the fit
def params_to_fit(peaks):
    """
   Set up the parameters for a fit model.

   Args:
       peaks (list): List of peak positions and intensities.

   Returns:
       params (Parameters): Parameters object for the fit.
   """
    # Set up the parameters for the fit
    params = Parameters()

    for item in range(len(peaks)):
        # Load the parameters for the peaks
        params.add('Peak_'+str(item+1)+'_Center', value=peaks[item][0], min=400, max=700)
        params.add('Peak_'+str(item+1)+'_Gauss_FWHM', value=peaks[item][2], min=1, max=1.001)
        params.add('Peak_'+str(item+1)+'_Lorentz_FWHM', value=peaks[item][3], min=1, max=50)
        params.add('Peak_'+str(item+1)+'_Intensity', value=peaks[item][1], min=30)

    return params

def residual(params, position_frequency_data, data=None, peaks=None):
    """
   Objective function for fitting a model to data.

   Parameters:
       params (lmfit.Parameters): Model parameters to be optimized.
       x (array-like): Independent variable data.
       data (array-like): Dependent variable data to fit the model to.
       peaks (list): List of peak positions and intensities.
       model_type (list, optional): The type of peak model to use for each peak, the array must have the same size as peaks.
                                    Options are 'Gaussian', 'Lorentz', 'Gauss-Lorentz', 'Voigt',
                                    'Fano-Simply', and 'Fano-Full'. Default is 'all Gaussian'.

   Returns:
       array-like: Difference between the model values and the data.
   """
    x = position_frequency_data

    model_values = model_f(params, x, peaks)

```

```
260
261     return model_values - data
262
263 # Fit data with the Voigt function using lmfit
264 def fit(data_dict, peaks):
265     fit_params_dict = {}
266     fit_results_dict = {}
267     r_squared_dict = {}
268     double_peak_params_dict_1 = {}
269     double_peak_params_dict_2 = {}
270     r_squared_double_peak_dict_1 = {}
271     r_squared_double_peak_dict_2 = {}
272     r_squared_double_peak_dict = {}
273     modified_double_peak_params_dict_1 = {}
274     fit_results_double_peak_1_dict = {}
275     fit_results_double_peak_2_dict = {}
276
277     for var_name, data_to_fit in data_dict.items():
278         fit_params_list = []
279         fit_results = []  # Store fit results
280         r_squared_list = []  # Store r^2 values for each spectrum
281
282         filename = f'{var_name}'
283
284         # Extract x and y coordinates
285         x = data_to_fit.iloc[1:, 0].values.astype(float)
286
287         # Range of frequencies in the spectrum and intensities of each spectrum
288         position_frequency_data = data_to_fit.iloc[0, 1:].values.astype(float)
289         intensity_data = data_to_fit.iloc[1:, 1:].values.astype(float)
290
291         # Adjust single peak model first
292         single_peak = [peaks[0]]
293
294         for j in range(len(x)):
295             # Eliminate NaN values before fitting
296             y_fit = intensity_data[j, :]
297
298             pars = params_to_fit(single_peak)
299
300             minimizer = Minimizer(residual, pars, fcn_args=(
                    position_frequency_data, y_fit, single_peak))
301             result = minimizer.least_squares(**{'xtol': 1e-5,
302                                                 'gtol': 1e-5,
303                                                 'ftol':1e-5,
304                                                 'max_nfev':1e6})
305
306             # Calculate r^2
307             ss_residual = np.sum(result.residual ** 2)
308             ss_total = np.sum((y_fit - np.mean(y_fit)) ** 2)
309             r_squared = 1 - (ss_residual / ss_total)
310
311             # Append the fit result
312             fit_results.append(result)
313
314             # Append the r^2 value
315             r_squared_list.append(r_squared)
316
```

```
            # Extract the fitted parameters and store them in a list
            fit_params = [result.params[param].value for param in result.params]
            fit_params_list.append(fit_params)

        fit_pars_array = np.array(fit_params_list)

        # Convert r_squared_list to array
        r_squared_array = np.array(r_squared_list)

        # Print the array of fit parameters
        print("Array of Fit Parameters:")
        print(fit_pars_array)
        print("Array of r^2 values:")
        print(r_squared_array)

        # Extract parameters
        l_FWHM = fit_pars_array[:, 2]
        l_FWHM_min = np.min(l_FWHM)

        x0 = fit_pars_array[:, 0]
        x0_min = np.min(x0)

        A = fit_pars_array[:, 3]
        A_min = np.min(A)

        def params_to_fit2(peaks, lorentz_fwhm_bounds=None, x0_min=None, A_min=
            None):
            params = Parameters()

            for item in range(len(peaks)):
                if x0_min and item == 1:
                    params.add(f'Peak_{item+1}_Center', value=peaks[item][0],
                        min=400, max=x0_min)
                else:
                    params.add(f'Peak_{item+1}_Center', value=peaks[item][0],
                        min=x0_min, max=700)

                params.add(f'Peak_{item+1}_Gauss_FWHM', value=peaks[item][2],
                    min=1, max=1.001)

                if lorentz_fwhm_bounds and item == 0:
                    min_lorentz = lorentz_fwhm_bounds[0]
                    max_lorentz = lorentz_fwhm_bounds[1]
                    params.add(f'Peak_{item+1}_Lorentz_FWHM', value=peaks[item
                        ][3], min=min_lorentz, max=max_lorentz)
                else:
                    params.add(f'Peak_{item+1}_Lorentz_FWHM', value=peaks[item
                        ][3], min=(lorentz_fwhm_bounds[0]/0.85), max=50)

                if A_min and item == 1:
                    params.add(f'Peak_{item+1}_Intensity', value=peaks[item][1],
                         min=30, max='0.8 * Peak_1_Intensity')
                else:
                    params.add(f'Peak_{item+1}_Intensity', value=peaks[item][1],
                         min=30)

            return params
```

```
        # Calculate bounds for Lorentz_FWHM for the first peak in double peak fit

        min_lorentz = l_FWHM_min
        max_lorentz = l_FWHM_min * 3
        lorentz_fwhm_bounds = (min_lorentz, max_lorentz)

        # Identify spectra that need a double peak fit
        if np.any(r_squared_array < 0.99):
            r_squared_min = np.min(r_squared_array)
            r_squared_max = np.max(r_squared_array)
            r_squared_diff = r_squared_max - r_squared_min
            threshold = r_squared_min + r_squared_diff * 0.9

            # Identify indices that meet both conditions
            double_peak_indices = np.where(
                (r_squared_array < threshold) &  # Condici n 1: Menor que el umbral
                (r_squared_array < 0.99) &       # Condici n 2: Menor que 0.99
                (r_squared_array > 0.83)         # Condici n 3: Mayor que 0.83
                )[0]
        else:
            double_peak_indices = []
        print(double_peak_indices)


        double_peak_params_list_1 = np.zeros_like(fit_pars_array)
        double_peak_params_list_2 = np.zeros_like(fit_pars_array)
        r_squared_double_peak_list_1 = np.zeros(len(x))
        r_squared_double_peak_list_2 = np.zeros(len(x))
        r_squared_double_peak_list = np.zeros(len(x))

        fit_results_double_peak_1 = []  # Store fit results of Peak 1
        fit_results_double_peak_2 = []  # Store fit results of Peak 2
        fit_results_double_peak = []  # Store fit results of double Peak

        for idx in double_peak_indices:
            y_fit = intensity_data[idx, :]

            pars = params_to_fit2(peaks, lorentz_fwhm_bounds, x0_min)

            minimizer = Minimizer(residual, pars, fcn_args=(
                position_frequency_data, y_fit, peaks))
            result = minimizer.least_squares(**{'xtol': 1e-5,
                                        'gtol': 1e-5,
                                        'ftol':1e-5,
                                        'max_nfev':1e6})

            # Calculate r^2 for double peak fit
            ss_residual = np.sum(result.residual ** 2)
            ss_total = np.sum((y_fit - np.mean(y_fit)) ** 2)
            r_squared = 1 - (ss_residual / ss_total)

            r_squared_double_peak_list[idx] = r_squared

            fit_results_double_peak.append(result)  # Append the fit result for Double Peak

```

```
            fit_params = [result.params[param].value for param in result.params]

            # Update single peak fit array
            double_peak_params_list_1[idx] = fit_params[:len(fit_params)//2]

            # Update double peak fit array
            double_peak_params_list_2[idx] = fit_params[len(fit_params)//2:]

            # Convert lists to dictionaries for model_f
            params_dict_1 = {f'Peak_{i+1}_{param}': double_peak_params_list_1[
                idx][j]
                             for i in range(len(peaks))
                             for j, param in enumerate(['Center', 'Gauss_FWHM',
                                 'Lorentz_FWHM', 'Intensity'])}

            params_dict_2 = {f'Peak_{i+1}_{param}': double_peak_params_list_2[
                idx][j]
                             for i in range(len(peaks))
                             for j, param in enumerate(['Center', 'Gauss_FWHM',
                                 'Lorentz_FWHM', 'Intensity'])}

            # Calculate r^2 for each peak in the double peak fit
            ss_residual_1 = np.sum((y_fit - model_f(params_dict_1,
                position_frequency_data, [peaks[0]])).flatten() ** 2)
            ss_residual_2 = np.sum((y_fit - model_f(params_dict_2,
                position_frequency_data, [peaks[1]])).flatten() ** 2)

            r_squared_1 = 1 - (ss_residual_1 / ss_total)
            r_squared_2 = 1 - (ss_residual_2 / ss_total)

            r_squared_double_peak_list_1[idx] = r_squared_1
            r_squared_double_peak_list_2[idx] = r_squared_2

            fit_results_double_peak_1.append(result)  # Append the fit result
                for Peak 1
            fit_results_double_peak_2.append(result)  # Append the fit result
                for Peak 2

        double_peak_params_array_1 = np.array(double_peak_params_list_1)
        double_peak_params_array_2 = np.array(double_peak_params_list_2)

        # Convert r_squared_double_peak_list to array
        r_squared_double_peak_array_1 = np.array(r_squared_double_peak_list_1)
        r_squared_double_peak_array_2 = np.array(r_squared_double_peak_list_2)

        r_squared_double_peak_array = np.array(r_squared_double_peak_list)

        # Check intensity of the second peak and set rows to zero if intensity <
            35
        intensity_threshold = 35
        invalid_indices = double_peak_params_array_2[:, 3] < intensity_threshold

        double_peak_params_array_1[invalid_indices] = 0
        double_peak_params_array_2[invalid_indices] = 0
        r_squared_double_peak_array_1[invalid_indices] = 0
        r_squared_double_peak_array_2[invalid_indices] = 0
        r_squared_double_peak_array[invalid_indices] = 0

```

```
        # Print the array of fit parameters
        print("Array of Fit Parameters Peak 1:")
        print(double_peak_params_array_1)
        print("Array of Fit Parameters Peak 2:")
        print(double_peak_params_array_2)
        print("Array of r^2 values Peak 1:")
        print(r_squared_double_peak_array_1)
        print("Array of r^2 values Peak 2:")
        print(r_squared_double_peak_array_2)
        print("Array of r^2 values Double Peak:")
        print(r_squared_double_peak_array)

        # Replace zeros in double_peak_params_array_1 with fit_pars_array values
        modified_double_peak_params_array_1 = np.where(
            double_peak_params_array_1 == 0, fit_pars_array,
            double_peak_params_array_1)

        # Print the modified array of fit parameters
        print("Modified Array of Fit Parameters Peak 1:")
        print(modified_double_peak_params_array_1)

        # Store results in dictionaries
        fit_params_dict[var_name] = fit_pars_array
        fit_results_dict[var_name] = fit_results
        r_squared_dict[var_name] = r_squared_array
        double_peak_params_dict_1[var_name] = double_peak_params_array_1
        double_peak_params_dict_2[var_name] = double_peak_params_array_2
        r_squared_double_peak_dict_1[var_name] = r_squared_double_peak_array_1
        r_squared_double_peak_dict_2[var_name] = r_squared_double_peak_array_2
        r_squared_double_peak_dict[var_name] = r_squared_double_peak_array
        modified_double_peak_params_dict_1[var_name] =
            modified_double_peak_params_array_1
        fit_results_double_peak_1_dict[var_name] = double_peak_params_array_1
        fit_results_double_peak_2_dict[var_name] = double_peak_params_array_2

    return (fit_params_dict, fit_results_dict, r_squared_dict,
        double_peak_params_dict_1,
            double_peak_params_dict_2, r_squared_double_peak_dict_1,
                r_squared_double_peak_dict_2,
            r_squared_double_peak_dict, fit_results_double_peak_1_dict,
                fit_results_double_peak_2_dict,
            modified_double_peak_params_dict_1)


# Report of the fitting
def fit_info(fit_results_dict):
    for var_name, fit_results in fit_results_dict.items():
        print(f"Fit Results for {var_name}:")
        for i, result in enumerate(fit_results):
            print(f"Fit Result for row {i+1}:")
            print(fit_report(result))
        print()

# Define a function to save a plot as a PNG file
def save_plot_as_png(fig, directory, filename, title_suffix):
    # Create the directory if it doesn't exist
    os.makedirs(directory, exist_ok=True)

```

```
    # Construct the full path for saving the PNG file
    full_path = os.path.join(directory, f"{filename}_{title_suffix}.png")

    # Save the figure as a PNG file
    fig.savefig(full_path, dpi=300, bbox_inches='tight')
    print(f"Plot saved as {full_path}")
    plt.close(fig)  # Close the figure to avoid displaying it

# Function to plot r_squared results of the fitting
def plot_r_squared_linescan(data_dict, r_squared_dict, title_suffix, directory, filenames=None):
    """
    Plots the R-squared profile along the X axis.

    Parameters:
        data_dict (dict): Dictionary containing data to fit.
        r_squared_dict (dict): Dictionary containing R-squared values.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
        directory (str): Directory to save the plots.
        filenames (dict, optional): Dictionary with filenames for each plot.
    """

    for var_name, data_to_fit in data_dict.items():
        r_squared_array = r_squared_dict[var_name]
        # Extract x coordinates
        x = data_to_fit.iloc[1:, 0].astype(float)

        # Create a figure and axis
        fig, ax = plt.subplots(figsize=(8, 6))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)

        # Find the min and max values for the color scale, ignoring values <= 0
        if np.any(r_squared_array > 0):
            r_squared_min = np.max(np.min(r_squared_array[r_squared_array > 0]), 0)
            r_squared_max = np.max(r_squared_array)  # Limit max to 1
            r_squared_diff = r_squared_max - r_squared_min
            r_squared_min_cero = np.max([r_squared_min - r_squared_diff * 0.1, 0])
            r_squared_max_cero = np.min([r_squared_max + r_squared_diff*0.1, 1])
        else:
            r_squared_min_cero = 0
            r_squared_max_cero = 0

        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1

        # To establish the major locators
        x_locator = abs(x_max_cero - x_min_cero) / 2
        r_squared_locator = abs(r_squared_max_cero - r_squared_min_cero) / 2

        # Ensure the locator step is positive
        if r_squared_locator <= 0:
            r_squared_locator = 0.1  # You can adjust this to a more suitable default value
```

```
576
577         # Limits of the axes
578         ax.set_xlim(x_min_cero, x_max_cero)
579         ax.set_ylim(r_squared_min_cero, r_squared_max_cero)
580
581         # Set the size of major ticks
582         ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
                =1.5)
583         ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
                =1.5)
584         # Set the size of minor ticks
585         ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
586         ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)
587
588         # Set major and minor tick locators based on subplot
589         ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
                ticks
590         ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
591         ax.yaxis.set_major_locator(MultipleLocator(r_squared_locator))  # Set
                major ticks
592         ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
593
594         # Set the minimum and maximum major tick locations for x and y axes
595         ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
                minor=False)
596         ax.set_yticks(np.arange(r_squared_min_cero, r_squared_max_cero +
                r_squared_locator, r_squared_locator), minor=False)
597
598         # Plot the r_squared values
599         ax.plot(x, r_squared_array, color='blue', linestyle='-', marker='o',
                markersize=5)
600         ax.set_title(f'{title_suffix} Coefficient of Determination ($r^2$)',
                fontsize=22, pad=18)
601
602         # Update x-axis label based on filename
603         filename = f'{var_name}'
604
605         if 'Z_linescan' in filename:
606             ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
607         elif 'Y_linescan' in filename:
608             ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
609         elif 'X_linescan' in filename:
610             ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
611         else:
612             ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)
613
614         ax.set_ylabel('Coefficient of Determination ($r^2$)', fontsize=20,
                labelpad=13)
615
616         # Invert x axis
617         #ax.invert_xaxis()
618
619         # Add grid
620         ax.grid(True, which='both', linestyle='--', linewidth=0.5)
621
622         # Adjust layout
```

```
623         plt.tight_layout()
624
625         # Save plot as PNG if filenames are provided
626         if filenames:
627             save_plot_as_png(fig, directory, filename, title_suffix)
628
629
630
631 def plot_r_squared_linescan_all_params(data_dict, params_list, title_suffixes,
        colors, directory, filenames=None):
632     """
633     Transforms the map of positions (frequencies) into a stress map and plots it
            .
634
635     Parameters:
636         data_dict (dict): Dictionary of data sets to fit.
637         params_list (list): List of dictionaries of parameters arrays to plot.
638         title_suffixes (list): List of suffixes for the plot titles to
                differentiate plots.
639         colors (list): List of colors for each set of parameters.
640         directory (str): Directory where the plots will be saved.
641         filenames (dict, optional): Dictionary of filenames for saving the plots
                . Defaults to None.
642     """
643     def save_plot_as_png(fig, directory, filename):
644         """
645         Saves the plot as a PNG file.
646
647         Parameters:
648             fig (matplotlib.figure.Figure): Figure object to save.
649             directory (str): Directory where the plot will be saved.
650             filename (str): Filename for the saved plot.
651         """
652         os.makedirs(directory, exist_ok=True)
653         full_path = os.path.join(directory, f"{filename}.png")
654         fig.savefig(full_path, dpi=300, bbox_inches='tight')
655         plt.close(fig)
656         print(f"Plot saved as {full_path}")
657
658     def plot_parameter(ax, x, y, color, label, y_min, y_max):
659         mask = (y != 0) & (y > y_min) & (y < y_max)  # Create a mask within
                limits
660         ax.plot(x[mask], y[mask], color=color, linestyle='-', marker='o',
                markersize=5, label=label)
661         ax.plot(x[~mask], y[~mask], color=color, linestyle='', marker='o',
                markersize=5)  # Plot points outside the limits without connecting
                them
662
663     for var_name, data_to_fit in data_dict.items():
664         x = data_to_fit.iloc[1:, 0].astype(float)
665         x_min, x_max = np.min(x), np.max(x)
666         x_diff = x_max - x_min
667         x_min_cero = x_min - x_diff * 0.1
668         x_max_cero = x_max + x_diff * 0.1
669         x_locator = abs(x_max_cero - x_min_cero) / 2
670
671         r_squared_limits = [float('inf'), float('-inf')]
672
```

```
        for param_dict in params_list:
            r_squared_array = param_dict[var_name]
            if np.any(r_squared_array > 0):
                r_squared_min = np.max(np.min(r_squared_array[r_squared_array > 0]), 0)
                r_squared_max = np.max(r_squared_array)  # Limit max to 1
                r_squared_diff = r_squared_max - r_squared_min
                r_squared_min_cero = np.max([r_squared_min - r_squared_diff * 0.1, 0])
                r_squared_max_cero = np.min([r_squared_max + r_squared_diff*0.1, 1])
                if r_squared_min_cero < r_squared_limits[0]:
                    r_squared_limits[0] = r_squared_min_cero
                if r_squared_max_cero > r_squared_limits[1]:
                    r_squared_limits[1] = r_squared_max_cero

        r_squared_min_cero, r_squared_max_cero = r_squared_limits
        if r_squared_min_cero == float('inf') and r_squared_max_cero == float('-inf'):
            r_squared_min_cero, r_squared_max_cero = 0, 1

        r_squared_locator = abs(r_squared_max_cero - r_squared_min_cero) / 2

        if r_squared_locator <= 0:
            r_squared_locator = 0.1

        fig, ax = plt.subplots(figsize=(8, 6))

        for param_dict, color, title_suffix in zip(params_list, colors, title_suffixes):
            r_squared_array = param_dict[var_name]
            plot_parameter(ax, x, r_squared_array, color, title_suffix, r_squared_min_cero, r_squared_max_cero)

        ax.set_title(f'{title_suffixes[0]} Coefficient of Determination ($r^2$)', fontsize=22, pad=18)
        filename = f'{var_name}'
        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Coefficient of Determination ($r^2$)', fontsize=20, labelpad=13)
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(r_squared_min_cero, r_squared_max_cero)
        ax.legend(fontsize=18)

        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width=1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width=1.5)
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)
```

```
721
722         ax.xaxis.set_major_locator(MultipleLocator(x_locator))
723         ax.xaxis.set_minor_locator(AutoMinorLocator())
724         ax.yaxis.set_major_locator(MultipleLocator(r_squared_locator))
725         ax.yaxis.set_minor_locator(AutoMinorLocator())
726         ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
                minor=False)
727         ax.set_yticks(np.arange(r_squared_min_cero, r_squared_max_cero +
                r_squared_locator, r_squared_locator), minor=False)
728
729         # Invert x axis
730         #ax.invert_xaxis()
731
732         ax.grid(True, which='both', linestyle='--', linewidth=0.5)
733
734         plt.tight_layout()
735         if filenames:
736             filename = filenames[var_name]
737         save_plot_as_png(fig, directory, filename)
738
739
740
741 # Function to transform the map of positions (frequencies) into a strain map
742 def transform_position_linescan_and_plot_stress_linescan(data_dict, param_dict,
        title_suffix, directory, filenames=None):
743     """
744     Transforms the map of positions (frequencies) into a stress map and plots it
            .
745
746     Parameters:
747         data_to_fit (DataFrame): The data to fit.
748         param_array (ndarray): The array of fitted parameters.
749         title_suffix (str): Suffix for the plot titles to differentiate plots.
750
751         References:
752             - I. De Wolf (2011). Micro-Raman spectroscopy to study local
                    mechanical stress in silicon integrated ciurcuits, Volume 11,
                    https://iopscience.iop.org/article/10.1088/0268-1242/11/2/001
753             - Xu Li et. al The (2022) resolution and repeatability of stress
                    measurement by Raman and EBSD in silicon, DOI:10.1016/j.vacuum
                    .2022.111276
754     """
755     for var_name, data_to_fit in data_dict.items():
756         param_array = param_dict[var_name]
757
758         # Extract x coordinates
759         x = data_to_fit.iloc[1:, 0].astype(float)
760
761         # Create a figure and axis
762         fig, ax = plt.subplots(figsize=(8, 6))
763
764         wp_Si_ref = 520.7   # Frequency of reference Silicon
765
766         x0 = param_array[:, 0]
767
768         # Create a new array to store the restults
769         stress_new = np.zeros_like(x0)
770
```

```
771         # Calculate stress
772         stress_calculated = np.where(x0 != 0, (x0 - wp_Si_ref) * 0.434, 0)
773 
774         # Copiar los valores calculados en las posiciones correspondientes de x0
775         stress_new[x0 != 0] = stress_calculated[x0 != 0]
776 
777         # Set the limits of the axes based on data range
778         x_min, x_max = np.min(x), np.max(x)
779         x_diff = x_max - x_min
780         x_min_cero = x_min - x_diff * 0.1
781         x_max_cero = x_max + x_diff * 0.1
782 
783         stress_min, stress_max = np.min(stress_calculated), np.max(
                stress_calculated)
784         stress_diff = stress_max - stress_min
785         stress_min_cero = stress_min - stress_diff * 0.1
786         stress_max_cero = stress_max + stress_diff * 0.1
787 
788         # To establish the major locators
789         x_locator = abs(x_max_cero - x_min_cero) / 2
790         stress_locator = abs(stress_max_cero - stress_min_cero) / 2
791 
792         # Ensure the locator step is positive
793         if stress_locator <= 0:
794             stress_locator = 0.1  # You can adjust this to a more suitable
                    default value
795 
796         # Limits of the axes
797         ax.set_xlim(x_min_cero, x_max_cero)
798         ax.set_ylim(stress_min, stress_max)
799 
800         # Set the size of major ticks
801         ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
                =1.5)
802         ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
                =1.5)
803         # Set the size of minor ticks
804         ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
805         ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)
806 
807         # Set major and minor tick locators based on subplot
808         ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
                ticks
809         ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
810         ax.yaxis.set_major_locator(MultipleLocator(stress_locator))  # Set major
                 ticks
811         ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
812 
813         # Set the minimum and maximum major tick locations for x and y axes
814         ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
                minor=False)
815         ax.set_yticks(np.arange(stress_min_cero, stress_max_cero +
                stress_locator, stress_locator), minor=False)
816 
817         # Plot for stress
```

```
        ax.plot(x, stress_calculated, color='blue', linestyle='-', marker='o', markersize=5)
        ax.set_title(f'{title_suffix} Stress', fontsize=22, pad=18)

        # Update x-axis label based on filename
        filename = f'{var_name}'

        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Stress (GPa)', fontsize=20, labelpad=13)

        # Invert x axis
        #ax.invert_xaxis()

        # Add grid
        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        # Adjust layout
        plt.tight_layout()

        # Save plot as PNG if filenames are provided
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)




def transform_position_linescan_and_plot_stress_linescan_all_params(data_dict, params_list, title_suffixes, colors, directory, filenames=None):
    """
    Transforms the map of positions (frequencies) into a stress map and plots it.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        params_list (list): List of dictionaries of parameters arrays to plot.
        title_suffixes (list): List of suffixes for the plot titles to differentiate plots.
        colors (list): List of colors for each set of parameters.
        directory (str): Directory where the plots will be saved.
        filenames (dict, optional): Dictionary of filenames for saving the plots. Defaults to None.

        References:
            - I. De Wolf (2011). Micro-Raman spectroscopy to study local mechanical stress in silicon integrated circuits, Volume 11, https://iopscience.iop.org/article/10.1088/0268-1242/11/2/001
            - Xu Li et. al (2022). The resolution and repeatability of stress measurement by Raman and EBSD in silicon, DOI:10.1016/j.vacuum.2022.111276
```

```
    """
    def save_plot_as_png(fig, directory, filename):
        """
        Saves the plot as a PNG file.

        Parameters:
            fig (matplotlib.figure.Figure): Figure object to save.
            directory (str): Directory where the plot will be saved.
            filename (str): Filename for the saved plot.
        """
        os.makedirs(directory, exist_ok=True)
        full_path = os.path.join(directory, f"{filename}.png")
        fig.savefig(full_path, dpi=300, bbox_inches='tight')
        plt.close(fig)
        print(f"Plot saved as {full_path}")

    def plot_parameter(ax, x, y, color, label, y_min, y_max):
        mask = (y != wp_Si_ref * 0.434) & (y > y_min) & (y < y_max)  # Create a mask where y is not equal to wp_Si_ref * 0.434 and within limits
        ax.plot(x[mask], y[mask], color=color, linestyle='-', marker='o', markersize=5, label=label)
        ax.plot(x[~mask], y[~mask], color=color, linestyle='', marker='o', markersize=5)  # Plot points outside the limits without connecting them

    wp_Si_ref = 520.7  # Frequency of reference Silicon

    for var_name, data_to_fit in data_dict.items():
        x = data_to_fit.iloc[1:, 0].astype(float)
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1
        x_locator = abs(x_max_cero - x_min_cero) / 2

        stress_limits = [float('inf'), float('-inf')]

        for param_dict in params_list:
            param_array = param_dict[var_name]
            x0 = param_array[:, 0]
            stress_calculated = (x0 - wp_Si_ref) * 0.434
            stress_calculated = stress_calculated[x0 != 0]
            if len(stress_calculated) > 0:
                stress_min, stress_max = np.min(stress_calculated), np.max(stress_calculated)
                if stress_min < stress_limits[0]:
                    stress_limits[0] = stress_min
                if stress_max > stress_limits[1]:
                    stress_limits[1] = stress_max

        stress_min, stress_max = stress_limits
        if stress_min == float('inf') and stress_max == float('-inf'):
            stress_min, stress_max = 0, 1
        stress_diff = stress_max - stress_min
        stress_min_cero = stress_min - stress_diff * 0.1
        stress_max_cero = stress_max + stress_diff * 0.1
        stress_locator = abs(stress_max_cero - stress_min_cero) / 2

```

```
        if stress_locator <= 0:
            stress_locator = 0.1

        fig, ax = plt.subplots(figsize=(8, 6))

        for param_dict, color, title_suffix in zip(params_list, colors,
            title_suffixes):
            param_array = param_dict[var_name]
            x0 = param_array[:, 0]
            stress_calculated = (x0 - wp_Si_ref) * 0.434
            plot_parameter(ax, x, stress_calculated, color, title_suffix,
                stress_min_cero, stress_max_cero)

        ax.set_title(f'{title_suffixes[0]} Stress (GPa)', fontsize=22, pad=18)
        filename = f'{var_name}'
        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Stress (GPa)', fontsize=20, labelpad=13)
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(stress_min_cero, stress_max_cero)
        ax.legend(fontsize=18)

        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)

        ax.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax.xaxis.set_minor_locator(AutoMinorLocator())
        ax.yaxis.set_major_locator(MultipleLocator(stress_locator))
        ax.yaxis.set_minor_locator(AutoMinorLocator())
        ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
            minor=False)
        ax.set_yticks(np.arange(stress_min_cero, stress_max_cero +
            stress_locator, stress_locator), minor=False)

        # Invert x axis
        #ax.invert_xaxis()

        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        plt.tight_layout()
        if filenames:
            filename = filenames[var_name]
        save_plot_as_png(fig, directory, filename)
```

```
971 def plot_fitted_function_double_peak(data_dict, param_dict, title_suffix,
        directory, filenames=None):
972     """
973     Plots the fitted function for the double peak parameters for multiple data
            sets.
974 
975     Parameters:
976         data_dict (dict): Dictionary of data sets to fit.
977         param_dict (dict): Dictionary of parameters arrays to plot.
978         title_suffix (str): Suffix for the plot titles to differentiate plots.
979         directory (str): Directory where the plots will be saved.
980         filenames (dict): Dictionary of filenames for saving the plots.
981     """
982     def plot_parameter(x, y, title, ylabel, xlabel, y_min, y_max, x_min, x_max,
            x_locator, directory, filename, title_suffix, parameter_name, color):
983         """
984         Plots a single parameter with specified settings.
985 
986         Parameters:
987             x (array): X-axis data.
988             y (array): Y-axis data (parameter to plot).
989             title (str): Title of the plot.
990             ylabel (str): Y-axis label.
991             xlabel (str): X-axis label.
992             y_min (float): Minimum y-axis limit.
993             y_max (float): Maximum y-axis limit.
994             x_min (float): Minimum x-axis limit.
995             x_max (float): Maximum x-axis limit.
996             x_locator (float): Major locator for x-axis.
997             directory (str): Directory where the plot will be saved.
998             filename (str): Filename for the saved plot.
999             title_suffix (str): Suffix for the plot title.
1000            parameter_name (str): Parameter name for the plot file.
1001        """
1002        fig, ax = plt.subplots(figsize=(8, 6))
1003        ax.plot(x, y, color=color, linestyle='-', marker='o', markersize=5)
1004        ax.set_title(f'{title_suffix} {title}', fontsize=22, pad=18)
1005        ax.set_ylabel(ylabel, fontsize=20, labelpad=13)
1006        ax.set_xlabel(xlabel, fontsize=20, labelpad=13)
1007        ax.set_ylim(y_min, y_max)
1008        ax.set_xlim(x_min, x_max)
1009 
1010        # Adjust tick parameters
1011        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
                =1.5)
1012        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
                =1.5)
1013        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
1014        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)
1015 
1016        # Set major and minor tick locators based on subplot
1017        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
                ticks
1018        ax.xaxis.set_minor_locator(AutoMinorLocator())  # Set minor ticks
                automatically
1019 
1020        # Set the minimum and maximum major tick locations for x axis
```

```
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)

        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        # Desactivar la notaci n cient fica y el offset en el eje y
        ax.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax.yaxis.get_major_formatter().set_scientific(False)
        ax.ticklabel_format(axis='y', style='plain', useOffset=False)

        # Invert x axis
        #ax.invert_xaxis()

        plt.tight_layout()
        save_plot_as_png(fig, directory, f'{filename}_{parameter_name}',
            title_suffix)

    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

        # Extract x coordinates
        x = data_to_fit.iloc[1:, 0].astype(float)

        # Extract parameters
        x0 = param_array[:, 0]
        g_FWHM = param_array[:, 1]
        l_FWHM = param_array[:, 2]
        A = param_array[:, 3]

        # Calculate limits for each parameter if there are non-zero values
        def calculate_limits(param):
            if np.any(param > 0):
                param_min = np.min(param[param > 0])
                param_max = np.max(param)
                param_diff = param_max - param_min
                param_min_cero = param_min - param_diff * 0.1
                param_max_cero = param_max + param_diff * 0.1

                # Adjust param_min_cero if it's less than 0
                if param_min_cero < 0:
                    param_min_cero = 0
            else:
                param_min = 0
                param_max = 0
                param_min_cero = 0
                param_max_cero = 0

            return param_min_cero, param_max_cero, param_min, param_max

        # Set the limits of the axes based on data range and calculated limits
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1

        # To establish the major locators
        x_locator = abs(x_max_cero - x_min_cero) / 2

```

```
        x0_min_cero, x0_max_cero, x0_min, x0_max = calculate_limits(x0)
        g_FWHM_min_cero, g_FWHM_max_cero, g_FWHM_min, g_FWHM_max =
            calculate_limits(g_FWHM)
        l_FWHM_min_cero, l_FWHM_max_cero, l_FWHM_min, l_FWHM_max =
            calculate_limits(l_FWHM)
        A_min_cero, A_max_cero, A_min, A_max = calculate_limits(A)

        filename = f'{var_name}'

        if 'Z_linescan' in filename:
            x_label = 'Depth ($\mu m$)'
        elif 'Y_linescan' in filename:
            x_label = 'Y ($\mu m$)'
        elif 'X_linescan' in filename:
            x_label = 'X ($\mu m$)'
        else:
            x_label = 'X axis ($\mu m$)'

        # Plot each parameter
        plot_parameter(x, x0, 'Phonon Frequency', 'Phonon Frequency ($cm^{-1}$)'
            , x_label, x0_min_cero, x0_max_cero, x_min_cero, x_max_cero,
            x_locator, directory, filename, title_suffix, 'phonon_frequency', '
            blue')
        plot_parameter(x, g_FWHM, 'Gaussian FWHM', 'Gaussian FWHM ($cm^{-1}$)',
            x_label, g_FWHM_min_cero, g_FWHM_max_cero, x_min_cero, x_max_cero,
            x_locator, directory, filename, title_suffix, 'gaussian_FWHM', 'blue
            ')
        plot_parameter(x, l_FWHM, 'Lorentzian FWHM', 'Lorentzian FWHM ($cm^{-1}$
            )', x_label, l_FWHM_min_cero, l_FWHM_max_cero, x_min_cero,
            x_max_cero, x_locator, directory, filename, title_suffix, '
            lorentzian_FWHM', 'blue')
        plot_parameter(x, A, 'Intensity', 'Intensity (A.U.)', x_label,
            A_min_cero, A_max_cero, x_min_cero, x_max_cero, x_locator, directory
            , filename, title_suffix, 'intensity', 'blue')



def plot_fitted_function_double_peak_all_params_together(data_dict, params_list,
     title_suffixes, colors, directory, filenames=None, T_0=25, omega_0=520.7):
    def save_plot_as_png(fig, directory, filename):
        os.makedirs(directory, exist_ok=True)
        full_path = os.path.join(directory, f"{filename}.png")
        fig.savefig(full_path, dpi=300, bbox_inches='tight')
        print(f"Plot saved as {full_path}")
        plt.close(fig)

    def plot_parameter(ax, x, y, color, marker='o', linestyle='-', linewidth
        =1.5):
        mask = y > 0
        ax.plot(x[mask], y[mask], color='black', linestyle=linestyle, linewidth=
            linewidth)
        ax.plot(x[mask], y[mask], color=color, linestyle='None', marker=marker,
            markersize=6)

    def adjust_limits_with_legend(axis, x, y, limits):
        axis.plot(x, y)
        new_limits = axis.get_ylim()
        axis.set_ylim(limits)
```

```
        return new_limits

    def calculate_limits(param):
        if np.any(param > 0):
            param_min = np.min(param[param > 0])
            param_max = np.max(param)
            param_diff = param_max - param_min
            param_min_cero = param_min - param_diff * 0.1
            param_max_cero = param_max + param_diff * 0.1
            if param_min_cero < 0:
                param_min_cero = 0
        else:
            param_min = 0
            param_max = 0
            param_min_cero = 0
            param_max_cero = 0
        return param_min_cero, param_max_cero, param_min, param_max

    markers = ['o', 's', '^', 'D']  # Different markers for each param set

    for var_name, data_to_fit in data_dict.items():
        x = data_to_fit.iloc[1:, 0].astype(float)
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1
        x_locator = abs(x_max_cero - x_min_cero) / 4

        filename = f'{var_name}'
        if 'Z_linescan' in filename:
            x_label = 'Depth ($\mu m$)'
        elif 'Y_linescan' in filename:
            x_label = 'Y ($\mu m$)'
        elif 'X_linescan' in filename:
            x_label = 'X ($\mu m$)'
        else:
            x_label = 'X axis ($\mu m$)'

        param_names = ['phonon_frequency', 'gaussian_FWHM', 'lorentzian_FWHM', '
            intensity']
        y_labels = ['Phonon Frequency ($cm^{-1}$)', 'Gaussian FWHM ($cm^{-1}$)',
             'Lorentzian FWHM ($cm^{-1}$)', 'Intensity (A.U.)']
        titles = ['Phonon Frequency', 'Gaussian FWHM', 'Lorentzian FWHM', '
            Intensity']

        limits = {name: [float('inf'), float('-inf')] for name in param_names}

        for param_dict in params_list:
            param_array = param_dict[var_name]
            for i, param_name in enumerate(param_names):
                param = param_array[:, i]
                positive_param = param[param > 0]
                if len(positive_param) > 0:
                    param_min, param_max = np.min(positive_param), np.max(
                        positive_param)
                    if param_min < limits[param_name][0]:
                        limits[param_name][0] = param_min
                    if param_max > limits[param_name][1]:
```

```
                    limits[param_name][1] = param_max

        for param_name in param_names:
            param_min, param_max = limits[param_name]
            if param_min == float('inf') and param_max == float('-inf'):
                param_min, param_max = 0, 1
            param_diff = param_max - param_min
            param_min_cero = param_min - param_diff * 0.1
            param_max_cero = param_max + param_diff * 0.1
            limits[param_name] = (param_min_cero, param_max_cero)

        for i, param_name in enumerate(param_names):
            fig, ax = plt.subplots(figsize=(8, 6))
            for j, (param_dict, color, title_suffix) in enumerate(zip(
                params_list, colors, title_suffixes)):
                param_array = param_dict[var_name]
                param = param_array[:, i]
                plot_parameter(ax, x, param, color, marker=markers[j])
            ax.set_title(titles[i], fontsize=22, pad=18)
            ax.set_ylabel(y_labels[i], fontsize=20, labelpad=13)
            ax.set_xlabel(x_label, fontsize=20, labelpad=13)
            ax.set_ylim(*limits[param_name])
            ax.set_xlim(x_min_cero, x_max_cero)

            ax.tick_params(axis="x", labelsize=18, which="major", length=7.5,
                width=1.5)
            ax.tick_params(axis="y", labelsize=18, which="major", length=7.5,
                width=1.5)
            ax.tick_params(axis="x", labelsize=16, which="minor", length=5,
                width=1)
            ax.tick_params(axis="y", labelsize=16, which="minor", length=5,
                width=1)

            ax.xaxis.set_major_locator(MultipleLocator(x_locator))
            ax.xaxis.set_minor_locator(AutoMinorLocator())
            ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator,
                x_locator), minor=False)

            ax.grid(True, which='both', linestyle='--', linewidth=0.5)

            ax.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
            ax.yaxis.get_major_formatter().set_scientific(False)
            ax.ticklabel_format(axis='y', style='plain', useOffset=False)

            plt.tight_layout()
            save_plot_as_png(fig, directory, f'{filename}_{param_name}')

        # Plot Phonon Frequency and Temperature together
        fig, ax1 = plt.subplots(figsize=(8, 6))

        # Plot Position on the left y-axis
        ax1.set_xlabel(x_label, fontsize=20, labelpad=13)
        ax1.set_ylabel('Phonon Frequency ($cm^{-1}$)', fontsize=20, labelpad=13)
        ax1.set_xlim(x_min_cero, x_max_cero)
        ax1.set_ylim(*limits['phonon_frequency'])
        ax1.tick_params(axis='y')

```

```
        for j, (param_dict, color, title_suffix) in enumerate(zip(params_list,
            colors, title_suffixes)):
            param_array = param_dict[var_name]
            phonon_frequency = param_array[:, param_names.index('
                phonon_frequency')]
            ax1.plot(x, phonon_frequency, color=color)

        # Create a second y-axis for Temperature
        ax2 = ax1.twinx()
        ax2.set_ylabel('Temperature ( C )', fontsize=20, labelpad=13)

        ax1.set_title('Position and Temperature', fontsize=22, pad=18)

        # Adjusting limits for phonon_frequency
        limit_phonon_diff = limits['phonon_frequency'][1] - limits['
            phonon_frequency'][0]
        limits['phonon_frequency'] = (limits['phonon_frequency'][0], limits['
            phonon_frequency'][1])

        new_y_limits = adjust_limits_with_legend(ax1, x, phonon_frequency,
            limits['phonon_frequency'])

        # Transform the limits of the phonon frequency to temperature
        temp_min = frecuency_to_temperature(new_y_limits[0], T_0, omega_0)
        temp_max = frecuency_to_temperature(new_y_limits[1], T_0, omega_0)
        ax2.set_ylim(temp_min, temp_max)
        ax2.tick_params(axis='y')

        ax1.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax1.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax1.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax1.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax1.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax1.xaxis.set_minor_locator(AutoMinorLocator())
        ax1.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax2.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax2.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax2.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax2.xaxis.set_minor_locator(AutoMinorLocator())
        ax2.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

```

```
        ax1.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax1.yaxis.get_major_formatter().set_scientific(False)
        ax1.ticklabel_format(axis='y', style='plain', useOffset=False)

        plt.tight_layout()
        save_plot_as_png(fig, directory, f'{filename}
            _phonon_frequency_and_temperature')

        # Plot Intensity and Lorentzian FWHM together
        fig, ax1 = plt.subplots(figsize=(8, 6))

        # Plot Intensity on the left y-axis
        ax1.set_xlabel(x_label, fontsize=20, labelpad=13)
        ax1.set_ylabel('Intensity (A.U.)', fontsize=20, labelpad=13)
        ax1.set_xlim(x_min_cero, x_max_cero)
        ax1.set_ylim(*limits['intensity'])
        ax1.tick_params(axis='y')

        for j, (param_dict, color, title_suffix) in enumerate(zip(params_list,
            colors, title_suffixes)):
            param_array = param_dict[var_name]
            intensity = param_array[:, param_names.index('intensity')]
            mask = intensity != 0
            ax1.plot(x[mask], intensity[mask], color=color, linewidth=2)

        # Create a second y-axis for Lorentzian FWHM
        ax2 = ax1.twinx()
        ax2.set_ylabel('Lorentzian FWHM ($cm^{-1}$)', fontsize=20, labelpad=13)
        ax2.set_ylim(*limits['lorentzian_FWHM'])
        ax2.tick_params(axis='y')

        for j, (param_dict, color, title_suffix) in enumerate(zip(params_list,
            colors, title_suffixes)):
            param_array = param_dict[var_name]
            lorentzian_fwhm = param_array[:, param_names.index('lorentzian_FWHM'
                )]
            mask = lorentzian_fwhm != 0
            ax2.plot(x[mask], lorentzian_fwhm[mask], color=color, linestyle='
                None', marker='o', markersize=8)

        ax1.set_title('Intensity and Lorentzian FWHM', fontsize=22, pad=18)

        # Adjusting limits for intensity
        limit_intensity_diff = limits['intensity'][1] - limits['intensity'][0]
        limits['intensity'] = (limits['intensity'][0] - limit_intensity_diff,
            limits['intensity'][1])

        limit_FWHM_lorentz_diff = limits['lorentzian_FWHM'][1] - limits['
            lorentzian_FWHM'][0]
        limits['lorentzian_FWHM'] = (limits['lorentzian_FWHM'][0], limits['
            lorentzian_FWHM'][1] + limit_FWHM_lorentz_diff)

        adjust_limits_with_legend(ax1, x, intensity, limits['intensity'])
        adjust_limits_with_legend(ax2, x, lorentzian_fwhm, limits['
            lorentzian_FWHM'])

        ax1.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
```

```
        ax1.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax1.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax1.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax1.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax1.xaxis.set_minor_locator(AutoMinorLocator())
        ax1.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax2.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax2.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax2.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax2.xaxis.set_minor_locator(AutoMinorLocator())
        ax2.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

        ax1.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax1.yaxis.get_major_formatter().set_scientific(False)
        ax1.ticklabel_format(axis='y', style='plain', useOffset=False)

        ax2.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax2.yaxis.get_major_formatter().set_scientific(False)
        ax2.ticklabel_format(axis='y', style='plain', useOffset=False)

        plt.tight_layout()
        save_plot_as_png(fig, directory, f'{filename}_intensity_lorentzian_FWHM'
            )

        # Plot Position and Stress together
        fig, ax1 = plt.subplots(figsize=(8, 6))

        # Plot Position on the left y-axis
        ax1.set_xlabel(x_label, fontsize=20, labelpad=13)
        ax1.set_ylabel('Phonon Frequency ($cm^{-1}$)', fontsize=20, labelpad=13)
        ax1.set_xlim(x_min_cero, x_max_cero)
        ax1.set_ylim(*limits['phonon_frequency'])
        ax1.tick_params(axis='y')

        for j, (param_dict, color, title_suffix) in enumerate(zip(params_list,
            colors, title_suffixes)):
            param_array = param_dict[var_name]
            phonon_frequency = param_array[:, param_names.index('
                phonon_frequency')]
            ax1.plot(x, phonon_frequency, color=color)

        # Create a second y-axis for Stress
```

```
        ax2 = ax1.twinx()
        ax2.set_ylabel('Stress (GPa)', fontsize=20, labelpad=13)

        ax1.set_title('Position and Stress', fontsize=22, pad=18)

        # Adjusting limits for phonon_frequency
        limit_phonon_diff = limits['phonon_frequency'][1] - limits['
            phonon_frequency'][0]
        limits['phonon_frequency'] = (limits['phonon_frequency'][0], limits['
            phonon_frequency'][1])

        new_y_limits = adjust_limits_with_legend(ax1, x, phonon_frequency,
            limits['phonon_frequency'])

        # Transform the limits of the phonon frequency to temperature
        wp_Si_ref = 520.7  # Reference frequency for Silicon
        stress_min = (new_y_limits[0] - wp_Si_ref) * 0.434
        stress_max = (new_y_limits[1] - wp_Si_ref) * 0.434
        ax2.set_ylim(stress_min, stress_max)
        ax2.tick_params(axis='y')

        ax1.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax1.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax1.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax1.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax1.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax1.xaxis.set_minor_locator(AutoMinorLocator())
        ax1.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax2.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax2.tick_params(axis="x", labelsize=16, which="minor", length=5, width
            =1)
        ax2.tick_params(axis="y", labelsize=16, which="minor", length=5, width
            =1)

        ax2.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax2.xaxis.set_minor_locator(AutoMinorLocator())
        ax2.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
             minor=False)

        ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

        ax1.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax1.yaxis.get_major_formatter().set_scientific(False)
        ax1.ticklabel_format(axis='y', style='plain', useOffset=False)

        ax2.yaxis.set_major_formatter(plt.ScalarFormatter(useMathText=True))
        ax2.yaxis.get_major_formatter().set_scientific(False)
        ax2.ticklabel_format(axis='y', style='plain', useOffset=False)
```

```
1405
1406            plt.tight_layout()
1407            save_plot_as_png(fig, directory, f'{filename}_position_stress')
1408
1409            plt.close('all')
1410
1411
1412
1413    def frecuency_to_temperature(frecuencies, T_0, omega_0):
1414        """
1415        Converts a list of frequencies to temperatures using the provided formula.
1416
1417        Parameters:
1418        frequencies (list or np.ndarray): List or array of frequencies.
1419        T_0 (float): Constant T_0.
1420        omega_0 (float): Constant omega_0.
1421
1422        Returns:
1423        np.ndarray: Array of temperatures.
1424
1425        References:
1426            - Tsu R. and Gonzalez Hernandez J. (1982). Temperature dependence of
                  silicon Raman lines. Appl. Phys. Lett. 41, 1016-1018, https://doi.
                  org/10.1063/1.93394
1427
1428        """
1429        frecuencies = np.array(frecuencies)
1430        temperatures = T_0 + (1 / (-5.4e-5)) * np.log(frecuencies / omega_0)
1431
1432        #temperatures = T_0 + (1 / (-5.4e-5)) * ((frecuencies - omega_0) / omega_0)
1433
1434        return temperatures
1435
1436    def plot_temperature_linescan_position(data_dict, param_dict, title_suffix,
        directory, filenames=None, T_0=25, omega_0=520.7):
1437        """
1438        Plots the temperature map for the given data sets and parameters.
1439
1440        Parameters:
1441            data_dict (dict): Dictionary of data sets to fit.
1442            param_dict (dict): Dictionary of parameters arrays to plot.
1443            title_suffix (str): Suffix for the plot titles to differentiate plots.
1444            directory (str): Directory where the plots will be saved.
1445            filenames (dict): Dictionary of filenames for saving the plots.
1446            T_0 (float): Constant T_0 for temperature conversion. Default is 25.
1447            omega_0 (float): Constant omega_0 for temperature conversion. Default is
                  520.7.
1448        """
1449        for var_name, data_to_fit in data_dict.items():
1450            param_array = param_dict[var_name]
1451
1452            # Extract x coordinates
1453            x = data_to_fit.iloc[1:, 0].astype(float)
1454
1455            # Create figure
1456            fig, ax = plt.subplots(figsize=(8, 6))
1457
1458            # Extract parameters
```

```
        x0 = param_array[:, 0]

        # Convert frequencies into temperatures
        temperature_calculated = frecuency_to_temperature(x0, T_0, omega_0)

        # Create an array for the values of the temperatures
        temps_new = np.zeros_like(temperature_calculated)
        valid_temps = np.isfinite(temperature_calculated)  # Mask for valid
            values
        temps_new[valid_temps] = temperature_calculated[valid_temps]

        # Filter for 0 values
        non_zero_temps = temps_new[temps_new != 0]

        # Verify that there are valid values
        if non_zero_temps.size == 0:
            temperature_min, temperature_max = 0, 1
            temps_new = np.zeros_like(x)  # Graficar solo ceros
        else:
            # Set axis limits based on data range
            temperature_min, temperature_max = np.min(non_zero_temps), np.max(
                non_zero_temps)

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1

        temp_diff = temperature_max - temperature_min
        temp_min_cero = temperature_min - temp_diff * 0.1
        temp_max_cero = temperature_max + temp_diff * 0.1

        # Set primary locators
        x_locator = abs(x_max_cero - x_min_cero) / 2
        temperature_locator = abs(temp_max_cero - temp_min_cero) / 2

        # Ensure that the locator pitch is positive
        if temperature_locator <= 0:
            temperature_locator = 1  # You can adjust this to a more suitable
                default value

        # Axis limits
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(temp_min_cero, temp_max_cero)

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)

        # Set primary and secondary locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax.xaxis.set_minor_locator(AutoMinorLocator())
```

```
1512            ax.yaxis.set_major_locator(MultipleLocator(temperature_locator))
1513            ax.yaxis.set_minor_locator(AutoMinorLocator())
1514
1515            # Set the minimum and maximum locations of the major marks for the x and
                    y axes
1516            ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
                    minor=False)
1517            ax.set_yticks(np.arange(temp_min_cero, temp_max_cero +
                    temperature_locator, temperature_locator), minor=False)
1518
1519            # Plots
1520            ax.plot(x, temps_new, color='blue', linestyle='-', marker='o',
                    markersize=5)
1521            ax.set_title(f'{title_suffix} Temperature ( C )', fontsize=22, pad=18)
1522
1523            # Name of the file
1524            filename = f'{var_name}'
1525
1526            if 'Z_linescan' in filename:
1527                ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
1528            elif 'Y_linescan' in filename:
1529                ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
1530            elif 'X_linescan' in filename:
1531                ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
1532            else:
1533                ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)
1534
1535            ax.set_ylabel('Temperature ( C )', fontsize=20, labelpad=13)
1536
1537            # Invert x axis
1538            #ax.invert_xaxis()
1539
1540            # Grid
1541            ax.grid(True, which='both', linestyle='--', linewidth=0.5)
1542
1543            # Layout
1544            plt.tight_layout()
1545
1546            # Save plot as png
1547            if filenames:
1548                filename = filenames[var_name]
1549                save_plot_as_png(fig, directory, filename, title_suffix)
1550
1551
1552
1553    def plot_temperature_linescan_position_all_params(data_dict, params_list,
        title_suffixes, colors, directory, filenames=None, T_0=25, omega_0=520.7):
1554        """
1555        Transforms the map of positions (frequencies) into a stress map and plots it
                .
1556
1557        Parameters:
1558            data_dict (dict): Dictionary of data sets to fit.
1559            params_list (list): List of dictionaries of parameters arrays to plot.
1560            title_suffixes (list): List of suffixes for the plot titles to
                    differentiate plots.
1561            colors (list): List of colors for each set of parameters.
1562            directory (str): Directory where the plots will be saved.
```

```
        filenames (dict, optional): Dictionary of filenames for saving the plots. Defaults to None.
    """
    def save_plot_as_png(fig, directory, filename):
        """
        Saves the plot as a PNG file.

        Parameters:
            fig (matplotlib.figure.Figure): Figure object to save.
            directory (str): Directory where the plot will be saved.
            filename (str): Filename for the saved plot.
        """
        os.makedirs(directory, exist_ok=True)
        full_path = os.path.join(directory, f"{filename}.png")
        fig.savefig(full_path, dpi=300, bbox_inches='tight')
        plt.close(fig)
        print(f"Plot saved as {full_path}")

    def plot_parameter(ax, x, y, color, label, y_min, y_max):
        mask = (y != 0) & (y > y_min) & (y < y_max)  # Create a mask within limits
        ax.plot(x[mask], y[mask], color=color, linestyle='-', marker='o', markersize=5, label=label)
        ax.plot(x[~mask], y[~mask], color=color, linestyle='', marker='o', markersize=5)  # Plot points outside the limits without connecting them

    for var_name, data_to_fit in data_dict.items():
        x = data_to_fit.iloc[1:, 0].astype(float)
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1
        x_locator = abs(x_max_cero - x_min_cero) / 2

        temps_limits = [float('inf'), float('-inf')]

        for param_dict in params_list:
            param_array = param_dict[var_name]
            x0 = param_array[:, 0]
            temperature_calculated = frecuency_to_temperature(x0, T_0, omega_0)
            temps_new = np.zeros_like(temperature_calculated)
            valid_temps = np.isfinite(temperature_calculated)  # M scara de valores v lidos
            temps_new[valid_temps] = temperature_calculated[valid_temps]
            if len(temps_new) > 0:
                temps_min, temps_max = np.min(temps_new), np.max(temps_new)
                if temps_min < temps_limits[0]:
                    temps_limits[0] = temps_min
                if temps_max > temps_limits[1]:
                    temps_limits[1] = temps_max

        temps_min, temps_max = temps_limits
        if temps_min == float('inf') and temps_max == float('-inf'):
            temps_min, temps_max = 0, 1
        temps_diff = temps_max - temps_min
        temps_min_cero = np.max([temps_min - temps_diff * 0.1, 5])
        temps_max_cero = temps_max + temps_diff * 0.1
```

```
        temps_locator = abs(temps_max_cero - temps_min_cero) / 2

        if temps_locator <= 0:
            temps_locator = 0.1

        fig, ax = plt.subplots(figsize=(8, 6))

        for param_dict, color, title_suffix in zip(params_list, colors,
            title_suffixes):
            param_array = param_dict[var_name]
            x0 = param_array[:, 0]
            temperature_calculated = frecuency_to_temperature(x0, T_0, omega_0)
            temps_new = np.zeros_like(temperature_calculated)
            valid_temps = np.isfinite(temperature_calculated)  # M scara de
                valores v lidos
            temps_new[valid_temps] = temperature_calculated[valid_temps]
            plot_parameter(ax, x, temps_new, color, title_suffix, temps_min_cero
                , temps_max_cero)

        ax.set_title(f'{title_suffixes[0]} Temperature ( C )', fontsize=22, pad
            =18)
        filename = f'{var_name}'
        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Temperature ( C )', fontsize=20, labelpad=13)
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(temps_min_cero, temps_max_cero)
        ax.legend(fontsize=18)

        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)

        ax.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax.xaxis.set_minor_locator(AutoMinorLocator())
        ax.yaxis.set_major_locator(MultipleLocator(temps_locator))
        ax.yaxis.set_minor_locator(AutoMinorLocator())
        ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
            minor=False)
        ax.set_yticks(np.arange(temps_min_cero, temps_max_cero + temps_locator,
            temps_locator), minor=False)

        # Invert x axis
        #ax.invert_xaxis()

        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        plt.tight_layout()
```

```
        if filenames:
            filename = filenames[var_name]
        save_plot_as_png(fig, directory, filename)



def l_FWHM_to_temperature(gamma, Omega):
    """
    Resuelve la ecuaci n para encontrar T dado Gamma usando m todos num ricos.

    Parameters:
        gamma (float): El valor de Gamma.
        hbar (float): La constante de Planck reducida.
        Omega (float): La frecuencia angular.
        K (float): La constante de Boltzmann.

    Returns:
        float: La temperatura T en Kelvin.

        References:
            - Menendez J. and Cardona M. (1984). Temperature dependence of the
                first Raman scattering by phonons inSi, Ge, and \alpha-Sn:
                Anharmonic effects. Physical Review B, Vol 29, N  4 , https://
                doi.org/10.1103/PhysRevB.29.2051
    """
    def equation(T):
        # Definition of the equation to solve f(T) = 0
        X = (gamma - 1.24) / 1.24
        term1 = np.exp(0.35 * 1.4388 * Omega / T) - 1
        term2 = np.exp(0.65 * 1.4388 * Omega / T) - 1
        return X - (1 / term1 + 1 / term2)

    T_initial = 300  # Initial estimate of T in Kelvin
    T_solution = fsolve(equation, T_initial)  # Solve the equation
    return T_solution[0]  # Return the solution found

def plot_temperature_linescan_FWHM(data_dict, param_dict, title_suffix,
    directory, filenames=None, omega_0=None):
    """
    Plots the temperature map for the given data sets and parameters.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        param_dict (dict): Dictionary of parameters arrays to plot.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
        directory (str): Directory where the plots will be saved.
        filenames (dict): Dictionary of filenames for saving the plots.
        omega_0 (float): Constant omega_0 for temperature conversion. Default is
            None.
    """

    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

        # Extract x coordinates
        x = data_to_fit.iloc[1:, 0].astype(float)

```

```
        # Extract Gamma parameter (l_FWHM)
        l_FWHM = param_array[:, 2]

        # Convert Gamma to Temperatures
        temperature_calculated = np.array([l_FWHM_to_temperature(g, omega_0) for g in l_FWHM])

        # Create a figure and an axis
        fig, ax = plt.subplots(figsize=(8, 6))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        x_diff = x_max - x_min
        x_min_cero = x_min - x_diff * 0.1
        x_max_cero = x_max + x_diff * 0.1
        temperature_min, temperature_max = np.min(temperature_calculated), np.max(temperature_calculated)

        temp_diff = temperature_max - temperature_min
        temp_min_cero = temperature_min - temp_diff * 0.1
        temp_max_cero = temperature_max + temp_diff * 0.1

        # Set primary locators
        x_locator = abs(x_max_cero - x_min_cero) / 2
        temperature_locator = abs(temp_max_cero - temp_min_cero) / 2

        # Ensure that the locator pitch is positive
        if temperature_locator <= 0:
            temperature_locator = 1

        # Axis limits
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(temp_min_cero, temp_max_cero)

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width=1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width=1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)

        # Set primary and secondary locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax.xaxis.set_minor_locator(AutoMinorLocator())
        ax.yaxis.set_major_locator(MultipleLocator(temperature_locator))
        ax.yaxis.set_minor_locator(AutoMinorLocator())

        # Set the minimum and maximum locations of the major marks for the x and y axes
        ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator), minor=False)
        ax.set_yticks(np.arange(temp_min_cero, temp_max_cero + temperature_locator, temperature_locator), minor=False)

        # Plots
```

```
        ax.plot(x, temperature_calculated, color='blue', linestyle='-', marker='o', markersize=5)
        ax.set_title(f'{title_suffix} Temperature (K)', fontsize=22, pad=18)

        # Name of the file
        filename = f'{var_name}'

        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Temperature (K)', fontsize=20, labelpad=13)

        # Invert x axis
        #ax.invert_xaxis()

        # Grid
        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        # Layout
        plt.tight_layout()

        # Save plot as png
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)


def plot_temperature_linescan_FWHM_all_params(data_dict, params_list, title_suffixes, colors, directory, filenames=None, omega_0=520.7):
    """
    Transforms the map of positions (frequencies) into a stress map and plots it.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        params_list (list): List of dictionaries of parameters arrays to plot.
        title_suffixes (list): List of suffixes for the plot titles to differentiate plots.
        colors (list): List of colors for each set of parameters.
        directory (str): Directory where the plots will be saved.
        filenames (dict, optional): Dictionary of filenames for saving the plots. Defaults to None.
    """
    def save_plot_as_png(fig, directory, filename):
        """
        Saves the plot as a PNG file.

        Parameters:
            fig (matplotlib.figure.Figure): Figure object to save.
            directory (str): Directory where the plot will be saved.
            filename (str): Filename for the saved plot.
        """
```

```
1820            os.makedirs(directory, exist_ok=True)
1821            full_path = os.path.join(directory, f"{filename}.png")
1822            fig.savefig(full_path, dpi=300, bbox_inches='tight')
1823            plt.close(fig)
1824            print(f"Plot saved as {full_path}")
1825
1826        def plot_parameter(ax, x, y, color, label, y_min, y_max):
1827            mask = (y != 0) & (y > y_min) & (y < y_max)  # Create a mask within
                limits
1828            ax.plot(x[mask], y[mask], color=color, linestyle='-', marker='o',
                markersize=5, label=label)
1829            ax.plot(x[~mask], y[~mask], color=color, linestyle='', marker='o',
                markersize=5)  # Plot points outside the limits without connecting
                them
1830
1831        for var_name, data_to_fit in data_dict.items():
1832            x = data_to_fit.iloc[1:, 0].astype(float)
1833            x_min, x_max = np.min(x), np.max(x)
1834            x_diff = x_max - x_min
1835            x_min_cero = x_min - x_diff * 0.1
1836            x_max_cero = x_max + x_diff * 0.1
1837            x_locator = abs(x_max_cero - x_min_cero) / 2
1838
1839            temps_limits = [float('inf'), float('-inf')]
1840
1841            for param_dict in params_list:
1842                param_array = param_dict[var_name]
1843                l_FWHM = param_array[:, 2]
1844                temperature_calculated = np.array([l_FWHM_to_temperature(g, omega_0)
                        for g in l_FWHM])
1845                temps_new = np.zeros_like(temperature_calculated)
1846                valid_temps = np.isfinite(temperature_calculated)  # M scara de
                    valores v lidos
1847                temps_new[valid_temps] = temperature_calculated[valid_temps]
1848                if len(temps_new) > 0:
1849                    temps_min, temps_max = np.min(temps_new), np.max(temps_new)
1850                    if temps_min < temps_limits[0]:
1851                        temps_limits[0] = temps_min
1852                    if temps_max > temps_limits[1]:
1853                        temps_limits[1] = temps_max
1854
1855            temps_min, temps_max = temps_limits
1856            if temps_min == float('inf') and temps_max == float('-inf'):
1857                temps_min, temps_max = 0, 1
1858            temps_diff = temps_max - temps_min
1859            temps_min_cero = np.max([temps_min - temps_diff * 0.1, 5])
1860            temps_max_cero = temps_max + temps_diff * 0.1
1861            temps_locator = abs(temps_max_cero - temps_min_cero) / 2
1862
1863            if temps_locator <= 0:
1864                temps_locator = 0.1
1865
1866            fig, ax = plt.subplots(figsize=(8, 6))
1867
1868            for param_dict, color, title_suffix in zip(params_list, colors,
                title_suffixes):
1869                param_array = param_dict[var_name]
1870                l_FWHM = param_array[:, 2]
```

```
            temperature_calculated = np.array([l_FWHM_to_temperature(g, omega_0)
                for g in l_FWHM])
            temps_new = np.zeros_like(temperature_calculated)
            valid_temps = np.isfinite(temperature_calculated)  # M scara de
                valores v lidos
            temps_new[valid_temps] = temperature_calculated[valid_temps]
            plot_parameter(ax, x, temps_new, color, title_suffix, temps_min_cero
                , temps_max_cero)

        ax.set_title(f'{title_suffixes[0]} Temperature (K)', fontsize=22, pad
            =18)
        filename = f'{var_name}'
        if 'Z_linescan' in filename:
            ax.set_xlabel('Depth ($\mu m$)', fontsize=20, labelpad=13)
        elif 'Y_linescan' in filename:
            ax.set_xlabel('Y ($\mu m$)', fontsize=20, labelpad=13)
        elif 'X_linescan' in filename:
            ax.set_xlabel('X ($\mu m$)', fontsize=20, labelpad=13)
        else:
            ax.set_xlabel('X axis ($\mu m$)', fontsize=20, labelpad=13)

        ax.set_ylabel('Temperature (K)', fontsize=20, labelpad=13)
        ax.set_xlim(x_min_cero, x_max_cero)
        ax.set_ylim(temps_min_cero, temps_max_cero)
        ax.legend(fontsize=18)

        ax.tick_params(axis="x", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=18, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="x", labelsize=16, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=16, which="minor", length=5, width=1)

        ax.xaxis.set_major_locator(MultipleLocator(x_locator))
        ax.xaxis.set_minor_locator(AutoMinorLocator())
        ax.yaxis.set_major_locator(MultipleLocator(temps_locator))
        ax.yaxis.set_minor_locator(AutoMinorLocator())
        ax.set_xticks(np.arange(x_min_cero, x_max_cero + x_locator, x_locator),
            minor=False)
        ax.set_yticks(np.arange(temps_min_cero, temps_max_cero + temps_locator,
            temps_locator), minor=False)

        # Invert x axis
        #ax.invert_xaxis()

        ax.grid(True, which='both', linestyle='--', linewidth=0.5)

        plt.tight_layout()
        if filenames:
            filename = filenames[var_name]
        save_plot_as_png(fig, directory, filename)



# Directory to save the files
# Directory for Y linescan parallel and perpendicular to polarization Green
    laser Julian
```

```
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\Chip1_Green_laser_linescans\SiNW_Parallel_to_Polarization"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\Chip1_Green_laser_linescans\SiNW_Perp_to_Polarization"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_Y_linescans_NW_ParPol_without_heating"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_Z_linescans_OutContact_Julian"
# Directory for linescans with Green laser
directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_Z_linescans_heating\Paral_Pol"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_Z_linescans_heating\Perp_Pol"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_XY_linescans_heating\Paral_Pol"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_G_laser_XY_linescans_heating\Perp_Pol"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Green_laser_linescans_PosMark"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Green_laser_linescans_R_Edge"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\Prueba"

# List of file paths and corresponding variable names
file_var_mapping = {
#    "Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_10um_5_7mW_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240322_Paral_Perp_Polarization\SiNW_Parallel_to_Polarization\Ajustado_Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_Edge_2400 (400nm)_100x_10 m_10 % (5_7mW)_1.txt",
#    "Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_200um_5_7mW_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240322_Paral_Perp_Polarization\SiNW_Parallel_to_Polarization\Ajustado_Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_Edge_2400 (400nm)_100x_200 m_10 % (5_7mW)_.txt",
#    "Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_10um_5_7mW_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240322_Paral_Perp_Polarization\SiNW_Perpendicular_to_Polarization_90deg\Ajustado_Chip1_90deg_triangular_Row1_40nm_Col8_1200nm_532nm_2400_10_ m.txt",
#    "Chip1_Tri_Row1_40nm_Col8_1200nm_532nm_200um_5_7mW_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240322_Paral_Perp_Polarization\SiNW_Perpendicular_to_Polarization_90deg\
```

```
    Ajustado_Chip1_90deg_triangular_Row1_40nm_Col8_1200nm_532nm_2400_200_ m.txt
    ",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_10um_5_7mW_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_10  m_10 % (5
    _7mW)_1_1 s__01.txt",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_10um_5_7mW_2s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_10  m_10 % (5
    _7mW)_1_2 s__01.txt",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_10um_11mW_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_10  m_20 %
    (11mW)_1_1 s__01.txt",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_10um_11mW_2s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_10  m_20 %
    (11mW)_1_2 s__01.txt",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_200um_5_7mW_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_200  m_10 %
    (5_7mW)_1_1 s__01.txt",
#     "C4F9_Rside_Row2_Col2_Y_linescan_532nm_2400_200um_11mW_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240315_Paral_Perp_Polarization\
    SiNW_Parallel_to_Polarization\
    A_C4F9_Rside_Row2_Col2_Y_linescan_532nm_Edge_2400 (400nm)_100x_200  m_20 %
    (11mW)_1_1 s__02.txt",
#     "C4F9_on_substrate_SiO_Z_linescan_532nm_10um_5_7mW_1s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240321_Z_linescan\
    Ajustado_C4F9_on_substratum_SiO_inward_outward_4um_Step01_532nm_Edge_2400
    (400nm)_100x_10  m_10 % (5_7mW)_1 a_1 s__01.txt",
#     "C4F9_on_substrate_SiO_Z_linescan_532nm_200um_5_7mW_1s": r"C:\Users\Usuario
    \Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240321_Z_linescan\
    Ajustado_C4F9_on_substratum_SiO_inward_outward_4um_Step01_532nm_Edge_2400
    (400nm)_100x_200  m_10 % (5_7mW)_1 a_1 s__01.txt",
#     "C4F9_Rside_Row1_Col3_NTriCont_Z_linescan_ParPol_0.5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NTriContact_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_02.txt",
    "C4F9_Rside_Row1_Col3_NW_Z_linescan_Par_Pol_18mW_1s": r"C:\Users\Usuario\
        Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
        \20240202-C4F9\20240520\Green_laser\
        Ajustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
         (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
```

```
    "C4F9_Rside_Row1_Col3_NW_Z_linescan_Par_Pol_14mW_1s": r"C:\Users\Usuario\
        Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
        \20240202-C4F9\20240520\Green_laser\
        Ajustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
         (400nm)_100x_10  m_25 % (14mW)_1 a_1 s_01.txt",
    "C4F9_Rside_Row1_Col3_NW_Z_linescan_Par_Pol_11mW_1s_Mod": r"C:\Users\Usuario
        \Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
        \20240202-C4F9\20240520\Green_laser\
        ModAjustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
         (400nm)_100x_10  m_20 % (11mW)_1 a_1 s_02.txt",
    "C4F9_Rside_Row1_Col3_NW_Z_linescan_Par_Pol_5_7mW_0.5s_Mod": r"C:\Users\
        Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
        doped\20240202-C4F9\20240520\Green_laser\
        ModAjustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
         (400nm)_100x_10  m_10 % (5_7mW)_1 a_0_5 s_01.txt",
#    "C4F9_Rside_Row1_Col3_OutCont_Z_linescan_Par_Pol_0.5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_OutContact_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Rside_Row1_Col3_TriCont_Z_linescan_Par_Pol_0.5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_TriContact_Z_linescan_Inward_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_NW_Paral_Pol_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_NW_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_03.txt",
#    "C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_Subs_Paral_Pol_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_Substrate_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_03.txt",
#    "C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_NW_Paral_Pol_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_NW_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_02.txt",
#    "C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_Subs_Paral_Pol_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_Substrate_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rside_Row1_Col3_NW_X_linescan_Z_NW_Perp_Pol_532nm_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_NW_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_02.txt",
#    "C4F9_Rside_Row1_Col3_NW_X_linescan_Z_Subs_Perp_Pol_532nm_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_X_linescan_Z_Substrate_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rside_Row1_Col3_NW_Y_linescan_Z_NW_Perp_Pol_532nm_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
```

```
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_NW_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rside_Row1_Col3_NW_Y_linescan_Z_Subs_Perp_Pol_532nm_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Y_linescan_Z_Substrate_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rightside_Row1_Col3_NW_Z_linescan_Perp_Pol_532nm_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rightside_Row1_Col3_NW_Z_linescan_Perp_Pol_532nm_2_1s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240521\Green_laser\
    Ajustado_C4F9_Rightside_Row1_Col3_NW_Z_linescan_Inward_Perp_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_02.txt",
#    "C4F9_positioningmark_Y_linescan_0deg_532nm_5s": r"C:\Users\Usuario\Desktop
    \GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240603\Ajustado_C4F9_positioningmark_Y_linescan_0deg_532nm_Edge_2400 (400
    nm)_100x_10  m_100 % (57mW)_1 a_5 s_01.txt",
#    "C4F9_positioningmark_Y_linescan_15deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_15deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_01.txt",
#    "C4F9_positioningmark_Y_linescan_30deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_30deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_01.txt",
#    "C4F9_positioningmark_Y_linescan_45deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_45deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_02.txt",
#    "C4F9_positioningmark_Y_linescan_60deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_60deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_01.txt",
#    "C4F9_positioningmark_Y_linescan_75deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_75deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_01.txt",
#    "C4F9_positioningmark_Y_linescan_90deg_532nm_5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240603\
    Ajustado_C4F9_positioningmark_Y_linescan_90deg_532nm_Edge_2400 (400nm)
    _100x_10  m_100 % (57mW)_1 a_5 s_02.txt",
#    "C4F9_Y_linescan_RightEdge_0deg_532nm_5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240603\Ajustado_C4F9_Y_linescan_RightEdge_0deg_532nm_Edge_2400 (400nm)
    _100x_10  m_63 % (36mW)_1 a_5 s_01.txt",
#    "C4F9_Y_linescan_RightEdge_30deg_532nm_5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240603\Ajustado_C4F9_Y_linescan_RightEdge_30deg_532nm_Edge_2400 (400nm)
```

```
    _100x_10  m_32 % (18mW)_1 a_5 s_02.txt",
#    "C4F9_Y_linescan_RightEdge_60deg_532nm_5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240603\Ajustado_C4F9_Y_linescan_RightEdge_60deg_532nm_Edge_2400 (400nm)
    _100x_10  m_32 % (18mW)_1 a_5 s_03.txt",
#    "C4F9_Y_linescan_RightEdge_90deg_532nm_5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240603\Ajustado_C4F9_Y_linescan_RightEdge_90deg_532nm_Edge_2400 (400nm)
    _100x_10  m_32 % (18mW)_1 a_5 s_02.txt"
    # Add more files as needed
}

# Dictionary to store dataframes
data_dict = {}

for var_name, txt_file in file_var_mapping.items():
    print(f"Processing file: {txt_file}")
    data = open_file(txt_file)
    if data is not None:
        number_before_s = extract_number_before_s(var_name)
        data_values = data.iloc[1:, 2:].values.astype(float)
        data_values /= number_before_s  # Dividir solo los valores num ricos
        data.iloc[1:, 2:] = data_values  # Reemplazar los valores en data
        data_dict[var_name] = data  # Guardar los datos modificados en data_dict

# Ask for initial values of parameters
peaks = ask_for_peaks()

# Fit function
fit_params_dict, fit_results_dict, r_squared_dict, double_peak_params_dict_1,
    double_peak_params_dict_2, r_squared_double_peak_1_dict,
    r_squared_double_peak_2_dict, r_squared_double_peak_dict,
    fit_results_double_peak_1_dict, fit_results_double_peak_2_dict,
    modified_double_peak_params_1_dict = fit(data_dict, peaks)

# Call the plotting function for each set of parameters
filenames = {var_name: f"fit_plot_{var_name}" for var_name in data_dict.keys()}
     # Optional
plot_fitted_function_double_peak(data_dict, fit_params_dict, 'Single Peak',
    directory, filenames)
#plot_fitted_function_double_peak(data_dict, double_peak_params_dict_1, '
    Substrate Peak', directory, filenames)
#plot_fitted_function_double_peak(data_dict, double_peak_params_dict_2, 'NW Peak
    ', directory, filenames)
#plot_fitted_function_double_peak(data_dict, modified_double_peak_params_1_dict,
     'Modified Single Peak', directory, filenames)

# Plot the strain map
filenames = {var_name: f"stress_linescan_{var_name}" for var_name in data_dict.
    keys()}  # Optional
transform_position_linescan_and_plot_stress_linescan(data_dict, fit_params_dict,
     'Single Peak', directory, filenames)
#transform_position_linescan_and_plot_stress_linescan(data_dict,
    double_peak_params_dict_1, 'Substrate Peak', directory, filenames)
#transform_position_linescan_and_plot_stress_linescan(data_dict,
    double_peak_params_dict_2, 'NW Peak', directory, filenames)
#transform_position_linescan_and_plot_stress_linescan(data_dict,
    modified_double_peak_params_1_dict, 'Modified Single Peak', directory,
```

```
    filenames)
2009
2010
2011 # Plot the r_squared maps
2012 filenames = {var_name: f"r_squared_plot_{var_name}" for var_name in data_dict.
    keys()}  # Optional
2013 plot_r_squared_linescan(data_dict, r_squared_dict, 'Single Peak', directory,
    filenames)
2014 #plot_r_squared_linescan(data_dict, r_squared_double_peak_1_dict, 'Substrate
    Peak', directory, filenames)
2015 #plot_r_squared_linescan(data_dict, r_squared_double_peak_2_dict, 'NW Peak',
    directory, filenames)
2016 #plot_r_squared_linescan(data_dict, r_squared_double_peak_dict, 'Double Peak',
    directory, filenames)
2017
2018
2019 # Plot temperature map function from the position of the peak
2020 filenames = {var_name: f"Temp(Freq)_linescan_{var_name}" for var_name in
    data_dict.keys()}  # Optional
2021 plot_temperature_linescan_position(data_dict, fit_params_dict, 'Single Peak',
    directory, filenames, T_0=25, omega_0=520.7)
2022 #plot_temperature_linescan_position(data_dict, double_peak_params_dict_1, '
    Substrate Peak', directory, filenames, T_0=25, omega_0=520.7)
2023 #plot_temperature_linescan_position(data_dict, double_peak_params_dict_2, 'NW
    Peak', directory, filenames, T_0=25, omega_0=520.7)
2024 #plot_temperature_linescan_position(data_dict,
    modified_double_peak_params_1_dict, 'Modified Single Peak', directory,
    filenames, T_0=25, omega_0=520.7)
2025
2026 # Plot temperature map function from the FWHM of the peak
2027 filenames = {var_name: f"Temp(FWHM)_linescan_{var_name}" for var_name in
    data_dict.keys()}  # Optional
2028 plot_temperature_linescan_FWHM(data_dict, fit_params_dict, 'Single Peak',
    directory, filenames, omega_0=520.7)
2029 #plot_temperature_linescan_FWHM(data_dict, double_peak_params_dict_1, 'Substrate
     Peak', directory, filenames, omega_0=520.7)
2030 #plot_temperature_linescan_FWHM(data_dict, double_peak_params_dict_2, 'NW Peak',
     directory, filenames, omega_0=520.7)
2031 #plot_temperature_linescan_FWHM(data_dict, modified_double_peak_params_1_dict, '
    Modified Single Peak', directory, filenames, omega_0=520.7)
2032
2033
2034
2035
2036
2037 # Call the plotting function for each set of parameters, plotting them together
2038 filenames = {var_name: f"fit_plot_{var_name}" for var_name in data_dict.keys()}
2039 params_list = [fit_params_dict, double_peak_params_dict_1,
    double_peak_params_dict_2]
2040 title_suffixes = ['Single Peak Fit', 'Substrate', 'NW']
2041 colors = ['blue', 'red', 'green']
2042 plot_fitted_function_double_peak_all_params_together(data_dict, params_list,
    title_suffixes, colors, directory, filenames)
2043
2044
2045 filenames = {var_name: f"stress_linescan_{var_name}" for var_name in data_dict.
    keys()}
```

```
params_list = [fit_params_dict, double_peak_params_dict_1,
    double_peak_params_dict_2]
title_suffixes = ['Single Peak Fit', 'Substrate', 'NW']
colors = ['blue', 'red', 'green']
transform_position_linescan_and_plot_stress_linescan_all_params(data_dict,
    params_list, title_suffixes, colors, directory, filenames)


filenames = {var_name: f"r_squared_plot_{var_name}" for var_name in data_dict.
    keys()}
params_list = [r_squared_dict, r_squared_double_peak_dict]
title_suffixes = ['Single Peak Fit', 'Double Peak Fit']
colors = ['blue', 'red']
plot_r_squared_linescan_all_params(data_dict, params_list, title_suffixes,
    colors, directory, filenames)


filenames = {var_name: f"Temperature(position)_linescan_{var_name}" for var_name
     in data_dict.keys()}
params_list = [fit_params_dict, double_peak_params_dict_1,
    double_peak_params_dict_2]
title_suffixes = ['Single Peak Fit', 'Substrate', 'NW']
colors = ['blue', 'red', 'green']
plot_temperature_linescan_position_all_params(data_dict, params_list,
    title_suffixes, colors, directory, filenames, T_0=25, omega_0=520.7)


filenames = {var_name: f"Temperature(FWHM)_linescan_{var_name}" for var_name in
    data_dict.keys()}
params_list = [fit_params_dict, double_peak_params_dict_1,
    double_peak_params_dict_2]
title_suffixes = ['Single Peak Fit', 'Substrate', 'NW']
colors = ['blue', 'red', 'green']
plot_temperature_linescan_FWHM_all_params(data_dict, params_list, title_suffixes
    , colors, directory, filenames, omega_0=520.7)


# Print fitting information
#fit_info(fit_results)
```

Python Code for Maps

Listing 2: Python Code for Maps

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  5 10:25:30 2024

@author: Gines Gonzalez Guirado
"""

import os
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from lmfit import Parameters, Minimizer, fit_report
from scipy.special import wofz
```

```
from matplotlib.ticker import AutoMinorLocator, MultipleLocator
from scipy.ndimage import gaussian_filter
from scipy.optimize import fsolve

# Function to open the file
def open_file(txt_file):
    try:
        data = pd.read_table(txt_file, delim_whitespace=True, header=None)
        print(data)
        return data
    except Exception as e:
        print(f"Error opening file: {e}")
        return None

def extract_number_before_s(filename):
    # Usar regex para encontrar el n mero antes de 's'
    match = re.search(r'(\d*\.?\d+)s', filename)
    if match:
        return float(match.group(1))
    else:
        return 1  # Valor por defecto si no se encuentra n mero


# Define Voigt function using Faddeeva function
def voigt_f_not_normalised(x, x0, g_FWHM, l_FWHM):
    """
    Calculate the Voigt function.

    The Voigt function represents the convolution of a Gaussian and a Lorentzian function.

    Parameters:
        x (array-like): The input variable.
        x0 (float): The center of the Voigt function.
        g_FWHM (float): The full width at half maximum (FWHM) of the Gaussian component.
        l_FWHM (float): The full width at half maximum (FWHM) of the Lorentzian component.

    Returns:
        array-like: The Voigt function evaluated at x.

    Notes:
        The Voigt function is computed using the Faddeeva function (wofz) from the scipy library.

    References:
        - Mofreh R. Zaghloul and Ahmed N. Ali. (2011). Algorithm 916: Computing the Faddeyeva and Voigt Functions. ACM Transactions on Mathematical Software, Volume 38, Issue 2 Article No.: 15 pp 1  22  https://doi.org/10.1145/2049673.2049679
        - S. Schippers Analytical expression for the convolution of a Fano line profile with a gaussian,/ Journal of Quantitative Spectroscopy & Radiative Transfer 219 (2018) 33  36  , https://doi.org/10.1016/j.jqsrt.2018.08.003

    """

```

```
    x_i = x - x0
    alpha = np.maximum(g_FWHM / 2,1e-4)
    gamma =np.maximum(l_FWHM / 2,1e-4)
    sigma = alpha / np.sqrt(2 * np.log(2))

    # Calculate the complex argument for the Faddeeva function
    z = (x_i + 1j * gamma) / sigma / np.sqrt(2)

    # Evaluate the Faddeeva function to compute the Voigt function
    faddeeva = wofz(z)

    # Compute the real part of the Faddeeva function and normalize by the
        standard deviation
    voigt = np.real(faddeeva) / sigma / np.sqrt(2 * np.pi)

    return voigt

def voigt_f(x, x0, g_FWHM, l_FWHM, A):
    """
    Calculate the normalized Voigt function by normalizing the peak.

    Parameters:
        x (array-like): The input variable.
        x0 (float): The center of the Voigt function.
        g_FWHM (float): The full width at half maximum (FWHM) of the Gaussian
            component.
        l_FWHM (float): The full width at half maximum (FWHM) of the Lorentzian
            component.
        amplitude (float): The amplitude of the lineshape.
    Returns:
        array-like: The normalized Voigt function evaluated at x.

    """

    # Calculate the unnormalized Voigt function
    voigt = voigt_f_not_normalised(x, x0, g_FWHM, l_FWHM)

    # Find the maximum value of the Voigt function at x0
    max_value = np.maximum(voigt_f_not_normalised(x0, x0, g_FWHM, l_FWHM),1e-20)

    # Normalize the Voigt function by dividing by the maximum value
    voigt_normalized = voigt / max_value

    return voigt_normalized*A


# Function to create the intensities map
def intensities_map(data_dict):
    for var_name, data in data_dict.items():
        # Calculate the maximum value of the intensity for each pixel (excluding
            NaN values)
        max_values = np.nanmax(data.iloc[1:, 2:].values, axis=1)

        # Extract x and y coordinates
        x = [float(value) for value in data.iloc[1:, 1].values]
        y = [float(value) for value in data.iloc[1:, 0].values]

        # Create a grid of x and y coordinates
```

```
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Reshape the intensity values to match the grid
        intensity_grid = max_values.reshape((len(np.unique(y)), len(np.unique(x)
            )))

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            intensity_grid = intensity_grid[:, ::-1]

        # Create a figure and axis
        fig, ax = plt.subplots(figsize=(10, 8))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min)/4
        y_locator = abs(y_max - y_min)/4

        # Limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=10, which="major", length=5, width=1)
        ax.tick_params(axis="y", labelsize=10, which="major", length=5, width=1)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=8, which="minor", length=2.5, width
            =0.5)
        ax.tick_params(axis="y", labelsize=8, which="minor", length=2.5, width
            =0.5)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
            ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
            ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
            False)

        # Plot the intensity values using pcolormesh
        pcm = ax.pcolormesh(X, Y, intensity_grid, cmap='viridis')

        # Add colorbar and labels
        plt.colorbar(pcm, label='Intensity (counts)')
```

```
        ax.set_xlabel('X axis ($\mu m$)')
        ax.set_ylabel('Y axis ($\mu m$)')
        ax.set_title(f'Map of the Maximum Intensity of the Raman spectra in {
            var_name}')

        # Set equal aspect ratio
        ax.set_aspect('equal')

        # Show the map
        plt.show()


def intensities_map_smoothed(data_dict, sigma):
    for var_name, data in data_dict.items():
        # Calculate the maximum value of the intensity for each pixel (excluding
             NaN values)
        max_values = np.nanmax(data.iloc[1:, 2:].values, axis=1)

        # Extract x and y coordinates
        x = [float(value) for value in data.iloc[1:, 1].values]
        y = [float(value) for value in data.iloc[1:, 0].values]

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Reshape the intensity values to match the grid
        intensity_grid = max_values.reshape((len(np.unique(y)), len(np.unique(x)
            )))

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            intensity_grid = intensity_grid[:, ::-1]

        # Apply Gaussian filter to smooth the intensity grid
        intensity_smoothed = gaussian_filter(intensity_grid, sigma=sigma)

        # Create a figure and axis
        fig, ax = plt.subplots(figsize=(10, 8))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min)/4
        y_locator = abs(y_max - y_min)/4

        # Limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=10, which="major", length=5, width=1)
        ax.tick_params(axis="y", labelsize=10, which="major", length=5, width=1)
        # Set the size of minor ticks
```

```
        ax.tick_params(axis="x", labelsize=8, which="minor", length=2.5, width=0.5)
        ax.tick_params(axis="y", labelsize=8, which="minor", length=2.5, width=0.5)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=False)

        # Plot the intensity values using pcolormesh
        pcm = ax.pcolormesh(X, Y, intensity_smoothed, cmap='viridis')

        # Add colorbar and labels
        plt.colorbar(pcm, label='Intensity (counts)')
        ax.set_xlabel('X axis ($\mu m$)')
        ax.set_ylabel('Y axis ($\mu m$)')
        ax.set_title(f'Map of the Maximum Intensity of the Raman spectra in {var_name} (Smoothed)')

        # Set equal aspect ratio
        ax.set_aspect('equal')

        # Show the map
        plt.show()


# Function to ask for the number of peaks and the model type to adjust them
def ask_for_peaks():
    num_peaks = int(input("Enter the number of peaks: "))

    peaks = []

    for i in range(num_peaks):
        print(f"\nEntering parameters for peak {i+1}:")

        x0_initial = float(input("Enter initial value for peak center: "))
        Gauss_FWHM_initial = float(input("Enter initial value for FWHM Gaussian: "))
        Lorentz_FWHM_initial = float(input("Enter initial value for FWHM Lorentzian: "))
        I_initial = float(input("Enter initial value for intensity: "))
        peaks.append([x0_initial, I_initial, Gauss_FWHM_initial, Lorentz_FWHM_initial])

    return peaks

```

```
267
268 def model_f(params, x, peaks):
269     """
270     Calculate the simple model function consisting of set of peak functions
            without baseline.
271
272     Parameters:
273         params (lmfit.Parameters): The parameters object containing the fitting
                parameters.
274         x (array-like): The input variable.
275         peaks (list): The list of peak positions, and intensities.
276         model_type (list, optional): The type of peak model to use for each peak
                , the array must have the same size as peaks.
277                                      Options are 'Gaussian', 'Lorentz', 'Gauss-
                                         Lorentz', 'Voigt',
278                                      'Fano-Simply', and 'Fano-Full'. Default is '
                                         all Gaussian'.
279
280     Returns:
281         array-like: The calculated model function evaluated at x.
282     """
283
284
285     function_composed=[]
286
287     for  item in range(len(peaks)):
288         # Load the parameters for the peaks
289         function_composed.append(voigt_f(x,
290                                          params['Peak_'+str(item+1)+'_Center'],
291                                          params['Peak_'+str(item+1)+'_Gauss_FWHM'],
292                                          params['Peak_'+str(item+1)+'_Lorentz_FWHM'
                                             ],
293                                          params['Peak_'+str(item+1)+'_Intensity'])
294                                          )
295
296     peak_term=np.sum(function_composed, axis=0)
297
298     return peak_term
299
300
301 # Function to define the parameters for the fit
302 def params_to_fit(peaks):
303     """
304    Set up the parameters for a fit model.
305
306    Args:
307        peaks (list): List of peak positions and intensities.
308
309    Returns:
310        params (Parameters): Parameters object for the fit.
311    """
312     # Set up the parameters for the fit
313     params = Parameters()
314
315     for item in range(len(peaks)):
316         # Load the parameters for the peaks
317         params.add('Peak_'+str(item+1)+'_Center', value=peaks[item][0], min=500,
                max=550)
```

```
318         params.add('Peak_'+str(item+1)+'_Gauss_FWHM', value=peaks[item][2], min
                =1, max=1.001)
319         params.add('Peak_'+str(item+1)+'_Lorentz_FWHM', value=peaks[item][3],
                min=1)
320         params.add('Peak_'+str(item+1)+'_Intensity', value=peaks[item][1], min
                =30)
321
322     return params
323
324 def residual(params, position_frequency_data, data=None, peaks=None):
325     """
326     Objective function for fitting a model to data.
327
328     Parameters:
329         params (lmfit.Parameters): Model parameters to be optimized.
330         x (array-like): Independent variable data.
331         data (array-like): Dependent variable data to fit the model to.
332         peaks (list): List of peak positions and intensities.
333         model_type (list, optional): The type of peak model to use for each peak,
                the array must have the same size as peaks.
334                                     Options are 'Gaussian', 'Lorentz', 'Gauss-
                                        Lorentz', 'Voigt',
335                                     'Fano-Simply', and 'Fano-Full'. Default is '
                                        all Gaussian'.
336
337     Returns:
338         array-like: Difference between the model values and the data.
339     """
340     x = position_frequency_data
341
342     model_values = model_f(params, x, peaks)
343
344
345     return model_values - data
346
347 # Fit data with the Voigt function using lmfit
348 def fit(data_dict, peaks):
349     fit_params_dict = {}
350     fit_results_dict = {}
351     r_squared_dict = {}
352     double_peak_params_dict_1 = {}
353     double_peak_params_dict_2 = {}
354     r_squared_double_peak_dict_1 = {}
355     r_squared_double_peak_dict_2 = {}
356     r_squared_double_peak_dict = {}
357     modified_double_peak_params_dict_1 = {}
358     fit_results_double_peak_1_dict = {}
359     fit_results_double_peak_2_dict = {}
360
361     for var_name, data_to_fit in data_dict.items():
362         fit_params_list = []
363         fit_results = []  # Store fit results
364         r_squared_list = []  # Store r^2 values for each spectrum
365
366         filename = f'{var_name}'
367
368         # Extract x and y coordinates
369         y = data_to_fit.iloc[1:, 0].values.astype(float)
```

```
370         x = data_to_fit.iloc[1:, 1].values.astype(float)
371
372         # Range of frequencies in the spectrum and intensities of each spectrum
373         position_frequency_data = data_to_fit.iloc[0, 2:].values.astype(float)
374         intensity_data = data_to_fit.iloc[1:, 2:].values.astype(float)
375
376         # Adjust single peak model first
377         single_peak = [peaks[0]]
378
379         for j in range(0, len(y)):
380             # Eliminate NaN values before fitting
381             y_fit = intensity_data[j,:]
382
383             pars = params_to_fit(single_peak)
384
385             minimizer = Minimizer(residual, pars, fcn_args=(
                    position_frequency_data, y_fit, single_peak))
386             result = minimizer.least_squares(**{'xtol': 1e-5,
387                                         'gtol': 1e-5,
388                                         'ftol':1e-5,
389                                         'max_nfev':1e6})
390
391             # Calculate r^2
392             ss_residual = np.sum(result.residual ** 2)
393             ss_total = np.sum((y_fit - np.mean(y_fit)) ** 2)
394             r_squared = 1 - (ss_residual / ss_total)
395
396             # Append the fit result
397             fit_results.append(result)
398
399             # Append the r^2 value
400             r_squared_list.append(r_squared)
401
402             # Extract the fitted parameters and store them in a list
403             fit_params = [result.params[param].value for param in result.params]
404             fit_params_list.append(fit_params)
405
406         fit_pars_array = np.array(fit_params_list)
407
408         # Convert r_squared_list to array
409         r_squared_array = np.array(r_squared_list)
410
411         # Print the array of fit parameters
412         print(f"Array of Fit Parameters for {var_name}:")
413         print(fit_pars_array)
414         print(f"Array of r^2 values for {var_name}:")
415         print(r_squared_array)
416
417         # Extract parameters
418         l_FWHM = fit_pars_array[:, 2]
419         x0 = fit_pars_array[:, 0]
420
421         x0_min = np.min(x0)
422
423         A = fit_pars_array[:, 3]
424         A_min = np.min(A)
425
426         # Create a grid of x and y coordinates
```

```
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Reshape the intensity values to match the grid
        l_FWHM_grid = l_FWHM.reshape(X.shape)
        print(np.shape(l_FWHM_grid))

        def params_to_fit2(peaks, lorentz_fwhm_bounds=None, x0_min=None, A_min=
            None):
            params = Parameters()

            for item in range(len(peaks)):
                params.add(f'Peak_{item+1}_Center', value=peaks[item][0], min
                    =300, max=700)
                if x0_min and item == 1:
                    params.add(f'Peak_{item+1}_Center', value=peaks[item][0],
                        min=300, max=x0_min)
                else:
                    params.add(f'Peak_{item+1}_Center', value=peaks[item][0],
                        min=x0_min, max=700)

                params.add(f'Peak_{item+1}_Gauss_FWHM', value=peaks[item][2],
                    min=1, max=1.001)

                if lorentz_fwhm_bounds and item == 0:
                    min_lorentz = lorentz_fwhm_bounds[0]
                    max_lorentz = lorentz_fwhm_bounds[1]
                    params.add(f'Peak_{item+1}_Lorentz_FWHM', value=peaks[item
                        ][3], min=min_lorentz, max=max_lorentz)
                else:
                    params.add(f'Peak_{item+1}_Lorentz_FWHM', value=peaks[item
                        ][3], min=(lorentz_fwhm_bounds[0]/0.85))

                if A_min and item == 1:
                    params.add(f'Peak_{item+1}_Intensity', value=peaks[item][1],
                         min=30, max=A_min)
                else:
                    params.add(f'Peak_{item+1}_Intensity', value=peaks[item][1],
                         min=30)

            return params

        def get_mean_lorentz_fwhm(l_fwhm_grid, filename):
            # Determine the center column and row based on filename orientation
            if '0deg' in filename or '180deg' in filename:
                col = l_fwhm_grid.shape[1] // 2  # Half of the columns in the
                    row
                row = 2  # Second row
            elif '90deg' in filename or '270deg' in filename:
                col = 2  # Second column
                row = l_fwhm_grid.shape[0] // 2  # Half of the rows in the
                    column
            else:
                row, col = 2, l_fwhm_grid.shape[1] // 2  # Default to 0deg/180
                    deg behavior

            # Extract the neighbors
            neighbors = [
                l_fwhm_grid[row, col],
```

```
                l_fwhm_grid[row - 1, col], l_fwhm_grid[row + 1, col],
                l_fwhm_grid[row, col - 1], l_fwhm_grid[row, col + 1]
                ]

            # Calculate the mean Lorentz FWHM
            mean_lorentz_fwhm = np.mean(neighbors)
            return mean_lorentz_fwhm

        # Calculate bounds for Lorentz_FWHM for the first peak in double peak fit
        mean_lorentz_fwhm = get_mean_lorentz_fwhm(l_FWHM_grid, filename)
        min_lorentz = mean_lorentz_fwhm * 0.85
        max_lorentz = mean_lorentz_fwhm * 1.15
        lorentz_fwhm_bounds = (min_lorentz, max_lorentz)

        # Identify spectra that need a double peak fit
        if np.any(r_squared_array < 0.99):
            r_squared_min = np.min(r_squared_array)
            r_squared_max = np.max(r_squared_array)
            r_squared_diff = r_squared_max - r_squared_min
            threshold = r_squared_min + r_squared_diff * 0.9

            # Identify indices that meet both conditions
            double_peak_indices = np.where((r_squared_array < threshold) & (r_squared_array < 0.99))[0]
        else:
            double_peak_indices = []

        print(f"Double peak indices for {var_name}:")
        print(double_peak_indices)

        double_peak_params_list_1 = np.zeros_like(fit_pars_array)
        double_peak_params_list_2 = np.zeros_like(fit_pars_array)
        r_squared_double_peak_list_1 = np.zeros(len(y))
        r_squared_double_peak_list_2 = np.zeros(len(y))
        r_squared_double_peak_list = np.zeros(len(y))

        fit_results_double_peak_1 = []  # Store fit results of Peak 1
        fit_results_double_peak_2 = []  # Store fit results of Peak 2
        fit_results_double_peak = []  # Store fit results of double Peak

        for idx in double_peak_indices:
            y_fit = intensity_data[idx,:]

            pars = params_to_fit2(peaks, lorentz_fwhm_bounds, x0_min, A_min)

            minimizer = Minimizer(residual, pars, fcn_args=(position_frequency_data, y_fit, peaks))
            result = minimizer.least_squares(**{'xtol': 1e-5,
                                        'gtol': 1e-5,
                                        'ftol':1e-5,
                                        'max_nfev':1e6})

            # Calculate r^2 for double peak fit
            ss_residual = np.sum(result.residual ** 2)
            ss_total = np.sum((y_fit - np.mean(y_fit)) ** 2)
            r_squared = 1 - (ss_residual / ss_total)

```

```
            r_squared_double_peak_list[idx] = r_squared

            fit_results_double_peak.append(result)  # Append the fit result for
                Double Peak

            fit_params = [result.params[param].value for param in result.params]

            # Update single peak fit array
            double_peak_params_list_1[idx] = fit_params[:len(fit_params)//2]

            # Update double peak fit array
            double_peak_params_list_2[idx] = fit_params[len(fit_params)//2:]

            # Convert lists to dictionaries for model_f
            params_dict_1 = {f'Peak_{i+1}_{param}': double_peak_params_list_1[
                idx][j]
                             for i in range(len(peaks))
                             for j, param in enumerate(['Center', 'Gauss_FWHM',
                                 'Lorentz_FWHM', 'Intensity'])}

            params_dict_2 = {f'Peak_{i+1}_{param}': double_peak_params_list_2[
                idx][j]
                             for i in range(len(peaks))
                             for j, param in enumerate(['Center', 'Gauss_FWHM',
                                 'Lorentz_FWHM', 'Intensity'])}

            # Calculate r^2 for each peak in the double peak fit
            ss_residual_1 = np.sum((y_fit - model_f(params_dict_1,
                position_frequency_data, [peaks[0]])).flatten() ** 2)
            ss_residual_2 = np.sum((y_fit - model_f(params_dict_2,
                position_frequency_data, [peaks[1]])).flatten() ** 2)

            r_squared_1 = 1 - (ss_residual_1 / ss_total)
            r_squared_2 = 1 - (ss_residual_2 / ss_total)

            r_squared_double_peak_list_1[idx] = r_squared_1
            r_squared_double_peak_list_2[idx] = r_squared_2

            fit_results_double_peak_1.append(result)  # Append the fit result
                for Peak 1
            fit_results_double_peak_2.append(result)  # Append the fit result
                for Peak 2

        double_peak_params_array_1 = np.array(double_peak_params_list_1)
        double_peak_params_array_2 = np.array(double_peak_params_list_2)

        # Convert r_squared_double_peak_list to array
        r_squared_double_peak_array_1 = np.array(r_squared_double_peak_list_1)
        r_squared_double_peak_array_2 = np.array(r_squared_double_peak_list_2)

        r_squared_double_peak_array = np.array(r_squared_double_peak_list)

        # Check intensity of the second peak and set rows to zero if intensity <
            35
        intensity_threshold = 35
        invalid_indices = double_peak_params_array_2[:, 3] < intensity_threshold

        double_peak_params_array_1[invalid_indices] = 0
```

```
576         double_peak_params_array_2[invalid_indices] = 0
577         r_squared_double_peak_array_1[invalid_indices] = 0
578         r_squared_double_peak_array_2[invalid_indices] = 0
579         r_squared_double_peak_array[invalid_indices] = 0
580 
581         # Print the filtered arrays
582         print(f"Filtered Array of Fit Parameters Peak 1 for {var_name}:")
583         print(double_peak_params_array_1)
584         print(double_peak_params_array_1[1160:1600])
585         print(f"Filtered Array of Fit Parameters Peak 2 for {var_name}:")
586         print(double_peak_params_array_2)
587         print(double_peak_params_array_2[1160:1600])
588         print(f"Filtered Array of r^2 values Peak 1 for {var_name}:")
589         print(r_squared_double_peak_array_1)
590         print(r_squared_double_peak_array_1[1160:1600])
591         print(f"Filtered Array of r^2 values Peak 2 for {var_name}:")
592         print(r_squared_double_peak_array_2)
593         print(r_squared_double_peak_array_2[1160:1600])
594         print(f"Filtered Array of r^2 values Double Peak for {var_name}:")
595         print(r_squared_double_peak_array)
596 
597         # Replace zeros in double_peak_params_array_1 with fit_pars_array values
598         modified_double_peak_params_array_1 = np.where(
                double_peak_params_array_1 == 0, fit_pars_array,
                double_peak_params_array_1)
599 
600         # Print the modified array of fit parameters
601         print(f"Modified Array of Fit Parameters Peak 1 for {var_name}:")
602         print(modified_double_peak_params_array_1)
603 
604         # Store results in dictionaries
605         fit_params_dict[var_name] = fit_pars_array
606         fit_results_dict[var_name] = fit_results
607         r_squared_dict[var_name] = r_squared_array
608         double_peak_params_dict_1[var_name] = double_peak_params_array_1
609         double_peak_params_dict_2[var_name] = double_peak_params_array_2
610         r_squared_double_peak_dict_1[var_name] = r_squared_double_peak_array_1
611         r_squared_double_peak_dict_2[var_name] = r_squared_double_peak_array_2
612         r_squared_double_peak_dict[var_name] = r_squared_double_peak_array
613         modified_double_peak_params_dict_1[var_name] =
                modified_double_peak_params_array_1
614         fit_results_double_peak_1_dict[var_name] = double_peak_params_array_1
615         fit_results_double_peak_2_dict[var_name] = double_peak_params_array_2
616 
617     return (fit_params_dict, fit_results_dict, r_squared_dict,
            double_peak_params_dict_1,
618             double_peak_params_dict_2, r_squared_double_peak_dict_1,
                    r_squared_double_peak_dict_2,
619             r_squared_double_peak_dict, fit_results_double_peak_1_dict,
                    fit_results_double_peak_2_dict,
620             modified_double_peak_params_dict_1)
621 
622 
623 # Report of the fitting
624 def fit_info(fit_results_dict):
625     for var_name, fit_results in fit_results_dict.items():
626         print(f"Fit Results for {var_name}:")
627         for i, result in enumerate(fit_results):
```

```
            print(f"Fit Result for row {i+1}:")
            print(fit_report(result))
        print()


# Define a function to save a plot as a PNG file
def save_plot_as_png(fig, directory, filename, title_suffix):
    """
    Saves the plot as a PNG file.

    Parameters:
        fig (matplotlib.figure.Figure): Figure object to save.
        directory (str): Directory where the plot will be saved.
        filename (str): Filename for the saved plot.
        title_suffix (str): Suffix for the plot title.
    """
    # Create the directory if it doesn't exist
    os.makedirs(directory, exist_ok=True)

    # Construct the full path for saving the PNG file
    full_path = os.path.join(directory, f"{filename}_{title_suffix}.png")

    # Save the figure as a PNG file
    fig.savefig(full_path, dpi=300, bbox_inches='tight')
    print(f"Plot saved as {full_path}")
    plt.close(fig)  # Close the figure to avoid displaying it


# Function to plot r_squared results of the fitting
def plot_r_squared_map(data_dict, r_squared_dict, title_suffix, directory,
    filenames=None):
    """
    Plots the R-squared map for multiple data sets.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        r_squared_dict (dict): Dictionary of R-squared values arrays.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
    """
    for var_name, data_to_fit in data_dict.items():
        r_squared_array = r_squared_dict[var_name]

        # Extract x and y coordinates
        x = data_to_fit.iloc[1:, 1].astype(float)
        y = data_to_fit.iloc[1:, 0].astype(float)

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Reshape the r_squared values to match the grid
        r_squared_grid = r_squared_array.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            r_squared_grid = r_squared_grid[:, ::-1]

```

```
        # Find the min and max values for the color scale, ignoring values <= 0
        if np.any(r_squared_array > 0):
            r_squared_min = np.min(r_squared_array[r_squared_array > 0])
            r_squared_max = np.max(r_squared_array)
            r_squared_diff = r_squared_max - r_squared_min
            r_squared_min_cero = r_squared_min - r_squared_diff * 0.1
        else:
            r_squared_min_cero = 0
            r_squared_max = 0

        # Create a figure and axes
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
            ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
            ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
            False)

        # Plot the r_squared values
        pcm_r_squared = ax.pcolormesh(X, Y, r_squared_grid, cmap='gray', vmin=
            r_squared_min_cero, vmax=r_squared_max)
        ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
        ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
        ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)

```

```
734         # Add colorbar
735         cbar_r_squared = plt.colorbar(pcm_r_squared, ax=ax, label='$r^2$')
736         cbar_r_squared.ax.yaxis.label.set_size(24)  # Adjust colorbar label size
737         cbar_r_squared.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad
                =7)  # Adjust colorbar ticks size and distance from bar
738
739         # Set equal aspect ratio for the axis
740         ax.set_aspect('equal')
741
742         # Adjust layout
743         plt.tight_layout()
744
745         # Save plot as PNG if filenames are provided
746         if filenames:
747             filename = filenames[var_name]
748             save_plot_as_png(fig, directory, filename, title_suffix)
749
750
751 # Function to transform the map of positions (frequencies) into a strain map
752 def transform_position_map_and_plot_stress_map(data_dict, param_dict,
        title_suffix, directory, filenames=None):
753     """
754     Transforms the map of positions (frequencies) into a stress map and plots it
            for multiple data sets.
755
756     Parameters:
757         data_dict (dict): Dictionary of data sets to fit.
758         param_dict (dict): Dictionary of fitted parameters arrays.
759         title_suffix (str): Suffix for the plot titles to differentiate plots.
760
761         References:
762             - I. De Wolf (2011). Micro-Raman spectroscopy to study local
                    mechanical stress in silicon integrated ciurcuits, Volume 11,
                    https://iopscience.iop.org/article/10.1088/0268-1242/11/2/001
763             - Xu Li et. al The (2022) resolution and repeatability of stress
                    measurement by Raman and EBSD in silicon, DOI:10.1016/j.vacuum
                    .2022.111276
764     """
765     for var_name, data_to_fit in data_dict.items():
766         param_array = param_dict[var_name]
767
768         # Extract x and y coordinates
769         x = data_to_fit.iloc[1:, 1].astype(float)
770         y = data_to_fit.iloc[1:, 0].astype(float)
771
772         # Create a grid of x and y coordinates
773         X, Y = np.meshgrid(np.unique(x), np.unique(y))
774
775         wp_Si_ref = 520.7   # Frequency of reference Silicon
776
777         x0 = param_array[:, 0]
778
779         # Generar un nuevo array de estr s con la misma longitud que x0
780         stress_new = np.zeros_like(x0)
781
782         # Calcular el estr s solo para valores de x0 distintos de cero
783         if np.any(x0 > 0):
784             stress_calculated = (x0 - wp_Si_ref) * 0.434
```

```
        else:
            stress_calculated = np.zeros_like(x0)

        # Copiar los valores calculados en las posiciones correspondientes de x0
        stress_new[x0 > 0] = stress_calculated[x0 > 0]

        # Reshape the intensity values to match the grid
        stress_grid = stress_new.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            stress_grid = stress_grid[:, ::-1]

        # Create a figure and axes with 1 row and 1 column
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))

        # Set the limits of the axes based on data range
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Set the limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
            ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
            ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
            False)

        # Plot for stress
        pcm_stress = ax.pcolormesh(X, Y, stress_grid, cmap='jet')
```

```
        ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
        ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
        ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)

        # Add colorbars
        cbar_stress = plt.colorbar(pcm_stress, ax=ax, label='Stress (GPa)')
        cbar_stress.ax.yaxis.label.set_size(24)  # Adjust colorbar label size
        cbar_stress.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)
            # Adjust colorbar ticks size and distance from bar

        # Set equal aspect ratio for all axes
        ax.set_aspect('equal')

        # Adjust layout
        plt.tight_layout()

        # Save plot as PNG if filenames are provided
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)


def transform_position_map_and_plot_stress_map_smoothed(data_dict, param_dict,
    sigma, title_suffix, directory, filenames=None):
    """
    Transforms the map of positions (frequencies) into a stress map, applies
        Gaussian smoothing, and plots it for multiple data sets.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        param_dict (dict): Dictionary of arrays of fitted parameters.
        sigma (float): The standard deviation for Gaussian kernel.
        title_suffix (str): Suffix for the plot titles to differentiate plots.

        References:
            - I. De Wolf (2011). Micro-Raman spectroscopy to study local
                mechanical stress in silicon integrated ciurcuits, Volume 11,
                https://iopscience.iop.org/article/10.1088/0268-1242/11/2/001
            - Xu Li et. al The (2022) resolution and repeatability of stress
                measurement by Raman and EBSD in silicon, DOI:10.1016/j.vacuum
                .2022.111276
    """
    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

        # Extract x and y coordinates
        x = data_to_fit.iloc[1:, 1].astype(float)
        y = data_to_fit.iloc[1:, 0].astype(float)

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        wp_Si_ref = 520.7   # Frequency of reference Silicon

        x0 = param_array[:, 0]

        # Generar un nuevo array de estr s con la misma longitud que x0
        stress_new = np.zeros_like(x0)
```

```
886
887         # Calcular el estr s solo para valores de x0 distintos de cero
888         stress_calculated = np.where(x0 != 0, (x0 - wp_Si_ref) * 0.434, 0)
889
890         # Copiar los valores calculados en las posiciones correspondientes de x0
891         stress_new[x0 != 0] = stress_calculated[x0 != 0]
892
893         # Reshape the intensity values to match the grid
894         stress_grid = stress_new.reshape(X.shape)
895
896         filename = f'{var_name}'
897
898         # Invert columns if 'BarridoIzquierda' is in the filename
899         if 'BarridoIzquierda' in filename:
900             stress_grid = stress_grid[:, ::-1]
901
902         # Apply Gaussian smoothing
903         stress_grid_smoothed = gaussian_filter(stress_grid, sigma=sigma)
904
905         # Create a figure and axes with 1 row and 1 column
906         fig, ax = plt.subplots(1, 1, figsize=(8, 6))
907
908         # Set the limits of the axes based on data range
909         x_min, x_max = np.min(x), np.max(x)
910         y_min, y_max = np.min(y), np.max(y)
911
912         # To establish the major locators
913         x_locator = abs(x_max - x_min) / 2
914         y_locator = abs(y_max - y_min) / 2
915
916         # Set the limits of the axes
917         ax.set_xlim(x_min, x_max)
918         ax.set_ylim(y_min, y_max)
919         ax.invert_yaxis()  # Invert y-axis for all subplots
920
921         # Set the size of major ticks
922         ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
                =1.5)
923         ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
                =1.5)
924         # Set the size of minor ticks
925         ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
926         ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)
927
928         # Set major and minor tick locators based on subplot
929         ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
                ticks
930         ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
931         ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
                ticks
932         ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
                automatically
933
934         # Set the minimum and maximum major tick locations for x and y axes
935         ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
                False)
```

```
936         ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
                False)
937
938         # Plot for stress
939         pcm_stress = ax.pcolormesh(X, Y, stress_grid_smoothed, cmap='jet')
940         ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
941         ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
942         ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)
943
944         # Add colorbars
945         cbar_stress = plt.colorbar(pcm_stress, ax=ax, label='Stress (GPa)')
946         cbar_stress.ax.yaxis.label.set_size(24)  # Adjust colorbar label size
947         cbar_stress.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)
                # Adjust colorbar ticks size and distance from bar
948
949         # Set equal aspect ratio for all axes
950         ax.set_aspect('equal')
951
952         # Adjust layout
953         plt.tight_layout()
954
955         # Save plot as PNG if filenames are provided
956         if filenames:
957             filename = filenames[var_name]
958             save_plot_as_png(fig, directory, filename, title_suffix)
959
960
961
962 def plot_fitted_function_double_peak(data_dict, param_dict, title_suffix,
        directory, filenames=None):
963     """
964     Plots the fitted function for the double peak parameters for multiple data
            sets.
965
966     Parameters:
967         data_dict (dict): Dictionary of data sets to fit.
968         param_dict (dict): Dictionary of parameters arrays to plot.
969         title_suffix (str): Suffix for the plot titles to differentiate plots.
970         directory (str): Directory where the plots will be saved.
971         filenames (dict): Dictionary of filenames for saving the plots.
972     """
973     for var_name, data_to_fit in data_dict.items():
974         param_array = param_dict[var_name]
975
976         # Extract x and y coordinates
977         x = data_to_fit.iloc[1:, 1].astype(float)
978         y = data_to_fit.iloc[1:, 0].astype(float)
979
980         # Extract parameters
981         x0 = param_array[:, 0]
982         g_FWHM = param_array[:, 1]
983         l_FWHM = param_array[:, 2]
984         A = param_array[:, 3]
985
986         # Create a grid of x and y coordinates
987         X, Y = np.meshgrid(np.unique(x), np.unique(y))
988
989         # Calculate limits for each parameter if there are non-zero values
```

```
        def calculate_limits(param):
            if np.any(param > 0):
                param_min = np.min(param[param > 0])
                param_max = np.max(param)
                param_diff = param_max - param_min
                param_min_cero = param_min - param_diff * 0.1
                param_max_cero = param_max + param_diff * 0.1

                # Adjust param_min_cero if it's less than 0
                if param_min_cero < 0:
                    param_min_cero = 0
            else:
                param_min = 0
                param_max = 0
                param_min_cero = 0
                param_max_cero = 0

            return param_min_cero, param_max_cero, param_min, param_max

        x0_min_cero, x0_max_cero, x0_min, x0_max = calculate_limits(x0)
        g_FWHM_min_cero, g_FWHM_max_cero, g_FWHM_min, g_FWHM_max =
            calculate_limits(g_FWHM)
        l_FWHM_min_cero, l_FWHM_max_cero, l_FWHM_min, l_FWHM_max =
            calculate_limits(l_FWHM)
        A_min_cero, A_max_cero, A_min, A_max = calculate_limits(A)

        # Reshape the intensity values to match the grid
        x0_grid = x0.reshape(X.shape)
        g_FWHM_grid = g_FWHM.reshape(X.shape)
        l_FWHM_grid = l_FWHM.reshape(X.shape)
        A_grid = A.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            x0_grid = x0_grid[:, ::-1]
            g_FWHM_grid = g_FWHM_grid[:, ::-1]
            l_FWHM_grid = l_FWHM_grid[:, ::-1]
            A_grid = A_grid[:, ::-1]

        # Calculate the aspect ratio to maintain pixel size equality
        x_range = np.max(x) - np.min(x)
        y_range = np.max(y) - np.min(y)
        aspect_ratio = x_range / y_range

        # Calculate limits for the axes
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # Calculate locators for major ticks
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Define function to create and save each plot
        def create_and_save_plot(grid, vmin, vmax, title, color_label,
            file_suffix):
            fig, ax = plt.subplots(figsize=(8, 6))
```

```
            pcm = ax.pcolormesh(X, Y, grid, cmap='jet', vmin=vmin, vmax=vmax)
            ax.set_title(f'{title_suffix}', fontsize=28, pad=18)  # Adjust title pad
            ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)  # Adjust x label pad
            ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)  # Adjust y label pad
            ax.set_xlim(x_min, x_max)
            ax.set_ylim(y_min, y_max)
            ax.invert_yaxis()
            ax.set_aspect('equal')  # Maintain aspect ratio

            # Add colorbar with increased font size for label and ticks
            cbar = plt.colorbar(pcm, ax=ax)
            cbar.set_label(color_label, fontsize=24, labelpad=13)  # Adjust colorbar label size and distance from bar
            cbar.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)  # Adjust colorbar ticks size and distance from bar

            # Set the size of major ticks
            ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width=1.5)
            ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width=1.5)
            # Set the size of minor ticks
            ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
            ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

            # Set major and minor tick locators based on subplot
            ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major ticks
            ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically
            ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major ticks
            ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically

            # Set the minimum and maximum major tick locations for x and y axes
            ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=False)
            ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=False)

            plt.tight_layout()
            save_plot_as_png(fig, directory, f'{filename}_{file_suffix}', title_suffix)

        # Create and save each plot
        create_and_save_plot(x0_grid, x0_min_cero, x0_max, 'Raman shift', 'Raman shift ($cm^{-1}$)', 'Raman_shift')
        create_and_save_plot(g_FWHM_grid, g_FWHM_min, g_FWHM_max, 'Gaussian FWHM', 'Gaussian FWHM ($cm^{-1}$)', 'gaussian_FWHM')
        create_and_save_plot(l_FWHM_grid, l_FWHM_min_cero, l_FWHM_max, 'Lorentzian FWHM', 'Lorentzian FWHM ($cm^{-1}$)', 'lorentzian_FWHM')
```

```
        create_and_save_plot(A_grid, A_min_cero, A_max, 'Intensity', 'Intensity (A.U.)', 'intensity')



# Function to analyze the "derivative" of the Lorentzian FWHM
def plot_lorentzian_FWHM_derivative(data_dict, param_dict, title_suffix, directory, filenames=None):
    """
    Plots the derivative of the Lorentzian FWHM with respect to its neighbors for multiple data sets.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        param_dict (dict): Dictionary of parameters arrays to plot.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
        directory (str): Directory where the plots will be saved.
        filenames (dict): Dictionary of filenames for saving the plots.
    """
    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

        # Extract x and y coordinates
        x = data_to_fit.iloc[1:, 1].astype(float)
        y = data_to_fit.iloc[1:, 0].astype(float)

        # Extract parameters
        l_FWHM = param_array[:, 2]

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Reshape the intensity values to match the grid
        l_FWHM_grid = l_FWHM.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            l_FWHM_grid = l_FWHM_grid[:, ::-1]

        # Calculate the derivative with respect to neighbors
        l_FWHM_derivative = np.zeros_like(l_FWHM_grid)
        rows, cols = l_FWHM_grid.shape

        for i in range(rows):
            for j in range(cols):
                neighbors = []
                if i > 0:
                    neighbors.append(l_FWHM_grid[i-1, j])
                if i < rows - 1:
                    neighbors.append(l_FWHM_grid[i+1, j])
                if j > 0:
                    neighbors.append(l_FWHM_grid[i, j-1])
                if j < cols - 1:
                    neighbors.append(l_FWHM_grid[i, j+1])

                if neighbors:
```

```
                    l_FWHM_derivative[i, j] = l_FWHM_grid[i, j] - np.mean(
                        neighbors)
                else:
                    l_FWHM_derivative[i, j] = 0

        # Create a figure and axes with 1 row and 1 column
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))

        # Set the limits of the axes based on data range and calculated limits
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Set the limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
            ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
            ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
            False)

        # Plot the parameters with adjusted limits
        pcm_l_FWHM = ax.pcolormesh(X, Y, l_FWHM_derivative, cmap='jet', vmin=np.
            min(l_FWHM_derivative), vmax=np.max(l_FWHM_derivative))
        ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
        ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
        ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)

        # Add colorbar with increased font size for label and ticks
        cbar = plt.colorbar(pcm_l_FWHM, ax=ax)
        cbar.set_label('Lorentzian FWHM Derivative', fontsize=26, labelpad=13)
            # Adjust colorbar label size and distance from bar
```

```
        cbar.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)  # Adjust colorbar ticks size and distance from bar

        # Set equal aspect ratio for all axes
        ax.set_aspect('equal')

        # Adjust layout
        plt.tight_layout()

        # Save plot as PNG if filenames are provided
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)


def frecuency_to_temperature(frecuencies, T_0, omega_0):
    """
    Converts a list of frequencies to temperatures using the provided formula.

    Parameters:
    frequencies (list or np.ndarray): List or array of frequencies.
    T_0 (float): Constant T_0.
    omega_0 (float): Constant omega_0.

    Returns:
    np.ndarray: Array of temperatures.

    References:
        - Tsu R. and Gonzalez Hernandez J. (1982). Temperature dependence of silicon Raman lines. Appl. Phys. Lett. 41, 1016-1018, https://doi.org/10.1063/1.93394

    """
    frecuencies = np.array(frecuencies)
    temperatures = T_0 + (1 / (-5.4e-5)) * np.log(frecuencies / omega_0)

    #temperatures = T_0 + (1 / (-5.4e-5)) * ((frecuencies - omega_0) / omega_0)

    return temperatures

def plot_temperature_map(data_dict, param_dict, title_suffix, directory, filenames=None, T_0=25, omega_0=520.7):
    """
    Plots the temperature map for the given data sets and parameters.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        param_dict (dict): Dictionary of parameters arrays to plot.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
        directory (str): Directory where the plots will be saved.
        filenames (dict): Dictionary of filenames for saving the plots.
        T_0 (float): Constant T_0 for temperature conversion. Default is 25.
        omega_0 (float): Constant omega_0 for temperature conversion. Default is 520.7.
    """
    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

```

```
        # Extract x and y coordinates
        x = data_to_fit.iloc[1:, 1].astype(float)
        y = data_to_fit.iloc[1:, 0].astype(float)

        # Extract frequency parameter
        x0 = param_array[:, 0]

        # Convert frequencies to temperatures
        temps = frecuency_to_temperature(x0, T_0, omega_0)

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Calculate limits for temperature if there are non-zero values
        if np.any(x0 > 0):
            temps_min = np.min(temps[temps > 0])
            temps_max = np.max(temps)
            temps_diff = temps_max - temps_min
            temps_min_cero = temps_min - temps_diff * 0.1
            temps_max_cero = temps_max + temps_diff * 0.1
        else:
            temps_min_cero = 0
            temps_max = 0

        # Reshape the temperature values to match the grid
        temp_grid = temps.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            temp_grid = temp_grid[:, ::-1]

        # Create a figure and axes with 1 row and 1 column
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))

        # Set the limits of the axes based on data range and calculated limits
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Set the limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

```

```
        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=False)

        # Plot the temperature map
        pcm_temp = ax.pcolormesh(X, Y, temp_grid, cmap='jet')
        ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
        ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
        ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)

        # Add colorbar with increased font size for label and ticks
        cbar = plt.colorbar(pcm_temp, ax=ax)
        cbar.set_label('Temperature ( C )', fontsize=24, labelpad=13)  # Adjust colorbar label size and distance from bar
        cbar.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)  # Adjust colorbar ticks size and distance from bar

        # Set equal aspect ratio for the axes
        ax.set_aspect('equal')

        # Adjust layout
        plt.tight_layout()

        # Save plot as PNG if filenames are provided
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)



def l_FWHM_to_temperature(gamma, Omega):
    """
    Resuelve la ecuaci n para encontrar T dado Gamma usando m todos num ricos.

    Parameters:
        gamma (float): El valor de Gamma.
        hbar (float): La constante de Planck reducida.
        Omega (float): La frecuencia angular.
        K (float): La constante de Boltzmann.

    Returns:
        float: La temperatura T en Kelvin.

        References:
```

```
            - Menendez J. and Cardona M. (1984). Temperature dependence of the
                first Raman scattering by phonons inSi, Ge, and \alpha-Sn:
                Anharmonic effects. Physical Review B, Vol 29, N  4 , https://
                doi.org/10.1103/PhysRevB.29.2051
    """
    def equation(T):
        # Definici n de la ecuaci n a resolver f(T) = 0
        X = (gamma - 1.24) / 1.24
        term1 = np.exp(0.35 * 1.4388 * Omega / T) - 1
        term2 = np.exp(0.65 * 1.4388 * Omega / T) - 1
        return X - (1 / term1 + 1 / term2)

    T_initial = 300  # Estimaci n inicial de T en Kelvin
    T_solution = fsolve(equation, T_initial)  # Resolver la ecuaci n
    return T_solution[0]  # Devolver la soluci n encontrada

def plot_temperature_map_FWHM(data_dict, param_dict, title_suffix, directory,
    filenames=None, omega_0=None):
    """
    Plots the temperature map for the given data sets and parameters.

    Parameters:
        data_dict (dict): Dictionary of data sets to fit.
        param_dict (dict): Dictionary of parameters arrays to plot.
        title_suffix (str): Suffix for the plot titles to differentiate plots.
        directory (str): Directory where the plots will be saved.
        filenames (dict): Dictionary of filenames for saving the plots.
        omega_0 (float): Constant omega_0 for temperature conversion. Default is
            None.
    """

    for var_name, data_to_fit in data_dict.items():
        param_array = param_dict[var_name]

        # Extract x and y coordinates
        x = data_to_fit.iloc[1:, 1].astype(float)
        y = data_to_fit.iloc[1:, 0].astype(float)

        # Extract Gamma parameter (l_FWHM)
        l_FWHM = param_array[:, 2]

        # Convert Gamma to temperatures
        temps = np.array([l_FWHM_to_temperature(g, omega_0) for g in l_FWHM])

        # Create a grid of x and y coordinates
        X, Y = np.meshgrid(np.unique(x), np.unique(y))

        # Calculate limits for temperature if there are non-zero values
        if np.any(l_FWHM > 0):
            temps_min = np.min(temps[temps > 0])
            temps_max = np.max(temps)
            temps_diff = temps_max - temps_min
            temps_min_cero = temps_min - temps_diff * 0.1
            temps_max_cero = temps_max + temps_diff * 0.1
        else:
            temps_min_cero = 0
            temps_max = 0

```

```
        # Reshape the temperature values to match the grid
        temp_grid = temps.reshape(X.shape)

        filename = f'{var_name}'

        # Invert columns if 'BarridoIzquierda' is in the filename
        if 'BarridoIzquierda' in filename:
            temp_grid = temp_grid[:, ::-1]

        # Create a figure and axes with 1 row and 1 column
        fig, ax = plt.subplots(1, 1, figsize=(8, 6))

        # Set the limits of the axes based on data range and calculated limits
        x_min, x_max = np.min(x), np.max(x)
        y_min, y_max = np.min(y), np.max(y)

        # To establish the major locators
        x_locator = abs(x_max - x_min) / 2
        y_locator = abs(y_max - y_min) / 2

        # Set the limits of the axes
        ax.set_xlim(x_min, x_max)
        ax.set_ylim(y_min, y_max)
        ax.invert_yaxis()  # Invert y-axis for all subplots

        # Set the size of major ticks
        ax.tick_params(axis="x", labelsize=22, which="major", length=7.5, width
            =1.5)
        ax.tick_params(axis="y", labelsize=22, which="major", length=7.5, width
            =1.5)
        # Set the size of minor ticks
        ax.tick_params(axis="x", labelsize=18, which="minor", length=5, width=1)
        ax.tick_params(axis="y", labelsize=18, which="minor", length=5, width=1)

        # Set major and minor tick locators based on subplot
        ax.xaxis.set_major_locator(MultipleLocator(x_locator))  # Set major
            ticks
        ax.xaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically
        ax.yaxis.set_major_locator(MultipleLocator(y_locator))  # Set major
            ticks
        ax.yaxis.set_minor_locator(AutoMinorLocator())   # Set minor ticks
            automatically

        # Set the minimum and maximum major tick locations for x and y axes
        ax.set_xticks(np.arange(x_min, x_max + x_locator, x_locator), minor=
            False)
        ax.set_yticks(np.arange(y_min, y_max + y_locator, y_locator), minor=
            False)

        # Plot the temperature map
        pcm_temp = ax.pcolormesh(X, Y, temp_grid, cmap='jet', vmin=
            temps_min_cero, vmax=temps_max)
        ax.set_title(f'{title_suffix}', fontsize=28, pad=18)
        ax.set_xlabel('X axis ($\mu m$)', fontsize=24, labelpad=13)
        ax.set_ylabel('Y axis ($\mu m$)', fontsize=24, labelpad=13)

        # Add colorbar with increased font size for label and ticks
```

```
        cbar = plt.colorbar(pcm_temp, ax=ax)
        cbar.set_label('Temperature (K)', fontsize=24, labelpad=13)  # Adjust colorbar label size and distance from bar
        cbar.ax.tick_params(labelsize=22, length=7.5, width=1.5, pad=7)  # Adjust colorbar ticks size and distance from bar

        # Set equal aspect ratio for the axes
        ax.set_aspect('equal')

        # Adjust layout
        plt.tight_layout()

        # Save plot as PNG if filenames are provided
        if filenames:
            filename = filenames[var_name]
            save_plot_as_png(fig, directory, filename, title_suffix)



# Directory to save the files
# Directory for Chip 1 Red laser maps
directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\Chip1_Red_laser_maps"
# Directory for C4F9 Red laser maps
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Red_laser_maps"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Red_laser_maps_DiffDegreesPower"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_RedGreen_0deg_ScanLeftRight_Maps"
# Directory for C4F9 Green laser maps
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Green_laser_maps"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\C4F9_Green_laser_NW_Subs_map"
#directory = r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\Ajustes_con_codigo_Python\PSOI_Angles_Thickness_Red_laser"

# List of file paths and corresponding variable names
file_var_mapping = {
    "Chip1_Row1_col7_7x5um_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240126\Ajustado_Row1_col7_chip1_7x5um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1 s_1a_01.txt",
    "Chip1_Row2_col7_7x5um_1s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240126\Ajustado_Row2_col7_chip1_7x5um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1 s_1a_01.txt",
    "Chip1_Row3_col7_7x5um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240201\Ajustado_Row3_col7_chip1_7x5um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
    "Chip1_Row4_col7_7x5um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240109-CHIP1\20240201\
```

```
        Ajustado_Row4_col7_chip1_7x5um_638nm_Edge_1800 (500nm)_100x_100  m_100 %
         (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row1_col1_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240202\Ajustado_Row1_col1_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row1_col1_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240202\Ajustado_Row1_col1_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row2_col1_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240202\Ajustado_Row2_col1_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row2_col1_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240202\Ajustado_Row2_col1_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row3_col1_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240206\Ajustado_Row3_col1_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row3_col1_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240202\Ajustado_Row3_col1_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row4_col1_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240206\Ajustado_Row4_col1_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row4_col1_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240206\Ajustado_Row4_col1_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row1_col2_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240206\Ajustado_Row1_col2_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row1_col2_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240206\Ajustado_Row1_col2_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row2_col2_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240208\Ajustado_Row2_col2_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a_01.txt",
#    "C4F9_Row2_col2_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240208\Ajustado_Row2_col2_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a_01.txt",
#    "C4F9_Row3_col2_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES
    \Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240208\Ajustado_Row3_col2_C4F9_RightSide_2um_5x3um_638nm_Edge_1800 (500nm
    )_100x_100  m_100 % (20mW)_1_5 s_1 a_01.txt",
#    "C4F9_Row3_col2_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\
    GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9
    \20240208\Ajustado_Row3_col2_C4F9_RightSide_2um_10x8um_638nm_Edge_1800 (500
    nm)_100x_100  m_100 % (20mW)_1_5 s_1 a_01.txt",
```

```
#    "C4F9_Row4_col2_RightSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240223\Ajustado_Row4_col2_C4F9_Rightside_2um_5x3um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row4_col2_RightSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240223\Ajustado_Row4_col2_C4F9_Rightside_2um_10x8um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row2_col1_LeftSide_2um_5x3um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240223\Ajustado_Row2_col1_C4F9_Leftside_2um_5x3um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Row2_col1_LeftSide_2um_10x8um_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240223\Ajustado_Row2_col1_C4F9_Leftside_2um_10x8um_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_1_5 s_1 a__01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_Less_power_1.7s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240322_map_less_Power\C4F9_Rightside_2um_5x3um__Row1_Col1_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mw)_1 a_1_7 s__01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_50%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240506\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mW) _1 a_1_5 s_01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_50%power_2_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240506\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mW) _1 a_1_5 s_02.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_100%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240506\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW) _1 a_1_5 s_01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_90deg_50%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240507\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_90deg_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mW) _1 a_1_5 s_02.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_90deg_100%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240507\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_90deg_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW) _1 a_1_5 s_01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_50%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240507\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mW) _1 a_1_5 s_03.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_50%power_2_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped\20240202-C4F9\20240507\Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_638nm_Edge_1800 (500nm)_100x_100  m_50 % (10mW) _1 a_1_5 s_04.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_100%power_1.5s": r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
```

```
    doped\20240202-C4F9\20240507\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_180deg_638nm_Edge_1800 (500nm)
    _100x_100  m_100 % (20mW) _1 a_1_5 s_02.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_270deg_50%power_1.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240509\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_270deg_638nm_Edge_1800 (500nm)
    _100x_100  m_50 % (10mW)_1 a_1_5 s_50% (10mW)_01.txt",
#    "C4F9_Rightside_2um_5x3um__Row1_Col1_270deg_100%power_1.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240202-C4F9\20240509\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_270deg_638nm_Edge_1800 (500nm)
    _100x_100  m_100 % (20mW)_1 a_1_5 s_100% (20mW)_01.txt",
#    "C4F9_Rside_5x3um_R1_C1_0deg_BarridoDerecha_1.5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240514\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_BarridoDerecha_638nm_Edge_1800
     (500nm)_100x_100  m_50 % (10mW)_1 a_1_5 s_01.txt",
#    "C4F9_Rside_5x3um_R1_C1_0deg_BarridoIzquierda_1.5s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240514\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_BarridoIzquierda_638nm_Edge_1800
     (500nm)_100x_100  m_50 % (10mW)_1 a_1_5 s_02.txt",
#    "C4F9_Rside_2um_5x3um__Row1_Col1_0deg_BarDer_G_0.7s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240514\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_BarridoDerecha_532nm_Edge_1800
     (500nm)_100x_100  m_10 % (5_7mW)_1 a_0_7 s_01.txt",
#    "C4F9_Rside_2um_5x3um__Row1_Col1_0deg_BarIzq_G_0.7s": r"C:\Users\Usuario\
    Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-doped
    \20240202-C4F9\20240514\
    Ajustado_C4F9_Rightside_2um_5x3um__Row1_Col1_0deg_BarridoIzquierda_532nm_Edge_1800
     (500nm)_100x_100  m_10 % (5_7mW)_1 a_0_7 s_02.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_0deg_BarridoDerecha_G_0.5s": r"C
    :\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_0deg_BarridoDerecha_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_0deg_BarridoIzquierda_G_0.5s": r"
    C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_0deg_BarridoIzquierda_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_90deg_BarridoDerecha_G_0.5s": r"C
    :\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_90deg_BarridoDerecha_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_90deg_BarridoIzquierda_G_0.5s": r
    "C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_90deg_BarridoIzquierda_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_180deg_BarridoDerecha_G_0.5s": r"
    C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_180deg_BarridoDerecha_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_02.txt",
```

```
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_180deg_BarridoIzquierda_G_0.5s":
    r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_180deg_BarridoIzquierda_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_02.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_270deg_BarridoDerecha_G_0.5s": r"
    C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_270deg_BarridoDerecha_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_270deg_BarridoIzquierda_G_0.5s":
    r"C:\Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240516\Green_laser\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_270deg_BarridoIzquier_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_02.txt",
#    "C4F9_Rightside_2um_7x4um_Row1_Col3_Map_Z_Subs_Paral_Pol_532nm_1s": r"C:\
    Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_2um_7x4um_Row1_Col3_Map_Z_Substrate_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Rightside_2um_7x4um_Row1_Col3_NW_Map_Z_NW_Paral_Pol_532nm_1s": r"C:\
    Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240520\Green_laser\
    Ajustado_C4F9_Rightside_2um_7x4um_Row1_Col3_NW_Map_Z_NW_Paral_Pol_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_1 s_01.txt",
#    "C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_30deg_BarDer_532nm_0.5s": r"C:\
    Users\Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\
    Medidas_SiNW_P-doped\20240202-C4F9\20240603\
    Ajustado_C4F9_Leftside_2um_7x4um_Row3_Col3_NW_Map_30deg_BarridoDerecha_532nm_Edge_2400
     (400nm)_100x_10  m_32 % (18mW)_1 a_0_5 s_02.txt"
#    "PSOI2_Device1_Row2_col1_0deg_2000nm_7x5x0p2um_638nm_0.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240223-PSOI-Angulo\20240312\
    Ajustado_PSOI2_Device1_Row2_col1_0deg_2000nm_7x5x0p2um_1__0_5
    s_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_01.txt",
#    "PSOI2_Device1_Row4_col1_0deg_800nm_7x3x0p2um_638nm_0.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240223-PSOI-Angulo\20240312\
    Ajustado_PSOI2_Device1_Row4_col1_0deg_800nm_7x3x0p2um_1__0_5
    s_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_01.txt",
#    "PSOI2_Device1_Row6_col1_0deg_400nm_7x3x0p2um_638nm_0.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240223-PSOI-Angulo\20240312\
    Ajustado_PSOI2_Device1_Row6_col1_0deg_400nm_7x3x0p2um_1__0_5
    s_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_01.txt",
#    "PSOI2_Device1_Row8_col1_0deg_100nm_7x3x0p2um_638nm_0.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240223-PSOI-Angulo\20240312\
    Ajustado_PSOI2_Device1_Row8_col1_0deg_100nm_7x3x0p2um_1__0_5
    s_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_01.txt",
#    "PSOI2_Device1_Row10_col1_0deg_30nm_7x3x0p2um_638nm_0.5s": r"C:\Users\
    Usuario\Desktop\GINES\Universidad\Master\Asignaturas\TFM\Medidas_SiNW_P-
    doped\20240223-PSOI-Angulo\20240312\
    Ajustado_PSOI2_Device1_Row10_col1_0deg_30nm_7x3x0p2um_1__0_5
    s_638nm_Edge_1800 (500nm)_100x_100  m_100 % (20mW)_01.txt"
    # Add more files as needed
}

```

```
# Dictionary to store dataframes
data_dict = {}

# Open each file and save data in corresponding variable
for var_name, txt_file in file_var_mapping.items():
    print(f"Processing file: {txt_file}")
    data = open_file(txt_file)
    if data is not None:
        number_before_s = extract_number_before_s(var_name)
        data_values = data.iloc[1:, 2:].values.astype(float)
        data_values /= number_before_s  # Dividir solo los valores num ricos
        data.iloc[1:, 2:] = data_values  # Reemplazar los valores en data
        data_dict[var_name] = data  # Guardar los datos modificados en data_dict

# Parameter to smooth the fitted functions
sigma = 0.5

# Plotting the intensities of the experimental data
intensities_map(data_dict)

# Ask for initial values of parameters
peaks = ask_for_peaks()

# Fit function
fit_params_dict, fit_results_dict, r_squared_dict, double_peak_params_dict_1,
    double_peak_params_dict_2, r_squared_double_peak_1_dict,
    r_squared_double_peak_2_dict, r_squared_double_peak_dict,
    fit_results_double_peak_1_dict, fit_results_double_peak_2_dict,
    modified_double_peak_params_1_dict = fit(data_dict, peaks)

# Call the plotting function for each set of parameters
filenames = {var_name: f"fit_plot_{var_name}" for var_name in data_dict.keys()}
     # Optional
plot_fitted_function_double_peak(data_dict, fit_params_dict, 'Single Peak',
    directory, filenames)
plot_fitted_function_double_peak(data_dict, double_peak_params_dict_1, '
    Substrate layer', directory, filenames)
plot_fitted_function_double_peak(data_dict, double_peak_params_dict_2, 'NW layer
    ', directory, filenames)
plot_fitted_function_double_peak(data_dict, modified_double_peak_params_1_dict,
    'Modified Single Peak', directory, filenames)

# Plot the strain map
filenames = {var_name: f"stress_map_{var_name}" for var_name in data_dict.keys()
    }  # Optional
transform_position_map_and_plot_stress_map(data_dict, fit_params_dict, 'Single
    Peak', directory, filenames)
transform_position_map_and_plot_stress_map(data_dict, double_peak_params_dict_1,
     'Substrate layer', directory, filenames)
transform_position_map_and_plot_stress_map(data_dict, double_peak_params_dict_2,
     'NW layer', directory, filenames)
transform_position_map_and_plot_stress_map(data_dict,
    modified_double_peak_params_1_dict, 'Modified Single Peak', directory,
    filenames)


# Plot the r_squared maps
```

```
filenames = {var_name: f"r_squared_plot_{var_name}" for var_name in data_dict.keys()}  # Optional
plot_r_squared_map(data_dict, r_squared_dict, 'Single Peak', directory, filenames)
#plot_r_squared_map(data_dict, r_squared_double_peak_1_dict, 'Substrate', directory, filenames)
#plot_r_squared_map(data_dict, r_squared_double_peak_2_dict, 'NW', directory, filenames)
plot_r_squared_map(data_dict, r_squared_double_peak_dict, 'Double Peak', directory, filenames)

# Plot function to analyze the "derivative" of the Lorentzian FWHM
filenames = {var_name: f"Lorentzian_FWHM_derivative_{var_name}" for var_name in data_dict.keys()}  # Optional
plot_lorentzian_FWHM_derivative(data_dict, fit_params_dict, 'Single Peak', directory, filenames)

# Plot temperature map function from the position of the peak
filenames = {var_name: f"Temperature(position)_map_{var_name}" for var_name in data_dict.keys()}  # Optional
plot_temperature_map(data_dict, fit_params_dict, 'Single Peak', directory, filenames, T_0=25, omega_0=520.7)
plot_temperature_map(data_dict, double_peak_params_dict_1, 'Substrate layer', directory, filenames, T_0=25, omega_0=520.7)
plot_temperature_map(data_dict, double_peak_params_dict_2, 'NW layer', directory, filenames, T_0=25, omega_0=520.7)
plot_temperature_map(data_dict, modified_double_peak_params_1_dict, 'Modified Single Peak', directory, filenames, T_0=25, omega_0=520.7)

# Plot temperature map function from the FWHM of the peak
filenames = {var_name: f"Temperature(FWHM)_map_{var_name}" for var_name in data_dict.keys()}  # Optional
plot_temperature_map_FWHM(data_dict, fit_params_dict, 'Single Peak', directory, filenames, omega_0=520.7)
plot_temperature_map_FWHM(data_dict, double_peak_params_dict_1, 'Substrate layer', directory, filenames, omega_0=520.7)
plot_temperature_map_FWHM(data_dict, double_peak_params_dict_2, 'NW layer', directory, filenames, omega_0=520.7)
plot_temperature_map_FWHM(data_dict, modified_double_peak_params_1_dict, 'Modified Single Peak', directory, filenames, omega_0=520.7)

# Call the plotting function for each set of parameters
#plot_fitted_function_double_peak_smoothed(data_to_fit, fit_pars_array, sigma, 'Single Peak')
#plot_fitted_function_double_peak_smoothed(data_to_fit, double_peak_params_array_1, sigma, 'Substrate')
#plot_fitted_function_double_peak_smoothed(data_to_fit, double_peak_params_array_2, sigma, 'NW')
#plot_fitted_function_double_peak_smoothed(data_to_fit, modified_double_peak_params_array_1, sigma, 'Modified Single Peak')

# Plot the strain map
#transform_position_map_and_plot_stress_map_smoothed(data_to_fit, fit_pars_array, sigma, 'Single Peak')
#transform_position_map_and_plot_stress_map_smoothed(data_to_fit, double_peak_params_array_1, sigma, 'Substrate')
#transform_position_map_and_plot_stress_map_smoothed(data_to_fit, double_peak_params_array_2, sigma, 'NW')
```

```
#transform_position_map_and_plot_stress_map_smoothed(data_to_fit,
    modified_double_peak_params_array_1, sigma, 'Modified Single Peak')


# Print fitting information
#fit_info(fit_results)
```