Décharge capillaire

May 23, 2025

1 Exploitation de l'expérience du plasma pulsé

Table of contents

[1]:
```
# %matplotlib ipympl # for interactive plots

import imageio.v2 as imageio
```

```python
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import scipy.integrate as si
from scipy import constants as cons
import pandas as pd
import os

import sys
sys.path.append("../../xspectra/src")
from xspectra import simulation as xs_sim
from xspectra import utils as xs_utils
from xspectra import visualization as xs_vis

from mpl_toolkits.mplot3d import Axes3D
from scipy.signal import find_peaks
from copy import deepcopy
```

1.1 Protocole

Dans l’expérience précédente, nous avions observé un plasma en émission continue. Cependant, bien des plasmas n’émettent que sur un court instant, d’où l’intérêt de cette deuxième manipulation, qui vise à observer un plasma pulsé de diazote.

Pour cela on réalise le montage suivant :

La décharge a lieu dans la cellule ci-dessous :

Ici un gaz composé à **95%** de N2 et à **5%** de O2 est soumis à une tension de l’ordre de **10 kV** pour générer un plasma dans une cellule de décharge. Le tout est mis dans une cage de Faraday (cf. Figure 1) afin d’atténuer les bruits électromagnétiques dans le laboratoire. À l’aide d’une fibre optique, le plasma est observé au spectroscope, au niveau de la cathode (*high*), de l’anode (*ground*) et entre les deux (*center*). Celui-ci est centré sur la raie caractéristique **C3Π(v’ = 0) → B3Πg(v’’ = 0)** du diazote, se trouvant à = **337 nm**. Une des difficultés de l’expérience est de déclencher la caméra au bon moment. Pour cela, on récupère d’abord les différents signaux sur l’oscilloscope pour repérer les impulsions du plasma, puis on règle manuellement le déclenchement de la caméra en ajoutant un délai à partir du déclenchement externe récupéré sur le générateur fournissant la décharge dans la cellule. Pour obtenir un meilleur signal, on augmente le nombre d’acquisitions en effectuant des décharges de manière périodique à une fréquence de **10 Hz**.

D’autre part, la pression dans l’enceinte du plasma est contrôlée à la main (faute de mieux) pour la garder autour de **27 mbar** (si la pression est trop basse, il n’y a pas d’émission, tandis que si elle est trop haute, il faut plus d’énergie pour ioniser le tout). Le débit de gaz, quant à lui, est maintenu à **50 cm3/min** à l’aide d’un contrôleur de débit fonctionnant avec une boucle de rétroaction[1]. (Voir la Figure 2 pour le schéma de l’expérience).

---

1 Dans le cadre de notre expérience, nous avons utilisé un produit de Brooks Instrument.

1.2 Study of the tension inside the cable of alimentation

1.2.1 Load the data of the tension in the cable of alimentation

[2]:
```
## Load oscilloscope data
data = pd.read_csv('../data/2025-03-28-capillary-discharge/oscilloscope.txt',␣
↪skiprows=4, sep='\t')
```

[3]:
```
data["Ampl"]*=9.5e3/data["Ampl"].max() # Correcting the value due to the gain␣
↪added on the experiment
data.head()
```

[3]:

	Time	Ampl
0	-1.040082e-07	-63.972969
1	-1.038082e-07	160.820794
2	-1.036082e-07	31.986520
3	-1.034082e-07	-287.878185
4	-1.032082e-07	-71.081140

[4]:
```
oscilloscopes_pulses =␣
↪[((-10,80),(80,180)),((210,300),(300,400)),((450,550),(550,650))]
```

[5]:
```
plt.figure(figsize=(15, 7))
plt.plot(data["Time"]*1e9, data["Ampl"])
plt.xlabel("Time (ns)")
plt.ylabel("Amplitude (V)")
plt.title("Tension in the cable alimenting the capillary discharge")
plt.grid()
plt.xticks(ticks=np.arange(-100,900, 20))
plt.yticks(ticks=np.arange(-10_000,10_000, 1000))
plt.xticks(rotation=90)

for pulse in oscilloscopes_pulses:
    plt.axvspan(pulse[0][0], pulse[1][1], color='red', alpha=0.5, label="Pulse")
    plt.axvline(pulse[0][1], color='red', linestyle='--')


# plt.xlim((-34,100))
plt.savefig("res/oscilloscope.png")
plt.show()
```

Le signal de la tension dans le câble (cf. Figure ci-dessus) rend compte des impulsions auxquelles est soumis le plasma. Une impulsion - ou *pulse* - se réfléchit plusieurs fois sur les extrémités du câble, d'où les plusieurs échos observés, échos générant des décharges secondaires. On remarque une légère augmentation du bruit après le premier pic : ceci correspond à la première émission du plasma, d'où un délai d'une centaine de nanosecondes pour la première décharge.

1.2.2 Energie des pulses

Nous pouvons calculer les énergies dissipées dans le plasma à chaque pulse.

[6]:
```
impedance_cable = 50  # Ohm
energie_total = 0

results = []

for i, pulse in enumerate(oscilloscopes_pulses):
    # Filtrer les données dans l'intervalle [0, 200]
    filter_input = data[(data["Time"] * 1e9 >= pulse[0][0]) & (data["Time"] *␣
 ↪1e9 <= pulse[0][1])]
    filter_output = data[(data["Time"] * 1e9 >= pulse[1][0]) & (data["Time"] *␣
 ↪1e9 <= pulse[1][1])]

    # Calculer l'intégrale en utilisant la méthode de Simpson
    input = si.simpson(filter_input["Ampl"] ** 2, filter_input["Time"]) /␣
 ↪impedance_cable
    output = si.simpson(filter_output["Ampl"] ** 2, filter_output["Time"]) /␣
 ↪impedance_cable
    gain_energie = input - output
```

```
    energie_total += gain_energie

    # Ajouter les résultats au tableau
    results.append({
        "Pulse": i + 1,
        "Énergie Input (mJ)": input * 1e3,
        "Énergie Output (mJ)": output * 1e3,
        "Gain Énergie (mJ)": gain_energie * 1e3
    })

# Créer un DataFrame pandas
results_df = pd.DataFrame(results)


# Afficher l'énergie totale
print(f"Énergie totale : {energie_total * 1e3:4.2f} mJ")
results_df
```

```
Énergie totale : 35.06 mJ
```

[6]:

	Pulse	Énergie Input (mJ)	Énergie Output (mJ)	Gain Énergie (mJ)
0	1	42.851902	25.522590	17.329312
1	2	20.519548	5.118434	15.401113
2	3	4.233049	1.899990	2.333059

Effectuons le calcul théorique d'un échauffement homogène du gaz présent dans le tube capillaire.

[7]:

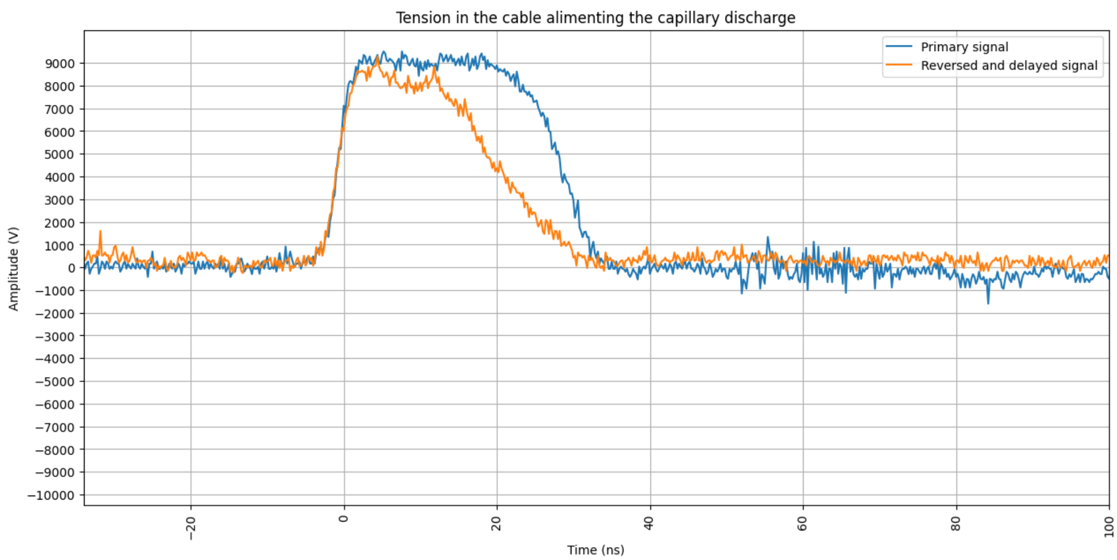
```
P=27e2 # Pa
rayon_capillaire = 0.5e-3 # m
longueur_capillaire = 5e-2 # m
V=np.pi * (rayon_capillaire**2) * longueur_capillaire # m3
T1 = 300 # K
T2 = 1200 # K

n = P * V / (cons.R * T1)  # mol
densite = cons.Avogadro * n / (V*10**6)  # nb_particules/cm3

energie_theorique = (5/2) * n * cons.R * (T2 - T1)  # J
print(f"Nombre de particules : {n:.2e} mol")
print(f"Densité : {densite:.2e} nb_particules/cm3")
print(f"Énergie théorique : {energie_theorique*1e3:4.2f} mJ")
```

```
Nombre de particules : 4.25e-08 mol
Densité : 6.52e+17 nb_particules/cm3
Énergie théorique : 0.80 mJ
```

1.2.3 Instant power

[8]:

```
delay=116 #ns

# Tentative d'alignement avec l'écho ....
plt.figure(figsize=(15, 7))
plt.plot(data["Time"]*1e9, data["Ampl"], label="Primary signal")
plt.plot(data["Time"]*1e9-delay, -data["Ampl"], label="Reversed and delayed␣
↪signal")
plt.xlabel("Time (ns)")
plt.ylabel("Amplitude (V)")
plt.title("Tension in the cable alimenting the capillary discharge")
plt.grid()
plt.savefig("res/oscilloscope.png")
plt.xticks(ticks=np.arange(-100,900, 20))
plt.yticks(ticks=np.arange(-10_000,10_000, 1000))
plt.xticks(rotation=90)

plt.legend()

plt.xlim((-34,100))
plt.show()
```

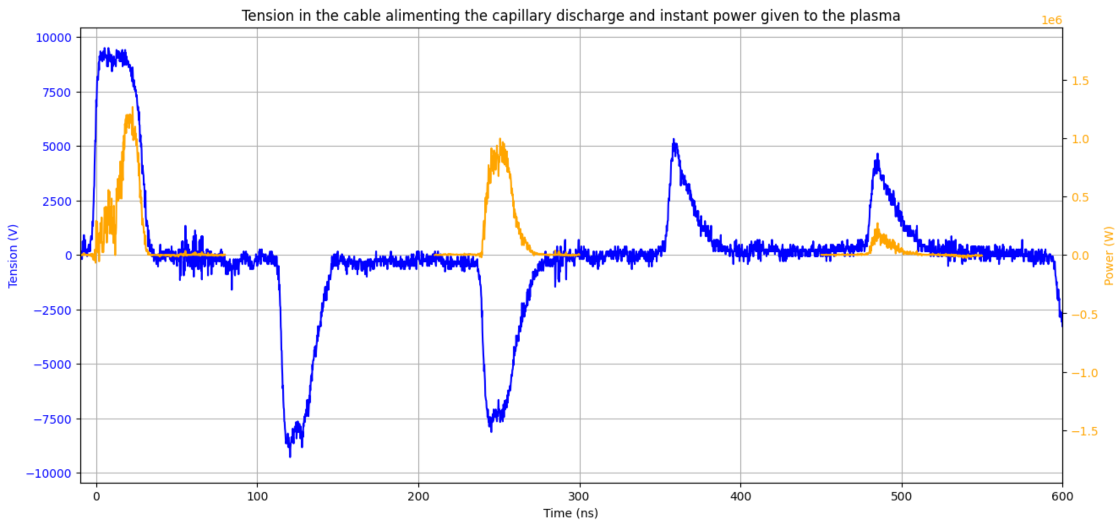


[9]:

```
time_step = (data["Time"].max() - data["Time"].min() ) / len(data)
delay_in_steps = int(delay * 1e-9 / time_step)

T = data["Time"].values[:-delay_in_steps]
tension = data["Ampl"].values[:-delay_in_steps]
```

```
echo = -data["Ampl"].values[delay_in_steps:]
instant_power = (tension**2 - echo**2) / impedance_cable
```

[10]:
```
fig, ax1 = plt.subplots(figsize=(15, 7))

# Plot tension on the first y-axis
ax1.plot(data["Time"]*1e9, data["Ampl"], color="blue")
ax1.set_ylabel("Tension (V)", color="blue")
ax1.tick_params(axis='y', labelcolor="blue")

# Center the y-axis of ax1 at 0
max_abs_ampl = max(abs(data["Ampl"].min()), abs(data["Ampl"].max())) * 1.1
ax1.set_ylim(-max_abs_ampl, max_abs_ampl)

# Create a second y-axis for instant power
ax2 = ax1.twinx()

for i, pulse in enumerate(oscilloscopes_pulses):
    filter = (T*1e9 >= pulse[0][0]) & (T*1e9 <= pulse[0][1])
    ax2.plot(T[filter]*1e9, instant_power[filter], color="orange")


ax2.set_ylabel("Power (W)", color="orange")
ax2.tick_params(axis='y', labelcolor="orange")

# Center the y-axis of ax2 at 0
max_abs_power = max(abs(instant_power.min()), abs(instant_power.max())) * 1.1
ax2.set_ylim(-max_abs_power, max_abs_power)

# Add grid, title, and labels
plt.title("Tension in the cable alimenting the capillary discharge and instant␣
 ↪power given to the plasma")
ax1.set_xlabel("Time (ns)")
ax1.grid()

plt.xlim((-10,600))
plt.savefig("res/instant_power.png")
plt.show()
```



[11]:
```
# On clear la mémoire du jupyter pour la suite
# %reset -f
```

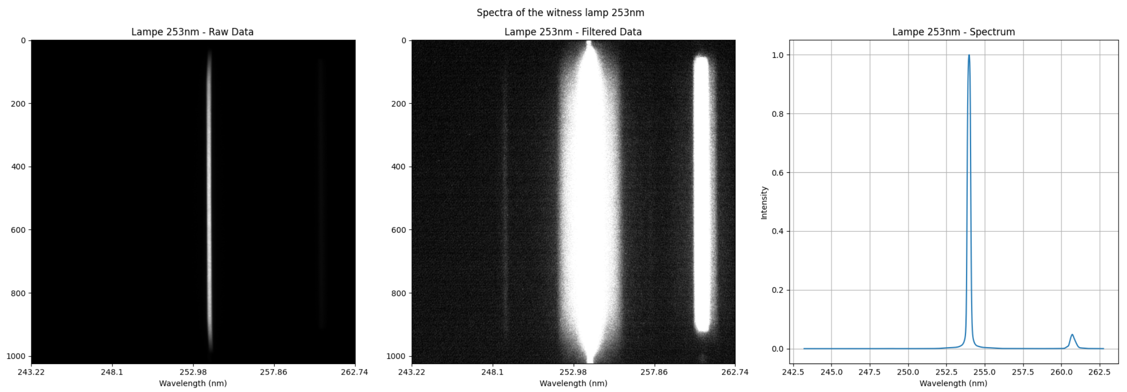
1.3 Loading the atom lights used for calibration

[12]:
```
# Récupération de manière séparée de la longueur d'onde
wavelengths = xs_utils.get_winspec_spectra("../data/
↪2025-03-28-capillary-discharge/spectra-337.txt")['"Wavelength"'].to_numpy().
↪astype(float)
wavelengths253 = xs_utils.get_winspec_spectra("../data/
↪2025-03-28-capillary-discharge/spectra-253.txt")['"Wavelength"'].to_numpy().
↪astype(float) # pour la lampe témoin
wavelengths
```

[12]:
```
array([327.39 , 327.408, 327.427, …, 346.529, 346.548, 346.567])
```

[13]:
```
# lampe_l337 = xs_utils.load_data("../data/2025-03-28-capillary-discharge/
↪lampes/g10_f150micro_r1200BLZ_w100ms_lampe337nm.SPE")
lampe_l253 = xs_utils.load_data("../data/2025-03-28-capillary-discharge/lampes/
↪g10_f150micro_r1200BLZ_w100ms_lampe253nm.SPE")

# Process lamp data
# lampe_l337_filtered = xs_utils.filter_data(lampe_l337)
lampe_l253_filtered = xs_utils.filter_data(lampe_l253,15)

# Calculate spectra
# spectra_337 = xs_utils.compute_spectra(lampe_l337)
spectra_253 = xs_utils.compute_spectra(lampe_l253)
```

```
[14]: # Create plot
fig, axes = plt.subplots(1, 3, figsize=(20, 7))



# Plot the 253nm lamp data
axes[0].imshow(lampe_l253, cmap='gray')
axes[0].set_title("Lampe 253nm - Raw Data")
axes[0].set_xticks(ticks=np.linspace(0, lampe_l253.shape[1], num=5))
axes[0].set_xticklabels(np.linspace(wavelengths253[0], wavelengths253[-1],␣
↪num=5).round(2))
axes[0].set_xlabel("Wavelength (nm)")

# Plot the filtered 253nm lamp data
axes[1].imshow(lampe_l253_filtered, cmap='gray')
axes[1].set_title("Lampe 253nm - Filtered Data")
axes[1].set_xticks(ticks=np.linspace(0, lampe_l253_filtered.shape[1], num=5))
axes[1].set_xticklabels(np.linspace(wavelengths253[0], wavelengths253[-1],␣
↪num=5).round(2))
axes[1].set_xlabel("Wavelength (nm)")

# Plot the 253nm lamp spectrum
axes[2].plot(wavelengths253, spectra_253)
axes[2].set_title("Lampe 253nm - Spectrum")
axes[2].set_xlabel("Wavelength (nm)")
axes[2].set_ylabel("Intensity")
axes[2].grid(True)

plt.suptitle("Spectra of the witness lamp 253nm")
plt.tight_layout()
# plt.savefig("res/lampe_253nm.png")
plt.show()
```



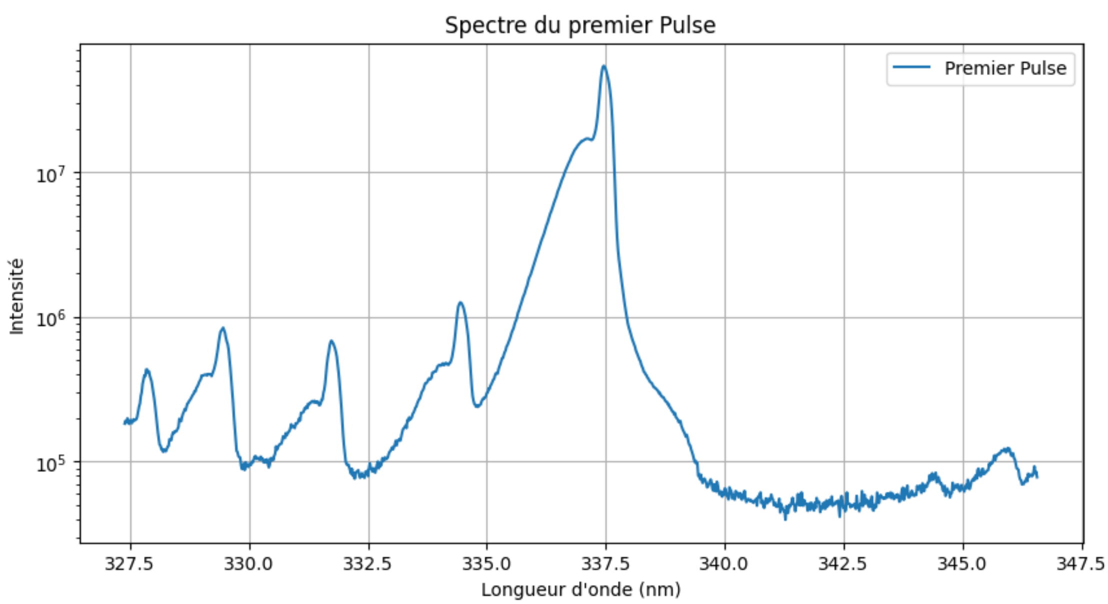
On another jupyter we fit the primary ray with a gaussian of width 0.1 nm

1.4 Loading of the data of the spectrum

[15]:
```
center_data = xs_utils.get_points_folder("../data/
 ↪2025-03-28-capillary-discharge/center")
ground_data = xs_utils.get_points_folder("../data/
 ↪2025-03-28-capillary-discharge/ground")
high_data = xs_utils.get_points_folder("../data/2025-03-28-capillary-discharge/
 ↪high")
# kyrill_data = xs_utils.get_points_folder("../data/
 ↪2025-03-28-capillary-discharge/kyrill", kyrill=True) # Data from Kyrill made␣
 ↪on week later
```

1.4.1 Background suppresion

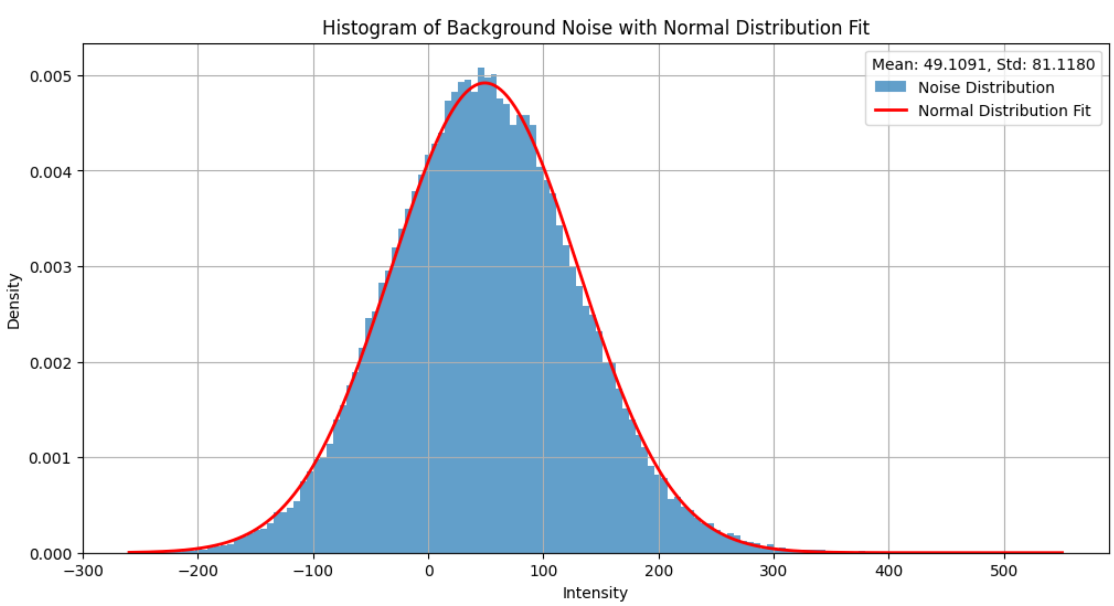
[16]:
```
spectre_pulse_1 = xs_utils.compute_spectra(center_data[1][1], False)

plt.figure(figsize=(10, 5))
plt.plot(wavelengths, spectre_pulse_1, label="Premier Pulse")
plt.xlabel("Longueur d'onde (nm)")
plt.ylabel("Intensité")
plt.title("Spectre du premier Pulse")
plt.yscale("log")
plt.legend()
plt.grid()
plt.show()
```



[17]:
```python
background = center_data[1][1][:,(341 < wavelengths) & (wavelengths < 342)].
    ↪flatten()
background.shape
```

[17]: (54272,)

[18]:
```python
## on affiche alors la distribution
# Plot the histogram and fit a normal distribution
plt.figure(figsize=(12, 6))

# Histogram
hist, bins, _ = plt.hist(background, bins="auto", alpha=0.7, density=True,␣
    ↪label='Noise Distribution')


# Fit normal distribution
mean = np.mean(background)
std = np.std(background)
x = np.linspace(min(background), max(background), 1000)
normal_dist = stats.norm.pdf(x, mean, std)


# Affichage de la courbe de la distribution normale
plt.plot(x, normal_dist, 'r-', label='Normal Distribution Fit', linewidth=2)
plt.xlabel('Intensity')
plt.ylabel('Density')
plt.title('Histogram of Background Noise with Normal Distribution Fit')
plt.legend()
plt.legend(title=f"Mean: {mean:.4f}, Std: {std:.4f}")
plt.grid()
```



[19]:
```python
def remove_background(data, bgboundaries):
    """
    Remove the background from the data using the mean of the background
    """
    # Calculate the mean of the background
    mean_background = np.mean(data[:, (bgboundaries[0] < wavelengths) &
 ↪(wavelengths < bgboundaries[1])])

    # Subtract the mean background from the data
    data_corrected = data - mean_background

    return data_corrected
```

[20]:
```python
plasmas_datas = {
    "center" : (center_data[0], [remove_background(d, (341, 342.5)) for d in
 ↪center_data[1]]), # couple (list_delays, images)
    "ground" : (ground_data[0], [remove_background(d, (341, 342.5)) for d in
 ↪ground_data[1]]),
    "high" : (high_data[0], [remove_background(d, (341, 342.5)) for d in
 ↪high_data[1]])
    # "kyrill" : (kyrill_data[0], [remove_background(d, (340, 342)) for d in
 ↪kyrill_data[1]])
}

# plasmas_datas = {
#     "center" : center_data,
#     "ground" : ground_data,
#     "high" : high_data,
#     # "kyrill" : kyrill_data
# }
```

1.4.2 Correction de la calibration

Enfin, il faut veiller à corriger la calibration en transformant `wavelengths` par une transformation linéaire. Les paramètres de cette transformation peuvent être obtenus par comparaison des têtes de bandes moléculaires puis par ajustement du spectre complet - cf jupyter précédent.

[21]:
```python
x1, x2 = 329.5,337.5 # tête de bande moléculaire 3->3 et 0->0 mesuré
y1, y2 = 328.63,337.16 # pic 3->3 et 0->0 théorique

scale_assumption = (y2 - y1) / (x2 - x1)
decalage_assumption = y1 - scale_assumption * x1

print(f"Scale assumption: {scale_assumption}")
print(f"Decalage assumption: {decalage_assumption}")
```

```
final_stretch_factor = scale_assumption
final_bias = -22.68
```

```
Scale assumption: 1.0662500000000037
Decalage assumption: -22.699375000001226
```

[22]:
```
W = final_stretch_factor * wavelengths + final_bias # wavelenths corrected
```

1.5 Quelques fonctions de traitement pour les prochains résultats

[23]:
```
pulses = np.array([(120,150),(355,380),(600,610)])
pulses
```

[23]:
```
array([[120, 150],
       [355, 380],
       [600, 610]])
```

[24]:
```
index_pulses_center = [np.abs(plasmas_datas["center"][0]-delay).argmin() for␣
 ↪delay in [130, 370, 600]]
c_image_pulses = [plasmas_datas["center"][1][i] for i in index_pulses_center]
c_spectrum_pulses = [xs_utils.compute_spectra(plasmas_datas["center"][1][i],␣
 ↪True) for i in index_pulses_center]
```

[25]:
```
def get_filter(data, index_pulse):
    """
    Get the filter for the data
    """
    return (pulses[index_pulse][0] <= data[0]) & (data[0] <=␣
 ↪pulses[index_pulse][1])
```

[26]:
```
def is_inside_pulse(delay, pulse):
    """
    Check if the data is inside the pulse
    """
    return (pulse[0] <= delay) & (delay <= pulse[1])

def is_inside_one_of_the_pulses(delay):
    """
    Check if the data is inside one of the pulses
    """
    return np.any([is_inside_pulse(delay, pulse) for pulse in pulses])
```

[27]:
```
# On va faire beaucoup de graphiques pour différentes situations donc on met ça␣
 ↪sous forme de fonction
```

```python
def plot_spectra_and_simulation(simulation_spectra, ax_lin, ax_log, epsilon=1e-3, fit_areas=[], colors = ['blue', 'orange', 'green'], simulation_labels=None, title="Spectres et simulations",xlims=(333, 339), decalages=None):
    if decalages is None:
        decalages = [0]*len(simulation_spectra)

    # Graphique avec échelle linéaire
    for i, (observed, simulated) in enumerate(zip(c_spectrum_pulses, simulation_spectra)):
        ax_lin.plot(W, observed, label=f'Spectre {i + 1}', c=colors[i])
        if simulation_labels is not None:
            label_sim = simulation_labels[i]
        ax_lin.plot(W-decalages[i], simulated, label=label_sim, linestyle='--', c=colors[i])


    ax_lin.set_ylabel('Intensité (linéaire)')
    ax_lin.set_title(title+ " échelle linéaire")
    ax_lin.legend()
    ax_lin.set_xlim(xlims)
    ax_lin.grid()

    # Graphique avec échelle logarithmique
    for i, (observed, simulated) in enumerate(zip(c_spectrum_pulses, simulation_spectra)):
        ax_log.plot(W, [x if x > epsilon else epsilon for x in observed], label=f'Spectre {i + 1}', c=colors[i])
        if simulation_labels is not None:
            label_sim = simulation_labels[i]
        ax_log.plot(W-decalages[i], [x if x > epsilon else epsilon for x in simulated], label=label_sim, linestyle='--', c=colors[i])


    if len(fit_areas) > 0:
        for ax in [ax_lin, ax_log]:
            for i, (start, end) in enumerate(fit_areas):
                if i==0:
                    ax.axvspan(start, end, color='r', alpha=0.5, label='Limites fit')
                else:
                    ax.axvspan(start, end, color='r', alpha=0.5)

    ax_log.set_yscale('log')
    ax_log.set_xlabel('Longueur d\'onde (nm)')
    ax_log.set_ylabel('Intensité (logarithmique)')
```

```
    ax_log.set_title(title+" échelle logarithmique")
    ax_log.legend()
    ax_log.set_xlim(xlims)
    ax_log.grid()
```

[28]:
```
colors_places = {
    "center": "green",
    "ground": "red",
    "high": "black",
    "kyrill": "blue"
}
```

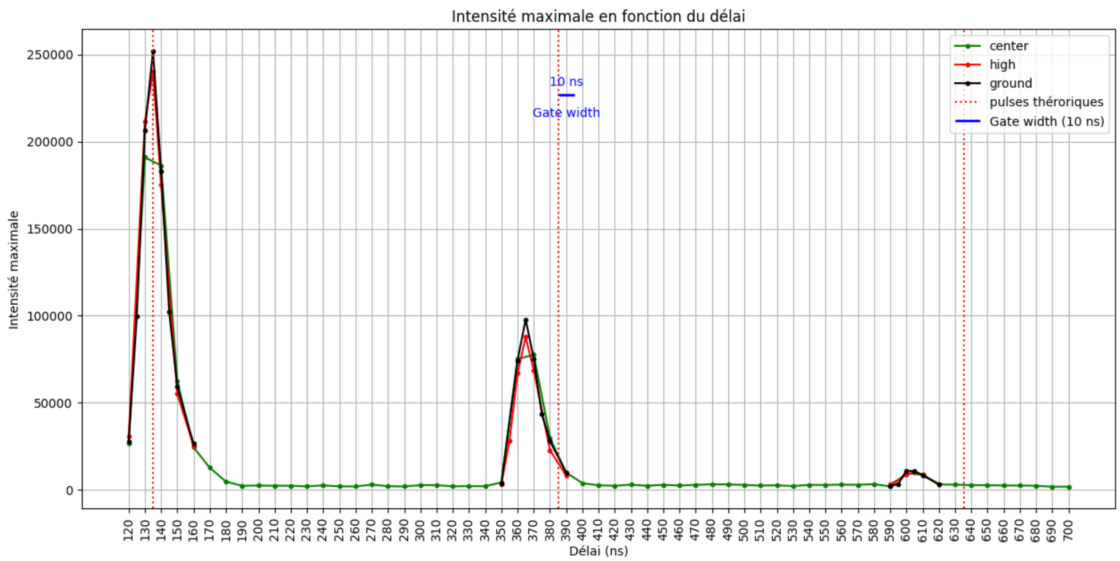
1.6 Study of the maxima as a function of time

[29]:
```
# Le câble faisant environ 25 m, on peut estimer le délai entre deux pulses

epsilon_pol = 2.25 # epsilon du polyéthylène
v = cons.c / np.sqrt(1*2.25)
L=25
D_theorique = 2*L / v * 10**9
print(f"Délai théorique entre les deux pulses: {D_theorique:.2f} ns")
```

```
Délai théorique entre les deux pulses: 250.17 ns
```

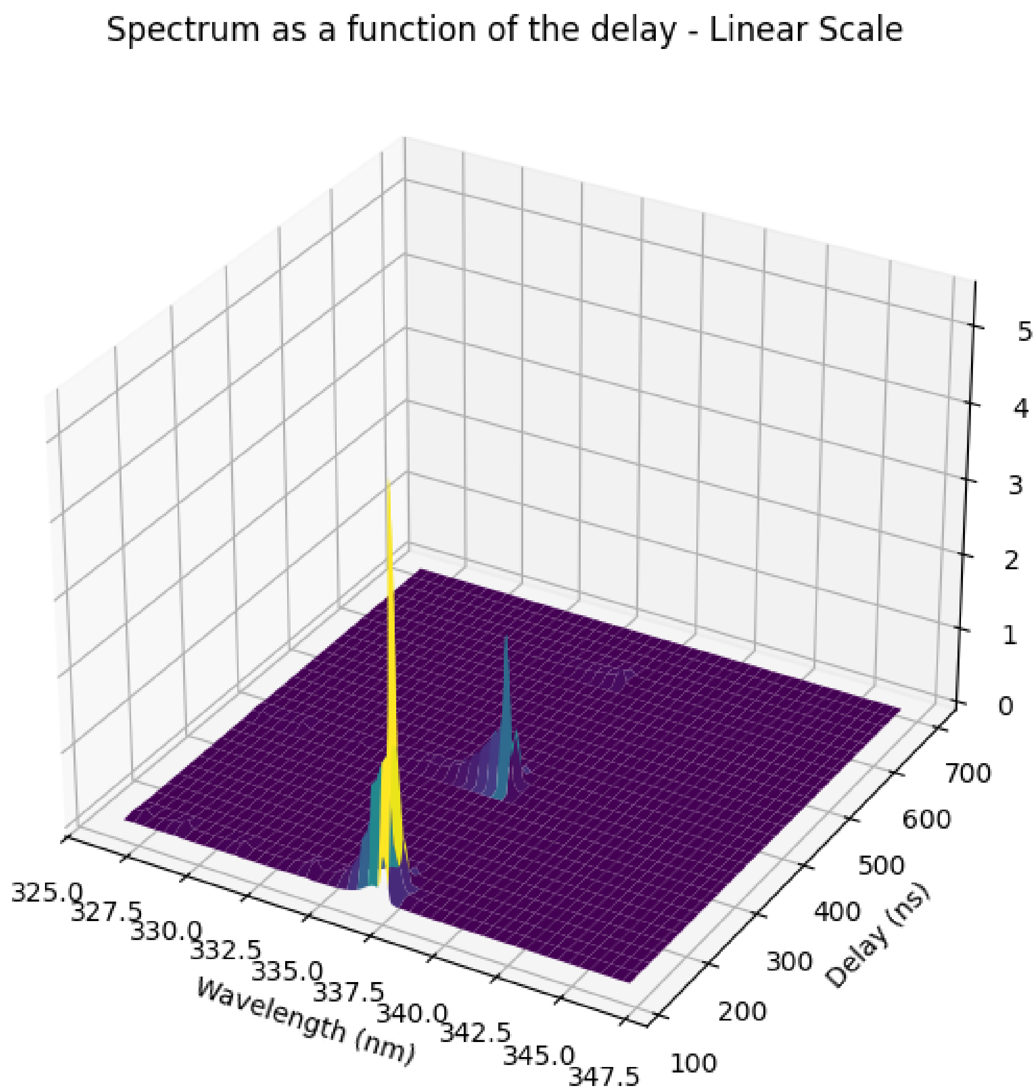
[30]:
```
plt.figure(figsize=(15,7))

maxima = {}
for name_serie, (delays, data) in plasmas_datas.items():
    maxima[name_serie] = [np.max(d) for d in data]
    maxima[name_serie] = np.array(maxima[name_serie])

plt.plot(plasmas_datas["center"][0], maxima["center"], ".-", color="green",␣
 ↪label="center")

plt.plot(plasmas_datas["high"][0][plasmas_datas["high"][0]<200],␣
 ↪maxima["high"][plasmas_datas["high"][0]<200], ".-", color="red",␣
 ↪label="high")
plt.plot(plasmas_datas["high"][0][(200<plasmas_datas["high"][0]) &␣
 ↪(plasmas_datas["high"][0]<500)],␣
 ↪maxima["high"][(200<plasmas_datas["high"][0]) &␣
 ↪(plasmas_datas["high"][0]<500)], ".-", color="red")
plt.plot(plasmas_datas["high"][0][500<plasmas_datas["high"][0]],␣
 ↪maxima["high"][500<plasmas_datas["high"][0]], ".-", color="red")

plt.plot(plasmas_datas["ground"][0][plasmas_datas["ground"][0]<200],␣
 ↪maxima["ground"][plasmas_datas["ground"][0]<200], ".-", color="black",␣
 ↪label="ground")
```

```python
plt.plot(plasmas_datas["ground"][0][(200<plasmas_datas["ground"][0]) & (plasmas_datas["ground"][0]<500)], maxima["ground"][(200<plasmas_datas["ground"][0]) & (plasmas_datas["ground"][0]<500)], ".-", color="black")
plt.plot(plasmas_datas["ground"][0][500<plasmas_datas["ground"][0]], maxima["ground"][500<plasmas_datas["ground"][0]], ".-", color="black")

plt.ylabel("Intensité maximale")
plt.xlabel("Délai (ns)")

plt.title("Intensité maximale en fonction du délai")

premier_pulse=135

plt.axvline(x=premier_pulse, color="red", linestyle=":", label="pulses théroriques")
plt.axvline(x=premier_pulse+D_theorique, color="red", linestyle=":")
plt.axvline(x=premier_pulse+2*D_theorique, color="red", linestyle=":")

# Add a 10 nanoseconds scale bar
plt.hlines(y=np.max(maxima["ground"]) * 0.9, xmin=premier_pulse+D_theorique, xmax=premier_pulse +D_theorique+ 10, color="blue", linewidth=2, label="Gate width (10 ns)")
plt.text(premier_pulse+D_theorique + 5, np.max(maxima["ground"]) * 0.92, "10 ns", color="blue", ha="center")
plt.text(premier_pulse+D_theorique + 5, np.max(maxima["ground"]) * 0.85, "Gate width", color="blue", ha="center")


plt.grid()
plt.xticks(center_data[0], rotation=90)
plt.legend()
plt.legend()
plt.savefig("./res/intensite_max.png")
plt.show()
```

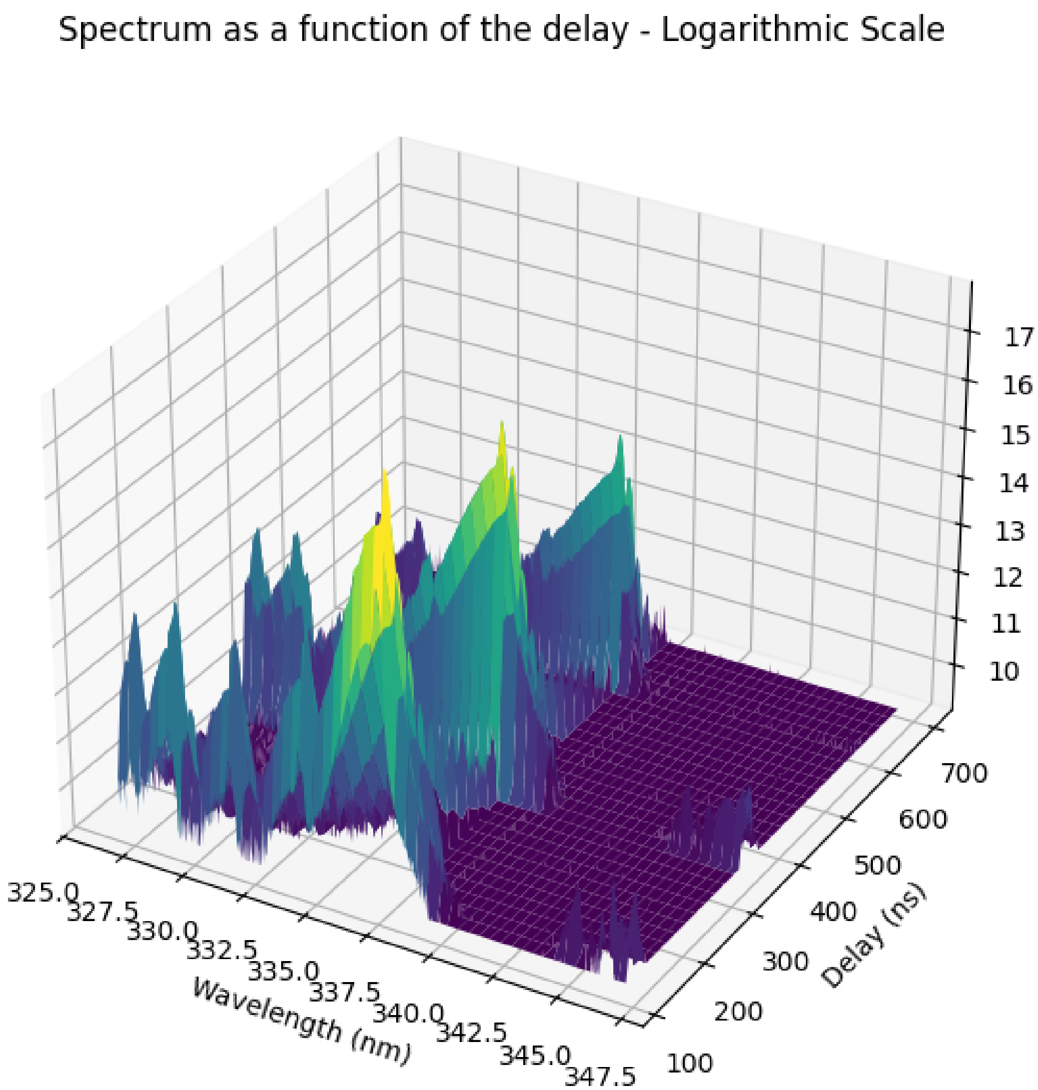


[31]:
```
epsilon = 1e4  # borne inférieure pour le log
S = np.array(list(map(lambda x: xs_utils.compute_spectra(deepcopy(x), False),
 ↪plasmas_datas["center"][1])))
S = np.array([[max(epsilon, x) for x in s] for s in S])
```

[32]:
```
# Premier graphique : Échelle linéaire
fig1 = plt.figure(figsize=(10, 7))
ax1 = fig1.add_subplot(111, projection='3d')

delays = plasmas_datas["center"][0]
X, Y = np.meshgrid(W, delays)
Z = S

ax1.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
ax1.set_xlabel('Wavelength (nm)')
ax1.set_ylabel('Delay (ns)')
ax1.set_zlabel('Intensity')
ax1.set_title('Spectrum as a function of the delay - Linear Scale')

# plt.subplots_adjust(left=0.3, right=0.6)

plt.savefig("./res/3dplot_linear.png")
plt.show()
```



[33]:
```
# Deuxième graphique : Échelle logarithmique
fig2 = plt.figure(figsize=(10, 7))
ax2 = fig2.add_subplot(111, projection='3d')

ax2.plot_surface(X, Y, np.log(Z), cmap='viridis', edgecolor='none')
ax2.set_xlabel('Wavelength (nm)')
ax2.set_ylabel('Delay (ns)')
ax2.set_zlabel('Log(Intensity)')
ax2.set_title('Spectrum as a function of the delay - Logarithmic Scale')

plt.savefig("./res/3dplot_logarithmic.png")
plt.show()
```

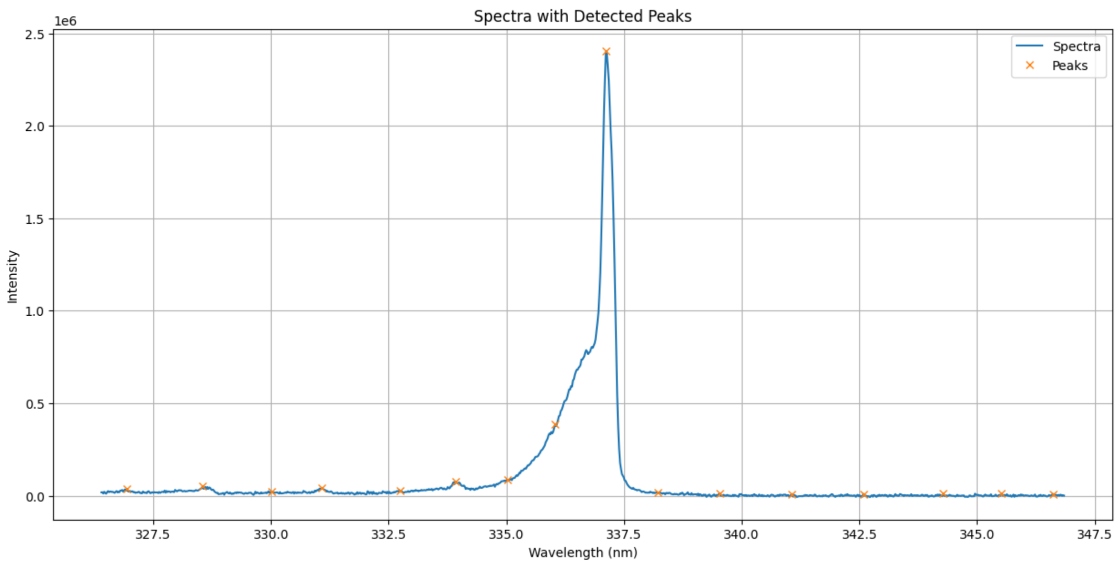


1.7 Extraction of the temperatures

1.7.1 Température vibrationnelle par ratio

[34]:
```
# Repérer les pics dans le spectre
spectra = xs_utils.compute_spectra(plasmas_datas["center"][1][5], False)
peaks, properties = find_peaks(spectra, height=0.01, distance=50)

# Afficher les résultats
plt.figure(figsize=(15, 7))
plt.plot(W, spectra, label="Spectra")
plt.plot(W[peaks], spectra[peaks], "x", label="Peaks")
plt.xlabel("Wavelength (nm)")
```

```
plt.ylabel("Intensity")
plt.title("Spectra with Detected Peaks")
plt.legend()
plt.grid()
plt.show()

# Afficher les longueurs d'onde des pics détectés
print(f"Indices des pics détectés : {peaks}")
print("Detected peaks at wavelengths:", W[peaks])
```




```
Indices des pics détectés : [  27  107  180  233  316  375  430  481  535  590
656  733  809  894
  956 1011]
Detected peaks at wavelengths: [326.9412425  328.547015   330.00991
331.07296125 332.735245
 333.91665    335.01702    336.036355   337.1154     338.21470375
 339.53258875 341.069055   342.5852625  344.2784675  345.513185
 346.60822375]
```

On repère ainsi le pic dominant en $337.5nm$ ainsi que des pics secondaires. On prend $334.44nm$ comme seconde référence.

En regardant le spectre théorique, on obtient que : - le pic dominant à $\lambda_1 = 337.5\,nm$ correspond à la transition $C^3\Pi(\nu' = 0) \to B^3\Pi_g(\nu'' = 0)$ - le pic dominant à $\lambda_2 = 334.44\,nm$ correspond à la transition $C^3\Pi(\nu' = 1) \to B^3\Pi_g(\nu'' = 1)$

En suivant ainsi les formules théoriques développées dans le jupyter de simulation, on a le rapport d'émission suivant :

$$r = \frac{\epsilon_1}{\epsilon_2} = \frac{n_1\nu_1}{n_2\nu_2} = \frac{g_{e1}(2J_1+1)e^{-\frac{T_{e1}}{kT_{el}}-\frac{G(\nu_1)}{kT_{vib}}-\frac{F(J_1)}{kT_{rot}}}}{g_{e2}(2J_2+1)e^{-\frac{T_{e2}}{kT_{el}}-\frac{G(\nu_2)}{kT_{vib}}-\frac{F(J_2)}{kT_{rot}}}}\frac{\nu_1}{\nu_2}$$

En éliminant les dégénérescences électroniques qui sont égales, ainsi que l'effet des rotations, on aboutit à :

$$r_{12} = \frac{\nu_1}{\nu_2}\exp\left(\frac{T_{e2}-T_{e1}}{kT_{el}} + \frac{G(\nu_2)-G(\nu_1)}{kT_{vib}}\right)$$

Puis, puisque l'on part du même niveau d'énergie électrique pour les deux ($C^3\Pi$), on a $T_{e1} = T_{e2}$, d'où :

$$r_{12} = \frac{\nu_1}{\nu_2}\exp\left(\frac{G(\nu_2)-G(\nu_1)}{kT_{vib}}\right)$$

Sinon on peut utiliser un troisième pic : $\lambda_3 = 331.735\,nm$ correspondant à la transition $C^3\Pi(\nu' = 2) \rightarrow B^3\Pi_g(\nu'' = 2)$ pour trouver les deux inconnues.

On a pas un spectre assez nette pour aller regarer les niveaux rotationnels.

$$T_{vib} = \frac{G(\nu_2)-G(\nu_1)}{k\ln\left(r_{12}\cdot\frac{\lambda_1}{\lambda_2}\right)}$$

En utilisant les valeurs des longueurs d'onde $\nu_1 = 337.5\,nm$ et $\nu_2 = 334.44\,nm$, ainsi que le rapport r_{12} calculé à partir des intensités des pics correspondants dans le spectre mesuré, on peut déterminer T_{vib}.

Théoriquement $G(\nu_2 = 1) - G(\nu_1 = 0) = 5.973\times10^{-20} - 2.019\times10^{-20}\,J = 3.954\times10^{-20}J$

Traçons l'évolution de la température.

[35]:
```
i1, i2 = peaks[np.argmin(abs(W[peaks]-337.5))], peaks[np.
↪argmin(abs(W[peaks]-334.4))]
i1, i2
```

[35]:
```
(np.int64(535), np.int64(375))
```

[36]:
```
plasmas_datas.keys()
```

[36]:
```
dict_keys(['center', 'ground', 'high'])
```

[37]:
```
# Calcul des températures pour chaque série
t_vib_by_ratio = {
    name : np.array([xs_utils.compute_t_vib_by_ratio(xs_utils.
↪compute_spectra(d), i1=i1, i2=i2) if is_inside_one_of_the_pulses(delay) else␣
↪0 for delay, d in zip(delays, data)]) for name, (delays, data) in␣
↪plasmas_datas.items()
}
```

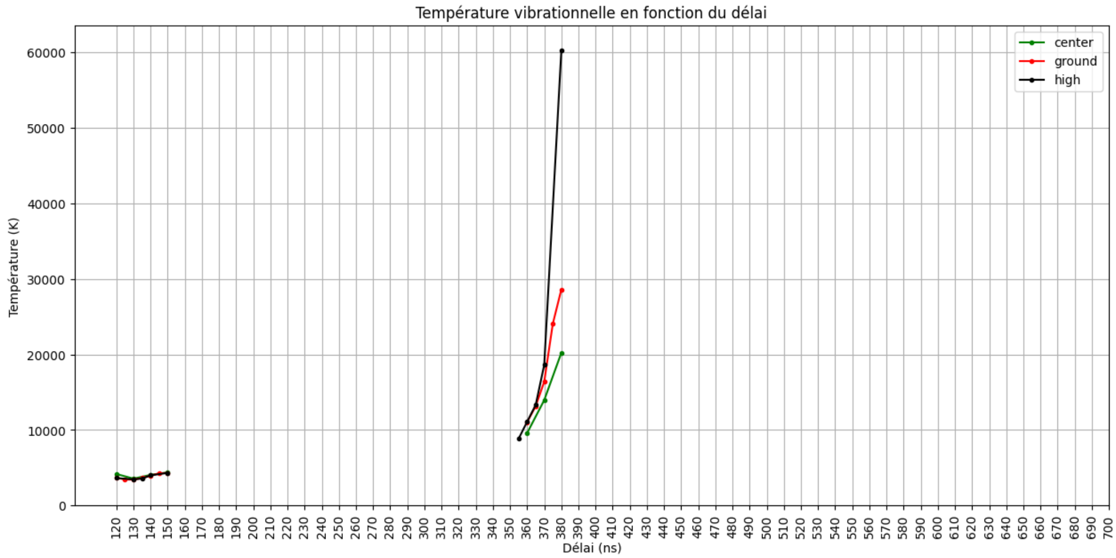
```
# Affichage des résultats
print("Center Temperatures:", t_vib_by_ratio["center"])
```

```
Center Temperatures: [ 4165.14605947  3551.15718714  4043.16000023
4383.05895891
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
  9585.4699657  13950.05060317 20216.25381843     0.
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
     0.             0.             0.             0.
 -5442.60656254 -4851.58760195     0.             0.
     0.             0.             0.             0.
     0.             0.             0.          ]
```

On s'aperçoit que notre méthode ne fonctionne pas du tout entre les pulses. En effet, puisqu'il n'y a pas de plasma, elle perd son sens, menant à des résultats incohérents. C'est pour cela que l'on se restreint par la suite aux pulses.

[38]:
```
plt.figure(figsize=(15, 7))

# Plot center temperatures
for name in t_vib_by_ratio.keys():
    filter_1, filter_2, filter_3 = get_filter(plasmas_datas[name], 0),␣
 ↪get_filter(plasmas_datas[name], 1), get_filter(plasmas_datas[name], 2)
    plt.plot(plasmas_datas[name][0][filter_1], t_vib_by_ratio[name][filter_1],␣
 ↪".-", color=colors_places[name], label=name)
    plt.plot(plasmas_datas[name][0][filter_2], t_vib_by_ratio[name][filter_2],␣
 ↪".-", color=colors_places[name])
    plt.plot(plasmas_datas[name][0][filter_3], t_vib_by_ratio[name][filter_3],␣
 ↪".-", color=colors_places[name])


plt.ylabel("Température (K)")
plt.xlabel("Délai (ns)")

plt.title("Température vibrationnelle en fonction du délai")
plt.ylim(bottom=0)
plt.grid()
plt.xticks(plasmas_datas["center"][0], rotation=90)
plt.legend()
plt.savefig("./res/temperature_vib_vs_delay.png")
```

```
plt.show()
```

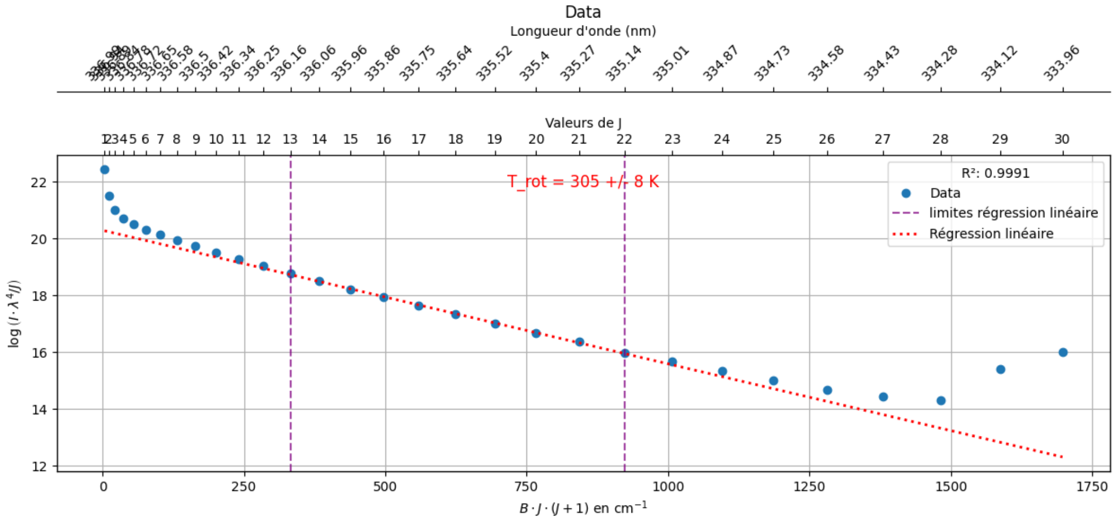


1.7.2 Extraction of the rotational temperature - Méthode analytique

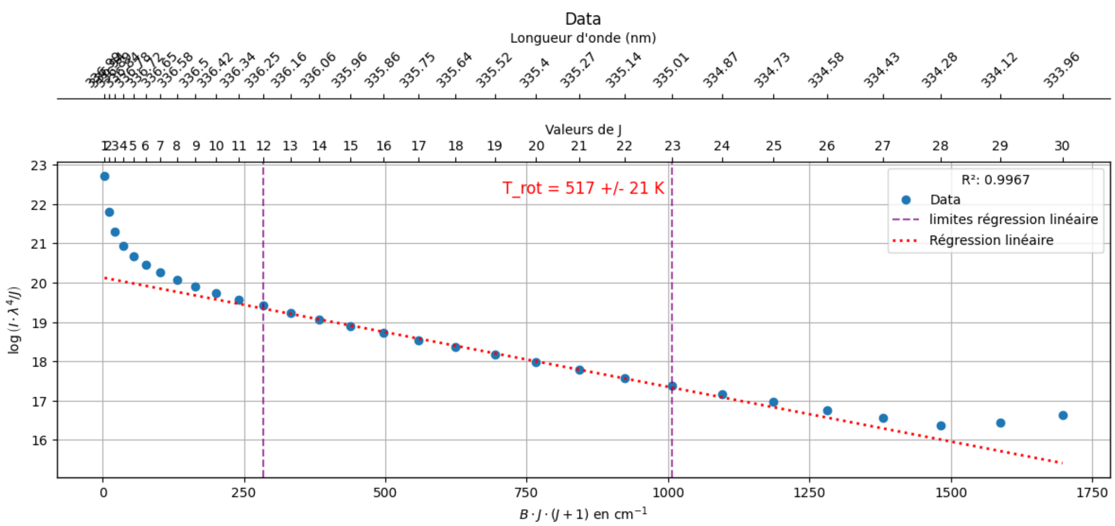
[39]:
```
def get_temp_rot_1(data):
    spectra = xs_utils.compute_spectra(data, True)
    return xs_utils.compute_Trot_with_branch(W, spectra, J_range=(12, 23),␣
 ↪certainty=0.95)




t_rot_analytic = {
    name : np.array([xs_utils.compute_Trot_with_branch(W, xs_utils.
 ↪compute_spectra(d), J_range=(12, 23), certainty=0.95) if␣
 ↪is_inside_one_of_the_pulses(delay) else (0,0) for delay, d in zip(delays,␣
 ↪data)]) for name, (delays, data) in plasmas_datas.items()
}
```

On vérifie la régression pour une valeur

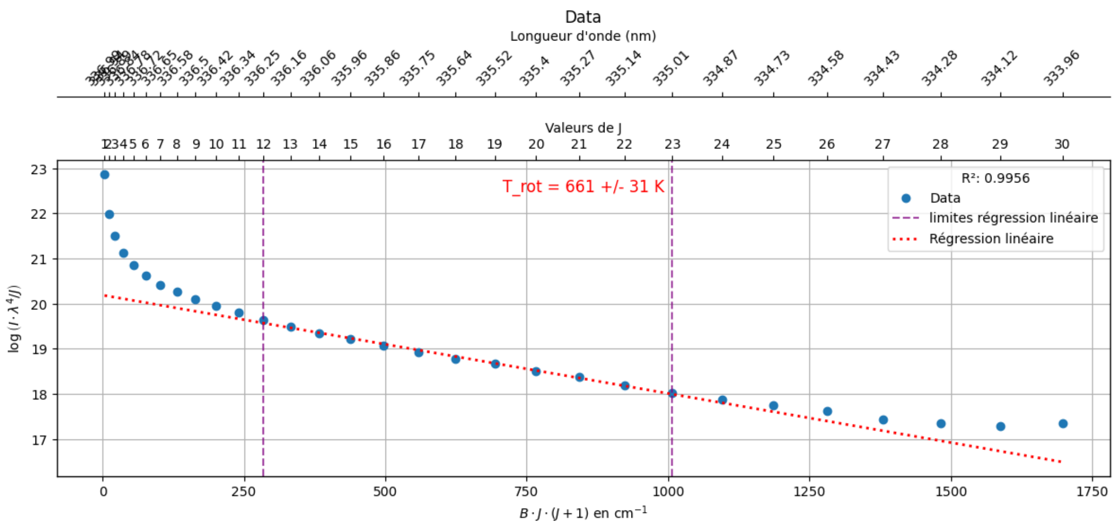
[40]:
```
xs_vis.show_result_calculation_Trot(W, c_spectrum_pulses[0], J_range=(13, 22),␣
 ↪certainty=0.95, max_J=30) # First pulse
```



[41]:
```
xs_vis.show_result_calculation_Trot(W, c_spectrum_pulses[1], J_range=(12, 23),
    certainty=0.95, max_J=30) # Second pulse
```



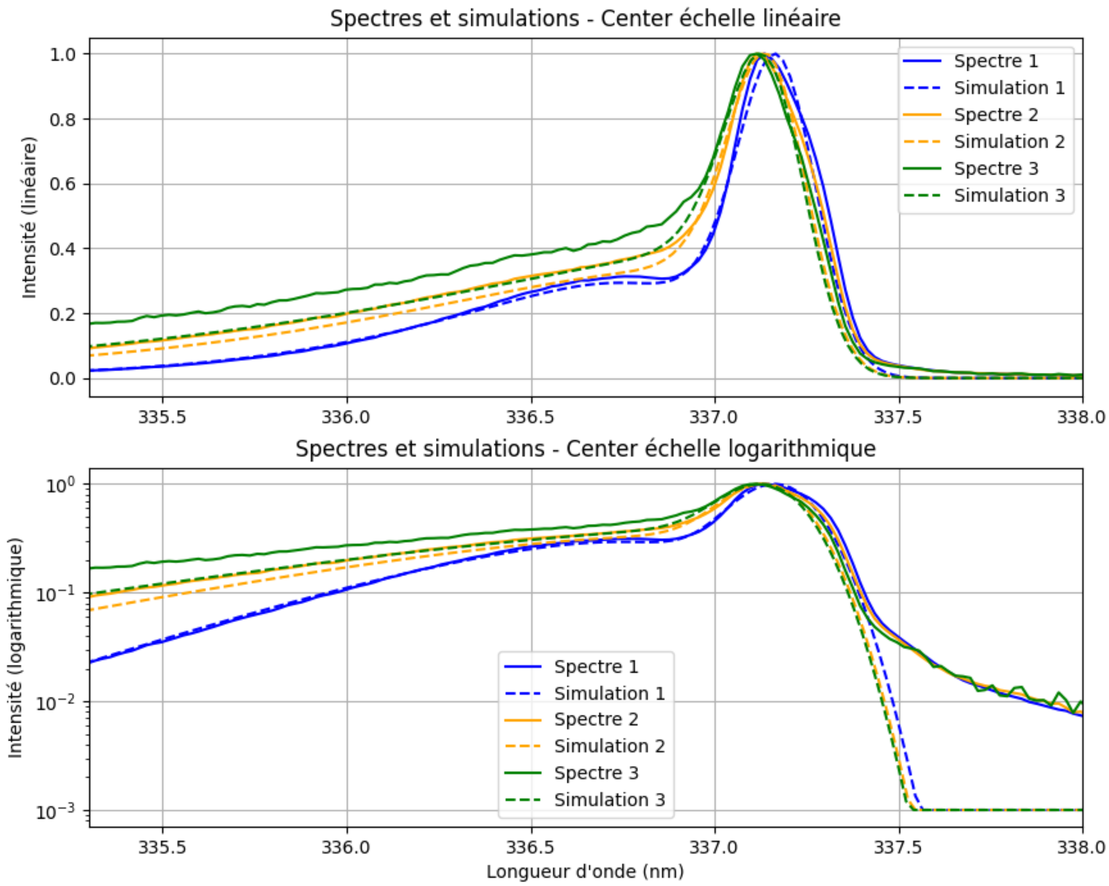
[42]:
```
xs_vis.show_result_calculation_Trot(W, c_spectrum_pulses[2], J_range=(12, 23),
    certainty=0.95, max_J=30) # Third pulse
```



Traçons les spectres simulés à partir de ces températures pour les trois pulses du centre.

[43]:
```python
elargissement = 0.1
llims = (335.3, 338)
# Calcul du spectre de simulation
simulation_spectra_from_reg = [xs_sim.get_spectrum(W, T_el=1_000, T_vib=t_vib_by_ratio["center"][i], T_rot=t_rot_analytic["center"][i][0], sigma_exp=0.1) for i in index_pulses_center]
```

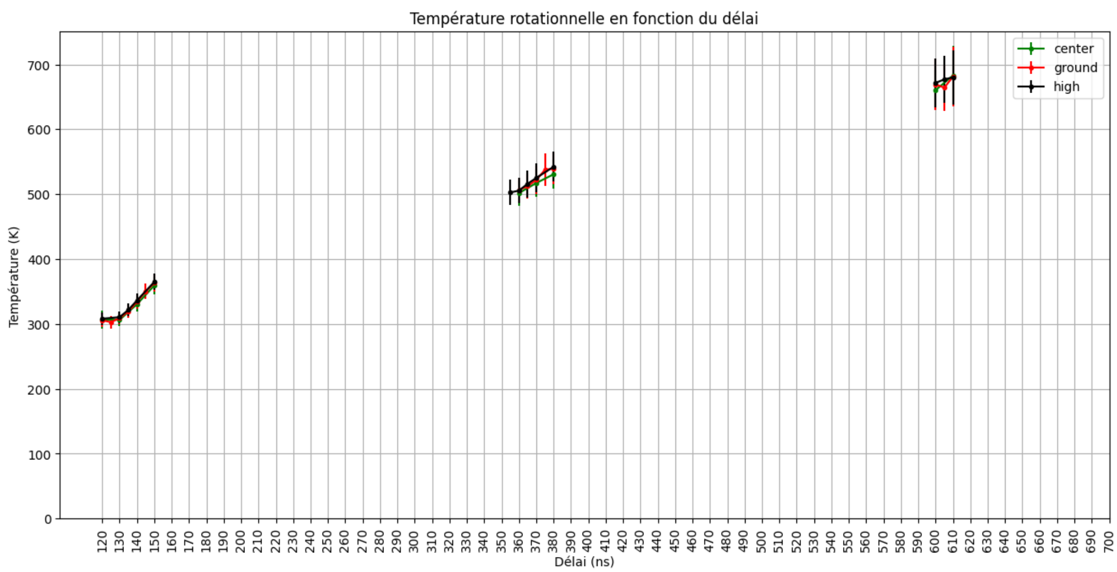
[44]:
```python
fig, axs = plt.subplots(2, 1, figsize=(10, 8))

plot_spectra_and_simulation(simulation_spectra_from_reg, axs[0], axs[1], simulation_labels=["Simulation 1", "Simulation 2", "Simulation 3"], title="Spectres et simulations - Center", decalages=[0.01,0.03,0.035], xlims=llims)
```



Affichons les résultats en fonction du délais.

[45]:
```
plt.figure(figsize=(15, 7))

for name in t_rot_analytic.keys():
    filter_1, filter_2, filter_3 = get_filter(plasmas_datas[name], 0),␣
 ↪get_filter(plasmas_datas[name], 1), get_filter(plasmas_datas[name], 2)
    plt.errorbar(plasmas_datas[name][0][filter_1],␣
 ↪t_rot_analytic[name][filter_1][:,0], yerr=t_rot_analytic[name][filter_1][:
 ↪,1], fmt=".-", color=colors_places[name], label=name)
    plt.errorbar(plasmas_datas[name][0][filter_2],␣
 ↪t_rot_analytic[name][filter_2][:,0], yerr=t_rot_analytic[name][filter_2][:
 ↪,1], fmt=".-", color=colors_places[name])
    plt.errorbar(plasmas_datas[name][0][filter_3],␣
 ↪t_rot_analytic[name][filter_3][:,0], yerr=t_rot_analytic[name][filter_3][:
 ↪,1], fmt=".-", color=colors_places[name])



plt.ylabel("Température (K)")
```

```
plt.xlabel("Délai (ns)")

plt.title("Température rotationnelle en fonction du délai")

plt.grid()
plt.xticks(plasmas_datas["center"][0], rotation=90)
plt.legend()
plt.ylim(bottom=0)
plt.savefig("./res/temperature_rot1_vs_delay.png")
plt.show()
```



1.7.3 Extraction of the rotational temperature - Fit on the v'=0->v''=0 transition

[46]:
```
llims = (335, 337.5)
mask = (W > llims[0]) & (W < llims[1])
```

[47]:
```
def get_temp_rot_2(data, T_vib, nb_points=None, method="sum_squares",␣
↪mask=mask):
    spectra = xs_utils.compute_spectra(data, True)
    if nb_points is not None:
        X = np.linspace(W[mask].min(), W[mask].max(), nb_points)
        Y = np.interp(X, W[mask], spectra[mask])
    else :
        X = W[mask]
        Y = spectra[mask]
    return xs_utils.get_best_fit(X, Y,
                            T_vib=T_vib,
                            elargissement=0.1,
```

```
                                w_decalage=0,
                            T_rot_range=(100, 2000),
                            # elargissement_range=(0.05,0.12),
                            w_decalage_range=(-0.5, 0.5),
                            verbose=True,
                            nb_steps=3,
                            score_method=method)[2]
```

[48]:
```
# Prends quelques minutes à exécuter
# On optimise déjà en ne prenant que les données à l'intérieur des pulses
# on pourrait également prendre moins de points # mais prend du temps même avec␣
↪seulement 100 points
t_rot_by_fit_simple = {
    name : np.array([get_temp_rot_2(d, t_vib_by_ratio[name][i]) if␣
↪is_inside_one_of_the_pulses(delay) else 0 for delay, d in zip(delays,␣
↪data)]) for name, (delays, data) in plasmas_datas.items()
}
```

```
Iteration   1 | Score:    0.063 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    317 K | Scale 1.00000000 | Décalage: 0.0043 nm
Iteration   2 | Score:    0.061 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    302 K | Scale 1.00000000 | Décalage: 0.0054 nm
Iteration   3 | Score:    0.061 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    298 K | Scale 1.00000000 | Décalage: 0.0057 nm
Iteration   1 | Score:    0.071 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0025 nm
Iteration   2 | Score:    0.071 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    323 K | Scale 1.00000000 | Décalage: 0.0031 nm
Iteration   3 | Score:    0.071 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    321 K | Scale 1.00000000 | Décalage: 0.0033 nm
Iteration   1 | Score:    0.073 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    375 K | Scale 1.00000000 | Décalage: 0.0026 nm
Iteration   2 | Score:    0.072 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    365 K | Scale 1.00000000 | Décalage: 0.0033 nm
Iteration   3 | Score:    0.072 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    363 K | Scale 1.00000000 | Décalage: 0.0035 nm
Iteration   1 | Score:    0.067 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    420 K | Scale 1.00000000 | Décalage: 0.0020 nm
Iteration   2 | Score:    0.067 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    413 K | Scale 1.00000000 | Décalage: 0.0024 nm
Iteration   3 | Score:    0.066 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    409 K | Scale 1.00000000 | Décalage: 0.0026 nm
Iteration   1 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    669 K | Scale 1.00000000 | Décalage: 0.0060 nm
Iteration   2 | Score:    0.045 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    652 K | Scale 1.00000000 | Décalage: 0.0068 nm
Iteration   3 | Score:    0.045 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    649 K | Scale 1.00000000 | Décalage: 0.0070 nm
```

```
Iteration   1 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    708 K | Scale 1.00000000 | Décalage: 0.0064 nm
Iteration   2 | Score:    0.047 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    690 K | Scale 1.00000000 | Décalage: 0.0073 nm
Iteration   3 | Score:    0.047 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    687 K | Scale 1.00000000 | Décalage: 0.0075 nm
Iteration   1 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    733 K | Scale 1.00000000 | Décalage: 0.0072 nm
Iteration   2 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    712 K | Scale 1.00000000 | Décalage: 0.0083 nm
Iteration   3 | Score:    0.045 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:    709 K | Scale 1.00000000 | Décalage: 0.0085 nm
Iteration   1 | Score:    0.039 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1104 K | Scale 1.00000000 | Décalage: 0.0175 nm
Iteration   2 | Score:    0.033 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1060 K | Scale 1.00000000 | Décalage: 0.0189 nm
Iteration   3 | Score:    0.033 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1056 K | Scale 1.00000000 | Décalage: 0.0190 nm
Iteration   1 | Score:    0.033 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1187 K | Scale 1.00000000 | Décalage: 0.0150 nm
Iteration   2 | Score:    0.030 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1150 K | Scale 1.00000000 | Décalage: 0.0160 nm
Iteration   3 | Score:    0.030 | Elargissement:   0.10 nm | T_vib =   4043 |
T_rot:   1147 K | Scale 1.00000000 | Décalage: 0.0161 nm
Iteration   1 | Score:    0.121 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    418 K | Scale 1.00000000 | Décalage: 0.0259 nm
Iteration   2 | Score:    0.066 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    325 K | Scale 1.00000000 | Décalage: 0.0313 nm
Iteration   3 | Score:    0.063 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    305 K | Scale 1.00000000 | Décalage: 0.0325 nm
Iteration   1 | Score:    0.125 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    421 K | Scale 1.00000000 | Décalage: 0.0235 nm
Iteration   2 | Score:    0.079 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    337 K | Scale 1.00000000 | Décalage: 0.0281 nm
Iteration   3 | Score:    0.076 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    317 K | Scale 1.00000000 | Décalage: 0.0293 nm
Iteration   1 | Score:    0.117 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    425 K | Scale 1.00000000 | Décalage: 0.0223 nm
Iteration   2 | Score:    0.076 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    347 K | Scale 1.00000000 | Décalage: 0.0265 nm
Iteration   3 | Score:    0.074 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    329 K | Scale 1.00000000 | Décalage: 0.0276 nm
Iteration   1 | Score:    0.116 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    444 K | Scale 1.00000000 | Décalage: 0.0219 nm
Iteration   2 | Score:    0.078 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    369 K | Scale 1.00000000 | Décalage: 0.0259 nm
Iteration   3 | Score:    0.076 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    351 K | Scale 1.00000000 | Décalage: 0.0270 nm
```

```
Iteration   1 | Score:    0.115 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    470 K | Scale 1.00000000 | Décalage: 0.0220 nm
Iteration   2 | Score:    0.080 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    395 K | Scale 1.00000000 | Décalage: 0.0260 nm
Iteration   3 | Score:    0.078 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    378 K | Scale 1.00000000 | Décalage: 0.0270 nm
Iteration   1 | Score:    0.113 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    495 K | Scale 1.00000000 | Décalage: 0.0235 nm
Iteration   2 | Score:    0.075 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    414 K | Scale 1.00000000 | Décalage: 0.0279 nm
Iteration   3 | Score:    0.074 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    401 K | Scale 1.00000000 | Décalage: 0.0287 nm
Iteration   1 | Score:    0.106 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    509 K | Scale 1.00000000 | Décalage: 0.0237 nm
Iteration   2 | Score:    0.069 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    430 K | Scale 1.00000000 | Décalage: 0.0280 nm
Iteration   3 | Score:    0.068 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    416 K | Scale 1.00000000 | Décalage: 0.0288 nm
Iteration   1 | Score:    0.072 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    727 K | Scale 1.00000000 | Décalage: 0.0247 nm
Iteration   2 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    657 K | Scale 1.00000000 | Décalage: 0.0281 nm
Iteration   3 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    647 K | Scale 1.00000000 | Décalage: 0.0286 nm
Iteration   1 | Score:    0.069 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    744 K | Scale 1.00000000 | Décalage: 0.0242 nm
Iteration   2 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    676 K | Scale 1.00000000 | Décalage: 0.0275 nm
Iteration   3 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    666 K | Scale 1.00000000 | Décalage: 0.0280 nm
Iteration   1 | Score:    0.071 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    769 K | Scale 1.00000000 | Décalage: 0.0244 nm
Iteration   2 | Score:    0.049 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    701 K | Scale 1.00000000 | Décalage: 0.0276 nm
Iteration   3 | Score:    0.049 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    692 K | Scale 1.00000000 | Décalage: 0.0281 nm
Iteration   1 | Score:    0.065 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    786 K | Scale 1.00000000 | Décalage: 0.0250 nm
Iteration   2 | Score:    0.042 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    716 K | Scale 1.00000000 | Décalage: 0.0283 nm
Iteration   3 | Score:    0.042 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    706 K | Scale 1.00000000 | Décalage: 0.0288 nm
Iteration   1 | Score:    0.069 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    787 K | Scale 1.00000000 | Décalage: 0.0262 nm
Iteration   2 | Score:    0.044 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    714 K | Scale 1.00000000 | Décalage: 0.0297 nm
Iteration   3 | Score:    0.043 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:    702 K | Scale 1.00000000 | Décalage: 0.0303 nm
```

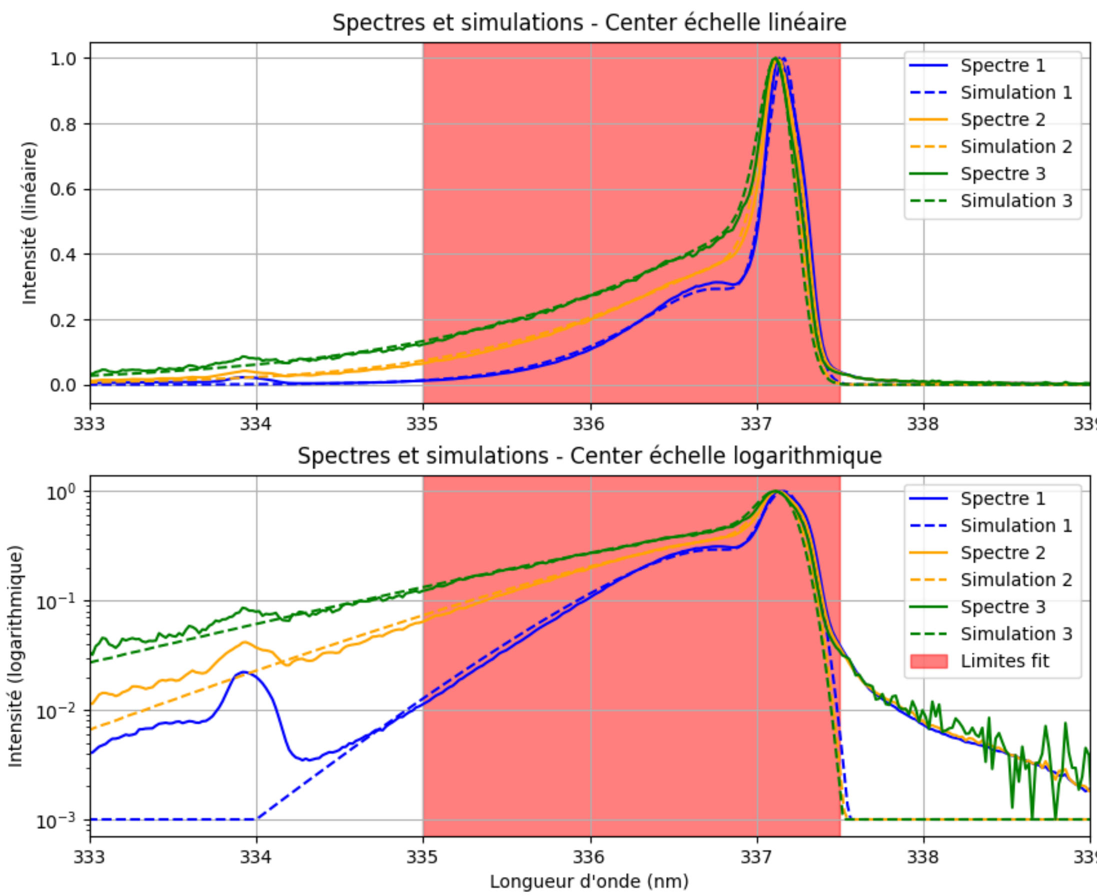
```
Iteration   1 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1147 K | Scale 1.00000000 | Décalage: 0.0311 nm
Iteration   2 | Score:    0.030 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1066 K | Scale 1.00000000 | Décalage: 0.0330 nm
Iteration   3 | Score:    0.030 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1062 K | Scale 1.00000000 | Décalage: 0.0331 nm
Iteration   1 | Score:    0.055 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1137 K | Scale 1.00000000 | Décalage: 0.0317 nm
Iteration   2 | Score:    0.037 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1056 K | Scale 1.00000000 | Décalage: 0.0336 nm
Iteration   3 | Score:    0.036 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1051 K | Scale 1.00000000 | Décalage: 0.0337 nm
Iteration   1 | Score:    0.060 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1184 K | Scale 1.00000000 | Décalage: 0.0319 nm
Iteration   2 | Score:    0.042 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1103 K | Scale 1.00000000 | Décalage: 0.0339 nm
Iteration   3 | Score:    0.042 | Elargissement:   0.10 nm | T_vib =   3421 |
T_rot:   1098 K | Scale 1.00000000 | Décalage: 0.0340 nm
Iteration   1 | Score:    0.122 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    424 K | Scale 1.00000000 | Décalage: 0.0257 nm
Iteration   2 | Score:    0.068 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    333 K | Scale 1.00000000 | Décalage: 0.0310 nm
Iteration   3 | Score:    0.066 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    314 K | Scale 1.00000000 | Décalage: 0.0321 nm
Iteration   1 | Score:    0.124 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    437 K | Scale 1.00000000 | Décalage: 0.0230 nm
Iteration   2 | Score:    0.081 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    357 K | Scale 1.00000000 | Décalage: 0.0274 nm
Iteration   3 | Score:    0.079 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    337 K | Scale 1.00000000 | Décalage: 0.0285 nm
Iteration   1 | Score:    0.119 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    452 K | Scale 1.00000000 | Décalage: 0.0228 nm
Iteration   2 | Score:    0.080 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    374 K | Scale 1.00000000 | Décalage: 0.0270 nm
Iteration   3 | Score:    0.078 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    356 K | Scale 1.00000000 | Décalage: 0.0281 nm
Iteration   1 | Score:    0.123 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    482 K | Scale 1.00000000 | Décalage: 0.0231 nm
Iteration   2 | Score:    0.084 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    400 K | Scale 1.00000000 | Décalage: 0.0276 nm
Iteration   3 | Score:    0.083 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    386 K | Scale 1.00000000 | Décalage: 0.0284 nm
Iteration   1 | Score:    0.107 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    517 K | Scale 1.00000000 | Décalage: 0.0244 nm
Iteration   2 | Score:    0.068 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    436 K | Scale 1.00000000 | Décalage: 0.0288 nm
Iteration   3 | Score:    0.067 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    422 K | Scale 1.00000000 | Décalage: 0.0297 nm
```

```
Iteration   1 | Score:    0.076 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    712 K | Scale 1.00000000 | Décalage: 0.0266 nm
Iteration   2 | Score:    0.047 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    636 K | Scale 1.00000000 | Décalage: 0.0305 nm
Iteration   3 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    624 K | Scale 1.00000000 | Décalage: 0.0312 nm
Iteration   1 | Score:    0.076 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    738 K | Scale 1.00000000 | Décalage: 0.0258 nm
Iteration   2 | Score:    0.049 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    666 K | Scale 1.00000000 | Décalage: 0.0294 nm
Iteration   3 | Score:    0.049 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    654 K | Scale 1.00000000 | Décalage: 0.0300 nm
Iteration   1 | Score:    0.072 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    758 K | Scale 1.00000000 | Décalage: 0.0257 nm
Iteration   2 | Score:    0.047 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    686 K | Scale 1.00000000 | Décalage: 0.0292 nm
Iteration   3 | Score:    0.046 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    675 K | Scale 1.00000000 | Décalage: 0.0298 nm
Iteration   1 | Score:    0.074 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    787 K | Scale 1.00000000 | Décalage: 0.0257 nm
Iteration   2 | Score:    0.050 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    715 K | Scale 1.00000000 | Décalage: 0.0293 nm
Iteration   3 | Score:    0.049 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    704 K | Scale 1.00000000 | Décalage: 0.0298 nm
Iteration   1 | Score:    0.068 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    816 K | Scale 1.00000000 | Décalage: 0.0270 nm
Iteration   2 | Score:    0.042 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    741 K | Scale 1.00000000 | Décalage: 0.0308 nm
Iteration   3 | Score:    0.041 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:    728 K | Scale 1.00000000 | Décalage: 0.0314 nm
Iteration   1 | Score:    0.053 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1168 K | Scale 1.00000000 | Décalage: 0.0318 nm
Iteration   2 | Score:    0.035 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1086 K | Scale 1.00000000 | Décalage: 0.0338 nm
Iteration   3 | Score:    0.035 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1081 K | Scale 1.00000000 | Décalage: 0.0339 nm
Iteration   1 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1192 K | Scale 1.00000000 | Décalage: 0.0307 nm
Iteration   2 | Score:    0.032 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1112 K | Scale 1.00000000 | Décalage: 0.0325 nm
Iteration   3 | Score:    0.032 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1108 K | Scale 1.00000000 | Décalage: 0.0326 nm
Iteration   1 | Score:    0.048 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1204 K | Scale 1.00000000 | Décalage: 0.0308 nm
Iteration   2 | Score:    0.032 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1125 K | Scale 1.00000000 | Décalage: 0.0325 nm
Iteration   3 | Score:    0.032 | Elargissement:   0.10 nm | T_vib =   3558 |
T_rot:   1121 K | Scale 1.00000000 | Décalage: 0.0326 nm
```

Affichons les fits pour les trois pulses au centre

[49]:
```
elargissement = 0.09
# Calcul du spectre de simulation
simulation_spectra_fit_sample = [xs_sim.get_spectrum(W, T_el=1_000,␣
↪T_vib=t_vib_by_ratio["center"][i], T_rot=t_rot_by_fit_simple["center"][i],␣
↪sigma_exp=0.1) for i in index_pulses_center]
```

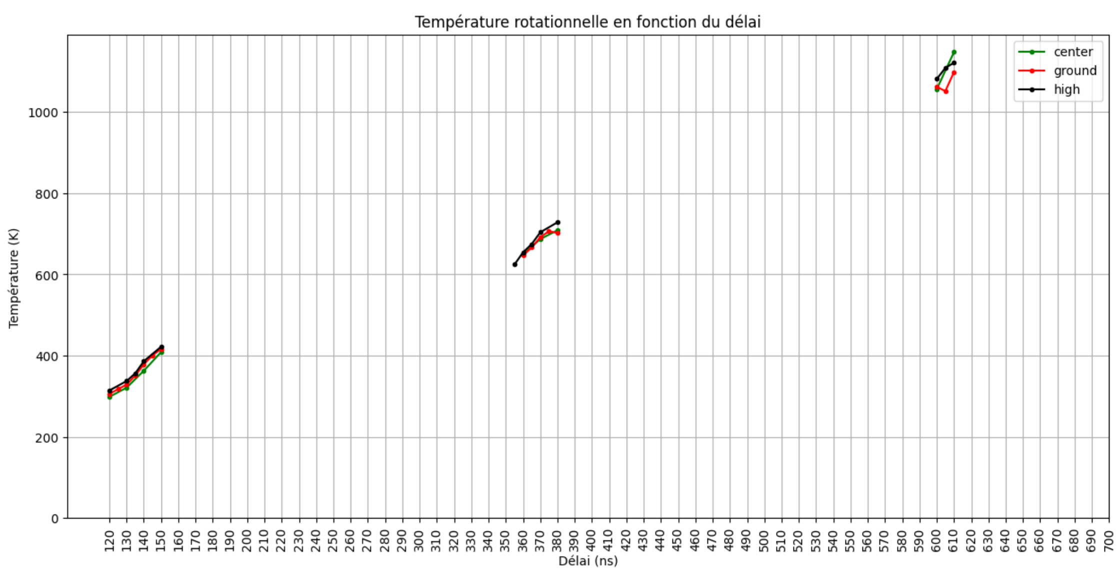
[50]:
```
fig, axs = plt.subplots(2, 1, figsize=(10, 8))

plot_spectra_and_simulation(simulation_spectra_fit_sample, axs[0], axs[1],␣
↪fit_areas=[llims], simulation_labels=["Simulation 1", "Simulation 2",␣
↪"Simulation 3"], title="Spectres et simulations - Center", decalages=[0.01,0.
↪03,0.04])
```



Testons différents paramètres pour le fit pour les trois pulses.

[51]:
```
plt.figure(figsize=(15, 7))

for name in t_rot_by_fit_simple.keys():
```

```
    filter_1, filter_2, filter_3 = get_filter(plasmas_datas[name], 0),␣
↪get_filter(plasmas_datas[name], 1), get_filter(plasmas_datas[name], 2)
    plt.plot(plasmas_datas[name][0][filter_1],␣
↪t_rot_by_fit_simple[name][filter_1], ".-", color=colors_places[name],␣
↪label=name)
    plt.plot(plasmas_datas[name][0][filter_2],␣
↪t_rot_by_fit_simple[name][filter_2], ".-", color=colors_places[name])
    plt.plot(plasmas_datas[name][0][filter_3],␣
↪t_rot_by_fit_simple[name][filter_3], ".-", color=colors_places[name])



plt.ylabel("Température (K)")
plt.xlabel("Délai (ns)")

plt.title("Température rotationnelle en fonction du délai")



plt.grid()
plt.xticks(plasmas_datas["center"][0], rotation=90)
plt.legend()
plt.ylim(bottom=0)
plt.savefig("./res/temperature_rot2_vs_delay.png")
plt.show()
```



1.7.4 Fit complet

[80]:
```
areas = [
    (335.1, 337.6),
    (332.5, 334.3),
    (329.7, 331.4),
    (327.5, 328.8)
]

areas = areas

filters_trans_vib = [(areas[i][0] <= W) & (W <=areas[i][1]) for i in␣
 ↪range(len(areas))]


filter_whole_fit = np.sum(filters_trans_vib, axis=0).astype(bool)
```

[81]:
```
def process_whole_fit(spectrum):
    return xs_utils.get_best_fit(W[filter_whole_fit],␣
 ↪spectrum[filter_whole_fit],
                                T_rot=400,
                                T_vib=1500,
                                elargissement=0.1,
                                w_decalage=0, # decalage_assumtion
                                w_scale=1,
                                T_rot_range=(300, 2_000),
                                T_vib_range=(300, 10_000), # Il faut une plage␣
 ↪assez grande pour que la fonction soit bien convexe
                                elargissement_range=(0.05,0.15),
                                # w_decalage_range=(-2,2),
                                # w_scale_range=(0.99,1.01), # ne pas trop␣
 ↪grand sinon la fonction n'est pas convexe
                                modelisation_spectrum_function=xs_sim.
 ↪get_whole_spectrum, # cette fois-ci on génère les autres transitions␣
 ↪vibrationnelles
                                verbose=True,
                                nb_steps=3)
```

[82]:
```
params_fit_center_pulses = np.array([process_whole_fit(c_spectrum_pulses[i])␣
 ↪for i in range(len(c_spectrum_pulses))])
```

```
Iteration   1 | Score:    0.089 | Elargissement:   0.10 nm | T_vib =   2761 |
T_rot:    333 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.089 | Elargissement:   0.10 nm | T_vib =   2841 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.089 | Elargissement:   0.10 nm | T_vib =   2841 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.094 | Elargissement:   0.10 nm | T_vib =   4426 |
```

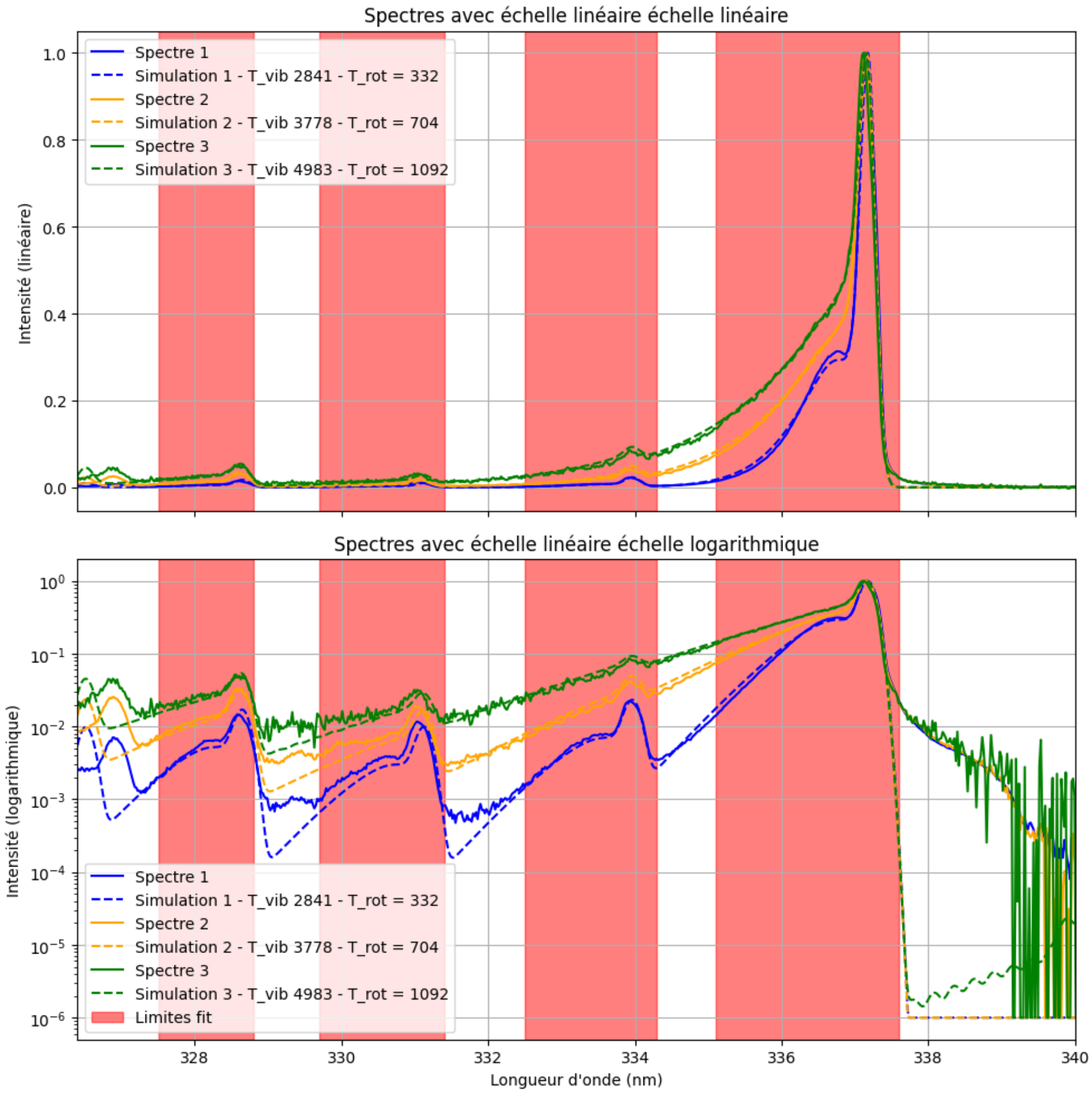
```
T_rot:    702 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.093 | Elargissement:   0.10 nm | T_vib =   3784 |
T_rot:    704 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.093 | Elargissement:   0.10 nm | T_vib =   3778 |
T_rot:    704 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.205 | Elargissement:   0.10 nm | T_vib =   7199 |
T_rot:   1081 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.202 | Elargissement:   0.10 nm | T_vib =   5027 |
T_rot:   1092 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.202 | Elargissement:   0.10 nm | T_vib =   4983 |
T_rot:   1092 K | Scale 1.00000000 | Décalage: 0.0000 nm
```

[83]:

```
spectra_simulation_whole = [xs_sim.
↪get_whole_spectrum(params_fit_center_pulses[i][5]*W+params_fit_center_pulses[i][4],␣
↪T_el=1_000, T_vib=params_fit_center_pulses[i][1],␣
↪T_rot=params_fit_center_pulses[i][2],␣
↪sigma_exp=params_fit_center_pulses[i][3]) for i in␣
↪range(len(params_fit_center_pulses))]
```

[84]:

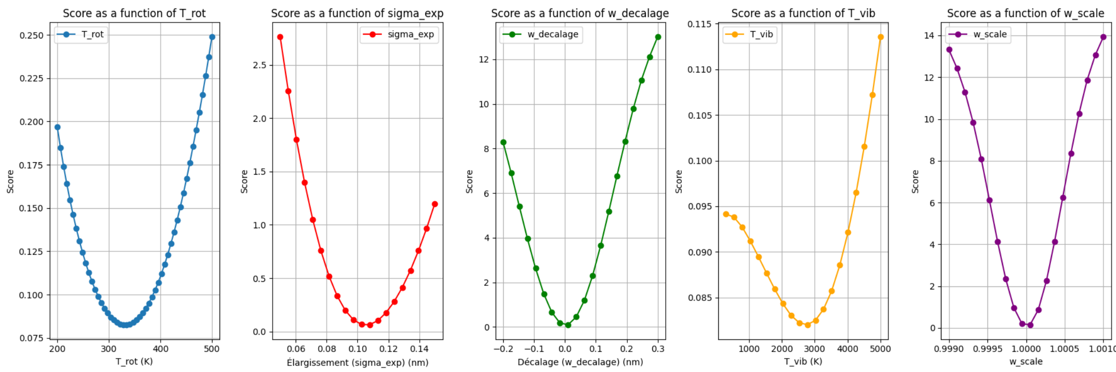
```
fig, axs = plt.subplots(2, 1, figsize=(10, 10), sharex=True)



# Call the function
plot_spectra_and_simulation(spectra_simulation_whole, axs[0], axs[1],␣
↪fit_areas=areas, simulation_labels = [f'Simulation {i+1} - T_vib {tv:4.0f} -␣
↪T_rot = {tr:3.0f}'  for i, (tv, tr) in␣
↪enumerate(zip(params_fit_center_pulses[:,1], params_fit_center_pulses[:
↪,2]))], title="Spectres avec échelle linéaire", xlims=(np.min(W), 340),␣
↪epsilon=1e-6)



plt.tight_layout()
plt.show()
```



[57]:
```
# Calculer les scores pour chaque valeur de T_rot
T_rot_range = np.linspace(200, 500, 50)
sigma_exp_range = np.linspace(0.05, 0.15, 20)
w_decalage_range = np.linspace(-0.2, 0.3, 20)
T_vib_range = np.linspace(300, 5_000, 20)
w_scale_range = np.linspace(0.999, 1.001, 20)

scores_T_rot = np.zeros_like(T_rot_range)
scores_sigma_exp = np.zeros_like(sigma_exp_range)
scores_w_decalage = np.zeros_like(w_decalage_range)
scores_T_vib = np.zeros_like(T_vib_range)
scores_w_scale = np.zeros_like(w_scale_range)
```

```
for idx, T_rot in enumerate(T_rot_range):
    scores_T_rot[idx] = xs_utils.compute_score_fit(
        c_spectrum_pulses[0],
        xs_sim.
 ↪get_whole_spectrum(params_fit_center_pulses[0][5]*W+params_fit_center_pulses[0][4],␣
 ↪T_el=1_000, T_vib=params_fit_center_pulses[0][1], T_rot=T_rot,␣
 ↪sigma_exp=params_fit_center_pulses[0][3])
    )

for idx, sigma_exp in enumerate(sigma_exp_range):
    scores_sigma_exp[idx] = xs_utils.compute_score_fit(
        c_spectrum_pulses[0],
        xs_sim.
 ↪get_whole_spectrum(params_fit_center_pulses[0][5]*W+params_fit_center_pulses[0][4],␣
 ↪T_el=1_000, T_vib=params_fit_center_pulses[0][1],␣
 ↪T_rot=params_fit_center_pulses[0][2], sigma_exp=sigma_exp)
    )

for idx, w_decalage in enumerate(w_decalage_range):
    scores_w_decalage[idx] = xs_utils.compute_score_fit(
        c_spectrum_pulses[0],
        xs_sim.get_whole_spectrum(params_fit_center_pulses[0][5]*W+w_decalage,␣
 ↪T_el=1_000, T_vib=params_fit_center_pulses[0][1],␣
 ↪T_rot=params_fit_center_pulses[0][2],␣
 ↪sigma_exp=params_fit_center_pulses[0][3])
    )

for idx, T_vib in enumerate(T_vib_range):
    scores_T_vib[idx] = xs_utils.compute_score_fit(
        c_spectrum_pulses[0],
        xs_sim.
 ↪get_whole_spectrum(params_fit_center_pulses[0][5]*W+params_fit_center_pulses[0][4],␣
 ↪T_el=1_000, T_vib=T_vib, T_rot=params_fit_center_pulses[0][2],␣
 ↪sigma_exp=params_fit_center_pulses[0][3])
    )

for idx, w_scale in enumerate(w_scale_range):
    scores_w_scale[idx] = xs_utils.compute_score_fit(
        c_spectrum_pulses[0],
        xs_sim.get_whole_spectrum(w_scale*W+params_fit_center_pulses[0][4],␣
 ↪T_el=1_000, T_vib=params_fit_center_pulses[0][1],␣
 ↪T_rot=params_fit_center_pulses[0][2],␣
 ↪sigma_exp=params_fit_center_pulses[0][3])
    )

# Tracer le graphe
fig, axs = plt.subplots(1, 5, figsize=(18, 6))
```

```python
axs[0].plot(T_rot_range, scores_T_rot, marker='o', label="T_rot")
axs[0].set_xlabel("T_rot (K)")
axs[0].set_ylabel("Score")
axs[0].set_title("Score as a function of T_rot")
axs[0].grid()
axs[0].legend()

axs[1].plot(sigma_exp_range, scores_sigma_exp, marker='o', label="sigma_exp", color='red')
axs[1].set_xlabel("Élargissement (sigma_exp) (nm)")
axs[1].set_ylabel("Score")
axs[1].set_title("Score as a function of sigma_exp")
axs[1].grid()
axs[1].legend()

axs[2].plot(w_decalage_range, scores_w_decalage, marker='o', label="w_decalage", color='green')
axs[2].set_xlabel("Décalage (w_decalage) (nm)")
axs[2].set_ylabel("Score")
axs[2].set_title("Score as a function of w_decalage")
axs[2].grid()
axs[2].legend()

axs[3].plot(T_vib_range, scores_T_vib, marker='o', label="T_vib", color='orange')
axs[3].set_xlabel("T_vib (K)")
axs[3].set_ylabel("Score")
axs[3].set_title("Score as a function of T_vib")
axs[3].grid()
axs[3].legend()

axs[4].plot(w_scale_range, scores_w_scale, marker='o', label="w_scale", color='purple')
axs[4].set_xlabel("w_scale")
axs[4].set_ylabel("Score")
axs[4].set_title("Score as a function of w_scale")
axs[4].grid()
axs[4].legend()


plt.tight_layout()
plt.savefig("./res/scores_function.png", dpi=300, bbox_inches='tight')
plt.show()
```



[58]:
```
# Récupérons les données pour plus de transitions

params_whole_sim = {
    # name : np.array([process_whole_fit(xs_utils.compute_spectra(d)) if␣
 ↪is_inside_one_of_the_pulses(delay) else [0]*6 for delay, d in zip(delays,␣
 ↪data)]) for name, (delays, data) in plasmas_datas.items()
    name : np.array([process_whole_fit(xs_utils.compute_spectra(d)) if␣
 ↪is_inside_one_of_the_pulses(delay) else [0]*6 for delay, d in zip(delays,␣
 ↪data)]) if name=="center" else np.zeros((len(delays), 6)) for name, (delays,␣
 ↪data) in plasmas_datas.items()
}
```

```
Iteration    1 | Score:    0.080 | Elargissement:   0.10 nm | T_vib =   2978 |
T_rot:    318 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    2 | Score:    0.080 | Elargissement:   0.10 nm | T_vib =   3350 |
T_rot:    317 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    3 | Score:    0.080 | Elargissement:   0.10 nm | T_vib =   3350 |
T_rot:    317 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    1 | Score:    0.084 | Elargissement:   0.10 nm | T_vib =   2747 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    2 | Score:    0.084 | Elargissement:   0.10 nm | T_vib =   3049 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    3 | Score:    0.084 | Elargissement:   0.10 nm | T_vib =   3049 |
T_rot:    332 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    1 | Score:    0.087 | Elargissement:   0.10 nm | T_vib =   3176 |
T_rot:    376 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    2 | Score:    0.087 | Elargissement:   0.10 nm | T_vib =   3318 |
T_rot:    376 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    3 | Score:    0.087 | Elargissement:   0.10 nm | T_vib =   3318 |
T_rot:    376 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    1 | Score:    0.081 | Elargissement:   0.10 nm | T_vib =   3456 |
T_rot:    421 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration    2 | Score:    0.081 | Elargissement:   0.10 nm | T_vib =   3308 |
T_rot:    421 K | Scale 1.00000000 | Décalage: 0.0000 nm
```

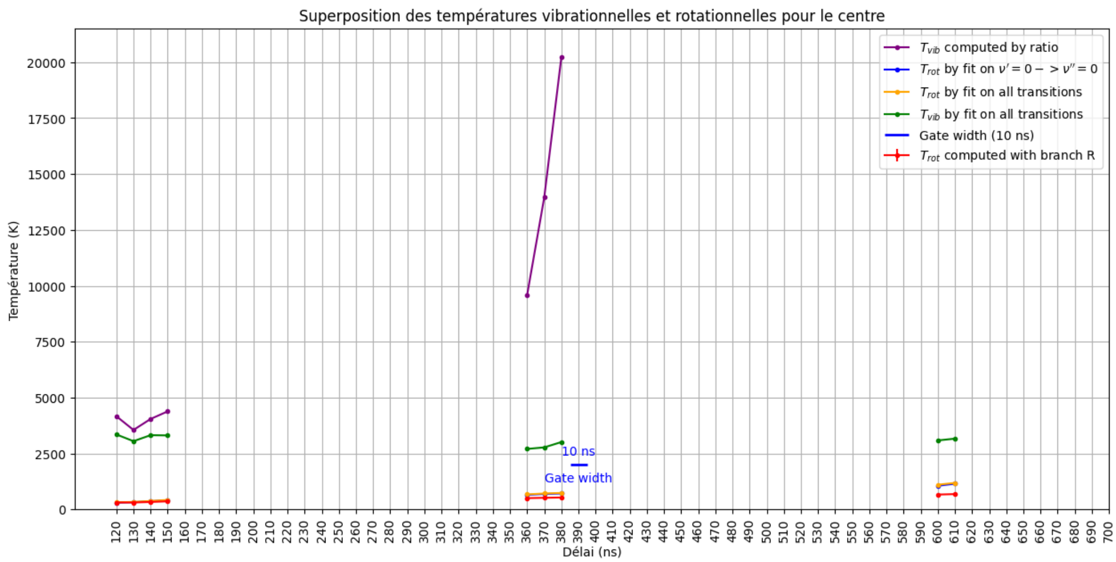
```
Iteration   3 | Score:    0.081 | Elargissement:   0.10 nm | T_vib =   3307 |
T_rot:    421 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.066 | Elargissement:   0.10 nm | T_vib =   6715 |
T_rot:    663 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.065 | Elargissement:   0.10 nm | T_vib =   2784 |
T_rot:    669 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.065 | Elargissement:   0.10 nm | T_vib =   2703 |
T_rot:    669 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.070 | Elargissement:   0.10 nm | T_vib =   8406 |
T_rot:    700 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.069 | Elargissement:   0.10 nm | T_vib =   2884 |
T_rot:    708 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.069 | Elargissement:   0.10 nm | T_vib =   2771 |
T_rot:    708 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.074 | Elargissement:   0.10 nm | T_vib =  10000 |
T_rot:    725 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.073 | Elargissement:   0.10 nm | T_vib =   3142 |
T_rot:    733 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.073 | Elargissement:   0.10 nm | T_vib =   3010 |
T_rot:    733 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.131 | Elargissement:   0.10 nm | T_vib =  10000 |
T_rot:   1086 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.129 | Elargissement:   0.10 nm | T_vib =   3400 |
T_rot:   1100 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.129 | Elargissement:   0.10 nm | T_vib =   3083 |
T_rot:   1101 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   1 | Score:    0.106 | Elargissement:   0.10 nm | T_vib =  10000 |
T_rot:   1166 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   2 | Score:    0.103 | Elargissement:   0.10 nm | T_vib =   3514 |
T_rot:   1181 K | Scale 1.00000000 | Décalage: 0.0000 nm
Iteration   3 | Score:    0.103 | Elargissement:   0.10 nm | T_vib =   3162 |
T_rot:   1183 K | Scale 1.00000000 | Décalage: 0.0000 nm
```

1.7.5 Résumé

[63]:

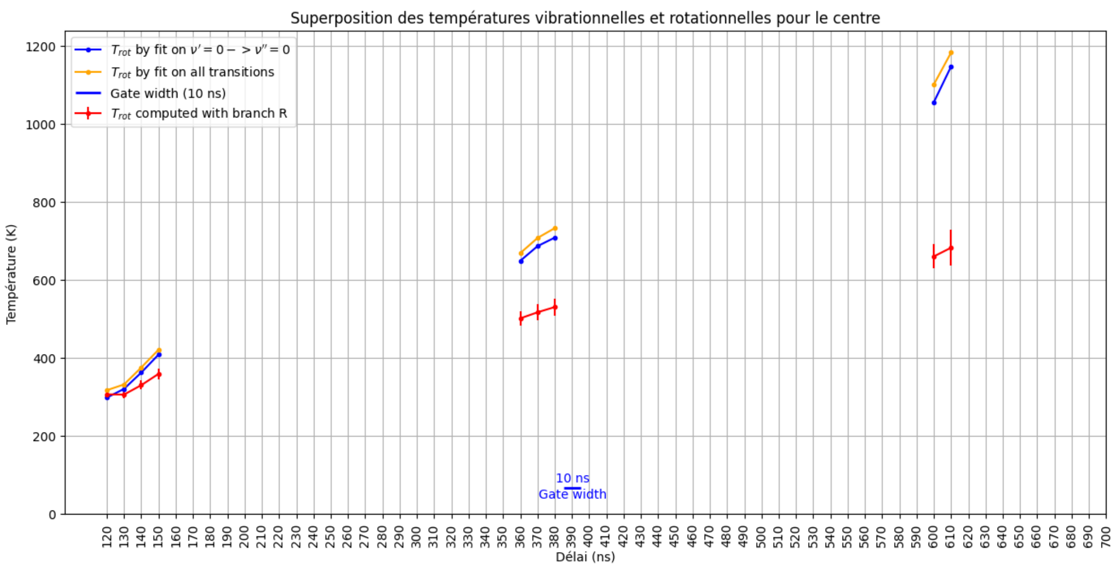
```python
plt.figure(figsize=(15, 7))

# Plot vibrational temperature
name = "center"
for i in range(3):
    filter = get_filter(plasmas_datas[name], i)
    label = r"$T_{vib}$ computed by ratio" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], t_vib_by_ratio[name][filter], ".
↪-", color="purple", label=label)

    label = r"$T_{rot}$ computed with branch R" if i==0 else None
```

```python
    plt.errorbar(plasmas_datas[name][0][filter], t_rot_analytic[name][filter][:,0], yerr=t_rot_analytic[name][filter][:,1], fmt=".-", color="red", label=label)

    label = r"$T_{rot}$ by fit on $\nu'=0->\nu''=0$" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], t_rot_by_fit_simple[name][filter], ".-", color="blue", label=label)

    label = r"$T_{rot}$ by fit on all transitions" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], params_whole_sim[name][filter][:,2], ".-", color="orange", label=label)

    label = r"$T_{vib}$ by fit on all transitions" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], params_whole_sim[name][filter][:,1], ".-", color="green", label=label)



# Add a 10 nanoseconds scale bar
M = np.max(t_vib_by_ratio[name])
plt.hlines(y=M * 0.1, xmin=premier_pulse + D_theorique, xmax=premier_pulse + D_theorique + 10, color="blue", linewidth=2, label="Gate width (10 ns)")
plt.text(premier_pulse + D_theorique + 5, M * 0.12, "10 ns", color="blue", ha="center")
plt.text(premier_pulse + D_theorique + 5, M * 0.06, "Gate width", color="blue", ha="center")


plt.ylabel("Température (K)")
plt.xlabel("Délai (ns)")

plt.title("Superposition des températures vibrationnelles et rotationnelles pour le centre")
plt.grid()
plt.xticks(plasmas_datas[name][0], rotation=90)
plt.legend()
plt.ylim(bottom=0)
plt.savefig("./res/superposed_temperatures.png")
plt.show()
```



[64]:
```python
plt.figure(figsize=(15, 7))

# Plot vibrational temperature
name = "center"
for i in range(3):
    filter = get_filter(plasmas_datas[name], i)

    label = r"$T_{rot}$ computed with branch R" if i==0 else None
    plt.errorbar(plasmas_datas[name][0][filter], t_rot_analytic[name][filter][:,0], yerr=t_rot_analytic[name][filter][:,1], fmt=".-", color="red", label=label)

    label = r"$T_{rot}$ by fit on $\nu'=0->\nu''=0$" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], t_rot_by_fit_simple[name][filter], ".-", color="blue", label=label)

    label = r"$T_{rot}$ by fit on all transitions" if i==0 else None
    plt.plot(plasmas_datas[name][0][filter], params_whole_sim[name][filter][:,2], ".-", color="orange", label=label)




# Add a 10 nanoseconds scale bar
M = np.max(t_rot_analytic[name])
plt.hlines(y=M * 0.1, xmin=premier_pulse + D_theorique, xmax=premier_pulse + D_theorique + 10, color="blue", linewidth=2, label="Gate width (10 ns)")
```

```
plt.text(premier_pulse + D_theorique + 5, M * 0.12, "10 ns", color="blue",␣
↪ha="center")
plt.text(premier_pulse + D_theorique + 5, M * 0.06, "Gate width", color="blue",␣
↪ha="center")


plt.ylabel("Température (K)")
plt.xlabel("Délai (ns)")

plt.title("Superposition des températures vibrationnelles et rotationnelles␣
↪pour le centre")
plt.grid()
plt.xticks(plasmas_datas[name][0], rotation=90)
plt.legend()
plt.ylim(bottom=0)
plt.savefig("./res/superposed_temperatures.png")
plt.show()
```



[ ]:
```
df = pd.DataFrame({
    "delay": plasmas_datas["center"][0][index_pulses_center],
    "T_vib par ratio (K)": t_vib_by_ratio["center"][index_pulses_center],
    "T_rot par régression (K)": t_rot_analytic["center"][index_pulses_center,0],
    "Incertitude T_rot par régression (K)":␣
↪t_rot_analytic["center"][index_pulses_center,1],
    "T_rot par fit simple (K)":␣
↪t_rot_by_fit_simple["center"][index_pulses_center],
```

```python
    "T_rot par fit complet (K)":␣
↪params_whole_sim["center"][index_pulses_center,2],
    "T_vib par fit complet (K)":␣
↪params_whole_sim["center"][index_pulses_center,1]
})
df.to_csv("./res/temperatures.csv", index=False)
# Afficher le DataFrame avec des valeurs en notation scientifique
pd.options.display.float_format = '{:6.0f}'.format
df
```

[ ]:

```
   delay  T_vib par ratio (K)  T_rot par régression (K)  \
0    130                 3551                       306
1    370                13950                       517
2    600                -5443                       661

   Incertitude T_rot par régression (K)  T_rot par fit simple (K)  \
0                                      9                       321
1                                     21                       687
2                                     31                      1056

   T_rot par fit complet (K)  T_vib par fit complet (K)
0                        332                       3049
1                        708                       2771
2                       1101                       3083
```

[73]:

```python
df_print = df.copy()
df_print["T_rot par régression (K)"] = df_print["T_rot par régression (K)"].
↪map("{:6.0f}".format) + " ± " + df_print["Incertitude T_rot par régression␣
↪(K)"].map("{:6.2f}".format)
df_print = df_print.drop(columns=["Incertitude T_rot par régression (K)"])
df_print
```

[73]:

```
   delay  T_vib par ratio (K) T_rot par régression (K)  \
0    130                 3551              306 ±   9.39
1    370                13950              517 ±  20.90
2    600                -5443              661 ±  30.84

   T_rot par fit simple (K)  T_rot par fit complet (K)  \
0                       321                        332
1                       687                        708
2                      1056                       1101

   T_vib par fit complet (K)
0                       3049
1                       2771
2                       3083
```