UCD School of Physics

PHYC30170 Physics Astronomy and Space Lab I

Modulus of Rigidity

Joana C.C. Adao

Student No.: 23311051

5 November 2025

Abstract

This experiment sought to investigate the torsional oscillatory response for three metal specimen rods: brass, stainless steel, and aluminium. The experiment was conducted over a temperature range from 60°C to 180°C, investigating effects on the modulus of rigidity and internal friction. Amplitude-frequency data collected with a laser sensor and National Instruments Programme were fitted with a Lorentzian curve to identify resonant frequency peaks and bandwidths. From these, the shear modulus and internal friction for all three metals was calculated as functions of temperature. All three specimen showcased a gradual decrease in resonant frequency and modulus of rigidity with increasing temperatures, consistent with theoretical predictions. Stainless steel displayed the lowest internal friction and sharpest resonant peaks, while aluminium displayed broader peaks and greater damping. Although experimental values were significantly lower than expected values from published literature, the physical trends aligned well with theoretical expectations, highlighting the temperature dependence of the shear modulus and energy dissipation in metals.

Contents

1. Theory

1.1. Torsional Oscillator

The torsional oscillator, or torsional pendulum, consists of a rigid mass hanging by a thin wire or rod that resists twisting with a resisting torque, which acts to restore the wire to its original state. The rigid body undergoes angular oscillations due to this torque, and for relatively small angles of twisting the magnitude of the restoring torque (τ) will be directly proportional to the angle [1]:

$$\tau = -C\theta \tag{1}$$

with C as the torsional constant, which is dependent on the shear modulus, or modulus of rigidity, of the wire's material, and can be expressed as [2]:

$$C = \frac{\pi G r^4}{2l} \tag{2}$$

with r as the radius and l length of the rod. For a body of moment of inertia I the rotational equation of motion of the system can be written as [1–3]:

$$I\frac{d^2\theta}{dt^2} = -C\theta \tag{3}$$

For the undamped case, the solution is simply $\theta = \theta_0 \cos(\omega t - \phi)$, with $\omega = \sqrt{C/I}$. For a damped oscillator, a damping factor γ is considered in the propagation of oscillations [2, 4]:

$$I\frac{d^2\theta}{dt^2} + \gamma\frac{\mathbf{d}\theta}{\mathbf{dt}} + C\theta = 0 \tag{4}$$

This gives the result $T_0 \exp(i\omega t)$, with T_0 as the period of oscillation $2\pi\sqrt{I/C}$.

1.2. Modulus of Rigidity

The modulus of rigidity, or shear modulus, is a measure of a material's resistance to shear stress deformation. It is expressed as the ration of shear stress τ to shear strain γ [5, 6]:

$$G = \frac{\tau}{\gamma} \tag{5}$$

with G as the modulus of rigidity. It qualifies how rigid or stiff the material is under shear loading, such that a higher value of G indicates a material harder to deform under shear stress. [6]

Knowing that for low damping θ_0 will be at maximum when $\omega \approx \sqrt{C/I} = \omega_0$ C can be substituted from Equation 2 into Equation 5 to give [2]:

$$G = \frac{2Il\omega_0^2}{\pi r^4} \tag{6}$$

The modulus of rigidity is influence by factors such as temperature, confining pressure, microstructure of the inner particles, material composition, and grain size. Numerical investigations of granular packings show that the modulus of rigidity is sensitive to the coordination number of contacts and load magnitudes, with poorly coordinated packings having a shear modulus that varies proportionally. [6, 7]

1.3. Internal Friction

Internal friction in a material refers to the dissipation of mechanical energy, or the force-resisting motion between elements, as it undergoes "elastic" or oscillatory deformation. [8, 9]

This internal friction Q^{-1} can be taken as [2]:

$$Q^{-1} \equiv \frac{\gamma\omega_0}{C} = \frac{\Delta\omega}{\sqrt{3}\omega_0} \tag{7}$$

with $\Delta\omega$ as the full width of the response curve at half the maximum amplitude $\sqrt{3}\gamma\omega_0^2/C$. This relation shows that internal friction increases proportionally with resonance bandwidth $\Delta\omega$ increases, such that the internal loss is greater and the resonance is poorer. [2]

2. Methodology

The experiment was set up as shown in Figure 1, with a laser pointing to a laser sensor by reflecting off of the mirror attached to the magnet attached to the specimen rod. The magnet was positioned between the Helmholtz coils, ensuring that the specimen rod was through the heater coil and clamped securely to the support stand.

With light taps to generate perturbations, the rod's vibrating capacity is checked to ensure proper readings of the amplitude as it vibrates with the torsional oscillations generated by vary-

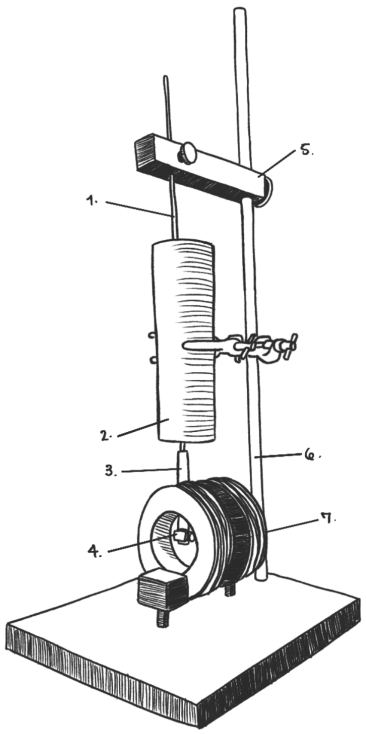


Figure 1: Diagram of the experimental setup: 1, Specimen rod; 2, heater coil; 3, magnet; 4, mirror; 5, clamp support; 6, support stand; 7, Helmholtz coil.

ing frequencies with an oscilloscope, fed through the Helmholtz coil.

The laser sensor produces readings fed through a DAC that can be plotted and graphed to determine the modulus of rigidity and internal friction of three metals: brass, stainless steel, and aluminium.

The data collected is then plotted with Python code to generate graphs that visually represent the data and relationships.

2.1. National Instrument Setup

The National Instruments Programme (NIP) device must be set up for the multiple readings to be converted into computational data. This instrument is connected to the apparatus and a computer and provided voltages determined with simple coding, and interprets the laser sensor DAC values.

The voltage provided by the NIP must be converted into frequencies for graphical readings. This can be achieved by calibrating the instrument. For varying voltage readings, the corresponding frequencies read from the oscilloscope were noted on a computer. A linear relationship is the plotted which determines the ratio of voltage to frequency.

2.2. Health and Safety

Due to the presence of a high-intensity laser, care must be taken to avoid direct eye contact. The heater coil is very hot to the touch as are the metal specimen rods, so care must be taken when removing the metal rods from the apparatus as to not burn and injure oneself.

3. Results

From the Python code, graphs of the amplitude obtained from the laser sensor plotted against the frequency (voltage) from the oscilloscope were used to find the resonant frequency of each metal at varying temperatures 60°C, 90°C, 120°C, 150°C, and 180°C. With these values, the modulus of rigidity was found and plotted alongside the internal friction for each metal type: brass, stainless steel, and aluminium.

3.1. Amplitude vs. Frequency

The amplitude-frequency plots for the brass, stainless steel, and aluminium rods were plotted with a Lorentzian curve fitted, using:

$$L(x) = y_0 + \frac{2A}{\pi}\frac{\omega_0}{4(x - x_c)^2 + \omega^2} \tag{8}$$

with A as the area under the curve, ω_0 as the width of the curve at half maximum, y_0 the initial value, and x_c the estimated value of the peak. From this, the resonant peak was found from the plotted data fit.

3.1.1 Brass

For the brass rod, the different obtained data points and respective Lorentzian curve fits are shown with Figure 2. In Figure 4, a decreasing trend in frequency can be seen with increasing temperature, each peak remaining at approximately the same amplitude and therefore intensity. The curves for this metal type were narrow and symmetric and showed a clear-temperature dependence for the resonant frequency peak. The full width at half maximum $omega_0$ remained relatively constant, which indicates that the damping on the material was largely uniform over the

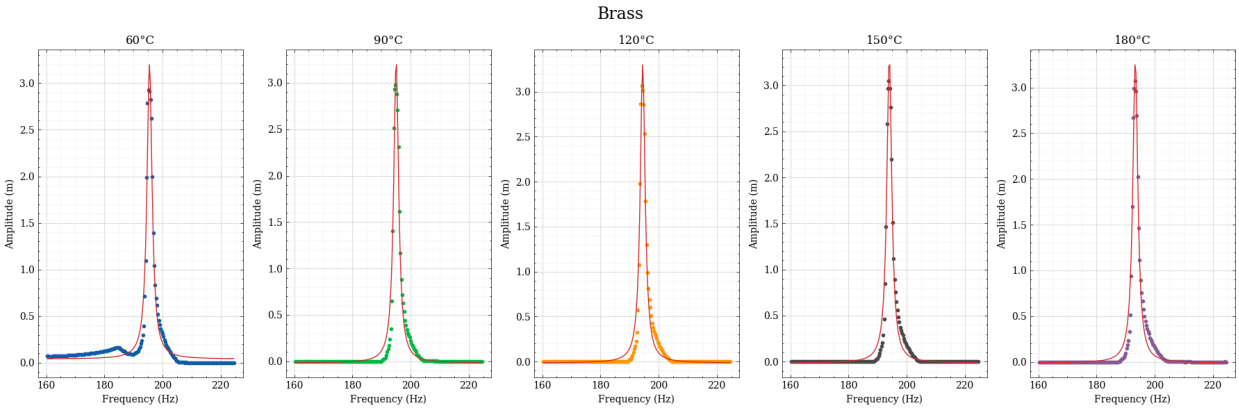


Figure 2: Brass values for varying temperatures with fitted Lorentzian.

entire temperature range. The resonant frequency over the range of temperatures was approximately **194.53 Hz**.

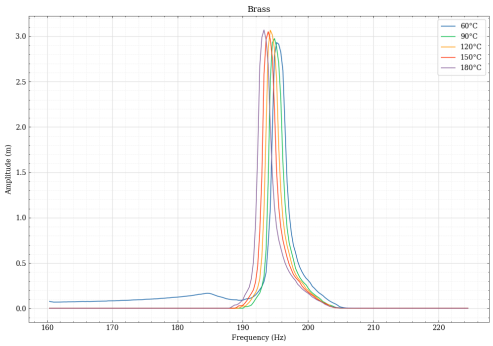


Figure 4: Plotted lines for brass values.

3.1.2 Stainless Steel

For the stainless steel rod, the data points obtained from the equipment were plotted with a Lorentzian curve fit for each temperature, shown in Figure 3. In Figure 5 showed a similar trend in decreasing resonant frequency with increasing temperature. The resonance peaks were very sharply defined and tall, which indicates high oscillation quality and low energy dissipation in the material. The variations in amplitude were very similar, with only minimal variations, making the stainless steel rod the most stable in terms of the resonant frequency and amplitude response. According to papers and lab coordinator however, the resonant frequency for this metal should be much higher than what was obtained, and should be approximately between what was obtained for brass and aluminium. The average resonant frequency for this metal was approximately **92.21**

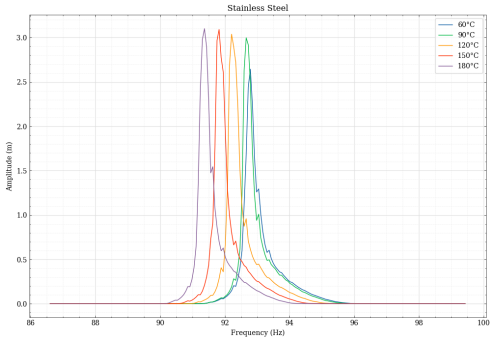


Figure 5: Plotted lines for stainless steel values.

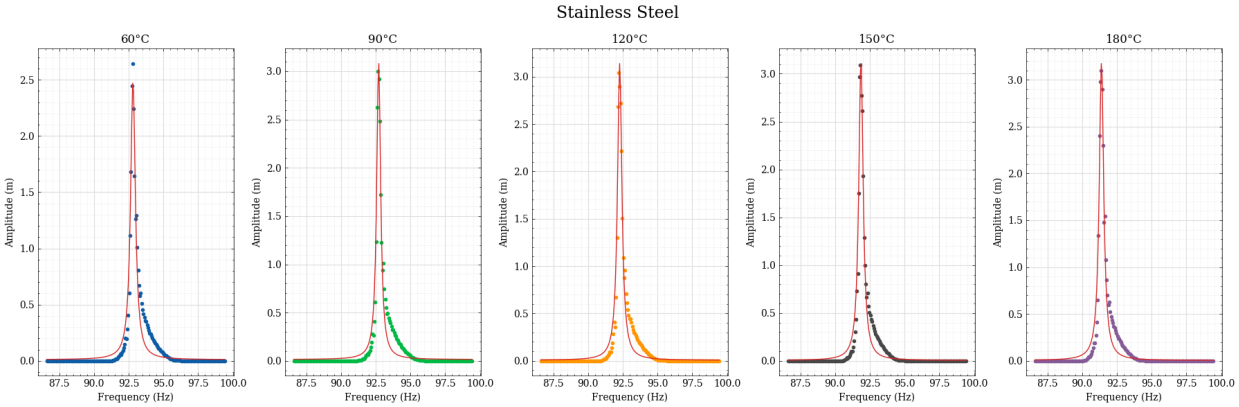


Figure 3: Stainless steel values for varying temperatures with fitted Lorentzian.

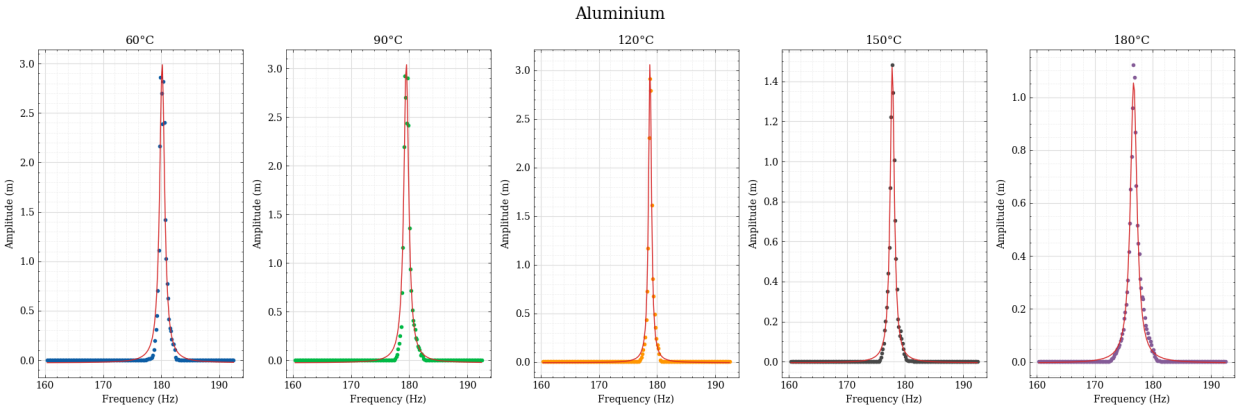


Figure 6: Aluminium values for varying temperatures with fitted Lorentzian.

Hz.

3.1.3 Aluminium

For aluminium, the data obtained was plotted on a graph and a Lorentzian curve was fitted to what was obtained, shown in Figure 6. The points as plotted in Figure 7 show a wider curve and considerably smaller resonance peaks. The graph shows a similar trend in decreasing resonant frequency with increasing temperature, but the amplitude is seen to be affected alongside it with an exponential decrease with linearly increasing temperatures. At higher temperatures, the aluminium rod appears to begin to flatten. The broader peaks suggest a greater damping effect and a decreased ability of the material to maintain torsional oscillations at resonance. The resonant frequency for aluminium averaged over the temperature range is approximately **178.59 Hz**.

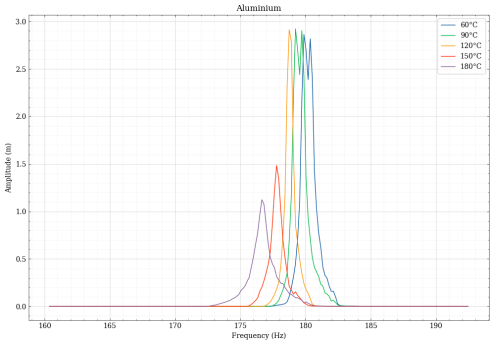


Figure 7: Plotted lines for aluminium values.

3.2. Modulus of Rigidity

The experimentally obtained values for the modulus of rigidity are shown in Table 1 and shown graphically in Figure 8 for brass, stainless steel, and aluminium with respect to temperature. Across all three metals, a clear trend of decreasing modulus with increasing temperature is observed. Values were calculated with Equation 6.

For brass, the gradual decrease from 24.3 GPa at 60°C to 23.8 GPa at 180°C shows a slope with a relatively small decline, indicating that the material maintained much of its torsional rigidity throughout the tested temperature range.

For stainless steel, the highest modulus of rigidity values were observed overall, ranging from 32.0 GPa at 60°C to 31.0 GPa at 180°C. The gradual decline of values is similar in behaviour to brass.

For aluminium, the modulus was lowest overall and showed the most pronounced decrease, ranging from 12.1 GPa at 60°C to 11.6 GPa at 180°C. The gradual decreasing behaviour was similar as previously.

3.3. Internal Friction

The values for the internal friction were measured and plotted against temperature with Equation 7. The obtained values are shown in Table 2 and graphically in Figure 9.

Small variations in brass between 0.00608 and 0.00632 remained relatively steady and indicated consistent damping characteristics.

Stainless steel showcased the lowest internal friction within the observed group, with values ranging from 0.00213 to 0.00258. The data showed little temperature dependence, indicating stable internal energy dissipation.

Aluminium internal friction values ranged from 0.00190 to 0.00444. The broader spread in the values correspond to the iregular resonance profiles observed previously.

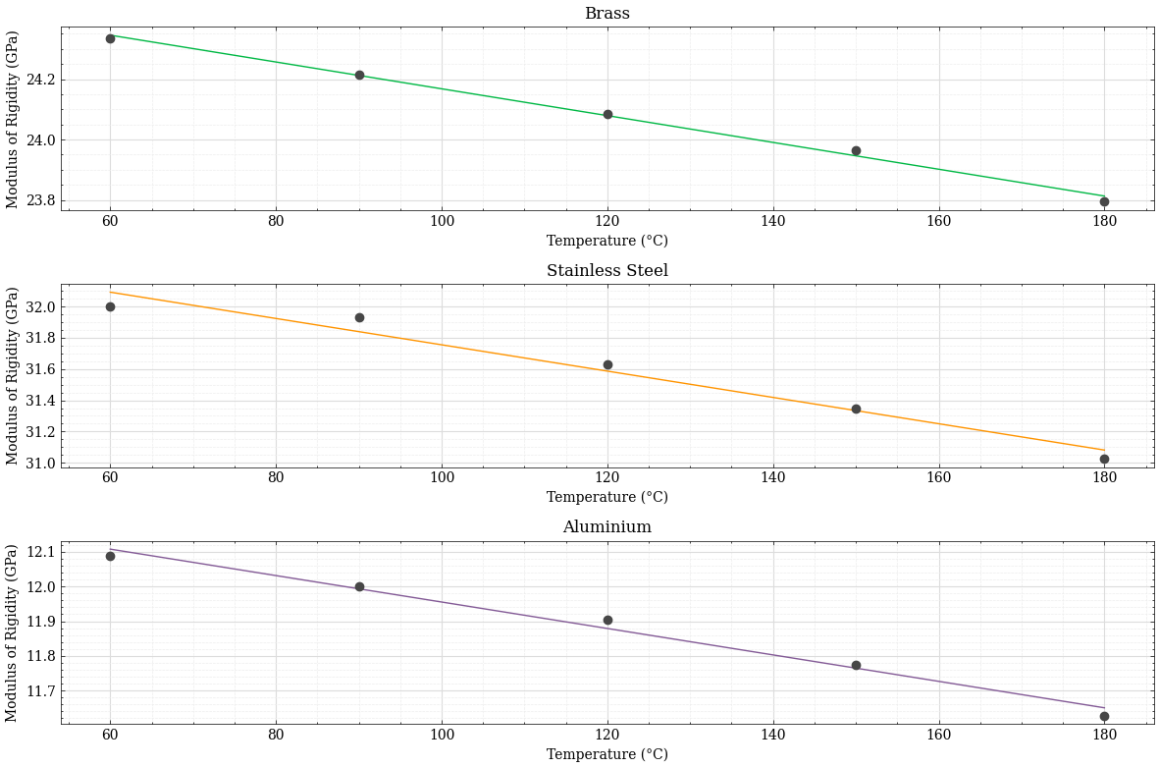


Figure 8: Graph of the modulus of rigidity for varying temperature for: brass (top), stainless steel (middle), aluminium (bottom).

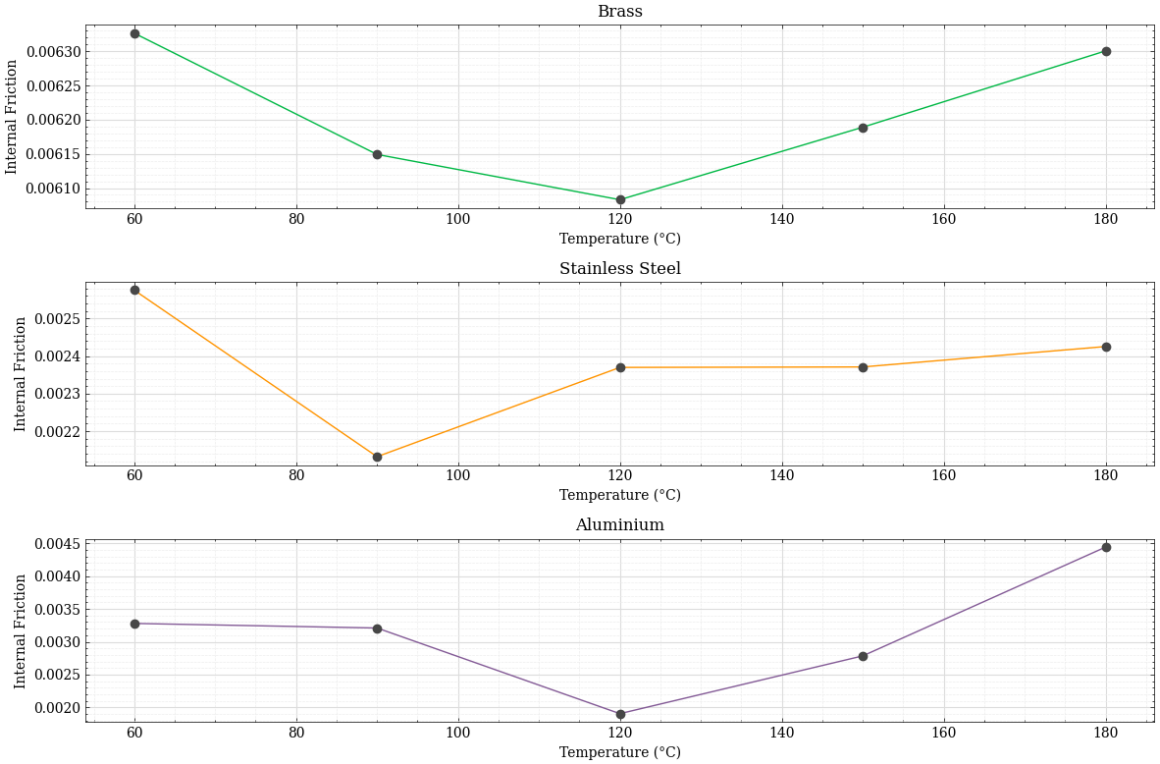


Figure 9: Graph of the internal friction for varying temperature for: brass (top), stainless steel (middle), aluminium (bottom)

4. Analysis and Discussion

The amplitude-frequency plots obtained for the brass, stainless steel, and aluminium metal rods demonstrated a clear resonant peak. This resonant frequency value decreased with increasing temperatures across all metals, which is behaviour predicted by theory as elasticity increases with increase in temperature, leading to a decrease in rigidity and therefore difference in resonant frequency. The approximate values for the resonant frequency averaged for the varying temperatures were found to be **194.53 Hz** for brass, **92.21 Hz** for stainless steel, and **178.59 Hz** for aluminium. The expected ordering places stainless steel between brass and aluminium, however no resonance was observed at those frequencies. This may suggest systematic error or changes to the composition from environmental factors, or incorrect wiring of the experimental setup.

Brass produced a narrow and symmetric Lorentzian curve shape, which indicated relatively low damping and consistent internal energy losses across the experimentally observed temperature range. The full width at half maximum remained almost constant, which therefore implied that the Internal friction did not vary with temperature. The small decrease of the modulus of rigidity for the relative values obtained showcase a torsional stability for the metal.

Stainless steel showcased the tallest and most narrow resonance peaks when fitted with the Lorentzian, which indicated low damping and high torsional oscillation quality. The internal friction values measured were the lowest, which support this interpretation of results, with its modulus of rigidity decreasing between only 32.0 GPa and 31.0 GPa with temperature variations, implying strong thermal resilience. However, the resonant frequency achieved for this metal was much lower than expected, suggesting that the effective torsional constant of the specimen rod was reduced. This discrepancy may be attributed to clamping inconsistencies, magnet misalignment, or imperfections in the rod material and shape.

Aluminium exhibited the broadest resonant frequency curves, with observationally decreasing amplitude with increasing temperature. This may suggest the material's poor ability to sustain oscillations, increased damping, and higher sensitivity of the modulus or rigidity for varying temperatures. The reduction in the modulus of rigidity, when accompanied by the values obtained for the internal friction, are consistent with the broader, loss sharply defined resonance peaks observed in the graphs.

The trend observed across all metals for the gradual decline in the modulus of rigidity with increasing temperature align with theory, in which atomic vibrations in the material due to temperature produce poorly coordinated packings which reduce the shear modulus. Internal friction showed weaker dependence on the temperature, with brass remaining relatively constant, stainless steel remained low with minor fluctuation, and stainless steel displayed irregularity consistent with the behaviour observed with a broader bandwidth and increased internal friction at higher temperature.

Sources of error in experimentation include misalignment of the mirror-laser path, too-high gain given to the voltage that produces amplitude greater than the sensor is able to detect resulting in cut-off values (dips shown in Fig. 6 for lower temperatures), non-uniform heating by the heating coil, calibration errors in the voltage-frequency conversion, environmental temperatures interacting with circuitry, or variability in torque in the specimen rods by the initial taps. Furthermore, the experimental values obtained for the modulus of rigidity were significantly lower than published values (Table 1), indicating losses due to experimental constraints, imperfect rod geometry, residual damping from the support stand and clamp, or errors in calculations and coding. Higher temperature readings for the metals could have been taken with proper equipment, which would offer a better look at the behaviour of the composition, specifically with the internal friction calculations.

Overall, despite deviations in absolute values, the trends observed from the data obtained for resonance behaviour, modulus of rigidity, and internal friction meaningfully reflect the material properties.

5. Conclusion

The experiment successfully demonstrated how temperature influences the rigidity and energy dissipation of brass, stainless steel, and aluminium through a torsional oscillator setup. Across all samples, the resonant frequency and shear mod-

ulus gradually decreased with increasing temperatures, showcasing the behaviour of atomic packing weakening with thermal influence. Stainless steel exhibited the highest modulus of rigidity and lowest internal friction across the three specimen, indicating superior torsional stability. Aluminium showcased the most pronounced damping behaviour and lowest modulus of rigidity. Brass showcased intermediate behaviour with consistent resonance profiles and minimal variations in internal friction values.

Although the achieved absolute values for the modulus of rigidity were significantly lower than those expected from published literature, the observed trends were physically consistent. Discrepancies likely arose from experimental limitations such as imperfections on the specimen rod, uneven heating, mirror-laser minor misalignments, incorrect gain provided, and damping from the clamping. Despite these factors, the experiment clearly graphically showcased the temperature dependence on torsional oscillations, confirming theoretical expectations and demonstrating how material microstructures and composition influence internal friction. Further improvements in apparatus calibration and temperature control would refine measurement accuracy and extend investigation on the factors discussed.

References

[1] R. Fitzpatrick, "The torsion pendulum," Online lecture notes, *The Farside (University of Texas)*, 2006, [Accessed 07-November-2025]. [Online]. Available: https://farside.ph.utexas.edu/teaching/301/lectures/node139.html

[2] G. Shanker, V. Gupta, B. Saraf, and N. Sharma, "Temperature variation of modulus of rigidity and internal friction: An experiment with torsional oscillator," *American Journal of Physics*, vol. 53, no. 12, pp. 1192–1195, 1985.

[3] J. Tatum, *Classical Mechanics*. LibreTexts, 2020, [Accessed 07-November-2025]. [Online]. Available: https://phys.libretexts.org/Bookshelves/Classical_Mechanics/Classical_Mechanics_(Tatum)/11:_Simple_and_Damped_Oscillatory_Motion/11.03:_Torsion_Pendulum

[4] J. Filippini, "Torsional oscillator (lecture 07, phys 401 spring 2020)," Lecture notes, Department of Physics, University of Illinois at Urbana-Champaign, 2020, [Accessed 07-November-2025]. [Online]. Available: https://courses.physics.illinois.edu/phys401/sp2020/lectures/lecture07.pdf

[5] S. Ghosh and H. Alsaud. (2023) Shear modulus formula, equation & units. Study.com. [Accessed 07-November-2025]. [Online]. Available: https://study.com/academy/lesson/modulus-of-rigidity-definition-equation.html

[6] C. Trento. (2025) Shear modulus (modulus of rigidity). Stanford Advanced Materials. [Accessed 07-November-2025]. [Online]. Available: https://www.samaterials.com/content/shear-modulus-%28modulus-of-rigidity%29.html

[7] I. Agnolin and J.-N. Roux, "Internal states of model isotropic granular packings. iii. elastic properties," *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, vol. 76, no. 6, p. 061304, 2007.

[8] M. Knapek. (2025) Resonance and internal friction. Charles University, Faculty of Mathematics and Physics, Department of Physics of Materials. [Accessed 07-November-2025]. [Online]. Available: https://www.mff.cuni.cz/en/kfm/experimental-facilities/resonance-and-internal-friction

[9] J. Hyde and C. English, "9 - microstructural characterisation techniques for the study of reactor pressure vessel (rpv) embrittlement," in *Irradiation Embrittlement of Reactor Pressure Vessels (RPVs) in Nuclear Power Plants*, ser. Woodhead Publishing Series in Energy, N. Soneda, Ed. Woodhead Publishing, 2015, pp. 211–294. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978184569967350009X

Appendix

Table 1: Table for the obtained and expected values of the modulus of rigidity for brass, stainless steel, aluminium in GPa.

	Modulus of Rigidity (GPa)					
Metal	60°C	90°C	120°C	150°C	180°C	Expected (GPa)
Brass	24.3	24.2	24.1	24.0	23.8	40.0
Stainless Steel	32.0	31.9	31.6	31.4	31.0	78.0
Aluminium	12.1	12.0	11.9	11.8	11.6	26.6

Table 2: Table for the obtained values of the internal friction for brass, stainless steel, aluminium.

	Internal Friction				
Metal	60°C	90°C	120°C	150°C	180°C
Brass	0.00632	0.00615	0.00608	0.00619	0.00630
Stainless Steel	0.00258	0.00213	0.00237	0.00237	0.00243
Aluminium	0.00328	0.00321	0.00190	0.00278	0.00444

modex_graphcode

November 9, 2025

[71]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.optimize import curve_fit

import scienceplots
```

[72]:
```
plt.style.use('science')

plt.rcParams['figure.figsize'] = [12, 8]
plt.rcParams['text.usetex'] = False

plt.rcParams['axes.prop_cycle'] = plt.cycler(color=[
'#0C5DA5', '#00B945', '#FF9500', '#FF2C00', '#845B97', '#474747', '#9e9e9e'
])
```

[73]:
```
T = np.array([60, 90, 120, 150, 180])
```

1 Modulus of Rigidity

[74]:
```
# eq.5
def mod_rigid(I, l, omega0, r):
    return (2 * I * l * omega0**2)/(np.pi * r**4)

# I = moment of inertia
# l = length
# omega0 = angular frequency
# r = radius
```

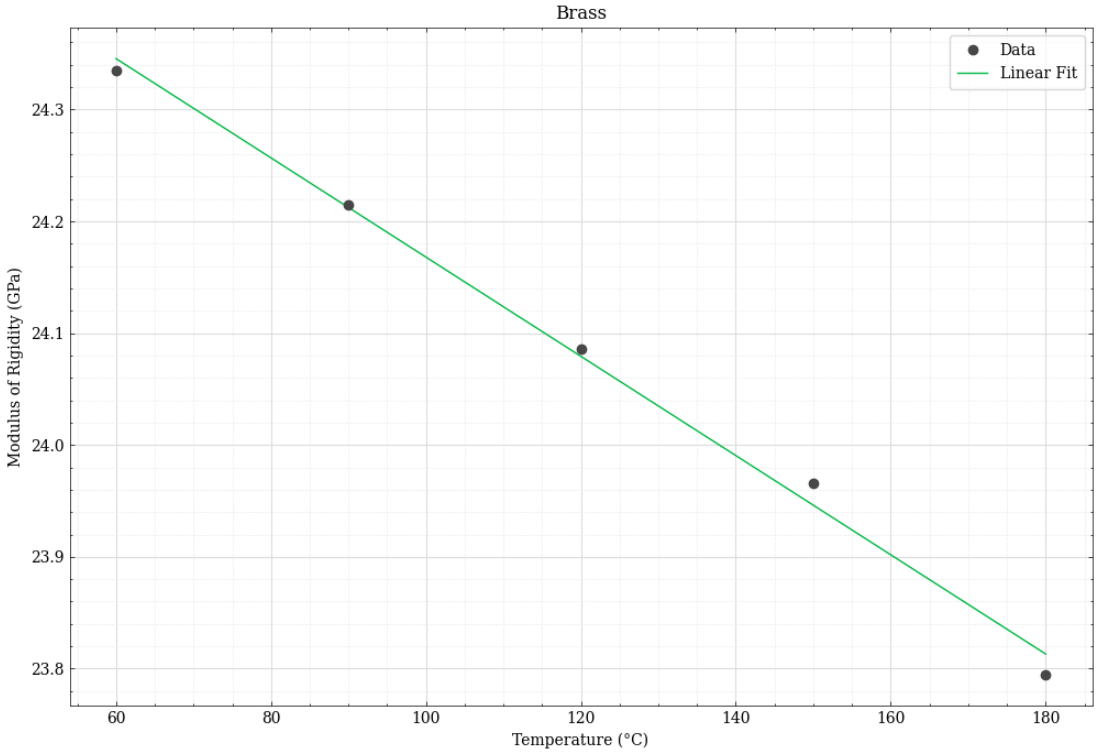
[75]:
```
mag_m = 54.54e-3
mag_r = 9.35e-3 / 2
```

[133]:
```
# brass - T= 60, 90, 120, 150, 180

m =  23.55e-3
l =  35.5e-2
r = 3.20e-3 / 2
```

```
I = 0.5 * m * r**2 + 0.5 * mag_m * mag_r**2
fres = np.array([195.558052, 195.075226, 194.552084, 194.069886, 193.373124])
omega0 = 2 * np.pi * fres

BG = mod_rigid(I, l, omega0, r)

print(BG * 1e-9)

print(np.sum(fres)/len(fres))
```

```
[32.5998737  32.43909653 32.26534305 32.10560174 31.87548045]
194.5256744
```

[77]:
```
Bval = np.polyfit(T, BG, 1)
Bfit = np.poly1d(Bval) * 1e-9

plt.plot(T, BG * 1e-9, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, Bfit(T), label='Linear Fit', color='#00B945')

plt.xlabel('Temperature (°C)')
plt.ylabel('Modulus of Rigidity (GPa)')
plt.title('Brass')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```

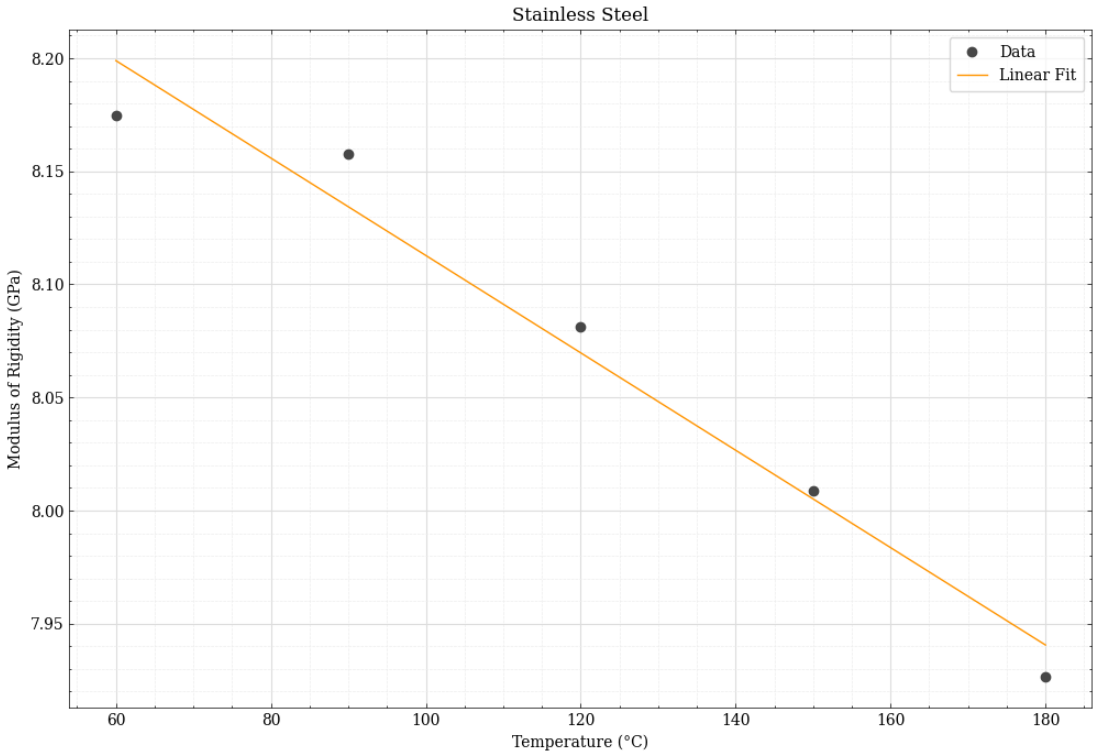


[ ]:
```
m =  22.23e-3
l =  35e-2
r = 3.10e-3 / 2
I = 0.5 * m * r**2 + 0.5 * mag_m * mag_r**2
fres = np.array([92.813146, 92.714482, 92.279800, 91.864033, 91.391520])
omega0 = 2 * np.pi * fres

SSG = mod_rigid(I, l, omega0, r)

print(SSG)

print(np.sum(fres)/len(fres))
```

```
[8.17492213e+09 8.15755084e+09 8.08123853e+09 8.00858248e+09
 7.92640826e+09]
92.2125962
```

[79]:
```
SSval = np.polyfit(T, SSG, 1)
SSfit = np.poly1d(SSval) * 1e-9

plt.plot(T, SSG * 1e-9, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, SSfit(T), label='Linear Fit', color='#FF9500')
```

```
plt.xlabel('Temperature (°C)')
plt.ylabel('Modulus of Rigidity (GPa)')
plt.title('Stainless Steel')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```

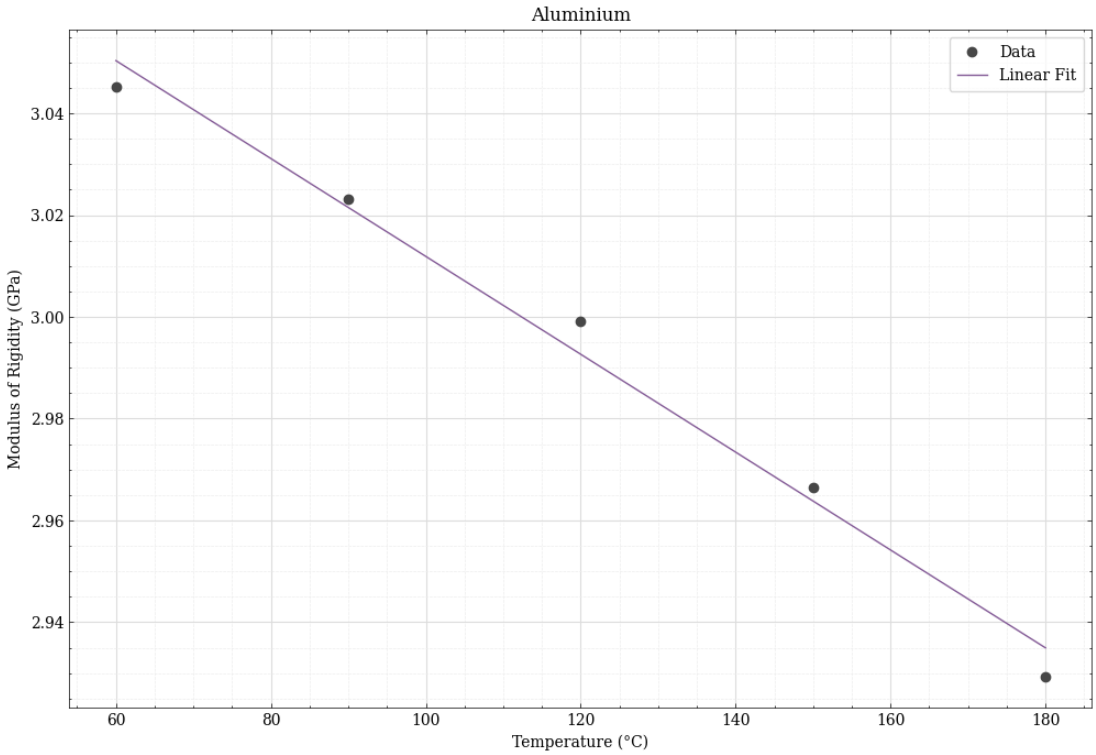


[132]:
```
# aluminium - l.grey - T= 60, 90, 120, 150, 180

m = 7.71e-3
l = 35.6e-3
r = 3.10e-3 / 2
I = 0.5 * m * r**2 + 0.5 * mag_m * mag_r**2
fres = np.array([180.156115, 179.506483, 178.792279, 177.814923, 176.694473])
omega0 = 2 * np.pi * fres
```

```
AG = mod_rigid(I, l, omega0, r)

print(AG * 1e-9)

print(np.sum(fres)/len(fres))
```

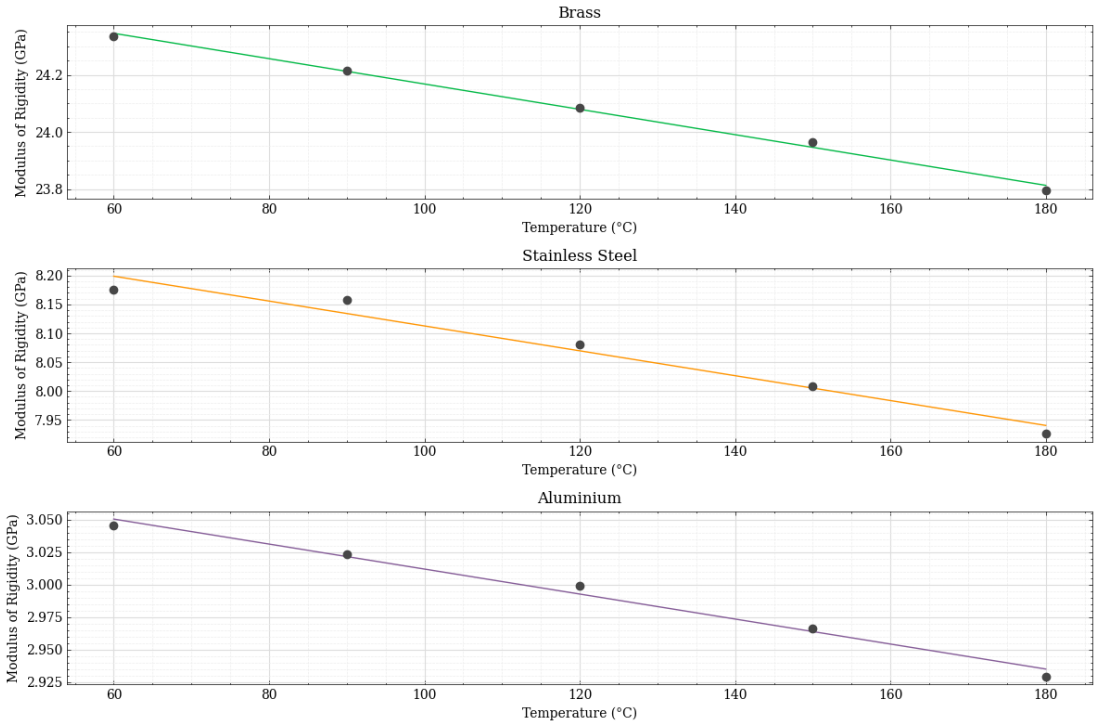
```
[3.04513328 3.02321174 2.99920264 2.96650239 2.92923503]
178.5928546
```

[81]:
```
Aval = np.polyfit(T, AG, 1)
Afit = np.poly1d(Aval) * 1e-9


plt.plot(T, AG * 1e-9, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, Afit(T), label='Linear Fit', color='#845B97')

plt.xlabel('Temperature (°C)')
plt.ylabel('Modulus of Rigidity (GPa)')
plt.title('Aluminium')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```



[82]:
```
fig, ax = plt.subplots(3, 1)

ax[0].plot(T, BG * 1e-9, 'o', label='Data', zorder=3, color='#474747')
ax[0].plot(T, Bfit(T), label='Linear Fit', color='#00B945')
ax[0].minorticks_on()
ax[0].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[0].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[0].set_ylabel('Modulus of Rigidity (GPa)')
ax[0].set_xlabel('Temperature (°C)')
ax[0].set_title('Brass')

ax[1].plot(T, SSG * 1e-9, 'o', label='Data', zorder=3, color='#474747')
ax[1].plot(T, SSfit(T), label='Linear Fit', color='#FF9500')
ax[1].minorticks_on()
ax[1].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[1].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[1].set_ylabel('Modulus of Rigidity (GPa)')
ax[1].set_xlabel('Temperature (°C)')
ax[1].set_title('Stainless Steel')
```

```
ax[2].plot(T, AG *1e-9, 'o', label='Data', zorder=3, color='#474747')
ax[2].plot(T, Afit(T), label='Linear Fit', color='#845B97')
ax[2].minorticks_on()
ax[2].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[2].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[2].set_ylabel('Modulus of Rigidity (GPa)')
ax[2].set_xlabel('Temperature (°C)')
ax[2].set_title('Aluminium')

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



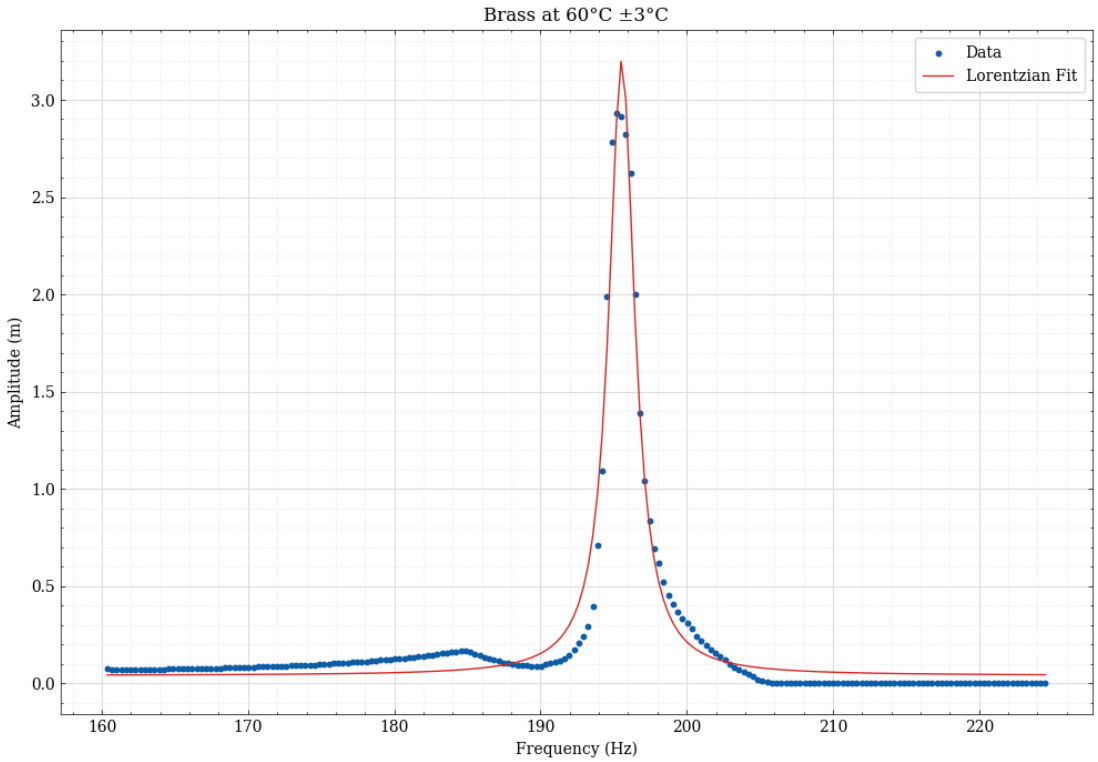
2 Fitting

[83]:
```
def lorentz_fit(x, x0, omega, A, y0):
    return y0 + (2 * A)/(np.pi) * omega/(4 * (x-x0)**2 + omega**2)

# A = area under curve
# omega = width of curve at half max A
# y0 = initial value
# x0 = peak
```

2.1 Brass

[84]:
```
B60C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/brass 60C')

VB60C = np.array(B60C[0]) * 64.15
ampB60C = np.array(B60C[1])
```

[85]:
```
initial_guess = [VB60C[np.argmax(ampB60C)], min(ampB60C), (max(ampB60C) -␣
↪min(ampB60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB60C, ampB60C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VB60C, ampB60C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VB60C, lorentz_fit(VB60C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass at 60°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

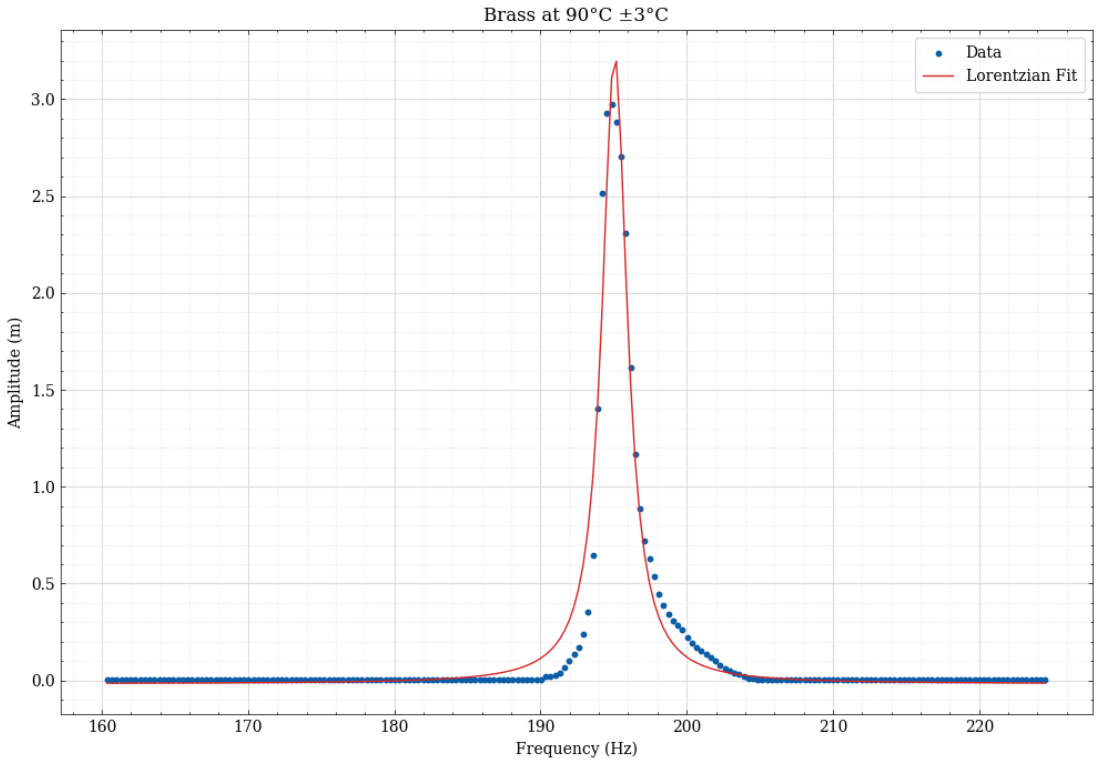



```
Peak from fit: 195.558052
Width of peak: 2.142843
```

[86]:
```
B90C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
  ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
  ↪labcode_s3/brass 90C')

VB90C = np.array(B90C[0]) * 64.15
ampB90C = np.array(B90C[1])
```

[87]:
```
initial_guess = [VB90C[np.argmax(ampB90C)], min(ampB90C), (max(ampB90C) -␣
  ↪min(ampB90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB90C, ampB90C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VB90C, ampB90C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VB90C, lorentz_fit(VB90C, *popt), color='tab:red', label='Lorentzian␣
  ↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass at 90°C ±3°C')
```

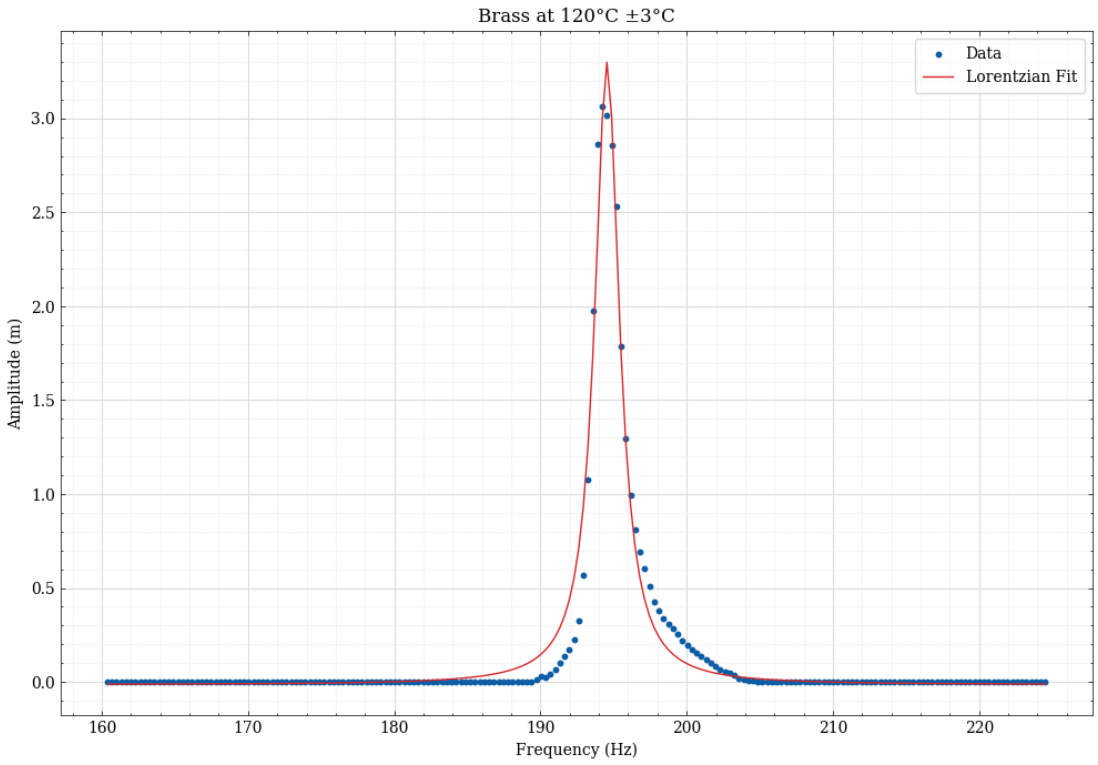
```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```




```
Peak from fit: 195.075226
Width of peak: 2.077667
```

[88]:
```
B120C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/brass 120C')

VB120C = np.array(B120C[0]) * 64.15
ampB120C = np.array(B120C[1])
```

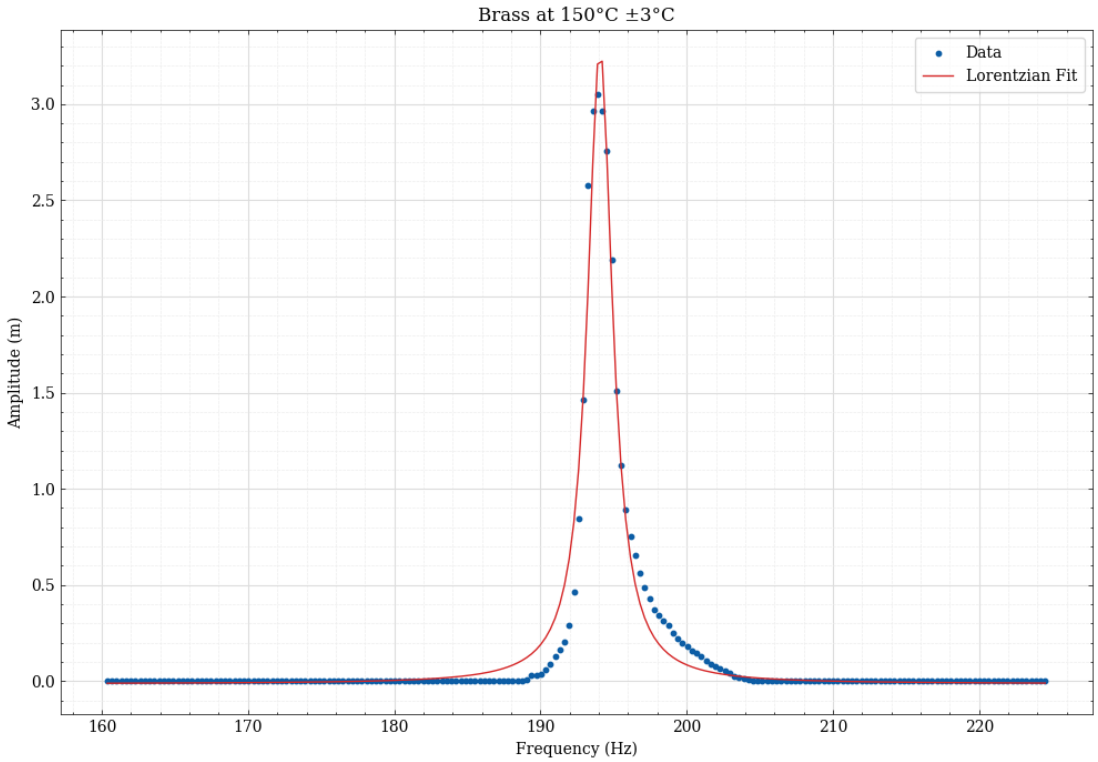
[89]:
```
initial_guess = [VB120C[np.argmax(ampB120C)], min(ampB120C), (max(ampB120C) -␣
↪min(ampB120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB120C, ampB120C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VB120C, ampB120C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VB120C, lorentz_fit(VB120C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass at 120°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```



```
Peak from fit: 194.552084
Width of peak: 2.049796
```

[90]:
```
B150C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
  ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
  ↪labcode_s3/brass 150C')

VB150C = np.array(B150C[0]) * 64.15
ampB150C = np.array(B150C[1])
```

[91]:
```
initial_guess = [VB150C[np.argmax(ampB150C)], min(ampB150C), (max(ampB150C) -␣
  ↪min(ampB150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB150C, ampB150C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VB150C, ampB150C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VB150C, lorentz_fit(VB150C, *popt), color='tab:red', label='Lorentzian␣
  ↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass at 150°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
  ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

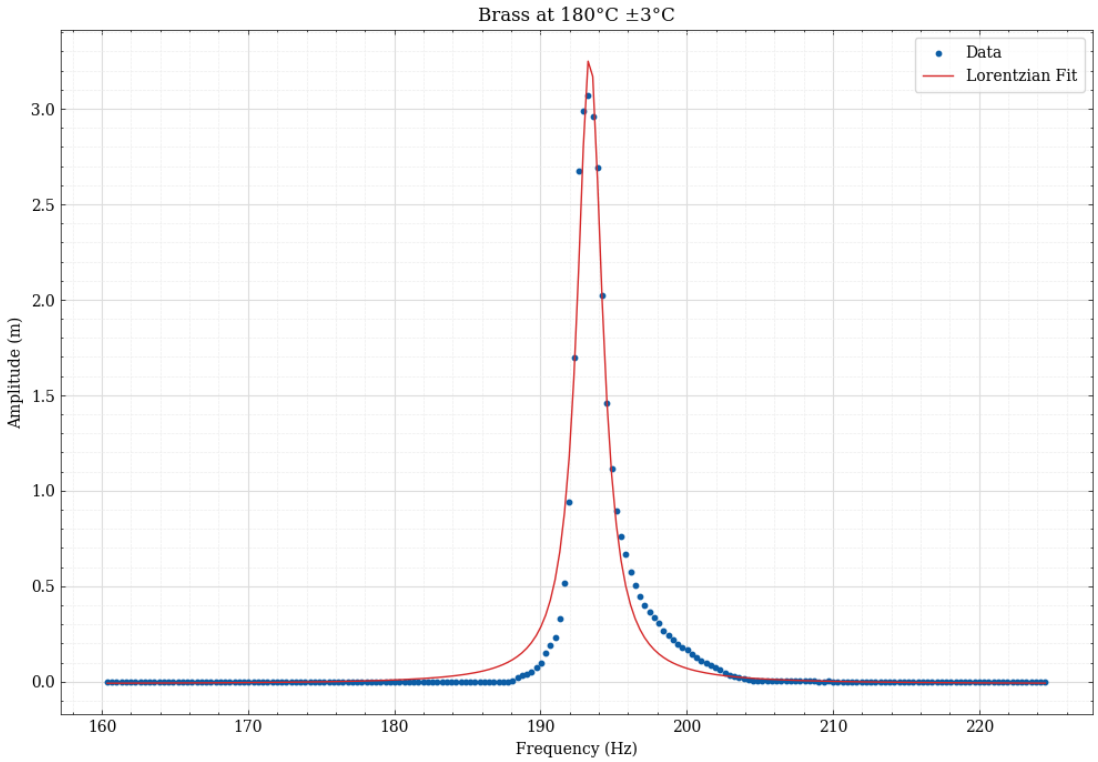



```
Peak from fit: 194.069886
Width of peak: 2.080339
```

[92]:
```
B180C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
  ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
  ↪labcode_s3/brass 180C')

VB180C = np.array(B180C[0]) * 64.15
ampB180C = np.array(B180C[1])
```

[93]:
```
initial_guess = [VB180C[np.argmax(ampB180C)], min(ampB180C), (max(ampB180C) -␣
  ↪min(ampB180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB180C, ampB180C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VB180C, ampB180C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VB180C, lorentz_fit(VB180C, *popt), color='tab:red', label='Lorentzian␣
  ↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass at 180°C ±3°C')
```

```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

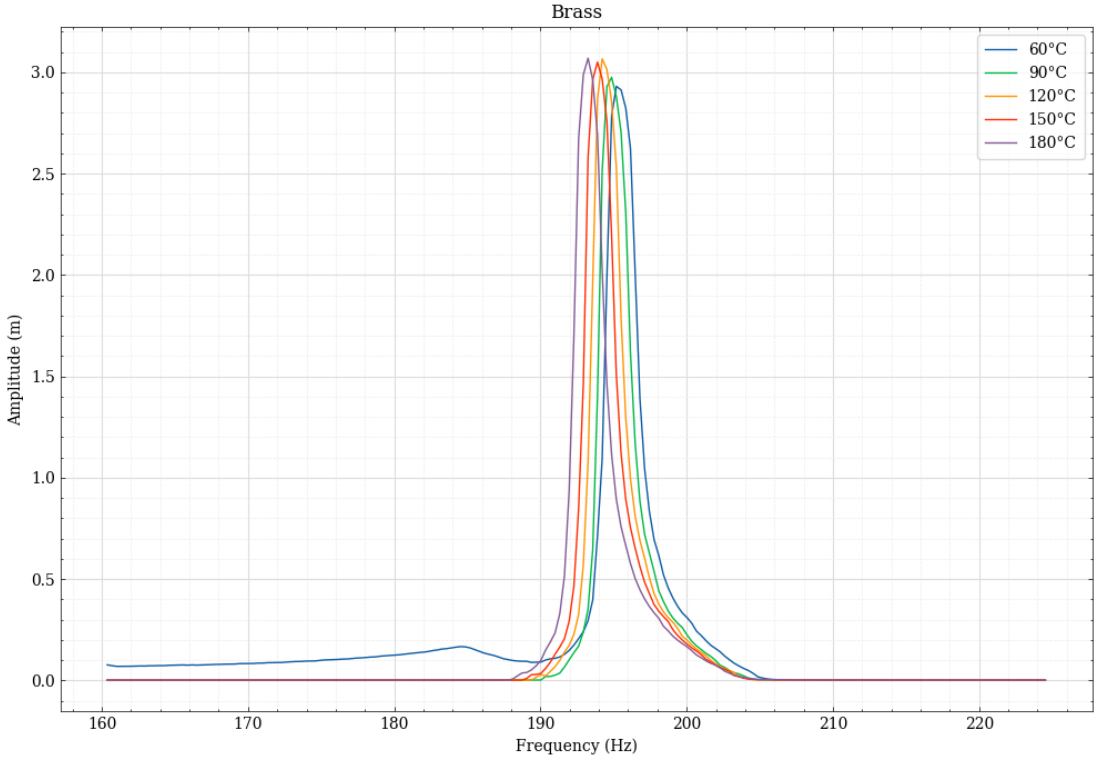



```
Peak from fit: 193.373124
Width of peak: 2.110216
```

2.1.1 ALL

[94]:
```
plt.plot(VB60C, ampB60C, label='60°C', zorder=3)
plt.plot(VB90C, ampB90C, label='90°C', zorder=4)
plt.plot(VB120C, ampB120C, label='120°C', zorder=5)
plt.plot(VB150C, ampB150C, label='150°C', zorder=6)
plt.plot(VB180C, ampB180C, label='180°C', zorder=7)
```

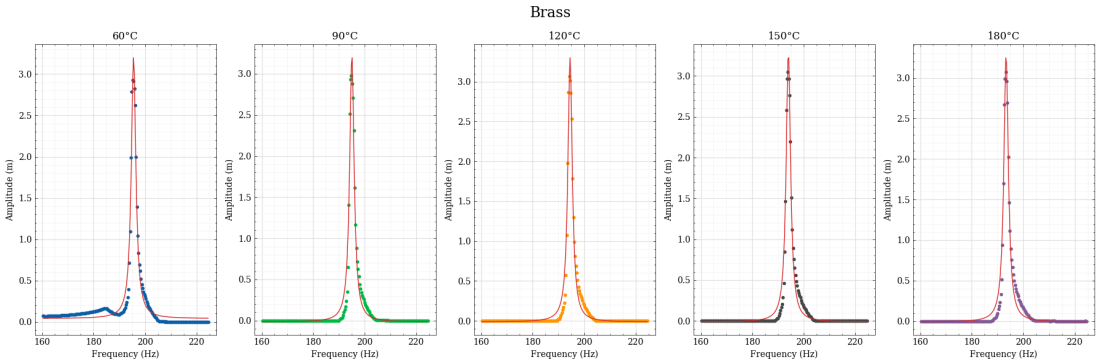
```
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Brass')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```



[95]:
```
fig, ax = plt.subplots(1,5, figsize=(18,6))
ax = ax.flatten()

initial_guess = [VB60C[np.argmax(ampB60C)], min(ampB60C), (max(ampB60C) -␣
↪min(ampB60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB60C, ampB60C, p0=initial_guess)
x0, omega, A, y0 = popt
```

```
ax[0].plot(VB60C, ampB60C, zorder=3, marker='.', linestyle='none', label='60°C')
ax[0].plot(VB60C, lorentz_fit(VB60C, *popt), color='tab:red', zorder=4)
ax[0].minorticks_on()
ax[0].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[0].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Amplitude (m)')
ax[0].set_title('60°C')

initial_guess = [VB90C[np.argmax(ampB90C)], min(ampB90C), (max(ampB90C) -␣
 ↪min(ampB90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB90C, ampB90C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[1].plot(VB90C, ampB90C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#00B945', label='90°C')
ax[1].plot(VB90C, lorentz_fit(VB90C, *popt), color='tab:red', zorder=4)
ax[1].minorticks_on()
ax[1].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[1].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Amplitude (m)')
ax[1].set_title('90°C')

initial_guess = [VB120C[np.argmax(ampB120C)], min(ampB120C), (max(ampB120C) -␣
 ↪min(ampB120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB120C, ampB120C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[2].plot(VB120C, ampB120C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#FF9500', label='120°C')
ax[2].plot(VB120C, lorentz_fit(VB120C, *popt), color='tab:red', zorder=4)
ax[2].minorticks_on()
ax[2].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[2].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Amplitude (m)')
ax[2].set_title('120°C')

initial_guess = [VB150C[np.argmax(ampB150C)], min(ampB150C), (max(ampB150C) -␣
 ↪min(ampB150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB150C, ampB150C, p0=initial_guess)
x0, omega, A, y0 = popt
```

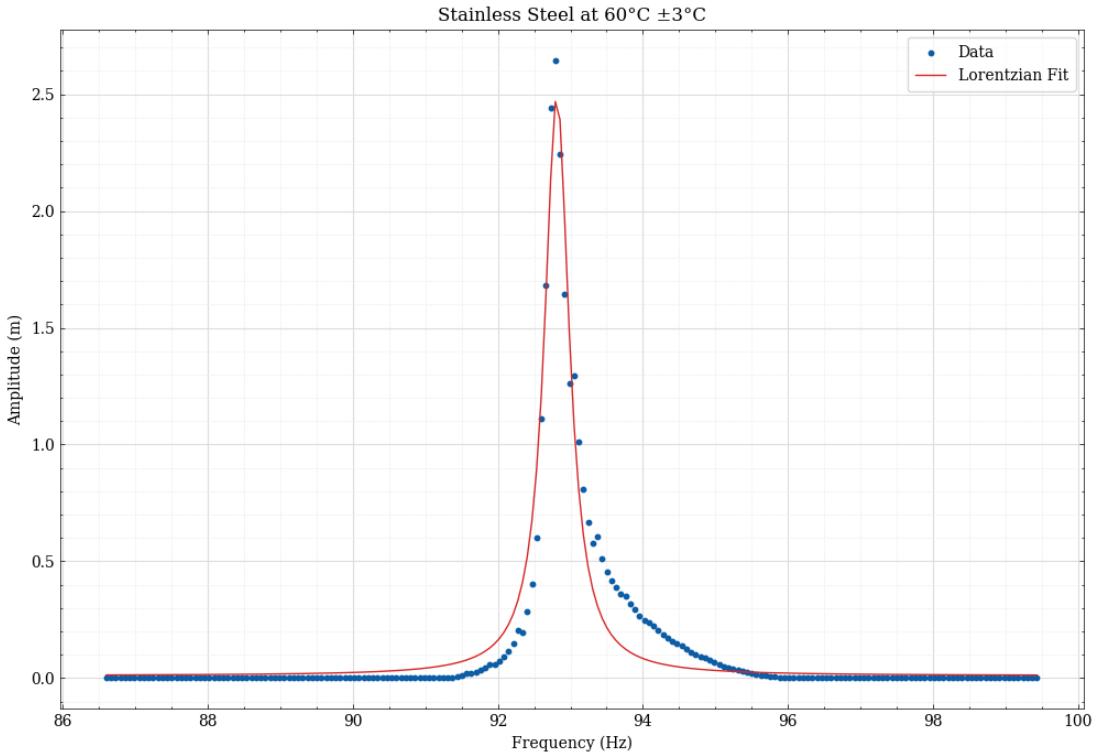
```
ax[3].plot(VB150C, ampB150C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#474747', label='150°C')
ax[3].plot(VB150C, lorentz_fit(VB150C, *popt), color='tab:red', zorder=4)
ax[3].minorticks_on()
ax[3].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[3].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Amplitude (m)')
ax[3].set_title('150°C')

initial_guess = [VB180C[np.argmax(ampB180C)], min(ampB180C), (max(ampB180C) -␣
 ↪min(ampB180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VB180C, ampB180C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[4].plot(VB180C, ampB180C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#845B97', label='180°C')
ax[4].plot(VB180C, lorentz_fit(VB180C, *popt), color='tab:red', zorder=4, ␣
 ↪label='Lorentzian Fit')
ax[4].minorticks_on()
ax[4].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[4].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Amplitude (m)')
ax[4].set_title('180°C')

fig.suptitle('Brass', size=17)
fig.tight_layout()

plt.show()
```



2.2 Stainless Steel

[96]:
```
SS60C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/stainless steel 60C2')

VSS60C = np.array(SS60C[0]) * 64.15
ampSS60C = np.array(SS60C[1])
```

[97]:
```
initial_guess = [VSS60C[np.argmax(ampSS60C)], min(ampSS60C), (max(ampSS60C) -␣
↪min(ampSS60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS60C, ampSS60C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VSS60C, ampSS60C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VSS60C, lorentz_fit(VSS60C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel at 60°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

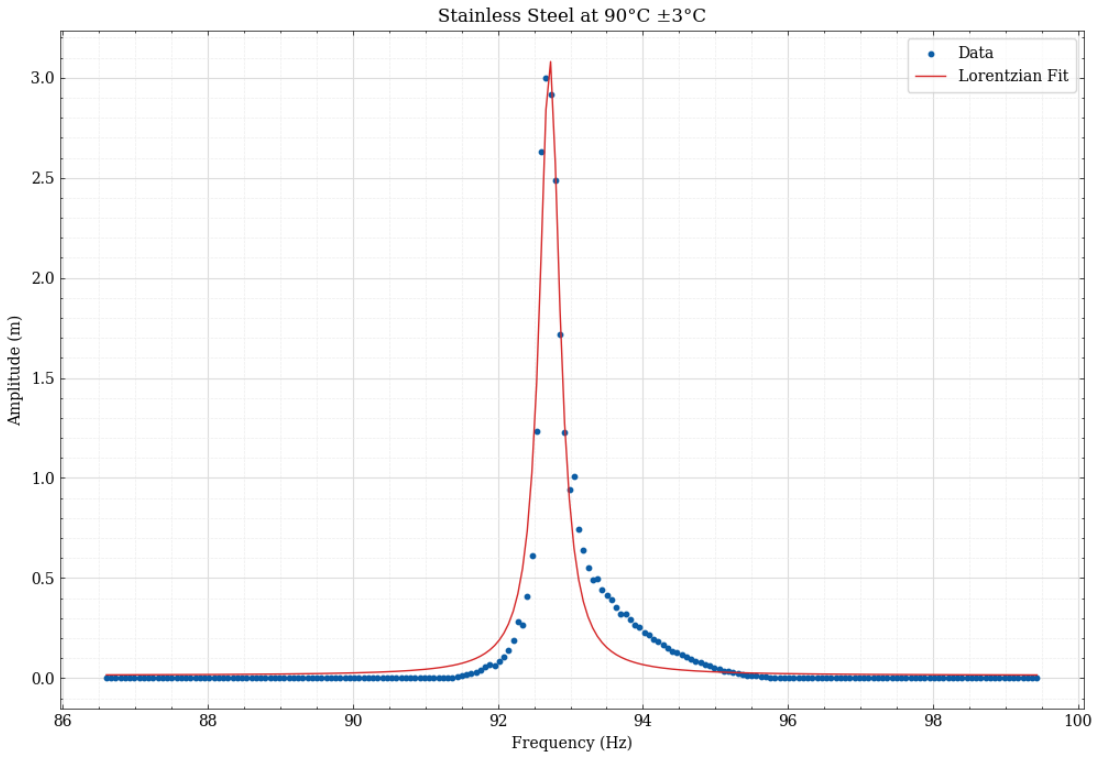



```
Peak from fit: 92.813146
Width of peak: 0.414041
```

[98]:
```
SS90C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
  ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
  ↪labcode_s3/stainless steel 90C2')

VSS90C = np.array(SS90C[0]) * 64.15
ampSS90C = np.array(SS90C[1])
```

[99]:
```
initial_guess = [VSS90C[np.argmax(ampSS90C)], min(ampSS90C), (max(ampSS90C) -␣
  ↪min(ampSS90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS90C, ampSS90C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VSS90C, ampSS90C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VSS90C, lorentz_fit(VSS90C, *popt), color='tab:red', label='Lorentzian␣
  ↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel at 90°C ±3°C')
```

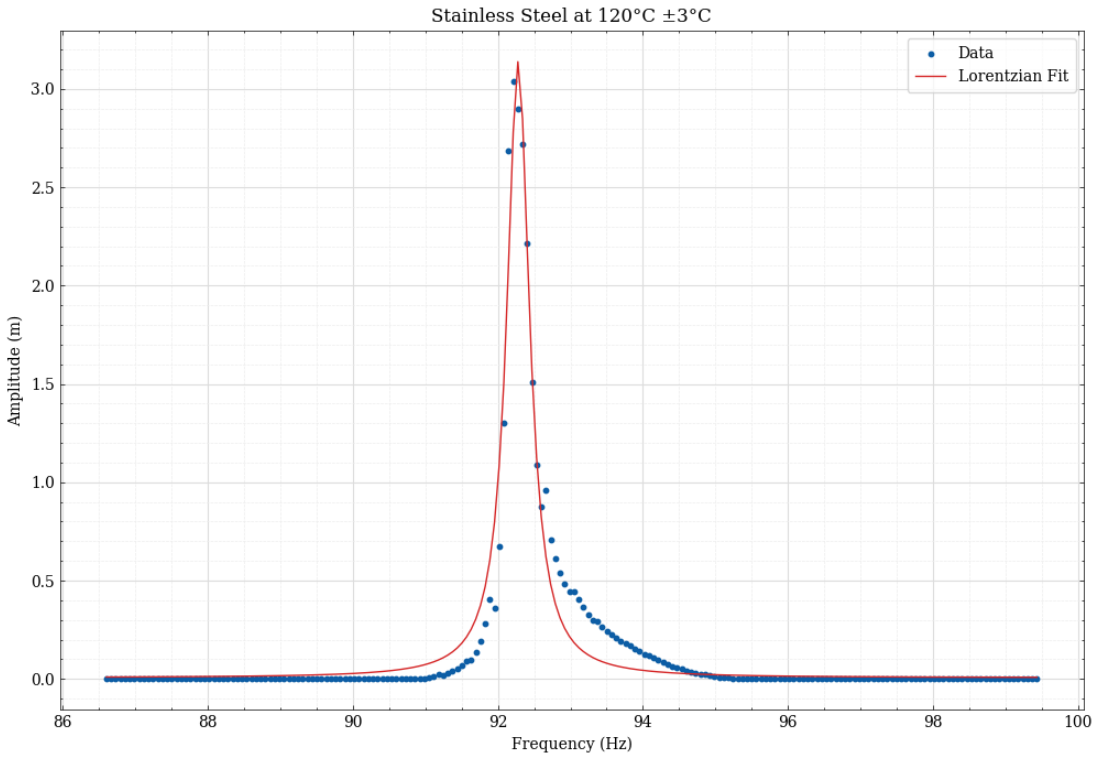
```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```




```
Peak from fit: 92.714482
Width of peak: 0.342378
```

[100]:
```
SS120C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
 ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
 ↪labcode_s3/stainless steel 120C2')

VSS120C = np.array(SS120C[0]) * 64.15
ampSS120C = np.array(SS120C[1])
```

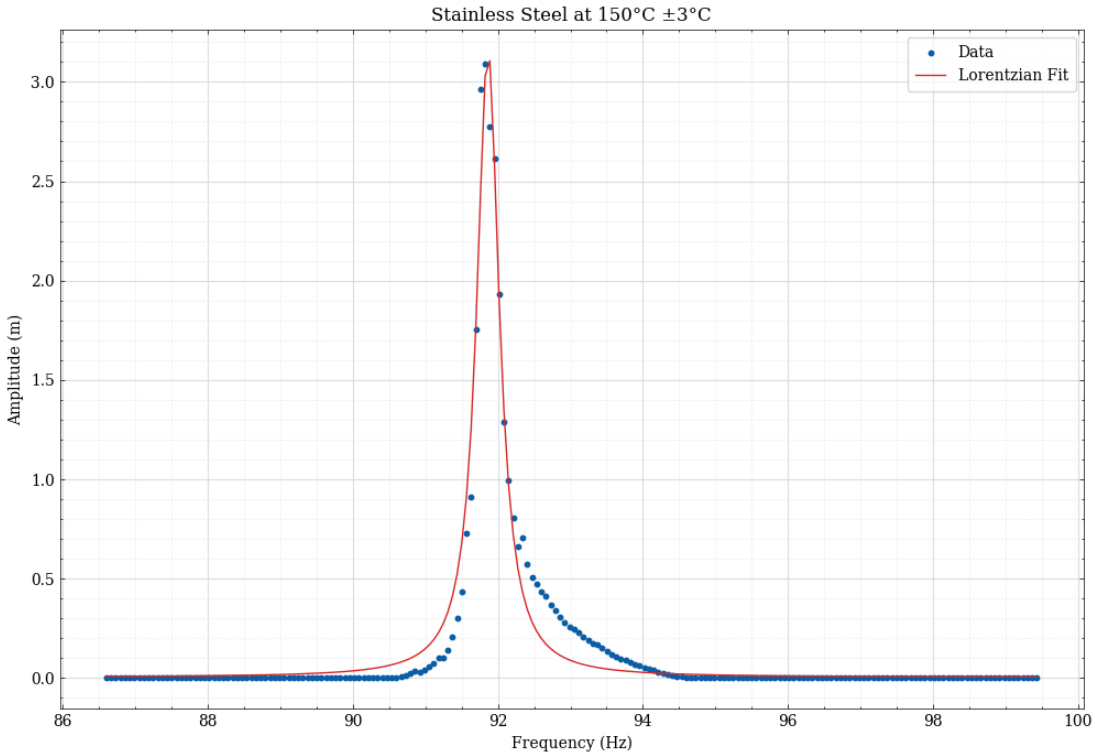
[101]:
```
initial_guess = [VSS120C[np.argmax(ampSS120C)], min(ampSS120C), (max(ampSS120C)␣
 ↪- min(ampSS120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS120C, ampSS120C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VSS120C, ampSS120C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VSS120C, lorentz_fit(VSS120C, *popt), color='tab:red',␣
 ↪label='Lorentzian Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel at 120°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```



```
Peak from fit: 92.279800
Width of peak: 0.378768
```

[102]:
```
SS150C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
 ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
 ↪labcode_s3/stainless steel 150C2')

VSS150C = np.array(SS150C[0]) * 64.15
ampSS150C = np.array(SS150C[1])
```

[103]:
```
initial_guess = [VSS150C[np.argmax(ampSS150C)], min(ampSS150C), (max(ampSS150C)␣
 ↪- min(ampSS150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS150C, ampSS150C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VSS150C, ampSS150C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VSS150C, lorentz_fit(VSS150C, *popt), color='tab:red',␣
 ↪label='Lorentzian Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel at 150°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

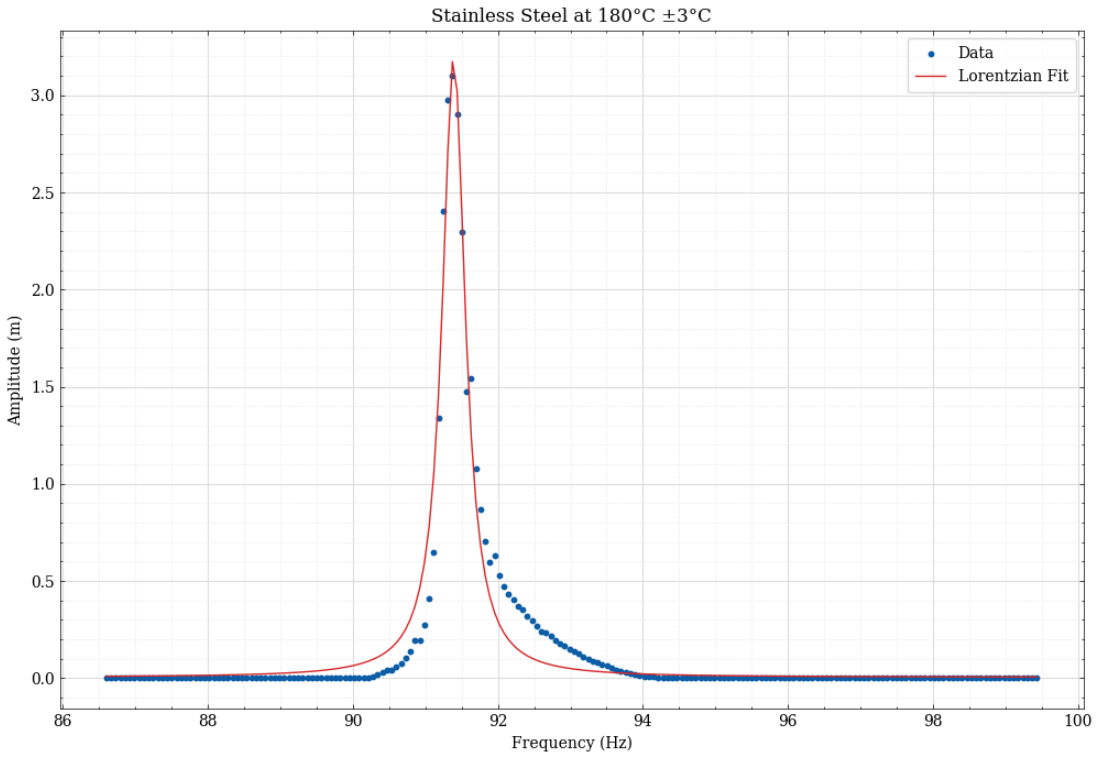



```
Peak from fit: 91.864033
Width of peak: 0.377263
```

[104]:
```
SS180C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/stainless steel 180C2')

VSS180C = np.array(SS180C[0]) * 64.15
ampSS180C = np.array(SS180C[1])
```

[105]:
```
initial_guess = [VSS180C[np.argmax(ampSS180C)], min(ampSS180C), (max(ampSS180C)␣
↪- min(ampSS180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS180C, ampSS180C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VSS180C, ampSS180C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VSS180C, lorentz_fit(VSS180C, *popt), color='tab:red',␣
↪label='Lorentzian Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel at 180°C ±3°C')
```

```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

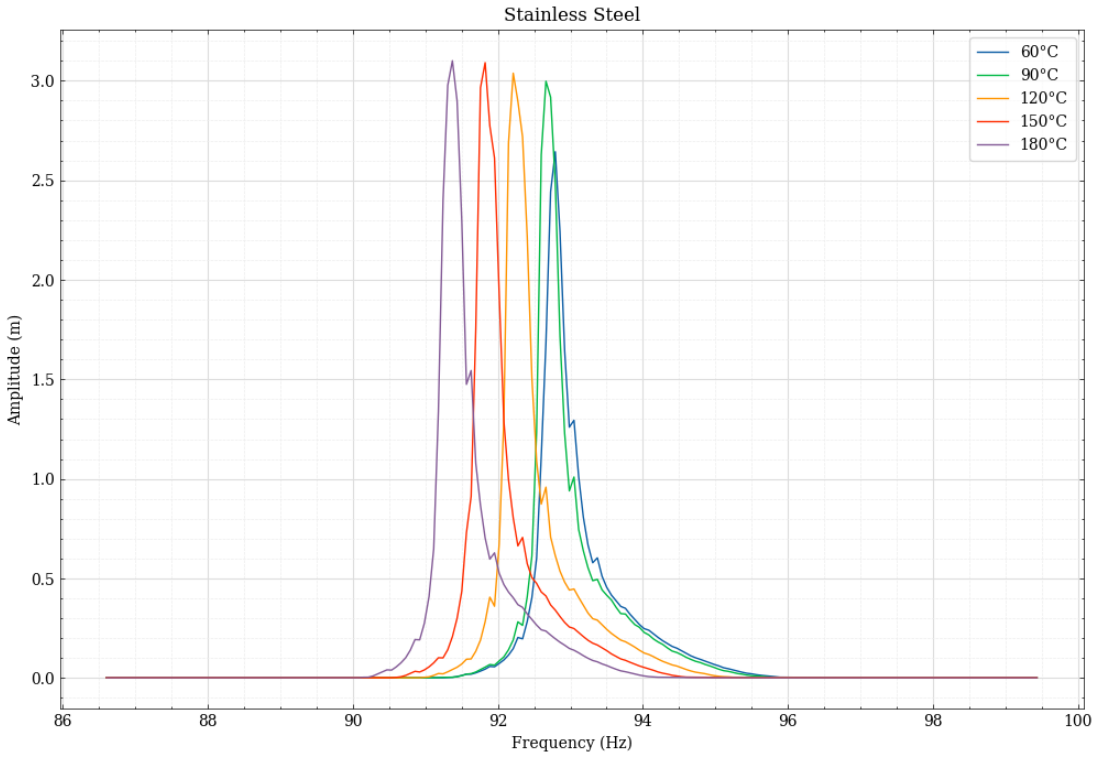



```
Peak from fit: 91.391520
Width of peak: 0.383930
```

2.2.1 ALL

[106]:
```
plt.plot(VSS60C, ampSS60C, label='60°C', zorder=3)
plt.plot(VSS90C, ampSS90C, label='90°C', zorder=4)
plt.plot(VSS120C, ampSS120C, label='120°C', zorder=5)
plt.plot(VSS150C, ampSS150C, label='150°C', zorder=6)
plt.plot(VSS180C, ampSS180C, label='180°C', zorder=7)
```

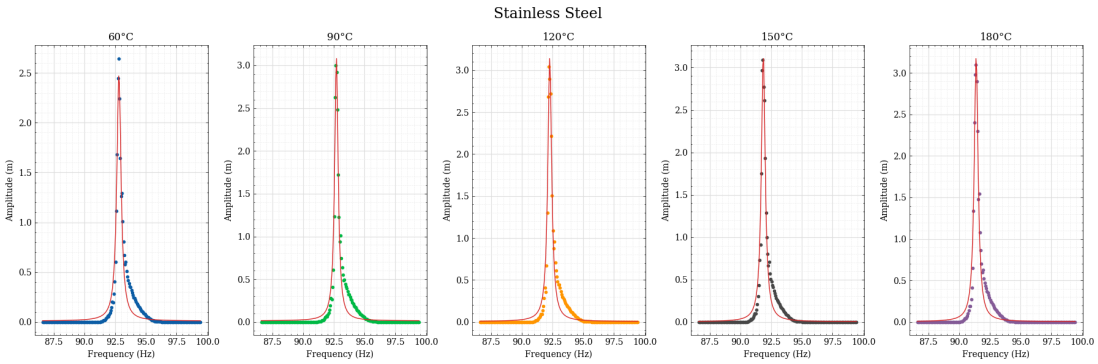
```
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Stainless Steel')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```



[107]:
```
fig, ax = plt.subplots(1,5, figsize=(18,6))
ax = ax.flatten()

initial_guess = [VSS60C[np.argmax(ampSS60C)], min(ampSS60C), (max(ampSS60C) -␣
↪min(ampSS60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS60C, ampSS60C, p0=initial_guess)
x0, omega, A, y0 = popt
```

```
ax[0].plot(VSS60C, ampSS60C, zorder=3, marker='.', linestyle='none',␣
 ↪label='60°C')
ax[0].plot(VSS60C, lorentz_fit(VSS60C, *popt), color='tab:red', zorder=4)
ax[0].minorticks_on()
ax[0].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[0].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Amplitude (m)')
ax[0].set_title('60°C')

initial_guess = [VSS90C[np.argmax(ampSS90C)], min(ampSS90C), (max(ampSS90C) -␣
 ↪min(ampSS90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS90C, ampSS90C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[1].plot(VSS90C, ampSS90C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#00B945', label='90°C')
ax[1].plot(VSS90C, lorentz_fit(VSS90C, *popt), color='tab:red', zorder=4)
ax[1].minorticks_on()
ax[1].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[1].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Amplitude (m)')
ax[1].set_title('90°C')

initial_guess = [VSS120C[np.argmax(ampSS120C)], min(ampSS120C), (max(ampSS120C)␣
 ↪- min(ampSS120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS120C, ampSS120C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[2].plot(VSS120C, ampSS120C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#FF9500', label='120°C')
ax[2].plot(VSS120C, lorentz_fit(VSS120C, *popt), color='tab:red', zorder=4)
ax[2].minorticks_on()
ax[2].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[2].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Amplitude (m)')
ax[2].set_title('120°C')

initial_guess = [VSS150C[np.argmax(ampSS150C)], min(ampSS150C), (max(ampSS150C)␣
 ↪- min(ampSS150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS150C, ampSS150C, p0=initial_guess)
```

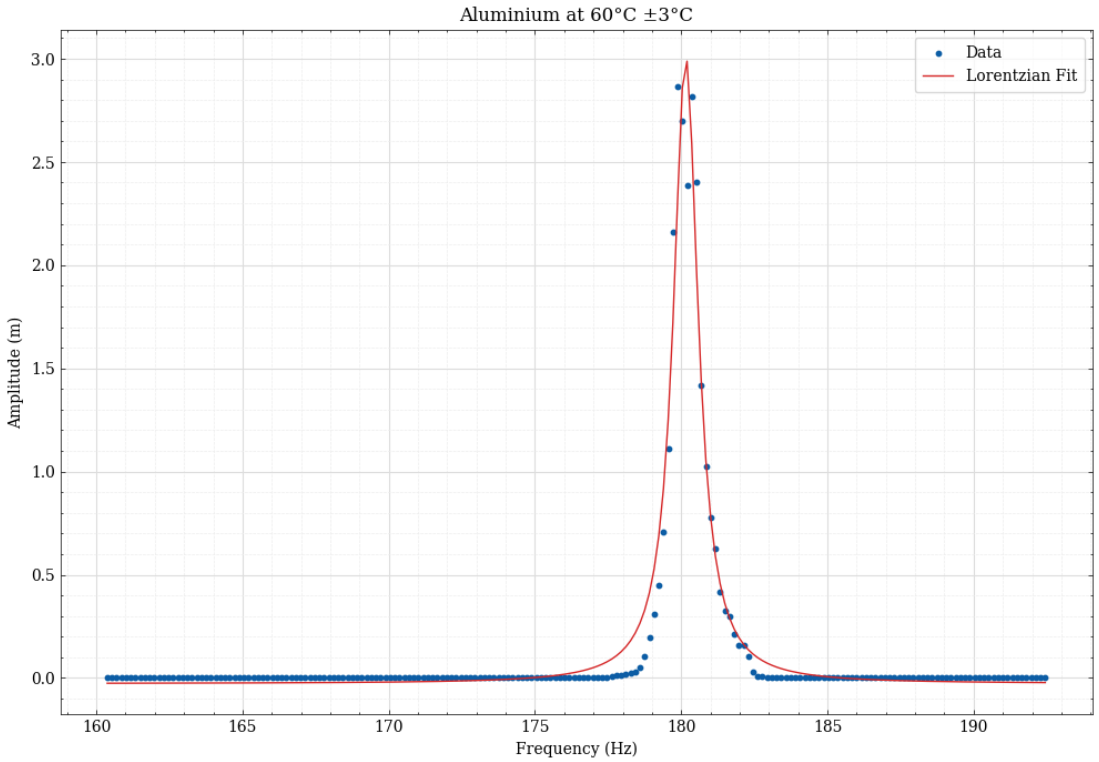
```
x0, omega, A, y0 = popt

ax[3].plot(VSS150C, ampSS150C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#474747', label='150°C')
ax[3].plot(VSS150C, lorentz_fit(VSS150C, *popt), color='tab:red', zorder=4)
ax[3].minorticks_on()
ax[3].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[3].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Amplitude (m)')
ax[3].set_title('150°C')

initial_guess = [VSS120C[np.argmax(ampSS180C)], min(ampSS180C), (max(ampSS180C)␣
 ↪- min(ampSS180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VSS180C, ampSS180C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[4].plot(VSS180C, ampSS180C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#845B97', label='180°C')
ax[4].plot(VSS180C, lorentz_fit(VSS180C, *popt), color='tab:red', zorder=4, ␣
 ↪label='Lorentzian Fit')
ax[4].minorticks_on()
ax[4].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[4].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Amplitude (m)')
ax[4].set_title('180°C')

fig.suptitle('Stainless Steel', size=17)
fig.tight_layout()

plt.show()
```



2.3 Aluminium

[108]:
```
A60C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
 ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
 ↪labcode_s3/aluminium 60C2')

VA60C = np.array(A60C[0]) * 64.15
ampA60C = np.array(A60C[1])
```

[109]:
```
initial_guess = [VA60C[np.argmax(ampA60C)], min(ampA60C), (max(ampA60C) -␣
 ↪min(ampA60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA60C, ampA60C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VA60C, ampA60C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VA60C, lorentz_fit(VA60C, *popt), color='tab:red', label='Lorentzian␣
 ↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium at 60°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

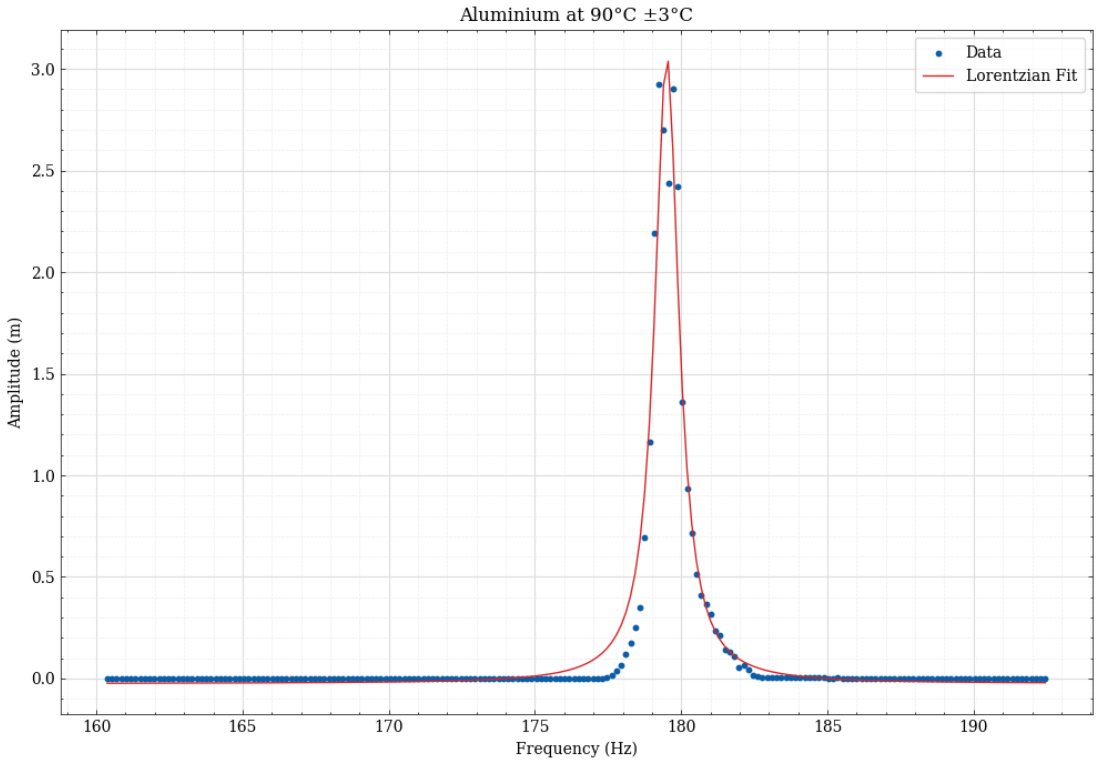



```
Peak from fit: 180.156115
Width of peak: 1.023388
```

[110]:
```
A90C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/aluminium 90C2')

VA90C = np.array(A90C[0]) * 64.15
ampA90C = np.array(A90C[1])
```

[111]:
```
initial_guess = [VA90C[np.argmax(ampA90C)], np.median(ampA90C), (np.
↪max(ampA90C) - np.min(ampA90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA90C, ampA90C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VA90C, ampA90C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VA90C, lorentz_fit(VA90C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium at 90°C ±3°C')
```

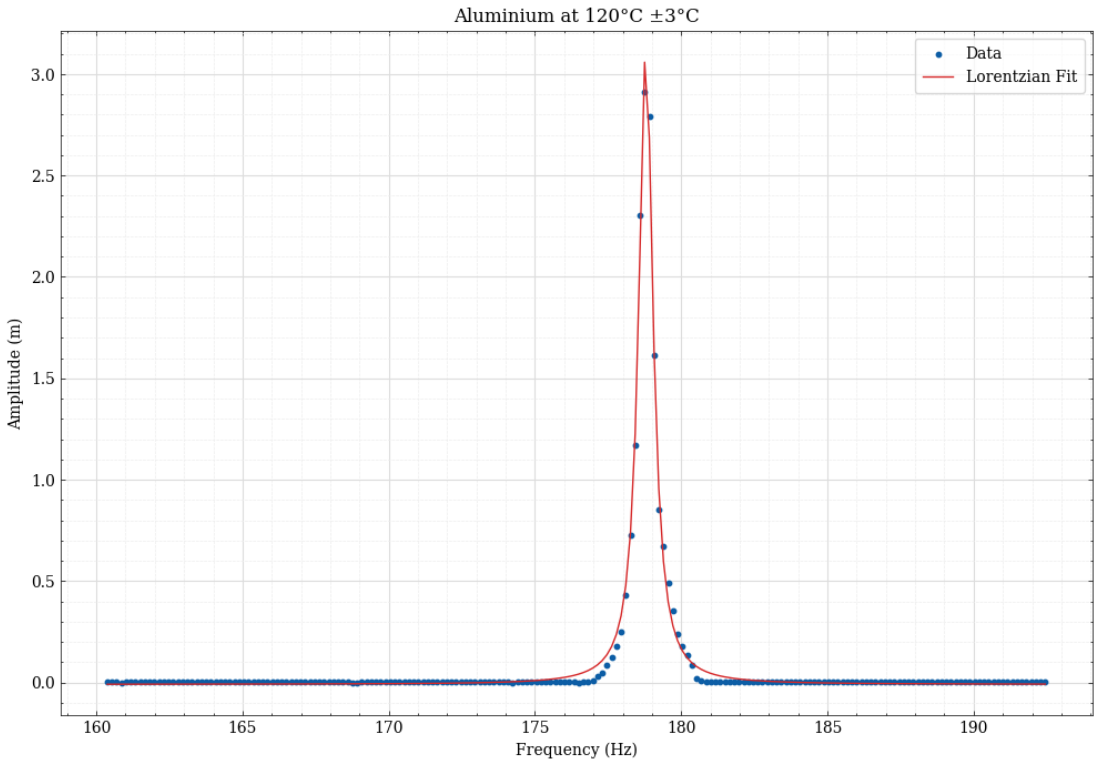
```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```




```
Peak from fit: 179.506483
Width of peak: 0.997994
```

[112]:
```
A120C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
 ↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
 ↪labcode_s3/aluminium 120C2')

VA120C = np.array(A120C[0]) * 64.15
ampA120C = np.array(A120C[1])
```

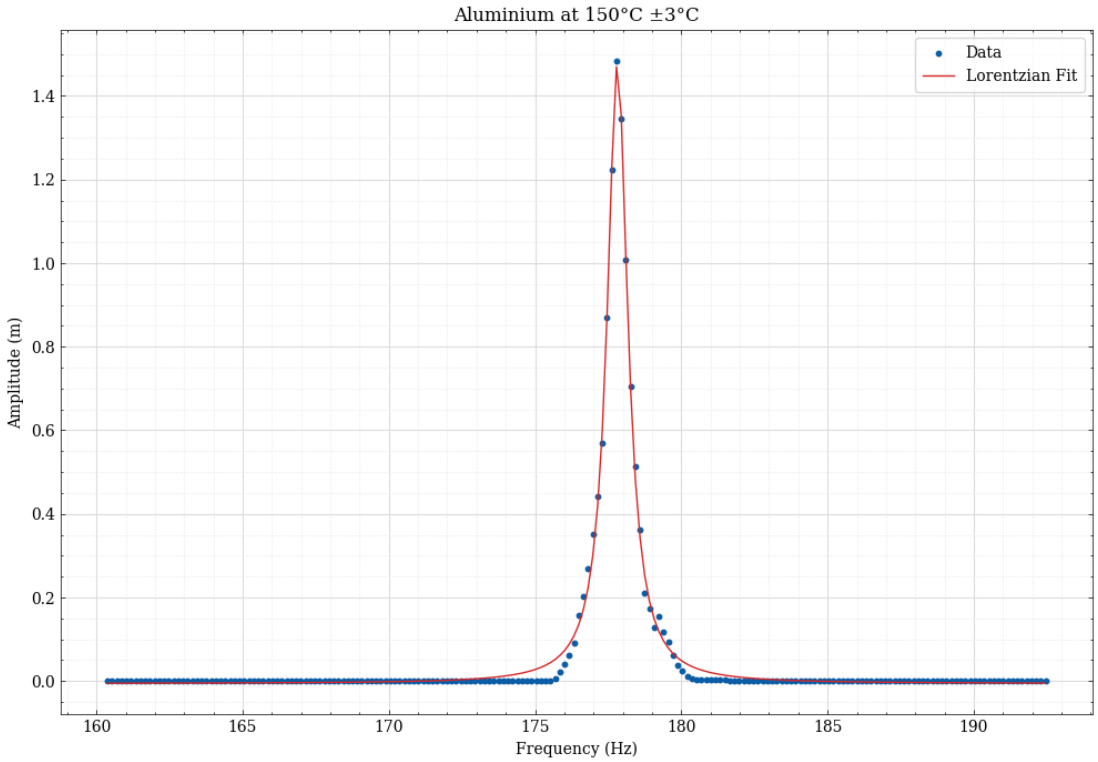
[113]:
```
initial_guess = [VA120C[np.argmax(ampA120C)], min(ampA120C), (max(ampA120C) -␣
↪min(ampA120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA120C, ampA120C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VA120C, ampA120C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VA120C, lorentz_fit(VA120C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium at 120°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```



```
Peak from fit: 178.792279
Width of peak: 0.589699
```

[114]:
```
A150C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/aluminium 150C2')

VA150C = np.array(A150C[0]) * 64.15
ampA150C = np.array(A150C[1])
```

[115]:
```
initial_guess = [VA150C[np.argmax(ampA150C)], min(ampA150C), (max(ampA150C) -␣
↪min(ampA150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA150C, ampA150C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VA150C, ampA150C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VA150C, lorentz_fit(VA150C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium at 150°C ±3°C')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

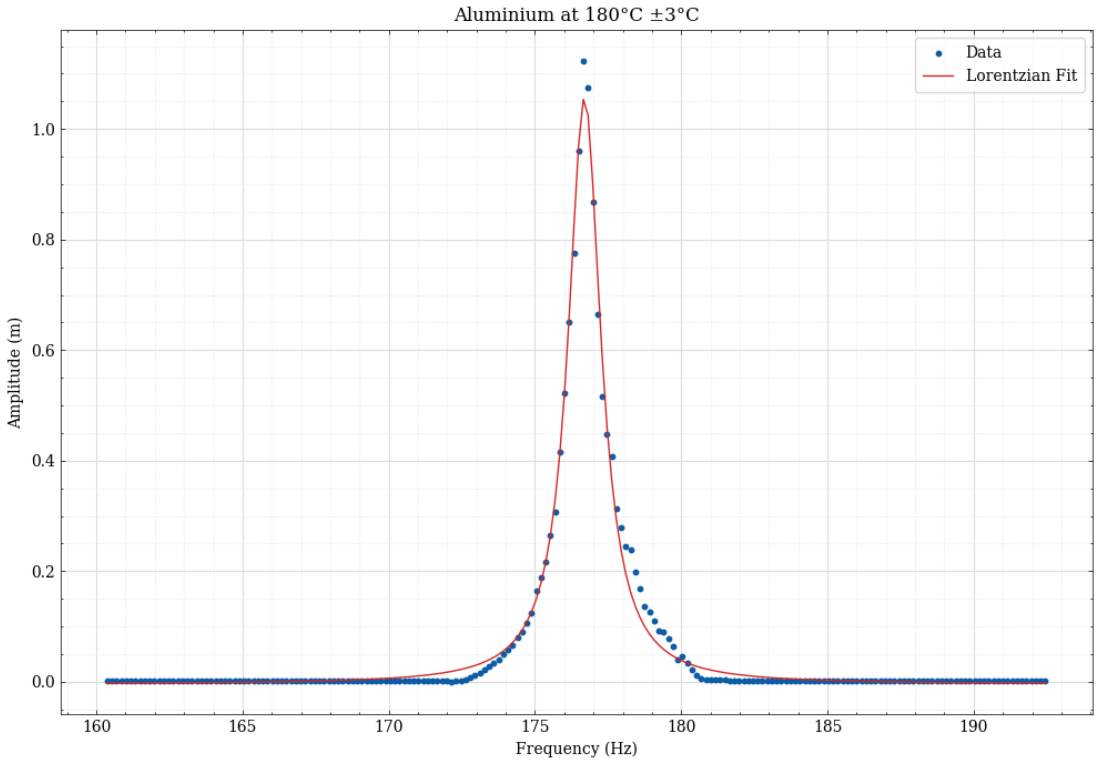



```
Peak from fit: 177.814923
Width of peak: 0.857605
```

[116]:
```
A180C = np.loadtxt('/Users/JoanaUCD/Library/CloudStorage/
↪OneDrive-UniversityCollegeDublin/Labs/labs-files/labs_code/Labs-Code/
↪labcode_s3/aluminium 180C2')

VA180C = np.array(A180C[0]) * 64.15
ampA180C = np.array(A180C[1])
```

[117]:
```
initial_guess = [VA180C[np.argmax(ampA180C)], min(ampA180C), (max(ampA180C) -␣
↪min(ampA180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA180C, ampA180C, p0=initial_guess)
x0, omega, A, y0 = popt

plt.scatter(VA180C, ampA180C, label='Data', zorder=3, marker='o', s=10)
plt.plot(VA180C, lorentz_fit(VA180C, *popt), color='tab:red', label='Lorentzian␣
↪Fit', zorder=4)

plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium at 180°C ±3°C')
```

```
plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()

print(f'Peak from fit: {x0:.6f}')
print(f'Width of peak: {abs(omega):.6f}')
```

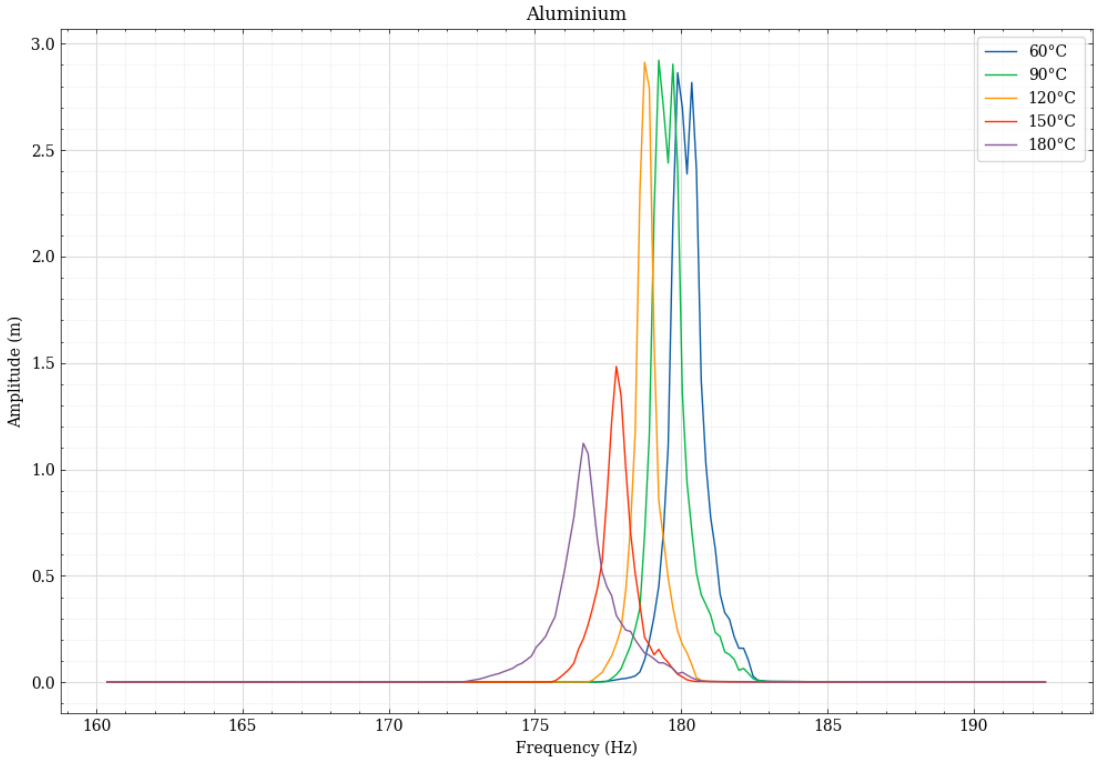



```
Peak from fit: 176.694473
Width of peak: 1.359736
```

2.3.1 ALL

[118]:
```
plt.plot(VA60C, ampA60C, label='60°C', zorder=3)
plt.plot(VA90C, ampA90C, label='90°C', zorder=4)
plt.plot(VA120C, ampA120C, label='120°C', zorder=5)
plt.plot(VA150C, ampA150C, label='150°C', zorder=6)
plt.plot(VA180C, ampA180C, label='180°C', zorder=7)
```

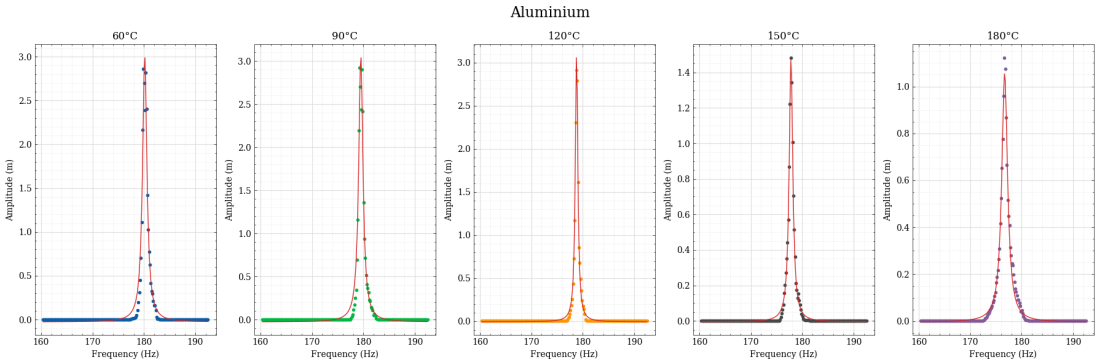
```
plt.xlabel('Frequency (Hz)')
plt.ylabel('Amplitude (m)')
plt.title('Aluminium')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```



[119]:
```
fig, ax = plt.subplots(1,5, figsize=(18,6))
ax = ax.flatten()

initial_guess = [VA60C[np.argmax(ampA60C)], min(ampA60C), (max(ampA60C) -␣
↪min(ampA60C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA60C, ampA60C, p0=initial_guess)
x0, omega, A, y0 = popt
```

```
ax[0].plot(VA60C, ampA60C, zorder=3, marker='.', linestyle='none', label='60°C')
ax[0].plot(VA60C, lorentz_fit(VA60C, *popt), color='tab:red', zorder=4)
ax[0].minorticks_on()
ax[0].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[0].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[0].set_xlabel('Frequency (Hz)')
ax[0].set_ylabel('Amplitude (m)')
ax[0].set_title('60°C')

initial_guess = [VA90C[np.argmax(ampA90C)], np.median(ampA90C), (max(ampA90C) -␣
 ↪min(ampA90C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA90C, ampA90C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[1].plot(VA90C, ampA90C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#00B945', label='90°C')
ax[1].plot(VA90C, lorentz_fit(VA90C, *popt), color='tab:red', zorder=4)
ax[1].minorticks_on()
ax[1].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[1].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[1].set_xlabel('Frequency (Hz)')
ax[1].set_ylabel('Amplitude (m)')
ax[1].set_title('90°C')

initial_guess = [VA120C[np.argmax(ampA120C)], min(ampA120C), (max(ampA120C) -␣
 ↪min(ampA120C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA120C, ampA120C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[2].plot(VA120C, ampA120C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#FF9500', label='120°C')
ax[2].plot(VA120C, lorentz_fit(VA120C, *popt), color='tab:red', zorder=4)
ax[2].minorticks_on()
ax[2].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[2].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[2].set_xlabel('Frequency (Hz)')
ax[2].set_ylabel('Amplitude (m)')
ax[2].set_title('120°C')

initial_guess = [VA150C[np.argmax(ampA150C)], min(ampA150C), (max(ampA150C) -␣
 ↪min(ampA150C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA150C, ampA150C, p0=initial_guess)
x0, omega, A, y0 = popt
```

```
ax[3].plot(VA150C, ampA150C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#474747', label='150°C')
ax[3].plot(VA150C, lorentz_fit(VA150C, *popt), color='tab:red', zorder=4)
ax[3].minorticks_on()
ax[3].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[3].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[3].set_xlabel('Frequency (Hz)')
ax[3].set_ylabel('Amplitude (m)')
ax[3].set_title('150°C')

initial_guess = [VA120C[np.argmax(ampA180C)], min(ampA180C), (max(ampA180C) -␣
 ↪min(ampA180C))*0.5, 0.5]
popt, pcov = curve_fit(lorentz_fit, VA180C, ampA180C, p0=initial_guess)
x0, omega, A, y0 = popt

ax[4].plot(VA180C, ampA180C, zorder=3, marker='.', linestyle='none',␣
 ↪color='#845B97', label='180°C')
ax[4].plot(VA180C, lorentz_fit(VA180C, *popt), color='tab:red', zorder=4, ␣
 ↪label='Lorentzian Fit')
ax[4].minorticks_on()
ax[4].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[4].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[4].set_xlabel('Frequency (Hz)')
ax[4].set_ylabel('Amplitude (m)')
ax[4].set_title('180°C')

fig.suptitle('Aluminium', size=17)
fig.tight_layout()

plt.show()
```



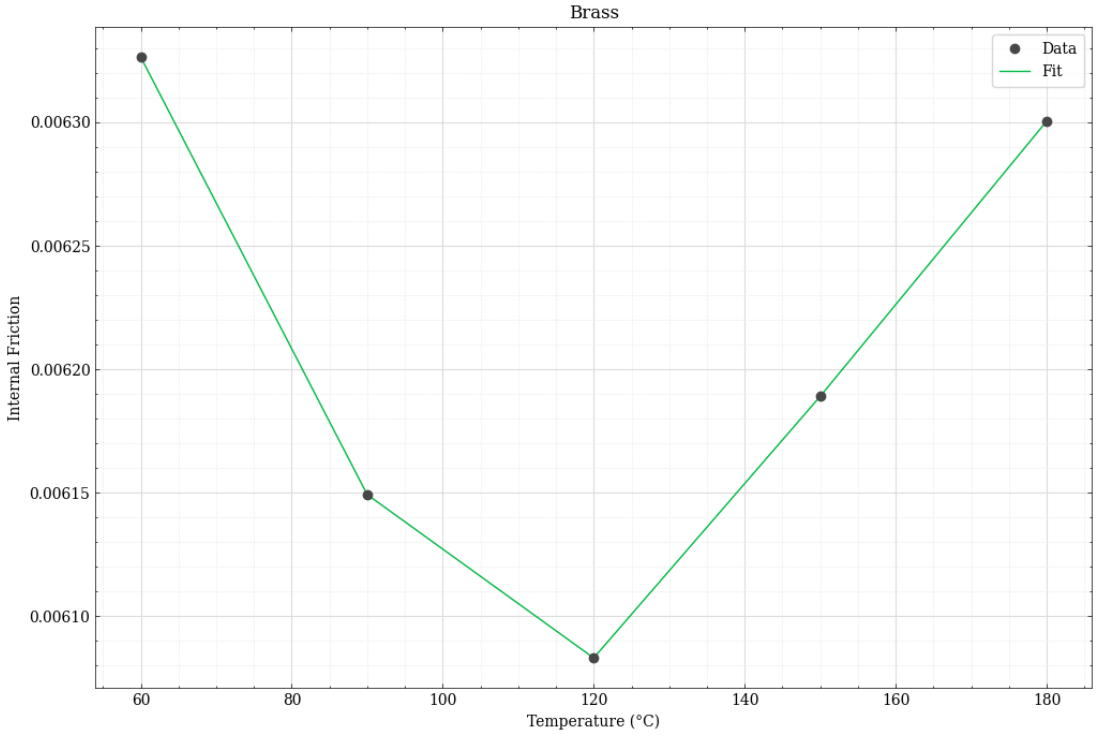
3 Internal Friction

[120]:
```
# eq.6
def int_friction(domega, omega0):
    return domega / (np.sqrt(3) * omega0)

# domega = width of curve at half max A
# omega0 = angular frequency
```

[121]:
```
# brass

fres = np.array([195.558052, 195.075226, 194.552084, 194.069886, 193.373124])
omega0 = 2 * np.pi * fres
omega = np.array([2.142843, 2.077667, 2.049796, 2.080339, 2.110216])
domega = 2 * np.pi * omega

BIF = int_friction(domega, omega0)

print(BIF)
```

```
[0.00632636 0.00614912 0.00608295 0.00618893 0.00630043]
```

[122]:
```
plt.plot(T, BIF, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, BIF, label='Fit', color='#00B945')

plt.xlabel('Temperature (°C)')
plt.ylabel('Internal Friction')
plt.title('Brass')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```

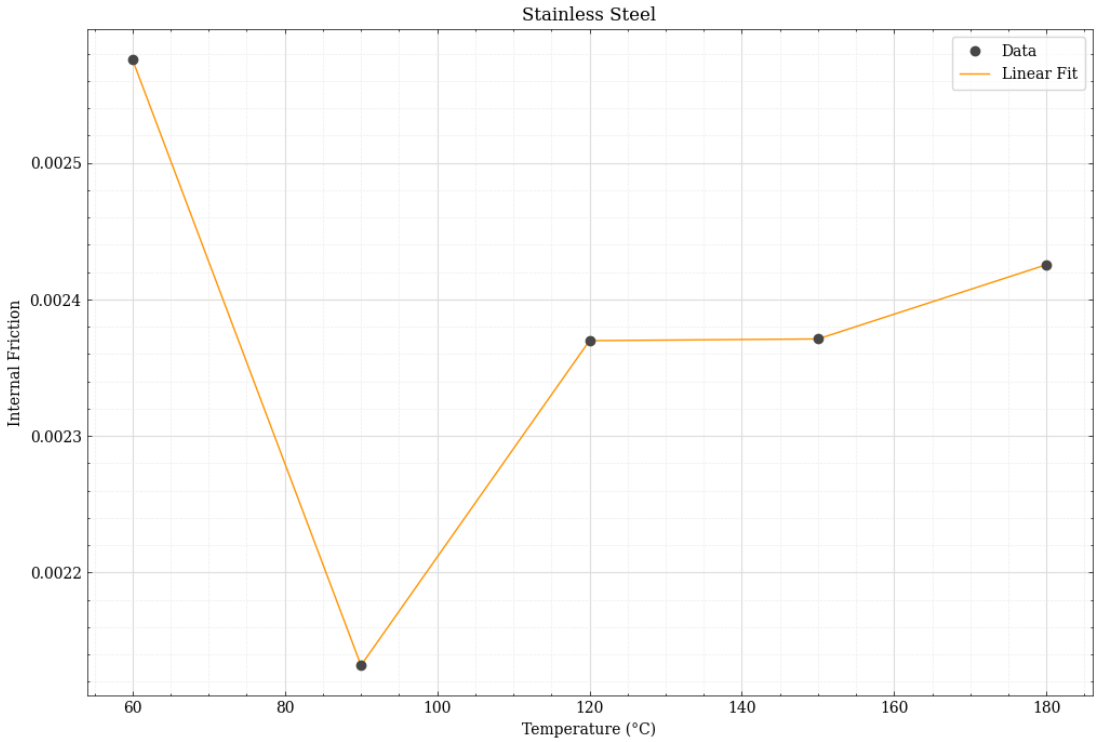


[123]:
```
# steel

fres = np.array([92.813146, 92.714482, 92.279800, 91.864033, 91.391520])
omega0 = 2 * np.pi * fres
omega = np.array([0.414041, 0.342378, 0.378768, 0.377263, 0.383930])
domega = 2 * np.pi * omega

SSIF = int_friction(domega, omega0)

print(SSIF)
```

```
[0.00257557 0.00213205 0.00236977 0.00237104 0.00242541]
```

[124]:
```
plt.plot(T, SSIF, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, SSIF, label='Linear Fit', color='#FF9500')

plt.xlabel('Temperature (°C)')
plt.ylabel('Internal Friction')
plt.title('Stainless Steel')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
```

```
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```

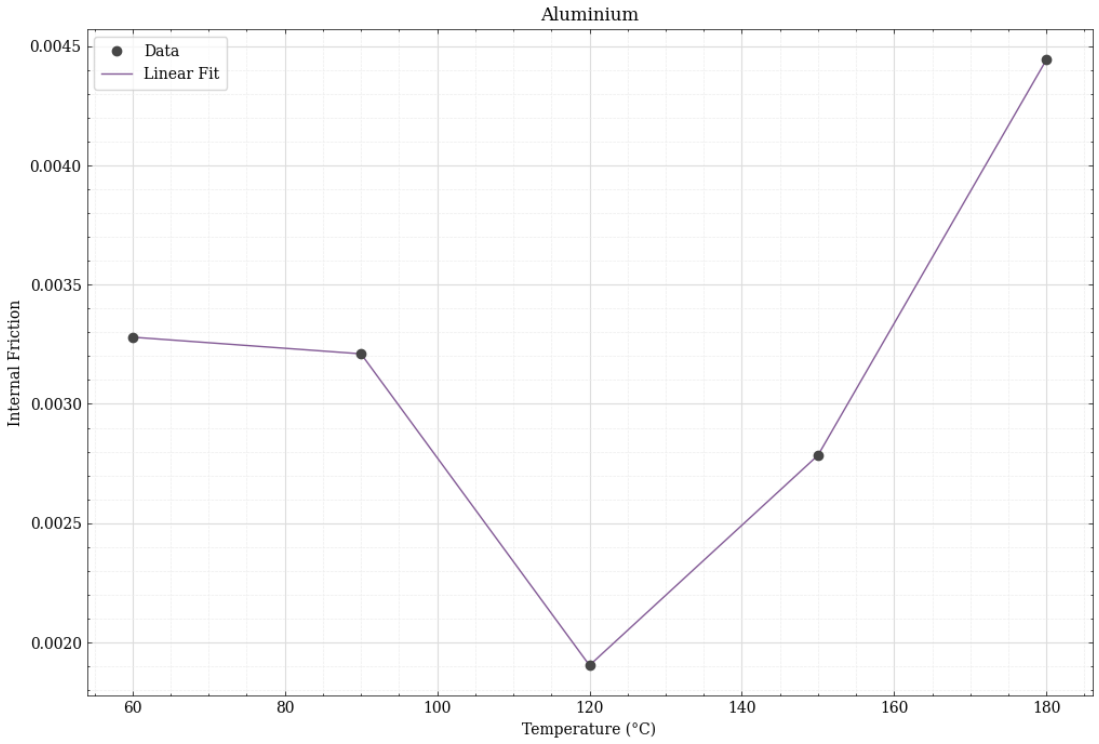


[125]:
```
# aluminium

fres = np.array([180.156115, 179.506483, 178.792279, 177.814923, 176.694473])
omega0 = 2 * np.pi * fres
omega = np.array([1.023388, 0.997994, 0.589699, 0.857605, 1.359736])
domega = 2 * np.pi * omega

AIF = int_friction(domega, omega0)

print(AIF)
```

```
[0.00327967 0.00320987 0.00190424 0.00278457 0.00444295]
```

[126]:
```
plt.plot(T, AIF, 'o', label='Data', zorder=3, color='#474747')
plt.plot(T, AIF, label='Linear Fit', color='#845B97')
```

```
plt.xlabel('Temperature (°C)')
plt.ylabel('Internal Friction')
plt.title('Aluminium')

plt.minorticks_on()
plt.grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
plt.grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)

plt.legend(frameon=True, fancybox=True, framealpha=0.75, borderpad=0.5)

plt.show()
```



[127]:
```
fig, ax = plt.subplots(3, 1)

ax[0].plot(T, BIF, 'o', label='Data', zorder=3, color='#474747')
ax[0].plot(T, BIF, label='Linear Fit', color='#00B945')
ax[0].minorticks_on()
ax[0].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[0].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[0].set_ylabel('Internal Friction')
```

```
ax[0].set_xlabel('Temperature (°C)')
ax[0].set_title('Brass')

ax[1].plot(T, SSIF, 'o', label='Data', zorder=3, color='#474747')
ax[1].plot(T, SSIF, label='Linear Fit', color='#FF9500')
ax[1].minorticks_on()
ax[1].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[1].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[1].set_ylabel('Internal Friction')
ax[1].set_xlabel('Temperature (°C)')
ax[1].set_title('Stainless Steel')

ax[2].plot(T, AIF, 'o', label='Data', zorder=3, color='#474747')
ax[2].plot(T, AIF, label='Linear Fit', color='#845B97')
ax[2].minorticks_on()
ax[2].grid(True, which="major", linewidth=0.8, color="#DDDDDD", zorder=2)
ax[2].grid(True, which="minor", linewidth=0.5, color="#EEEEEE", linestyle="--",␣
 ↪zorder=1)
ax[2].set_ylabel('Internal Friction')
ax[2].set_xlabel('Temperature (°C)')
ax[2].set_title('Aluminium')

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