Harmonic Analysis

Physics Laboratory 3

Visva Loganathan vloganathan@student.ethz.ch

Teaching Assistant

Takahiro Uto takuto@student.ethz.ch

Abstract In this experiment, harmonic analysis of periodic signals was investigated using both frequency-selective measurements and Fourier-based spectral analysis. The properties of a precision rectifier and a band-pass filter were characterized, and the filter response was fitted with a Lorentzian model, yielding a resonance frequency of $f_0 = (95.0 \pm 0.8)\,\mathrm{kHz}$ and a quality factor of $Q = 13 \pm 5$. Amplitude modulation was analyzed using three methods. The modulation depth was determined as $m = 0.611 \pm 0.005$ from the X–Y plot, $m = 0.727 \pm 0.002$ from the time-domain oscilloscope trace, and $m = 0.0092 \pm 0.0004$ from the FFT spectrum. The first two methods gave values of the same order of magnitude, whereas the FFT-based result was significantly smaller. In addition, Fourier spectra of sinusoidal and triangular signals were recorded with a digital oscilloscope. Both signals showed a fundamental frequency near 34.5 kHz, while the triangular waveform exhibited a more pronounced odd-harmonic structure. Finally, the total harmonic distortion of a distorted sinusoidal signal was found to be THD $= (1.11 \pm 0.02)\%$. Overall, the measurements were largely consistent with the theoretical expectations, except for the FFT-based determination of the modulation depth.

ETH Zürich, April 8, 2026

Visva Loganathan

Introduction

The analysis of signals in both the time and frequency domains is a fundamental tool in modern physics and electrical engineering. While signals are commonly observed as functions of time, their representation in the frequency domain often provides deeper insight into their structure, transmission properties, and interaction with linear systems. The mathematical foundation of this approach is given by Fourier analysis, which allows periodic signals to be decomposed into harmonic components and arbitrary signals to be represented by their frequency spectra [2].

In communication systems, frequency-domain analysis plays a crucial role. For efficient transmission of information via electromagnetic waves, low-frequency signals are typically shifted to higher frequency ranges. This process, known as modulation, enables wireless transmission, improves signal-to-noise ratios, and allows multiple signals to share a common transmission medium. In particular, amplitude modulation (AM) modifies the amplitude of a high-frequency carrier wave according to a lower-frequency information signal. The resulting spectrum contains the carrier frequency as well as sidebands that encode the transmitted information.

The present experiment focuses on harmonic analysis and amplitude modulation. The primary objective is to investigate the frequency spectra of various periodic signals and to compare different methods of spectral analysis. A frequency-mixing measurement technique using a modulator and a band-pass filter is employed to resolve individual frequency components. The results are compared with Fast Fourier Transform (FFT) measurements obtained from a digital oscilloscope and with numerical Fourier analysis of digitally recorded data.

Furthermore, the transfer functions of linear systems such as low-pass, high-pass, and band-pass filters are examined. Their influence on signal spectra is analyzed in terms of amplitude response, resonance behavior, and quality factor. Special attention is given to balanced amplitude modulation and the determination of the degree of modulation. Finally, the total harmonic distortion (THD) is measured as a quantitative indicator of signal purity.

This experiment, therefore, combines theoretical concepts of Fourier analysis and the frequency response of linear systems with practical measurement techniques used in signal processing and telecommunications.

Experiment

The experimental arrangement is designed to investigate amplitude modulation and to perform harmonic (frequency) analysis of periodic signals. A schematic overview of the setup is shown in the laboratory manual. The system consists of a signal source, a modulator, a band-pass filter, a rectifier, measurement instruments, and a computer for data acquisition.

Signal Sources

Two function generators are used. One generator provides the high-frequency carrier signal f_C, while the second generator produces the low-frequency modulation signal f_M. Depending on the measurement task, different waveforms (sinusoidal, rectangular, triangular, or pulse signals) can be selected.
For frequency-selective measurements, the carrier frequency can be varied linearly in time by means of a ramp generator. The ramp voltage controls a voltage-controlled oscillator (VCO), allowing a continuous scan of the frequency range of interest.

Modulator

The modulation is performed using an analogue multiplier. The device multiplies the carrier signal with the modulation signal, resulting in an amplitude-modulated output signal. The modulation depth can be adjusted via offset voltages, which allow operation in standard amplitude modulation or in balanced modulation (carrier suppression).
The amplification factor of the modulator is approximately constant within its linear operating range. Care must be taken to avoid overmodulation or saturation effects.

Band-Pass Filter

After modulation, the signal is fed into a third-order band-pass filter with a fixed resonance frequency of approximately 90 kHz. The filter exhibits a high quality factor ($Q \approx 100$), providing strong frequency selectivity.
By sweeping the carrier frequency, individual spectral components of the modulation signal are shifted into the resonance range of the filter. Only frequency components within the narrow bandwidth of the filter are transmitted with significant amplitude. This enables selective measurement of single harmonics.

Rectifier and Amplitude Measurement

The filtered signal is passed through a precision rectifier. The rectifier converts the alternating signal into a proportional DC voltage, allowing accurate amplitude measurements.

Depending on the selected amplification factor, small signals can be measured with increased sensitivity. The output voltage of the rectifier is recorded either with multimeters or displayed on the oscilloscope.

Oscilloscope and Data Acquisition

Both an analogue and a digital oscilloscope are used. The analogue oscilloscope enables direct visualization of time-domain signals and x-y plots (e.g. for the modulation trapezoid). The digital oscilloscope allows Fast Fourier Transform (FFT) analysis and digital storage of measured data.
The recorded data can be transferred to a computer for numerical Fourier analysis and comparison with theoretical Fourier coefficients.

Measurement Strategy

The experiment proceeds by first characterizing the individual components (rectifier and filter), followed by measurements of amplitude-modulated signals. Subsequently, Fourier spectra of various periodic signals are recorded using three different methods:

- frequency-selective measurement with the band-pass filter,
- FFT analysis using the digital oscilloscope,
- numerical Fourier transformation of stored signal data.

Finally, the total harmonic distortion (THD) of selected signals is determined from their measured spectra.

Results

Characteristics of the Rectifier (Raw Data)

Table 1: Rectifier raw data for the K_1 and K_{10} settings.

K_1 setting	
V_{in} (V)	V_{out} (V)
4.71	4.11
5.03	4.40
7.43	6.43
9.78	8.53
12.10	10.60
14.70	12.80
18.30	16.00
21.40	18.80
23.40	20.30
26.00	22.60
30.20	26.80
37.40	33.60
42.20	37.90
46.80	42.20
50.40	45.40
55.50	49.60
61.20	54.90
65.30	58.70
70.50	63.40
74.50	67.00

K_{10} setting	
V_{in} (V)	V_{out} (V)
4.71	41.2
5.07	44.5
7.41	64.5
9.94	86.5
12.10	105
14.60	128
15.30	135
15.80	139
16.00	141
16.40	141
18.50	141
21.30	141
23.30	141
30.20	141
40.20	141
50.30	141
74.50	141

Characteristics of the Filter (Raw Data)

Table 2: Raw measurement data used for the characterization of the band-pass filter.

(a) Band-pass filter raw data for linearity test (measured close to resonance).

V_{in} (V)	V_{out} (V)
4.67	2.28
5.05	2.51
7.56	3.60
10.00	4.75
15.10	7.26
19.90	9.51
25.10	12.00
30.00	14.50
35.10	16.90
39.90	19.20
45.00	21.60
50.20	23.90
55.10	26.20
60.10	28.70
65.10	31.00
68.90	32.80
73.90	35.10

(b) Band-pass filter resonance curve raw data ($V_{\text{in}} = 3.74\,\text{V}$ constant).

f (kHz)	V_{out} (V)
59.8	0.511
70.5	0.522
80.6	0.526
85.1	0.527
88.9	0.532
91.8	0.545
93.8	0.552
95.9	0.563
98.2	0.542
100.0	0.530
105.1	0.523
110.2	0.522

Balanced Modulation (Raw Data)

Two oscilloscope recordings of the amplitude-modulated signal were used for the later determination of the modulation depth: an X–Y plot of the signal and a conventional time-domain oscilloscope trace (KO display). In both cases, the relevant vertical heights were measured directly from the screenshots using the *Screen Ruler* utility from Microsoft PowerToys at a zoom level of 400%.

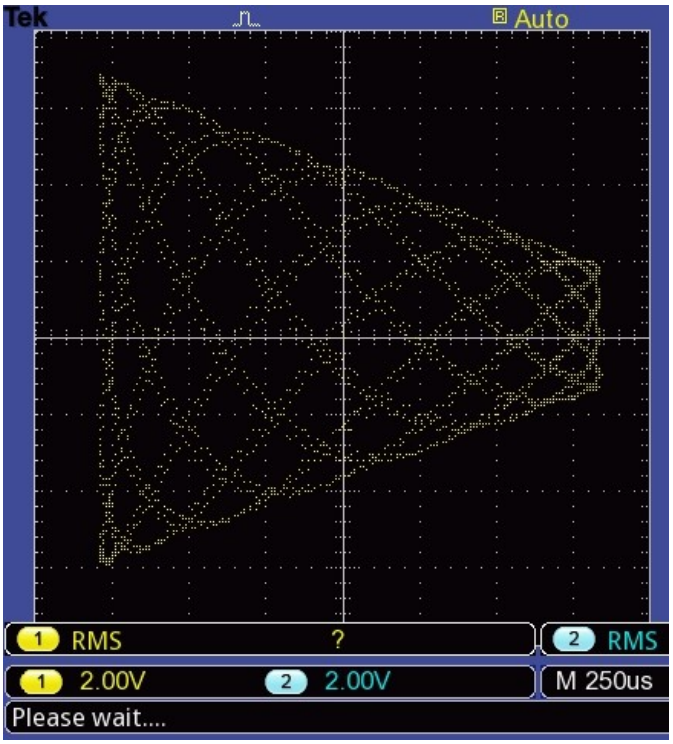

Figure 1: X–Y plot of the amplitude-modulated signal showing the characteristic trapezoidal pattern. The horizontal axis corresponds to the modulating signal, while the vertical axis corresponds to the AM signal. The maximum and minimum vertical extensions of the trapezoid were used for the determination of the modulation depth.

For the X–Y method, the measured vertical heights of the trapezoid were

$$H_{\max} = 1315 \text{ px}, \qquad H_{\min} = 318 \text{ px}.$$

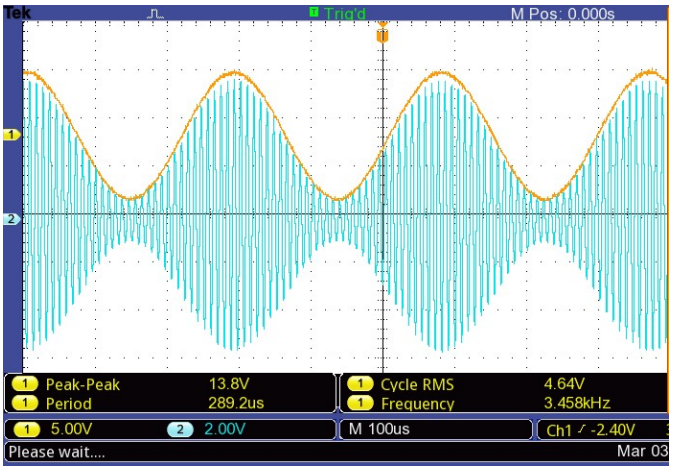

Figure 2: Time-domain oscilloscope trace of the amplitude-modulated signal (KO display). The maximum and minimum vertical signal heights, $H_{\max}$ and $H_{\min}$, were determined from the envelope and used for the time-domain evaluation of the modulation depth.

For the time-domain method, three repeated measurements were taken from the envelope. The measured values were

$$H_{\max} = (1135,\ 1136,\ 1130)\ \text{px}, \qquad H_{\min} = (179,\ 177,\ 182)\ \text{px}.$$

Finding Fourier Spectra (Extracted Peak Values)

Sinusoidal signal (Top peaks from FFT export)

Table 3: Extracted dominant spectral peaks from `SinusoidalFFT.CSV`.

Frequency (Hz)	Amplitude (dB)
34 524.94	−6.59
103 574.81	−11.79
89 271.62	−21.39
92 724.12	−34.99
95 683.40	−36.59
84 339.49	−40.99
86 805.55	−42.59
82 859.85	−42.99
78 914.14	−44.99
73 982.01	−45.79

Triangular signal (Top peaks from FFT export)

Table 4: Extracted dominant spectral peaks from `TriangularFFT.CSV`.

Frequency (Hz)	Amplitude (dB)
34 524.94	16.21
103 574.81	−3.39
310 231.22	−22.19
448 330.96	−28.59
492 720.17	−29.79
423 177.08	−30.99
354 620.42	−32.59
69 049.87	−32.99
138 099.75	−32.99
78 420.93	−36.19

Total Harmonic Distortion (Raw Peak Values)

The amplitudes of the dominant harmonic components were obtained from the FFT spectrum measured with the oscilloscope. These peak values serve as the raw data for the later evaluation of the total harmonic distortion (THD).

Table 5: Measured amplitudes of the first harmonic components extracted from the FFT spectrum and used for the THD calculation.

Harmonic index n	Amplitude (dB)
1	28.2
2	−12.5
3	−16.1

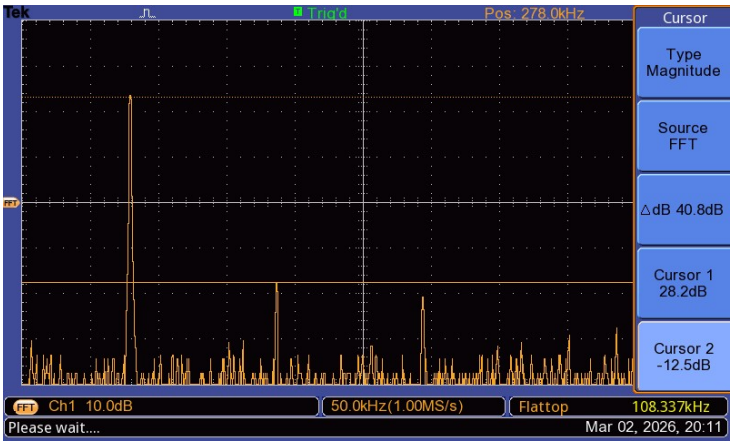

Figure 3: FFT spectrum of the measured signal showing the fundamental component and higher harmonic peaks used for the determination of the total harmonic distortion.

.

Data Analysis

Characteristics of the Rectifier

The transmission characteristics of the precision rectifier were measured for both gain settings K_1 and K_{10}. In each case, a linear model constrained through the origin was used:

$$V_{\text{out}} = K_i \, V_{\text{in}}. \tag{1}$$

For the K_1 setting, using all measured points, the fit yielded

$$K_1 = 0.8962 \pm 0.0018 \tag{2}$$

For the K_{10} setting, linear behavior was observed only up to $V_{\text{in}} \leq 16.0$ V. Restricting the fit to this region gave

$$K_{10} = 8.774 \pm 0.018. \tag{3}$$

Above $V_{\text{in}} \approx 16$ V, the output voltage saturated at approximately $V_{\text{out}} \approx 141$ V, indicating the limit of the linear operating range.

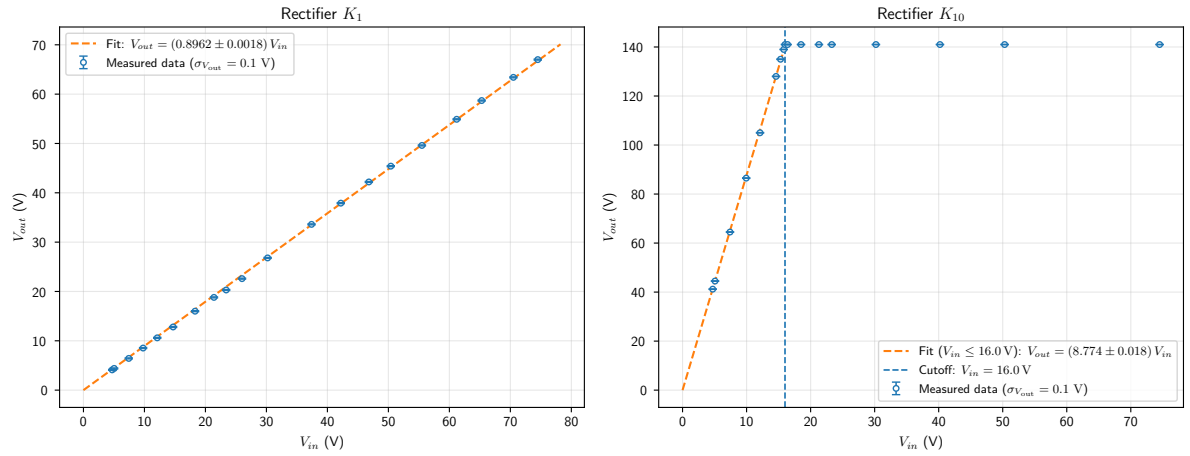

Figure 4: Measured transfer characteristics of the rectifier for the K_1 and K_{10} settings. The dashed lines indicate the linear fits to the data. For the K_{10} configuration the output voltage saturates above $V_{\text{in}} \approx 16\,\text{V}$.

Characteristics of the Filter (Linearity)

The linear operating range of the band-pass filter was investigated by measuring V_{out} as a function of V_{in} close to resonance. A linear regression yielded

$$V_{\text{out}} = (0.4752 \pm 0.0009)\, V_{\text{in}} + (0.10 \pm 0.04)\ \text{V}, \tag{4}$$

with a coefficient of determination

$$R^2 = 0.99994. \tag{5}$$

confirming highly linear behavior in the investigated amplitude range.

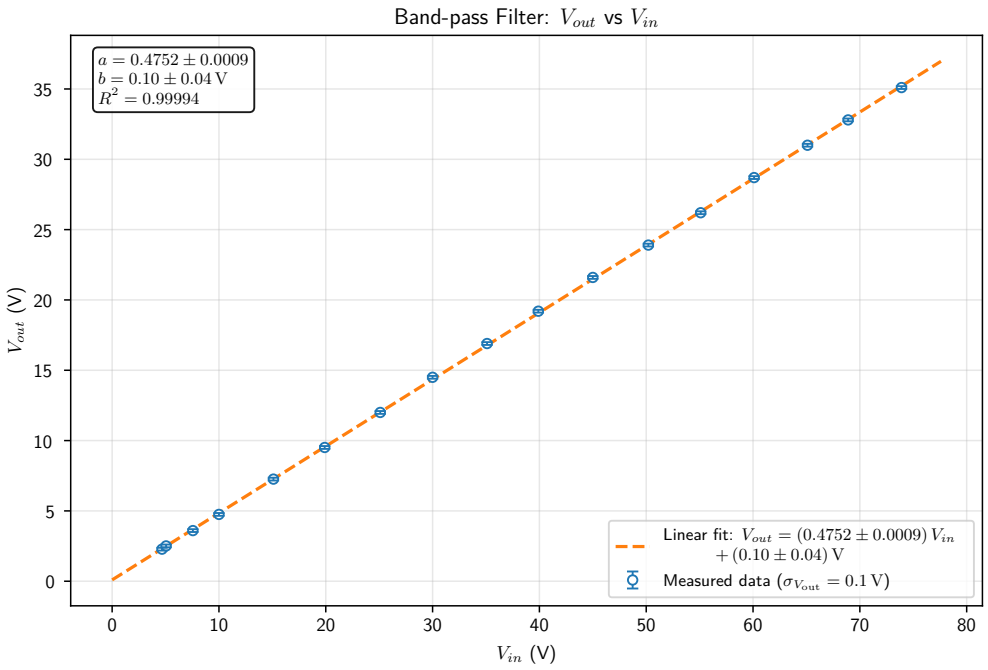

Figure 5: Measured output voltage V_{out} as a function of the input voltage V_{in} for the band-pass filter at resonance. The dashed line represents the linear regression used to determine the filter gain.

In addition, the gain

$$G = \frac{V_{\text{out}}}{V_{\text{in}}} \tag{6}$$

was evaluated as a function of V_{in}, showing an approximately constant value around $G = 0.477 \pm 0.001$, within measurement uncertainties.

Characteristics of the Filter (Resonance Curve)

The frequency response of the band-pass filter was investigated for a constant input voltage $V_{\text{in}} = 3.74$ V. The gain was defined as

$$G(f) = \frac{V_{\text{out}}(f)}{V_{\text{in}}}. \tag{7}$$

Before analyzing the resonance behavior, the ratio $V_{\text{out}}/V_{\text{in}}$ was examined as a function of the input voltage. Figure 6 shows that the gain remains approximately constant within the measurement uncertainties over the investigated voltage range. This indicates that the filter operates linearly under the chosen excitation conditions.

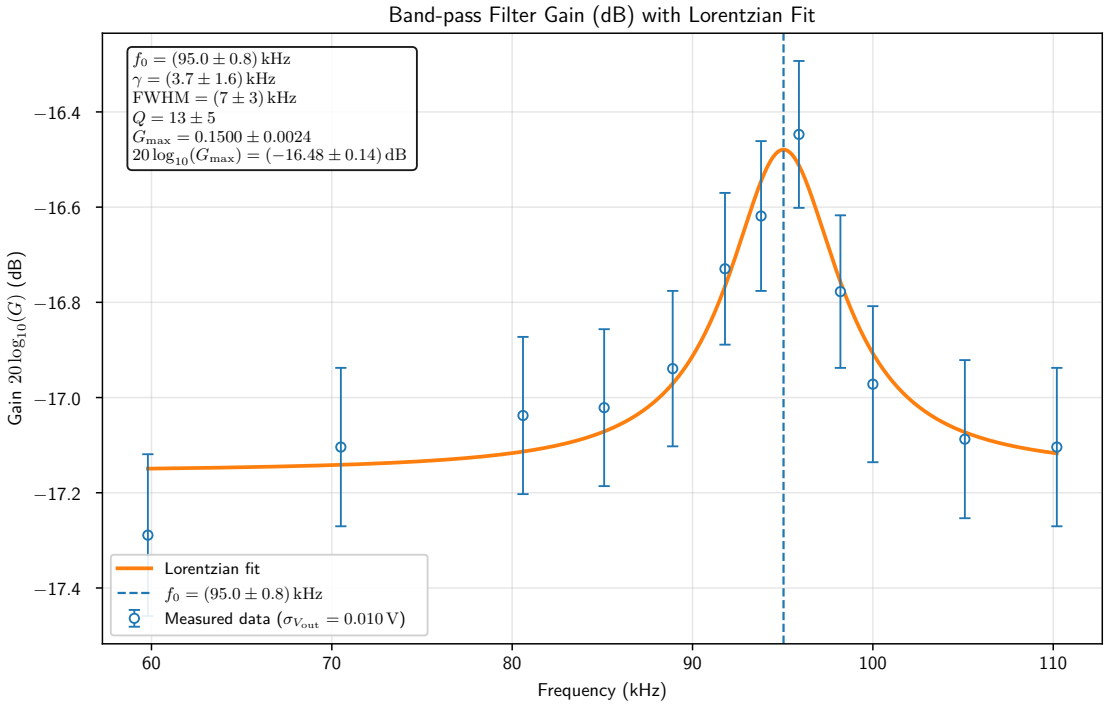

Figure 6: Measured gain $20\log_{10}(G)$ of the band-pass filter as a function of the excitation frequency f. The data points represent the measured values with their propagated uncertainties. The solid curve shows the Lorentzian fit used to determine the resonance frequency f_0 and the bandwidth of the filter. The dashed vertical line indicates the fitted resonance frequency.

To determine the resonance properties of the band-pass filter, the gain was then measured as a function of frequency. The resulting resonance curve was fitted with a Lorentzian model,

$$G(f) = y_0 + \frac{A}{1 + \left(\frac{f - f_0}{\gamma}\right)^2}, \tag{8}$$

where f_0 is the resonance frequency and γ is the half-width at half-maximum (HWHM). Figure 7 shows the measured gain in decibels together with the fitted Lorentzian function.

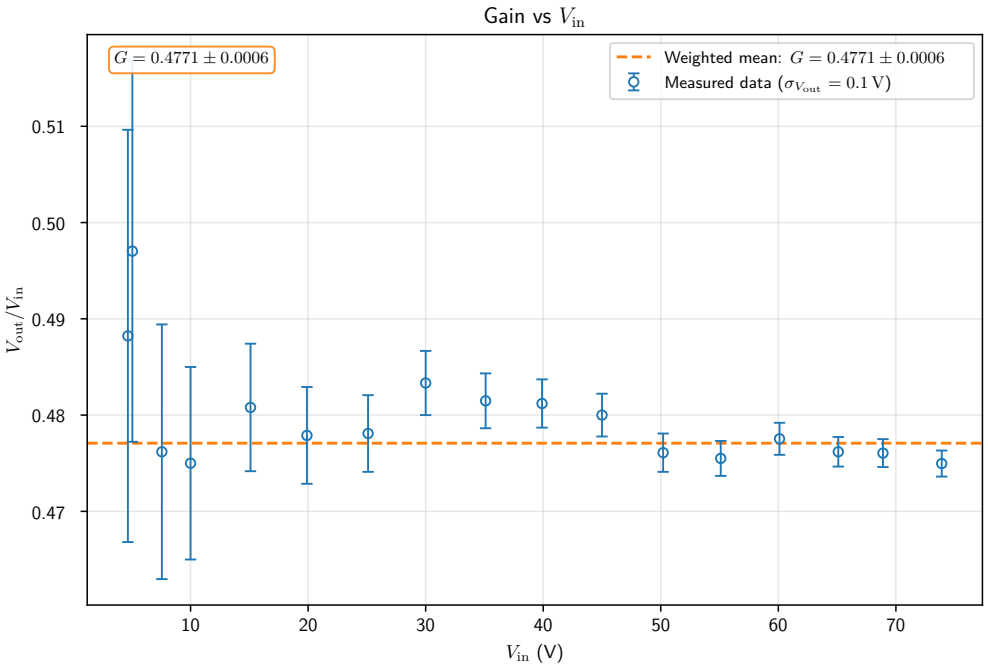

Figure 7: Measured gain $G = V_{\text{out}}/V_{\text{in}}$ as a function of the input voltage V_{in}. The error bars represent the propagated uncertainty resulting from the measurement uncertainty of V_{out}.

The fit yielded a resonance frequency of

$$f_0 = (95.0 \pm 0.8)\,\text{kHz}, \tag{9}$$

and a half-width at half-maximum of

$$\gamma = (3.7 \pm 1.6)\,\text{kHz}. \tag{10}$$

From this, the full width at half maximum was obtained as

$$\text{FWHM} = 2\gamma = (7.4 \pm 3.2)\,\text{kHz}, \tag{11}$$

which gives the quality factor

$$Q = \frac{f_0}{\text{FWHM}} = 13 \pm 5. \tag{12}$$

The maximum gain of the filter was found to be

$$G_{\text{max}} = 0.1500 \pm 0.0024, \tag{13}$$

$$20 \log_{10}(G_{\text{max}}) = (-16.48 \pm 0.14)\,\text{dB}. \tag{14}$$

Influence of f_M, Modulation Depth m, and Offset OS

Change of Modulation Frequency f_M

Changing the modulation frequency f_M affects the temporal behavior of the envelope and the spacing of the spectral components. Increasing f_M results in a faster oscillation of the envelope in the time domain. In the frequency spectrum, the sidebands appear at

$$f = f_C \pm f_M, \tag{15}$$

where f_C is the carrier frequency. Therefore, increasing f_M increases the distance between the carrier peak and the sidebands, while decreasing f_M reduces this spacing.

Change of Modulation Depth m

Adjusting the **MOD** control changes the modulation depth m and therefore the maximum and minimum values of the envelope. A larger modulation depth increases the difference between envelope maximum and minimum, resulting in a stronger amplitude variation. In the frequency spectrum, increasing m increases the amplitude of the sidebands relative to the carrier.

Change of Offset OS

Changing the **OFFSET** modifies the DC component of the modulating signal and affects the symmetry and overall shape of the envelope. Depending on the offset value, the envelope may appear shifted or distorted. If the offset is adjusted such that the carrier component is suppressed, the system approaches balanced modulation, where the carrier peak in the frequency spectrum disappears.

Determination of the Modulation Depth

The modulation depth m of the amplitude-modulated signal was determined using three independent methods: from the X–Y plot, from the time-domain oscilloscope trace (KO display), and from the frequency spectrum. Ideally, the modulation depth should be determined from three independent data sets for each of the three methods. However, only one recorded data set was available for each method in the present experiment, so the following analysis is based on one data set per method. For the first two methods, the relevant vertical dimensions were measured directly from screenshots in pixel units using the *Screen Ruler* utility from Microsoft PowerToys.
In the following, the modulation depth is defined as

$$m = \frac{A_{\max} - A_{\min}}{A_{\max} + A_{\min}}, \tag{16}$$

where $A_{\max}$ and $A_{\min}$ denote the maximum and minimum envelope amplitudes, respectively. In practice, the full vertical signal heights $H_{\max}$ and $H_{\min}$ were measured instead of the amplitudes themselves. Since

$$A_{\max} = \frac{H_{\max}}{2}, \qquad A_{\min} = \frac{H_{\min}}{2}, \tag{17}$$

the same expression can equivalently be written as

$$m = \frac{H_{\max} - H_{\min}}{H_{\max} + H_{\min}}. \tag{18}$$

Thus, the quantities A and H are used interchangeably in the following wherever only their ratio enters the calculation of m.

Determination of the Modulation Depth via X–Y Plot

In X–Y mode, the modulating signal was connected to the horizontal axis and the AM signal to the vertical axis, resulting in the characteristic trapezoidal pattern. The maximum and minimum vertical extensions of the trapezoid were measured directly from the screenshot fig. 1. The measured values were

$$H_{\max} = (1315 \pm 5)\,\text{px}, \qquad H_{\min} = (318 \pm 5)\,\text{px}.$$

A conservative uncertainty of 5 px was assigned to both values to account for the finite line thickness, limited image sharpness, and the manual placement of the measurement markers. Using eq. 18 the modulation depth was found to be

$$m = 0.611 \pm 0.005. \tag{19}$$

Determination of the Modulation Depth via Time-Domain Oscilloscope Trace

In the normal oscilloscope time-domain display (KO), the envelope of the amplitude-modulated signal was used to determine the maximum and minimum vertical signal heights. Three repeated measurements were taken from the screenshot fig. 2, yielding

$$H_{\max} = (1135,\ 1136,\ 1130)\ \text{px}, \qquad H_{\min} = (179,\ 177,\ 182)\ \text{px}.$$

From these values, the mean heights and their uncertainties were obtained as

$$H_{\max} = (1133.7 \pm 1.9)\ \text{px}, \qquad H_{\min} = (179.3 \pm 1.5)\ \text{px}.$$

Using eq. 18 the modulation depth was determined to be

$$m = 0.7268 \pm 0.0019. \tag{20}$$

Determination of the Modulation Depth via Frequency Spectrum

In the FFT spectrum of the amplitude-modulated signal, the carrier peak and the two sidebands were identified manually from the recorded spectrum. The selected peak frequencies were

$$f_{\mathrm{LSB}} = 705.3 \text{ Hz}, \qquad f_C = 754.6 \text{ Hz}, \qquad f_{\mathrm{USB}} = 803.9 \text{ Hz}.$$

The corresponding amplitudes read from the FFT trace were

$$A_{\mathrm{LSB}} = -32.95 \text{ dB}, \qquad A_C = 13.45 \text{ dB}, \qquad A_{\mathrm{USB}} = -33.75 \text{ dB}.$$

Since the FFT amplitudes were given in decibels, they were converted to linear amplitudes according to

$$A_{\mathrm{lin}} = 10^{A_{\mathrm{dB}}/20}. \tag{21}$$

For an amplitude-modulated signal with carrier, the sideband amplitudes satisfy

$$A_{\mathrm{LSB}} = A_{\mathrm{USB}} = \frac{m}{2} A_C.$$

Thus, the modulation depth can be obtained from the spectrum using

$$m = \frac{A_{\mathrm{LSB}} + A_{\mathrm{USB}}}{A_C}. \tag{22}$$

Using Eq. (21) and Eq. (22), the modulation depth was found to be

$$m = 0.0092 \pm 0.0004. \tag{23}$$

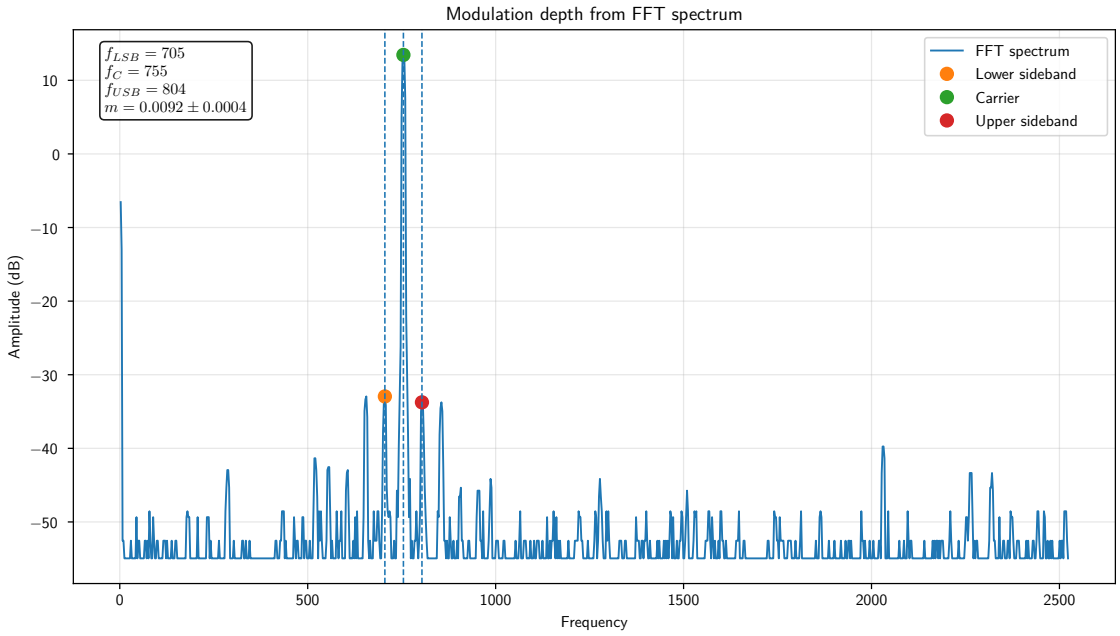

Figure 8: FFT spectrum of the amplitude-modulated signal. The carrier peak and the two sidebands used for the spectral determination of the modulation depth are marked.

Comparison of the Three Methods

For clarity, the modulation depths obtained with the three independent methods are summarized in Table 6. The X–Y plot and the time-domain oscilloscope trace yield values of the same order of magnitude, whereas the value obtained from the FFT spectrum is significantly smaller.

Table 6: Comparison of the modulation depth determined by the three evaluation methods.

Method	Modulation depth m
X–Y plot	0.611 ± 0.005
Time-domain oscilloscope trace (KO)	0.727 ± 0.002
Frequency spectrum (FFT)	0.0092 ± 0.0004

Finding Fourier Spectra

The Fourier spectra of different periodic input signals were analyzed using the FFT function of the digital oscilloscope. The exported spectra were evaluated by identifying the dominant spectral peaks and comparing their positions with the expected harmonic frequencies.
For both the sinusoidal and triangular signals, the strongest component was found near

$$f_1 \approx 34.5 \text{ kHz}, \tag{24}$$

which is identified as the fundamental frequency of the excitation signal. Additional peaks at higher frequencies were interpreted as harmonic components.

Sinusoidal Signal

For the sinusoidal signal, the dominant spectral peak was found at

$$f_1 = (34.5 \pm 0.3) \text{ kHz}, \qquad A_1 = (-6.6 \pm 0.4) \text{ dB}. \tag{25}$$

A further pronounced component appeared at

$$f_3 = (103.6 \pm 0.3) \text{ kHz}, \qquad A_3 = (-11.8 \pm 0.6) \text{ dB}. \tag{26}$$

Since

$$103.6 \text{ kHz} \approx 3 \cdot 34.5 \text{ kHz}, \tag{27}$$

this peak can be assigned to the third harmonic. Ideally, a purely sinusoidal signal should contain only the fundamental frequency. The appearance of higher-frequency components therefore indicates deviations from an ideal sine wave, for example due to signal distortion, limited generator quality, or instrumental effects of the measurement chain.

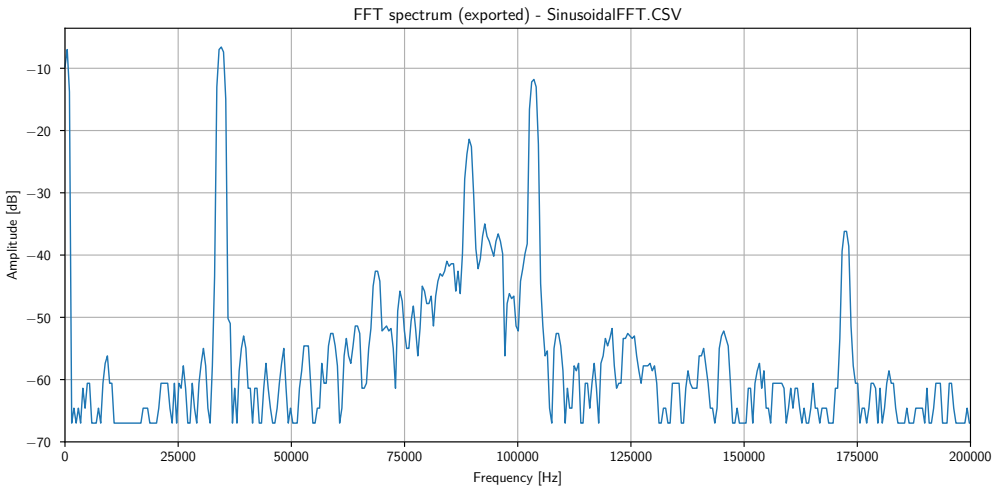

Figure 9: FFT spectrum of the sinusoidal signal. The dominant peak corresponds to the fundamental frequency at approximately 34.5 kHz, while a smaller peak is visible near the third harmonic at approximately 103.6 kHz.

Triangular Signal

For the triangular signal, the fundamental component was observed at

$$f_1 = (34.5 \pm 0.3)\ \text{kHz}, \qquad A_1 = (16.2 \pm 0.4)\ \text{dB}. \tag{28}$$

A pronounced higher harmonic was found at

$$f_3 = (103.6 \pm 0.3)\ \text{kHz}, \qquad A_3 = (-3.4 \pm 0.6)\ \text{dB}. \tag{29}$$

Again,

$$103.6\ \text{kHz} \approx 3 \cdot 34.5\ \text{kHz}, \tag{30}$$

so this component is identified as the third harmonic. This is consistent with the Fourier series of a triangular waveform, which contains predominantly odd harmonics, while the amplitudes decrease strongly with increasing harmonic order. Compared with the sinusoidal signal, the triangular signal therefore exhibits a richer harmonic structure.

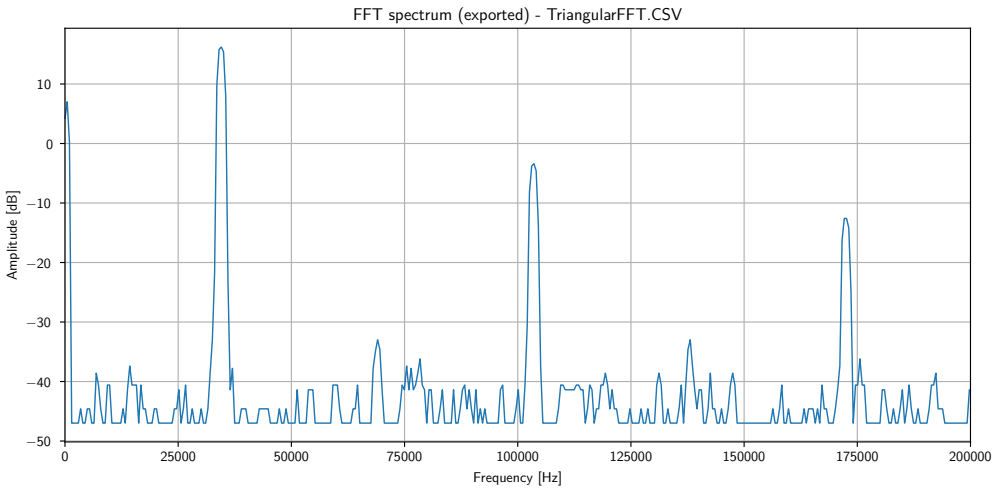

Figure 10: FFT spectrum of the triangular signal. Besides the fundamental frequency at approximately 34.5 kHz, a clear third harmonic near 103.6 kHz is visible, as expected for a triangular waveform.

Comparison of the Spectra

The extracted dominant spectral components are summarized in Table 7. In both cases, the fundamental frequency is located at approximately 34.5 kHz. For the sinusoidal signal, the spectrum is dominated by the fundamental component, with only a comparatively weak higher harmonic. For the triangular signal, higher harmonics are more clearly visible, in agreement with the expected Fourier decomposition of a non-sinusoidal periodic waveform.

Table 7: Dominant spectral peaks extracted from the FFT spectra.

Signal	Frequency (kHz)	Amplitude (dB)
Sinusoidal	34.5 ± 0.3	-6.6 ± 0.4
Sinusoidal	103.6 ± 0.3	-11.8 ± 0.6
Triangular	34.5 ± 0.3	16.2 ± 0.4
Triangular	103.6 ± 0.3	-3.4 ± 0.6

Overall, the FFT analysis confirms the expected spectral behavior of the investigated signals. The sinusoidal waveform is characterized mainly by its fundamental frequency, whereas the triangular waveform contains pronounced odd harmonics. Small additional peaks and fluctuations in the spectra can be attributed to measurement noise, the finite frequency resolution of the FFT, and non-ideal signal generation.

Determining the Total Harmonic Distortion

To quantify the distortion of a low-quality sinusoidal signal, the total harmonic distortion (THD) was determined from the measured Fourier spectrum. The FFT spectrum recorded on the oscilloscope is shown in Figure 11.

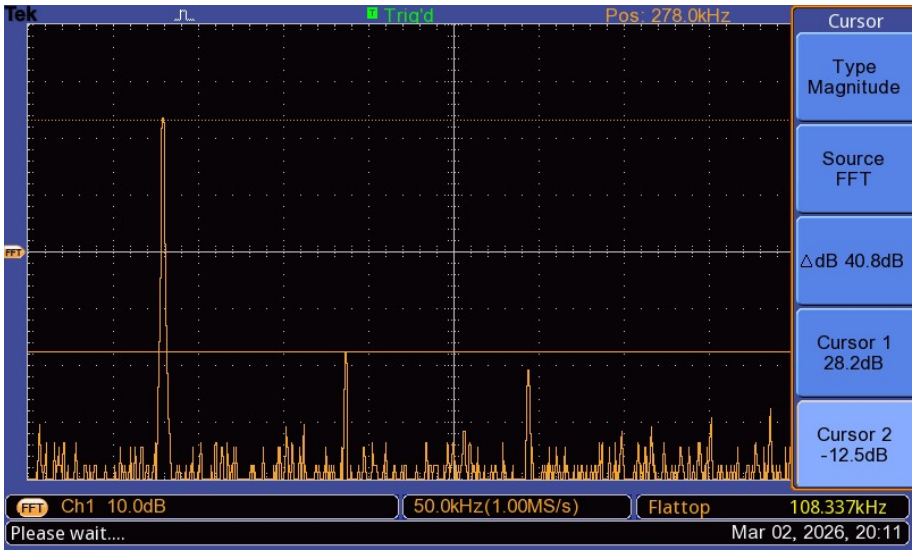

Figure 11: FFT spectrum of a distorted sinusoidal signal measured with the oscilloscope. The cursor positions mark the amplitudes used to determine the harmonic distortion.

From the spectrum, the three dominant components were identified as:

$$A_1 = (28.2 \pm 0.1)\ \text{dB}, \qquad A_2 = (-12.5 \pm 0.1)\ \text{dB}, \qquad A_3 = (-16.1 \pm 0.1)\ \text{dB}. \tag{31}$$

Here A_1 represents the amplitude of the fundamental frequency, while A_2 and A_3 correspond to higher harmonic components.
Since the FFT amplitudes are given in decibel units, they were first converted to linear amplitudes using

$$A_{\text{lin}} = 10^{A_{\text{dB}}/20}. \tag{32}$$

The total harmonic distortion (THD) is defined as the ratio of the root-mean-square amplitude of the harmonic components to the amplitude of the fundamental frequency:

$$\text{THD} = \sqrt{\frac{A_2^2 + A_3^2}{A_1^2}}. \tag{33}$$

Using the measured amplitudes, the distortion was calculated as

$$\text{THD} \approx 0.0111 \pm 0.0002 = (1.11 \pm 0.02)\%. \tag{34}$$

This result indicates that the investigated signal contains a small but measurable contribution from higher harmonics. Ideally, a perfect sinusoidal signal would exhibit only the fundamental frequency in the Fourier spectrum, corresponding to a THD of zero. The observed distortion can be attributed to imperfections of the signal generator and measurement electronics.

Discussion

The performed measurements allow a comprehensive investigation of harmonic analysis and amplitude modulation using both frequency-selective techniques and Fourier-based spectral analysis. Overall, the experimental results show good agreement with the theoretical expectations for linear systems, modulation behavior, and Fourier spectra of periodic signals.

The characteristics of the precision rectifier were first examined for the gain settings K_1 and K_{10}. For the lower gain setting, the measured data followed a linear relation between V_{out} and V_{in}, yielding a proportionality factor of $K_1 = 0.8962 \pm 0.0018$. This confirms that the rectifier operates approximately linearly in this regime. For the higher gain setting, linear behavior was observed only for input voltages up to approximately $V_{\text{in}} \leq 16$ V. Beyond this limit, the output voltage saturated at about 141 V, indicating the maximum output range of the rectifier electronics. Such saturation effects are expected in practical circuits due to limited supply voltages and amplifier constraints.

The linearity of the band-pass filter was investigated close to its resonance frequency. The measured relation between V_{out} and V_{in} showed excellent linear behavior with a regression coefficient $R^2 = 0.99994$. This confirms that the filter operated within its linear regime for the investigated amplitude range. The measured gain of approximately $G = 0.477 \pm 0.001$ remained nearly constant over the investigated voltage interval, indicating that no significant nonlinear effects or signal distortions occurred during these measurements.

The resonance behavior of the band-pass filter was analyzed by measuring the gain as a function of frequency. A Lorentzian function provided a good description of the measured resonance curve. The fit yielded a resonance frequency of $f_0 = (95.0 \pm 0.8)$ kHz and a full width at half maximum of FWHM $= (7.4 \pm 3.2)$ kHz, corresponding to a quality factor of $Q \approx 13 \pm 5$. The comparatively small quality factor compared to the value suggested in the laboratory manual can be attributed to additional damping in the circuit, non-ideal electronic components, and the finite bandwidth of the measurement electronics [1]. While the laboratory description suggests a considerably higher theoretical quality factor, the experimentally observed value is smaller. This discrepancy can be attributed to several factors, including additional damping in the electronic components, non-ideal filter elements, and measurement uncertainties in the amplitude determination. In practical analog filters, such deviations from ideal behavior are common due to resistive losses and finite component tolerances.

The behavior of amplitude modulation was investigated using three independent methods for determining the modulation depth: the X–Y plot, the time-domain oscilloscope trace (KO display), and the frequency spectrum. In X–Y mode, the oscilloscope showed the characteristic trapezoidal pattern of an amplitude-modulated signal, from which a modulation depth of $m = 0.611 \pm 0.005$ was obtained. The evaluation of the time-domain envelope yielded $m = 0.727 \pm 0.002$, which is of the same order of magnitude, although somewhat larger than the value obtained from the X–Y plot. Both results indicate moderate modulation, for which the envelope remains clearly visible and no overmodulation occurs.

In contrast, the modulation depth obtained from the FFT spectrum, $m = 0.0092 \pm 0.0004$, is significantly smaller than the values determined by the other two methods and is therefore not consistent with them. This strongly suggests that the spectral evaluation is affected by systematic errors or by an incorrect identification of the relevant sideband amplitudes. Possible reasons include the limited frequency resolution of the recorded FFT trace, insufficient separation of the sidebands from nearby spectral peaks, inaccuracies in the manual peak selection, or additional spectral components and noise. The FFT-based result should therefore be treated with caution.

The Fourier spectra obtained from the digital oscilloscope further illustrate the spectral composition of periodic signals. For the sinusoidal waveform, the spectrum was dominated by a strong fundamental component at approximately (34.5 ± 0.3) kHz. A smaller peak at about (103.6 ± 0.3) kHz corresponds to the third harmonic. Ideally, a perfect sinusoidal signal should contain only the fundamental frequency; therefore, the presence of higher harmonics indicates slight distortions of the generated waveform or measurement artifacts in the signal chain.

For the triangular signal, the spectrum contained a more pronounced harmonic structure. In addition to the fundamental frequency, a clear third harmonic was observed, which is consistent with the Fourier series representation of a triangular waveform. Such signals are known to contain only odd harmonics, with amplitudes decreasing rapidly with increasing harmonic order. The experimental spectra, therefore, qualitatively confirm the theoretical harmonic content of triangular signals.

Finally, the total harmonic distortion of a deliberately degraded sinusoidal signal was determined from its Fourier spectrum. The calculated value of THD $\approx (1.11 \pm 0.02)\%$ indicates that only a small fraction of the signal power is contained in higher harmonics. Although this distortion is clearly measurable, the signal can still be considered relatively pure. The remaining harmonic components are most likely caused by imperfections of the

signal generator, nonlinearities in the electronic components, and the finite resolution of the FFT measurement.

Overall, the experiment demonstrates the close connection between time-domain signals and their frequency-domain representation. The combination of frequency-selective filtering, FFT analysis, and numerical evaluation provides consistent results and confirms the theoretical predictions of Fourier analysis and linear system response.

Conclusion

In this experiment, harmonic analysis of periodic signals was investigated using both frequency-selective measurements and Fourier-based spectral analysis. The characteristics of the precision rectifier and of the band-pass filter were successfully determined, showing linear behavior within the expected operating ranges. For the filter, the resonance frequency was found to be $f_0 = (95.0 \pm 0.8)\,\mathrm{kHz}$, with a corresponding full width at half maximum of $\mathrm{FWHM} = (7.4 \pm 3.2)\,\mathrm{kHz}$ and a quality factor of $Q = 13 \pm 5$.

The behavior of amplitude modulation was examined using three independent methods for determining the modulation depth. The X–Y plot yielded $m = 0.611 \pm 0.005$, while the time-domain oscilloscope trace gave $m = 0.727 \pm 0.002$. These two values are of the same order of magnitude and both indicate moderate modulation without overmodulation. In contrast, the value obtained from the FFT spectrum, $m = 0.0092 \pm 0.0004$, was significantly smaller and therefore not consistent with the other two methods. This discrepancy suggests that the spectral determination was affected by systematic uncertainties, most likely related to the identification of the relevant sideband amplitudes and the limited resolution of the available FFT trace.

Furthermore, the Fourier spectra of sinusoidal and triangular signals were analyzed using the FFT function of the digital oscilloscope. The measured spectra confirmed the expected harmonic structure of the signals: the sinusoidal waveform was dominated by its fundamental frequency, while the triangular waveform exhibited a more pronounced odd-harmonic content, including a clear third harmonic. Finally, the total harmonic distortion of a distorted sinusoidal signal was determined to be $\mathrm{THD} = (1.11 \pm 0.02)\%$, indicating a small but measurable contribution from higher harmonics.

Overall, the experiment demonstrates the close connection between time-domain signals and their frequency-domain representation. With the exception of the FFT-based modulation-depth determination, the measurements are consistent with the theoretical expectations of Fourier analysis, amplitude modulation, and linear system response.

Appendix

Acknowledgments

I would like to thank my teaching assistant, Takahiro Uto, for his supervision and support during the experiment. His guidance and explanations were very helpful throughout the laboratory work and the interpretation of the results.

AI Usage Declaration

This report was prepared with assistance from ChatGPT (OpenAI) for language polishing, formatting suggestions, and for helping to develop, debug, and correct the syntax of the Python code. The AI was not used to generate raw data, to select results, or to determine the final conclusions. All scientific content (methods, calculations, results, and interpretation) was critically reviewed and validated by the authors. The authors take full responsibility for the accuracy of the analysis and for any remaining errors. No confidential or personal data were provided to the AI system during the preparation of this report.

References

[1] Paul Horowitz and Winfield Hill. *The Art of Electronics*. 3rd ed. Cambridge University Press, 2015.

[2] Reiner Mühle. *Harmonic Analysis*. Advanced Physics Laboratory Manual. Translation by Colin v. Negenborn (2010); laboratory manual for the Advanced Physics Laboratory, ETH Zürich; accessed via Moodle. ETH Zürich, Department of Physics, 2001.

Python Code

```python
import os
import glob
import re

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from scipy.stats import linregress
from scipy.optimize import curve_fit

from scipy.signal import find_peaks


# ----------------------------------------
# 5.1 Characteristics of the Rectifier
# ----------------------------------------

# K1 data
V_in_k1 = np.array([
    4.71, 5.03, 7.43, 9.78, 12.1, 14.7, 18.3, 21.4, 23.4, 26.0,
    30.2, 37.4, 42.2, 46.8, 50.4, 55.5, 61.2, 65.3, 70.5, 74.5
])
V_out_k1 = np.array([
    4.11, 4.40, 6.43, 8.53, 10.6, 12.8, 16.0, 18.8, 20.3, 22.6,
    26.8, 33.6, 37.9, 42.2, 45.4, 49.6, 54.9, 58.7, 63.4, 67.0
])

# K10 data
V_in_k10 = np.array([
    4.71, 5.07, 7.41, 9.94, 12.1, 14.6, 15.3, 15.8, 16.0, 16.4,
    18.5, 21.3, 23.3, 30.2, 40.2, 50.3, 74.5
])
V_out_k10 = np.array([
    41.2, 44.5, 64.5, 86.5, 105, 128, 135, 139, 141, 141,
    141, 141, 141, 141, 141, 141, 141
])

# -------------------------
# Measurement uncertainty (output only)
```

```
# -------------------------
V_out_err_k1 = np.full_like(V_out_k1, 0.1, dtype=float)
V_out_err_k10 = np.full_like(V_out_k10, 0.1, dtype=float)

# -------------------------
# Linear range cutoff for K10
# -------------------------
cutoff_k10 = 16.0  # linear behavior assumed for V_in <= 16.0 V

# -------------------------
# Fit through origin: V_out = K * V_in
# with standard error from residuals
# -------------------------
def fit_through_origin_with_error(x: np.ndarray, y: np.ndarray):
    """
    Least-squares fit through the origin: y = K*x
    Returns:
        K       : fitted slope
        K_err   : standard error of slope from residual scatter
        y_fit   : fitted y values
        resid   : residuals
    """
    K = (x @ y) / (x @ x)
    y_fit = K * x
    resid = y - y_fit
    n = len(x)

    # standard error of slope for fit through origin
    K_err = np.sqrt((resid @ resid) / ((n - 1) * (x @ x)))
    return float(K), float(K_err), y_fit, resid

# -------------------------
# Rounding helper
# Rule:
# - usually 1 significant digit in uncertainty
# - 2 significant digits if first digit is 1 or 2
# - value rounded to same decimal place as uncertainty
# -------------------------
def round_value_uncertainty(value: float, uncertainty: float):
    if uncertainty <= 0 or not np.isfinite(uncertainty):
        return value, uncertainty, str(value), str(uncertainty)
```

```
82
83      exponent = int(np.floor(np.log10(abs(uncertainty))))
84      mantissa = abs(uncertainty) / (10 ** exponent)
85
86      # 2 sig figs if leading digit is 1 or 2, else 1 sig fig
87      sig_figs = 2 if mantissa < 3 else 1
88      decimals = -exponent + (sig_figs - 1)
89
90      uncertainty_rounded = round(uncertainty, decimals)
91      value_rounded = round(value, decimals)
92
93      value_str = f"{value_rounded:.{decimals}f}"
94      uncertainty_str = f"{uncertainty_rounded:.{decimals}f}"
95
96      return value_rounded, uncertainty_rounded, value_str, uncertainty_str
97
98  # -------------------------
99  # Perform fits
100 # -------------------------
101 K1, K1_err, _, _ = fit_through_origin_with_error(V_in_k1, V_out_k1)
102
103 mask_k10_linear = V_in_k10 <= cutoff_k10
104 K10, K10_err, _, _ = fit_through_origin_with_error(
105     V_in_k10[mask_k10_linear],
106     V_out_k10[mask_k10_linear]
107 )
108
109 # Rounded values for report output
110 K1_r, K1_err_r, K1_str, K1_err_str = round_value_uncertainty(K1, K1_err)
111 K10_r, K10_err_r, K10_str, K10_err_str = round_value_uncertainty(K10, K10_err)
112
113 # -------------------------
114 # Print output
115 # -------------------------
116 print("Raw fit results:")
117 print(f"K1   = {K1:.6f} pm {K1_err:.6f}")
118 print(f"K10  = {K10:.6f} pm {K10_err:.6f}")
119
120 print("\nReport-ready rounded results:")
121 print(f"K1   = {K1_str} pm {K1_err_str}")
122 print(f"K10  = {K10_str} pm {K10_err_str}")
```

```
123
124 # -------------------------
125 # Plot
126 # -------------------------
127 fig, ax = plt.subplots(1, 2, figsize=(12, 4.8))
128
129 # ---- K1 ----
130 ax[0].errorbar(
131     V_in_k1, V_out_k1,
132     yerr=V_out_err_k1,
133     fmt="o",
134     markersize=4,
135     markerfacecolor="white",
136     markeredgewidth=1.0,
137     capsize=3,
138     elinewidth=1.1,
139     barsabove=True,
140     zorder=3,
141     label=r"Measured data ($\sigma_{V_{\mathrm{out}}}=0.1\ \mathrm{V}$)"
142 )
143
144 xline_k1 = np.linspace(0, V_in_k1.max() * 1.05, 300)
145 ax[0].plot(
146     xline_k1, K1 * xline_k1,
147     linestyle="--",
148     linewidth=1.5,
149     label=rf"Fit: $V_{{out}} = ({K1_str}\pm {K1_err_str})\,V_{{in}}$"
150 )
151
152 ax[0].set_title(r"Rectifier $K_1$")
153 ax[0].set_xlabel(r"$V_{in}$ (V)")
154 ax[0].set_ylabel(r"$V_{out}$ (V)")
155 ax[0].grid(True, alpha=0.3)
156 ax[0].legend(fontsize=9)
157
158 # ---- K10 ----
159 ax[1].errorbar(
160     V_in_k10, V_out_k10,
161     yerr=V_out_err_k10,
162     fmt="o",
163     markersize=4,
```

```
    markerfacecolor="white",
    markeredgewidth=1.0,
    capsize=3,
    elinewidth=1.1,
    barsabove=True,
    zorder=3,
    label=r"Measured data ($\sigma_{V_{\mathrm{out}}}=0.1\ \mathrm{V}$)"
)

xline_k10 = np.linspace(0, cutoff_k10, 300)
ax[1].plot(
    xline_k10, K10 * xline_k10,
    linestyle="--",
    linewidth=1.5,
    label=rf"Fit ($V_{{in}}\leq {cutoff_k10:.1f}\,\mathrm{{V}}$): "
          rf"$V_{{out}} = ({K10_str}\pm {K10_err_str})\,V_{{in}}$"
)

ax[1].axvline(
    cutoff_k10,
    linestyle="--",
    linewidth=1.2,
    label=rf"Cutoff: $V_{{in}}={cutoff_k10:.1f}\,\mathrm{{V}}$"
)

ax[1].set_title(r"Rectifier $K_{10}$")
ax[1].set_xlabel(r"$V_{in}$ (V)")
ax[1].set_ylabel(r"$V_{out}$ (V)")
ax[1].grid(True, alpha=0.3)
ax[1].legend(fontsize=9)

plt.tight_layout()
plt.show()



# ----------------------------------
# 5.2a) Characteristics of the Filter
# ----------------------------------

V_in = np.array([
```

```
    4.67, 5.05, 7.56, 10.0, 15.1, 19.9, 25.1, 30.0,
    35.1, 39.9, 45.0, 50.2, 55.1, 60.1, 65.1, 68.9, 73.9
], dtype=float)

V_out = np.array([
    2.28, 2.51, 3.60, 4.75, 7.26, 9.51, 12.0, 14.5,
    16.9, 19.2, 21.6, 23.9, 26.2, 28.7, 31.0, 32.8, 35.1
], dtype=float)

# Measurement uncertainty (output only)
err_out = np.full_like(V_out, 0.1, dtype=float)

# ---------------------------------
# Linear model
# ---------------------------------
def linear_model(x, a, b):
    return a * x + b

# ---------------------------------
# Rounding helper
# Rule:
# - uncertainty usually 1 significant digit
# - use 2 significant digits if first digit is 1 or 2
# - value rounded to same decimal place as uncertainty
# ---------------------------------
def round_value_uncertainty(value: float, uncertainty: float):
    if uncertainty <= 0 or not np.isfinite(uncertainty):
        return str(value), str(uncertainty)

    exponent = int(np.floor(np.log10(abs(uncertainty))))
    mantissa = abs(uncertainty) / (10 ** exponent)

    sig_figs = 2 if mantissa < 3 else 1
    decimals = -exponent + (sig_figs - 1)

    value_rounded = round(value, decimals)
    uncertainty_rounded = round(uncertainty, decimals)

    value_str = f"{value_rounded:.{decimals}f}"
    uncertainty_str = f"{uncertainty_rounded:.{decimals}f}"
    return value_str, uncertainty_str
```

```
246
247 # ---------------------------------
248 # Unweighted fit
249 # -> parameter errors come from residual scatter
250 # ---------------------------------
251 popt, pcov = curve_fit(linear_model, V_in, V_out)
252 slope, intercept = popt
253 slope_err, intercept_err = np.sqrt(np.diag(pcov))
254
255 # ---------------------------------
256 # Goodness of fit: R^2
257 # ---------------------------------
258 V_fit = linear_model(V_in, slope, intercept)
259 ss_res = np.sum((V_out - V_fit) ** 2)
260 ss_tot = np.sum((V_out - np.mean(V_out)) ** 2)
261 R2 = 1.0 - ss_res / ss_tot
262
263 # ---------------------------------
264 # Rounded values for report
265 # ---------------------------------
266 slope_str, slope_err_str = round_value_uncertainty(slope, slope_err)
267 intercept_str, intercept_err_str = round_value_uncertainty(intercept,
    intercept_err)
268
269 # ---------------------------------
270 # Print output
271 # ---------------------------------
272 print("Raw fit results:")
273 print(f"slope     = {slope:.8f} pm {slope_err:.8f}")
274 print(f"intercept = {intercept:.8f} pm {intercept_err:.8f} V")
275 print(f"R^2       = {R2:.8f}")
276
277 print("\nReport-ready rounded results:")
278 print(f"slope     = {slope_str} pm {slope_err_str}")
279 print(f"intercept = {intercept_str} pm {intercept_err_str} V")
280 print(f"R^2       = {R2:.5f}")
281
282 print("\nSuggested equation for the report:")
283 print(
284     rf"V_out = ({slope_str} pm {slope_err_str}) V_in + "
285     rf"({intercept_str} pm {intercept_err_str}) V"
```

```
)

# ---------------------------------
# Plot: V_out vs V_in with error bars
# ---------------------------------
plt.figure(figsize=(7.5, 5.2))

plt.errorbar(
    V_in,
    V_out,
    yerr=err_out,
    fmt="o",
    markersize=5,
    markerfacecolor="white",
    markeredgewidth=1.0,
    capsize=3,
    elinewidth=1.0,
    barsabove=True,
    zorder=3,
    label=r"Measured data ($\sigma_{V_{\mathrm{out}}}=0.1\,\mathrm{V}$)"
)

xline = np.linspace(0, max(V_in) * 1.05, 300)
plt.plot(
    xline,
    linear_model(xline, slope, intercept),
    linestyle="--",
    linewidth=1.7,
    label=(
        rf"Linear fit: $V_{{out}} = ({slope_str}\pm {slope_err_str})\,V_{{in}}$"
        "\n"
        rf"$\qquad\quad +\,({intercept_str}\pm {intercept_err_str})\,\mathrm{{V}}$"
    )
)

# Optional result box inside the plot
summary_text = (
    rf"$a = {slope_str}\pm {slope_err_str}$" "\n"
    rf"$b = {intercept_str}\pm {intercept_err_str}\,\mathrm{{V}}$" "\n"
```

```
    rf"$R^2 = {R2:.5f}$"
)

plt.text(
    0.03, 0.97, summary_text,
    transform=plt.gca().transAxes,
    fontsize=9,
    verticalalignment="top",
    bbox=dict(boxstyle="round", facecolor="white", alpha=0.9)
)

plt.xlabel(r"$V_{in}$ (V)")
plt.ylabel(r"$V_{out}$ (V)")
plt.title(r"Band-pass Filter: $V_{out}$ vs $V_{in}$")
plt.grid(True, alpha=0.3)
plt.legend(fontsize=9)
plt.tight_layout()
plt.show()




# ----------------------------------
# Plot 2: Gain vs V_in with propagated uncertainty
# Assumption: only V_out has uncertainty, V_in treated as exact
# ----------------------------------

V_in = np.array([
    4.67, 5.05, 7.56, 10.0, 15.1, 19.9, 25.1, 30.0,
    35.1, 39.9, 45.0, 50.2, 55.1, 60.1, 65.1, 68.9, 73.9
], dtype=float)

V_out = np.array([
    2.28, 2.51, 3.60, 4.75, 7.26, 9.51, 12.0, 14.5,
    16.9, 19.2, 21.6, 23.9, 26.2, 28.7, 31.0, 32.8, 35.1
], dtype=float)

err_out = np.full_like(V_out, 0.1, dtype=float)

# Gain and propagated uncertainty
gain = V_out / V_in
```

```
gain_error = err_out / V_in

# Optional: weighted mean gain
weights = 1.0 / gain_error**2
gain_mean = np.sum(weights * gain) / np.sum(weights)
gain_mean_err = np.sqrt(1.0 / np.sum(weights))

# ---------------------------------
# Plot
# ---------------------------------
plt.figure(figsize=(7.5, 5.2))

plt.errorbar(
    V_in,
    gain,
    yerr=gain_error,
    fmt="o",
    markersize=5,
    markerfacecolor="white",
    markeredgewidth=1.0,
    capsize=3,
    elinewidth=1.0,
    barsabove=True,
    zorder=3,
    label=r"Measured data ($\sigma_{V_{\mathrm{out}}}=0.1\,\mathrm{V}$)"
)

# Optional horizontal line for average gain
plt.axhline(
    gain_mean,
    linestyle="--",
    linewidth=1.5,
    color="tab:orange",
    label=rf"Weighted mean: $G = {gain_mean:.4f} \pm {gain_mean_err:.4f}$"
)

# Optional textbox
summary_text = (
    rf"$G = {gain_mean:.4f} \pm {gain_mean_err:.4f}$"
)

```

```
plt.text(
    0.03, 0.97, summary_text,
    transform=plt.gca().transAxes,
    fontsize=9,
    verticalalignment="top",
    bbox=dict(boxstyle="round", facecolor="white", edgecolor="tab:orange",
    alpha=0.9)
)

plt.xlabel(r"$V_{\mathrm{in}}$ (V)")
plt.ylabel(r"$V_{\mathrm{out}}/V_{\mathrm{in}}$")
plt.title(r"Gain vs $V_{\mathrm{in}}$")
plt.grid(True, alpha=0.3)
plt.legend(fontsize=9)
plt.tight_layout()
plt.show()




# ---------------------------------
# 5.2b) Characteristics of the Filter (Resonance Curve)
# ---------------------------------

V_in = 3.74  # V (treated as exact here)

f_khz = np.array([59.8, 70.5, 80.6, 85.1, 88.9, 91.8, 93.8, 95.9, 98.2, 100.0,
    105.1, 110.2])
V_out = np.array([0.511, 0.522, 0.526, 0.527, 0.532, 0.545, 0.552, 0.563,
    0.542, 0.530, 0.523, 0.522])
V_out_err = np.full_like(V_out, 0.010, dtype=float)  # V

# ----------------------------
# Gain and propagated errors
# ----------------------------
G = V_out / V_in
G_err = V_out_err / V_in

G_dB = 20.0 * np.log10(G)
G_dB_err = (20.0 / np.log(10.0)) * (G_err / G)

```

```
# ----------------------------
# Lorentzian model in linear gain
# G(f) = y0 + A / (1 + ((f - f0)/gamma)^2)
# gamma = HWHM, so FWHM = 2*gamma
# ----------------------------
def lorentzian(f, A, f0, gamma, y0):
    return y0 + A / (1.0 + ((f - f0) / gamma)**2)

# ----------------------------
# Rounding helper:
# uncertainty -> usually 1 sig fig
# use 2 sig figs if first digit is 1 or 2
# value rounded to same decimal place
# ----------------------------
def round_value_uncertainty(value: float, uncertainty: float):
    if uncertainty <= 0 or not np.isfinite(uncertainty):
        return str(value), str(uncertainty)

    exponent = int(np.floor(np.log10(abs(uncertainty))))
    mantissa = abs(uncertainty) / (10 ** exponent)

    sig_figs = 2 if mantissa < 3 else 1
    decimals = -exponent + (sig_figs - 1)

    value_rounded = round(value, decimals)
    uncertainty_rounded = round(uncertainty, decimals)

    value_str = f"{value_rounded:.{decimals}f}"
    uncertainty_str = f"{uncertainty_rounded:.{decimals}f}"
    return value_str, uncertainty_str

# ----------------------------
# Initial guesses
# ----------------------------
idx0 = np.argmax(G)
f0_guess = f_khz[idx0]
y0_guess = np.min(G)
A_guess = np.max(G) - y0_guess
gamma_guess = 5.0  # kHz

p0 = [A_guess, f0_guess, gamma_guess, y0_guess]
```

```
bounds = (
    [0.0, f_khz.min(), 1e-6, -np.inf],
    [np.inf, f_khz.max(), np.inf,  np.inf]
)

params, cov = curve_fit(
    lorentzian,
    f_khz,
    G,
    p0=p0,
    sigma=G_err,
    absolute_sigma=True,
    bounds=bounds,
    maxfev=20000
)

A_fit, f0_fit, gamma_fit, y0_fit = params
perr = np.sqrt(np.diag(cov))

# ------------------------------
# Derived quantities + propagated errors
# ------------------------------
# FWHM = 2*gamma
FWHM_khz = 2.0 * gamma_fit
FWHM_err = 2.0 * perr[2]

# Q = f0 / FWHM = f0 / (2*gamma)
Q = f0_fit / FWHM_khz

# full covariance propagation for Q(f0, gamma)
dQ_df0 = 1.0 / (2.0 * gamma_fit)
dQ_dgamma = -f0_fit / (2.0 * gamma_fit**2)

var_Q = (
    dQ_df0**2 * cov[1, 1]
    + dQ_dgamma**2 * cov[2, 2]
    + 2.0 * dQ_df0 * dQ_dgamma * cov[1, 2]
)
Q_err = np.sqrt(var_Q)

```

```
# G_peak = y0 + A
G_peak = y0_fit + A_fit
var_G_peak = cov[0, 0] + cov[3, 3] + 2.0 * cov[0, 3]
G_peak_err = np.sqrt(var_G_peak)

# G_peak in dB
G_peak_dB = 20.0 * np.log10(G_peak)
G_peak_dB_err = (20.0 / np.log(10.0)) * (G_peak_err / G_peak)

# ------------------------------
# Rounded strings for report
# ------------------------------
A_str, A_err_str = round_value_uncertainty(A_fit, perr[0])
f0_str, f0_err_str = round_value_uncertainty(f0_fit, perr[1])
gamma_str, gamma_err_str = round_value_uncertainty(gamma_fit, perr[2])
y0_str, y0_err_str = round_value_uncertainty(y0_fit, perr[3])

FWHM_str, FWHM_err_str = round_value_uncertainty(FWHM_khz, FWHM_err)
Q_str, Q_err_str = round_value_uncertainty(Q, Q_err)
Gpeak_str, Gpeak_err_str = round_value_uncertainty(G_peak, G_peak_err)
Gpeak_dB_str, Gpeak_dB_err_str = round_value_uncertainty(G_peak_dB,
    G_peak_dB_err)

# ------------------------------
# Print output
# ------------------------------
print("Raw fit parameters:")
print(f"A      = {A_fit:.6f} pm {perr[0]:.6f}")
print(f"f0     = {f0_fit:.6f} pm {perr[1]:.6f} kHz")
print(f"gamma  = {gamma_fit:.6f} pm {perr[2]:.6f} kHz")
print(f"y0     = {y0_fit:.6f} pm {perr[3]:.6f}")

print("\nRaw derived quantities:")
print(f"FWHM   = {FWHM_khz:.6f} pm {FWHM_err:.6f} kHz")
print(f"Q      = {Q:.6f} pm {Q_err:.6f}")
print(f"G_max  = {G_peak:.6f} pm {G_peak_err:.6f}")
print(f"G_max  = {G_peak_dB:.6f} pm {G_peak_dB_err:.6f} dB")

print("\nReport-ready rounded results:")
print(f"f0     = ({f0_str} pm {f0_err_str}) kHz")
print(f"gamma  = ({gamma_str} pm {gamma_err_str}) kHz")
```

```
print(f"FWHM   = ({FWHM_str} pm {FWHM_err_str}) kHz")
print(f"Q      = {Q_str} pm {Q_err_str}")
print(f"G_max  = {Gpeak_str} pm {Gpeak_err_str}")
print(f"G_max  = ({Gpeak_dB_str} pm {Gpeak_dB_err_str}) dB")

# ----------------------------
# Smooth curve for plotting
# ----------------------------
f_smooth = np.linspace(f_khz.min(), f_khz.max(), 2000)
G_fit = lorentzian(f_smooth, *params)
G_fit_dB = 20.0 * np.log10(G_fit)

# ----------------------------
# Plot
# ----------------------------
plt.figure(figsize=(8.5, 5.5))

plt.errorbar(
    f_khz,
    G_dB,
    yerr=G_dB_err,
    fmt='o',
    markersize=5,
    markerfacecolor='white',
    markeredgewidth=1.0,
    capsize=3,
    elinewidth=1.0,
    barsabove=True,
    zorder=3,
    label=r"Measured data ($\sigma_{V_{\mathrm{out}}}=0.010\,\mathrm{V}$)"
)

plt.plot(
    f_smooth,
    G_fit_dB,
    linewidth=2,
    color="tab:orange",
    label="Lorentzian fit"
)

plt.axvline(
```

```
    f0_fit,
    linestyle="--",
    linewidth=1.2,
    color="tab:blue",
    label=rf"$f_0 = ({f0_str}\pm {f0_err_str})\,\mathrm{{kHz}}$"
)

summary_text = (
    rf"$f_0 = ({f0_str}\pm {f0_err_str})\,\mathrm{{kHz}}$" "\n"
    rf"$\gamma = ({gamma_str}\pm {gamma_err_str})\,\mathrm{{kHz}}$" "\n"
    rf"$\mathrm{{FWHM}} = ({FWHM_str}\pm {FWHM_err_str})\,\mathrm{{kHz}}$" "\n"
    rf"$Q = {Q_str}\pm {Q_err_str}$" "\n"
    rf"$G_{{\max}} = {Gpeak_str}\pm {Gpeak_err_str}$" "\n"
    rf"$20\log_{{10}}(G_{{\max}}) = ({Gpeak_dB_str}\pm
   {Gpeak_dB_err_str})\,\mathrm{{dB}}$"
)

# Summary box: upper left
plt.text(
    0.03, 0.97, summary_text,
    transform=plt.gca().transAxes,
    fontsize=9,
    verticalalignment='top',
    bbox=dict(boxstyle="round", facecolor="white", alpha=0.9)
)

plt.xlabel("Frequency (kHz)")
plt.ylabel(r"Gain $20\log_{10}(G)$ (dB)")
plt.title("Band-pass Filter Gain (dB) with Lorentzian Fit")
plt.grid(True, alpha=0.3)

# Legend: lower left
plt.legend(
    fontsize=9,
    loc="lower left",
    framealpha=0.9
)

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

```
# ========================
# 5.3b) Determination of the Modulation Depth: X-Y Plot
# ========================

# Measured values from the X-Y plot
A_max = 1315.0
A_min = 318.0

# Choose your reading uncertainties in px
sigma_A_max = 5.0
sigma_A_min = 5.0

# Modulation depth
m = (A_max - A_min) / (A_max + A_min)

# Error propagation
sigma_m = (2.0 / (A_max + A_min)**2) * np.sqrt(
    (A_min * sigma_A_max)**2 + (A_max * sigma_A_min)**2
)

print(f"m = {m:.6f} pm {sigma_m:.6f}")



# ---------------------------------
# 5.3 Modulation depth from time-domain envelope (KO method)
# Using full vertical heights H_max and H_min measured in pixels
# ---------------------------------

# Measured pixel heights
H_min = np.array([179, 177, 182], dtype=float)
H_max = np.array([1135, 1136, 1130], dtype=float)

# Mean values
H_min_mean = np.mean(H_min)
H_max_mean = np.mean(H_max)

# Standard error of the mean (from 3 measurements)
```

```
H_min_err = np.std(H_min, ddof=1) / np.sqrt(len(H_min))
H_max_err = np.std(H_max, ddof=1) / np.sqrt(len(H_max))

# Modulation depth:
# m = (H_max - H_min) / (H_max + H_min)
m = (H_max_mean - H_min_mean) / (H_max_mean + H_min_mean)

# Error propagation
sigma_m = (2.0 / (H_max_mean + H_min_mean)**2) * np.sqrt(
    (H_min_mean * H_max_err)**2 + (H_max_mean * H_min_err)**2
)

print("Measured values:")
print(f"H_min = {H_min_mean:.2f} pm {H_min_err:.2f} px")
print(f"H_max = {H_max_mean:.2f} pm {H_max_err:.2f} px")

print("\nModulation depth from KO/time-domain method:")
print(f"m = {m:.4f} pm {sigma_m:.4f}")



# ============================================================
# 5.3 Modulation depth from FFT spectrum
# Semi-manual peak selection:
# click approximately on LSB, Carrier, USB
# ============================================================

csv_path = r"C:\Users\L-Vis\OneDrive\Desktop\P3 Lab\Harmonic Analysis\USB
    Stick\5.3 total\5.3b Sinusoidal\F0002FFT.CSV"

# ------------------------------------------------------------
# Robust CSV loader for Tektronix-like FFT exports
# ------------------------------------------------------------
def load_fft_csv(path):
    # first try normal read
    for kwargs in [
        dict(sep=",", engine="python"),
        dict(sep=";", engine="python"),
        dict(sep=None, engine="python"),
    ]:
        try:
```

```
            df = pd.read_csv(path, **kwargs)
            if df.shape[1] >= 2:
                break
        except Exception:
            continue
    else:
        raise RuntimeError("Could not read CSV file.")

    # convert all columns to numeric if possible
    df_num = df.copy()
    for col in df_num.columns:
        df_num[col] = pd.to_numeric(df_num[col], errors="coerce")

    valid_cols = [c for c in df_num.columns if df_num[c].notna().sum() > 5]

    # fallback if headers are weird
    if len(valid_cols) < 2:
        raw = pd.read_csv(path, header=None, sep=None, engine="python")
        for col in raw.columns:
            raw[col] = pd.to_numeric(raw[col], errors="coerce")
        valid_cols = [c for c in raw.columns if raw[c].notna().sum() > 5]
        if len(valid_cols) < 2:
            raise RuntimeError("Could not identify two numeric columns.")
        df_num = raw

    # choose frequency column as the one that is mostly increasing
    freq_col = None
    for c in valid_cols:
        vals = df_num[c].dropna().to_numpy()
        if len(vals) > 10:
            frac_inc = np.mean(np.diff(vals) >= 0)
            if frac_inc > 0.7:
                freq_col = c
                break

    if freq_col is None:
        # choose numeric column with most unique values
        freq_col = max(valid_cols, key=lambda c: df_num[c].dropna().nunique())

    amp_candidates = [c for c in valid_cols if c != freq_col]
    if not amp_candidates:
```

```
        raise RuntimeError("Could not identify amplitude column.")

    amp_col = max(amp_candidates, key=lambda c: df_num[c].dropna().nunique())

    out = pd.DataFrame({
        "frequency": pd.to_numeric(df_num[freq_col], errors="coerce"),
        "amplitude": pd.to_numeric(df_num[amp_col], errors="coerce"),
    }).dropna()

    out = out.sort_values("frequency").reset_index(drop=True)

    print("Detected columns:")
    print(f"  Frequency column: {freq_col}")
    print(f"  Amplitude column: {amp_col}")
    print(f"  Number of valid rows: {len(out)}")

    return out

# ------------------------------------------------------------
# Snap click to strongest local maximum within a search window
# ------------------------------------------------------------
def snap_to_local_max(f, y, x_click, window_hz=20.0):
    mask = (f >= x_click - window_hz) & (f <= x_click + window_hz)
    if not np.any(mask):
        raise RuntimeError(f"No data in search window around x =
    {x_click:.3f}")

    f_local = f[mask]
    y_local = y[mask]

    idx_local = np.argmax(y_local)
    f_peak = f_local[idx_local]
    y_peak = y_local[idx_local]

    # map back to global index
    global_indices = np.where(mask)[0]
    idx_global = global_indices[idx_local]

    return idx_global, f_peak, y_peak

# ------------------------------------------------------------
```

```
# Load data
# ------------------------------------------------------------
df = load_fft_csv(csv_path)
f = df["frequency"].to_numpy(dtype=float)
amp_db = df["amplitude"].to_numpy(dtype=float)

# convert dB -> linear amplitude for m calculation
amp_lin = 10 ** (amp_db / 20.0)

# ------------------------------------------------------------
# First plot: user clicks 3 approximate peaks
# ------------------------------------------------------------
plt.figure(figsize=(10, 5.8))
plt.plot(f, amp_db, linewidth=1.2)
plt.xlabel("Frequency")
plt.ylabel("Amplitude (dB)")
plt.title("Click approximately on: Lower sideband, Carrier, Upper sideband")
plt.grid(True, alpha=0.3)
plt.tight_layout()

print("\nPlease click on 3 peaks in this order:")
print("  1) lower sideband")
print("  2) carrier")
print("  3) upper sideband")

clicked = plt.ginput(3, timeout=0)
plt.close()

if len(clicked) != 3:
    raise RuntimeError("You must click exactly 3 peaks.")

# search window around each click in frequency units
# adjust if needed
search_window_hz = 20.0

# snap to nearest local maximum in dB trace
idx_lsb, f_lsb, A_lsb_db = snap_to_local_max(f, amp_db, clicked[0][0],
    window_hz=search_window_hz)
idx_car, f_c,   A_c_db   = snap_to_local_max(f, amp_db, clicked[1][0],
    window_hz=search_window_hz)
```

```
idx_usb, f_usb, A_usb_db = snap_to_local_max(f, amp_db, clicked[2][0],
    window_hz=search_window_hz)

A_lsb = amp_lin[idx_lsb]
A_c   = amp_lin[idx_car]
A_usb = amp_lin[idx_usb]

# ------------------------------------------------------------
# Modulation depth from spectrum
# A_LSB = A_USB = (m/2) A_C
# => m = (A_LSB + A_USB) / A_C
# ------------------------------------------------------------
m_left = 2.0 * A_lsb / A_c
m_right = 2.0 * A_usb / A_c
m_avg = (A_lsb + A_usb) / A_c

# simple uncertainty estimate from sideband asymmetry
m_err = abs(m_left - m_right) / 2.0

# ------------------------------------------------------------
# Print results
# ------------------------------------------------------------
print("\nSelected peaks:")
print(f"  Lower SB: f = {f_lsb:.6g}, amplitude = {A_lsb_db:.4f} dB")
print(f"  Carrier : f = {f_c:.6g}, amplitude = {A_c_db:.4f} dB")
print(f"  Upper SB: f = {f_usb:.6g}, amplitude = {A_usb_db:.4f} dB")

print("\nConverted linear amplitudes:")
print(f"  A_LSB = {A_lsb:.6g}")
print(f"  A_C   = {A_c:.6g}")
print(f"  A_USB = {A_usb:.6g}")

print("\nModulation depth from spectrum:")
print(f"  m_left  = {m_left:.4f}")
print(f"  m_right = {m_right:.4f}")
print(f"  m_avg   = {m_avg:.4f} pm {m_err:.4f}")

# ------------------------------------------------------------
# Final plot with selected peaks marked
# ------------------------------------------------------------
plt.figure(figsize=(10, 5.8))
```

```
plt.plot(f, amp_db, linewidth=1.2, label="FFT spectrum")

plt.plot(f_lsb, A_lsb_db, "o", markersize=8, label="Lower sideband")
plt.plot(f_c,   A_c_db,   "o", markersize=8, label="Carrier")
plt.plot(f_usb, A_usb_db, "o", markersize=8, label="Upper sideband")

plt.axvline(f_lsb, linestyle="--", linewidth=1.0)
plt.axvline(f_c, linestyle="--", linewidth=1.0)
plt.axvline(f_usb, linestyle="--", linewidth=1.0)

summary = (
    f"$f_{{LSB}} = {f_lsb:.3g}$\n"
    f"$f_C = {f_c:.3g}$\n"
    f"$f_{{USB}} = {f_usb:.3g}$\n"
    f"$m = {m_avg:.4f} \\pm {m_err:.4f}$"
)

plt.text(
    0.03, 0.97, summary,
    transform=plt.gca().transAxes,
    fontsize=10,
    verticalalignment="top",
    bbox=dict(boxstyle="round", facecolor="white", alpha=0.9)
)

plt.xlabel("Frequency")
plt.ylabel("Amplitude (dB)")
plt.title("Modulation depth from FFT spectrum")
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()



# =========================
# 5.4 Finding Fourier Spectra
# =========================
FOLDER = r"C:\Users\L-Vis\OneDrive\Desktop\P3 Lab\Harmonic Analysis\Harmonic
    Analysis Fourier Spectra"
PATTERN = "*FFT.CSV"          # SinusoidalFFT.CSV, TriangularFFT.CSV, ...
```

```
928 SHOW_PLOTS = True
929 SAVE_PLOTS = True
930
931 # Zoom frs Plotten
932 XMAX_HZ = 200_000
933 N_PEAKS = 10
934
935
936 # =========================
937 # Reader for TEK FFT CSV
938 # =========================
939 def read_tek_fft_csv(path: str) -> tuple[np.ndarray, np.ndarray]:
940     """
941     Reads Tektronix FFT export CSV.
942     Format:
943       col 3 = Frequency [Hz]
944       col 4 = Amplitude [dB]
945     Separator is ';'
946     """
947     df = pd.read_csv(path, sep=";", header=None, engine="python")
948
949     freq = pd.to_numeric(df.iloc[:, 3], errors="coerce")
950     amp  = pd.to_numeric(df.iloc[:, 4].astype(str).str.replace(",", "",
    regex=False),
951                          errors="coerce")
952
953     mask = freq.notna() & amp.notna()
954     freq = freq[mask].to_numpy(dtype=float)
955     amp  = amp[mask].to_numpy(dtype=float)
956
957     idx = np.argsort(freq)
958     return freq[idx], amp[idx]
959
960
961 # =========================
962 # Simple peak finder
963 # =========================
964 def simple_peak_list(freq: np.ndarray, amp_db: np.ndarray, n_peaks: int = 10,
965                      min_freq: float = 1.0, guard_hz: float = 500.0):
966     """
967     Very simple peak finder:
```

```
    - looks for local maxima
    - picks top n_peaks by amplitude
    - uses a guard band so you dont get many peaks from same bump
    """
    freq = np.asarray(freq)
    amp_db = np.asarray(amp_db)

    mask = (freq >= min_freq)
    f = freq[mask]
    a = amp_db[mask]

    if len(f) < 5:
        return []

    loc = np.where((a[1:-1] > a[:-2]) & (a[1:-1] > a[2:]))[0] + 1
    if len(loc) == 0:
        return []

    cand = loc[np.argsort(a[loc])[::-1]]

    chosen = []
    for i in cand:
        fi = f[i]
        ai = a[i]
        if all(abs(fi - fj) > guard_hz for fj, _ in chosen):
            chosen.append((fi, ai))
        if len(chosen) >= n_peaks:
            break

    return chosen


# =========================
# Error estimation helpers
# =========================
def estimate_fft_binwidth(freq: np.ndarray) -> float:
    """Median FFT frequency spacing."""
    return float(np.median(np.diff(freq)))


def characterize_peak(freq: np.ndarray, amp_db: np.ndarray, f_guess: float):
```

```
    """
    Characterize a peak near frequency f_guess.

    Returns:
        f_peak, A_peak, sigma_f, sigma_A, idx
    where
        sigma_f ~ half FFT bin width
        sigma_A ~ half max difference to neighboring bins
    """
    idx = int(np.argmin(np.abs(freq - f_guess)))

    f_peak = float(freq[idx])
    A_peak = float(amp_db[idx])

    # frequency uncertainty from FFT bin width
    df = estimate_fft_binwidth(freq)
    sigma_f = df / 2.0

    # amplitude uncertainty from local bin-to-bin variation
    diffs = []
    if idx > 0:
        diffs.append(abs(amp_db[idx] - amp_db[idx - 1]))
    if idx < len(amp_db) - 1:
        diffs.append(abs(amp_db[idx] - amp_db[idx + 1]))

    sigma_A = max(diffs) / 2.0 if diffs else np.nan

    return f_peak, A_peak, sigma_f, sigma_A, idx


def round_value_uncertainty(value: float, uncertainty: float):
    """
    Round value and uncertainty:
    - uncertainty: 1 sig fig usually
    - 2 sig figs if leading digit is 1 or 2
    - value rounded to same decimal place
    """
    if uncertainty <= 0 or not np.isfinite(uncertainty):
        return str(value), str(uncertainty)

    exponent = int(np.floor(np.log10(abs(uncertainty))))
```

```
    mantissa = abs(uncertainty) / (10 ** exponent)

    sig_figs = 2 if mantissa < 3 else 1
    decimals = -exponent + (sig_figs - 1)

    value_rounded = round(value, decimals)
    uncertainty_rounded = round(uncertainty, decimals)

    value_str = f"{value_rounded:.{decimals}f}"
    uncertainty_str = f"{uncertainty_rounded:.{decimals}f}"
    return value_str, uncertainty_str


# =========================
# Main analysis loop
# =========================
for path in glob.glob(os.path.join(FOLDER, PATTERN)):
    name = os.path.basename(path)

    freq, amp_db = read_tek_fft_csv(path)

    # Find candidate peaks
    peaks = simple_peak_list(
        freq,
        amp_db,
        n_peaks=N_PEAKS,
        min_freq=1.0,
        guard_hz=500.0
    )

    if len(peaks) == 0:
        print(f"\n{name}: no peaks found.")
        continue

    # Fundamental = strongest non-DC peak
    f1_guess, A1_guess = max(peaks, key=lambda x: x[1])

    # Third harmonic = candidate closest to 3*f1
    peaks_no_f1 = [p for p in peaks if abs(p[0] - f1_guess) > 100.0]
    if len(peaks_no_f1) > 0:
```

```
        f3_guess, A3_guess = min(peaks_no_f1, key=lambda x: abs(x[0] - 3.0 * f1_guess))
        f3, A3, sf3, sA3, idx3 = characterize_peak(freq, amp_db, f3_guess)
    else:
        f3 = A3 = sf3 = sA3 = np.nan
        idx3 = None

    # Refine fundamental
    f1, A1, sf1, sA1, idx1 = characterize_peak(freq, amp_db, f1_guess)

    # Rounded strings
    f1_khz_str, sf1_khz_str = round_value_uncertainty(f1 / 1000.0, sf1 / 1000.0)
    A1_str, sA1_str = round_value_uncertainty(A1, sA1)

    if np.isfinite(f3):
        f3_khz_str, sf3_khz_str = round_value_uncertainty(f3 / 1000.0, sf3 / 1000.0)
        A3_str, sA3_str = round_value_uncertainty(A3, sA3)

    # ==========================
    # Console output
    # ==========================
    print("\n" + "=" * 70)
    print(f"File: {name}")
    print("=" * 70)

    print("\nRaw peak values:")
    print(f"f1 = {f1:.3f} pm {sf1:.3f} Hz")
    print(f"A1 = {A1:.3f} pm {sA1:.3f} dB")

    if np.isfinite(f3):
        print(f"f3 = {f3:.3f} pm {sf3:.3f} Hz")
        print(f"A3 = {A3:.3f} pm {sA3:.3f} dB")

    print("\nReport-ready rounded values:")
    print(f"f1 = ({f1_khz_str} pm {sf1_khz_str}) kHz")
    print(f"A1 = ({A1_str} pm {sA1_str}) dB")

    if np.isfinite(f3):
        print(f"f3 = ({f3_khz_str} pm {sf3_khz_str}) kHz")
```

```
        print(f"A3 = ({A3_str} pm {sA3_str}) dB")
        print(f"Check: f3 / f1 = {f3/f1:.3f}")

    # =========================
    # Plot (unchanged style)
    # =========================
    if SHOW_PLOTS or SAVE_PLOTS:
        plt.figure(figsize=(9, 4.8))
        plt.plot(freq, amp_db, linewidth=0.8)
        plt.xlim(0, XMAX_HZ)
        plt.xlabel("Frequency [Hz]")
        plt.ylabel("Amplitude [dB]")
        plt.title(f"FFT spectrum (exported) - {name}")
        plt.grid(True, alpha=0.3)

        if idx1 is not None:
            plt.plot(freq[idx1], amp_db[idx1], "o", markersize=5)

        if idx3 is not None:
            plt.plot(freq[idx3], amp_db[idx3], "o", markersize=5)

        plt.tight_layout()

        if SAVE_PLOTS:
            out_png = os.path.splitext(path)[0] + ".png"
            plt.savefig(out_png, dpi=300)

        if SHOW_PLOTS:
            plt.show()
        else:
            plt.close()



# =========================
# Main
# =========================
files = sorted(glob.glob(os.path.join(FOLDER, PATTERN)))
if not files:
    raise FileNotFoundError(f"No files matching {PATTERN} in {FOLDER}")

```

```
for path in files:
    name = os.path.basename(path)
    freq, amp_db = read_tek_fft_csv(path)

    # Peaks (for report table)
    peaks = simple_peak_list(freq, amp_db, n_peaks=N_PEAKS)

    print("\n" + "=" * 70)
    print("File:", name)
    print(f"Freq range: {freq.min():.1f} .. {freq.max():.1f} Hz")
    print("Top peaks (freq [Hz], amp [dB]):")
    for f0, a0 in peaks:
        print(f"  {f0:12.2f}  {a0:8.2f}")

    # Plot
    plt.figure(figsize=(10, 5))
    plt.plot(freq, amp_db, linewidth=1)
    plt.xlabel("Frequency [Hz]")
    plt.ylabel("Amplitude [dB]")
    plt.title(f"FFT spectrum (exported) - {name}")
    plt.grid(True)
    plt.xlim(0, XMAX_HZ)
    plt.tight_layout()

    if SAVE_PLOTS:
        out_png = os.path.splitext(path)[0] + "_clean.png"
        plt.savefig(out_png, dpi=200)
        print("Saved plot:", out_png)

    if SHOW_PLOTS:
        plt.show()
    else:
        plt.close()
```