# Frequency Analysis Lab Report

September 8, 2025

## 1 Frequency Analysis Lab Report (ELA24)

**Author:** Your Name
**Lab date:** YYYY-MM-DD
**Report date:** YYYY-MM-DD

### 1.1 Abstract

This lab explores Fourier analysis of periodic signals using an oscilloscope FFT function and Python. We compare measured harmonics of sawtooth, triangle, and sine waves with theoretical Fourier series coefficients, and investigate the effects of an RC low-pass filter in both time and frequency domains.

### 1.2 1. Introduction

The purpose of this lab is to understand how signals can be represented in both the time and frequency domains. Using Fourier series, we can predict harmonic content of periodic signals such as sawtooth and triangle waves. The oscilloscope FFT function provides experimental spectra that we compare with theory. We also study filtering effects using a simple RC low-pass filter.

**Key formulas** - Sawtooth (Vpeak harmonics): $A_n = \frac{2A}{\pi n}$ - Triangle (odd $n$ only, Vpeak): $A_n = \frac{8A}{\pi^2 n^2}$ - RC low-pass: $|H(f)| = \frac{1}{\sqrt{1+(f/f_c)^2}}$, $f_c = \frac{1}{2\pi RC}$ - Convert Vpeak $\rightarrow$ dBV: $V_{\text{rms}} = \frac{V_{\text{peak}}}{\sqrt{2}}$, dBV $= 20\log_{10}(V_{\text{rms}})$

### 1.3 2. Experiment

#### 1.3.1 Setup

- Signal generator: 1 kHz, 2.5 Vpp, 0 V DC offset.
- Oscilloscope: Keysight DSOX1204A (FFT via **Math $\rightarrow$ FFT**; screenshots or CSV export via BenchVue).
- Signals analyzed: Sawtooth, Triangle, Sine.
- Filter: RC low-pass, R = 8.2 kΩ, C = 10 nF ((f_c  1.94) kHz).

#### 1.3.2 Procedure

1. Measure first 10 harmonics (1–10 kHz) of the **sawtooth** with FFT.
2. Repeat for **triangle** (expect odd harmonics).
3. Repeat for **sine** (expect only the fundamental).

4. Insert **RC filter** on the sawtooth, measure first 10 harmonics at the output; capture input simultaneously when possible.
5. Save scope screenshots **and/or** export **CSV time series** for Python FFT.
6. Compare theory vs. measurements and discuss discrepancies.

[19]:
```python
# 3. Imports & global settings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['figure.dpi'] = 120

# Lab defaults
F0 = 1000.0      # fundamental frequency [Hz]
VPP = 2.5        # generator setting [Vpp]
A = VPP/2        # peak amplitude used in Fourier formulas [V]
N_HARM = 10      # number of harmonics
R, C = 8200.0, 10e-9
FC = 1.0/(2*np.pi*R*C)

print(f"fc (RC)   {FC:.2f} Hz")
```

```
fc (RC)   1940.91 Hz
```

### 1.4 3. Theory – Fourier series (sawtooth, triangle, sine)

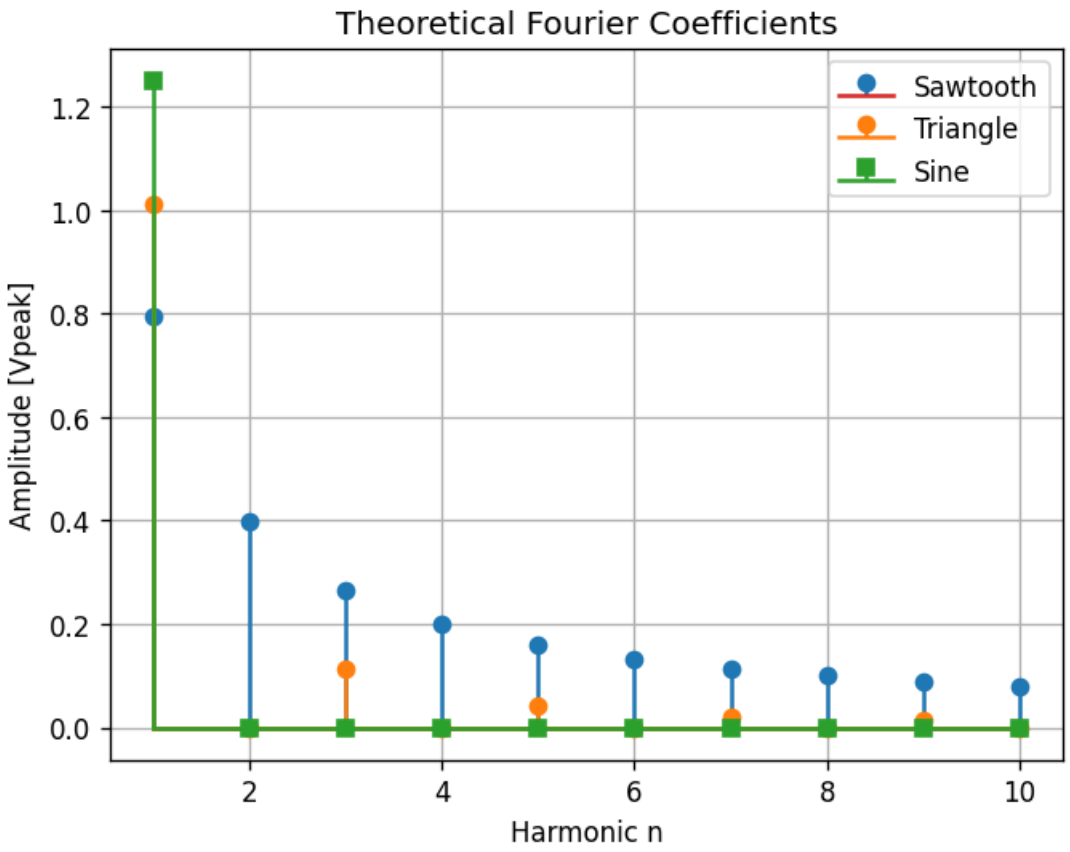
[20]:
```python
# Theoretical harmonic amplitudes (Vpeak) and conversion to dBV
def sawtooth_vpeak(n, A=A):
    return 2*A/(np.pi*n)

def triangle_vpeak(n, A=A):
    return 0.0 if (n % 2 == 0) else 8*A/(np.pi**2 * n**2)

def vpeak_to_dbv(vp):
    vrms = vp/np.sqrt(2)
    return 20*np.log10(max(vrms, 1e-12))

n = np.arange(1, N_HARM+1)
saw_Vp = np.array([sawtooth_vpeak(k) for k in n])
tri_Vp = np.array([triangle_vpeak(k) for k in n])
sin_Vp = np.array([A] + [0]*(N_HARM-1))  # sine: only fundamental

saw_dBV = np.array([vpeak_to_dbv(v) for v in saw_Vp])
tri_dBV = np.array([vpeak_to_dbv(v) for v in tri_Vp])
sin_dBV = np.array([vpeak_to_dbv(v) for v in sin_Vp])

df_theory = pd.DataFrame({
    'n': n,
```

```
    'freq_Hz': n*F0,
    'saw_Vpeak': saw_Vp,
    'saw_dBV': saw_dBV,
    'tri_Vpeak': tri_Vp,
    'tri_dBV': tri_dBV,
    'sine_Vpeak': sin_Vp,
    'sine_dBV': sin_dBV
})
df_theory
```

[20]:

| | n | freq_Hz | saw_Vpeak | saw_dBV | tri_Vpeak | tri_dBV | sine_Vpeak | sine_dBV |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1000.0 | 0.795775 | -4.994497 | 1.013212 | -2.896295 | 1.25 | -1.0721 |
| 1 | 2 | 2000.0 | 0.397887 | -11.015097 | 0.000000 | -240.000000 | 0.00 | -240.0000 |
| 2 | 3 | 3000.0 | 0.265258 | -14.536922 | 0.112579 | -21.981145 | 0.00 | -240.0000 |
| 3 | 4 | 4000.0 | 0.198944 | -17.035697 | 0.000000 | -240.000000 | 0.00 | -240.0000 |
| 4 | 5 | 5000.0 | 0.159155 | -18.973897 | 0.040528 | -30.855095 | 0.00 | -240.0000 |
| 5 | 6 | 6000.0 | 0.132629 | -20.557522 | 0.000000 | -240.000000 | 0.00 | -240.0000 |
| 6 | 7 | 7000.0 | 0.113682 | -21.896458 | 0.020678 | -36.700216 | 0.00 | -240.0000 |
| 7 | 8 | 8000.0 | 0.099472 | -23.056297 | 0.000000 | -240.000000 | 0.00 | -240.0000 |
| 8 | 9 | 9000.0 | 0.088419 | -24.079347 | 0.012509 | -41.065995 | 0.00 | -240.0000 |
| 9 | 10 | 10000.0 | 0.079577 | -24.994497 | 0.000000 | -240.000000 | 0.00 | -240.0000 |

[21]:
```
# Plot theoretical amplitudes (Vpeak)
plt.figure()
plt.stem(n, saw_Vp, label='Sawtooth')
plt.stem(n, tri_Vp, linefmt='C1-', markerfmt='C1o', basefmt='C1-',␣
 ↪label='Triangle')
plt.stem(n, sin_Vp, linefmt='C2-', markerfmt='C2s', basefmt='C2-', label='Sine')
plt.xlabel('Harmonic n')
plt.ylabel('Amplitude [Vpeak]')
plt.title('Theoretical Fourier Coefficients')
plt.legend(); plt.grid(True); plt.show()
```




### 1.5 4. Theory – RC low-pass filter (j method)

[22]:
```
def rc_mag_db(f, fc=FC):
    H = 1.0/np.sqrt(1.0 + (f/fc)**2)
    return 20*np.log10(np.maximum(H, 1e-15))

f_harm = n*F0
H_dB = rc_mag_db(f_harm)
df_rc = pd.DataFrame({'n': n, 'freq_Hz': f_harm, 'RC_atten_dB': H_dB})
df_rc
```

[22]:

| | n | freq_Hz | RC_atten_dB |
|---|---|---|---|
| 0 | 1 | 1000.0 | -1.022460 |
| 1 | 2 | 2000.0 | -3.142490 |
| 2 | 3 | 3000.0 | -5.300813 |
| 3 | 4 | 4000.0 | -7.199314 |
| 4 | 5 | 5000.0 | -8.828842 |
| 5 | 6 | 6000.0 | -10.235119 |
| 6 | 7 | 7000.0 | -11.463511 |

```
7   8   8000.0   -12.550066
8   9   9000.0   -13.522150
9  10  10000.0   -14.400473
```

### 1.6 5. Import oscilloscope CSV (time series) & FFT (Hann, dBV)

Export your time-domain data for **input** (unfiltered) and **output** (after RC) channels as semicolon-separated CSVs (BenchVue default).

[23]:
```python
def _to_num(s):
    if isinstance(s, str):
        s = s.replace(',', '.')
    try:
        return float(s)
    except:
        return np.nan

def read_keysight_csv(path):
    raw = pd.read_csv(path, sep=';', header=None, dtype=str, engine='python')
    num = raw.applymap(_to_num)
    valid = num.notna().sum()
    cols = valid[valid > 0].index[-2:]  # last two populated columns → time, voltage
    t = num.iloc[:, cols[0]].dropna().to_numpy()
    v = num.iloc[:, cols[1]].dropna().to_numpy()
    n = min(len(t), len(v))
    return t[:n], v[:n]

def hann_fft_dbv(t, v):
    v = v - np.mean(v)
    dt = np.median(np.diff(t))
    fs = 1.0/dt
    N = len(v)
    w = np.hanning(N)
    U = w.sum()/N  # coherent gain
    X = np.fft.rfft(v*w)
    freqs = np.fft.rfftfreq(N, d=dt)
    mag_Vpeak = np.abs(X)/(N*U)
    if len(mag_Vpeak) > 2:
        mag_Vpeak[1:-1] *= 2.0
    mag_Vrms = mag_Vpeak/np.sqrt(2)
    mag_dBV = 20*np.log10(np.maximum(mag_Vrms, 1e-12))
    return freqs, mag_dBV, fs

def plot_fft(freqs, mag_dBV, fs, title, xlim=None):
    plt.figure()
    plt.plot(freqs, mag_dBV)
```

```
    plt.xlabel('Frequency [Hz]')
    plt.ylabel('Amplitude [dBV]')
    plt.title(title)
    plt.grid(True)
    if xlim:
        plt.xlim(*xlim)
    else:
        plt.xlim(0, fs/2)
    plt.show()

# Filenames to edit to your actual exports
IN_CSV  = 'in_signal.csv'   # unfiltered channel
OUT_CSV = 'out_signal.csv'  # after RC filter
SIN_CSV = 'sine_signal.csv' # optional sine time series
TRI_CSV = 'triangle_signal.csv' # optional triangle time series
SAW_CSV = 'sawtooth_signal.csv' # optional sawtooth time series

def safe_load(path):
    try:
        t, v = read_keysight_csv(path)
        print(f"Loaded {path}: {len(t)} samples, Δt {np.median(np.diff(t)):.
 ↪3e}s")
        return t, v
    except Exception as e:
        print(f"Could not read {path}: {e}")
        return None, None

t_in, v_in   = safe_load(IN_CSV)
t_out, v_out = safe_load(OUT_CSV)
t_sin, v_sin = safe_load(SIN_CSV)
t_tri, v_tri = safe_load(TRI_CSV)
t_saw, v_saw = safe_load(SAW_CSV)
```

```
Could not read in_signal.csv: [Errno 2] No such file or directory:
'in_signal.csv'
Could not read out_signal.csv: [Errno 2] No such file or directory:
'out_signal.csv'
Could not read sine_signal.csv: [Errno 2] No such file or directory:
'sine_signal.csv'
Could not read triangle_signal.csv: [Errno 2] No such file or directory:
'triangle_signal.csv'
Could not read sawtooth_signal.csv: [Errno 2] No such file or directory:
'sawtooth_signal.csv'
```

## 1.7 6. FFT plots (measurement data)

[24]:
```
if t_saw is not None:
    f_saw, dbv_saw, fs = hann_fft_dbv(t_saw, v_saw)
    plot_fft(f_saw, dbv_saw, fs, 'FFT - Sawtooth (input)')
if t_tri is not None:
    f_tri, dbv_tri, fs = hann_fft_dbv(t_tri, v_tri)
    plot_fft(f_tri, dbv_tri, fs, 'FFT - Triangle (input)')
if t_sin is not None:
    f_sine, dbv_sine, fs = hann_fft_dbv(t_sin, v_sin)
    plot_fft(f_sine, dbv_sine, fs, 'FFT - Sine (input)')
if t_in is not None:
    f_in, dbv_in, fs_in = hann_fft_dbv(t_in, v_in)
    plot_fft(f_in, dbv_in, fs_in, 'FFT - Input (unfiltered)')
if t_out is not None:
    f_out, dbv_out, fs_out = hann_fft_dbv(t_out, v_out)
    plot_fft(f_out, dbv_out, fs_out, 'FFT - Output (after RC)')
```

## 1.8 7. Extract harmonic lines and compare to theory

This cell finds the maximum amplitude within ±2% around each harmonic n · F0.

[25]:
```
def pick_harmonics(freqs, mag_dbv, f0=F0, n_max=N_HARM, rel_bw=0.02):
    rows = []
    for k in range(1, n_max+1):
        ft = k*f0
        bw = ft*rel_bw
        mask = (freqs >= ft-bw) & (freqs <= ft+bw)
        if not np.any(mask):
            rows.append({'n': k, 'freq_Hz': ft, 'dBV_meas': np.nan})
            continue
        idx = np.argmax(mag_dbv[mask])
        rows.append({'n': k, 'freq_Hz': freqs[mask][idx], 'dBV_meas':␣
 ↪mag_dbv[mask][idx]})
    return pd.DataFrame(rows)

theory_saw = pd.DataFrame({'n': n, 'freq_Hz': n*F0, 'dBV_theory': saw_dBV})
theory_tri = pd.DataFrame({'n': n, 'freq_Hz': n*F0, 'dBV_theory': tri_dBV})
theory_sin = pd.DataFrame({'n': n, 'freq_Hz': n*F0, 'dBV_theory': sin_dBV})

tables = {}
if t_saw is not None:
    df_saw_meas = pick_harmonics(f_saw, dbv_saw)
    comp_saw = pd.merge(theory_saw, df_saw_meas[['n','dBV_meas']], on='n',␣
 ↪how='left')
    comp_saw['delta_dB'] = comp_saw['dBV_meas'] - comp_saw['dBV_theory']
    tables['sawtooth_comparison'] = comp_saw
    display(comp_saw)
```

```
if t_tri is not None:
    df_tri_meas = pick_harmonics(f_tri, dbv_tri)
    comp_tri = pd.merge(theory_tri, df_tri_meas[['n','dBV_meas']], on='n',␣
 ↪how='left')
    comp_tri['delta_dB'] = comp_tri['dBV_meas'] - comp_tri['dBV_theory']
    tables['triangle_comparison'] = comp_tri
    display(comp_tri)
if t_sin is not None:
    df_sin_meas = pick_harmonics(f_sine, dbv_sine)
    comp_sin = pd.merge(theory_sin, df_sin_meas[['n','dBV_meas']], on='n',␣
 ↪how='left')
    comp_sin['delta_dB'] = comp_sin['dBV_meas'] - comp_sin['dBV_theory']
    tables['sine_comparison'] = comp_sin
    display(comp_sin)
```

### 1.9 8. RC filter: input→output comparison and theoretical attenuation

Compares measured attenuation (OUT–IN) at each harmonic to RC theory.

[26]:
```
if (t_in is not None) and (t_out is not None):
    df_in_lines  = pick_harmonics(f_in,  dbv_in)
    df_out_lines = pick_harmonics(f_out, dbv_out)
    df_rc_theory = pd.DataFrame({'n': n, 'freq_Hz': n*F0, 'RC_atten_dB': H_dB})
    comp = pd.merge(df_in_lines[['n','dBV_meas']].rename(columns={'dBV_meas':
 ↪'in_dBV'}),
                    df_out_lines[['n','dBV_meas']].rename(columns={'dBV_meas':
 ↪'out_dBV'}), on='n', how='inner')
    comp = pd.merge(comp, df_rc_theory[['n','RC_atten_dB']], on='n', how='left')
    comp['measured_atten_dB'] = comp['out_dBV'] - comp['in_dBV']
    comp['error_dB'] = comp['measured_atten_dB'] - comp['RC_atten_dB']
    display(comp)
else:
    print('Provide both IN_CSV and OUT_CSV for RC comparison.')
```

```
Provide both IN_CSV and OUT_CSV for RC comparison.
```

### 1.10 9. Time-domain waveforms (visual comparison)

Overlay filtered vs unfiltered signals to see how the RC filter smooths the sawtooth.

[27]:
```
def plot_time_overlay(t1, v1, t2, v2, title='Time-domain overlay', span=None):
    if (t1 is None) or (t2 is None):
        print('Both signals required for overlay.')
        return
    # Trim to same length/time window for a fair visual
    n = min(len(t1), len(t2))
    t1, v1 = t1[:n], v1[:n]
    t2, v2 = t2[:n], v2[:n]
```

```
    plt.figure()
    plt.plot(t1, v1, label='Input')
    plt.plot(t2, v2, label='Output (RC)')
    if span:
        plt.xlim(span)
    plt.xlabel('Time [s]'); plt.ylabel('Voltage [V]')
    plt.title(title)
    plt.grid(True); plt.legend(); plt.show()

plot_time_overlay(t_in, v_in, t_out, v_out, title='Time overlay - Input vs␣
 ↪Output (RC)')
```

```
Both signals required for overlay.
```

### 1.11 10. Results

- **Sawtooth:** Fundamental and harmonics up to 10 kHz visible; amplitudes decay ~1/n.
- **Triangle:** Odd harmonics only; amplitudes decay ~$1/n^2$ (faster than sawtooth).
- **Sine:** Only the fundamental expected; extra small peaks indicate noise/distortion.
- **RC filter:** Higher harmonics attenuated more; time-domain output is smoother.

### 1.12 11. Discussion

- Compare sawtooth vs triangle spectra and relate decay rates (1/n vs $1/n^2$) to time-domain shape (sharp edges vs smoother ramps).
- Explain why the sine ideally shows only one spectral line; discuss observed extra lines (windowing leakage, instrument noise, generator distortion).
- RC filter: Discuss how measured attenuation vs frequency matches theory (tabulated above). Comment on deviations (probe loading, FFT resolution, window choice, vertical scaling, anti-aliasing).
- Comment on sampling settings: ensure fs » 20 kHz to capture up to 10th harmonic; discuss aliasing risk and windowing effects on amplitude accuracy.

### 1.13 12. Conclusions

- Fourier series predictions align with measured spectra for sawtooth/triangle in trend and line positions.
- The RC low-pass filter reduces higher harmonics as predicted, yielding a smoother time waveform.
- Discrepancies are explainable by measurement setup and FFT processing choices.

### 1.14 References


- H. Hallenberg, *Signalbehandling och kommunikationssystem*, Yrgo, 2021.
- Keysight DSOX1204A user documentation.
- Wikipedia contributors, "RC filter," https://en.wikipedia.org/wiki/Low-pass_filter#RC_filter