
Digital Signal Processing for Engineering Applications

Laboratory Experiments



Prof. Dr. C. Roppel
Dipl.-Ing. (FH) M. Margraf
Faculty of Electrical Engineering
Laboratory for Telecommunications

Contents:

General Information	1
Python and the Jupyter Notebook	2
1 Using the Function Generator and the Oscilloscope	6
1.1 Check basic settings	6
1.2 Using BenchVue	7
1.3 Fast Fourier Transform (FFT) with the oscilloscope	9
1.4 Aliasing	11
1.5 Spectrum of a Square Wave	12
1.6 Amplitude Modulation (AM)	12
2 Sine and Dual Tone Signal	13
2.1 Sine Signal	13
2.2 Dual Tone Signal	16
3 Spectrum of the Dual Tone Signal	18
4 Bearing Vibration Analysis	20
4.1 Spectral Analysis	20
4.2 Filter Design	23
4.3 Envelope Detection	24

General Information

- For experiment 1 you work in the laboratory, while for the other experiments you work from home using Python.
- Handle the hardware in the laboratory (the measurement equipment, PCs, cables, connectors etc.) with care!
- Username and password for the lab PCs is "student".
- **All user files have to be stored in C:\usr. At the end of each lab session, send your files to your own e-mail account, and delete all your files on the lab PC.**

You can use your HSM mail account via webmail:
<https://webmail.fh-schmalkalden.de/>

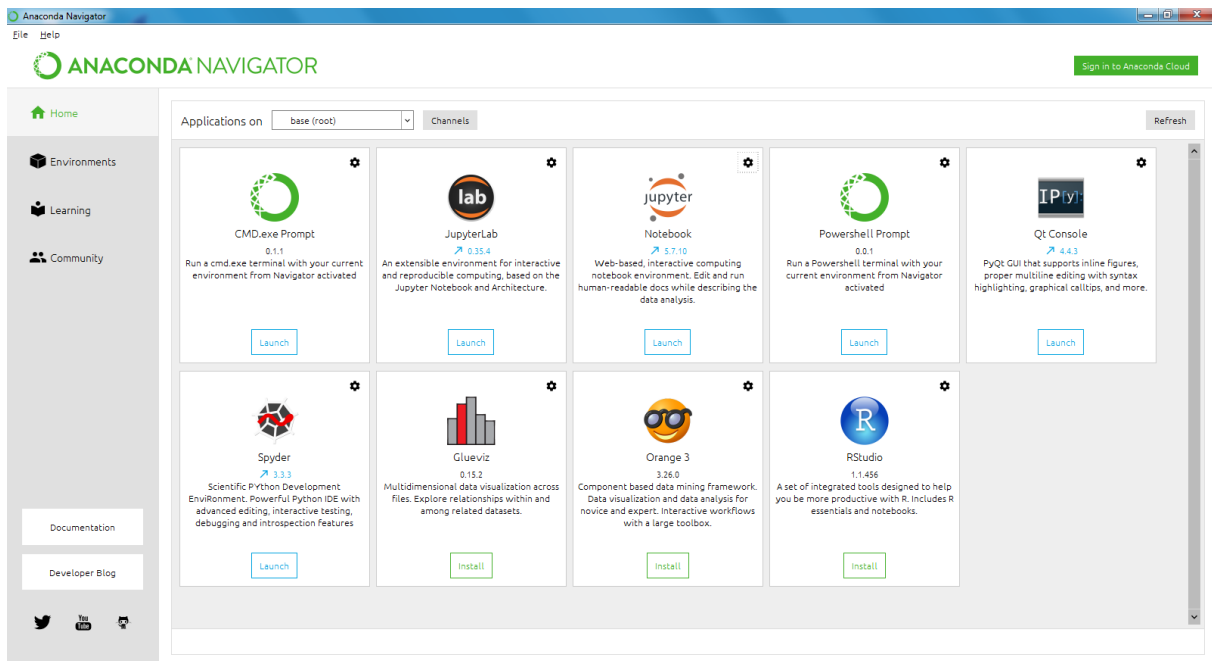
- Do **not** connect USB devices to the PC. **These devices will be scanned by the virus scanner and will generate an alarm, if a virus is detected!**
- Laboratory report: The Jupyter Notebook is the report. Generate a PDF (via HTML) from the notebook, do not send the Notebook itself. Change the filename:

The filename has to be DSP_EA_XXXXXX.pdf, where XXXXXX is your matriculation number.

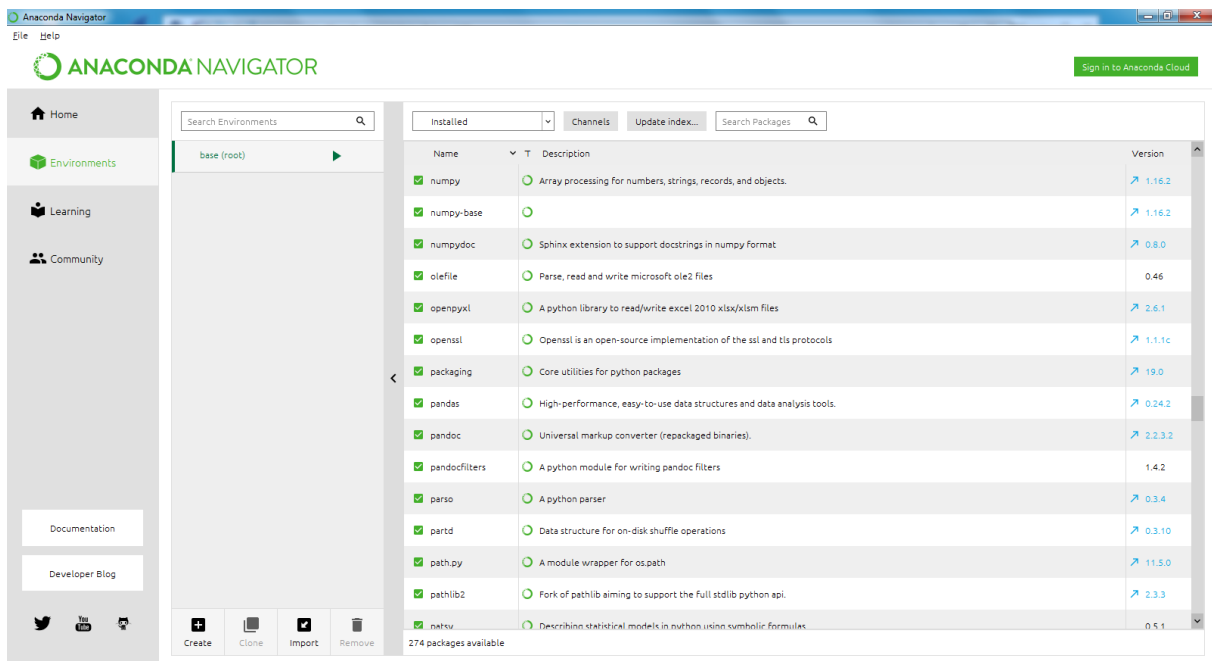
- Send the pdf document per e-mail to labort@hs-schmalkalden.de. The deadline for the report will be announced in the lecture.

Python and the Jupyter Notebook

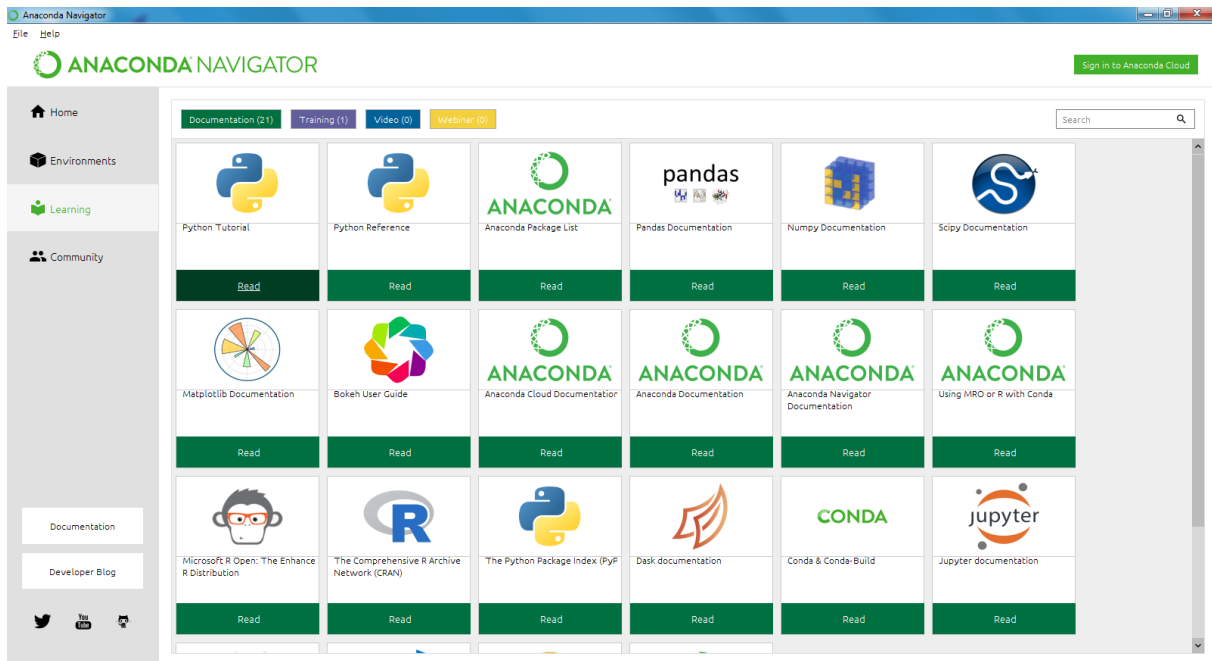
In this Laboratory we use the Python scientific ecosystem. As a user interface we use Jupyter notebooks, an interactive computing environment where you can combine text, code and visualizations. We recommend the Anaconda distribution, which includes all the necessary packages and libraries. Download and install *Anaconda Individual Edition* from www.anaconda.com. Launch the Anaconda Navigator:



Select *Environments*. Here you can see and manage the packages. We need Numpy, Matplotlib, Scipy and Pandas, which are installed.

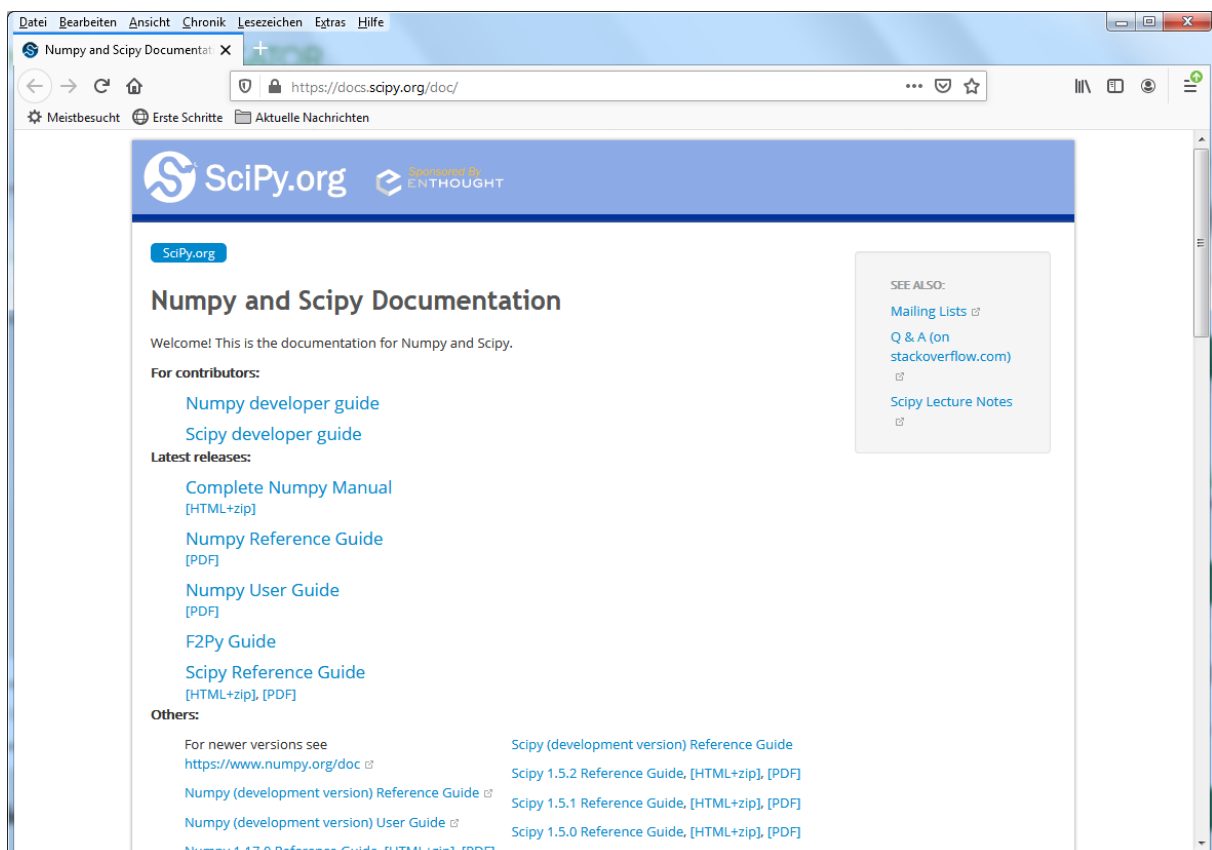


Select *Learning*. Here you can access the documentation for Anaconda and the packages.

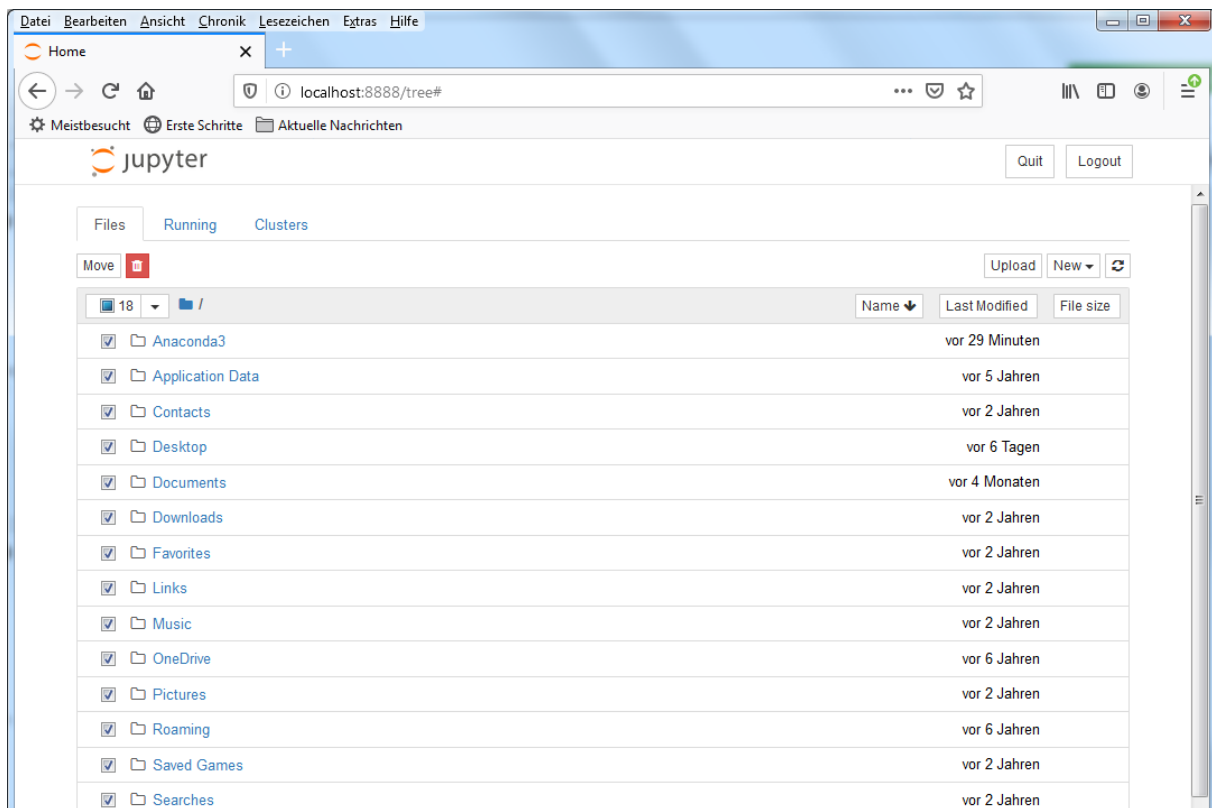


To get started, select *SciPy Documentation*, next select *SciPy Lecture Notes*, and work through the lecture notes in order to learn the basics about Python, Numpy, Matplotlib and SciPy.

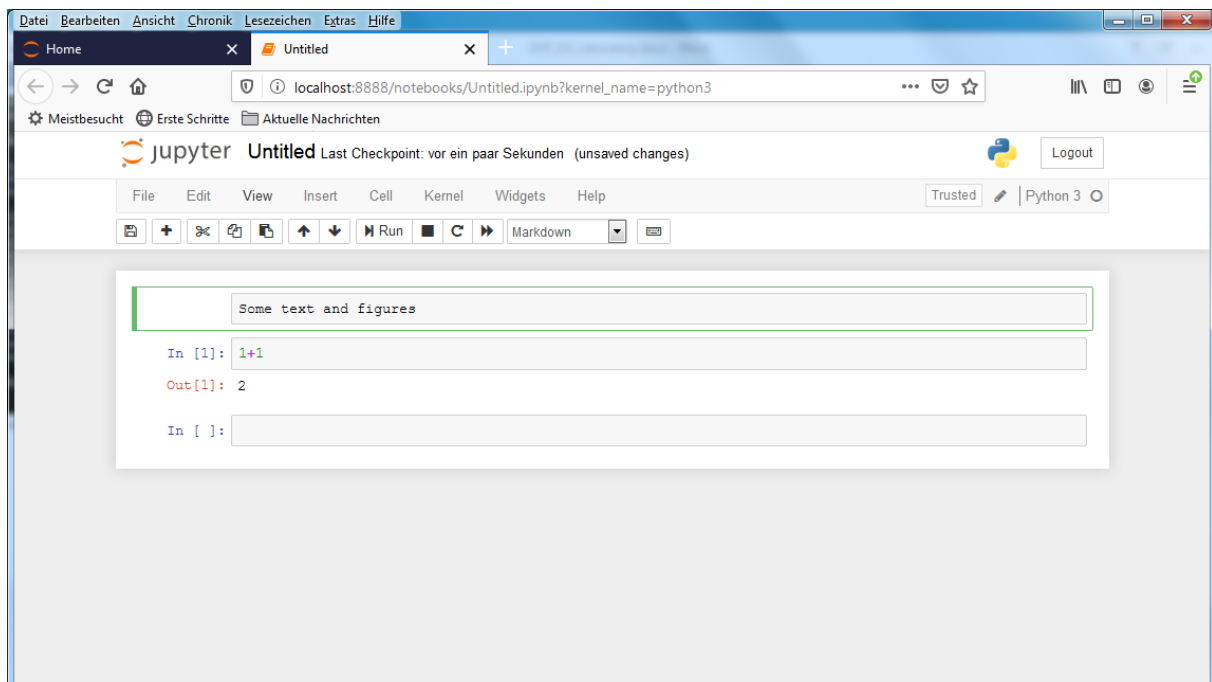
Also have a look at the full documentation for Python, Numpy, Matplotlib, SciPy and Pandas, which you will need while working through the lab experiments.



Launch either Anaconda Navigator, and then Jupyter Notebook, or Jupyter Notebook directly from the start menu. This starts the notebook server and opens the notebook dashboard in a browser window:



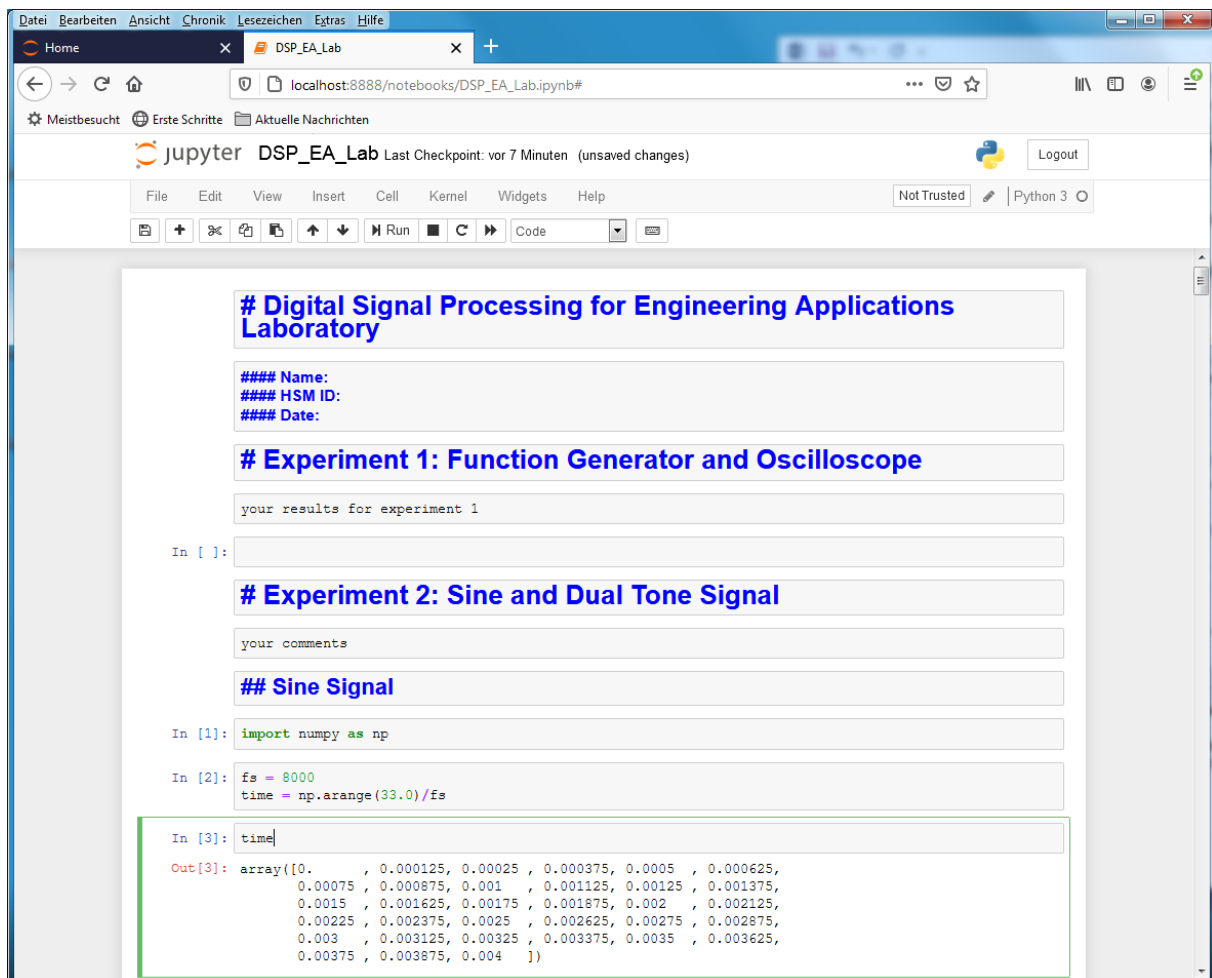
Select *New* -> *Notebook* (Python 3), to create a new notebook.



New cells are by default code cells. If you change the cell type to *Markdown*, you can enter text and insert images.

To finish, go to the Home tab and select *Quit*. This stops the notebook server and you can close the browser (just closing the browser does not stop the notebook server).

In a markdown cell, enter the title "Digital Signal Processing for Engineering Applications Laboratory". Add another markdown cell and enter your name, HSM ID and the date. Add another markdown cell and enter the heading "Experiment 1". Under this heading, insert your results for experiment 1. Proceed in a similar way for the other experiments.



To generate a PDF document from the notebook, select *File -> Download as -> pdf*. This requires additional software like Latex. Alternatively, you can select *Download as -> HTML*, open the HTML document in your browser and print to PDF.

1 Using the Function Generator and the Oscilloscope

In the lab you work with the HP33120A function generator and the digital oscilloscopes Keysight DSOX 2002A or DSO 6032A. The user manual for the HP33120A is available in printed form in the lab. The user's guide for the oscilloscopes is available on the PCs in the directory C:\usr\Keysight. Screen shots of the oscilloscope can be taken with the BenchVue software.

Do not change the language settings (English) of both the oscilloscope and the BenchVue software.

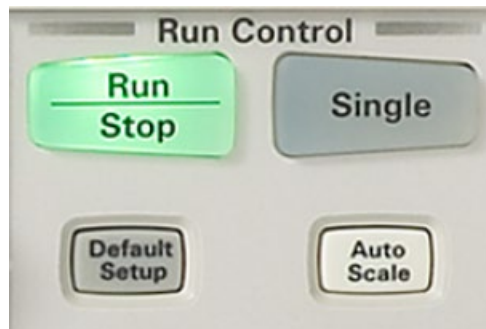
The following instructions refer to the DSOX 2002A. The operation of the DSO 6032A is similar; the BenchVue software is used for both oscilloscopes. When different parameters for the DSO 6032A have to be used, this is indicated like

[DSO 6032A: horizontal scale 2 ms/, span 25 kHz, sampling rate 50 kHz].

In the text below, [key] refers to a key and SK[key] refers to a softkey of the oscilloscope. A softkey is one of the five keys below the screen. By rotating and pressing the entry knob to the right of the screen you select entries in the softkey menus. With the round key (Back) to the left of the softkeys you can step back in the softkey menu.

1.1 Check basic settings

1. Press [Default Setup] on the oscilloscope (DSOX 2002A only).



2. Connect the function generator output to channel 1 input of the oscilloscope. Generate a 1 kHz sine wave with amplitude 2 V_{pp} (peak-to-peak). Press [Autoscale] on the oscilloscope.

The status line at the top of the display shows the vertical and the horizontal scale settings. The vertical scale now is 500 mV/ and the horizontal scale is 200 μs/.

3. Change the amplitude and frequency of the function generator and observe the oscilloscope display. When finished, set it again to 1 kHz and 2 V_{pp}.
4. Change the vertical and horizontal scale of the oscilloscope and observe the display. When finished, press [Autoscale] again.

5. Measure the amplitude (peak-to-peak) and the frequency of the sine signal:

- Key [Meas] → SK[Source] → Select channel 1
- SK[Type] → Select "Peak-Peak" with the entry knob
- SK[Add Measurement] → Select "Frequency" with the entry knob



The measurements of the sine signal should show an amplitude of 2 V_{pp} and a frequency of 1 kHz . If the amplitude is too large, the load setting of the function generator has to be corrected. The function generator has a source impedance of $50\ \Omega$. If the generator is connected to a $50\ \Omega$ load, the amplitude is only one half compared to the case when the generator is connected to a high impedance load. The generator can be configured for a $50\ \Omega$ load or a high impedance load in order to show the correct amplitude. The setting can be modified in the menu. Press the keys [Shift] [Menu], navigate with the arrow keys to D:SYS MENU and further to 1:OUT TERM, where you can select either $50\ \text{OHM}$ or HIGH Z . Since the oscilloscope is a high impedance load (approx. $1\text{ M}\Omega$), choose HIGH Z .

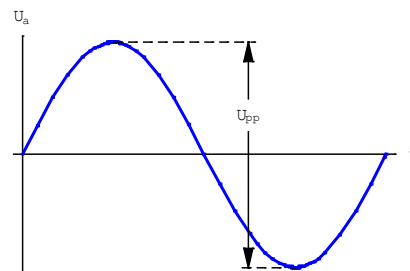
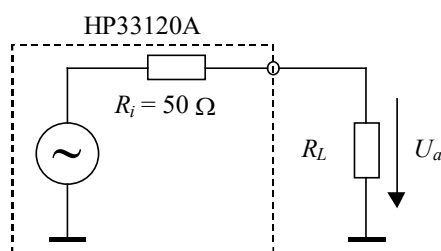
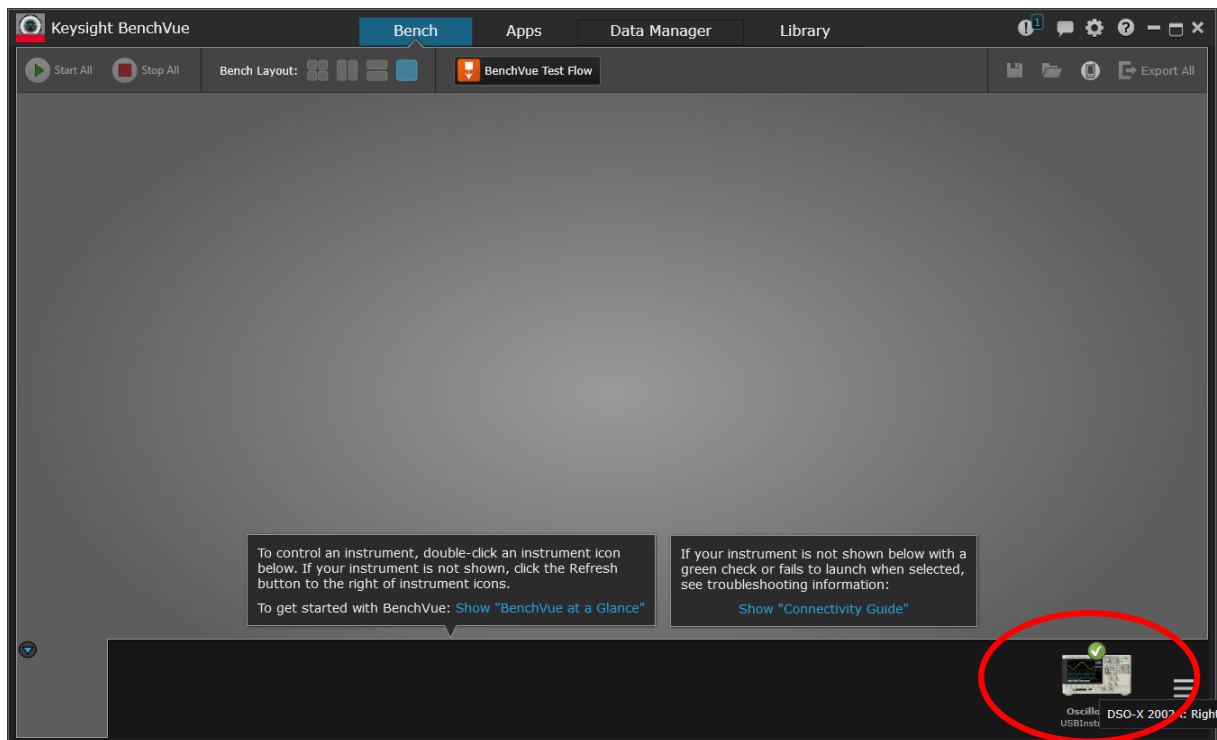


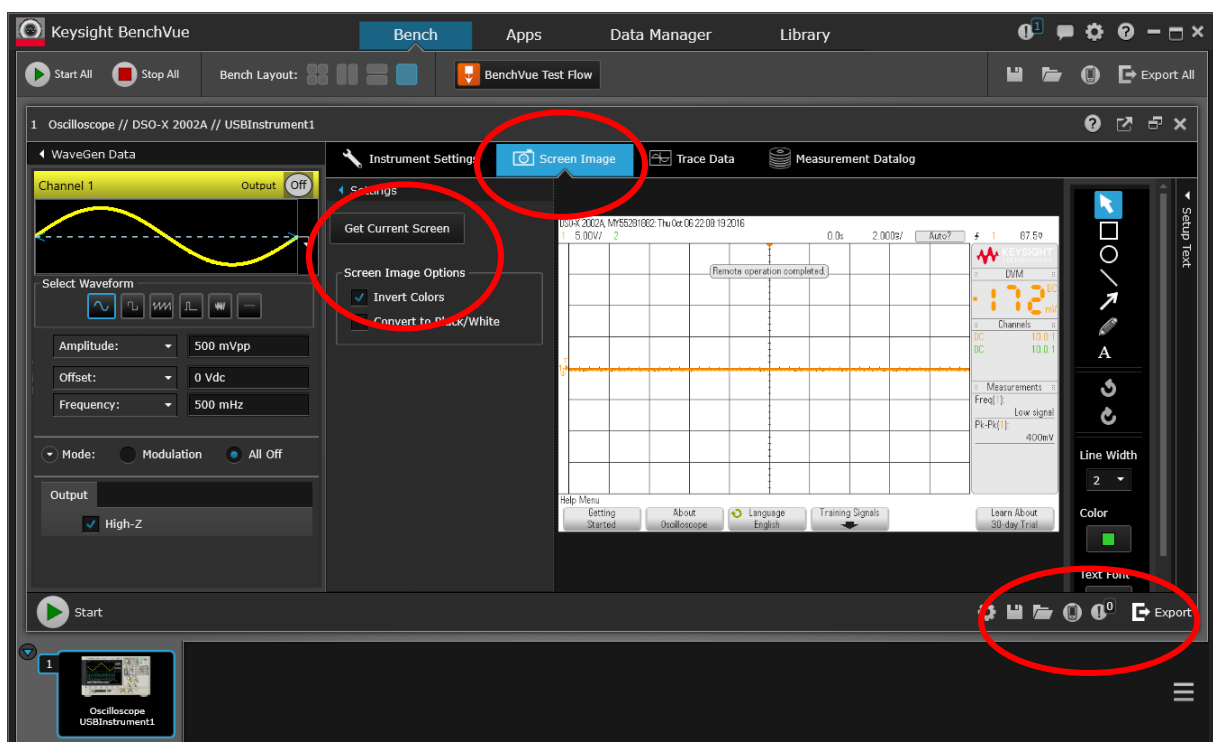
Figure 1-1: The amplitude of the function generator depends on the load impedance R_L

1.2 Using BenchVue

1. Start the BenchVue software on the PC.
2. To start the oscilloscope app, double-click on the oscilloscope symbol in the lower right corner.



3. Go to the Screen Image tab. Select "Invert Colors" (otherwise the graphics have a black background color, which is unfavorable for printing). Click on "Get Current Screen".
4. Click on "Export" or the save symbol in order to save screenshots of the oscilloscope in various formats.

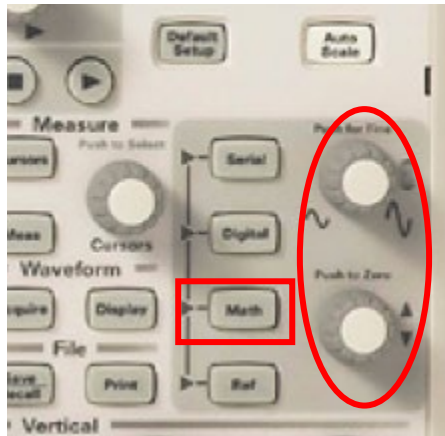


1.3 Fast Fourier Transform (FFT) with the oscilloscope

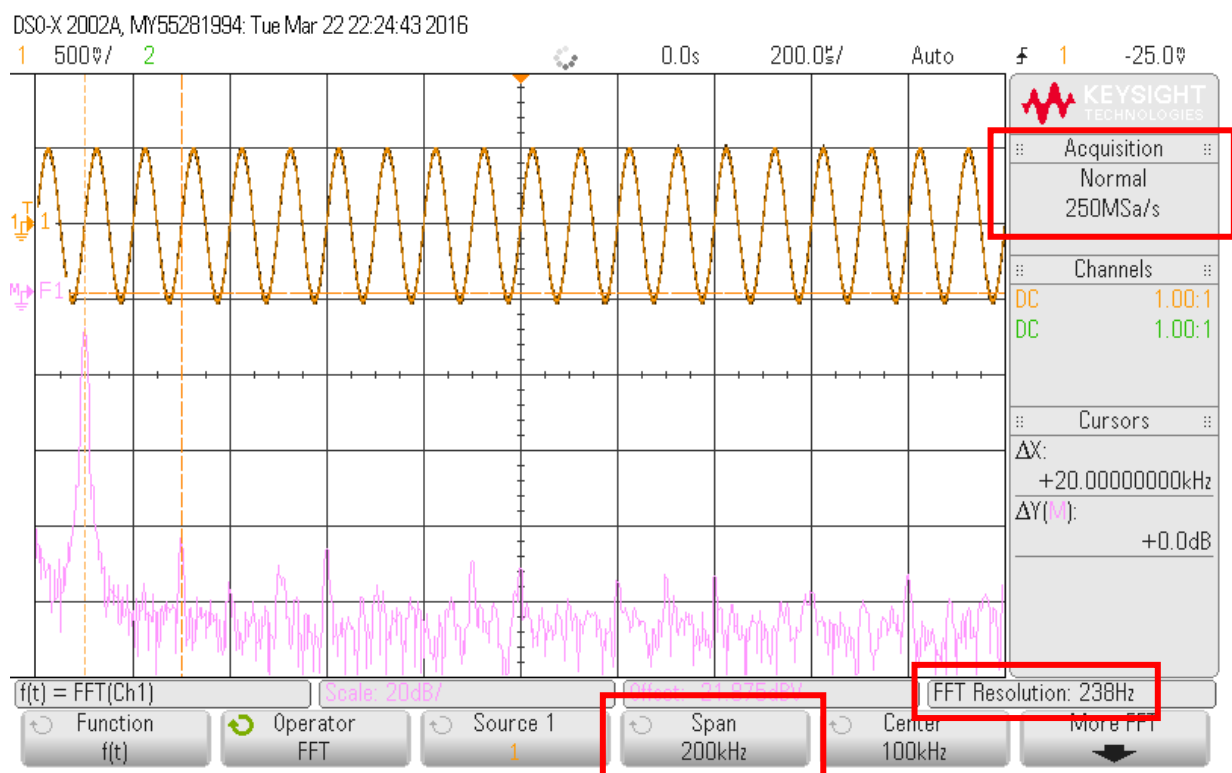
Measurements in the frequency domain are based on the Discrete Fourier Transform (DFT). Fast Fourier Transformation (FFT) is a term for algorithms, which allow for a fast and efficient computation of the DFT. In this section we use the oscilloscopes built-in FFT function to display the spectrum of the signal.

1. Activate the FFT:

Key [Math] → SK[Operator] → select FFT → SK[Source] → select Channel 1, SK[More FFT] → SK[Auto Setup] to show the entire available spectrum



On the screen the magnitude spectrum of the signal is displayed. The frequency range (span) and the resolution are displayed above the softkeys.



- The span depends on the sampling rate. The sampling rate is shown in the upper right corner and depends on the horizontal scale setting. A horizontal scale of 200 $\mu\text{s}/$ results in a sampling rate of 250 MSa/s (Million Samples per second). However, the *FFT sampling rate* is usually lower and is related to the span by

$$\text{FFT sampling rate} = 2 \text{ times Span}$$

The resolution depends on the FFT sampling rate:

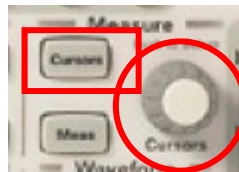
$$\text{Resolution} = \text{FFT sampling rate} / \text{Number of samples}$$

Change the horizontal scale and observe how this changes the signal in the time domain, the sampling rate, the span and resolution of the FFT.

By manually setting the span and the center frequency you can zoom into the FFT (this was done in the screenshot shown). But this does not change the resolution! With the knobs to the right of the [Math] key you can change the vertical scale of the FFT.

- Autoscale the FFT (press SK[More FFT] \rightarrow SK[Auto Setup]). Set the horizontal scale to 200 ms/. The span now is 15.6 kHz (sampling rate 31.2 kHz). The spectral line of the sine wave at 1 kHz is clearly visible [\[DSO 6032A: horizontal scale 2 ms/, span 25 kHz, sampling rate 50 kHz\]](#). Use the cursor to measure the frequency:

Key [Cursors] \rightarrow SK[Source] \rightarrow select Source: Math, SK[Cursor] \rightarrow select X1, move the cursor in x direction by turning the cursor knob. Move the cursor to the peak, the frequency is displayed in the cursor box.



- The FFT vertical units are dBV (Decibel-Volt referenced to an rms voltage of 1 V):

$$U_{\text{dBV}} = 20 \log \frac{U_{\text{rms}}}{1 \text{ V}}$$

A sine wave having an rms (root-mean square) voltage of 1 V has an amplitude of 0 dBV.

Use the cursor to measure the amplitude:

Key [Cursors] \rightarrow SK[Source] \rightarrow select Source: Math, SK[Cursor] \rightarrow select Y1, move the cursor in y direction by turning the cursor knob. Move the cursor to the peak, the amplitude is displayed in the cursor box.

For precise measurements of the amplitude, zoom into the FFT by manually setting the span and the center frequency. Then it will be easier to move the cursor to the tip of the peak in the spectrum.

Test different window functions: Press SK[More FFT] \rightarrow SK[Window], and select different window functions with the entry knob. Observe the effect on the shape of the FFT. Use the Flat Top window for best amplitude measurements.

5. Select different amplitudes at the function generator, and note down in a table the following values:
 - The peak-to-peak amplitude of the function generator
 - The calculated rms value
 - The calculated dBV value
 - The measured dBV value
6. Autoscale the FFT (SK[More FFT] → SK[Auto Setup]). When the amplitude of the input signal exceeds the vertical range of the oscilloscope, the signal is clipped. Increase the amplitude of the function generator until clipping occurs, while observing the spectrum. In order to see more details in the spectrum, you can switch off the time-domain signal by pressing key [1] for channel 1 two times.

1.4 Aliasing

1. Adjust the function generator to generate a sine wave with a frequency of 1 kHz and an amplitude of 2 Vpp. Adjust the horizontal scale of the oscilloscope to 200 ms/, activate the FFT and select auto setup (SK[More FFT] → SK[Auto Setup]). The span will be 15.6 kHz [DSO 6032A: 2 ms/, Span 25 kHz].
2. Increase slowly the frequency of the function generator. Wait for a stable FFT display and watch the peak in the spectrum moving to the right.
3. Increase the frequency beyond 15.6 kHz. Since the FFT sampling rate is 31.2 kHz, signals above 15.6 kHz will be aliased. Watch the peak, while increasing the frequency further [DSO 6032A: sampling rate 50 kHz, aliasing above 25 kHz].
4. What happens, when you reach a frequency of 31.2 kHz [DSO 6032A: 50 kHz]? What happens, when you increase the frequency further?
5. Select different frequencies at the function generator, and note down in a table the following values:
 - Generator frequency f_0
 - Effective sampling rate (FFT sampling rate) f_s
 - FFT frequency (frequency of the peak in the spectrum)
 - The difference $f_s - f_0$
 - The sum $-f_s + f_0$

1.5 Spectrum of a Square Wave

1. Adjust the function generator to generate a square wave with a frequency of 10 kHz and an amplitude of 2 Vpp. Press [Autoscale] and observe the time domain signal. Measure the period of the square wave.
2. Activate the FFT. Adjust the horizontal scale of the oscilloscope in order to see a frequency range of about 200 kHz.
3. Measure the frequency of the harmonics, and note down the values in a table.

1.6 Amplitude Modulation (AM)

1. Adjust the function generator to generate an AM signal with these parameters:
 - Carrier frequency 5 kHz
 - Carrier amplitude 1 Vpp
 - Modulating frequency 200 Hz
 - Modulation index 0.7 (70 %)
2. Press [Autoscale] and observe the time domain signal. Change the modulating frequency and the modulation index and observe the effect on the.
3. Activate the FFT. Adjust the horizontal scale of the oscilloscope in order to see a span of 15.6 kHz.
4. Measure the frequency of the spectral components of the AM signal, and note down the values in a table.
5. Change the modulating frequency and the modulation index and observe the effect on both the time domain waveform and the spectrum.

2 Sine and Dual Tone Signal

In this experiment we generate a discrete-time version of a sine wave with Python and we plot the signal. By using the sound card of the PC we can generate an analog signal, which we can hear and measure with the oscilloscope. Next we generate a dual tone signal, which is composed of two sine waves having different frequencies and amplitudes.

2.1 Sine Signal

Generate a sine wave with the frequency $f_1 = 1000$ Hz, amplitude $A_1 = 1$ and the sampling rate $f_s = 8000$ Hz.

1. In the Jupyter notebook, add a section for Experiment 2.
2. Import numpy. Define a variable $fs = 8000$ (sampling rate). Generate a vector *time* with the sampling times (0, $1/f_s$, $2/f_s$, ...) with a total length of 33 samples, using the numpy function `arange()`.

Note that we use `arange(33.0)` instead of `arange(33)`. The latter results in an array of integer numbers, which can cause overflow problems when we multiply the array with a large number (try it!). `arange(33.0)` results in an array of floating point numbers. You can check the data type of a variable with `variablename.dtype`.

Experiment 1: Function Generator and Oscilloscope

your results for experiment 1

In []:

Experiment 2: Sine and Dual Tone Signal

your comments

Sine Signal

In [1]: `import numpy as np`

In [2]: `fs = 8000`
`time = np.arange(33.0)/fs`

In [3]: `time`

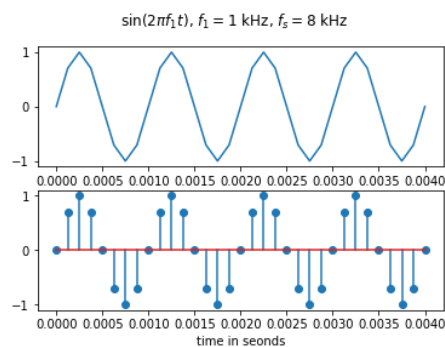
Out[3]: `array([0. , 0.000125, 0.00025 , 0.000375, 0.0005 , 0.000625,`
`0.00075 , 0.000875, 0.001 , 0.001125, 0.00125 , 0.001375,`
`0.0015 , 0.001625, 0.00175 , 0.001875, 0.002 , 0.002125,`
`0.00225 , 0.002375, 0.0025 , 0.002625, 0.00275 , 0.002875,`
`0.003 , 0.003125, 0.00325 , 0.003375, 0.0035 , 0.003625,`
`0.00375 , 0.003875, 0.004])`

3. Define a variable $f_1 = 1000$ and calculate the sine values $\sin(2\pi f_1 t)$ of the sampling times defined by the vector *time*.
4. Import matplotlib.pyplot. Plot the sine signal using the functions `plot()` and `stem()`. Generate a plot with two subplots, with axis label and plot title, similar to the plot shown in the screen shot below. For mathematical expressions in text like labels, the Latex style is supported.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt

In [2]: fs = 8000
f1 = 1000
time = np.arange(33.0)/fs
sine1 = np.sin(2*np.pi*f1*time)

In [3]: plt.figure()
plt.subplot(2, 1, 1)
plt.plot(time, sine1)
plt.subplot(2, 1, 2)
plt.stem(time, sine1)
plt.xlabel('time in seconds')
plt.suptitle('$\sin(2 \pi f_1 t)$, $f_1 = 1$ kHz, $f_s = 8$ kHz')
plt.show()
```



- Typically, we start with a few samples and check the output. Now we extend the number of samples to 16000 (corresponding to a signal time of 2 seconds). Because a plot of the 2 seconds signals does not make sense, we pass only the first 33 samples of the time vector and the signal vector to the plot function.

Check the signal length using the function `len()`.

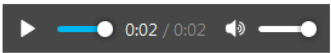
- Import the Audio class from `IPython.display`, and use the `Audio()` function to listen to the sound.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio

In [2]: fs = 8000
f1 = 1000
time = np.arange(16000.0)/fs
sine1 = np.sin(2*np.pi*f1*time)

In [3]: len(sine1)
Out[3]: 16000

In [6]: plt.figure()
plt.subplot(2, 1, 1)
plt.plot(time[0:33], sine1[0:33])
plt.subplot(2, 1, 2)
plt.stem(time[0:33], sine1[0:33])
plt.xlabel('time in seconds')
plt.suptitle('$\sin(2 \pi f_1 t)$, $f_1 = 1$ kHz, $f_s = 8$ kHz')
plt.show()

In [4]: Audio(sine1, rate = fs)
Out[4]: 
```

This is only possible, if you have access to the lab:

- Connect the oscilloscope to the audio output of the PC.
- Measure frequency and amplitude of the signal in the time domain.
- Measure frequency and amplitude of the signal using the FFT function. Convert the dBV value of the amplitude to the peak-to-peak value.

2.2 Dual Tone Signal

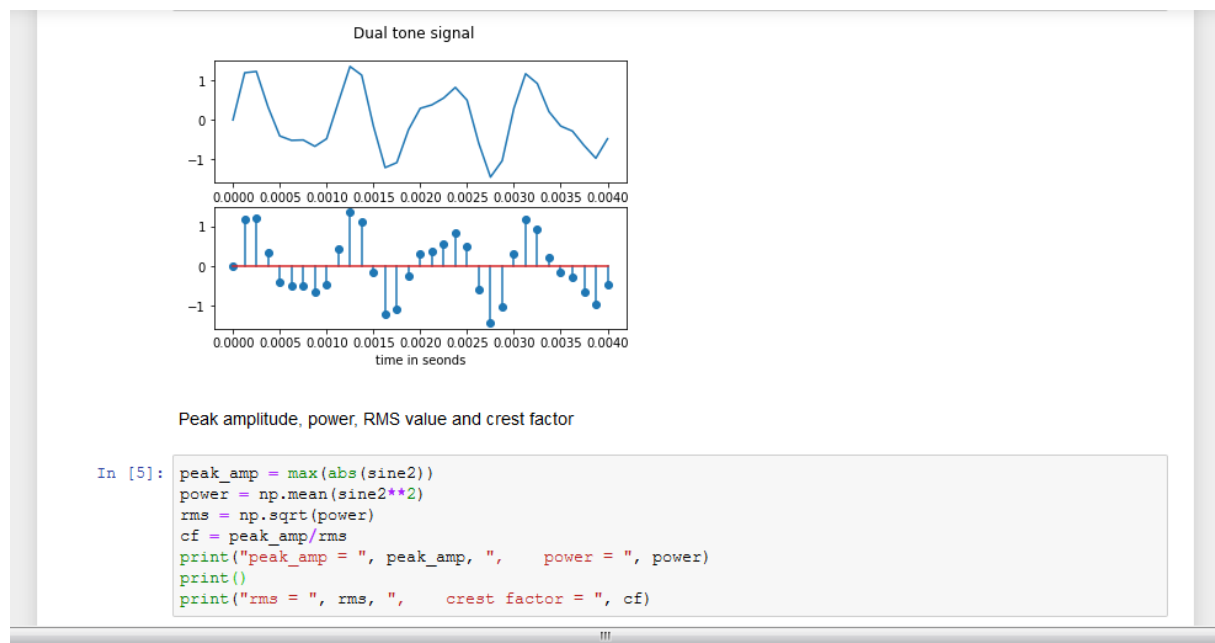
1. Add a subsection with the heading "Dual Tone Signal".
2. Although not necessary, copy the code from the previous section including the import of the packages. This allows you to work only with the code in the current subsection.
3. Add a second sine wave to the 1 kHz signal. The second signal shall have the amplitude $A_2 = 0.5$ and the frequency $f = 1.7$ kHz. Again, generate a plot of the first 33 samples, and use the audio function to listen to the signal.

Dual Tone Signal

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Audio

In [2]: fs = 8000
f1 = 1000
f2 = 1700
time = np.arange(16000.0)/fs
sine2 = np.sin(2*np.pi*f1*time) + 0.5*np.sin(2*np.pi*f2*time)
```

4. Determine the peak amplitude, the power, the rms value and the crest factor of the signal using the `abs()`, `max()` and `mean()` functions.



5. Calculate the theoretical values for the power, the rms value and the crest factor. Assume a peak amplitude of $A_1 + A_2 = 1.5$, and that the two sine waves are uncorrelated and that the total power is the sum of the individual power of the two sine waves.
6. By default, the audio function normalizes the amplitude of the signal to the range ± 1 . Now use the option `normalize = False`. This will result in an error message, because the peak amplitude of the signal is larger than 1.

Scale the amplitude of the dual tone signal accordingly.

Note: When you import or generate signals, it is good practice to normalize the amplitude to 1.



```
Normalize amplitude to 1

In [44]: sine2 = sine2/max(abs(sine2))
         max(sine2)

Out[44]: 0.999999999999982

In [34]: Audio(sine2, rate = fs, normalize = False)

Out[34]:
```

The screenshot shows a Jupyter Notebook interface. At the top, there is a text label "Normalize amplitude to 1". Below it, the first code cell (In [44]) contains the code `sine2 = sine2/max(abs(sine2))` and `max(sine2)`. The output (Out[44]) is `0.999999999999982`. The second code cell (In [34]) contains the code `Audio(sine2, rate = fs, normalize = False)`. The output (Out[34]) is a small audio player widget showing a play button, a progress bar at 0:02 / 0:02, and a volume icon.

This is only possible, if you have access to the lab:

- Connect the oscilloscope to the audio output of the PC.
- Measure the frequencies and amplitudes of the two sine signals using the FFT function.
- Convert the dBV values of the amplitudes to their peak-to-peak values, and confirm that the ratio is 2:1.
- Calculate the ratio of the amplitudes using the dBV values.

3 Spectrum of the Dual Tone Signal

In this experiment we will calculate the spectrum of the dual tone signal of Section 2.2 using the discrete Fourier transform (DFT):

$$S_{\text{DFT}}(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nk/N}, \quad k = 0, 1, \dots, N-1$$

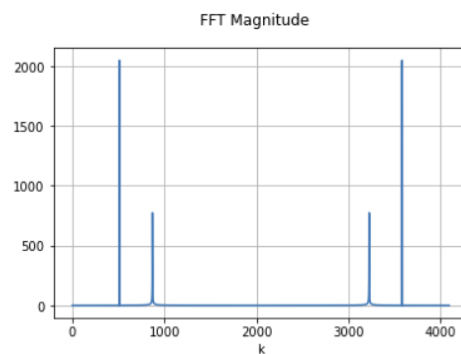
The spectrum calculated by means of an N -point DFT is composed of N values spaced $\Delta f = f_s/N$ apart. All N values represent the spectrum in the frequency range from 0 to f_s , so for the k th value we have

$$f_k = \frac{k}{N} f_s, \quad k = 0, 1, \dots, N-1$$

1. In the Jupyter notebook, add a section for experiment 3. Copy the relevant part of the code from Section 2.2 to generate the dual tone signal. Change the signal length to $N = 4096$ samples.
2. Use the function `fft()` in the `numpy.fft` module to calculate the DFT. Plot the whole spectrum (more precisely, the magnitude of the DFT) over k ($k = 0, 1, \dots, N-1$).

```
In [18]: spec = abs(np.fft.fft(sine2))
```

```
In [20]: plt.plot(spec)
plt.grid()
plt.xlabel('k')
plt.suptitle('FFT Magnitude')
plt.show()
```



3. Plot the whole spectrum over frequency f_k ($k = 0, 1, \dots, N-1$). For this purpose define a vector `freq_axis` using the `numpy` command `arange()` (see Section 2.1).
4. The previous plot shows the spectrum in the range from 0 to f_s , where the range from $f_s/2$ to f_s is the periodic repetition of the spectrum due to the sampling. We obtain the two-sided spectrum in the range $-f_s/2 \leq f \leq f_s/2$ by swapping the upper half of spectrum with its lower half. This can be done using the function `fftshift()` in the `numpy.fft` module. Define a new frequency axis for the range $-f_s/2 \leq f \leq f_s/2$ and plot the two-sided spectrum over frequency.

5. Because of the symmetry of the spectrum, it is often shown as a one-sided spectrum in the range $0 \leq f \leq f_s/2$. Plot this one-sided spectrum over frequency.
6. Finally, plot the one-sided spectrum in Decibels (dB) over frequency. Use the numpy function `log10()` to convert the spectrum to Decibels. Now you should clearly see the leakage effect for the 1700 Hz sine wave.

4 Bearing Vibration Analysis

Here we analyse the signals from bearings with different faults available from the Society For Machinery Failure Prevention Technology (<https://mfpt.org/fault-data-sets/>). The bearings have the following parameters:

Revolution per second of the axle:	$f_{rev} = 25 \text{ Hz}$
Number of balls:	$N = 8$
Ball diameter:	$d = 0.235 \text{ inch}$
Average bearing diameter:	$D = 1.245 \text{ inch}$
Contact angle:	$\phi = 0$

From this data set, we use three files:

- baseline.xlsx
- OuterRaceFault.xlsx
- InnerRaceFault.xlsx

These Excel files were obtained from the original binary files and contain signals sampled with a rate of 48828 Hz for 3 seconds.

4.1 Spectral Analysis

1. In the Jupyter notebook, add a section for experiment 4. Calculate the characteristic frequencies of the bearing:
 - Fundamental Train Frequency (FTF)
 - Ball Spin Frequency (BSF)
 - Ball Pass Frequency Outer Race (BPFO)
 - Ball Pass Frequency Inner Race (BPFI)
2. We use the pandas function `read_excel` to import excel files. Import pandas and read in baseline.xlsx. This results in a pandas data frame. Convert the data frame to a numpy array using `to_numpy`. Define a variable $fs = 48828$ (sampling rate) and check the signal length (number of samples).

Import Excel File

```
In [3]: dataframe = pd.read_excel("c:/baseline.xlsx")
# signal = dataframe.values
signal = dataframe.to_numpy()
```

```
In [4]: fs = 48828
nsamples = len(signal)
print('Sampling Rate = ', fs, 'Hz', '\nNumber of Samples = ', nsamples)
```

```
Sampling Rate = 48828 Hz
Number of Samples = 146483
```

- Now *signal* is a matrix with only one column, that is an `array([[...]])`. Change this to a vector, that is to `array([...])`, using `signal = signal[:, 0]`.

Plot the signal over time in seconds.

```
In [6]: signal
Out[6]: array([[ -3.86744880e-06,
                  2.22286128e-04,
                  1.91153992e-04,
                  ...,
                 -8.95136886e-01,
                 -1.26335620e+00,
                 7.03594592e-01]])

In [7]: signal = signal[:, 0]
signal
Out[7]: array([ -3.86744880e-06,  2.22286128e-04,  1.91153992e-04, ...,
                -8.95136886e-01, -1.26335620e+00,  7.03594592e-01])
```

- Normalize the amplitude of the signal to 1. Again, plot the signal over time in seconds, and confirm that the signal is in the range ± 1 . Use the `Audio` function to listen to the sound.
- For the bearing vibration analysis we have to plot the spectrum of signals often. Because the characteristic frequencies are quite low, we are interested in the spectrum up to a frequency of 500 Hz. Therefore we define a function `plot_spectrum_500`, which plots the one-sided spectrum of a signal up to 500 Hz, scaled in Decibels.

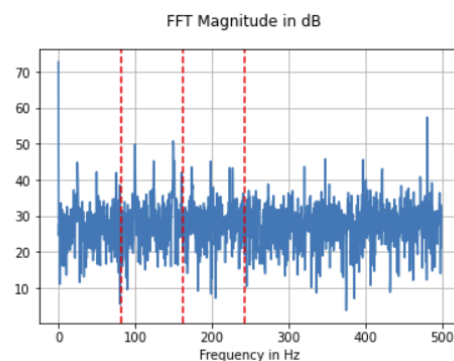
Use this function to plot the spectrum.

```
def plot_spectrum_500(x, fs):
    # Plot the one-sided spectrum (magnitude of DFT) of signal x, scaled in Decibels,
    # in the frequency range from 0 to fmax
    # fs: sampling rate
    fmax = 500
    nsamples = len(x)
    kmax = round((fmax/fs)*nsamples)
    spec = abs(np.fft.fft(x))
    freq_axis = np.arange(float(nsamples))*fs/nsamples
    plt.plot(freq_axis[0:kmax], 20*np.log10(spec[0:kmax]))
    plt.grid()
    plt.xlabel('Frequency in Hz')
    plt.suptitle('FFT Magnitude in dB')
```

- Calculate the crest factor of the signal.
- Calculate the kurtosis of the signal.

8. Now repeat steps 2 to 7 for OuterRaceFault.xlsx.
- When plotting the spectrum, add vertical lines at integer multiples of BPFO. Can you see a spectral line at BPFO?
 - Compare the crest factor and the kurtosis with the values for the baseline signal.

```
In [16]: plot_spectrum_500(signal, fs)
plt.axvline(BPFO, color = "r", linestyle = "--");
plt.axvline(2*BPFO, color = "r", linestyle = "--");
plt.axvline(3*BPFO, color = "r", linestyle = "--");
```



9. Now repeat steps 2 to 7 for InnerRaceFault.xlsx.
- When plotting the spectrum, add vertical lines at integer multiples of BPFI. Can you see a spectral line at BPFI?
 - Compare the crest factor and the kurtosis with the values for the baseline signal.

4.2 Filter Design

Before we continue with the analysis of the vibration signal, we will consider the design of FIR filters, which we will need for extracting the envelope of the vibration signal.

FIR filters are designed with the function `firwin()` from the `scipy.signal` module. A lowpass filter is designed as follows:

```
import scipy.signal as scs
b = scs.firwin(n_coeffs, f0)
```

This is a window based design using the Hamming window as a default. `n_coeffs` is the number of coefficients, and `f0` is the bandwidth. `f0` is a number in the range $[0 \dots 1]$, where 1 corresponds to $f_s/2$. The function returns the array `b`, the filter coefficients.

A bandpass filter is designed using

```
b = scs.firwin(n_coeffs, [f0, f1], pass_zero = False)
```

The band edges of the passband are `f0` and `f1`. With `pass_zero = False` the range from 0 to `f0` is a stopband.

The function

```
w, H = scs.freqz(b)
```

calculates the frequency response of the filter in the range from 0 to $f_s/2$. `w` is the frequency vector and `H` is the frequency response vector. The frequencies `w` are given in radians in the range from 0 to π , where π corresponds to $f_s/2$. The magnitude of `H` is the amplitude response of the filter.

The function `lfilter()` calculates the filtered signal `y`, using the filter coefficients `b` and the input signal `x`:

```
y = scs.lfilter(b, 1, x)
```

1. Design a lowpass filter with 127 coefficients and the bandwidth $0.1 f_s$.

Plot the impulse response of the filter.

Plot the amplitude response of the filter, scaled in dB, over frequency in the range from 0 to $f_s/2$.

2. Design a bandpass filter with 127 coefficients and the band edges $0.1 f_s$ and $0.2 f_s$.

Plot the impulse response of the filter.

Plot the amplitude response of the filter, scaled in dB, over frequency in the range from 0 to $f_s/2$.

4.3 Envelope Detection

When a ball hits a fault on the outer or inner race, a signal with BPFO or BPFI is generated, respectively. However this signal modulates signals at other resonant frequencies. One method for demodulating these amplitude modulated signals is envelope detection.

Because we do not know the resonant frequency, we can use a combination of filters, crest factor, kurtosis or spectral kurtosis to select a promising frequency band. Here we filter the signal first with different filters and calculate the crest factor. Use the following filters:

Lowpass	2 kHz		
Bandpass	2 kHz	to	4 kHz
Bandpass	4 kHz	to	6 kHz
Bandpass	6 kHz	to	8 kHz
Bandpass	8 kHz	to	10 kHz
Bandpass	10 kHz	to	12 kHz
Bandpass	12 kHz	to	14 kHz

Choose the filter which results in the highest crest factor. To extract the envelope of the signal, we have to take the absolute value of the signal, and we have to send this signal through a lowpass filter with the bandwidth 500 Hz.

1. Import OuterRaceFault.xlsx as described in Section 4.1.
2. Design lowpass/bandpass filters as described above with 127 coefficients. Generate a single plot showing the amplitude response (scaled in dB) of all filters.
3. Filter the vibration signal and calculate the crest factor. Select the filtered signal with the highest crest factor.
4. Generate a plot with two subplots, showing the original and the filtered signal, respectively, for 100 ms.
5. Take the absolute value of the filtered signal (function `abs()`), and apply a 500 Hz lowpass filter with 127 coefficients. Plot the demodulated signal for 100 ms.
6. Use the function `plot_spectrum_500` from Section 4.1 to plot the spectrum of the demodulated signal. Add vertical lines at integer multiples of BPFO. Can you see a spectral line at BPFO?
7. Repeat steps 3 to 6 for InnerRaceFault.xlsx.