University of South-Eastern Norway

Introduction to Complex Exponentials Direction Finding

TSE2280 Signal Processing. Lab 1

Spring 2025

1 Introduction

This lab is a modified version of the lab *Lab P-3: Introduction to Complex Exponentials – Direction Finding*[1] that accompanies the course text-book *DSP First* by McClellan et al [2]. The original lab has been converted from Matlab to Python and some of exercises have been changed.

The lab demonstrates concepts from Chapters 2 and 3 in the text-book. The intention is to give a better understanding of sinusoidal signals and how they are described by complex amplitude vectors, *phasors*.

The exercise was made for the course *TSE2280 - Measurements and Signal Processing* taught at the University of South-Eastern Norway.

1.1 Aim

The first part of this lab gives training in how to create and manipulate sinusoidal signals using complex exponentials in Python. The last part uses this to estimate the direction of a sound source from the phase difference between signals received by two microphones. This type of direction estimates is the basis for steering and focusing ultrasound beams with *phased arrays* in sonar and medical ultrasound. The same principle is also used to steer radar beams and in wifi and 5G antennas.

This lab is a computer simulation only, but can be extended to a real situation withou much extra equipment. Record a sound with two microphones using the stereo channels in your sound card, and then test the method on the recorded signal.

1.2 Software Tools: Python with Spyder and JupyterLab

The programs for the lab shall be written in Python using the modules NumPy for representing signals and Matplotlib for plotting graphs. Later exercises will use the signal processing module in SciPy (`scipy.signal`) but SciPy is not needed in this exercise.

Python's module for complex numbers cmath can make the code easier to read. But be aware that cmath handles scalar numbers only. The signals are more conveniently represented by arrays as in NumPy. The recommended setup of libraries for this exerceise is shown in Table 1

The recommended Python programming environment for this and the following labs is *Spyder*[3], which is included in the *Anaconda*[4] package management. However, any Python package management and programming environment should function.

All code files shall be included with the lab report. We recommend collecting everything, code, text and figures, into a single JupyterLab Notebook[5], but a pdf with separate Python files is also acceptable.

2 Theory

2.1 Sinusoidal Signals and Phasors

The aim of the first part of this lab is to obtain experience with complex numbers, and using phasors and complex exponentials to represent sinusoidal signals. Recall from the lectures that a sinusoidal signal

Table 1. Recommended format for importing the Python modules. NumPy is used to manipulate signals as arrays, Matplotlib to plot results. The complex math library cmath can be included to have simpler access to mathematical constants and functions, e.g., π, the complex exponential, and the square root.

```
import numpy as np
import matplotlib.pyplot as plt
from cmath import pi, exp, sqrt      # For readability, also covered by NumPy
```

Table 2. Overview of basic complex number operations in Python. Details are found in the documentation for NumPy. The function calls are prefixed by `np` in accordance with how NumPy was imported, see Table 1.

Command	Member	Description	Mathematical notation
`z = complex(2,3)`		Creates a complex number.	$z = x + jy = 2 + 3j.$
`z = 2 + 3j`		Same as above	
`1j`		Imaginary unit, i or j.	$i = j = \sqrt{-1}$
`np.conj(z)`	`z.conjugate()`	Complex conjugate.	$z^* = x - jy$
`np.abs(z)`		Absolute value.	$\|z\| = \sqrt{x^2 + y^2}$
`np.angle(z)`		Phase in radians.	$\angle z$
`np.real(z)`	`z.real`	Real part.	$\mathrm{Re}\{z\} = x$
`np.imag(z)`	`z.imag`	Imaginary part.	$\mathrm{Im}\{z\} = y$
`np.exp(1j*theta)`		Complex exponential.	$e^{j\theta} = cos\theta + j\sin\theta$

$x(t)$ is written as

$$x(t) = Acos(\omega t + \phi) = \mathrm{Re}\left\{Ae^{j(\omega t+\phi)}\right\} = \mathrm{Re}\left\{Xe^{j\omega t}\right\} \quad , \qquad X = Ae^{j\phi} \tag{1}$$

where A is the amplitude, $\omega = 2\pi f$ is the angular frequency, f is the frequency, and ϕ is the phase. $X = Ae^{j\phi}$ is a *phasor* or *complex amplitude* that includes both the amplitude and the phase of the signal. Analysis of sinusoidal signals like $x(t)$ is in general simpler by manipulating phasors as complex numbers compared to using the amplitude and phase separately.

2.2 Complex Numbers in Python with NumPy

Analysis of complex quantities in Python is simple. When using the library NumPy, complex numbers are handled almost exactly as real numbers, i.e. most of the usual mathematical operators and functions can be used on complex numbers.

Table 2 lists the basic operations in complex numbers in Python using NumPy. Note that all function calls in this table are prefixed by `np` due to the way NumPy was imported, see Table 1. Note also that a complex number in Python has three public members, `real`, `imag`, and `conjugate()`.

2.3 Adding Sinusiods using Complex Exponentials

Sinusoidal signals are most conveniently handled using complex exponentials. The theory behind this is given in Chapter 2 in the textbook and was presented in the lectures. It is briefly summarised here.

Look at a signal that is the sum of sinusoids that all have the same f_0, while the amplitude A_k and phase ϕ_k of the individual signals can be different,

$$x_s(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_0 t + \phi_k) \, . \tag{2}$$

The resulting summed signal can be found by summing the complex exponentials and then taking the real part. This is called *phasor summation* and is easier than using trigonometric identities,

$$x_s(t) = \mathrm{Re}\left\{\sum_{k=1}^{N} A_k e^{j\phi_k} e^{j2\pi f_0 t}\right\} = \mathrm{Re}\left\{\sum_{k=1}^{N} X_k e^{j2\pi f_0 t}\right\} \, . \tag{3}$$

The *complex amplitude* or *phasor* X_k is defined as

$$X_k = A_k e^{j\phi_k} \ . \tag{4}$$

The factor $e^{j2\pi f_0 t}$ in (3) is equal for all the individual signals, i.e., independent of k, so the amplitude A_k and phase ϕ_k of the summed signal $x_s(t)$ can be found by summing the complex amplitudes,

$$x_s(t) = \mathrm{Re}\left\{X_s e^{j\omega t}\right\} = A_s \cos(2\pi f_0 t + \phi_s) \ , \qquad X_s = \sum_{k=1}^{N} X_k = A_s e^{j\phi_k} \ . \tag{5}$$

The resulting signal will have the same frequency f_0 and period $T_0 = 1/f_0$ as the original signals.

2.4 Harmonics and Periodic signals

Consider now a signal $x_h(t)$ where the frequencies f_k of the individual cosine-waves are different, but are integer multiples of a fundamental frequency f_0,

$$f_k = kf_0 \quad , \qquad k = 0, 1, 2, \ldots \ . \tag{6}$$

The individual signals $\cos(2\pi k f_0 t + \phi_k)$ are called *harmonics* and the summed signal $x_h(t)$ can be written as

$$x_h(t) = \sum_{k=1}^{N} A_k \cos(2\pi k f_0 t + j\phi_k) = \mathrm{Re}\left\{\sum_{k=1}^{N} X_k e^{j2\pi k f_0 t}\right\} \ . \tag{7}$$

The periods T_k of the individual waves are

$$T_k = T_0/k \tag{8}$$

After the period $T_0 = kT_k$ of the fundamental frequency, the individual signals have repeated themselves k. Hence, all he frequency components will also be periodic with period T_0, and the resulting signal $x_h(t)$ will be periodic with period given by the fundamental frequency f_0 as $T_0 = 1/f_0$.

3 Programming Tips

3.1 Complex Numbers and Phasors

The Python file `zplot.py` contains functions to plot complex numbers as phasors, add them, and show the resulting phasor plots and sinusoids. The the description for the function `phasor()` in `zplot` is shown in Table 3.

3.2 Vectorization

NumPy module allows mathematical operations to be used on arrays. This is convenient when defining signals such as $x(t) = A\cos(2\pi f t + \phi)$. Here, the amplitude A and phase ϕ are scalars, while the time t is a vector spanning the time interval to be investigated.

Vectors spanning e.g., a time interval are created in NumPy by one of these methods

1) Specifying start, stop and step by the function `arange`.
   A time vector t is typically defined as `t = np.arange(t_start, t_end, dt)`,
   where `t_start` is the first point in the time-vector `t`, `t_end` marks the end of `t`, and `dt` is the interval between the time points. Note that the value `t_end` is not included in the time vector, `t` will end on the last point before `t_end`.

2) Specifying start, stop, and total number of points by the function `linspace`.
   A time vector t is typically defined as `t = np.linspace(t_start, t_end, n_points)`,
   where `t_start` is the first point in t, `t_end` marks the end of `t`, and `n_points` is number of points in the vector. Note again that the last value `t_end` is not included in the time vector, `t` will end on the last point before `t_end`.

3) Specifying start, stop, and total number of points by the function `logspace`.
   This is the same as `linspace`, but the numbers are evenly spaced on a logarithmic scale, and the start and end points are specified by their logarithms: `start=2` means the first value is 10^2=100.

Table 3. Function `phasor()` to show complex numbers as phasors in the complex plane (Argand-diagrams). The functions are found in the file `zplot.py` included for this lab.

```
def phasor(zk,
           include_sum=True,
           include_label=False,
           include_signal=False,
           frequency=1,
           ax=None):
    """Show complex amplitudes and resulting signals.

    Parameters
    ----------
    zk : complex or list of complex
        Complex numbers to show

    frequency=1 : Float, optional
        Frequency used to plot the signals

    include_sum=True : Boolean, optional
        Show the sum of the numbers in the plots

    include_label=False : Boolean, optional
        Label the phasors z1, z2, ...

    include_signal=False : Boolean, optional
        Include a plot of signals as function of time

    ax=None : Axes object of Matplotlib, optional
        Axis to plot the results. A new axis is created if not specified
    """
```

4 Training Exercises

4.1 Complex Numbers

The purpose of this task is to get used to complex numbers, how to visualize them as phasors, and how to use them in Python.

Reporting

Collect the answers and code in one notebook in Jupyter Lab and upload this to Canvas.

Exercises

1) Load the library `zplot` to visualize the results.
2) Enter the two complex numbers $z_1 = 2e^{j\pi/3}$ and $z_2 = -\sqrt{2} + 5j$.

   Use the Python functions or methods to find the real and imaginary part, $\mathrm{Re}\{z\}$ and $\mathrm{Im}\{z\}$, the magnitude $|z|$ and phase $\angle z$.

   Display z_1 and z_2 and the sum $z_1 + z_2$ as phasors with `zplot.phasor`. The input to `zplot.phasor` is specified as a `list`, this is done by enclosing the numbers in square brackets, e.g., `[z1, z2]`.
3) Find the complex conjugate z^* and inverse $1/z$ for z_1 and z_2 and plot them using `zplot`.

   Recall what you have learned about complex numbers in math courses. Are the results as expected?
4) Calculate the product $z_1 z_2$ and ratio z_1/z_2 and plot them using `zplot`.

   Are these results as expected?
5) Calculate the products of the conjugates, $z_1 z_1^*$ and $z_2 z_2^*$.

   Plot them in the same diagram as z_1 and z_2 and explain the result.

6) Calculate the sums $z_1 + z_1^*$ and differences $z_1 - z_1^*$ of the conjugates and plot them in the same diagram as z_1. Do the same for z_2. Explain these results.

4.2 Python Function to Generate a Sinusoid Signal

Reporting

Collect the answers and code in one notebook in Jupyter Lab and upload this to Canvas.

Exercises

1) Write a function (`def` in Python) that generates a single sinusoid, $x(t) = A\cos(\omega t + \phi)$ from the four input arguments amplitude A, frequency f, phase ϕ and duration. The angular frequency is $\omega = 2\pi f$.

   The function shall return the sinusoidal signal $x(t)$ and the time vector t where the signal is evaluated.

   The function shall generate exactly 32 values of the sinusoid per period.

   A skeleton of the function with the recommended function call and ducumentation string is listed in Table 4.

2) Demonstrate that your function works by plotting the output for the following parameters:

   $$A = 10^4 \qquad f = 1.5\,\text{MHz} \qquad \phi = -\angle 45 \qquad \text{Duration } 10^{-6}\,\text{s}$$

   Note that the phase must be converted to radians before calculating the result. What is the phase in radians in this case?

   Use Matplotlib to plot the results.

   Calculate the value of $x(t)$ at $t = 0$. Does this agree with the plot?

3) Calculate the period of the resulting signal and check that this agrees with the plot.

Table 4. Skeleton for a function to generate a cosine signal from amplitude, frequency, and phase. The first lines are the recommended function call and docstring. The last line specifies that the signal `x` and time vector `t` are returned.

```
def make_cos(A, f0, phase, duration):
    """Make a cosine-function from specified parameters.

    Parameters
    ----------
    A : float
        Amplitude
    f0: float
        Frequency [Hz]
    phase: float
        Phase [radians]
    duration: float
        Duration of signal [seconds]

    Returns
    -------
    x: 1D array of float
        Cosine-wave
    t: 1D array of float
        Time vector [seconds]
    """

    --- Your code comes here ---

    return x, t
```

4.3 Python Function to Generate a Sum of Sinusoid Signals

Signals are often described as a sum of sinusoids with different amplitudes A_k, frequencies f_k, and phases ϕ_k. Hence, it can be convenient to have a function that generates a signal from several cosine-functions, each specified by its amplitude, frequency, and phase.

Reporting

Collect the answers and code in one notebook in Jupyter Lab and upload this to Canvas.

Exercises

1) Write a function that generates a signal

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) = \sum_{k=1}^{N} X_k e^{j2\pi f_k t}. \tag{9}$$

The input arguments are the complex amplitude $X_k = A_k e^{j\phi}$, frequency f_k, sample rate f_s and the signal duration.

The function shall return the summed signal $x(t)$ and the time vector t where the signal is evaluated.

The frequencies f_k and complex amplitudes X_k shall be specified as NumPy arrays, and the function shall accept any number of frequency components.

Each frequency f_k shall match a complex amplitude X_k, so these vectors must have equal length. The resulting function must check for this and shall return an error message if the lengths are different.

A skeleton of the function with the recommended function call and header is listed in Table 5.

2) Demonstrate that your function works by plotting the output for a signal that is the sum of the following components.

k	Frequency f_k [Hz]	Complex amplitude X_k
1	0	10
2	100	$14e^{-j\pi/3}$
3	250	$8j$

Set the sample rate to $10\,000\,\mathrm{Samples/s}$, the duration of the signal to $0.1\,\mathrm{s}$, and the start time to $0\,\mathrm{s}$. Plot the result with Matplotlib.

3) Measure the period T_0 of the signal from the graph. Compare this to the periods T_k of the individual frequency components f_k.

Explain how the period of the summed signal can be calculated from the periods of the individual components.

4) Generate the signal

$$x(t) = \mathrm{Re}\Big\{-2e^{j50\pi t} - e^{j50\pi(t-0.02)} + (2-3j)e^{j50\pi t}\Big\}$$

over a time range that covers 3 periods.

Plot the signal x a function of time t.

5 Lab Exercise: Direction finding

The text in this exercise is taken from [1], with some small modifications.

Why do humans have two ears? One answer is that the brain can process acoustic signals received at the two ears and determine the direction to the source of the acoustic energy. Using sinusoids, we can describe and analyze a simple scenario that explains this direction finding capability in terms of phase differences (or time-delay differences).

This same principle is used in many other applications including radars that locate and track airplanes, and it gives the basis for phased array transducers used in medical ultrasound and sonar.

Table 5. Skeleton for a function to generate signal by summing cosine-functions with different complex amplitudes and frequencies. The first lines are the recommended function call and docstring. The last line specifies that the signal `x` and time vector `t` are to be returned. Note how the start time `t_start` is specified as an optional argument with default value 0.

```
def make_summed_cos(fk, Xk, fs, duration, t_start=0):
    """Synthesize a signal as a sum of cosine waves

    Parameters
    ----------
    fk: List of floats
        Frequencies [Hz]
    Xk: List of floats
        Complex amplitudes (phasors)
    fs: float
        Sample rate [Samples/second]
    duration: float
        Duration of signal [s]
    t_start=0 : float, optional
        Start time, first point of time-vector [seconds]

    fk and Xk must have the same lengths.

    Returns
    -------
    x: 1D array of float
        Signal as the sum of the frequency components
    t: 1D array of float
        Time vector [seconds]
    """

    --- Your code comes here ---

    return x, t
```

Reporting

Collect the answers and code in one notebook in Jupyter Lab and upload this to Canvas.

5.1 Exercises: Direction Finding with Microphones

Consider a simple measurement system that consists of two microphones that can both hear the same source signal. If the microphones are placed some distance apart, then the sound must travel different paths from the source to the receivers. When the travel paths have different lengths, the two signals will arrive at different times. Thus a comparison of the two received signals will allow us to measure the relative time difference (between peaks), and from that we can calculate the direction. If the source signal is a sinusoid, we can measure the travel time differences by measuring phases.

The scenario is illustrated in Fig. 1 where a vehicle travelling on the roadway has a siren that is transmitting a very loud sinusoidal waveform whose frequency is f_s=400 Hz. The roadway forms the x-axis of a coordinate system.

The two receivers (microphones) are located some distance away and are aligned parallel to the roadway. The distance from the road is y_r=100 m, and the receiver separation is d=0.4 m. The signals at the receivers must be processed to find the angle from Receiver M1 to the vehicle, which is denoted as θ in Fig. 1.

a) The delay from the sound is transmitted by the source to it is received by the microphone can be computed for both propagation paths. First, consider path r_1 from the vehicle to receiver M1. The time delay is the distance from the vehicle location at coordinate (x_v,0) to the receiver at coordinate (0,y_r), divided by the speed of sound c. The speed of sound in air can be set to c=340 m/s.

   Write a mathematical expression for the time delay as function of the vehicle position x_v. Call this delay t_1.

b) Write a mathematical formula for the time delay of the signal that travels path r_2 from the transmitter at coordinate (x_v,0) to receiver M2 at coordinate (d, y_r).

Call this delay t_2 and express it as a function of the vehicle position x_v.

c) The signals at the two receivers, $x_1(t)$ at M1 and $x_2(t)$ at M2, are delayed copies of the transmitted signal,

$$x_1(t) = s(t - t_1) \qquad\qquad x_2(t) = s(t - t_2)$$

where $s(t)$ is the transmitted sinusoidal signal.

Assume that the source signal $s(t)$ is a zero-phase sinusoid at f_0=400 Hz, and set the amplitude of the transmitted signal to A_s=1000.

Make a plot of $x_1(t)$ and $x_2(t)$ when the vehicle is at position x_v=100 m.

Plot 3 periods and measure the relative time-shift between the two received signals by comparing the peak locations.

d) How do we convert relative time-shift into the direction θ?

The distance from the microphones to the source is often much larger than the distance between the microphones. We can then ignore that the angles from M1 and M2 are slightly different, and assume

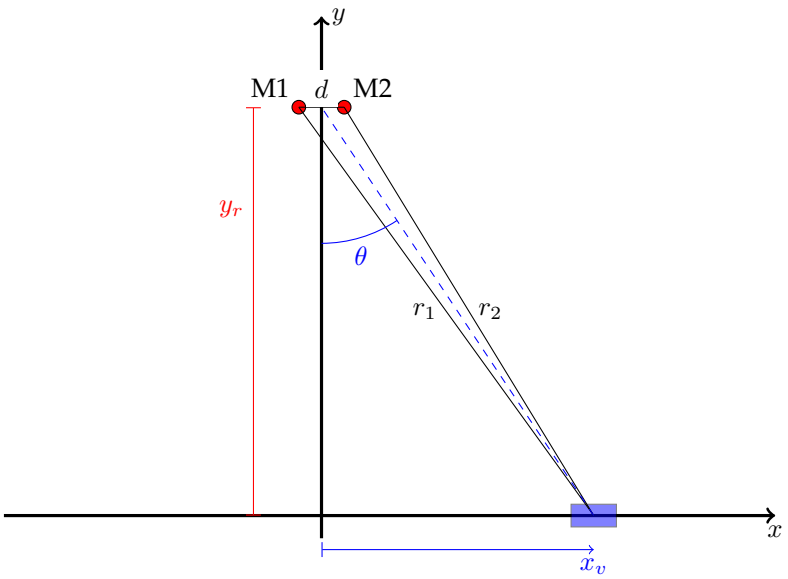

Figure 1. Direction finding using two microphones. A vehicle at position x_v travels along the x-axis while emitting a sound with frequency f_s=400 Hz. The sound is picked up by two microphones M1 and M2 positioned with spacing d=0.40 m. The difference in propagation distance $\Delta r = r_1 - r_2$ causes a phase-shift between the signals received by the two microphones. This phase shift can be used to estimate the direction to the vehicle, specified by the angle θ.

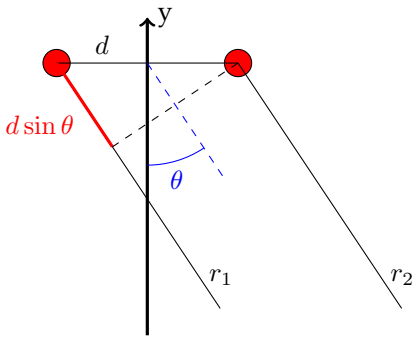

Figure 2. Zoomed-in version of Fig. 1 to show the difference in propagation distance $\Delta r = r_1 - r_2$ from the sound source to the two microphones. When the distance to the source is very long compared to the distance between the microphones, $r_1, r_2 \gg d$, the paths can be approximated as parallel and the difference is $\Delta r \approx dsin\theta$.

the paths to be close to parallel with the same angle θ. This situation is illustrated in Fig. 2, where we have zoomed in on the microphones in Fig. 1.

The difference Δr in propagation distance for the two paths r_1 and r_2 from the source to the microphones can now be approximated to

$$\Delta r = r_1 - r_2 = d \sin\theta \tag{10}$$

This is called the *far field approximation* and is often used to find the beam pattern from antennas, loudspeakers, ultrasound trasnducers, and other sources.

The propagation time t along the paths is given as $t = r/c$, giving the time-shift Δt between the signals arriving at the two microphones as

$$\Delta t = t_1 - t_2 = \frac{d \sin\theta}{c} \,, \tag{11}$$

which corresponds to a phase-shift $\Delta\phi$ between the two signal of

$$\Delta\phi = -2\pi \Delta t f_s = -2\pi \frac{d f_s \sin\theta}{c} = -2\pi \frac{d \sin\theta}{\lambda_s} \,, \tag{12}$$

where $\lambda_s = c/f_s$ is the wavelength of the emitted sound.

Calculate θ for the time-shift you found in the example above.

In addition, use geometry and the values of x_v and y_r to find the true value of θ, i.e., not using the approximation (12).

Compare the two results and compare your calculated value of θ to the true value.

e) The objective in the rest of this lab is to write a Python function that will process the received signals to find the direction θ.

Calculate the complex amplitudes $X_1 = A_1 e^{j\phi_1}$ and $X_2 = A_2 e^{j\phi_2}$ received at M1 and M2 as function of vehicle position x_v

Show that you can compute the phase difference between two phasors X_1 and X_2 by the equation

$$\Delta\phi = \angle\{X_1 X_2^*\}$$

where the superscript * denotes the complex conjugate. Use the ideas presented here, summarised in Fig. 2 and (12), to write a Python-function that will compute the direction θ from the complex amplitudes.

f) Run your function for the vehicle moving from x_v=$-400\,\mathrm{m}$ meters to x_v=$500\,\mathrm{m}$ in steps of one meter.

Compare the value of θ calculated from the phase difference to the true value of θ by plotting both on the same graph.

Comment the result.

References

[1] J. H. McClellan, R. Schafer, and M. Yoder, "Lab P-3: Introduction to Complex Exponentials - Direction Finding," tech. rep., 2016.

[2] J. H. McClellan, R. Schafer, and M. Yoder, *DSP First*. United Kingdom: Pearson Education Limited, 2nd ed., 2016.

[3] P. Raybaut, "Spyder," 2024.

[4] "Anaconda," 2024.

[5] {Project Jupyter}, "Jupyter Lab."