Lab 2: Complex numbers and phasors

1 Complex exponentials

1.1 Grading

This Lab consists of four exercises. Once you have submitted your code in Matlab Grader AND once the deadline has past, your code will be checked for correctness. Note here, that upon submission, your code is already subjected to some basic checks that are aimed to verify whether your code will compile; these basics checks don't say anything about the correctness of your submission. You can visit Matlab Grader again after the deadline (give the servers some time to do all the assessments; this might even take a few days) to see how well you did. In case Matlab Grader indicates you failed an exercise, this does not automatically imply that you failed the entire exercise. Each exercise is subjected to n tests, where the number of tests can vary between exercises. In case Matlab Grader indicates you failed the exercise, this means that not all tests were passed (e.g. in an exercise with 7 tests, you could have passed 6 and Matlab Grader will indicate you failed the exercise). Your grade is calculated based on the number of tests you passed and not on the number of exercises you passed.

1.2 Real signals

Manipulating sinusoidal functions using complex exponentials turns trigonometric problems into simple arithmetic and algebra. The goal of this laboratory is to gain familiarity with complex numbers and their use in representing sinusoidal signals such as $x(t) = Acos(2\pi ft+\phi)$ as complex exponentials $z(t) = Ae^{j\phi}e^{j2\pi ft}$. The key is to use the appropriate complex amplitude together with the real part operator applied as follows:

$$x(t) = Acos(2\pi ft + \phi) = \mathbb{R}e\{Ae^{j\phi}e^{j2\pi ft}\}$$

After a summarizing overview of both complex number theory and MATLAB structures, we will use some coding exercises to get some more hands on experience later on.

1.3 Sinusoid addition using complex exponentials

As a start, we will firstly review the complex exponential signal and the phasor addition property needed for adding cosine waves, all with the same frequency, $f = f_0[Hz]$. In the exercises later on, we will have to do several additions of sinusoidal functions. The Phasor Addition Rule shows how to add several sinusoids that have the same frequency f_0:

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

This sum is difficult to simplify using trigonometric identities, but by using Eulers formula ($e^{j\theta} = \cos(\theta) + j\sin(\theta)$), it can be transformed to a sum of complex numbers. The sum of sinusoids thus reduces to an algebraic sum of complex numbers.

$$\begin{aligned} x(t) = \sum_{k=1}^{N} A_k cos(2\pi f_0 t + \phi_k) &= \mathbb{R}e\{\sum_{k=1}^{N} A_k e^{j\phi_k} e^{j(2\pi f_0 t)}\} \qquad (2)\\ &= \mathbb{R}e\{\left(\sum_{k=1}^{N} A_k e^{j\phi_k}\right) \cdot e^{j2\pi f_0 t}\}\\ &= \mathbb{R}e\{\left(A_s e^{j\phi_s}\right) \cdot e^{j2\pi f_0 t}\}\\ &= A_s \cos\left(2\pi f_0 t + \phi_s\right), \end{aligned}$$

with the complex amplitude $A_s e^{j\phi_s}$ defined as:

$$A_s e^{j\phi_s} = \sum_{k=1}^{N} A_k e^{j\phi_k}. \tag{3}$$

From this we can conclude that, when adding different sinusoidal signals with the same frequency f_0, we will obtain a sinusoidal signal that has the same frequency f_0.

1.4 Complex numbers in Matlab

Matlab can be used to compute complex-valued formulas and also to display the results as vector or "phasor" diagrams. Matlab has several functions already included to work with complex numbers and phasors. Here are some of Matlab's built-in complex number operators:

`conj`	Complex conjugate
`abs`	Magnitude
`angle`	Angle (or phase) in radians
`real`	Real part
`imag`	Imaginary part
`i,j`	pre-defined as $\sqrt{-1}$
`x = 3 + 4i`	i suffix defines imaginary constant (same for j suffix)
`exp(j*theta)`	Function for the complex exponential $e^{j\theta}$

Each of these functions takes a vector (or matrix) as its input argument and operates on each element of the vector.

When unsure about a command, type `help command` or `doc command`.

Exercise 1 [6 tests]

Complex numbers can be represented in the complex plane, a plane with a real axis and an imaginary axis. When performing arithmetics on complex numbers, it can be insightful to visualise the effect of these arithmetics in the complex plane. In this exercise you should get familiar with complex numbers and learn how simple arithmetics, such as addition and multiplication, make the complex numbers behave in the complex plane. For this exercise it is therefore strongly advised to study the figures that the template script outputs and see how the complex numbers behave.

In the template solution for this exercise, fill in z1, z2, z4 and z5 as given in equations (4) and (5). z3 is the addition of z1 and z2 and z6 is the multiplication of z4 and z5. Also calculate the absolute value and angle of z4, z5 and z6 and see how they are related to each other.

$$\begin{aligned} z1 &= 2 + 3j \\ z2 &= -1 + 2j \\ z3 &= z1 + z2 \end{aligned} \tag{4}$$

$$\begin{aligned} z4 &= \tfrac{3}{2} \cdot e^{j \cdot \frac{\pi}{6}} \\ z5 &= 2 \cdot e^{j \cdot \frac{\pi}{3}} \\ z6 &= z4 \cdot z5 \end{aligned} \tag{5}$$

2 Matlab introduction

2.1 Vectorization

The power of MATLAB comes from its matrix-vector syntax. In most cases, loops can be replaced with vector operations because functions such as `exp()` and `cos()` are defined for vector inputs, e.g.,

```
cos(vv) = [cos(vv(1)), cos(vv(2)), cos(vv(3)), ..., cos(vv(N))]
```

where `vv` is an N-element row vector. Vectorization can be used to both simplify and speed up your code. If you have the following code that plots a certain signal,

```
M = 200;
for k = 1:M
  x(k) = k;
  y(k) = cos( 0.001 * pi * x(k)*x(k) );
end
plot( x, y )
```

then you can replace the for loop with a single vector operation and get the same result with 3 lines of code:

```
M = 200;
y = cos( 0.001 * pi * (1:M).*(1:M) );
plot( 1:M, y )
```

2.2 Functions

Functions are a special type of M-file that can accept inputs (matrices and vectors) and also return outputs. The keyword function must appear as the first word in the ASCII file that defines the function, and the first line of the M-file defines how the function will pass input and output arguments. The file name must be the same as the function name and the file extension must be a lower case "m" as in `generateCosine.m` for the code example shown below.[1]

```
function [xx,tt] = generateCosine(frequency, duration)
  tt = 0:1/(100*frequency):duration;
  xx = cos(2*pi*frequency*tt);
end
```

Notice the word "function" in the first line. The function has "frequency" and "duration" as inputs and "xx" and "tt" as outputs. Both of the outputs should appear in the left-hand side of at least one assignment line within the function body.

2.2.1 Default Inputs

In MATLAB you can make the last input argument(s) of a function take on default values in case these arguments are omitted in the function call. You can use the `nargin` operator to determine the number of passed arguments. The example below shows how `duration` can be made optional using `nargin`:

```
function [x,t] = generateCosine(frequency, duration)
  if nargin < 2
    duration = 3;
  end

  ...
end
```

[1]See Section B.5 in Appendix B of the 'Signal Processing First' book for more discussion.

Exercise 2 [2 tests]

Create a function that plots the cosine wave $x(t) = A\cos(\omega t + \phi)$, given the input values: *Amplitude* , *AngularFrequency* [rad/s], *Phase* [rad] and *Duration* [s]. Note that angular frequency $\omega = 2\pi f$. The function also has to output the values of t and $x(t)$, where t is the time and $x(t)$ is the value of the cosine at a given point t.

Furthermore, the function has to generate exactly 32 values of the sinusoid per period. This should result into a total of $32M + 1$ values, where M is the number of periods. (Hint: The total amount of values of a variable can be checked by having a look at the variables in the workspace. Here the size of the array of a variable can be seen. Alternatively, you can use the command "who x" which will show you the size of x.)

In this case, the function is not only used to generate the cosine wave, but also plots it. To this end, make sure that you include the plotting within the function body.

The function should be extensively tested in MATLAB to make sure you have the correct time units t.

2.3 Complex numbers: imaginary part

2.3.1 Euler's Formula

The conversion between phasors (time-dependent complex exponentials) and sinusoids can easily be performed by the following identity:

$$e^{j\theta} = \cos(\theta) + j\sin(\theta)\,. \tag{6}$$

This identity is called Euler's formula. It can be explained using the Taylor polynomials discussed in the Calculus course of quartile 1. Below the phasor is written as a Taylor polynomial:

$$\begin{aligned}
e^{j\theta} &= 1 + j\theta + \frac{(j\theta)^2}{2!} + \frac{(j\theta)^3}{3!} + \frac{(j\theta)^4}{4!} + \frac{(j\theta)^5}{5!} + \frac{(j\theta)^6}{6!} + \frac{(j\theta)^7}{7!} + \frac{(j\theta)^8}{8!} + \cdots \\
&= 1 + j\theta - \frac{\theta^2}{2!} - \frac{j\theta^3}{3!} + \frac{\theta^4}{4!} + \frac{j\theta^5}{5!} - \frac{\theta^6}{6!} - \frac{j\theta^7}{7!} + \frac{\theta^8}{8!} + \cdots \\
&= \left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \frac{\theta^8}{8!} - \cdots\right) + j\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots\right)
\end{aligned}$$

The final two terms inside the brackets can be simplified by making use of the Taylor polynomials of the sine and cosine functions:

$$\sin\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!}\theta^{2n+1} = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots \qquad \text{for all } \theta$$
$$\cos\theta = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!}\theta^{2n} = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots \qquad \text{for all } \theta$$

This results in Euler's formula.

Now by using Euler's expression we can write a cosine function as the real part of a phasor as follows:

$$\mathbb{R}e\left\{e^{j(\omega t+\phi)}\right\} = \cos(\omega t + \phi)$$

Another explanation from Euler's formula can be obtained from the polar representation of complex exponentials, as has been discussed during the lectures.

2.3.2 Imaginary part in MATLAB

Recall from exercise 5 of Lab 1 that plotting the function $w(x) = \frac{\sqrt[3]{10-x}-1}{\sqrt{4-x^2}}$ gave the following warning in MATLAB:

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

In the obtained plot, it can be seen that for $|x| < 2$ the function is in the real domain. At $|x| = 2$, $w(x)$ goes to infinity, because the denominator is equal to 0. For $|x| > 2$ the function switches from the real to the complex domain. Before the introduction of complex numbers, the assumption was made that the function did not exist, because a negative number underneath a square root was not possible. However, from complex numbers you have learned that the negative number of a square root is possible by describing the function in the complex domain. In the graph you can see that for $|x| > 2$ the real part of $w(x)$ is 0, while the imaginary part is nonzero.

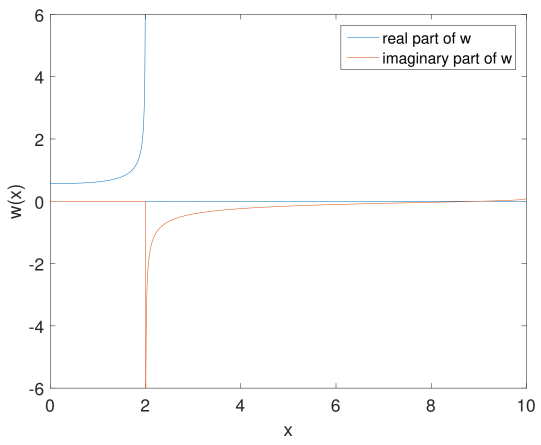
Hence, for $|x| > 2$ the following holds:

$$w(x) = \frac{\sqrt[3]{10-x}-1}{\sqrt{4-x^2}} = \frac{\sqrt[3]{10-x}-1}{\sqrt{-1}\cdot\sqrt{x^2-4}} = \frac{\sqrt[3]{10-x}-1}{j\sqrt{x^2-4}} = 0 - j\frac{\sqrt[3]{10-x}-1}{\sqrt{x^2-4}} \qquad \text{for } |x| > 2$$

By making use of the imag() function of MATLAB the answer to this function can be plotted in the same graph in orange as is shown in the following figure:

Exercise 3 [7 tests]

Consider the phasor $y(t) = Ae^{j(2\pi ft+\phi)}$ with amplitude $A = 1$, frequency $f = 1[Hz]$, and phase $\phi = 0$. Plot this phasor over the time domain from [0,2] seconds, with 201 measuring points over this range. Plot the phasor in blue



on the time domain and notice the warning message appear in the command window. The plot function ignores the imaginary part if the answer contains both a real and imaginary part. By making use of the imag() function, you can separate the imaginary part of the answer from the real part. Now add in the same plot the imaginary part of the answer in the color red. **Make sure to first plot the real part and after that the imaginary part!** If done properly, a real cosine signal and an imaginary sine signal should appear, as also described by Euler's formula.

Exercise 4 [17 tests]

Audible (sound) waves have frequencies ranging from 20 Hz to 20.000 Hz. This means that sound waves complete a period between 0.05 and 0.00005 seconds. When these waves travel through the air at approximately 344m/s, they travel approximately 2 cm up to 17 meters during one period.

In real-life, the sound waves will not always reach their target without reflecting of another object (e.g. the sound of a thunderstorm can reflect of tall buildings). Due to the reflection of the waves of different objects, some waves will travel more distance than others before reaching their target. So even when they are transmitted ”in-phase” (i.e. they have the same phase), when they reach the receiver the sound waves can have a phase difference due to different lengths over which they have propagated. This phase difference can affect the signal that is being received by the receiver.

This exercise will try to give you insight in the problems that can occur with the reflection of periodic signals.

Let's assume an example with a periodic signal transmitted by a source in all possible directions. In one direction, the signal will travel to the receiver via a straight line. In another direction, the sound wave will reflect from a surface and then travel to the receiver. Based on the difference in propagation distance the phase difference between the two received signals can vary.

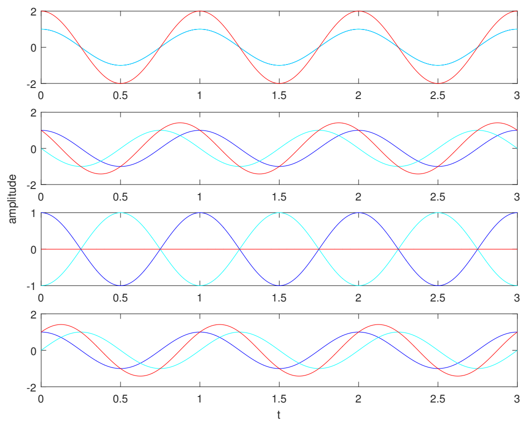
For this exercise you need to make a subplot, containing four different plots underneath each other. Each plot should have the time t on the horizontal axis, ranging from [0, 3] seconds with 301 measuring points. The directly received signal is represented by the real part of a phasor with a frequency $f = 1[Hz]$, a phase of $\phi = 0$ and an amplitude of $A = 1$ and needs to be plotted in every single one of the subplots. This signal should have the color blue. Next, a second signal needs to be plotted with a phase difference. This should represent the reflected signal. The amplitude and frequency are equal to those of the directly received signal, however, the phase is respectively 0, $\pi/2$, π, $3\pi/2$ for each of the subplots. This signal should be colored cyan. *NOTE: when the phase is positive, this means the signal will 'move to the left'*

Up to this point you should have four subplots underneath each other, with each one containing the directly received signal and the reflected signal with different phases in each of the plots.

Imagine the receiver, which receives both the reference and reflected signal at once. The received signal is then equal to the addition of the directly received and reflected signals. Plot this signal too in red for each of the four subplots.

Now add a grid to all the subplots and have a look at the amplitude of the received signal. What do you notice?

The script you wrote should give a figure similar to the figure below (note that the label along the vertical axis is not required in your solution). **For each subplot, the plotted graphs must be made in the following order: Directly received, Reflected and Addition!**


amplitude
t