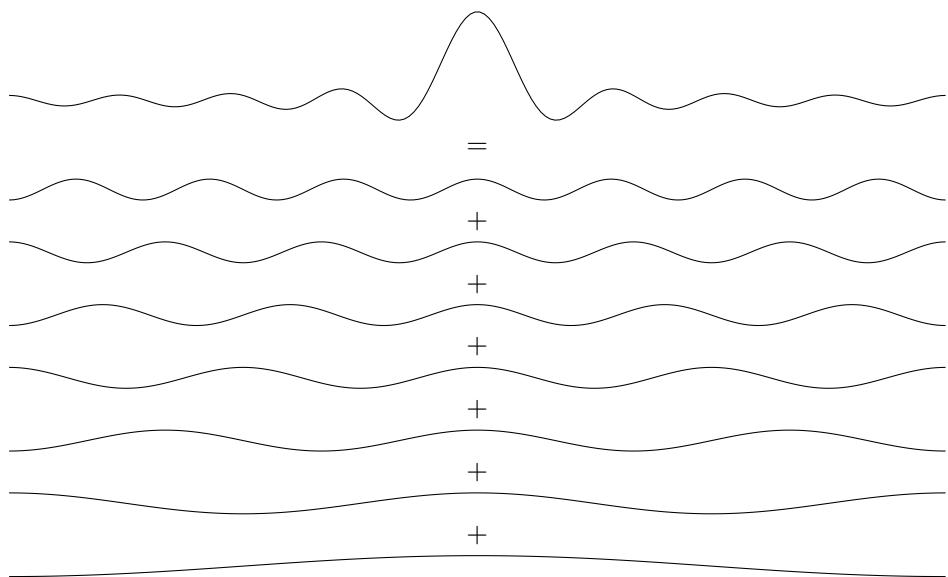


JUHA VIERINEN, JØRN OLAV JENSEN

SIGNAL PROCESSING



UNIVERSITY OF TROMSØ

Copyright © 2024 Juha Vierinen, Jørn Olav Jensen

PUBLISHED AS LECTURE NOTES.

http://github.com/jvierine/signal_processing_course

September 2024

Contents

1	<i>Syllabus</i>	7
2	<i>Introduction</i>	9
3	<i>Python</i>	15
4	<i>Complex Algebra</i>	23
5	<i>Signals and Systems</i>	33
6	<i>Elementary Signals</i>	51
7	<i>Sinusoidal Signals</i>	57
8	<i>Fourier Series</i>	77
9	<i>Programming Assignment 1</i>	111
10	<i>Fourier Transform</i>	113
11	<i>Discrete-time Signals</i>	137

12	<i>Linear Time-invariant Systems</i>	167
13	<i>Frequency Response</i>	197
14	<i>Programming Assignment 2</i>	215
15	<i>Discrete-time Fourier Transform (DTFT)</i>	221
16	<i>Ideal and Tapered Filters</i>	231
17	<i>Time-frequency Uncertainty Principle</i>	247
18	<i>Discrete Fourier Transform</i>	259
19	<i>Spectral Analysis</i>	277
20	<i>Arbitrary Frequency Response Filters</i>	291
21	<i>Programming Assignment 3</i>	305
22	<i>Z-transform</i>	315
23	<i>Infinite Impulse Response Filters</i>	341
	<i>Bibliography</i>	365

I'd like to thank the following people¹ for corrections and suggestions that have improved these lecture notes over the years: Björn Gustavsson, Patrick Guio, Mikkel Isak Gaup, Adrian Sletten³², Rikke Bjarnesen Andresen, Jostein Henriksen⁷, Daniel Nordahl Jørgensen, Ivan Mikheev, Oskar Marthinussen, Iver Martinsen, Marit Breimo, Sigurd Haugse, Ragnar Helgaas, Sondre Thomassen, Sofie Svenøe, Sigurd Yngvar Ekern, Vetle Hofsøy-Woie, Øyvind Alexander Larssen, Erlend Thorkildsen, Amalie Gjelsvik, Tommy Ryan, Eivind Dragset, Attiqa Abrar, Daniel Breiland Teigen, Sigrid Holm, Åse Fauske, Yvonne Johansen, Frank Martin Fossland, Runar Folke-Olsen, Morten Paulsen, Teodor Skotnes, Martin Stave¹⁶, Kristine Rein³, Anna Odh², Vegar Einarsen¹, Christian Salomonsen⁵, Jonas Riise¹, Jørn Jensen⁶³, Anton Zyranov¹, Nikolai Anfeltmo¹, Håkon Johansen¹⁰, Kian Sadeghi³, Johannes Bjørnhaug², Ines Seeliger², Dana King³, Emil Jettli¹, Sigurd Hanssen¹, Liza Liz¹, Sebastian Iversen¹, Daniel Johansen¹, Tobias W. Tobiassen¹, and Johanna Mankova Buseth¹. I'd also like to thank countless others whose names I have forgotten to mention. A number indicates how many corrections were found by each person. This is a very rough estimate based on my somewhat incomplete bookkeeping, which I only started midway through this project. All mistakes are purely mine.

¹ or their aliases, as some may have chosen not to use their real names when participating in the course commentary on Perusall.

1

Syllabus

Date	Topic	Special notes
19.08.	Intro, Python, Complex Algebra	
20.08.	Signals and Systems, Elementary Signals	
26.08.	Sinusoidal Signals	
27.08.	Fourier Series 1	
02.09.	Fourier Series 2	
03.09.	Fourier Transform 1	
09.09.	Fourier Transform 2	Programming Assignment 1, Jørn Olav Jensen
10.09.	Discrete-time Signals	Jørn Olav Jensen
16.09.	Sampling	
19.09.	Linear Time-Invariant Systems	Note unusual time!
23.09.	Frequency Response	
24.09.	Discrete-time Fourier Transform	
30.09.	No lecture, Høstferie	
01.10.	No lecture, Høstferie	Programming assignment 2
07.10.	Ideal and tapered filters	
08.10.	Time-frequency uncertainty principle	
14.10.	Discrete Fourier Transform	
15.10.	Spectral Analysis	
21.10.	Arbitrary Frequency Response Filter	Note unusual time!
22.10.	Laplace Transform	
28.10.	Z-Transform and Finite Impulse Response Filters	
31.10.	Z-Transform and Infinite Impulse Response Filters	
04.11.		
05.11.		
11.11.		Programming assignment 3
12.11.		
18.11.		
19.11.		
25.11.	Exam prep	
26.11.	Exam prep	
28.11.	Final Exam	

2

Introduction

MANY ASPECTS OF THE LIFE OF A MODERN HUMAN depend on the theory of signal processing. Here are a few examples. When you are listening to a digital audio recording of music, displaying an image on your computer, or using a wireless internet connection, you are relying on mathematical concepts that are taught in a basic course on signal processing. These same concepts are encountered when, e.g., solving differential equations, applying deep learning techniques, measuring the gravitational wave signature of colliding neutron stars with a laser interferometer, or making an image of a black hole using a worldwide network of radio telescopes. The theory of *signals and systems* is a collection of applied mathematical tools that have important applications in science, technology, and engineering. The term signal processing and the terminology of signal processing have origins in electrical engineering.

LET'S LOOK MORE CLOSELY AT THE AUDIO EXAMPLE. When you listen to a recording of music, you are probably relying on a digital representation of the acoustic signal that is *compressed*. With the help of signal processing mathematics, the audio signal is encoded in a special way, which allows the size of the file to be greatly reduced. This lowers the minimum internet speed required to stream the audio without annoying interruptions and allows 10 to 100 times more audio files to be stored on a digital storage device than without compression.

Audio compression algorithms often rely on the fact that the *frequency domain* or *spectral* representation of a signal is far more sparse than the time domain representation. This means that we can approximate the original audio signal using a sum of a relatively small number of periodic sinusoidal signal components. Let's say that the original audio signal is $x(t)$, which represents air pressure as a function of time. Using the Fourier series we can use the following

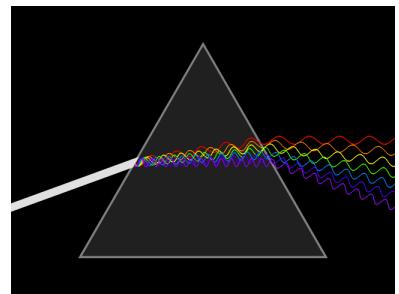


Figure 2.1: Light can be viewed as a superposition of electromagnetic waves with different amplitudes, phases, and frequencies. This can be investigated in practice with the help of a prism or a diffraction grating. My hope is that after taking this course, you will metaphorically be able to "see" arbitrary signals as a sum of periodic harmonic functions or *spectral components*.

type of sum to approximate $x(t)$:

$$x(t) \approx \sum_{n=1}^N A_n \sin(\omega_n t + \phi_n), \quad (2.1)$$

with each sinusoidal signal only described by three parameters: an amplitude $A_n \in \mathbb{R}$, a phase $\phi_n \in \mathbb{R}$, and an angular frequency $\omega_n \in \mathbb{R}$.

If the value of N in Equation 2.1 is sufficiently small, then the amount of information required to represent the audio signal as a sum of sinusoids can be significantly less than what would be required if the signal was just sampled with a fixed sample spacing and the amplitude of the waveform at each sampled point would be stored.

Here's an example. Let's say that you want to store 1024 *samples*¹ of audio signal. With the signal represented using 44100 samples per second, this would be approximately 0.023 seconds of signal. You'd need to store 1024 numbers without compression. With 16 bits per sample, this audio signal would require 16384 bits of storage space. However, if the signal is sufficiently well represented using $N = 10$ sinusoidal components, you'd only need to store 30 numbers to retain the information about the phase, amplitude and frequency of each sinusoidal component. This is only approximately 3% of the original data, or 480 bits of storage space at 16 bits per number².

Most of the videos, images, and music that you access over the internet is compressed in this manner. Without spectral compression techniques, the internet would most likely need to have at least 10 times more capacity than it currently has, in order to be able to provide us with the same amount of entertainment. For example, the MP3 audio compression standard and the JPEG image compression standard both utilize spectral compression methods.

WE OFTEN USE MORE GENERAL COMPLEX-VALUED SINUSOIDAL FUNCTIONS instead of real-valued ones when describing the theory of signal processing. These are defined as follows:

$$Ae^{i\phi}e^{i\omega t} = A[\cos(\omega t + \phi) + i \sin(\omega t + \phi)]. \quad (2.2)$$

Just like a real-valued sinusoid, the complex sinusoid is also represented using an amplitude $A \in \mathbb{R}$, phase $\phi \in \mathbb{R}$, and angular frequency $\omega \in \mathbb{R}$. Note that the above equation is just an application of Euler's formula³, which is probably the most frequently occurring equation in signal processing. If you master this equation, you'll have covered most of the mathematical prerequisites for this course!

If you are not familiar with complex algebra, Equation 2.2 might at first seem intractable. But don't worry. If you have a hard time



Figure 2.2: Leonhard Euler. Credit: Jakob Handmann (1753)

¹ One sample is a number representing the audio signal amplitude at one instant of time

² One "bit" of information is an answer to a yes or no question. One bit would allow you to say that a number is either 0 or 1. A two bit signal would allow you to say that a number is either 0, 1, 2, or 3.

³ The famous physicist Richard Feynman had this to say about Euler's formula: "We summarize with this, the most remarkable formula in mathematics. This is our jewel." (Feynman Lectures on Physics, Vol I). He had clearly benefited greatly from this formula.

grasping the function $Ae^{i\phi}e^{i\omega t}$, just think of it as a periodic function like $A \cos(\omega t + \phi)$, but with a real and imaginary sinusoidal component. The reason for using the complex sinusoidal signal instead of the real-valued one is that most of the mathematical calculations are greatly simplified.

A GENERAL FREQUENCY DOMAIN DECOMPOSITION OF SIGNALS IS KNOWN AS THE *Fourier transform*. This is just an extended version of the sum of sinusoidal signals given in Equation 2.1. The reverse and forward Fourier transforms allow transforming a signal between a time domain and a frequency domain representation:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega \quad (2.3)$$

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt. \quad (2.4)$$

The Fourier transform allows nearly any function to be expressed as an infinite sum of complex sinusoidal functions, or in other words, an integral. This is one of the main theoretical foundations of signal processing. In this equation, the term $x(t)$ is the time domain representation of the signal, and $\hat{x}(\omega)$ is the frequency domain representation of the signal. The function $\hat{x}(\omega)$ simply tells you what is the phase ϕ and amplitude A of each spectral component of the signal, assuming that it consists of infinitely many sinusoidal components.

Figure 2.3 shows a sketch of the basic idea behind the Fourier decomposition of signals. By adding together the seven individual sinusoidal signals on the bottom of the figure, we obtain the more pulse-like signal shown on the top of the figure. Conversely, the pulse-like signal on the top can therefore be decomposed into seven sinusoidal signals. This is in essence the idea behind the forward and reverse Fourier transform. Had we continued to sum together higher and higher frequency sinusoidal signals in the same manner, we would have ended up with an infinitely narrow spike $\delta(t)$, which is called the Dirac delta function. This special function has many important theoretical uses in signal processing.

The Fourier transform is named after Jean-Baptiste Joseph Fourier, who is shown in Figure 2.4. He was the first to come up with the idea of expressing a function as a sum of sinusoidal signals, or in other words, spectral components. This occurred in the 1820s while he was studying heat conduction. Using sinusoidal valued spectral components to represent the distribution of heat within a body allowed him to analytically solve the heat equation, because solving differential equations is relatively straightforward for sinusoidal functions. Even though this brilliant mathematical discovery occurred 200 years ago,

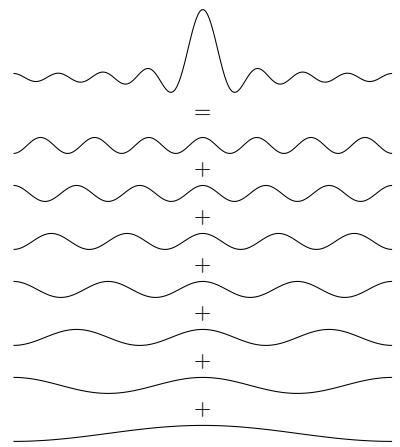


Figure 2.3: Spectral decomposition of a narrow pulse that consists of seven sinusoidal signals.



Figure 2.4: Jean-Baptiste Joseph Fourier

we are still coming up with new ways to apply it⁴. I would say that the ability to divide nearly any function you encounter into a superposition of simple elementary sinusoidal waves (the concept of the Fourier transform) is the most useful idea you will learn from a basic course on signal processing.

If you are not yet completely sure what the equations in this chapter mean, that is perfectly fine. Learning what the equations are and how to apply them in practice is the purpose of this course. If you are already familiar with complex sinusoidal signals and the Fourier transform, then you may be able to skip much of the introductory material and spend more time on the application examples.

SIGNAL PROCESSING IS AN INTERDISCIPLINARY FIELD, WHERE MATHEMATICS, PHYSICS, COMPUTER SCIENCE, AND ENGINEERING INTERSECT. The theoretical aspects of signal processing are essentially just applied mathematics, which often have already been introduced hundreds of years ago. However, applying the theory to solve real world problems involves programming or building electrical circuits. This aspect of signal processing is technology driven, and is continuously evolving as technology advances. This course will go through the theoretical foundations of signal processing without trying to skip over any important theoretical concepts, but the main emphasis will be in how these concepts can be applied to real world problems.

These lecture notes aim to teach theoretical and applied aspects of signal processing through mathematical derivations and programming examples. The mathematics we will use is not rigorous or generalized in a sense that it would appeal to a mathematician. Similarly, we will not emphasize aspects of proper software engineering. Instead, the aim is to provide programming examples that are as simple and as easy to understand as possible.

My hope is that these notes will take an approach to signal processing which is appealing to an undergraduate level physics or engineering student. I do not expect you to have ever encountered a Fourier transform before, to be familiar with complex algebra, or even be proficient in programming.

It is natural to divide the topic of signal processing into two main categories: *continuous-time* and *discrete-time*. The former deals with signals that are continuous functions. The latter deals with signals that are sequences of numbers, which is more natural for signals that reside on digital computers and that are used for digital signal processing. The latter type of signal processing has gained in importance for quite a long time due to the exponential increase in computing and storage capacity, which has made it possible to perform more

⁴ For example, your smartphone relies on the Fourier transform in several ways for enabling transfer of data over radio waves, and for storing pictures, videos, and audio files.

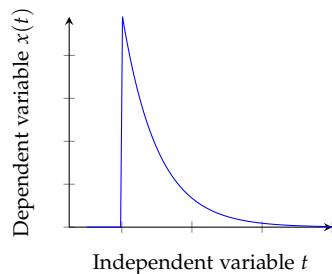


Figure 2.5: Continuous-time signal.

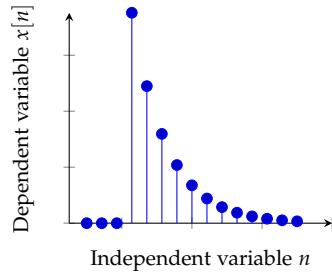


Figure 2.6: Discrete-time signal.

and more signal processing tasks digitally. The theory for continuous-time signals lays the theoretical foundation for also the discrete-time signal processing, so it is still important to teach this part.

THIS COMPENDIUM COVERS the topics in the syllabus of the undergraduate signal processing course “FYS-2006” held at University of Tromsø. While many other universities will have a separate “signals and systems” course that covers continuous-time signals, and a separate course dedicated to “digital signal processing”, this course combines both topics in one course.

These lecture notes are complete in terms of the topics that are covered in the course “FYS-2006”. You are not required to obtain any additional reading material. The reason why I went through the trouble of typing out these notes is that there currently are no signal processing textbooks that cover continuous-time and discrete-time signal processing, and use the Python programming language for examples. If you wish to deepen your knowledge, there are several published textbooks that you can use to complement this material. One book that I recommend is “Applied Digital Signal Processing: Theory and Practice” by D. G. Manolakis and V. K. Ingle.

YOU MIGHT BE ASKING YOURSELF: WHY SHOULD I LEARN ABOUT SIGNAL PROCESSING? This may be in the minds of, especially those for who have this as a mandatory course, and this is the primary reason that you are attending it. I believe that the elementary concepts taught in a basic course on this topic will provide you with the tools to be a good scientist. You will encounter the mathematical tools introduced in this course throughout science, technology, and engineering, where knowledge of these tools can often be seen as prerequisite background knowledge that allows you to understand more advanced concepts.

Perhaps the most powerful practical tool you will learn during this course is the Fast Fourier Transform (FFT), which is a specific algorithm used to compute a discrete Fourier transformation. It will allow you to perform a wide range of different calculations efficiently, using techniques that are not trivial unless you understand the basic concepts of signal processing. The FFT algorithm can be used e.g., for spectral analysis, evaluation of a convolution operation, to efficiently solve certain types of matrix operations, or to solve differential equations. Any time when processing signals later on in your life, there is a good chance that you will find the FFT algorithm to be highly useful.

If you go on to work with e.g. machine learning, this course will teach you the concept of convolution, which is a mathematical

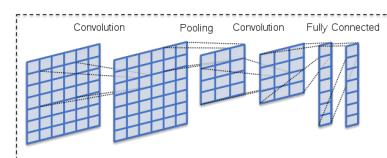


Figure 2.7: Simplified diagram of a convolutional neural network, where a convolution operation is one of the fundamental components. Figure adapted from [Maier et al., 2019].

operation that describes an arbitrary linear time-invariant system, and thus plays a major role in signal processing. You may have heard of convolutional neural networks, right? This course will not teach you anything about statistics or neural networks, but it will teach you the concept of a convolution sum, which is often a fundamental building block of image processing or deep learning algorithms⁵.

⁵ Andreas Maier, Christopher Syben, Tobias Lasser, and Christian Riess. A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik*, 29(2):86–101, 2019

3

Python

WE'LL DEMONSTRATE VARIOUS SIGNAL PROCESSING TOPICS USING PROGRAMMING examples. Why Python and not some other language? Here are three justifications:

- It is free for everyone to use,
- there are a wide range of efficient numerical libraries available,
- and Python is currently the language that students are most familiar with.

All the programming examples we'll show in these lecture notes will support Python 3. We will restrict ourselves to a bare minimum number of library dependencies. The main libraries you will need are: `numpy`, `scipy`, and `matplotlib`. These should be readily available for nearly every operating system.

If you don't know how to obtain Python for your computer, you could try out the Anaconda distribution of Python. It provides a relatively complete set of the most commonly used Python modules. You can obtain this distribution by visiting the anaconda web page: <http://anaconda.org> and downloading the installer for your platform.

The Anaconda package manager will allow you to install NumPy, SciPy, and Matplotlib. Anaconda also comes with a Python development environment as well. If you don't have your mind set on a specific programming editor yet, you could try out the Spyder development environment that comes with Anaconda. Another popular development environment is Microsoft's Visual Studio Code.

If you are using Linux, like I am most of the time, it is probably easiest to just obtain Python using the system package manager. On Ubuntu, you would install the necessary packages with the following command:



Figure 3.1: The Python programming language is an open source language that is increasingly popular for data science and signal processing.

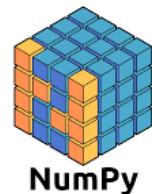


Figure 3.2: The NumPy package implements a large collection of numerical routines that can be used for signal processing.



Figure 3.3: The SciPy package contains a number of signal processing routines for Python.



Figure 3.4: Matplotlib implements basic plotting routines for Python.

```
sudo apt-get install python3 python3-numpy python3-scipy python3
    -imageio python3-matplotlib ipython3
```

Listing 3.1: Installing Python on Ubuntu Linux

After installing Python and the required modules, I recommend that you write a short test program to make sure that all the modules are installed correctly. To test your Python installation, open up your programming editor and write a small Python program:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as c

print("Hello World!")
# Test numpy and scipy.
print(np.pi)
print(c.pi)

sample_rate = 44100.0 # 44100 samples per second.
t = np.arange(100)/sample_rate
csin = np.exp(1j*2.0*np.pi*440.0*t) # A 440 Hz signal.

plt.plot(t*1e3, csin.real, color="blue", label=r"$\mathrm{Re}\{z(t)\}$")
plt.plot(t*1e3, csin.imag, color="red", label=r"$\mathrm{Im}\{z(t)\}$")

plt.xlabel("Time (ms)")
plt.ylabel(r"$z(t)=e^{i\omega t}$")
plt.legend()
plt.show()
```

Listing 3.2: 001_hello_world/hello.py

This test program prints out the standard hello world message, and uses NumPy and SciPy to print the approximate value of π . The program then plots a complex sinusoidal signal with the help of NumPy and Matplotlib. If you can run this program without getting any errors, you are all set. The output should look something like this:

CODE EXAMPLES that are spread throughout these lecture notes can be downloaded from GitHub: https://github.com/jvierine/signal_processing_course. The easiest way to access the code is to just visit this page with your web browser. In order to download the source code for the examples in these lecture notes, you can also use git on the command line:

```
git clone https://github.com/jvierine/signal_processing_course
```

Listing 3.3: Obtaining the source code for the programming examples with git



ANACONDA

Figure 3.5: The Anaconda distribution is currently one of the most popular distributions of the Python programming language, offering nearly all existing open source modules.

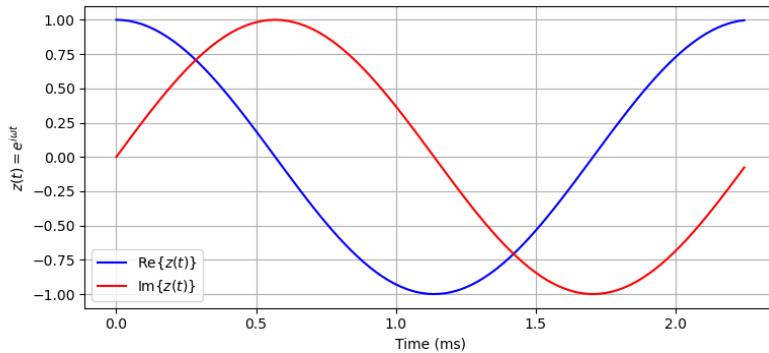


Figure 3.6: Output of the `hello.py` test program.

After obtaining the source code, you will have a repository that contains the source code for these lecture notes and all of the programming examples. The programming examples are under the code subdirectory of the repository.

If you are using Anaconda, you will have to install git (if you don't have it already) by typing in

```
conda install -c anaconda git
```

Listing 3.4: Obtaining the source code for the programming examples with git

Then you can proceed with the command in Listing 3.3.

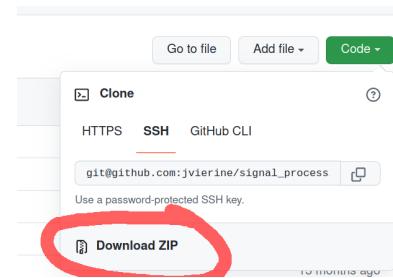


Figure 3.7: If you are not familiar with git, you can easily download the code from the GitHub webpage as a zip file instead of using the git clone command.



Figure 3.8: Program examples from these lecture notes are available on GitHub.

Exercises: Python

These exercises are intended to get you started with programming with *Python* and the *NumPy* and *Matplotlib* libraries. If you are already familiar with these topics, you may want to skip these exercises.

1. Obtain the source code for the examples by cloning the GitHub repository for this course. Locate the directory `code/001_hello_world/hello.py` program.
2. The Hello World program shown in Listing 3.2 plots a complex sinusoidal signal.
 - a) Go on the NumPy website <http://numpy.org>, and find the documentation of the functions included in the package.
 - b) Use this documentation to determine what `numpy.arange`, and `numpy.exp` do.
 - c) Modify the code so that it plots a circle on the complex plane, by plotting the real part of the complex sinusoidal signal `csin` on the x-axis, and the imaginary part on the y-axis.
 - d) Explain why the real and imaginary component draw a circle on the complex plane.
3. Use Python to calculate and print out the value of $e^{i\pi} + 1$. Note that i in Python, is denoted with `1j`.
4. The use of built-in NumPy *array functions* will let you program efficiently. The program shown in Listing 3.5 demonstrates the use of NumPy functions to evaluate the Mandelbrot set with a slight twist.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as c

x = np.linspace(-2.5, 1.5, num=2000, dtype=np.float32)
y = np.linspace(-2, 2, num=2000, dtype=np.float32)
cx, cy = np.meshgrid(x, y)
c = cx + 1j*cy

# Figure out what is the datatype of variable c
# to make sure it's complex64.
print(c.dtype)

z = np.zeros(c.shape, dtype=np.complex64)

for _ in range(12):
    z = z**2 + c

z[np.isnan(z)] = 0.0
```

```

z[np.isinf(z)] = 0.0

plt.imshow(np.abs(z), extent=[-2.5, 1.5, -2, 2], cmap="turbo",
           vmin=0,vmax=2)
plt.colorbar()
plt.xlabel(r"$\mathbf{Re}\{c\}$")
plt.ylabel(r"$\mathbf{Im}\{c\}$")
plt.show()

```

Listing 3.5: 001_hello_world/mystery.py

- Run the program shown in Listing 3.5. You should see a plot like the one shown in Figure 3.9.
- Describe what each line of the program does by adding comments to the code.
- It is possible to specify the data type of any NumPy array using the `dtype` attribute. There are two complex-valued datatypes available in NumPy: `complex64` and `complex128`. What are the pros and cons of using the `complex64` datatype instead of the `complex128` datatype?

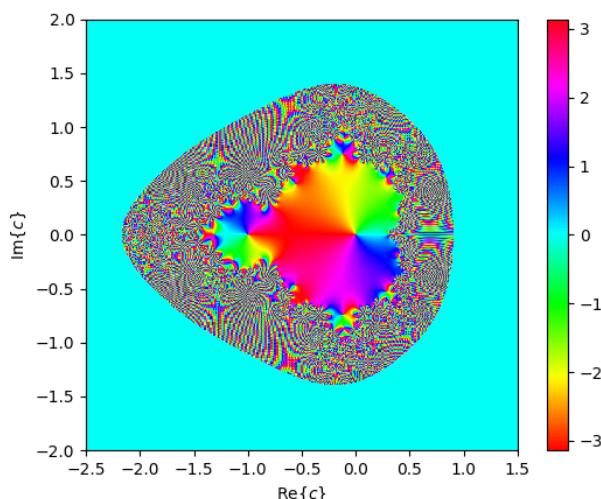


Figure 3.9: The Mandelbrot set example demonstrates the use of NumPy array functions and complex numbers. The plot shows the phase of complex number z_{12} after 12 iterations of $z_{n+1} \leftarrow z_n^2 + c$, starting with $z_0 = 0$.

Suggested solutions: Python

1. If you have git installed, run: `git clone https://github.com/jvierine/signal_processing` in the terminal. If you don't have git, you can follow the link and there should be a download option available on GitHub.
2. a) Go here <http://numpy.org> to read documentation.
 b) Reading the documentation you'll find that `np.arange` creates a NumPy array of evenly spaced numbers from 0 to the provided number minus 1. In the case of the code here, the `np.arange` function will sequence the numbers from 0 to 99 and return a NumPy array containing these numbers. See documentation: <https://numpy.org/doc/stable/reference/generated/numpy.arange.html>. The documentation for `np.exp` can be found here <https://numpy.org/doc/stable/reference/generated/numpy.exp.html>. The function simply computes e^x , where x can be real or complex, or even a NumPy array. If so, a NumPy array is returned.
 c) Listing 3.6 now plots a circle.

```

import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as c

print("Hello World!")
# Test NumPy and SciPy.
print(np.pi)
print(c.pi)

sample_rate = 44100.0 # 44100 samples per second.
t = np.arange(101)/sample_rate # <-- Added 101 to obtain
                             # the full circle.

csin = np.exp(1j*2.0*np.pi*440.0*t) # A 440 Hz signal.

# Make axis aspect ratio equal for visually pleasing output
plt.axes().set_aspect('equal')

# Plot a circle using the real and imaginary part of e^(i
# *2*pi*440*t).
plt.plot(csin.real, csin.imag, color="blue", label="Unit
circle")

plt.xlabel(r"\cos(2\pi440t)")
plt.ylabel(r"\sin(2\pi440t)")
plt.legend()
# Call this if needed.
# plt.show()

```

Listing 3.6: Code is modified to plot a circle

- d) The unit circle can be described by the equation $x^2 + y^2 = 1$, which is satisfied by $x = \cos(t)$ and $y = \sin(t)$.

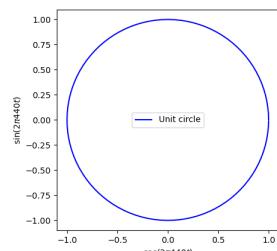


Figure 3.10: Output of Listing 3.6

3. Listing 3.7 shows how to compute and print $e^{i\pi} + 1$ in Python. The resulting output is $1.2246467991473532e-16j$, which is not zero. The reason for this is due to floating point errors. The value is very close to 0, but not exactly 0.

```
import numpy as np
print(np.exp(1j*np.pi)+1)
```

Listing 3.7: Computing $e^{i\pi} + 1$

4. The following is a solution for Exercise 4.

- Running the code generates the plot shown in the exercise.
- Adding comments in Python can be done using `#`, Listing 3.8 shows some comments.

```
# Import some libraries.
import matplotlib.pyplot as plt
import numpy as np
import scipy.constants as c

# Partition the interval (-2.5, 1.5) or (-2, 2) into 2000
# numbers.
# Separated by the same distance, store these numbers as
# 32-bit floating point numbers.
x = np.linspace(-2.5, 1.5, num=2000, dtype=np.float32)
y = np.linspace(-2, 2, num=2000, dtype=np.float32)

# Create a grid using x and y as the x- and y-axis.
cx, cy = np.meshgrid(x, y)
# ^cx is a copy of x stored as a matrix with y number of
# rows,
# same is true for cy, but opposite, there is a copy of y
# and there is x number of rows.

# Create a complex plane grid using cx and cy.
c = cx + 1j*cy

# Figure out what is the datatype of variable c
# to make sure it is complex64.
print(c.dtype)

# Create a matrix of zeros having the same dimension as c.
z = np.zeros(c.shape, dtype=np.complex64)

# Compute the sequence z_{n+1} = z_{n}^2 + c for n =
# 0,...,11.
for _ in range(12):
    z = z**2 + c

# Set infinities and NaN (not a number) to 0.0.
z[np.isnan(z)] = 0.0
z[np.isinf(z)] = 0.0

# Plot the result using a color gradient.
plt.imshow(np.angle(z), extent=[-2.5, 1.5, -2, 2], cmap="hsv")
```

```
plt.colorbar()
plt.xlabel(r"\mathrm{Re}\{c\}")
plt.ylabel(r"\mathrm{Im}\{c\}")
plt.show()
```

Listing 3.8: Commented Python code

- c) One can specify the datatype of a NumPy array. The data types supported can be found in the documentation. These include: `complex64`, `complex128`, `float32`, `int32`, among others. Pros and cons with the `complex64` and `complex128` include: size and accuracy. The `complex64` takes less memory, but is less accurate, while `complex128` can store more accurate numbers, but takes more memory.

4

Complex Algebra

COMPLEX ALGEBRA IS FOUND THROUGHOUT SIGNAL PROCESSING.

In this chapter, we'll briefly review the basics of this topic. Of primary importance is Euler's formula, which will be used extensively throughout this course.

EULER'S FORMULA RELATES AN ARBITRARY COMPLEX NUMBER $z \in \mathbb{C}$ TO AN EXPONENTIAL FUNCTION OF THE NATURAL NUMBER e and the sin and cos trigonometric functions as follows:

$$z = x + iy = Ae^{i\theta} = A[\cos(\theta) + i \sin(\theta)]. \quad (4.1)$$

This formula is useful, as it provides a relationship between the Cartesian and polar representation of a complex number¹.

In this formula, $A = |z| = \sqrt{x^2 + y^2}$ is the absolute value of the complex number z . This is sometimes called the *magnitude* or *modulus* of z .

The angle θ can be obtained with simple geometry $\theta = \tan^{-1}(y/x)$. The angle is also sometimes called the *argument* of z . We'll use the following notation to denote the argument of a complex number: $\angle z = \theta = \tan^{-1}(y/x)$. In some other texts you might run into the following notation: $\angle z = \text{Arg}\{z\}$.

It is worth pointing out here is that it is possible to add an integer multiple of 2π to θ and still get the same complex number:

$$Ae^{i\theta} = Ae^{i(\theta+2\pi k)} \quad (4.2)$$

This is due to the fact that $e^{i2\pi k} = 1$, where $k \in \mathbb{Z}$ is an arbitrary integer. The fact that there are infinitely many different solutions to the argument of a complex number is an important property, which will be encountered often in signal processing. For example, the concept of *aliasing* of discretized signals, which will encounter later on, occurs due to this this property.

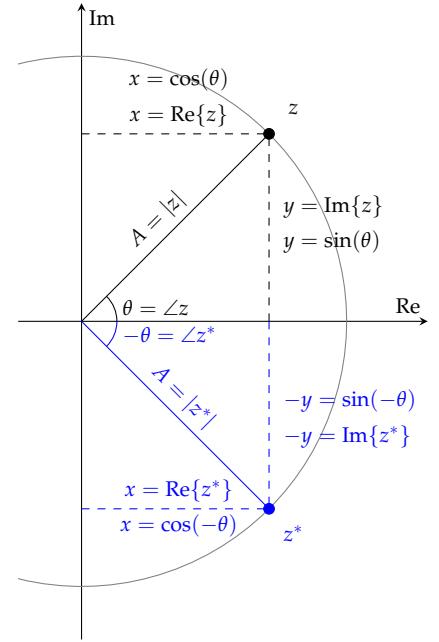


Figure 4.1: The polar representation of a complex number $z = x + iy = Ae^{i\theta}$ and its conjugate $z^* = x - iy = Ae^{-i\theta}$

¹ The proof of Euler's formula can be obtained in several different ways. We will use the derivative method demonstrated out by Youtuber MichaelPennMath. It relies on investigating that $f(\theta) = e^{-i\theta}e^{i\theta} = 1$ holds when $e^{i\theta} = \cos\theta + i \sin\theta$. Consider the following function:

$$f(\theta) = e^{-i\theta}[\cos(\theta) + i \sin(\theta)].$$

We know that $f(0) = 1$ just by relying on $e^0 = 0$, $\cos(0) = 1$ and $\sin(0) = 0$. The derivative of this function is zero everywhere:

$$\begin{aligned} \frac{df}{d\theta} &= e^{-i\theta}[-\sin(\theta) + i \cos(\theta)] \\ &\quad - ie^{-i\theta}[\cos(\theta) + i \sin(\theta)] \\ &= 0 \end{aligned}$$

From this, we know that $f(\theta) = f(0) = 1$. The function is constant everywhere.

With a little bit of algebra, we can see that because $f(\theta) = 1$, then $e^{i\theta} = e^{i\theta}f(\theta)$, and thus

$$e^{i\theta} = \cos(\theta) + i \sin(\theta).$$

This proves Euler's formula.

The term i in Equation 4.1 is the imaginary number, which has the following properties: $i = \sqrt{-1}$ and $i^2 = -1$. In engineering and programming, the symbol j is also often used for the imaginary number instead of i . The Python programming language uses the symbol j , and to denote e.g., $z = 2 + 5i$ in Python, you would use the code snippet `z=2+5j`.

The geometric representation of a complex number is shown in Figure 4.1, which shows the real and imaginary components of a complex number in a two-dimensional coordinate system— the complex plane.

The complex exponential obey the same exponentiation rules as the real exponential function:

$$\boxed{e^{z_1}e^{z_2} = e^{z_1+z_2}}. \quad (4.3)$$

for all complex numbers $z_1, z_2 \in \mathbb{C}$, moreover $(e^{z_1})^{z_2} = e^{z_1 z_2}$.

THE CONJUGATE z^* of complex number z is defined as:

$$z^* = x - iy \quad (4.4)$$

$$= A[\cos(\theta) - i \sin(\theta)] \quad (4.5)$$

$$= A[\cos(-\theta) + i \sin(-\theta)] \quad (4.6)$$

$$= Ae^{-i\theta}. \quad (4.7)$$

The conjugation operation flips the sign of the imaginary component. The geometric interpretation of the complex conjugate is shown in Figure 4.1. We'll use the superscript star notation to denote the conjugation operator.

THE COMPLEX CONJUGATE CAN BE USED TO OBTAIN THE MAGNITUDE OF THE COMPLEX NUMBER:

$$\boxed{|z| = \sqrt{zz^*} = \sqrt{x^2 + y^2}}. \quad (4.8)$$

A COMPLEX CONJUGATE CAN ALSO BE USED TO SELECT THE REAL AND IMAGINARY COMPONENTS OF A COMPLEX NUMBER as follows:

$$\boxed{\text{Re}\{z\} = \frac{1}{2}(z + z^*) = x} \quad (4.9)$$

and

$$\boxed{\text{Im}\{z\} = \frac{1}{2i}(z - z^*) = y} \quad (4.10)$$

These formulas are often encountered when dealing with real-valued signals. It is often easier to algebraically manipulate signals of the form $Ae^{i\theta}$, and after the calculations are done, it is simply a matter of

using equation 4.9 to extract the real component of the signal from its complex representation.

THE USE OF A SUM OF A COMPLEX NUMBER AND IT'S CONJUGATE CAN BE USED TO RELATE THE EXPONENT FUNCTION TO A COSINE AND SINE FUNCTION. Using Euler's formula for $z = e^{i\theta}$ and Equations 4.9 and 4.10, we can obtain:

$$\cos(\theta) = \frac{1}{2} (e^{i\theta} + e^{-i\theta}) \quad (4.11)$$

$$\sin(\theta) = \frac{1}{2i} (e^{i\theta} - e^{-i\theta}). \quad (4.12)$$

These relations are sometimes called the *inverse Euler* relations. You'll encounter these formulas when converting a cos or sin function into two complex exponent functions. The first step of a signal processing related calculation involving real-valued signals is often making this conversion, as functions of the form $Ae^{i\theta}$ are significantly easier to deal with.

COMPLEX MULTIPLICATION CAN BE VIEWED AS MULTIPLICATION OF MAGNITUDES AND SUMMATION OF PHASES. Let's express two complex numbers in polar form as $z_1 = A_1 e^{i\theta_1}$ and $z_2 = A_2 e^{i\theta_2}$. We can now see that multiplication with complex numbers has an intuitive interpretation.

$$z_1 z_2 = A_1 e^{i\theta_1} A_2 e^{i\theta_2} = \underbrace{A_1 A_2}_{A_3} \underbrace{e^{i(\theta_1 + \theta_2)}}_{e^{i\theta_3}} = A_3 e^{i\theta_3}. \quad (4.13)$$

When multiplying two numbers, the resulting angle is a sum of the two angles $\theta_3 = \theta_1 + \theta_2$, which can be also seen as a rotation of the point indicated by a complex number z_1 by angle θ_2 on the complex plane. The new magnitude is the magnitudes of the two numbers multiplied together $A_3 = A_1 A_2$. Figure 4.2 demonstrates multiplication geometrically on the complex plane.

RAISING A COMPLEX NUMBER TO THE n TH POWER can be seen as exponential scaling and rotation. Consider a complex number

$$z = A e^{i\theta} \quad (4.14)$$

where $A \in \mathbb{R}_{\geq 0}$ and $\theta \in \mathbb{R}$. If we raise this to the n th power, we get:

$$z^n = A^n e^{i\theta n} = A^n [\cos(\theta n) + i \sin(\theta n)]. \quad (4.15)$$

Scaling and rotation is demonstrated in Figure 4.3.

HERE ARE SOME PYTHON EXAMPLES OF COMPLEX NUMBER OPERATIONS.

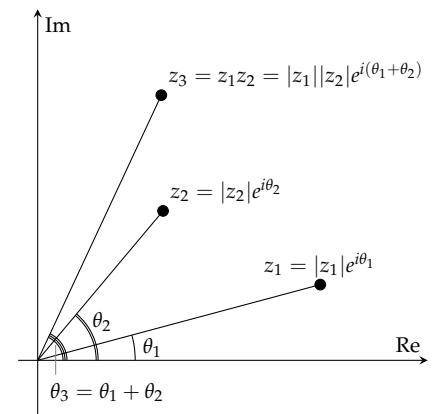


Figure 4.2: Multiplication of two complex numbers can be thought of as scaling and rotation.

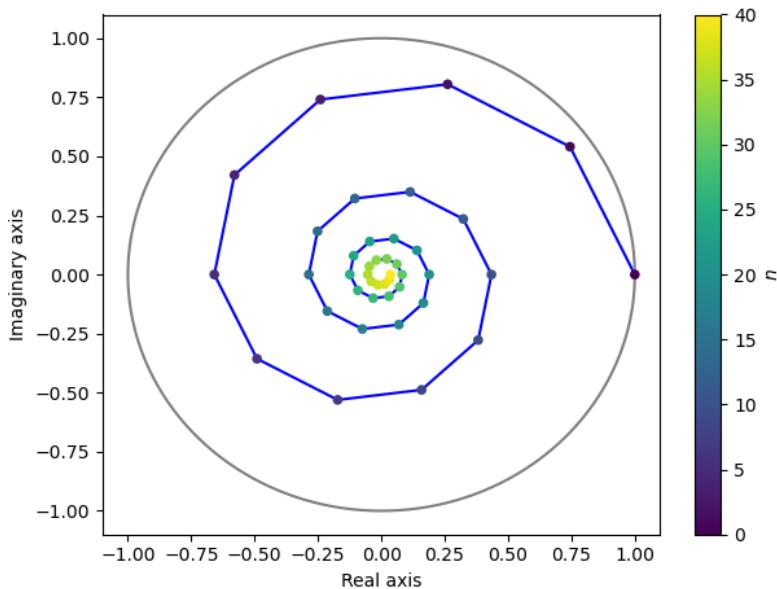


Figure 4.3: A spiral is formed by evaluating z^n with integer values of n between 0 and 41. In this case $z = 0.92e^{j2\pi/20}$. The parametric curve $e^{j\theta}$ with $\theta \in \mathbb{R}$ draws a circle in the complex plane, which is depicted with a gray color. The code that generated this plot can be found in `006_spiral/spiral.py`.

```
import numpy as np

# Initialize a complex number variable z.
z = 1.0 + 0.5j
# Get the real component of z
z.real
# and imaginary component of z.
z.imag
# The absolute value of z.
np.abs(z)

# What is i^{-1}?
z2 = 1j**(-1)

# Create complex numbers using the complex exponential function.
z1 = np.exp(1j*np.pi)
z2 = np.exp(1j*np.pi/2.0)

# Complex multiply.
z3 = z1*z2
# Complex conjugation.
z_squared = z1*np.conj(z1)

# Determine phase angle.
theta3 = np.angle(z3)
```

Listing 4.1: `008_complex_ops/ops_example.py`

Exercises: Complex Algebra

1. Prove $e^{i\pi} + 1 = 0$ using Euler's formula.
2. Use Euler's formula to write $1/i$ into polar form $Ae^{i\phi}$ with $A, \phi \in \mathbb{R}$. What is the phase angle ϕ ?
3. Show that i^i is real-valued using Euler's formula. Use Python to calculate this value and verify that it is real.
4. Use Euler's formula to show that de Moivre's formula is valid for $n \in \mathbb{Z}$:

$$[\cos(x) + i \sin(x)]^n = \cos(nx) + i \sin(nx)$$

5. Using Euler's formula, it is possible to determine the n th root of unity. What this means is that we look for all unique values of z which satisfy the following equation where n is a positive integer:

$$z^n = 1. \quad (4.16)$$

You probably already know the answer in the case of $n = 2$, for which there are two solutions: $z_0 = 1$ and $z_1 = -1$ as both $1^2 = 1$ and $(-1)^2 = 1$.

Provide a general formula that gives n unique values of $z \in \mathbb{C}$ that satisfy Equation 4.16. How many unique solutions of z are there for each value of n ? Hint: remember that $e^{i2\pi k} = 1$ where k is an integer.

6. Consider the equation $z^5 = 1$, where $z \in \mathbb{C}$.
 - a) Find the five solutions w_k for $k = 0, 1, 2, 3, 4$, to the equation $z^5 = 1$.
 - b) Create a Python program to plot the solutions to this equation in the complex plane. Add lines to your plot which connect the point w_k to $w_{(k-1) \bmod 5}$. What is the shape drawn?
 - c) Add the unit circle to the plot you found in b).
7. Use the inverse Euler formula to convert $(1 - i)e^{-i\omega t} + (1 + i)e^{i\omega t}$ into the following form:

$$A \cos(\omega t + \phi)$$

with $A \in \mathbb{R}$. What is A and what is ϕ ?

8. Prove, using Euler's formula that:

$$\cos(3\theta) = \cos^3(\theta) - 3 \cos(\theta) \sin^2(\theta).$$

9. Use Euler's formula to prove the following trigonometric identity:

$$\cos(\alpha + \beta) = \cos(\alpha) \cos(\beta) - \sin(\alpha) \sin(\beta).$$

10. Another definition of e^z (the complex exponential) is by an extension of the Taylor series expansion of e^x (the real exponential), as follows

$$e^z := \sum_{k=0}^{\infty} \frac{z^k}{k!}, \quad \text{for } z \in \mathbb{C}.$$

That is, e^z is taken to mean this infinite series. Use this Taylor series expansion definition to show that:

$$e^{i\theta} = \cos(\theta) + i \sin(\theta). \quad (4.17)$$

To do this, compute the Taylor series expansion of $e^{i\theta}$.

Suggested solutions: Complex Algebra

1. Euler's formula:

$$e^{i\theta} = \cos \theta + i \sin \theta,$$

so for $\theta = \pi$,

$$e^{\pi i} = \cos(\pi) + i \sin(\pi) = -1,$$

as $\cos(\pi) = -1$ and $\sin(\pi) = 0$, hence $e^{\pi i} + 1 = 0$.

2. Multiply by i in the numerator and denominator to obtain:

$$\frac{1}{i} = \frac{i}{i^2} = -i.$$

Using Euler's formula $e^{i\theta} = \cos \theta + i \sin \theta$ we take $\theta = \frac{3\pi}{2}$, this gives a polar representation of $-i$ as:

$$-i = e^{\frac{3\pi}{2}i}.$$

Remark: Multiplication by i rotates any vector in the complex plane by $\pi/2$ counterclockwise and $(1/i)(i) = 1$, so $1/i$ rotates by $\pi/2$ clockwise, so our result makes sense.

3. Use Euler's formula: $e^{i\theta} = \cos \theta + i \sin \theta$, with $\theta = \pi/2$. Then we can write $e^{\frac{\pi}{2}i} = i$ giving us:

$$i^i = (e^{\frac{\pi}{2}i})^i = e^{\frac{\pi}{2}i^2} = e^{-\frac{\pi}{2}} \in \mathbb{R}.$$

Thus, i^i is a real number.

4. Let x be any real number and n be an integer. Using Equation 4.3

we have $(e^a)^n = \underbrace{e^a e^a \dots e^a}_{n \text{ times}} = \overbrace{e^a + a + \dots + a}^{n \text{ times}} = e^{an}$ therefore:

$$[\cos(x) + i \sin(x)]^n = (e^{ix})^n = e^{inx} = \cos(nx) + i \sin(nx),$$

as desired.

5. From the hint we have that $e^{i2\pi k} = 1$, which can be combined with the fact that $(e^a)^n = e^{an}$, giving that the solutions to $z^n = 1$ are of the form:

$$w_k = e^{\frac{2\pi ik}{n}}, \quad k = 0, \dots, n-1.$$

A simple check to see that this is what we want is that w_k satisfy:

$$(w_k)^n = (e^{\frac{2\pi ik}{n}})^n = e^{2\pi ik}.$$

This gives 1 since $e^{2\pi ik} = 1$ for all $k \in \mathbb{Z}$. This gives n unique solutions to the equation, $z^n = 1$ as expected by the fundamental theorem of algebra.

6. Referring to the previous problem.

- a) Have that $w_k = e^{2\pi ik/5}$ for $k = 0, 1, 2, 3, 4$.
- b-c) See Listing 4.2. The figure formed by the fifth roots of unity is a regular pentagon.

```
import matplotlib.pyplot as plt
import numpy as np

# Simple function to compute the kth order root of unity of
# degree m.
def root_unity(k: int, m: int) -> float:
    return np.exp(2*np.pi*k*1j/m)

# Loop from 0 to 5, recall that the % operator computes the
# remainder, giving 0-4, then 0 again.
roots = np.array([root_unity(k % 5, 5) for k in range(6)])

# Partition an interval of one period.
t = np.linspace(0, 2*np.pi, num=50)

# Used to plot the unit circle.
x = np.cos(t)
y = np.sin(t)

# Extract the real and imaginary components.
z1 = roots.real
z2 = roots.imag

plt.figure(figsize=(4, 4))
ax = plt.gca()
plt.plot(z1, z2)
plt.plot(x, y)
plt.grid(True)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Fifth roots of unity")
ax.set_aspect("equal")
# Call this if needed.
# plt.show()
```

Listing 4.2: Solution for Exercise 3.6

7. Have that:

$$\cos(\theta) = \frac{1}{2}(e^{i\theta} + e^{-i\theta}),$$

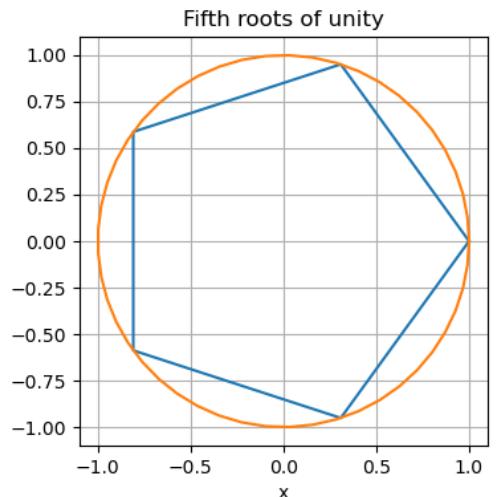
$$\sin(\theta) = \frac{1}{2i}(e^{i\theta} - e^{-i\theta}).$$

Using that $1/i = -i$ from Exercise 2 we get:

$$(1-i)e^{-i\omega t} + (1+i)e^{i\omega t} = (e^{i\omega t} + e^{-i\omega t}) - \frac{1}{i}(e^{i\omega t} - e^{-i\omega t}).$$

By the inverse Euler relations we may write this as:

$$(1-i)e^{-i\omega t} + (1+i)e^{i\omega t} = 2\cos(\omega t) - 2\sin(\omega t).$$



Next step is to write this as a pure cosine. That is, determine A and ϕ such that:

$$A \cos(\omega t + \phi) = 2 \cos(\omega t) - 2 \sin(\omega t).$$

Using $\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$ this can be rewritten as:

$$A \cos(\omega t) \cos(\phi) - A \sin(\omega t) \sin(\phi) = 2 \cos(\omega t) - 2 \sin(\omega t).$$

Therefore, A and ϕ must satisfy:

$$\begin{aligned} A \cos(\phi) &= 2, \\ A \sin(\phi) &= 2. \end{aligned}$$

In other words $A = \sqrt{2^2 + 2^2} = \sqrt{8} = 2\sqrt{2}$ and $\tan(\phi) = 1$. With ϕ in the first quadrant, this gives $\phi = \pi/4$. Hence:

$$(1 - i)e^{-i\omega t} + (1 + i)e^{i\omega t} = 2\sqrt{2} \cos\left(\omega t + \frac{\pi}{4}\right).$$

8. Replace the cosine to obtain:

$$\cos(3\theta) = \frac{1}{2}(e^{3i\theta} + e^{-3i\theta}) = \frac{1}{2}((e^{i\theta})^3 + (e^{-i\theta})^3).$$

When expanded the first term is:

$$(\cos \theta + i \sin \theta)^3 = \cos^3 \theta + 3i \cos^2 \theta \sin \theta - 3 \cos \theta \sin^2 \theta - i \sin^3 \theta,$$

and the second term:

$$(\cos \theta - i \sin \theta)^3 = \cos^3 \theta - 3i \cos^2 \theta \sin \theta - 3 \cos \theta \sin^2 \theta + i \sin^3 \theta.$$

Adding these together then gives:

$$\cos(3\theta) = \cos^3 \theta - 3 \cos \theta \sin^2 \theta.$$

9. Let $z_1 = e^{\alpha i}$ and $z_2 = e^{\beta i}$ be complex numbers. Then:

$$z_1 z_2 = e^{\alpha i} e^{\beta i} = e^{(\alpha+\beta)i} = \cos(\alpha + \beta) + i \sin(\alpha + \beta).$$

In addition, we have that:

$$e^{\alpha i} e^{\beta i} = (\cos \alpha + i \sin \alpha)(\cos \beta + i \sin \beta) = (\cos \alpha \cos \beta - \sin \alpha \sin \beta) + i(\sin \alpha \cos \beta + \cos \alpha \sin \beta).$$

Taking the real and imaginary parts we get that:

$$\begin{aligned} \cos(\alpha + \beta) &= \cos \alpha \cos \beta - \sin \alpha \sin \beta, \\ \sin(\alpha + \beta) &= \sin \alpha \cos \beta + \cos \alpha \sin \beta. \end{aligned}$$

10. Consider the complex exponential of the form $e^{i\theta}$, then by this new definition we have:

$$e^{i\theta} = \sum_{k=0}^{\infty} \frac{(i\theta)^k}{k!}.$$

Using this definition we have:

$$e^{i\theta} = \sum_{k=0}^{\infty} \frac{(i\theta)^k}{k!} = 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \frac{(i\theta)^4}{4!} + \dots$$

Using that $i^2 = -1$ we get:

$$e^{i\theta} = 1 + i\theta - \frac{\theta^2}{2!} - i\frac{\theta^3}{3!} + \frac{\theta^4}{4!} + \dots$$

Grouping terms with and without i this can be written as:

$$e^{i\theta} = \left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} + \dots\right) + i\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \dots\right).$$

The first terms correspond to the Taylor series for $\cos \theta$ and the other $\sin \theta$, hence we get Euler's formula:

$$e^{i\theta} = \cos \theta + i \sin \theta.$$

5

Signals and Systems

WHAT IS A SIGNAL AND WHAT IS A SYSTEM? By a *signal*, we mean an information carrying mathematical function. Any function that has a value as a function of one or more variables is in essence a signal. By a *system*, we denote a mathematical operation that modifies a signal. A system consists of the precise mathematical description of how a signal fed into a system is modified by the system to produce an output signal.

The definition of signals and systems are merely abstract concepts, which have gained acceptance in the engineering community. In the context of mathematics, signals could also be called functions or vectors. Systems would be called functions or operators. In computer science, signals are often treated as arrays of numbers in the memory of a computer and systems are algorithms and computer programs that operate on these arrays.

Signals

A **SIGNAL** is a mathematical function $x(t)$, which describes the value of a dependent variable x , as a function of an independent variable t . An independent variable “sweeps” through all possible values. The dependent variable is the variable that changes as a function of the independent variable and conveys information.

Here’s an example. When describing electric potential as a function of time $V(t)$, time t is the independent variable and the electric potential $V(t)$ as a function of time is the dependent variable. Time by itself does not convey information, but electric potential as a function of time does.

Physical “real world” signals are modeled as continuous-time signals. Examples include, amongst countless others:

- temperature,

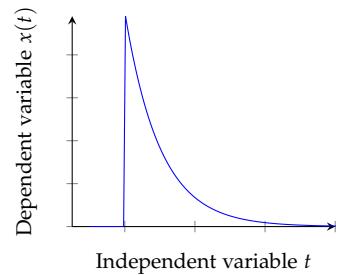


Figure 5.1: Continuous-time signal.

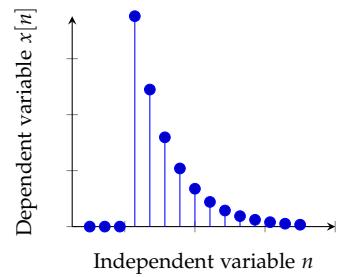


Figure 5.2: Discrete-time signal.

- density,
- pressure as a function of time and space (sound, seismic waves),
- electric field as a function of time and position (electromagnetic waves), or
- electrical current in a circuit.

Physics relies on differential calculus, with integration and differentiation as elementary operators. As we will later see, differential calculus can be studied through methods of signal processing, especially the spectral techniques and the Fourier transform are useful tools that can be applied in differential calculus.

This course will focus primarily on one-dimensional signals. These signals are complex-valued functions $x(t) \in \mathbb{C}$ of a real-valued argument $t \in \mathbb{R}$. This will naturally also cover the special case, where the signal is real-valued $x(t) \in \mathbb{R} \subset \mathbb{C}$.

By convention, we will refer to the independent variable of a signal as time, even though this variable doesn't necessarily have to indicate time. It can represent anything. For example, the independent variable can just as well be, e.g., distance.

Signals can be continuous or discrete. Following a commonly adopted practice, we will use round brackets for continuous-time signals (e.g., $x(t)$) and square brackets for discrete-time signals (e.g., $x[n]$).

In the case of discrete-time signals, the sample index $n \in \mathbb{Z}$ is unitless. A discrete-time signal is merely a sequence of numbers. The only way to associate meaning to this sequence of numbers is the a priori knowledge of how the signal was discretized. This allows us to, e.g., map the n th sample to a real-valued time.

An example of a continuous-time and a discrete-time signal is shown in Figures 5.1 and 5.2. When plotting signals graphically, it is customary (but not mandatory) to use the horizontal axis for the independent variable, and the dependent variable on the vertical axis.

To summarize, the two main types of signals that this course deals with are one-dimensional continuous-time $x(t)$ and one-dimensional discrete-time signals $x[n]$. Continuous-time signals are mappings from the real axis (time) to the set of complex numbers:

$$x : \mathbb{R} \rightarrow \mathbb{C} \quad (5.1)$$

Discrete-time signals are mappings from the set of integers to the set of complex numbers:

$$x : \mathbb{Z} \rightarrow \mathbb{C} \quad (5.2)$$

Signal processing of higher dimensional signals are essentially func-

tions of the form

$$x : \mathbb{R}^N \rightarrow \mathbb{C} \quad (5.3)$$

or in the case of discrete-time:

$$x : \mathbb{Z}^N \rightarrow \mathbb{C} \quad (5.4)$$

THE FIRST EXPERIMENTAL DETECTION OF GRAVITATIONAL WAVES using the Laser Interferometer Gravitational-Wave Observatory (LIGO) is shown in Figure 5.3. This signal is an example of a one-dimensional signal. The signal is thought to be caused by two black holes with masses around 30 solar masses merging together, 1.3 billion light-years from Earth. The independent variable on the x-axis is time, and the dependent variable is strain (stretching) of space that occurs due to a gravitational wave passing through the instrument. This is measured by comparing the relative lengths of two 4 km long laser interferometer arms.

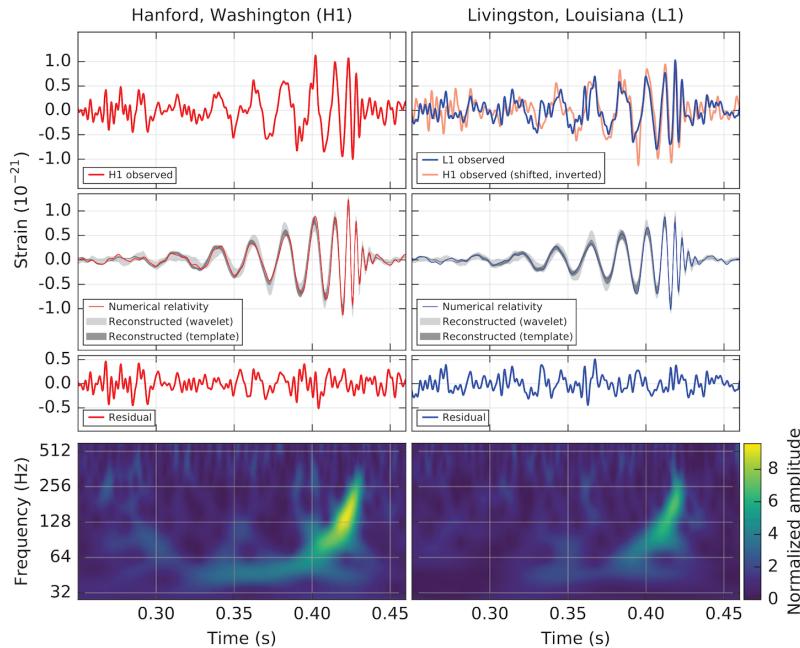


Figure 5.3: Two independent one dimensional signals measured by LIGO depicting strain. This is stretching of space due to a gravitational wave passing through two geographically separated laser interferometer observatories. One in Hanford, WA (left), and one in Livingston, LA (right). The figure is from: B. P. Abbott et al. (LIGO Scientific Collaboration and Virgo Collaboration) Phys. Rev. Lett. 116, 061102 – Published 11 February 2016.

SIGNALS CAN BE OF ARBITRARY DIMENSION. For example, an image is a 2d signal, where the dependent variable is intensity $I(x, y)$ measured as a function of two independent spatial variables x and y that indicate distance from origin along two orthogonal axes. An example of a discrete-time two-dimensional signal is shown in Figure 5.4. It represents an image of the emission from hot gas in the event horizon of a black hole around the M87 galaxy. The image is obtained

using a technique called very long baseline interferometry, which utilizes measurements from radio telescopes around the world. These measurements are combined to simulate a large telescope with the resolution equivalent to a telescope approximately the size of Earth.¹

Even higher dimensional signals can be used. Consider for example a video. It is a signal that contains the intensity of an image as a function of time. You can think of a moving picture (movie) as a three-dimensional signal: $I(x, y, t)$, where the intensity of the image is a function of two-dimensional position as well as time.

While we will focus primarily on one dimensional signals in order to keep things simple, many of the concepts we discuss can be generalized to multiple dimensions. Two-dimensional signal processing is called image processing, and it shares many of the same basic concepts with one dimensional signal processing. I will try to occasionally give examples of signal processing with higher dimensional signals.

¹ The Fourier transform, which is one of the central themes in this course, is a key part of the mathematics of very long baseline interferometric imaging.

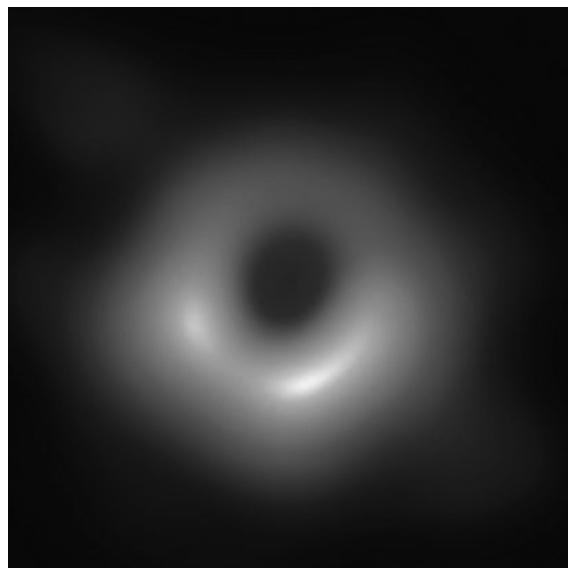


Figure 5.4: An example of a 2d signal: A Very Long Baseline Interferometric radio image of the black hole at the center of galaxy M87. The intensity depicts what is thought to be hot gas outside the event horizon of the black hole. Credit: Event Horizon Telescope collaboration et al. 2019.

DIGITAL SIGNALS on a computer, are also quantized. This means that there is only a finite number of possible values for the dependent value of a signal. This type of signal is called a *quantized* signal. The effects of quantization can often be treated as a random noise process that is added to the signal. When using sufficiently good bit depth to represent the signal value, this quantization noise can typically be ignored. We will not discuss quantization in this course.

Systems

A SIGNAL PROCESSING SYSTEM can be represented graphically as a block diagram, which describes signals going into a system, and signals going out of a system. An example is shown in Figure 5.5



Figure 5.5: A simple signal processing system block diagram.

A graphical representation is useful for understanding a signal processing system, especially if it is a complicated one, which includes many systems and signals.

The following block diagram is a real-world example of the block diagram of the EISCAT Svalbard radar transmitter subsystem.

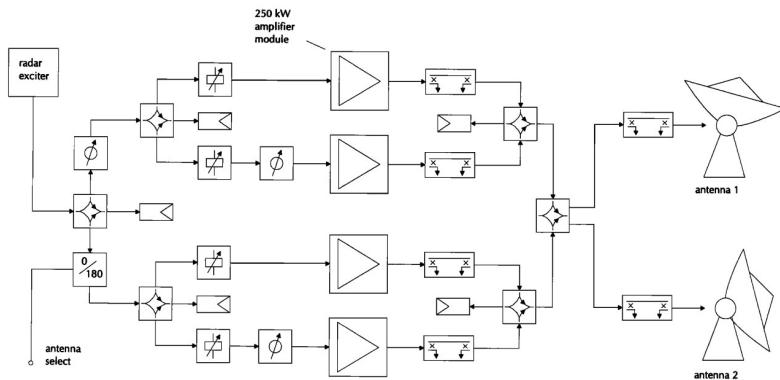


Figure 5.6: An example of a more complicated block diagram. From: Wannberg et.al., 1997.

The mathematical description of a system (what goes on in the box) is a general transformation of a signal. The mathematical notation for a continuous-time system in general is²:

$$y(t) = \mathcal{T}\{x(t)\} \quad (5.5)$$

and for discrete-time system:

$$y[n] = \mathcal{T}\{x[n]\} \quad (5.6)$$

An example of a system could be a function that delays the input signal by a delay τ :

$$y(t) = x(t - \tau). \quad (5.7)$$

Another example is a linear amplifier, which multiplies the amplitude of the signal with a constant $\alpha \in \mathbb{C}$:

$$y(t) = \alpha x(t). \quad (5.8)$$

² Here $\mathcal{T}\{\cdot\}$ is a mathematical description of how the input signal is modified by the signal processing system. This is also called an operator

You will often encounter both of these types of systems.

FOR EXAMPLE, WE CAN USE THESE TWO BASIC SYSTEMS TO MAKE A SIMPLIFIED MODEL OF A RADAR. Imagine that you have a radar. This radar sends out a waveform that is described by a waveform $x(t)$. Let's say that the time it takes an electromagnetic wave to travel from the radar transmitter to a point-like radar target and back is τ seconds. The received radar echo will be a delayed version of the transmitted signal, which is delayed by τ . In addition to this, the signal will be scaled, as a radar echo is typically much weaker than the transmitted signal.

Therefore, a very simple system that describes a radar echo is the following:

$$y(t) = \alpha x(t - \tau) \quad (5.9)$$

where $y(t)$ is the received radar echo signal.

We can further expand this concept to write the equation that describes a radar echo for a situation where there can be a continuous radar echo at time delays between 0 and T :

$$y(t) = \int_0^T \alpha(\tau)x(t - \tau)d\tau. \quad (5.10)$$

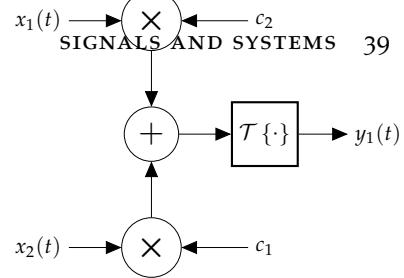
In this case, $\alpha(\tau)$ describes the radar echo amplitude as a function of propagation delay. This type of equation is called a *convolution* equation. You will encounter this type of equation very often in signal processing, and find that the Fourier transform is a very useful tool when dealing with convolution equations.

DIFFERENTIAL OPERATORS CAN ALSO BE SEEN AS SYSTEMS. Consider the first time-derivative of the continuous-time signal $x(t)$:

$$y(t) = \frac{d}{dt}x(t). \quad (5.11)$$

The system in this case is the time derivative operator $\frac{d}{dt}$. Later on, when we analyze the frequency response of systems, we will see that a derivative is a very basic high-pass filter, which reduces the magnitudes of low frequency components of a signal.

SYSTEMS ARE CLASSIFIED BASED ON THEIR MATHEMATICAL PROPERTIES. The most important two are: *linearity* and *time-invariance*. A system which is both linear and time-invariant is called linear time-invariant (LTI). Such systems have beneficial mathematical properties that make analysis and design of such systems very straightforward. We'll later on prove that LTI systems are fully characterized by something known as an impulse response. This is an important concept in signal processing.



Linear system

A system is linear, if a linear combination of inputs fed into the system yields the same as the linear combination of outputs:

$$\mathcal{T}\{c_1x_1(t) + c_2x_2(t)\} = c_1\mathcal{T}\{x_1(t)\} + c_2\mathcal{T}\{x_2(t)\} \quad (5.12)$$

for arbitrary constants $c_1 \in \mathbb{C}$ and $c_2 \in \mathbb{C}$ and arbitrary input signals $x_1(t)$ and $x_2(t)$. This property is highly useful, and appears throughout signal processing.

AN EXAMPLE OF A LINEAR SYSTEM is a system that scales the input signal by a constant factor α :

$$y(t) = \alpha x(t). \quad (5.13)$$

If $|\alpha| > 1$, the signal is amplified. This type of system would typically be called an amplifier. If $0 < |\alpha| < 1$, the system would be called an attenuator, as the output signal amplitude would be attenuated.

It is quite easy to determine that the test for linearity is passed for this system:

$$\alpha[c_1x_1(t) + c_2x_2(t)] = c_1[\alpha x_1(t)] + c_2[\alpha x_2(t)]. \quad (5.14)$$

AN EXAMPLE OF A NON-LINEAR SYSTEM is the following system, which obtains the absolute value of $x(t)$:

$$y(t) = |x(t)| \quad (5.15)$$

It is quite clear that this does not pass the test for linearity:

$$|c_1x_1(t) + c_2x_2(t)| \neq c_1|x_1(t)| + c_2|x_2(t)|, \quad (5.16)$$

for all possible values $c_1, c_2, x_1(t)$, and $x_2(t)$.

Time-invariant system

Let us first define a delay system $\mathcal{D}\{x(t)\} = x(t - \tau)$. Let us assume that the output of a system is $y(t) = \mathcal{T}\{x(t)\}$. This system is time-invariant if:

$$\mathcal{T}\{\mathcal{D}\{x(t)\}\} = \mathcal{D}\{\mathcal{T}\{x(t)\}\} \quad (5.17)$$

In other words, it does not matter if the signal is delayed before or after the system $\mathcal{T}\{·\}$ is applied to the signal. To check time-invariance, we need to verify Equation 5.17.

The test for time-invariance is illustrated as a block diagram in Figure 5.8. If you want, you could think of a time-invariant system as a system that satisfies:

$$[\mathcal{T}, \mathcal{D}]x(t) = 0,$$

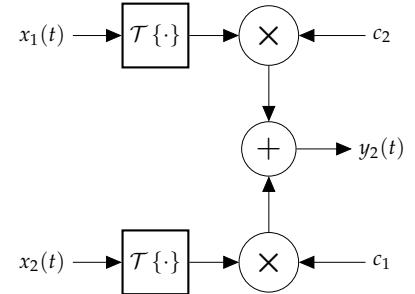


Figure 5.7: In order for the system specified by $\mathcal{T}\{·\}$ to be linear, $y_1(t) = y_2(t)$ must be satisfied.

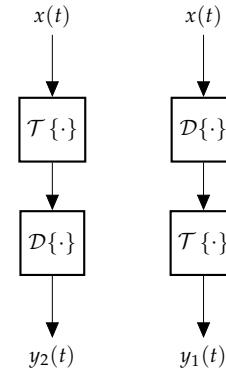


Figure 5.8: In order for the system specified by $\mathcal{T}\{·\}$ to be time-invariant, $y_1(t) = y_2(t)$ must be satisfied.

where $[\mathcal{T}, \mathcal{D}] = \mathcal{T}\mathcal{D} - \mathcal{D}\mathcal{T}$ is the commutator of two operators. Thus, a time-invariant system \mathcal{T} is a system that commutes with the time-delay operator $[\mathcal{T}, \mathcal{D}] = 0$.

AN EXAMPLE OF A TIME-INVARIANT SYSTEM is the system that returns the absolute value of the input signal $y(t) = |x(t)|$. We can see this by evaluating:

$$\mathcal{T}\{\mathcal{D}\{x(t)\}\} = \mathcal{T}\{x(t - \tau)\} = |x(t - \tau)| \quad (5.18)$$

$$\mathcal{D}\{\mathcal{T}\{x(t)\}\} = \mathcal{D}\{|x(t)|\} = |x(t - \tau)| \quad (5.19)$$

Time-invariance holds as the outputs are equal. It doesn't matter if a time-delay is applied to the signal before or after obtaining the absolute value of the input signal.

AN EXAMPLE OF A CASE THAT IS NOT TIME-INVARIANT is $y(t) = \mathcal{T}\{x(t)\} = \sin(t) + x(t)$. We can immediately see that the system directly depends on t , not only through the input $x(t)$, but also through the signal $\sin(t)$ that is added to the output signal. The formal test also shows that time-invariance is not met:

$$\mathcal{T}\{\mathcal{D}\{x(t)\}\} = \mathcal{T}\{x(t - \tau)\} = \sin(t) + x(t - \tau) \quad (5.20)$$

$$\mathcal{D}\{\mathcal{T}\{x(t)\}\} = \mathcal{D}\{\sin(t) + x(t)\} = \sin(t - \tau) + x(t - \tau) \quad (5.21)$$

In this case, $\mathcal{T}\{\mathcal{D}\{x(t)\}\} \neq \mathcal{D}\{\mathcal{T}\{x(t)\}\}$ so the system is not time-invariant.

Example: Overdriven amplifier

A simple model of a distorted guitar amplifier system would be the following clipping amplifier system, which is specified as follows:

$$y_d(t) = \begin{cases} -\beta & \text{when } \alpha x(t) < -\beta \\ \alpha x(t) & \text{when } |\alpha x(t)| \leq \beta \\ \beta & \text{when } \alpha x(t) > \beta \end{cases} \quad (5.22)$$

This is a very approximative model of an overdriven guitar amplifier. This type of system is often encountered in guitar music from the 50s and onwards. I'm sure that once you later implement this in practice, you'll recognize the sound that this system makes.

What does this system do? It amplifies the signal, but only up to a certain point. Beyond a certain absolute value of the input, the output maintains a constant positive or negative value. This type of behavior is also sometimes called clipping, and the effect is also sometimes called distortion, as the input signal amplitude is not linearly scaled, but it is rather distorted.

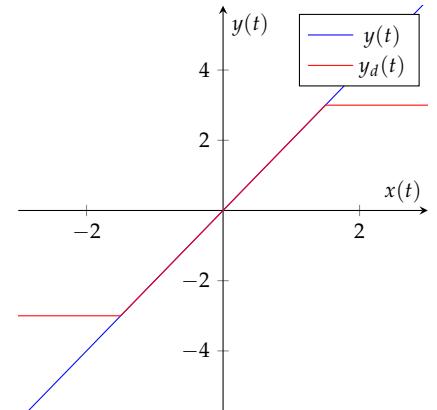


Figure 5.9: The system function of a linear amplifier $y(t)$ and a clipping amplifier system $y_d(t)$.

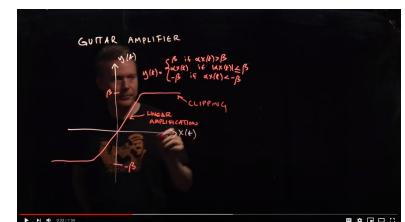


Figure 5.10: A video discussing an overdriven guitar amplifier can be found here: https://youtu.be/I30Mn_-yYF8.

Many real world amplifier systems have this type of saturation behavior. What this often means in practice is that the system is linear when the input absolute amplitude is less than some critical value, but beyond this, linearity no longer holds.

Figure 5.9 illustrates what $y_d(t)$ would look like as a function of $x(t)$. It is compared with a normal amplifier system $y(t) = \alpha x(t)$, which does not clip.

Exercises: Signals and Systems

1. Consider the following signal $V(t)$, which describes the electric potential of a circuit as a function of time. The instantaneous power flowing through this circuit is given by the following equation:

$$P(t) = R^{-1}[V(t)]^2.$$

It is possible to consider the function $R^{-1}[\cdot]^2$ that converts electric potential into power as a system in signal processing terminology.

- a) Is this system linear?
- b) Is this system time-invariant?

Prove your result.

2. A discrete-time system is defined as:

$$y[n] = \frac{1}{5} \sum_{k=0}^4 x[n-k]. \quad (5.23)$$

- a) Explain in words what this system does to the input signal $x[n]$.
- b) Is this system linear?
- c) Is this system time-invariant?

Use the tests for linearity and time-invariance to justify your result.

3. A time scaling system adjusts the scaling of the independent variable: $y(t) = x(\alpha t)$, when $x(t)$ is the signal fed into the system and $y(t)$ is the output.
 - a) What is the effect on the signal when $0 < \alpha < 1$?
 - b) What about $\alpha > 1$?
 - c) Is this system linear?
 - d) Time-invariant?
4. Prove that the time derivative operator $y(t) = \mathcal{T}\{x(t)\} = \frac{d}{dt}x(t)$ is a linear time-invariant system.
5. The guitar amplifier system given in Equation 5.22 is not a linear system. Prove this. Hint: you can use proof by counterexample, by finding a case where the requirement of linearity is not met.
6. Is the guitar amplifier system given in Equation 5.22 a time-invariant system? Prove your result by using the formal test for time-invariance.

7. The code in Listing 5.1 implements a linear amplifier system for an audio signal:

$$y(t) = \alpha x(t) \quad (5.24)$$

where the input signal is $x(t)$, the output signal is $y(t)$, and the amplification is α . You can find the code and audio file on GitHub.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sio

# Amplifier gain.
alpha = 10.0

# A function to compress signal x (signal processing system.)
def amplify(x, alpha):
    return alpha*x

# Read wav file (read only one stereo channel.)
#
# Guitar_clean.wav copyright
# Original author: LG downloaded from freesound.org,
# Original file name: Guitar clean rif.wav
wav = sio.read("guitar_clean.wav")
sample_rate = wav[0]
# Read only one stereo channel.
x = wav[1][:, 0]

# Create time vector (independent variable.)
time_vec = np.arange(len(x))/float(sample_rate)

# Plot original and amplified.
plt.plot(time_vec, amplify(x, alpha), label="Amplified")
plt.plot(time_vec, x, label="Original")
plt.legend()
plt.xlabel("Time $t$")
plt.ylabel("Relative air pressure $y(t)$")
plt.show()

out = amplify(x, alpha)

# Scale maximum absolute amplitude to 0.9,
# because 1.0 is the maximum allowed by the file .wav file
# format.
# Note that this will not allow you to hear the audio signal
# amplitude increasing.
out = 0.9*out/np.max(np.abs(out))
# Write compressed output to wav file.

# Patch from Jostein and Adrian (cast to 32 bit float.)
sio.write("guitar_amp.wav", sample_rate, np.array(out, dtype=
    np.float32))
```

Listing 5.1: 003_guitar/amplifier.py

Run this code and verify that the audio signal stored in file `guitar_amp.wav` is amplified. You will need to do this by inspecting a before and after amplification plot of the audio signal, as the

example code will normalize the amplitude of the signal before storing it to file.

You can use this script as a basis for experimenting with audio signal processing later on during this course.

8. Change the code in Listing 5.1 is such a way that it implements a clipping amplifier, as shown in Equation: 5.22. Figure out suitable values for α and β that make the guitar sound distorted.

Suggested solutions: Signals and Systems

1. a) To check linearity of a system, we check Equation 5.12:

$$\begin{aligned}\mathcal{T}\{c_1V_1(t) + c_2V_2(t)\} &= R^{-1}[c_1V_1(t) + c_2V_2(t)], \\ c_1\mathcal{T}\{V_1(t)\} + c_2\mathcal{T}\{V_2(t)\} &= c_1R^{-1}[V_1(t)]^2 + c_2R^{-1}[V_2(t)]^2,\end{aligned}$$

these are not equal, so the system is not a linear system.

- b) To check time-invariance for this system, verify Equation 5.17:

$$\begin{aligned}\mathcal{T}\{\mathcal{D}(V(t))\} &= \mathcal{T}\{V(t - \tau)\} = R^{-1}[V(t - \tau)]^2, \\ \mathcal{D}\{\mathcal{T}\{V(t)\}\} &= \mathcal{D}\{R^{-1}[V(t)]^2\} = R^{-1}[V(t - \tau)]^2.\end{aligned}$$

Here we get that the outputs are equal, hence the system is time-invariant.

2. Consider the discrete-time system defined as:

$$y[n] = \frac{1}{5} \sum_{k=0}^4 x[n-k].$$

- a) This system averages $x[n]$ with four of its past values.

- b) Have that:

$$\begin{aligned}\mathcal{T}\{c_1x_1[n] + c_2x_2[n]\} &= \frac{1}{5} \sum_{m=0}^4 [c_1x_1[n-m] + c_2x_2[n-m]], \\ c_1\mathcal{T}\{x_1[n]\} + c_2\mathcal{T}\{x_2[n]\} &= c_1 \frac{1}{5} \sum_{j=0}^4 x_1[n-j] + c_2 \frac{1}{5} \sum_{k=0}^4 x_2[n-k] = \frac{1}{5} \sum_{m=0}^4 [c_1x_1[n-m] + c_2x_2[n-m]].\end{aligned}$$

In the last step, the two sums were combined into one and the index was renamed to m . From this we can see that the output are equal, so the system is linear.

- c) To check time-invariance we calculate:

$$\begin{aligned}\mathcal{T}\{\mathcal{D}\{x[n]\}\} &= \mathcal{T}\{x[n - \tau]\} = \frac{1}{5} \sum_{k=0}^4 x[(n - \tau) - k], \\ \mathcal{D}\{\mathcal{T}\{x[n]\}\} &= \mathcal{D}\left\{\frac{1}{5} \sum_{k=0}^4 x[n-k]\right\} = \frac{1}{5} \sum_{k=0}^4 x[(n - k) - \tau].\end{aligned}$$

These are equal upon rearranging the parenthesis, hence $\mathcal{T}\{\cdot\}$ is time-invariant.

3. Consider a time-scaling system of the form $y(t) = x(\alpha t)$, where $x(t)$ is the input signal to the system.

- a) If $0 < \alpha < 1$, the output signal gets stretched, becoming wider. In effect, it reduces the frequency of the original signal. To see this compare $\sin(t)$ with $\sin(0.5t)$ as an example.
- b) If $\alpha > 1$, the original frequencies are increased by a factor of α , giving more oscillations in the signal. Again, compare $\sin(t)$ with $\sin(2t)$ as an example.
- c) Let's check if the system is linear. Have that:

$$\begin{aligned}\mathcal{T}\{c_1x_1(t) + c_2x_2(t)\} &= c_1x_1(\alpha t) + c_2x_2(\alpha t), \\ c_1\mathcal{T}\{x_1(t)\} + c_2\mathcal{T}\{x_2(t)\} &= c_1x_1(\alpha t) + c_2x_2(\alpha t),\end{aligned}$$

so the system is linear as these are equal.

- d) Checking time-invariance:

$$\begin{aligned}\mathcal{T}\{\mathcal{D}\{x(t)\}\} &= \mathcal{T}\{x(t - \tau)\} = x(\alpha(t - \tau)), \\ \mathcal{D}\{\mathcal{T}\{x(t)\}\} &= \mathcal{D}\{x(\alpha t)\} = x(\alpha t - \tau).\end{aligned}$$

The system is not time-invariant as these do not satisfy Equation 5.17.

- 4. Consider the derivative as a system $\mathcal{T}\{x(t)\} = \frac{d}{dt}x(t)$. This system is time-invariant and linear. It is linear since:

$$\begin{aligned}\mathcal{T}\{c_1x_1(t) + c_2x_2(t)\} &= c_1\frac{d}{dt}x_1(t) + c_2\frac{d}{dt}x_2(t), \\ c_1\mathcal{T}\{x_1(t)\} + c_2\mathcal{T}\{x_2(t)\} &= c_1\frac{d}{dt}x_1(t) + c_2\frac{d}{dt}x_2(t),\end{aligned}$$

are equal, which follows from the linearity of the differential operator.

For time-invariance, we check the condition in Equation 5.17:

$$\begin{aligned}\mathcal{T}\{\mathcal{D}\{x(t)\}\} &= \mathcal{T}\{x(t - \tau)\} = \frac{d}{dt}x(t - \tau), \\ \mathcal{D}\{\mathcal{T}\{x(t)\}\} &= \mathcal{D}\left\{\frac{d}{dt}x(t)\right\} = \frac{d}{dt}x(t - \tau).\end{aligned}$$

Showing that Equation 5.17 holds, hence the system is time-invariant.

- 5. The guitar amplifier is a system of the form:

$$y(t) = \begin{cases} -\beta, & \alpha x(t) < -\beta, \\ \alpha x(t), & |\alpha x(t)| \leq \beta, \\ \beta, & \alpha x(t) > \beta. \end{cases}$$

The system is not linear. To show this, find a counter example. The linearity should hold for all signals $x(t)$. In particular, consider

two signals for which $\alpha x_1(t) = \beta$ and $\alpha x_2(t) = \beta$. For these signals we get:

$$\mathcal{T}\{\alpha x_1(t)\} + \mathcal{T}\{\alpha x_2(t)\} = \beta + \beta = 2\beta,$$

while:

$$\mathcal{T}\{\alpha x_1(t) + \alpha x_2(t)\} = \mathcal{T}\{\beta + \beta\} = \mathcal{T}\{2\beta\} = \beta,$$

as $2\beta > \beta$ by the last condition, hence we get $\mathcal{T}\{2\beta\} = \beta$. Thus, we get a contradiction in that $2\beta = \beta$ for all β , hence the system is non-linear.

6. For the guitar amplifier of the form:

$$y_d(t) = \begin{cases} -\beta & \text{when } \alpha x(t) < -\beta, \\ \alpha x(t) & \text{when } |\alpha x(t)| \leq \beta, \\ \beta & \text{when } \alpha x(t) > \beta. \end{cases}$$

Have that:

$$\begin{aligned} \mathcal{T}\{\mathcal{D}\{x(t)\}\} &= \mathcal{T}\{x(t-\tau)\} = \begin{cases} -\beta, & \text{for } \alpha x(t-\tau) < -\beta, \\ \alpha x(t-\tau), & \text{for } |\alpha x(t-\tau)| \leq \beta, \\ \beta, & \text{for } \alpha x(t-\tau) > \beta. \end{cases} \\ \mathcal{D}\{\mathcal{T}\{x(t)\}\} &= \mathcal{D}\left\{\begin{cases} -\beta, & \text{for } \alpha x(t) < -\beta, \\ \alpha x(t), & \text{for } |\alpha x(t)| \leq \beta, \\ \beta, & \text{for } \alpha x(t) > \beta. \end{cases}\right\} = \begin{cases} -\beta, & \text{for } \alpha x(t-\tau) < -\beta, \\ \alpha x(t-\tau), & \text{for } |\alpha x(t-\tau)| \leq \beta, \\ \beta, & \text{for } \alpha x(t-\tau) > \beta, \end{cases} \end{aligned}$$

so the system is time-invariant.

7. Running Listing 5.1 and looking at the resulting plot, one can see that the signal has been amplified.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sio

# Amplifier gain.
alpha = 10.0

def amplify(x: np.ndarray, alpha: float) -> np.ndarray:
    """A function to compress signal x (signal processing system)."""
    return (alpha*x)

# guitar_clean.wav copyright
# Original author: LG downloaded from freesound.org,
# Original file name: Guitar clean rif.wav

# The read function returns two arguments,
```

```

# these being the sample rate and the actual data.
wav = sio.read("guitar_clean.wav")
# Read only one stereo channel.
sample_rate = wav[0]
# Returned is a 2-dimensional NumPy array corresponding
# to the left and right channel.
x = wav[1][:, 0]

# Create time vector (independent variable).
time_vec = np.arange(len(x)) / float(sample_rate)

# Plot original and amplified.
plt.plot(time_vec, amplify(x, alpha), label="Amplified")
plt.plot(time_vec, x, label="Original")
plt.legend()
plt.xlabel("Time $t$")
plt.ylabel("Relative air pressure $y(t)$")
# Call this if needed.
# plt.show()

# Amplify the signal.
out = amplify(x, alpha)

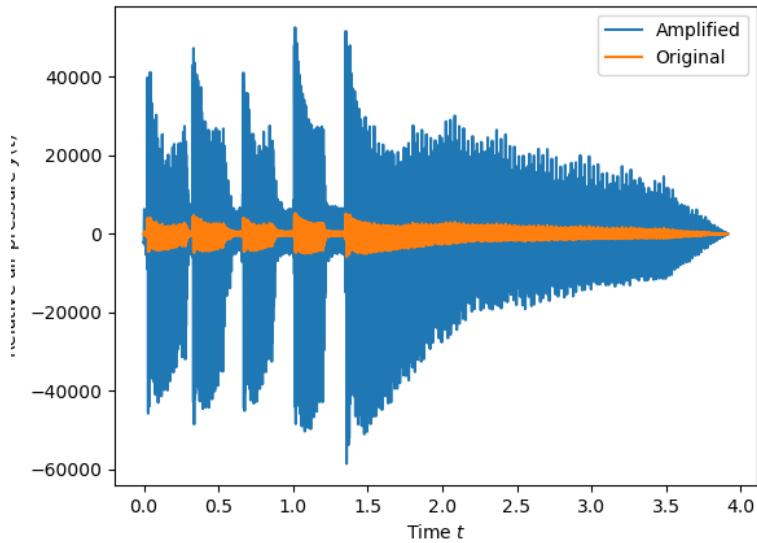
# Scale maximum absolute amplitude to 0.9, because 1.0 is
# the maximum allowed by the file format.
out = 0.9*out / np.max(np.abs(out))

# Write compressed output to wav file.
sio.write("guitar_amp.wav", sample_rate, np.array(out, dtype=
np.float32))

```

Listing 5.2: Solution for Exercise 4.7

The output from this system is shown below.



8. Running Listing 5.3 applies a distortion effect to the signal.

```

import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sio

alpha = 2.0 # Amplifier gain.
beta = 0.08 # Cut off.

def distortion(x: np.ndarray, alpha: float, beta: float) -> np.ndarray:
    """A function to apply distortion to a signal x."""
    y = np.zeros(x.size)

    print(x.dtype)

    # Create masks based on the conditions.
    mask1 = alpha*x < -beta
    mask2 = np.abs(alpha*x) <= beta
    mask3 = alpha*x > beta

    # The masks are arrays that are True/False, assign values
    # based on the True/False value.
    y[mask1] = -beta
    y[mask2] = alpha*x[mask2]
    y[mask3] = beta

    return y

# Read wav file (read only one stereo channel).
wav = sio.read("guitar_clean.wav")
sample_rate = wav[0]
# Read only one stereo channel.
x = wav[1][:, 0]

# Create time vector (independent variable).
time_vec = np.arange(x.size)/float(sample_rate)

# Apply distortion to the signal.
out = distortion(x, alpha, beta)

# Plot original and amplified.
plt.plot(time_vec, out, label="Distorted")
plt.plot(time_vec, x, label="Original")
plt.legend()
plt.xlabel("Time $t$")
plt.ylabel("Relative air pressure $y(t)$")
# Call this if needed.
# plt.plot()

# Scale maximum absolute amplitude to 0.9, because 1.0 is the
# maximum allowed by the file format.
out = 0.9*out/np.max(np.abs(out))
# Write compressed output to wav file.
sio.write("guitar_dist.wav", sample_rate, np.array(out, dtype=
    np.float32))

```

Listing 5.3: Solution for Exercise 4.8

6

Elementary Signals

THERE ARE TWO IMPORTANT ELEMENTARY SIGNALS which you will encounter throughout signal processing: the *unit impulse* $\delta(t)$ and the *unit step* function $u(t)$. These are important functions, as we can use these signals to represent any arbitrary signal. They play an *especially important role in defining signals in time domain*. If we know how a linear time-invariant system modifies these functions, then we are able to model how arbitrary signals behave. The unit impulse and unit step function have both continuous-time and discrete-time versions.

THE UNIT IMPULSE FUNCTION $\delta(t)$ is a special distribution function¹, which is infinitely narrow, non-zero valued only at $t = 0$. And when you integrate over this function, the result will be 1.

It is possible to define $\delta(t)$ in several ways as a limit of functions with a well-defined area and width σ . I'll use the Gaussian density function:

$$\delta_\sigma(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{t^2}{2\sigma^2}}. \quad (6.1)$$

The Gaussian density function is shown in Figure 6.1 for two different standard deviations σ_1 and σ_2 . The smaller the value of σ , the more narrow the function is.

As a result of how the function is defined, an integral over $\delta_\sigma(t)$ is always unity. This is perhaps not a surprise, as this function is used in statistics as a probability density function:

$$\int_{-\infty}^{\infty} \delta_\sigma(t) dt = 1. \quad (6.2)$$

The Dirac delta function $\delta(t)$ can be thought of as the limit when the standard deviation of this distribution approaches zero:

$$\lim_{\sigma \rightarrow 0} \int_{-\infty}^{\infty} \delta_\sigma(t) dt = \int_{-\infty}^{\infty} \delta(t) dt = 1 \quad (6.3)$$

¹ The unit impulse function is also known as the *Dirac delta function*, after physicist Paul Dirac, who used this function for modeling the distribution of electrical point-charges (e.g., electrons and protons)

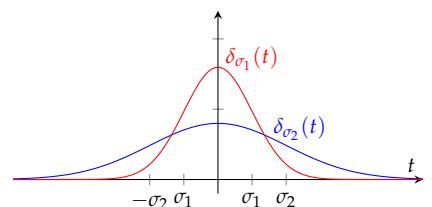


Figure 6.1: A Gaussian density function becomes a unit impulse $\delta(t)$ when the width parameter approaches zero $\sigma \rightarrow 0$.

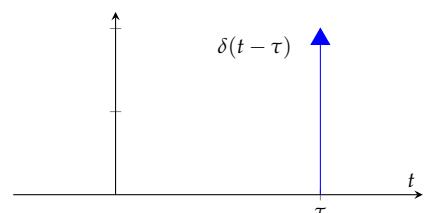


Figure 6.2: A unit impulse is typically depicted with a vertical arrow. A unit impulse $\delta(t - \tau)$ is centered at $t = \tau$. This is the only value of t where the unit impulse is non-zero.

For the sake of this course, we are not going to need to mathematically investigate the limit of this function. However, it is hopefully not difficult to convince yourself that the function becomes more and more narrow as $\sigma \rightarrow 0$, and that for all values of σ , the integral evaluates to unity.

But what does the function $\delta(t)$ look like? It is infinitely narrow, with only one non-zero value at zero. The equation below is not strictly a rigorous mathematical definition, but it gives you an idea:

$$\delta(t) = \begin{cases} \infty & \text{when } t = 0 \\ 0 & \text{when } t \neq 0 \end{cases} \quad (6.4)$$

When plotting this function, it is customary to utilize an up arrow, as shown in Figure 6.2. The figure also demonstrates how it is possible to shift the location of the peak by subtracting a constant τ from the argument of the function.

The main application of the unit impulse function in signal processing is selecting or “sampling” a value of a signal. This is also sometimes referred to as the *sifting* property. Consider a function $x(t)$ which has the value $x(\tau) = a$. If we integrate over the function $x(t)\delta(t - \tau)$, we obtain:

$$\int_{-\infty}^{\infty} x(t)\delta(t - \tau)dt = x(\tau) = a. \quad (6.5)$$

This type of integral will often appear when relating a continuous-time theory to a discrete-time theory. If you are still having a difficult time convincing yourself why the integral above results in the value it does, try thinking about it through the limit:

$$\lim_{\sigma \rightarrow 0} \int_{-\infty}^{\infty} x(t)\delta_{\sigma}(t - \tau)dt = x(\tau) = a. \quad (6.6)$$

The Dirac delta function is for the most part, used in this course inside an integral, and we will never need to deal with the singularity (∞).

THE DISCRETE-TIME UNIT IMPULSE $\delta[n]$ is defined as:

$$\delta[n] = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}. \quad (6.7)$$

The discrete-time unit impulse signal is shown in Figure 6.4. In most cases, the discrete-time unit impulse serves a similar function as its continuous-time equivalent. It is often theoretically advantageous to study linear systems by representing an arbitrary signal $x[n]$ as a sum of time shifted and scaled unit impulses:

$$x[n] = \sum_k x[k]\delta[n - k]. \quad (6.8)$$

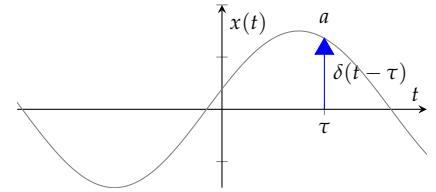


Figure 6.3: The unit impulse “selects” the value of a continuous-time signal $x(\tau) = a$.

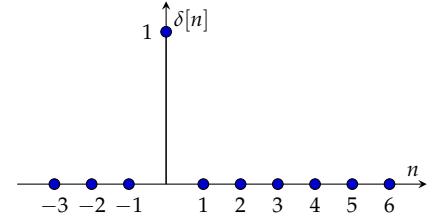


Figure 6.4: The discrete-time unit impulse.

This may seem like a unnecessarily complicated way of expressing a signal, but the advantage is that it simplifies the study of systems, allowing us to focus on analyzing what a linear system does to a time shifted unit impulse $\delta[n - k]$.

THE UNIT STEP FUNCTION can be defined as follows²:

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases}. \quad (6.9)$$

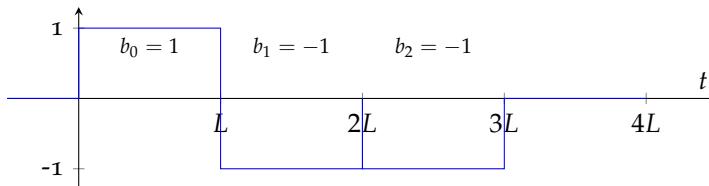
It is a step-like function, which abruptly transitions to 1 at zero. The following figure depicts the unit step function. Notice that there is a discontinuity at $t = 0$.

The unit step function can be used to create a rectangular function $\mu_L(t)$ of a certain length L . Here are two ways that this can be done:

$$\begin{aligned} \mu_L(t) &= u(t) - u(t - L) \\ &= u(t)u(L - t). \end{aligned}$$

Figure 6.5 shows a plot of the rectangular function. This type of function appears, e.g., when dealing with ideal filters that only select spectral components that lie within a specific band of frequencies. Rectangular function can also be used to model signal carrying radar or telecommunications signals, which have a constant value over a specific time period. Consider e.g., the following example, which happens to be the three bit Barker code used for radar pulse compression:

$$x(t) = \sum_{k=0}^2 b_k \mu_L(t - kL) \quad (6.10)$$



² This function is also called the Heaviside-function, after Oliver Heaviside, who used this function to model signals used in telecommunications.

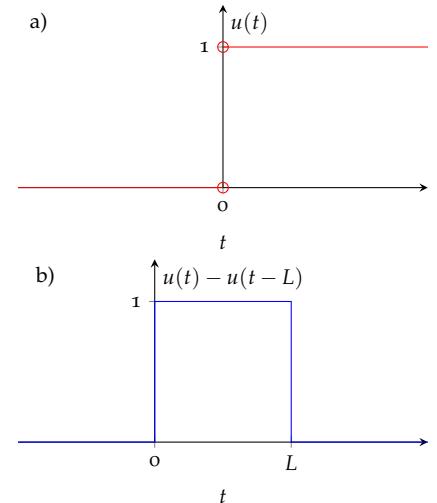


Figure 6.5: a) The unit step function $u(t)$ shown in blue transitions from 0 to 1 at the origin. b) Rectangular function expressed using two unit step functions.

THE DISCRETE-TIME UNIT STEP $u[n]$ FUNCTION is defined as:

$$u[n] = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}. \quad (6.11)$$

The discrete-time unit step function is shown in Figure 6.7.

Figure 6.6: A three-bit Barker code represented with the help of three rectangular functions.

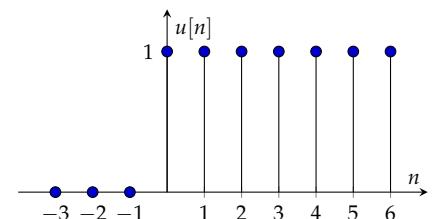


Figure 6.7: A discrete-time unit step function.

Exercises: Elementary Signals

The following exercises cover typical integrals with elementary signals, which you'll encounter in signal processing.

1. Simplify the following integrals:

a)

$$\int_{-\infty}^{\infty} \delta(t) e^{i\omega t} dt = ?$$

b)

$$\int_{-\infty}^{\infty} \delta(t - \tau) e^{i\omega t} dt = ?$$

c)

$$\int_{-\infty}^{\infty} \delta(\omega) e^{-i\omega t} d\omega = ?$$

d)

$$\int_{-\infty}^{\infty} \delta(\omega - \omega_0) e^{-i\omega t} d\omega = ?$$

2. Assume $L > 0$. Show that the integral:

$$\int_{-\infty}^{\infty} [u(t + L) - u(t - L)] e^{i\omega t} dt$$

evaluates to:

$$\alpha(e^{i\omega L} - e^{-i\omega L})$$

What is α ?

3. Consider the following integral:

$$\int_{-\infty}^{\infty} \sin(t - c) \delta(t) dt.$$

For which value of c do we have:

$$\int_{-\infty}^{\infty} \sin(t - c) \delta(t) dt = 0?$$

4. Evaluate the integral:

$$\int_{-\infty}^{\infty} u(t) u(L - t) \cos(2\pi\omega t) dt,$$

assuming $L > 0$.

Suggested solutions: Elementary Signals

1. By definition and known properties for the Dirac delta function, we have:

$$\int_{-\infty}^{\infty} x(t)\delta(t-\tau)dt = x(\tau).$$

a) Using the above, we get:

$$\int_{-\infty}^{\infty} \delta(t)e^{i\omega t}dt = e^{i\omega \cdot 0} = \underline{\underline{1}}.$$

b) Similarly, we get:

$$\int_{-\infty}^{\infty} \delta(t-\tau)e^{i\omega t}dt = \underline{\underline{e^{i\omega\tau}}}.$$

c) If the variable is ω , we have:

$$\int_{-\infty}^{\infty} \delta(\omega)e^{-i\omega t}d\omega = \underline{\underline{1}},$$

by the same reasoning as a).

d)

$$\int_{-\infty}^{\infty} \delta(\omega - \omega_0)e^{-i\omega t}d\omega = \underline{\underline{e^{-i\omega_0 t}}}.$$

2. Consider the integral:

$$\int_{-\infty}^{\infty} [u(t+L) - u(t-L)]e^{i\omega t}dt.$$

This corresponds to a rectangular function with height 1 and width $2L$ centered around 0, when $L > 0$. Thus, the integral can be simplified to:

$$\int_{-\infty}^{\infty} [u(t+L) - u(t-L)]e^{i\omega t}dt = \int_{-L}^L [u(t+L) - u(t-L)]e^{i\omega t}dt.$$

The function $u(t+L) - u(t-L) \equiv 1$ on the interval $[-L, L]$, hence:

$$\int_{-L}^L [u(t+L) - u(t-L)]e^{i\omega t}dt = \int_{-L}^L e^{i\omega t}dt = \left[\frac{1}{i\omega} e^{i\omega t} \right]_{-L}^L = \frac{1}{i\omega} (e^{i\omega L} - e^{-i\omega L}).$$

Therefore, $\alpha = \frac{1}{i\omega}$.

3. Using the properties of the Dirac delta function, we have:

$$\int_{-\infty}^{\infty} \sin(t-c)\delta(t)dt = \sin(0-c) = \sin(-c) = -\sin(c).$$

This evaluates to 0 for $c = \pi n$ where $n \in \mathbb{Z}$. Thus, c has to be

$$c = \pi n.$$

4. The function $u(t)u(L-t)$ corresponds to a rectangular function of length L and is 0 whenever $t < 0$ and $t > L$ and 1 if $0 < t < L$, therefore the integral can be written as:

$$\int_{-\infty}^{\infty} u(t)u(L-t)\cos(2\pi\omega t)dt = \int_0^L \cos(2\pi\omega t)dt = \frac{1}{2\pi\omega} [\sin(2\pi\omega t)]_{t=0}^{t=L} = \frac{1}{2\pi\omega} \sin(2\pi\omega L).$$

7

Sinusoidal Signals

A COMPLEX SINUSOIDAL SIGNAL IS THE ELEMENTARY BUILDING BLOCK OF THE FOURIER DECOMPOSITION OF A SIGNAL. This signal is fully determined by three parameters: amplitude $A \in \mathbb{R}_{\geq 0}$, phase $\phi \in \mathbb{R}$ (rad), and angular frequency $\omega \in \mathbb{R}$ (rad/s).

$$z(t) = Ae^{i(\omega t + \phi)} = Ae^{i\phi}e^{i\omega t} = Xe^{i\omega t}. \quad (7.1)$$

It is possible to combine phase and amplitude as one complex constant $X = Ae^{i\phi} \in \mathbb{C}$. This constant contains information about the amplitude and phase of the signal. Here X is often referred to as a *phasor*, whenever A , ω and ϕ have no time-dependence.

Using Euler's formula, we can express the complex sinusoidal signal as a function of cos and sin:

$$Ae^{i(\omega t + \phi)} = A[\cos(\omega t + \phi) + i \sin(\omega t + \phi)]. \quad (7.2)$$

With this formulation, it is apparent that the real and imaginary components are identical sinusoidal signals, which are 90 degrees out of phase with each other, because we know that $\cos(\theta) = \sin(\theta + \pi/2)$. Figure 7.1 shows an example of a complex sinusoidal signal.

You can also think of a complex sinusoidal signal as a point on the complex plane, which rotates around in a circle around the origin. This is sometimes referred to as the *rotating phasor*. The radius of the circle is the amplitude of the signal $A = |X|$. At $t = 0$, the point is at a phase angle of ϕ . This phase angle as a function of time is given by $\phi + \omega t$ and determines the position of the point on the circle at a given time. If $\omega > 0$, then the point moves counterclockwise around the circle when the value of t increases. If $\omega < 0$, then the point moves in the clockwise direction around the circle as the value of t increases. Figure 7.2 depicts this concept.

This might at first seem like an unnecessarily complicated way to express a sinusoidal signal, but this representation often simplifies

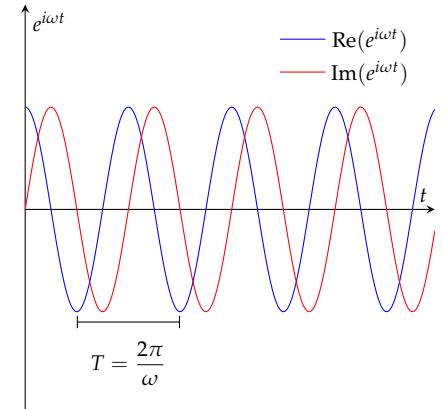


Figure 7.1: A complex sinusoidal signal is the basic building block (basis function) in a Fourier decomposition of a signal. The fundamental period $T = 2\pi/\omega$ of this signal is also shown.

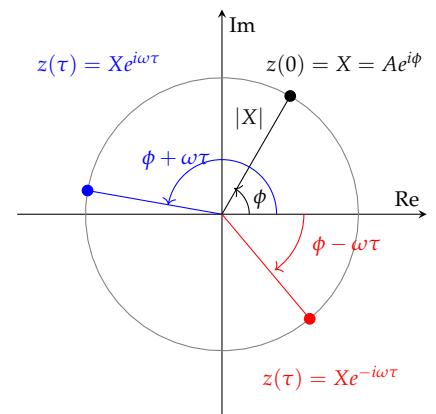


Figure 7.2: A complex sinusoidal signal on the complex plane can be seen as a point moving around a circle. For positive frequencies, the point rotates around the circle in counterclockwise direction. For negative frequencies, the rotation is clockwise.

the mathematics. For example, it is in most cases much easier to algebraically work with the function $\frac{1}{2}(e^{i\phi}e^{i\omega t} + e^{-i\phi}e^{-i\omega t})$ than it is with $\cos(\omega t + \phi)$, even though they are the same thing.

You'll also see, later on, that a complex sinusoidal signal is an *eigenfunction*¹ for any linear time-invariant operator. It allows us to characterize the properties of a linear time-invariant system just by simply investigating what effect the system has on a complex sinusoidal signal. If we spend the effort to learn how to use a complex sinusoidal signal, then we'll know everything there is to know about linear time-invariant systems!

¹ If $\mathcal{T}\{\cdot\}$ is linear and time-invariant, then: $\mathcal{T}\{e^{i(\omega t+\phi)}\} = \gamma e^{i(\omega t+\phi)}$ with $\gamma \in \mathbb{C}$.

REAL-VALUED SINUSOIDAL SIGNALS CAN BE RELATED TO COMPLEX SINUSOIDAL SIGNALS. Using the inverse Euler relationship, we can express a real-valued sinusoidal signal of some amplitude, angular frequency, and phase. By taking the real or imaginary part of a complex sinusoidal signal, we obtain a cosine or sine signal.

Using the earlier definition of a complex sinusoidal signal, we find that:

$$\text{Re}\{z(t)\} = \frac{1}{2} [Xe^{i\omega t} + X^*e^{-i\omega t}] = A \cos(\omega t + \phi) \quad (7.3)$$

$$\text{Im}\{z(t)\} = \frac{1}{2i} [Xe^{i\omega t} - X^*e^{-i\omega t}] = A \sin(\omega t + \phi) \quad (7.4)$$

This tells us that a real-valued sinusoidal signal with angular frequency ω is a sum of a positive and negative frequency complex sinusoidal signal with angular frequencies $\pm\omega$. Note that the sign of the phase angles of the two complex sinusoidal signals are also $\pm\phi$.

ANGULAR FREQUENCY ω of a complex sinusoidal signal, determines how many radians the phase of the signal advances per unit of time. If the unit of time is seconds, the angular frequency has units of *radians per second* (rad/s).

It is possible to relate angular frequency to frequency f in units of cycles per time unit, i.e., how many times does the point whose position is indicated by the complex sinusoidal signal rotate around the circle per unit of time:

$$\boxed{\omega = 2\pi f} \quad (7.5)$$

If the unit of time is seconds, then the unit of frequency f is hertz (Hz or 1/s). Sometimes *cycles per second* is indicated as the unit for frequency.

I will try to consistently use the name *angular frequency* and symbol ω when referring to a frequency that is in units of radians per unit of t . I will use the name *frequency* and the symbol f when referring to the frequency that is in units of cycles per unit of t . In some

cases, when I am talking about frequency in general, and it does not matter what the units are, I will drop the word angular and just talk about frequency.

It is unfortunate that these two competing units exist. This occasionally causes confusion even for people who have worked with signals for many years! It is not unusual to find a factor of 2π missing or unnecessarily added in a textbook that is discussing the frequency of a signal.

Throughout these lecture notes, we will most of the time call the independent variable t of a one dimensional signal time and assign it units of seconds. However, the physical units of t do not necessarily have to be seconds, and t does not have to denote time.

For example, if the units of t were distance in meters, then we would have an angular frequency ω that is in units of radians per meter, and frequency f that is in units of cycles per meter. In physics, a spatial frequency is typically called a *wavenumber*. It is customary to use the symbol k for angular wavenumber and the Greek letter ν for linear wavenumber. These types of spatial frequencies are often encountered when dealing with propagating waves. There is an example at the end of this chapter on this topic.

COMPLEX AND REAL-VALUED SINUSOIDAL SIGNALS are periodic. The definition of a periodic function requires that the function repeats after a certain time period T for all values of t :

$$\boxed{z(t) = z(t + nT).} \quad (7.6)$$

where $n \in \mathbb{Z}$ and $T \in \mathbb{R}_{>0}$.

The smallest possible positive non-zero value of T is called the *fundamental period* of the function.

In the case of complex sinusoidal signals, the definition of periodicity yields:

$$Ae^{i(\omega t + \phi)} = Ae^{i(\omega(t+T) + \phi)} = Ae^{i(\omega t + \omega T + \phi)} \quad (7.7)$$

By definition, we know that a complex sinusoidal signal is 2π -periodic, in other words

$$Ae^{i\theta} = Ae^{i(\theta + 2\pi k)} \quad (7.8)$$

for $k \in \mathbb{Z}$. By setting $\theta = \omega t + \phi$, this implies that $\omega T = 2\pi k$.

In order to look for the smallest value of T , we inspect the case where $k = 1$, from which it follows that the fundamental period is:

$$\boxed{T = \frac{2\pi}{\omega} = \frac{1}{f}.} \quad (7.9)$$

The fundamental period for a complex sinusoidal signal is shown in Figure 7.1.

MULTIPLE COMPLEX SINUSOIDAL SIGNALS WITH THE SAME FREQUENCY BUT POSSIBLY DIFFERENT PHASES AND AMPLITUDES CAN BE ADDED TOGETHER TO FORM ONE COMPLEX SINUSOIDAL SIGNAL.

This is sometimes referred to as the *phasor summation*² property.

Let us assume that we have N complex sinusoidal signals. Each one of these signals has the same angular frequency ω but different phases ϕ_n and amplitudes A_n . If we add these together, we get one complex sinusoidal signal with an amplitude A and phase ϕ :

$$\sum_{n=1}^N A_n e^{i(\omega t + \phi_n)} = A e^{i(\omega t + \phi)} \quad (7.10)$$

This is relatively easy to see. Let's define $X_n = A_n e^{i\phi_n}$, which allows us to rewrite the sum in the following form:

$$\sum_{n=1}^N A_n e^{i(\omega t + \phi_n)} = \sum_{n=1}^N X_n e^{i\omega t} \quad (7.11)$$

$$= \underbrace{\left(\sum_{n=1}^N X_n \right)}_X e^{i\omega t} \quad (7.12)$$

$$= X e^{i\omega t} \quad (7.13)$$

$$= |X| e^{i\angle X} e^{i\omega t} \quad (7.14)$$

$$= A e^{i\phi} e^{i\omega t} \quad (7.15)$$

Here $X = \sum_n X_n$ is just a sum of constant valued complex numbers that contain information about the phases and amplitudes of the individual complex sinusoids.

MULTIPLYING TWO COMPLEX SINUSOIDAL SIGNALS TOGETHER IS EQUIVALENT TO A SHIFT IN FREQUENCY AND PHASE. Consider two complex sinusoidal signals with angular frequencies ω_1 and ω_2 , and phase angles ϕ_1 and ϕ_2 . For the sake of simplicity, we'll assume that both signals are unit amplitude:

$$z_1(t) = e^{i(\omega_1 t + \phi_1)} \quad (7.16)$$

$$z_2(t) = e^{i(\omega_2 t + \phi_2)} \quad (7.17)$$

Multiplying together these two signals $z_3(t) = z_1(t)z_2(t)$ we get:

$$e^{i(\omega_3 t + \phi_3)} = e^{i[(\omega_1 + \omega_2)t + (\phi_1 + \phi_2)]} \quad (7.18)$$

The resulting signal $z_3(t)$ is a complex sinusoid, at a new frequency: $\omega_3 = \omega_1 + \omega_2$ and phase $\phi_3 = \phi_1 + \phi_2$.

² The signal $A e^{i\phi} e^{i\omega t}$ is called a phasor if A, ϕ , and ω have no time dependence.

This property is one of the elementary operations in signal processing. For example, we'll use multiplication of two complex sinusoids signals later on as part of the proof of the Shannon-Nyquist sampling theorem.

The frequency shift property is also widely used in radio engineering and telecommunications to shift high frequency signals to lower frequencies and vice versa. This operation is called downconversion or upconversion in radio engineering terminology.

IT IS POSSIBLE TO EXTEND THE CONCEPT OF A COMPLEX SINUSOIDAL SIGNAL INTO MULTIPLE DIMENSIONS. For example, a two-dimensional complex sinusoidal signal would be of the form:

$$z(x, y) = Ae^{i\phi} e^{ikx} e^{il'y}. \quad (7.19)$$

Here k and l are the angular frequencies in the x and y directions.

In the last example³ of this chapter, we'll show how to use a two-dimensional complex sinusoidal signal to derive the plane wave solution of electromagnetic waves propagating in free space from Maxwell's equations.

A TIME-SHIFT IS EQUIVALENT TO A PHASE SHIFT FOR A COMPLEX SINUSOIDAL SIGNAL. Let's investigate a time-shift system, which shifts time by a constant τ :

$$y(t) = \mathcal{T}\{x(t)\} = x(t - \tau), \quad (7.20)$$

which means that the output signal at time t is the same as the input signal at time $t - \tau$. One could also say that the output signal is delayed by a time constant τ .

For example, if $x(t) = \delta(t)$, then the location of the only non-zero valued peak in the input signal will be at $t = 0$. If we delay the signal by τ , then $y(t) = \delta(t - \tau)$ and the peak will now occur at $t = \tau$.

What does the time-delay system do to a complex sinusoidal signal? If the input signal is $x(t) = e^{i(\omega t + \phi)}$, the output becomes:

$$y(t) = \mathcal{T}\{e^{i(\omega t + \phi)}\} = e^{i[\omega(t - \tau) + \phi]} \quad (7.21)$$

$$= e^{i(\omega t + \phi - \omega\tau)} \quad (7.22)$$

$$= e^{-i\omega\tau} x(t) \quad (7.23)$$

The output of the time-shift system for a complex sinusoidal signal is the input signal multiplied by $e^{-i\omega\tau}$. In other words, phase shifted by $-\omega\tau$ radians. There is a linear dependence of phase shift with frequency, as shown in Figure 7.4.

Because of the 2π -periodicity of the complex sinusoidal signal, the phase shift introduced by a certain time delay is not unique. There

³ I couldn't resist using this nice example, even though this course is about 1d signals.

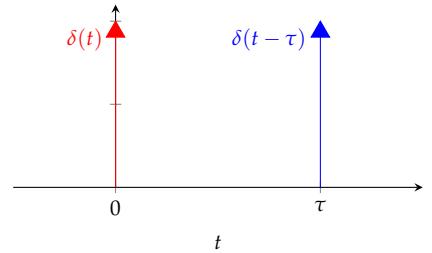


Figure 7.3: Time-shifted unit impulse. The time-shift system $y(t) = x(t - \tau)$ will delay the input signal by τ .

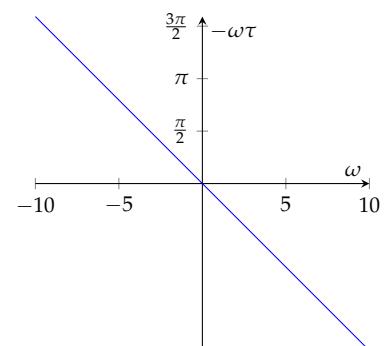


Figure 7.4: Phase shift introduced to a complex sinusoidal signal by a time-shift system. The phase shift caused by a constant time shift is linearly dependent on angular frequency.

are infinitely many time shifts that can cause a certain phase shift at a fixed frequency.

Let's say that a complex sinusoidal signal is phase shifted by phase θ . Due to the 2π periodicity, we have to consider all values $\theta + 2\pi k$ with $k \in \mathbb{Z}$. If we attribute this phase shift only to a time-delay, then the following relationship holds

$$e^{i(\theta+2\pi k)} = e^{-i\omega\tau}. \quad (7.24)$$

This means that all the time delays that could have caused a phase shift θ are:

$$\tau = -\omega^{-1}(\theta + 2\pi k). \quad (7.25)$$

This problem is sometimes called phase-time ambiguity.

Sometimes it is possible to solve this ambiguity by making the assumption that the true value of the time-shift is within a certain interval $\tau_{\min} < \tau < \tau_{\max}$ and find a unique solution for τ .

Another possible solution to the phase-time ambiguity is to measure the rate-of-change of the phase as a function of frequency:

$$\frac{d\theta}{d\omega} = -\tau. \quad (7.26)$$

This type of measurement is used by devices called network analyzers for measuring the length of a cable. This principle is also used in the radio astronomical method called very long baseline interferometry (VLBI) to determine the relative time delay between a point-like radio emission source and two different radio telescopes.

Example: Very long baseline interferometry

A point-like radio star emits a signal across a broad range of frequencies. This means that we can observe the signal originating from the radio star with multiple angular frequencies ω . We can denote the signal originating from the star at a certain frequency with:

$$z(t, \omega) = A(\omega)e^{i\phi(\omega)}e^{i\omega t} \quad (7.27)$$

Here $A(\omega) \in \mathbb{R}_{\geq 0}$ is the amplitude and $\phi(\omega)$ is the phase of the signal at angular frequency ω . Because the signal is generated by an enormous collection of charged particles (primarily electrons) in an accelerating motion, the amplitudes and phases are random variables.

If we measure the signal originating from the radio star at two different places with a radio telescope simultaneously, the time of arrival of the signal between the two telescopes will differ by τ due to geometry (see Figure 7.5). It is this value of τ that is often of interest, when, e.g., estimating the orientation of Earth relative to the stars.

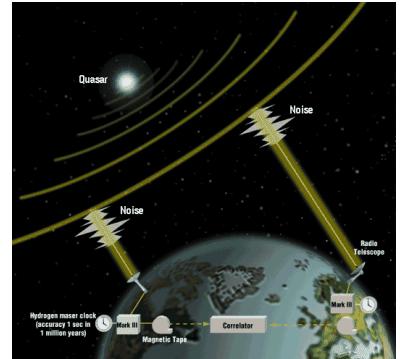


Figure 7.5: In radio astronomy, the relative time delay between two Earth-based radio telescopes and a point like radio emission source is determining the rate of change of the relative phase as a function of frequency. Figure: NASA GFSC.

Let's denote the signal received with telescope 1 with z_1 and the signal received with telescope 2 with z_2 :

$$z_1(t, \omega) = A(\omega)e^{i\phi(\omega)}e^{i\omega t} \quad (7.28)$$

$$z_2(t, \omega) = A(\omega)e^{i\phi(\omega)}e^{i\omega(t-\tau)} \quad (7.29)$$

We'll assume that z_1 is the same signal as z_2 , with only one exception, the signal z_2 is delayed by a time constant τ due to the receiver being further away from the source of the signal⁴.

In order to determine the value τ , we can use the following operation:

$$z_1(t, \omega)z_2^*(t, \omega) = |A(\omega)|^2 e^{i\omega\tau} \quad (7.30)$$

The random phase $\phi(\omega)$ cancels out with this operation, as well as the oscillating term ωt . The phase of this cross-product only depends on frequency and time shift τ :

$$\angle z_1(t, \omega)z_2^*(t, \omega) = \theta(\omega) = \omega\tau \quad (7.31)$$

If we measure the cross-product $z_1(t, \omega)z_2^*(t, \omega)$ across a sufficiently wide range of frequencies, we can estimate τ using:

$$\frac{d\theta}{d\omega} = \tau \quad (7.32)$$

This is a very simplified description of how the very long baseline interferometry technique used in radio astronomy and geodesy allows us to precisely measure the relative difference in time of arrival of a signal originating from a point-like radio source when measured with two different radio receivers.

Python example: shifting in frequency

The following Python example demonstrates in practice Equation 7.18 which tells us that when multiplying two complex sinusoidal signals together, the resulting signal will have a frequency corresponding to the sum of the two frequencies of the multiplied signals.

```
import matplotlib.pyplot as plt
import numpy as np

# Time vector 0 to 1 seconds, 1000 Hz sampler rate.
t = np.arange(1000)/float(1000.0)
# Frequency of 10 Hz.
omo = 2.0*np.pi*10.0
# Frequency of 5 Hz.
om1 = 2.0*np.pi*5.0
# Create complex sinusoidal signal.
z = np.exp(1j*omo*t)
# Shift z in frequency by multiplying with another
# complex sinusoidal signal of frequency om1.
```

⁴ For radio signals, we need the radio source to be in the Fraunhofer far-field in order for this to be valid.

```

z_shifted = z*np.exp(1j*om1*t)

# Plot signals.
plt.subplot(211)
plt.plot(t, z.real, label="Real")
plt.plot(t, z.imag, label="Imag")
plt.title("Original signal")
plt.xlabel("Time (s)")
plt.legend()
plt.subplot(212)
plt.plot(t, z_shifted.real, label="Real")
plt.plot(t, z_shifted.imag, label="Imag")
plt.title("Frequency shifted signal")
plt.legend()
plt.xlabel("Time (s)")
plt.show()

```

Listing 7.1: 005_mixing/mixing.py

Example: Electromagnetic waves

This example will show that complex sinusoidal signals are a useful mathematical tool also for solving differential equations by deriving a plane wave solution of electromagnetic waves from Maxwell's equations.

Don't worry if you are not familiar with these equations. I mainly want you to pay attention to how the differential equation is solved for a plane wave using a two-dimensional complex sinusoidal signal $\vec{E}(x, t) = \hat{y}E_0 e^{ikx}e^{i\omega t}$. Because the exponential function is easy to differentiate, it is easy to apply this type of function for solving differential equations.

Assuming that there are no currents ($\vec{J} = 0$) or space charges ($\rho = 0$), Maxwell's equations are as follows:

$$\nabla \cdot \vec{E} = 0 \quad (7.33)$$

$$\nabla \cdot \vec{B} = 0 \quad (7.34)$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad (7.35)$$

$$\nabla \times \vec{B} = \epsilon_0 \mu_0 \frac{\partial \vec{E}}{\partial t} \quad (7.36)$$

If we curl Faraday's law, we get:

$$\nabla \times (\nabla \times \vec{E}) = -\frac{\partial}{\partial t} \nabla \times \vec{B} \quad (7.37)$$

If we then use the vector identity $\nabla \times (\nabla \times \vec{E}) = \nabla(\nabla \cdot \vec{E}) - \nabla^2 \vec{E}$ and include Gauss' law for electric field $\nabla \cdot \vec{E} = 0$ in the case when there are no charges, we get

$$\nabla^2 \vec{E} = \frac{\partial}{\partial t} \nabla \times \vec{B} \quad (7.38)$$

If we now insert the Maxwell–Ampere's law (Equation 7.36) into Faraday's law (Equation 7.35), we get:

$$\frac{\partial^2 \vec{E}}{\partial t^2} - \frac{1}{\mu_0 \epsilon_0} \nabla^2 \vec{E} = 0. \quad (7.39)$$

This type of equation is called a wave equation. You can also derive the same equation for the magnetic field, but we are not going to do that here.

Now let us assume that a solution to the wave equation is of the form:

$$\vec{E}(x, t) = E_0 \hat{y} e^{-ikx} e^{i\omega t} \quad (7.40)$$

Here $E_0 \in \mathbb{C}$ is a complex number that determines the amplitude and phase of the electric field. The term \hat{y} is a unit vector in the y -axis direction. In other words, a plane wave, where the electric field in the y - z plane is constant-valued. The electric field only changes as a function of position in the x -axis as a function of time t .

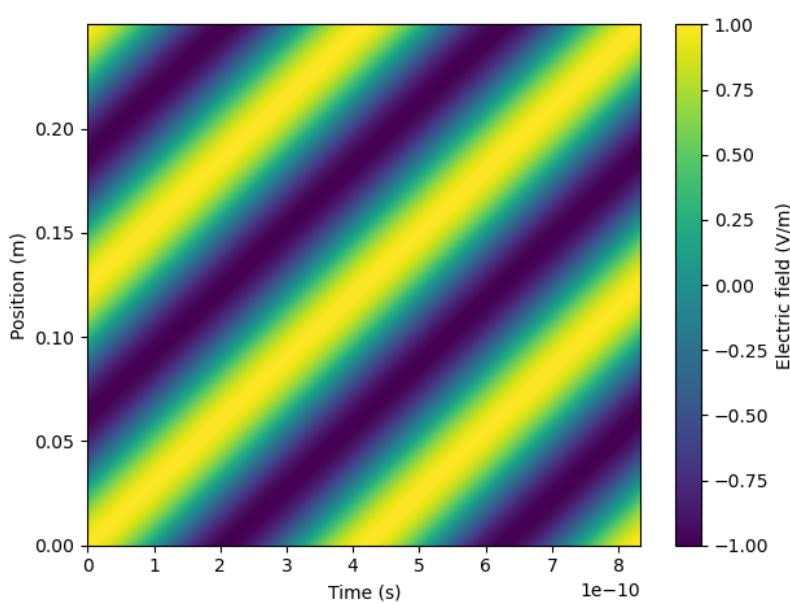


Figure 7.6: A plane wave solution to Maxwell's equations, describing propagation of electromagnetic waves in space and time. The script that produced this plot is 007_2dwave/em_plane_wave.py.

It is easy to differentiate a complex exponential function twice in space and time. Remember that the Laplacian operator, which is a linear system, is defined as:

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} \quad (7.41)$$

Because the electric field is constant in the y - z plane, only the second derivative in the x -direction remains. Using the two-dimensional

plane wave formulation of Equation 7.40, we can see that

$$\frac{\partial^2 \vec{E}}{\partial t^2} = -\omega^2 \vec{E}(x, t) \quad (7.42)$$

$$\nabla^2 \vec{E} = -k^2 \vec{E}(x, t). \quad (7.43)$$

And therefore, the wave equation becomes:

$$\frac{\partial^2 \vec{E}}{\partial t^2} - \frac{1}{\mu_0 \epsilon_0} \nabla^2 \vec{E} = 0 \quad (7.44)$$

$$(c^2 k^2 - \omega^2) \vec{E}(x, t) = 0 \quad (7.45)$$

I've taken the liberty of changing the constant $c = (\mu_0 \epsilon_0)^{-1/2}$ for reasons that will become apparent soon. The constant μ_0 is permeability of free space, and ϵ_0 is permittivity of free space. The value of $c \approx 3 \cdot 10^8$ m/s.

In order for the two-dimensional complex sinusoidal signal to be consistent with Maxwell's equations, either $E(x, t) = 0$ or $c^2 k^2 - \omega^2 = 0$. The trivial solution $E(x, t) = 0$ is not very interesting, as there would be no electric field. The latter describes a solution with a non-zero electric field, as long as the following relationship:

$$k = \pm \frac{\omega}{c} \quad (7.46)$$

is satisfied. This means that the following two-dimensional signal is a solution to the Maxwell's equations:

$$\vec{E}(x, t) = E_0 \hat{y} e^{-i(\omega/c)x} e^{i\omega t} \quad (7.47)$$

$$= E_0 \hat{y} e^{-i2\pi\lambda^{-1}x} e^{i2\pi f t} \quad (7.48)$$

Here, I've used the relation $2\pi f = \omega$. I've also used the wavelength of the electromagnetic wave, which is $\lambda = c/f$. In other words, we have used a two-dimensional complex sinusoidal signal and the Maxwell's equations to describe how electromagnetic waves propagate in space and time.

I am showing the real part of the plane wave solution in Figure 7.6. In this case, the frequency is chosen to be $f = 2.4$ GHz, which is the frequency typically used by microwave ovens and wireless internet (which you are probably using right now). Try to figure out what is the velocity that a region of constant phase propagates in the positive x-axis direction at.

Exercises: Sinusoidal Signals

Hint: You can often write a Python program to verify your result if you are not sure if it is correct!

1. You may have heard somebody say that noise-cancelling headphones rely on playing an audio signal where each spectral component of the signal is perfectly “out of phase” with the input signal. Consider a noise source that is sinusoidal and described by the following equation:

$$x_{\text{noise}}(t) = \cos(\omega t + \phi). \quad (7.49)$$

Your noise-cancelling system can generate a signal $x_{\text{cancel}}(t)$ with the same amplitude and frequency, but a different phase:

$$x_{\text{cancel}}(t) = \cos(\omega t + \phi_c) \quad (7.50)$$

What value of ϕ_c results in perfect cancellation

$$x_{\text{cancel}}(t) + x_{\text{noise}}(t) = 0? \quad (7.51)$$

2. Prove that:

$$A \cos(\omega t + \phi) = a \cos(\omega t) + b \sin(\omega t). \quad (7.52)$$

How is A and ϕ related with a and b ? Do this by converting the cosine and sine signals to complex sinusoidal signals (Equations 7.3 and 7.4). Use the phasor summation property found in Equation 7.15.

The representation of the form $a \cos(\omega t) + b \sin(\omega t)$ is often preferred over $A \cos(\omega t + \phi)$ when estimating the phase and amplitude of a real-valued sinusoidal signal from noisy measurements, because we get a set of linear equations with two unknown constants a and b .

3. Consider the following signal, which consists of two sinusoidal signals multiplied with one another:

$$x(t) = \cos(\omega_1 t) \sin(\omega_2 t). \quad (7.53)$$

Show that it is possible to express this signal in the form:

$$x(t) = a \sin(\omega_3 t) + b \sin(\omega_4 t). \quad (7.54)$$

What is the value of a , b , ω_3 , and ω_4 ?

4. The electromagnetic wave example demonstrated that a two-dimensional complex sinusoidal signal of the form:

$$\vec{E}(x, t) = \hat{y} E_0 e^{ikx} e^{i\omega t} \quad (7.55)$$

is a solution to the Maxwell's equation in free space.

Your task is to show that a real-valued solution $\text{Re}\{\vec{E}(x, t)\}$ with $\vec{E}(x, t) \neq 0$ is also a solution to the wave equation, so that we know that it is possible to have solutions with a real-valued electric field. In other words, show that Equation 7.45 holds also in this case.

5. This is a follow-up question to the previous one. Let's say that we had an arbitrary number of solutions to the wave equation with different angular frequencies ω_n and phasors E_n :

$$\vec{E}(x, t) = \hat{y} \sum_n E_n e^{i(\omega_n/c)x} e^{i\omega_n t} \quad (7.56)$$

Would this still satisfy the Maxwell's equations (i.e., the wave equation)? Can electromagnetic waves be composed of a superposition of two-dimensional plane waves of different frequencies (wavelengths, or colors), which each individually satisfy the wave equation?

6. Copy the example program in Listing 7.1 that performs mixing and run it. The example shows how to shift a complex exponential signal with a certain frequency to another frequency. Modify the script in such a way that the resulting complex exponential signal `z_shifted` has a frequency of 42 Hz. Use the plot to verify that the frequency of the signal is approximately 42 Hz.
7. Go back to the guitar amplifier program `003_guitar/amplifier.py`. We will now add one more effect. We'll modify the program so that it implements a modulation effect.

- a) Multiply the clean guitar signal with a 5 Hz sinusoidal signal.
Plot the signal.

```
# Multiply with sinusoid.
x_mod = x*np.cos(2.0*np.pi*time_vec*5.0)
# Plot result.
plt.plot(time_vec, x_mod)
plt.show()
```

- b) Write the output into an audio file and play the result. What does the output sound like?

```
# Write compressed output to wav file.
sio.write("guitar_leslie.wav", sample_rate, x_mod)
```

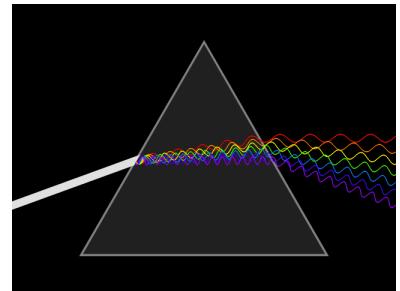


Figure 7.7: Light can be seen as a superposition of electromagnetic waves with different amplitudes, phases, and frequencies. This can be easily visualized in practice with the help of a prism or a diffraction grating.

- c) Change the frequency of the modulating sinusoid from 5 Hz to 2.5 Hz. Predict what the audio will sound like. Play the resulting audio to confirm your hypothesis.

If you want to reflect on what is going on here, think of the original audio signal $x(t)$ as something that consists of a sum of sinusoidal signal components⁵

⁵ We'll introduce this properly later on

$$x(t) = \sum_n A_n \cos(\omega_n t + \phi_n) \quad (7.57)$$

When multiplying this signal with a sinusoidal signal of frequency ω , we get a modulated audio signal $y(t)$:

$$y(t) = x(t) \cos(\omega t) \quad (7.58)$$

$$y(t) = \left[\sum_n A_n \cos(\omega_n t + \phi_n) \right] \cos(\omega t). \quad (7.59)$$

Each spectral component of a signal $x(t)$ gets the same treatment as you already investigated in Exercise 3.

Suggested solutions: Sinusoidal Signals

1. If $x_{\text{noise}}(t) = \cos(\omega t + \phi)$ and $x_{\text{cancel}}(t) = \cos(\omega t + \phi_c)$ and:

$$x_{\text{noise}}(t) + x_{\text{cancel}}(t) = 0,$$

then we must have:

$$\cos(\omega t + \phi) + \cos(\omega t + \phi_c) = 0,$$

for all t . The phase doesn't depend on t , so it is the same for every $t \in \mathbb{R}$, in particular for $t = 0$ so:

$$\begin{aligned}\cos(\phi) &= -\cos(\phi_c), \\ \cos(\phi) &= \cos(\pi - \phi_c), \\ \phi + 2\pi k &= \pi - \phi_c + 2\pi l,\end{aligned}$$

for $k, l \in \mathbb{Z}$, which gives $\phi_c = \pi - \phi + 2\pi(l - k) = \pi - \phi + 2\pi m$, where $m \in \mathbb{Z}$. Take $m = 0$ to obtain a simple solution, which gives $\phi_c = \pi - \phi$. Hence, the canceling signal must be of the form:

$$x_{\text{cancel}}(t) = \cos(\omega t + (\pi - \phi)).$$

A simple test program to verify this is shown in Listing 7.2 with an arbitrary choice of ω , ϕ and m . If the code is run, two signals of opposite amplitude are shown, which cancel out when added, as hoped.

```
import matplotlib.pyplot as plt
import numpy as np

N = 1000    # Number of sample points.
om = 2.6    # Angular frequency.
phi = 3.1   # Phase.
m = 0       # Integer.
# Compute the phase needed to cancel the signal.
phi_cancel = np.pi - phi + 2*np.pi*m

# Partition the t axis with a range from
# 0 to 4pi with N samples.
t = np.linspace(start=0, stop=4*np.pi, num=N)

# Original signal.
x = np.cos(om * t + phi)

# Noise canceling signal.
y = np.cos(om * t + phi_cancel)

plt.plot(t, x)
plt.plot(t, y)
# Call this if needed.
# plt.show()
```

Listing 7.2: Noise canceling signal

2. Have that:

$$a \cos(\omega t) + b \sin(\omega t) = \frac{a}{2}(e^{i\omega t} + e^{-i\omega t}) + \frac{b}{2i}(e^{i\omega t} - e^{-i\omega t}),$$

or written out:

$$a \cos(\omega t) + b \sin(\omega t) = \frac{a}{2}e^{i\omega t} + \frac{a}{2}e^{-i\omega t} + \frac{b}{2i}e^{i\omega t} - \frac{b}{2i}e^{-i\omega t}.$$

Using the phasor summation property we have:

$$X = \sum_n X_n = \frac{a}{2} + \frac{b}{2i} = \frac{1}{2}(a - bi),$$

and

$$Y = \sum_m Y_m = \frac{a}{2} - \frac{b}{2i} = \frac{1}{2}(a + bi).$$

Then we have that:

$$\begin{aligned} |X| &= \sqrt{\frac{1}{4}(a^2 + b^2)} = \frac{1}{2}\sqrt{a^2 + b^2}, \\ |Y| &= \sqrt{\frac{1}{4}(a^2 + b^2)} = \frac{1}{2}\sqrt{a^2 + b^2}, \end{aligned}$$

and

$$\begin{aligned} \angle X &= \arctan\left(\frac{-b}{a}\right) = \phi, \\ \angle Y &= \arctan\left(\frac{b}{a}\right). \end{aligned}$$

Then the whole expression can be rewritten as:

$$\begin{aligned} a \cos(\omega t) + b \sin(\omega t) &= \frac{1}{2}\sqrt{a^2 + b^2}e^{i\phi}e^{i\omega t} + \frac{1}{2}\sqrt{a^2 + b^2}e^{-i\phi}e^{-i\omega t}, \\ &= \frac{1}{2}\sqrt{a^2 + b^2}e^{i(\omega t + \phi)} + \frac{1}{2}\sqrt{a^2 + b^2}e^{-i(\omega t + \phi)}, \\ &= \sqrt{a^2 + b^2} \left[\frac{1}{2} \left(e^{i(\omega t + \phi)} + e^{-i(\omega t + \phi)} \right) \right], \\ &= \sqrt{a^2 + b^2} \cos(\omega t + \phi), \\ &= A \cos(\omega t + \phi). \end{aligned}$$

Here $A = \sqrt{a^2 + b^2}$ and $\phi = -\arctan(b/a)$, where $\arctan(b/a)$ should give the angle corresponding to which quadrant the point (a, b) lies in the xy -plane.

3. Consider the signal:

$$x(t) = \cos(\omega_1 t) \sin(\omega_2 t),$$

which we can write as:

$$x(t) = \frac{1}{2}(e^{i\omega_1 t} + e^{-i\omega_1 t}) \frac{1}{2i}(e^{i\omega_2 t} - e^{-i\omega_2 t}) = \frac{1}{4i}[e^{i(\omega_1 + \omega_2)t} - e^{i(\omega_1 - \omega_2)t} + e^{i(-\omega_1 + \omega_2)t} - e^{i(-\omega_1 - \omega_2)t}].$$

Collecting terms we get:

$$\begin{aligned} x(t) &= \frac{1}{4i}[e^{i(\omega_1+\omega_2)t} - e^{-i(\omega_1+\omega_2)t}] + \frac{1}{4i}[e^{i(-\omega_1+\omega_2)t} - e^{-i(-\omega_1+\omega_2)t}], \\ &= \frac{1}{2}\sin((\omega_1 + \omega_2)t) + \frac{1}{2}\sin((\omega_2 - \omega_1)t). \end{aligned}$$

This yields the values $a = 1/2$ and $b = 1/2$, while $\omega_3 = \omega_1 + \omega_2$ and $\omega_4 = \omega_2 - \omega_1$.

4. The plane wave:

$$E(x, t) = E_0 e^{i(kx + \omega t)} \partial_y$$

is a solution to the wave equation. Consider the real part $\text{Re}\{E(x, t)\}$:

$$\text{Re}\{E(x, t)\} = \frac{1}{2}[E(x, t) + E^*(x, t)] = \frac{1}{2}E_0(e^{i(kx + \omega t)} + e^{-i(kx + \omega t)})\partial_y = E_0 \cos(kx + \omega t)\partial_y.$$

Recall that the wave equation is:

$$\partial_t^2 E - \frac{1}{\mu_0 \epsilon_0} \nabla^2 E = 0,$$

which gives:

$$-\omega^2 E_0 \cos(kx + \omega t)\partial_y + \frac{1}{\mu_0 \epsilon_0} k^2 E_0 \cos(kx + \omega t)\partial_y = E_0 \left(\frac{1}{\mu_0 \epsilon_0} k^2 - \omega^2 \right) \cos(kx + \omega t)\partial_y = 0,$$

since $k^2 c^2 - \omega^2 = 0$, as $c = \sqrt{1/\mu_0 \epsilon_0}$. In conclusion, the signal:

$$\text{Re}\{E(x, t)\} = E_0 \cos(kx + \omega t)\partial_y,$$

is a solution to the wave equation.

5. If we have an arbitrary number of solutions to the wave equation, each with different phasors E_n and different angular frequencies ω_n . Then a superposition:

$$E(x, t) = \sum_n E_n e^{i(\omega_n/c)x} e^{i\omega_n t} \partial_y,$$

is also a solution. This follows by the linearity of the wave equation.

6. The code from the notes has been modified such that the output signal has a frequency of approximately 42.0 Hz.

```
import matplotlib.pyplot as plt
import numpy as np

# Time vector 0 to 1 seconds, 1000 Hz sample rate.
t = np.arange(1000) / 1000.0

# Frequency of 10.0 Hz and 32.0 Hz.
omo = 2.0 * np.pi * 10.0
om1 = 2.0 * np.pi * 32.0
```

```

# Create complex sinusoidal signal.
z = np.exp(1j*omo*t)

# Shift z in frequency by multiplying with another
# complex sinusoidal signal of frequency om1,
# the result has a frequency of 42.0 Hz = 10.0 Hz + 32.0 Hz.
z_shifted = z*np.exp(1j*om1*t)

plt.subplot(211)
plt.plot(t, z.real, label="Real", color="darkviolet")
plt.plot(t, z.imag, label="Imag", color="lime")
plt.title("Original signal")
plt.xlabel("Time (s)")
plt.legend()
plt.subplot(212)
plt.plot(t, z_shifted.real, label="Real", color="darkviolet")
plt.plot(t, z_shifted.imag, label="Imag", color="lime")
plt.title("New signal")
plt.xlabel("Time (s)")
plt.legend()
plt.tight_layout()
# Call this if needed.
# plt.show()

```

Listing 7.3: Adding frequencies

Running the code in Listing 7.3 yields the following output:

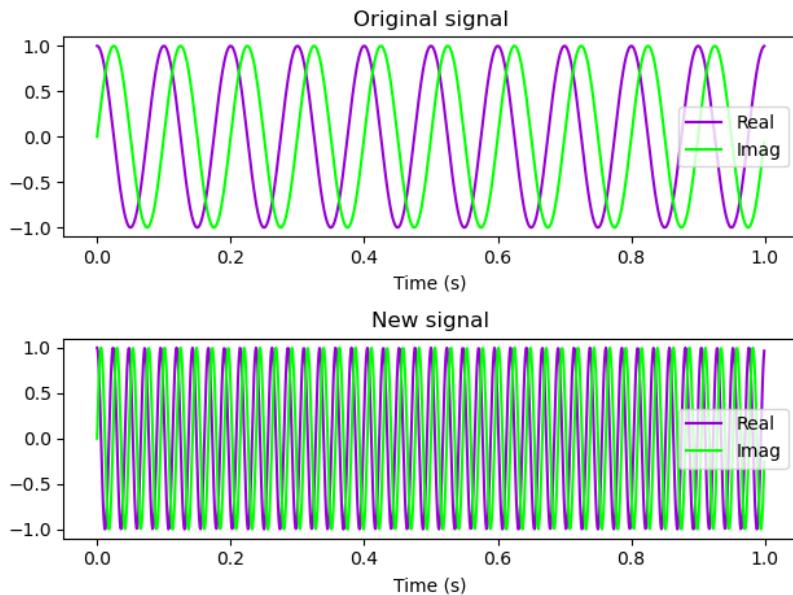


Figure 7.8: Signal with 42 Hz frequency

7. The following code is a modification that multiplies the signal with a periodic 5.0 Hz signal:
 - a) Listing 7.4 provides a suggested solution.

```

import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sio

wav = sio.read("guitar_clean.wav")
sample_rate = wav[0] # Sample rate.
x = wav[1][:, 0] # Read only one stereo channel.

# Create time vector (independent variable).
time_vec = np.arange(len(x))/float(sample_rate)

# Multiply the signal with a cosine wave
# with frequency f = 5.0 Hz.
out = x*np.cos(2.0*np.pi*time_vec*5.0)

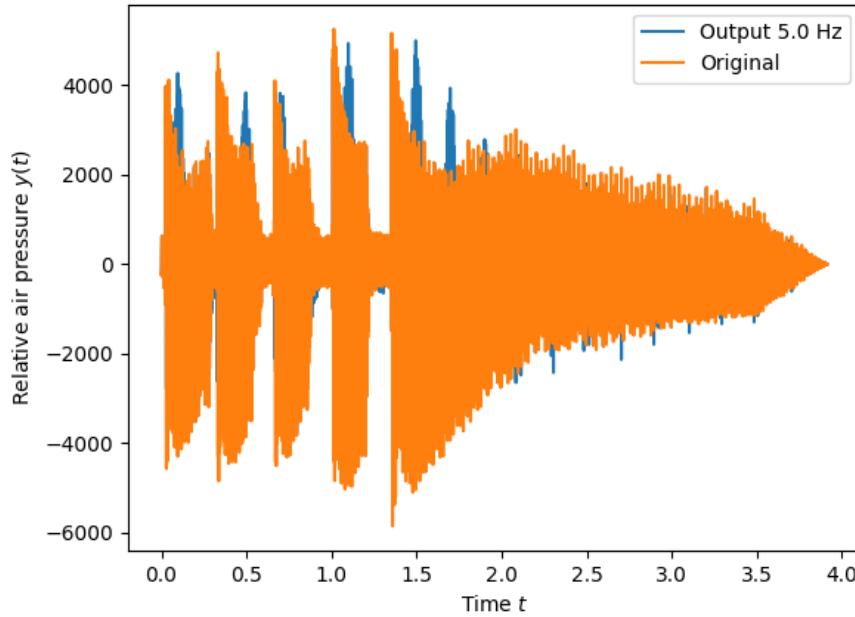
# Plot original and modified.
plt.plot(time_vec, out, label="Output 5.0 Hz", color="darkviolet")
plt.plot(time_vec, x, label="Original", color="lime")
plt.legend()
plt.xlabel("Time $t$")
plt.ylabel("Relative air pressure $y(t)$")
# Call this if needed.
# plt.show()

```

Listing 7.4: Modifying signals

Running Listing 7.4 produces Figure 7.9.

Figure 7.9: Original and modified guitar signal



- b) Adding the following code to Listing 7.4 will save the result as an audio file.

```
out = 0.9*out/np.max(np.abs(out))

# Multiply the signals.
out2 = x*n.cos(2.0*n.pi*time_vec*2.5)

# Write compressed output to wav file.
sio.write("guitar_leslie.wav", sample_rate, np.array(out,
dtype=np.float32))
```

Listing 7.5: Saving an audiofile

Listening to the audio file, we hear that the guitar sound has a wavy oscillating effect.

- c) Based on the previous task, the result of multiplying with a 2.5 Hz signal will result in the same effect, but the oscillation will be slower. After testing this by running Listing 7.5 this is exactly what happens.

8

Fourier Series

This chapter will introduce the *Fourier series*, which is used to express a periodic signal as a sum of sinusoidal signals. This is the first time that a spectral representation of signals is introduced. We will explore the following topics:

- Signals as a sum of sinusoids.
- Periodic signals, *fundamental frequency* and *fundamental period*.
- Expressing a periodic function as a sum of complex sinusoidal signals (Fourier series synthesis).
- Determining the phase and amplitude of each Fourier series coefficient (Fourier series analysis).
- Derivation of the *continuous-time Fourier transform* as a generalization of the Fourier series.

HERE IS A SIGNAL REPRESENTED AS A SUM OF COMPLEX SINUSOIDAL SIGNALS:

$$x(t) = \sum_{k=1}^N c_k e^{i\omega_k t}. \quad (8.1)$$

The symbol $c_k = A_k e^{i\phi_k} \in \mathbb{C}$ is used for complex-valued constants that contain information about the amplitudes $A_k \in \mathbb{R}_{\geq 0}$ and phases $\phi_k \in \mathbb{R}$ of the complex exponential signals with angular frequencies $\omega_k \in \mathbb{R}$.

It is possible to plot the complex coefficients c_k as a function of angular frequency ω as shown in Figure 8.1, which depicts a function of the form:

$$\hat{x}(\omega) = \sum_{k=1}^N c_k \delta(\omega - \omega_k). \quad (8.2)$$

This is a spectral or frequency domain representation of the signal described in Equation 8.1. In this case, one can think of the spectrum

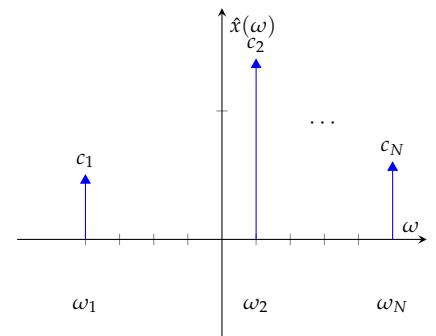


Figure 8.1: A spectral representation of a signal consisting of N complex sinusoidal signals. It is not a coincidence that I'm using the same arrow symbol here that I used earlier when introducing the Dirac delta function.

as consisting of infinitely narrow spectral lines, each defined by complex constants c_k that are located at angular frequencies ω_k . We will derive Equation 8.2 later when discussing the continuous-time Fourier transform.

A SPECTRAL REPRESENTATION OF A COSINE SIGNAL consists of two frequency components. One with a positive and one with a negative frequency. This is a direct consequence of Euler's formula.

Consider the following signal:

$$x(t) = A \cos(\omega t + \phi), \quad (8.3)$$

where $A \in \mathbb{R}_{\geq 0}$. We can use Euler's formula to obtain the spectral representation:

$$x(t) = \frac{A}{2} e^{i\phi} e^{i\omega t} + \frac{A}{2} e^{-i\phi} e^{-i\omega t} \quad (8.4)$$

$$= c_1 e^{i\omega_1 t} + c_{-1} e^{i\omega_{-1} t} \quad (8.5)$$

$$= \sum_{k \in \{-1, 1\}} c_k e^{i\omega_k t}. \quad (8.6)$$

The second line is simply representing the first line in the format of Equation 8.1.

Note that there is symmetry within the frequency components. The pairing of frequencies exists as $\omega_1 = -\omega_{-1}$. The complex constants are also conjugate symmetric $c_{-1} = c_1^*$. We'll later on see that this type of conjugate symmetry exists for the frequency components of all real-valued signals.

A FOURIER SERIES IS A SPECTRAL REPRESENTATION OF A SIGNAL WHERE EACH FREQUENCY IS A MULTIPLE OF SOME COMMON BASE FREQUENCY ω :

$$x(t) = \sum_{k=1}^N c_k e^{i\omega_k t} = \sum_{k=1}^N c_k e^{i\ell_k \omega t}. \quad (8.7)$$

Here $\ell_k \in \mathbb{Z}$ is integer valued. If we compare Equation 8.7 with the first definition of a spectral representation in Equation 8.1, we can see that all angular frequencies are integer multiples of a positive valued angular base frequency $\omega \in \mathbb{R}_{\geq 0}$:

$$\omega_k = \ell_k \omega. \quad (8.8)$$

The largest possible value ω is called the *fundamental angular frequency* of the signal.

The *fundamental period* T of the signal is related with the fundamental angular frequency ω as follows:

$$T = \frac{2\pi}{\omega}.$$

(8.9)

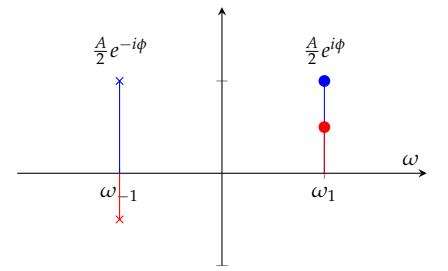


Figure 8.2: A spectral representation of a cosine signal consists of two frequency components: $\frac{1}{2} Ae^{i\phi} e^{i\omega t}$ and $\frac{1}{2} Ae^{-i\phi} e^{-i\omega t}$. Here A is a non-negative real-valued amplitude. Blue denotes the real and the red denotes the imaginary component of c_k .

The definition of periodicity for a function is that the following condition is satisfied:

$$x(t) = x(t + T) . \quad (8.10)$$

It is relatively easy to show that periodicity follows from the definition of the Fourier series, by inspecting the periodicity of each frequency component separately:

$$c_k e^{i\ell_k \omega t} = c_k e^{i\ell_k \omega(t+T)} = c_k e^{i(\ell_k \omega t + 2\pi\ell_k)} = c_k e^{i\ell_k \omega t} . \quad (8.11)$$

We have used the definition of T in Equation 8.9. Because all frequency components are periodic signals with period T , it follows that the sum of all frequency components is periodic as well.

A consequence of Equation 8.8 is that the ratios of all pairs of angular frequencies ω_k and ω_ℓ in Equation 8.1 need to be rational numbers. In other words, it must be possible to express them in the following form

$$\frac{\omega_k}{\omega_\ell} = \frac{n}{m} \in \mathbb{Q} \quad (8.12)$$

in order for the signal to be periodic. Here n and m are arbitrary integers and k and ℓ are all unique combinations of the frequency indices. If the ratio of two real numbers $\omega_1, \omega_2 \in \mathbb{R}$ is a rational number $\omega_1/\omega_2 \in \mathbb{Q}$, the numbers ω_1 and ω_2 are called *commensurable*. All angular frequencies for a Fourier series need to be commensurable relative to one another in order for the signal to be periodic.

How does one determine the fundamental frequency ω ? It is possible to apply *Euclid's algorithm*. This is sometimes referred to as the *greatest common divisor* for the set of frequencies ω_k . Note that this algorithm will only produce a result in a finite number of steps if the set of numbers $\{\omega_k\}$ are commensurable!

HERE IS AN EXAMPLE. Let us assume that a signal is defined as

$$x(t) = c_1 e^{i\omega_1 t} + c_2 e^{i\omega_2 t} = e^{i2\pi9t} - e^{i2\pi15t} . \quad (8.13)$$

Is this signal periodic? If so, what is its fundamental frequency ω and fundamental period T ?

In order for this signal to be periodic, ω_1 and ω_2 must be commensurable. In other words, $\omega_1/\omega_2 \in \mathbb{Q}$. This is the case, as it is clear that $\omega_1/\omega_2 = 3/5$ is a rational number.

This signal has two unique angular frequencies: $\omega_1 = 2\pi9$ and $\omega_2 = 2\pi15$. Let's use Euclid's algorithm¹ on these two numbers to determine the fundamental angular frequency ω . We start with the two numbers: $(2\pi9, 2\pi15)$. Then subtract the smallest number from the largest number $2\pi15 - 2\pi9 = 2\pi6$ and replace the larger number

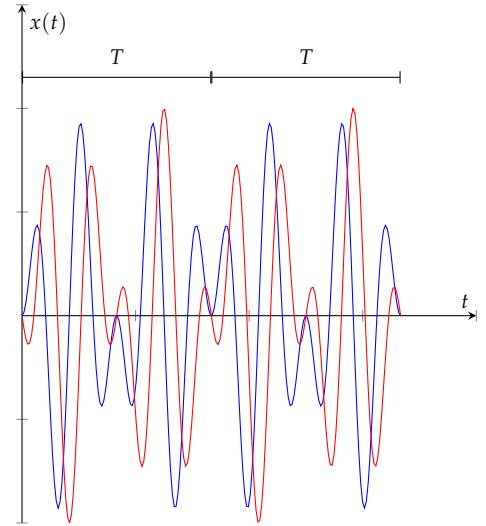


Figure 8.3: Periodic function $e^{i2\pi9t} - e^{i2\pi15t}$. Blue denotes the real and red denotes the imaginary component of the signal.

¹ The general principle behind the algorithm is that if $\omega_1 = \omega\ell_1$ and $\omega_2 = \omega\ell_2$ with ℓ_1 and ℓ_2 integers, then $\omega_1 - \omega_2 = \omega(\ell_1 - \ell_2)$ is still divisible with ω .

with the remainder: $(2\pi 9, 2\pi 6)$. We repeat this process until both numbers are the same. All the iterations are shown below:

$$\begin{aligned} & (2\pi 15, 2\pi 9) \\ & (2\pi 15 - 2\pi 9, 2\pi 9) = (2\pi 6, 2\pi 9) \\ & (2\pi 6, 2\pi 9 - 2\pi 6) = (2\pi 6, 2\pi 3) \\ & (2\pi 6 - 2\pi 3, 2\pi 3) = (2\pi 3, 2\pi 3). \end{aligned}$$

And we thus find that the fundamental frequency is $\omega = 2\pi 3$ and, using Equation 8.9, we find that $T = 1/3$.

THE FOURIER SERIES IS A SPECTRAL REPRESENTATION OF A PERIODIC FUNCTION and is defined as

$$x_N(t) = \sum_{k=-N}^N c_k e^{i \frac{2\pi}{T} kt}. \quad (8.14)$$

The fundamental period of a periodic signal is related to the fundamental angular frequency as follows: $T = 2\pi/\omega$. We can turn this around and express the fundamental angular frequency as a function of the fundamental period $\omega = 2\pi/T$, as we have done above.

This Fourier series is named after Jean-Baptiste Joseph Fourier (1768–1830). He introduced this representation for the purpose of solving the heat equation using a series of sinusoidal characteristic solutions. Perhaps the most revolutionary outcome of his study was that a wide range of functions can be represented as a sum of elementary sinusoidal functions.

Given a set of Fourier series coefficients c_k , we can synthesize a signal $x_N(t)$. This procedure is called *synthesis*. This formula shown in Equation 8.14 is also called the Fourier series synthesis equation.²

The inverse operation, determining coefficients $c_k \in \mathbb{C}$ from a certain periodic signal $x(t)$, is called *analysis*. We'll slowly work our way to this formula.

We will not study the convergence of the Fourier series, which means investigating if $\lim_{N \rightarrow \infty} x_N(t) = x(t)$. This is outside the scope of this course.

IN ORDER TO FIND THE ANALYSIS FORMULA, WE'LL FIRST NEED TO INTRODUCE THE CONCEPT OF BASIS FUNCTIONS AND AN INNER PRODUCT. A Fourier series basis function is defined as $\psi_k(t) = e^{i \frac{2\pi}{T} kt}$. We can use this to write the Fourier series in the following form:

$$x_N(t) = \sum_{k=-N}^N c_k e^{i \frac{2\pi}{T} kt} = \sum_{k=-N}^N c_k \psi_k(t). \quad (8.15)$$



Figure 8.4: Jean-Baptiste Joseph Fourier

² Synthesis:

$$c_{-N}, c_{-N+1}, \dots, c_N \rightarrow x_N(t)$$

Analysis:

$$x(t) \rightarrow c_{-N}, c_{-N+1}, \dots, c_N$$

The terms $\psi_k(t)$ can be seen as a set of orthogonal basis functions for a periodic function with a period T . What does it mean that basis functions $\psi_k(t)$ are orthogonal? We'll first need to define an inner product.

For periodic functions $a(t)$ and $b(t)$ with period T , an inner product is defined as:

$$\langle a(t), b(t) \rangle = \int_{t_0}^{t_0+T} a(t)b^*(t)dt = \int_T a(t)b^*(t)dt , \quad (8.16)$$

where t_0 is an arbitrary value of time, and T is the fundamental period of the signal. Because of the periodicity of the signals, we can evaluate the integral at any offset t_0 we wish to. The right-hand side is just an alternative way to denote the equation in the middle, i.e., that we integrate over one period of the periodic function.

When taking the inner product between two basis functions, we obtain:

$$\langle \psi_\ell(t), \psi_k(t) \rangle = \int_T \psi_\ell(t)\psi_k^*(t)dt \quad (8.17)$$

$$= \int_T e^{i\frac{2\pi}{T}(\ell-k)t} dt . \quad (8.18)$$

Let's break this down into two cases. First let's look at $\ell = k$. We know that $e^{i(2\pi t/T)(\ell-\ell)} = e^{i \cdot 0} = 1$. It is easy to see that:

$$\langle \psi_\ell(t), \psi_\ell(t) \rangle = \int_0^T dt \quad (8.19)$$

$$= T . \quad (8.20)$$

I've arbitrarily chosen to use $t_0 = 0$ when evaluating the integral (see the definition of the inner product in Equation 8.16). I would have obtained the same result with any value of t_0 .

Let's look at the other case where $\ell \neq k$. This means that the frequencies of the complex sinusoidal signals are different. We know that $\ell - k$ is a non-zero integer. Evaluating the inner product in this case gives us:

$$\langle \psi_\ell(t), \psi_k(t) \rangle = \int_0^T e^{i\frac{2\pi}{T}t(\ell-k)} dt \quad (8.21)$$

$$= \frac{T}{2\pi i(\ell-k)} e^{i\frac{2\pi}{T}t(\ell-k)} \Big|_{t=0}^T \quad (8.22)$$

$$= \frac{T}{2\pi i(\ell-k)} \left(e^{i\frac{2\pi}{T}T(\ell-k)} - e^{i\frac{2\pi}{T}0(\ell-k)} \right) \quad (8.23)$$

$$= \frac{T}{2\pi i(\ell-k)} (1 - 1) \quad (8.24)$$

$$= 0 . \quad (8.25)$$

We can combine these two results conveniently as:

$$\langle \psi_\ell(t), \psi_k(t) \rangle = T\delta_{\ell,k} . \quad (8.26)$$

The integral thus evaluates to T when $\ell = k$ and zero otherwise.

We have used the Kronecker delta function $\delta_{k,\ell}$ for convenience. It is defined as:

$$\delta_{k,\ell} = \begin{cases} 1 & \text{when } \ell = k \\ 0 & \text{when } \ell \neq k \end{cases}. \quad (8.27)$$

This is the definition of *orthogonality of basis functions*. We now have what we need to introduce the analysis formula, which relies on orthogonality.

THE FOURIER SERIES ANALYSIS FORMULA IS DEFINED AS:

$$c_k = \frac{1}{T} \langle x(t), \psi_k(t) \rangle = \frac{1}{T} \int_T x(t) \psi_k^*(t) dt = \frac{1}{T} \int_T x(t) e^{-i\frac{2\pi}{T} kt} dt. \quad (8.28)$$

This formula can be used to determine what the Fourier series coefficients are for a periodic function $x(t)$. The proof for the analysis formula relies on the orthogonality of basis functions. We'll make the assumption that there exists a Fourier series representation for the signal $x(t)$ in the form of an infinite sum and that this sum converges uniformly³:

$$x(t) = \sum_{\ell=-\infty}^{\infty} c_{\ell} e^{i\frac{2\pi}{T} \ell t}. \quad (8.29)$$

If this is the case, then the inner product, i.e., the analysis equation, has the following result:

$$\frac{1}{T} \langle x(t), \psi_k(t) \rangle = \frac{1}{T} \int_T \left(\sum_{\ell=-\infty}^{\infty} c_{\ell} e^{i\frac{2\pi}{T} \ell t} \right) e^{-i\frac{2\pi}{T} kt} dt \quad (8.30)$$

$$= \frac{1}{T} \sum_{\ell=-\infty}^{\infty} \int_T c_{\ell} e^{i\frac{2\pi}{T} \ell t} e^{-i\frac{2\pi}{T} kt} dt \quad (8.31)$$

$$= \frac{1}{T} \sum_{\ell=-\infty}^{\infty} c_{\ell} \int_T e^{i\frac{2\pi}{T} (\ell-k)t} dt \quad (8.32)$$

$$= \frac{1}{T} \sum_{\ell=-\infty}^{\infty} c_{\ell} T \delta_{\ell,k} \quad (8.33)$$

$$= c_k. \quad (8.34)$$

This proves that we can go back and forth between the Fourier synthesis and analysis formula, as long as there exists a Fourier series representation that converges uniformly for the periodic signal $x(t)$.

WE NOW HAVE THE SYNTHESIS AND ANALYSIS FORMULAS FOR A FOURIER SERIES. The synthesis formula allows us to obtain the time domain representation of a periodic signal using Fourier coefficients:

$$x_N(t) = \sum_{k=-N}^N c_k e^{i\frac{2\pi}{T} kt}. \quad (8.35)$$

³ The mathematician A.N. Kolmogorov pointed out in his 1923 paper several examples of functions for which the Fourier series diverges. The topic Fourier series convergence is beyond the scope of this course.

Andrey Kolmogoroff. Une série de Fourier-Lebesgue divergente presque partout. *Fundamenta mathematicae*, 1(4):324–328, 1923

The inverse operation, i.e., obtaining the Fourier coefficients c_k from a periodic function $x(t)$, is as follows:

$$c_k = \frac{1}{T} \int_T x(t) e^{-i\frac{2\pi}{T}kt} dt . \quad (8.36)$$

WHAT ABOUT REAL-VALUED SIGNALS? We can use Euler's formula to look at the real part of the Fourier series synthesis formula. I'll assume again that $x(t)$ is a periodic function that can be represented using an infinite sum of complex sinusoidal signals:

$$x(t) = \sum_{\ell=-\infty}^{\infty} c_{\ell} e^{i\frac{2\pi}{T}\ell t} . \quad (8.37)$$

The real part of the signal $x(t)$ is:

$$\operatorname{Re}\{x(t)\} = \frac{1}{2}(x(t) + x^*(t)) \quad (8.38)$$

$$= \frac{1}{2} \sum_{\ell=-\infty}^{\infty} c_{\ell} e^{i\frac{2\pi}{T}\ell t} + \frac{1}{2} \sum_{\ell=-\infty}^{\infty} c_{\ell}^* e^{-i\frac{2\pi}{T}\ell t} \quad (8.39)$$

$$= \frac{1}{2}(c_0 + c_0^*) + \frac{1}{2} \sum_{\ell=1}^{\infty} [(c_{\ell} + c_{-\ell}^*) e^{i\frac{2\pi}{T}\ell t} + (c_{\ell}^* + c_{-\ell}) e^{-i\frac{2\pi}{T}\ell t}] \quad (8.40)$$

$$= A_0 + \sum_{\ell=1}^{\infty} A_{\ell} \cos\left(\frac{2\pi}{T}\ell t + \phi_{\ell}\right) . \quad (8.41)$$

For the last part, I've introduced new variables: $A_0 = \frac{1}{2}(c_0 + c_0^*) \in \mathbb{R}$, $A_{\ell} = |c_{\ell} + c_{-\ell}^*| \in \mathbb{R}_{\geq 0}$ and $\phi_{\ell} = \angle(c_{\ell} + c_{-\ell}^*) \in \mathbb{R}$.

When analyzing Fourier series coefficients for a real-valued function, you will find that they come in conjugate symmetric pairs, which will allow you to write the Fourier series synthesis equation in the form shown on the last line of Equation 8.41. The Fourier series synthesis equation is sometimes introduced in this form. But this only applies to real valued signals!

THE ZERO FREQUENCY OR THE DIRECT CURRENT (DC) FREQUENCY COMPONENT c_0 often has a special meaning. In the case of real-valued signals, this is the only frequency component that does not have a positive and negative frequency conjugate symmetric pairing. The DC component also has the meaning that it is the mean value of the signal:

$$c_0 = \frac{1}{T} \int_T x(t) dt . \quad (8.42)$$

A PARTIAL SUM FOURIER SERIES CAN BE USED TO APPROXIMATE A PERIODIC SIGNAL. This is especially useful, if one knows that the

signal is band-limited, and one knows *a priori* that higher order terms (high frequency components) are not present in the signal, or they have such low magnitudes that they are not significant.

A partial sum synthesis formula only uses a subset of the Fourier coefficients:

$$x_S(t) = \sum_{k \in S} c_k e^{j2\pi kt/T}, \quad (8.43)$$

where S is a set of Fourier series coefficient indices.

A partial sum approximation of this type is often used in audio, image, and video compression, which rely on storing only a small subset of Fourier series coefficients, which typically have the largest magnitudes, leaving out the frequency components with small magnitudes.

EXAMPLE: LET'S REPRESENT THE SO-CALLED DIRAC COMB USING A FOURIER SERIES. The Dirac comb is a signal that consists of a train of unit impulse functions spaced T apart from one another. It is defined as:

$$x(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT). \quad (8.44)$$

This signal is shown in Figure 8.5. We'll encounter this signal when introducing the model for discretizing a continuous-time signal.

The fundamental period for this signal is T . Therefore, one would expect that we can form a Fourier series representation of this signal. What are the values of the coefficients c_k ? Let's find out using the analysis formula.

We'll evaluate the analysis integral formula from $t = -T/2$ to $t = T/2$, as this is convenient in the case of this signal. The analysis formula evaluates to:

$$c_k = \langle x(t), \psi_k(t) \rangle = \frac{1}{T} \int_{-T/2}^{T/2} \delta(t) e^{-i\frac{2\pi}{T}kt} dt \quad (8.45)$$

$$= \frac{1}{T}. \quad (8.46)$$

The signal can thus be represented as the following Fourier series:

$$x_N(t) = \frac{1}{T} \sum_{k=-N}^N e^{i\frac{2\pi}{T}kt}. \quad (8.47)$$

Take a look at Figure 8.6. This is an approximation of the Dirac comb $x_7(t)$ using 15 frequency components. I hope it is not too difficult for you to convince yourself that as we increase the value of N , the spikes will become sharper and sharper.

You should also be able to convince yourself that this signal is real-valued, by investigating the pairing of positive and negative

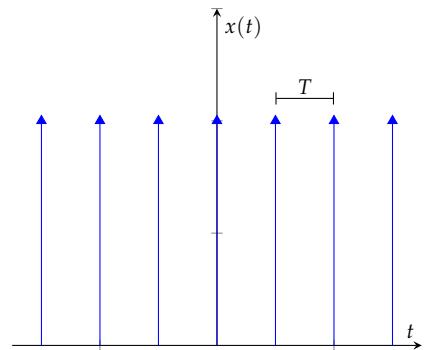


Figure 8.5: The Dirac comb signal, an infinitely long train of unit impulses spaced apart by T . The Dirac comb is a periodic function with a fundamental period T . The Dirac comb is used to model sampling values of a continuous-time signal spaced evenly apart to provide an idealized model for discretizing a continuous-time signal.

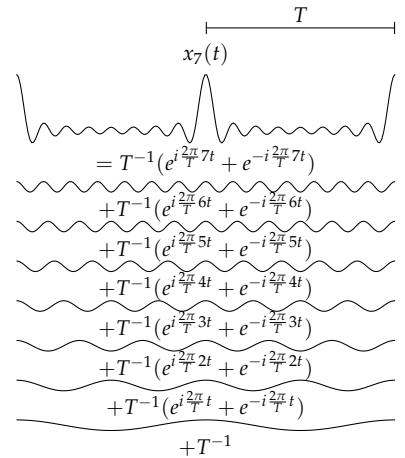


Figure 8.6: A Fourier series representation $x_7(t) = \frac{1}{T} \sum_{k=-7}^7 e^{i\frac{2\pi}{T}kt}$ of a Dirac comb signal $x(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT)$ with a period T . Two periods of the signal are shown. The Fourier series representation of the signal is made using seven sinusoidal signals.

frequency terms:

$$x_N(t) = \frac{1}{T} \sum_{k=-N}^N e^{i\frac{2\pi}{T}kt} \quad (8.48)$$

$$= \frac{1}{T} + \frac{1}{T} \sum_{k=1}^N (e^{i\frac{2\pi}{T}kt} + e^{-i\frac{2\pi}{T}kt}) \quad (8.49)$$

$$= \frac{1}{T} + \frac{2}{T} \sum_{k=1}^N \cos\left(\frac{2\pi}{T}kt\right). \quad (8.50)$$

This is how the positive and negative frequency terms are organized in Figure 8.6. For real-valued Fourier series representations, this sort of grouping of positive and negative frequency coefficients always occurs.

EXAMPLE: FOURIER SERIES REPRESENTATION FOR A SQUARE WAVE.

A square wave is a classic example of a periodic function that can be evaluated using a Fourier series. This type of signal is often used, e.g., in the pulse width modulation scheme of adjusting mean power flowing through a circuit. This type of on-off pulsing scheme is also used in avalanche rescue beacons used to radiolocate victims buried under snow. The rectangular function is shown in Figure 8.7. A measurement of power emitted by an avalanche rescue beacon as a function of time is shown in Figure 8.8.

Consider a general square wave signal with period T , pulse length P and amplitude 1. Between $0 \leq t < T$, this signal is mathematically described with:

$$x(t) = \begin{cases} 1 & \text{when } 0 \leq t < P \\ 0 & \text{otherwise} \end{cases}. \quad (8.51)$$

By analytically evaluating the Fourier series analysis integral one has:

$$c_k = \frac{1}{T} \int_0^T x(t) e^{-i2\pi kt/T} dt, \quad (8.52)$$

we can obtain an equation for coefficients c_k . In the case of $k = 0$, the integral evaluates to:

$$c_0 = \frac{1}{T} \int_0^T x(t) e^{i \cdot 0} dt \quad (8.53)$$

$$= \frac{1}{T} \int_0^P dt \quad (8.54)$$

$$= \frac{P}{T}. \quad (8.55)$$

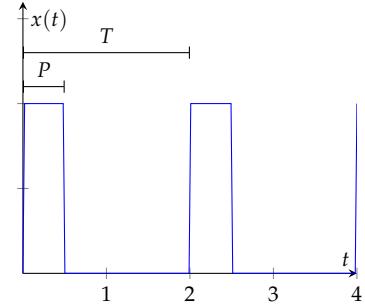


Figure 8.7: A classic example of a commonly encountered periodic function, the square wave signal. This type of signal is encountered, e.g., with pulse width modulation in electrical engineering.

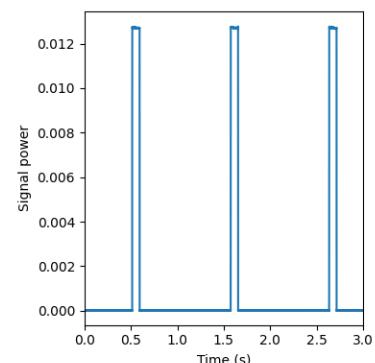


Figure 8.8: A measurement of power as a function of time transmitted by an avalanche rescue locator beacon. Approximately a 0.1 second pulse is emitted every second.

For other values of $k \neq 0$:

$$c_k = \frac{1}{T} \int_0^T x(t) e^{-i\frac{2\pi}{T}tk} dt \quad (8.56)$$

$$= \frac{1}{T} \int_0^P e^{-i\frac{2\pi}{T}tk} dt \quad (8.57)$$

$$= -\frac{1}{2i\pi k} e^{-i\frac{2\pi}{T}tk} \Big|_{t=0}^P \quad (8.58)$$

$$= -\frac{1}{2i\pi k} (e^{-i\frac{2\pi}{T}kP} - 1) \quad (8.59)$$

$$= \frac{1}{2i\pi k} e^{-i\frac{\pi}{T}kP} (e^{i\frac{\pi}{T}kP} - e^{-i\frac{\pi}{T}kP}) \quad (8.60)$$

$$= \frac{1}{\pi k} e^{-i\frac{\pi}{T}kP} \sin\left(\frac{\pi}{T}kP\right). \quad (8.61)$$

We have used: the inverse Euler's formula for the $\sin(\cdot)$ signal, the fact that $e^{-i\frac{\pi}{T}kP} e^{i\frac{\pi}{T}kP} = 1$, and that $\sin(-\theta) = -\sin(\theta)$.

Now we can evaluate the synthesis formula to form a Fourier series approximation of this function:

$$x_N(t) = c_0 + \sum_{k=1}^N c_k e^{i\frac{2\pi}{T}kt} + c_{-k} e^{-i\frac{2\pi}{T}kt} \quad (8.62)$$

$$= \frac{P}{T} + \sum_{k=1}^N \frac{\sin\left(\frac{\pi}{T}kP\right)}{\pi k} \left(e^{-i\frac{\pi}{T}kP} e^{i\frac{2\pi}{T}kt} + e^{i\frac{\pi}{T}kP} e^{-i\frac{2\pi}{T}kt} \right) \quad (8.63)$$

$$= \frac{P}{T} + \sum_{k=1}^N \frac{2 \sin\left(\frac{\pi}{T}kP\right)}{\pi k} \cos\left(\frac{2\pi}{T}kt - \frac{\pi}{T}kP\right). \quad (8.64)$$

Let's see what this function looks like. I've written a little Python program to visualize the Fourier series approximation of this function. The Python code is found in Listing 8.1. The output of this program is shown in Figure 8.9.

```
import matplotlib.pyplot as plt
import numpy as np

P = 0.1
T = 1.0

sample_rate = 1000.0
t = np.arange(int(sample_rate*T))/sample_rate

zn = np.zeros(len(t), dtype=np.complex64)

N = 101

ks = np.arange(-N, N+1)
cks = np.zeros(2*N+1, dtype=np.complex64)
ki = 0

for k in range(-N, N+1):
    if k == 0:
        zn += P/T
    else:
        cks[ki] = 2 * np.sin((2 * np.pi / T) * k * P) / (np.pi * k)
        zn += cks[ki] * np.exp(1j * 2 * np.pi / T * k * t)
        ki += 1
```

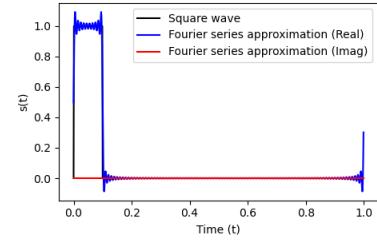


Figure 8.9: Fourier series approximation of a square wave evaluated with the Python script 009_square_wave/square_wave.py.

```

        cks[ki] = P/T
    else:
        k2 = float(k)
        ck = (1.0/(np.pi*k2))*(np.exp(-1j*(np.pi/T)*k2*T))*np.
        sin(np.pi*k2*T)
        cks[ki] = ck
        zn += ck*np.exp(1j*(2.0*np.pi/T)*k2*t)
    ki += 1

# Plot the absolute values of the Fourier series coefficients.
plt.plot(2.0*np.pi*ks/T, np.abs(cks))
plt.show()

# Create a square wave signal.
square_wave = np.zeros(len(t), dtype=np.float64)
square_wave[(0 < t) & (t < P)] = 1.0

plt.figure(figsize=(0.8*6, 0.8*4))
plt.plot(t, square_wave, label="Square wave", color="black")
plt.plot(t, zn.real, label="Fourier series approximation (Real)",
         color="blue")
plt.plot(t, zn.imag, label="Fourier series approximation (Imag)"
         , color="red")
plt.legend()
plt.xlabel("Time (t)")
plt.ylabel("s(t)")
plt.tight_layout()
plt.savefig("square_wave.png")
plt.show()

```

Listing 8.1: 009_square_wave/square_wave.py

THE TIME DERIVATIVE OPERATOR $\frac{d}{dt}$ HAS A VERY SIMPLE FORM FOR A FOURIER SERIES REPRESENTATION OF A SIGNAL.

We know that a Fourier series is a sum of complex sinusoidal signals. We also know that it is easy to differentiate this signal:

$$\frac{d}{dt} ce^{i\omega t} = i\omega c e^{i\omega t}, \quad (8.65)$$

where $c \in \mathbb{C}$ is a complex constant.

It is also relatively easy to see that the n th time derivative is:

$$\frac{d^n}{dt^n} ce^{i\omega t} = (i\omega)^n c e^{i\omega t}. \quad (8.66)$$

Thus, if $x(t)$ has a Fourier series representation

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi}{T}kt}, \quad (8.67)$$

then the n th time derivative has the following Fourier series representation:

$$\frac{d^n}{dt^n} x(t) = \sum_{k=-\infty}^{\infty} i^n \left(\frac{2\pi k}{T}\right)^n c_k e^{i\frac{2\pi}{T}kt} \quad (8.68)$$

$$= \sum_{k=-\infty}^{\infty} c_k^{(n)} e^{i\frac{2\pi}{T}kt}. \quad (8.69)$$

It is still a Fourier series, but with each Fourier coefficient modified in the following way:

$$c_k^{(n)} := i^n \left(\frac{2\pi k}{T} \right)^n c_k . \quad (8.70)$$

One thing to ask yourself is how does differentiation adjust the amplitude of each Fourier coefficient? When taking the n th time derivative, what happens to the complex amplitudes at low angular frequencies ($|2\pi k/T| < 1$)? What about high angular frequencies ($|2\pi k/T| > 1$)? Hint, inspect how the coefficients defined in Equation 8.70 are scaled.

The answer is that the magnitude of the low frequency Fourier coefficients are attenuated, and high frequency ones are amplified. The DC component is completely removed as $2\pi k/T = 0$ for $k = 0$. Differentiation can be seen as a filtering operation on the signal. A filter is something that modifies the amplitude and phase of the Fourier series coefficients in some way. For example, a *low-pass filtering* operation would reduce the amplitudes of high frequency coefficients relative to the amplitudes of the low frequency coefficients. After a low-pass filtering operation, the ratio of low frequency coefficient amplitudes to the high frequency coefficient amplitudes would be increased. A high pass filter on the other hand would do the opposite. Differentiation can therefore be viewed as a *high-pass filtering* operation.

THE TIME SHIFT OPERATION $x(t - \tau)$ APPLIED TO A FOURIER SERIES REPRESENTATION is also pretty handy. Let's recall how a time shift affects a complex sinusoidal signal:

$$x(t) = ce^{i\omega t} . \quad (8.71)$$

If we apply a delay by τ to this signal $y(t) = x(t - \tau)$, we obtain:

$$y(t) = ce^{i\omega(t-\tau)} = e^{-i\omega\tau}ce^{i\omega t} = e^{-i\omega\tau}x(t) . \quad (8.72)$$

This means that a time delay by τ corresponds to a phase shift by $-\omega\tau$ for a complex sinusoidal signal. We can apply this to a Fourier series, which is a sum of complex sinusoidal signals.

If the Fourier series representation of a signal is:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{i\frac{2\pi}{T}kt} , \quad (8.73)$$

then the Fourier series representation of a time shifted signal is:

$$x(t - \tau) = \sum_{k=-\infty}^{\infty} \left(c_k e^{-i\frac{2\pi}{T}k\tau} \right) e^{i\frac{2\pi}{T}kt} \quad (8.74)$$

$$= \sum_{k=-\infty}^{\infty} c'_k e^{i\frac{2\pi}{T}kt} \quad (8.75)$$

where c'_k is rotated on the complex plane in the clockwise direction by an angle of $2\pi k\tau/T$ with respect to c_k .

This means that if you know the Fourier series representation of a signal, you can easily modify the complex amplitude c_k of each Fourier coefficient to obtain the coefficients of the time shifted signal.

The linear relationship between phase shift and time delay of frequency components of signals is used extensively throughout signal processing. For example, a network analyzer measures an accurate time delay in a cable by investigating how the phase in the cable changes as a function of frequency.

WHAT IF A SIGNAL IS NOT PERIODIC, CAN WE STILL FORM A SPECTRAL REPRESENTATION FOR A SIGNAL? Yes, we can. This investigation leads to the *continuous-time Fourier transform*.

Let's derive the continuous-time Fourier transform from the Fourier series. Pretty much all we'll need to do is to study what happens when the fundamental period of the signal approaches infinity: $T \rightarrow \infty$. We'll arrive with the result that introduces the Fourier transform, and the inverse Fourier transform, introducing, for the first time, this general spectral representation for continuous-time signals.

We start the derivation of the continuous-time Fourier transform with the definition of a Fourier series for a function with period T . We'll denote by $\Delta\omega$ the spacing between frequency components. The term $\Delta\omega$ also denotes the fundamental angular frequency of the periodic signal:

$$x_T(t) = \sum_{k=-\infty}^{\infty} c_k e^{ik\Delta\omega t}. \quad (8.76)$$

We can now add the definition of c_k corresponding to the k th Fourier series coefficient using the synthesis equation:

$$x_T(t) = \sum_{k=-\infty}^{\infty} \left[\frac{1}{T} \int_{-T/2}^{T/2} x_T(t) e^{-ik\Delta\omega t} dt \right] e^{ik\Delta\omega t}. \quad (8.77)$$

We then note that $1/T = \Delta\omega/2\pi$. This is the relationship between the fundamental period of the signal and the fundamental angular frequency that we discussed in the beginning of this chapter.

We now obtain:

$$x_T(t) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left[\int_{-T/2}^{T/2} x_T(t) e^{-ik\Delta\omega t} dt \right] e^{ik\Delta\omega t} \Delta\omega \quad (8.78)$$

$$= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \hat{x}_T(k\Delta\omega) e^{ik\Delta\omega t} \Delta\omega. \quad (8.79)$$

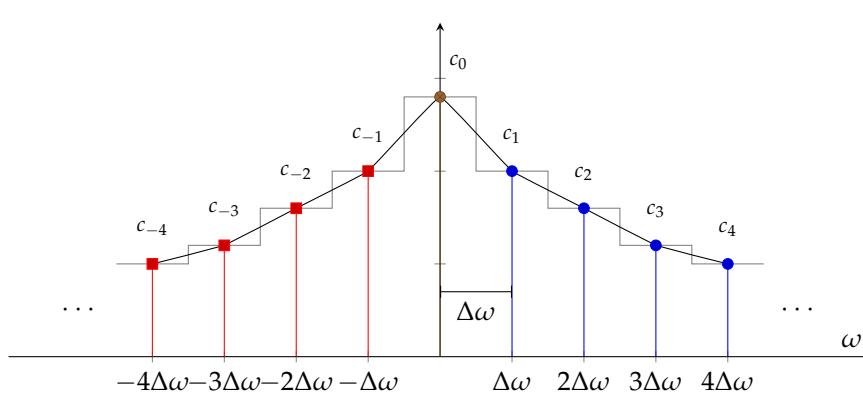


Figure 8.10: A depiction of how the Fourier series can be seen as a Riemann sum approximation of an integral of a continuous function.

In this form, the equation is a Riemann sum approximation of an integral of a continuous function $\hat{x}_T(\omega)e^{-i\omega t}$.

We now inspect the case where $T \rightarrow \infty$. When the period of the function $x_T(t)$ approaches infinity, we drop the subscript T in the notation and denote it with $x(t)$. The Riemann sum then approaches the following integral equation ($\Delta\omega \rightarrow d\omega$, $k\Delta\omega \rightarrow \omega$):

$$\lim_{T \rightarrow \infty} x_T(t) = \lim_{T \rightarrow \infty} \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \hat{x}_T(k\Delta\omega) e^{ik\Delta\omega t} \Delta\omega \quad (8.80)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \right] e^{i\omega t} d\omega \quad (8.81)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega \quad (8.82)$$

$$= x(t) . \quad (8.83)$$

This integral is known as the inverse Fourier transform. It allows us to go from a continuous function $\hat{x}(\omega)$, which is the continuous spectral representation of the signal, to the time domain signal $x(t)$.

The innermost integral that allows us to obtain $\hat{x}(\omega)$ from the continuous-time signal $x(t)$ is:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt . \quad (8.84)$$

This is the forward Fourier transform.

THE FORWARD FOURIER TRANSFORM is defined as:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt .$$

(8.85)

THE INVERSE FOURIER TRANSFORM is defined as:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega . \quad (8.86)$$

Just as the Fourier series, the continuous-time Fourier transform is in most practical cases invertible.

We'll use the following notation to indicate Fourier transform pairs:

$$x(t) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) . \quad (8.87)$$

We'll return to the Fourier transform later and show that the Fourier series is a special case of the Fourier transform!

The spectrum of an audio signal

The Python program in Listing 8.2 shows how to use the Fast Fourier Transform (FFT) command in Python to analyze an audio signal for its spectral contents. The output of this program is shown in Figure 8.11.

The FFT algorithm is used to evaluate a discrete-time Fourier transform. This operation is mathematically defined as follows:

$$c[k] = \sum_{t=0}^{N-1} x[t] e^{-i\frac{2\pi}{N}kt} . \quad (8.88)$$

Here $c[k]$ is the complex amplitude corresponding to a spectral component with angular frequency $\omega_k = 2\pi k / (T_s N)$, where $T_s = 1/f_s$ is the sample-spacing, which is the inverse of the sample-rate of the signal.

You can think of an FFT as a discrete-time equivalent of the Fourier series analysis equation, which finds the Fourier series coefficients for a periodic discrete-time signal. We will cover the discrete Fourier transform and discrete-time signals in more detail later on, I just wanted to already expose you to the FFT algorithm at this point.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sio

wav = sio.read("guitar_clean.wav")
# Figure out what the sample-rate is.
sample_rate = wav[0]
# Read the signal for only one stereo channel,
# select samples inside the file.
x = wav[1][50000:60000, 0]

# How many samples to plot and analyse.
N_samples = 1024

# Calculate spectral components using FFT
# (discrete-time Fourier series.)
```

```

c_k = np.fft.fft(x[:N_samples])

# Time in seconds.
time_vec = np.arange(N_samples)/float(sample_rate)

# Create a vector with frequencies in kHz.
freq_vec = np.fft.fftshift(np.fft.fftfreq(N_samples, d=1.0/
    sample_rate))/1e3
# What is magnitude squared of each spectral component in dB.
power_db = np.fft.fftshift(10.0*np.log10(np.abs(c_k)**2.0))

# Plot time domain signal.
plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.plot(1e3*time_vec, x[:1024])
plt.xlabel("Time (ms)")
plt.ylabel("Relative air pressure (unitless)")
plt.title("Time domain signal")

# Plot frequency domain signal.
plt.subplot(122)
plt.plot(freq_vec, power_db)
plt.title("Spectral components")
plt.xlabel("Frequency (kHz)")
plt.ylabel("Power (dB)")
plt.tight_layout()
plt.savefig("audio_spec.png")
plt.show()

```

Listing 8.2: 013_audio_spec/audio_spec.py

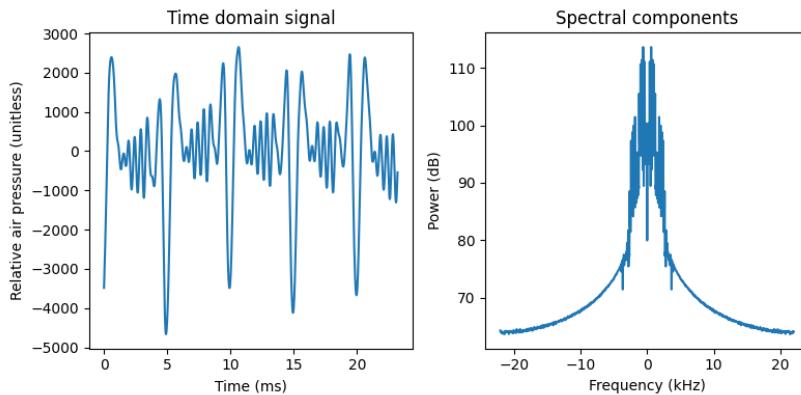


Figure 8.11: The spectral components of an audio signal (guitar string plucked). Left: time domain signal $x(t)$, Right: The magnitude of the spectral components in decibel scale: $10 \log_{10}(|c_k|^2)$.

Exercises: Fourier Series

1. A signal is defined as:

$$x(t) = 7 \sin(3\pi t + 0.2\pi) + 3 \cos(7\pi t + 0.5\pi) \quad (8.89)$$

The independent variable t is in units of seconds.

- a) Express this signal in the form of Equation 8.1. What are the values of the coefficients $c_k \in \mathbb{C}$ and angular frequencies $\omega_k \in \mathbb{R}$?
- b) The coefficients c_k , satisfy $c_{-k} = c_k^*$, why?
- c) Show that this signal is periodic by showing that the frequencies of the individual frequency components are commensurable.
- d) Use Euclid's algorithm to find the fundamental angular frequency ω . What is the fundamental frequency in units of hertz and rad/s?
- e) What is the fundamental period T of this signal in seconds?
- f) We delay the signal $x(t)$ by 0.5 seconds $y(t) = x(t - \frac{1}{2})$. Write $y(t)$ in the following format:

$$y(t) = 7 \sin(3\pi t + \phi_0) + 3 \cos(7\pi t + \phi_1) \quad (8.90)$$

What are the values ϕ_0 and ϕ_1 ?

- g) We create a new signal $z(t) = x(t) + e^{i\sqrt{2}t+13}$. Why is the signal $z(t)$ no longer periodic with a finite period? Is the signal $z(t)$ real-valued?

2. A signal is defined as:

$$x(t) = e^{-i(6\pi t+0.3)} + 4e^{i(60\pi t+0.42)} + 4e^{-i(60\pi t+0.42)} + e^{i(6\pi t+0.3)} \quad (8.91)$$

The independent variable t is in units of seconds.

- a) Why is this signal real-valued?
- b) Is this signal periodic? If so, what is the fundamental angular frequency ω and the fundamental period T ?
- 3. The Fourier series coefficients for a pulsed signal $x(t)$ with a fundamental period $T = 1$ is described as:

$$c_k = \begin{cases} \frac{1}{10} & \text{when } k = 0 \\ \frac{1}{\pi k} e^{-i\frac{\pi}{10}k} \sin\left(\frac{\pi}{10}k\right) & \text{otherwise} \end{cases} \quad (8.92)$$

See Equations 8.55 and 8.61 for the derivation of the Fourier coefficients.

- a) Plot the partial sum $x_N(t)$ of the Fourier series using $N = 101$.
You can use Listing 8.1 to help you out.
- b) What are the Fourier series coefficients for a signal delayed in time by 0.2 seconds. Plot the partial sum to verify.
- c) What are the Fourier series coefficients for $y(t) = \frac{d}{dt}x(t)$? Make a plot to verify.

Suggested solutions: Fourier Series

1. Let $x(t)$ be the signal:

$$x(t) = 7 \sin(3\pi t + 0.2\pi) + 3 \cos(7\pi t + 0.5\pi).$$

a) Have that:

$$\begin{aligned}\sin \theta &= \frac{1}{2i}(e^{i\theta} - e^{-i\theta}), \\ \cos \theta &= \frac{1}{2}(e^{i\theta} + e^{-i\theta}).\end{aligned}$$

This gives:

$$\begin{aligned}x(t) &= \frac{7}{2i}(e^{i(3\pi t+0.2\pi)} - e^{-i(3\pi t+0.2\pi)}) + \frac{3}{2}(e^{i(7\pi t+0.5\pi)} + e^{-i(7\pi t+0.5\pi)}), \\ &= \left(\frac{7}{2i}e^{i0.2\pi}\right)e^{i3\pi t} - \left(\frac{7}{2i}e^{-i0.2\pi}\right)e^{-i3\pi t} + \left(\frac{3}{2}e^{i0.5\pi}\right)e^{i7\pi t} + \left(\frac{3}{2}e^{-i0.5\pi}\right)e^{-i7\pi t},\end{aligned}$$

giving:

$$c_k = \left(\frac{3}{2}e^{-i0.5\pi}, -\frac{7}{2i}e^{-i0.2\pi}, \frac{7}{2i}e^{i0.2\pi}, \frac{3}{2}e^{i0.5\pi}\right),$$

for which the corresponding angular frequencies are:

$$\omega_k = (-7\pi, -3\pi, 3\pi, 7\pi).$$

- b) For any real signal, the Fourier coefficients satisfy $c_{-k} = c_k^*$ and $x(t)$ is real.
- c) The signal is periodic if $\omega_i/\omega_j \in \mathbb{Q}$ for every pair $i, j \in \{-7, -3, 3, 7\}$ with $i \neq j$. In this case every ω_i is an integer multiple of π , so every ratio is a rational number as all the π cancel.
- d) Using Euclid's algorithm we have:

$$\begin{aligned}&(3\pi, 7\pi), \\ &(3\pi, 7\pi - 3\pi), \\ &(3\pi, 4\pi), \\ &(3\pi, 4\pi - 3\pi), \\ &(3\pi, \pi), \\ &(3\pi - \pi, \pi), \\ &(2\pi, \pi), \\ &(2\pi - \pi, \pi), \\ &(\pi, \pi).\end{aligned}$$

Hence, the fundamental angular frequency is $\omega = \pi$ rad/s. In units of hertz, we have that $\omega = 2\pi f$, so

$$f = \frac{\omega}{2\pi} = \frac{\pi}{2\pi} = \frac{1}{2}.$$

Thus, the fundamental frequency is $f = 1/2$ Hz.

- e) If the fundamental angular frequency is $\omega = \pi$, then the fundamental period is related by $T = 2\pi/\omega$, hence

$$T = \frac{2\pi}{\omega} = \frac{2\pi}{\pi} = 2 \text{ seconds.}$$

- f) Define a new signal $y(t) = x(t - \frac{1}{2})$. That is, we delay the signal $x(t)$ by $\frac{1}{2}$. We get:

$$y(t) = 7 \sin\left(3\pi\left(t - \frac{1}{2}\right) + 0.2\pi\right) + 3 \cos\left(7\pi\left(t - \frac{1}{2}\right) + 0.5\pi\right).$$

Simplifying gives:

$$y(t) = 7 \sin(3\pi t - 1.3\pi) + 3 \cos(7\pi t - 3\pi).$$

Hence, $\phi_0 = -1.3\pi$ and $\phi_1 = -3\pi$.

- g) Define another signal $z(t) = x(t) + e^{i\sqrt{2}+13}$. This new signal is not periodic as the frequencies are not commensurable, since we have a factor of $\sqrt{2}$, the signal is also not real-valued since we don't have a corresponding complex conjugate pair.

2. Let $x(t)$ be a signal defined as:

$$x(t) = e^{-i(6\pi t+0.3)} + 4e^{i(60\pi t+0.42)} + 4e^{-i(60\pi t+0.42)} + e^{i(6\pi t+0.3)},$$

where t is measured in seconds.

- a) This signal is real-valued since we have two pairs, each with the same frequency, but different signs. Thus, the signal can be written as a real signal for which it takes the following form:

$$x(t) = 8 \cos(60\pi t + 0.42) + 2 \cos(6\pi t + 0.3).$$

- b) In this case, we have the angular frequencies of $\omega_1 = 60\pi$ and $\omega_2 = 6\pi$ for which $\omega_1/\omega_2 = 60\pi/6\pi = 10$. We get a rational number, so the signal is periodic.
- c) To find the fundamental angular frequency, we use Euclid's

algorithm. This gives:

$$\begin{aligned} & (60\pi, 6\pi), \\ & (54\pi, 6\pi), \\ & (48\pi, 6\pi), \\ & (42\pi, 6\pi), \\ & (36\pi, 6\pi), \\ & (30\pi, 6\pi), \\ & (24\pi, 6\pi), \\ & (18\pi, 6\pi), \\ & (12\pi, 6\pi), \\ & (6\pi, 6\pi), \end{aligned}$$

therefore, the fundamental angular frequency is 6π in units of radians per second.

- d) The fundamental period is:

$$T = \frac{2\pi}{\omega} = \frac{2\pi}{6\pi} = \frac{1}{3},$$

hence $T = \frac{1}{3}$ in units of seconds.

3. Let the Fourier series coefficients of a periodic signal $x(t)$ with fundamental period $T = 1$ be:

$$c_k = \begin{cases} \frac{1}{10}, & k = 0, \\ \frac{1}{\pi k} e^{-i\frac{\pi}{10}k} \sin\left(\frac{\pi}{10}k\right), & \text{otherwise.} \end{cases}$$

- a) Using Python, we can implement a simple program for the partial sum with $N = 101$. Listing 8.3 shows a way.

```
import matplotlib.pyplot as plt
import numpy as np

T = 1 # Period.
sample_rate = 1000.0 # Sample rate.
N = 101 # Number of terms.

# Partition the t-axis.
t = np.arange(int(sample_rate*T))/sample_rate

# Allocate an array for the signal.
xn = np.zeros(len(t), dtype=np.complex64)

def ck(k: int) -> float:
    """Function for the Fourier coefficients."""
    if k == 0:
        return 1/10
    else:
```

```

        return 1/(np.pi*k)*np.exp(-1j*np.pi*k/10)*np.sin(np
            .pi*k/10)

# Loop through the indices and add the signal terms.
for i in range(-N, N+1):
    xn += ck(i)*np.exp(1j*2*np.pi*i*t/T)

plt.plot(t, xn.real, color="blue")
plt.plot(t, xn.imag, color="red")
plt.xlabel("Time (t)")
plt.ylabel("$x_{[N]}(t)$")
# If needed.
# plt.plot()

```

Listing 8.3: Suggested solution to a)

- b) The Fourier coefficients of a delayed signal are related to the undelayed signal by $c'_k = e^{-i\frac{2\pi k \tau}{T}} c_k$. The script in Listing 8.3 can be modified to account for delay. The delayed version is shown in Listing 8.4.

```

import matplotlib.pyplot as plt
import numpy as np

T = 1                      # Period.
sample_rate = 1000.0         # Sample rate.
N = 101                     # Number of terms.
tau = 0.2                   # Delay.

# Partition the t-axis.
t = np.arange(int(sample_rate*T))/sample_rate

# Allocate an array for the signal.
xn = np.zeros(len(t), dtype=np.complex64)

def ck(k: int) -> float:
    """Function for the Fourier coefficients."""
    if k == 0:
        return 1/10
    else:
        return 1/(np.pi*k)*np.exp(-1j*np.pi*k/10)*np.sin(np
            .pi*k/10)

# Compute the delayed signal.
for i in range(-N, N+1):
    xn += ck(i)*np.exp(1j*2*np.pi*i*(t-tau)/T)

plt.plot(t, xn.real, color="blue")
plt.plot(t, xn.imag, color="red")
plt.xlabel("Time (t)")
plt.ylabel("$x_{[N]}(t)$")
# If needed.
# plt.plot()

```

Listing 8.4: Suggested solution to b)

- c) Define $y(t) = \frac{d}{dt}x(t)$. If c_k are the Fourier coefficients for $x(t)$, then the Fourier coefficients for $y(t)$ is:

$$d_k(t) = i\frac{2\pi k}{T}c_k.$$

Using Python we have Listing 8.5.

```
import matplotlib.pyplot as plt
import numpy as np

T = 1                      # Period.
sample_rate = 1000.0         # Sample rate.
N = 101                     # Number of terms.
tau = 0.0                    # Delay.

# Partition the t-axis.
t = np.arange(int(sample_rate*T))/sample_rate

# Allocate an array for the signal.
xn = np.zeros(len(t), dtype=np.complex64)

def ck(k: int) -> float:
    """Function for the Fourier coefficients."""
    if k == 0:
        return 1/10
    else:
        return 1/(np.pi*k)*np.exp(-1j*np.pi*k/10)*np.sin(np.pi*k/10)

# Derivative signal.
for i in range(-N, N+1):
    xn += (1j*2*np.pi*i/T)*ck(i)*np.exp(1j*2*np.pi*i*(t-tau)/T)

# Plot the result.
plt.plot(t, xn.real, color="blue")
plt.plot(t, xn.imag, color="red")
plt.xlabel("Time (t)")
plt.ylabel("$y_{\{N\}}(t)$")
# If needed.
# plt.plot()
```

Listing 8.5: Suggested solution to c)

Audio Compression Example [Optional]

In this application example, I'll show you how to estimate the spectral components of an audio signal and how this can be used for audio compression. I won't go very deep into either of these topics. The main idea is to expose you to these concepts.

EXAMPLE: AUDIO COMPRESSION ALGORITHMS OFTEN USE A SPARSE SPECTRAL REPRESENTATION OF AN AUDIO SIGNAL. The sounds produced by musical instruments when playing a single note are often good examples of periodic signals that occur in our daily life. Of course, real signals are typically not exactly periodic, but for short intervals this approximation is good.

A good example of this is the signal shown in Figure 8.11, which depicts the sound signal (instantaneous relative air pressure at the microphone) emitted by a guitar when one string is plucked.

A Fourier series approximation of an audio signal is often used in audio compression, as it is often possible to approximate an audio signal relatively accurately with relatively few non-zero Fourier series coefficients. This is the basic idea behind the MP3 audio compression algorithm⁴.

The Python code in Listing 8.6 demonstrates audio compression in practice. Understanding the program may require you to be somewhat familiar with analyzing what computer programs do. If you are just starting with programming, feel free to skip going through this program example.

The audio compression example program relies on the discrete Fourier transform (FFT), which we haven't covered yet. Just think of the `numpy.fft.fft` operation as the analysis step of the Fourier series, which gives you the phase and amplitude of each frequency component c_k . We'll cover the discrete Fourier transform in more detail at a later part of this course.

```
#!/usr/bin/env python
#
# Represent an audio signal with only a small number of
# spectral components (basic idea behind audio compression)
# Juha Vierinen, 2020
#
import scipy.io.wavfile as sw
import numpy as n
import matplotlib.pyplot as plt

# compress and decompress audio file
# compression_ratio=0.95 means that 95%
# of the smallest amplitude spectral
# components are thrown away
cr=0.9
```

⁴ Yes, there is more to it. Note, that real world compression algorithms use a lot more tricks, such as psychoacoustics – not storing spectral components that humans cannot hear very well in music. These are outside the scope of this basic course on signal processing though. However, it is already possible to achieve significant compression just using the simple idea of representing the signal only using the N largest amplitude spectral components. I would wager that most of the compression is achieved with the sparse spectral representation.

```

# read wav file
ts=sw.read("7na.wav")
sr=ts[0]      # sample rate
clip=ts[1]    # extract audio file as numpy data vector
if len(clip.shape)==2: # if stereo, only use one channel
    print("converting to mono")
    clip=ts[1][:,0]+ts[1][:,1]

# if clip is too long, make it shorter (1 million samples)
if len(clip)>1000000:
    clip=clip[0:1000000]

# Fourier transform to obtain the phase and amplitude of each
# spectral component
F = n.fft.fft(clip)
# Make a copy of the original for plotting purposes
F_orig=n.copy(F)

# what is the absolute value (amplitude) of each spectral
# component
mag=n.abs(F)

# sort by amplitude
idx=n.argsort(mag)

# figure out what is the smallest component that we'll include
smallest_comp=mag[idx[int(cr*len(idx))]]

# remove spectral components with magnitude less than
# smallest_comp
F[mag < smallest_comp]=0.0

# inverse Fourier transform to obtain signal composed of just a
# few spectral components
compressed=n.real(n.fft.ifft(F))

# plot the absolute values of the original and sparse spectral
# representation
freq_vec=n.fft.fftshift(n.fft.fftfreq(len(F),d=1/float(sr)))
plt.plot(freq_vec/1e3,n.fft.fftshift(10.0*n.log10(n.abs(F_orig)
    **2.0)),label="Original")
plt.xlim([0,float(sr)/1e3/2.0])
plt.plot(freq_vec/1e3,n.fft.fftshift(10.0*n.log10(n.abs(F)**2.0)
    ),label="Compressed")
plt.legend()
plt.xlabel("Frequency (kHz)")
plt.ylabel("Power (dB)")
plt.title("Spectrum")
plt.show()

# plot the first 10000 samples of the time-domain audio signal
time_vec=n.arange(len(clip))/float(sr)
plt.title("Waveform")
plt.plot(time_vec[0:10000]*1e3,clip[0:10000],label="Original")
plt.plot(time_vec[0:10000]*1e3,compressed[0:10000],label="Compressed")
plt.legend()
plt.xlabel("Time (ms)")
plt.ylabel("Audio waveform")
plt.show()

```

```
# Save result as wav file so that we can easily listen to the
# audio
# quality after decompression of the compressed signal.

# scale numbers to 0..0.95 scale
compressed = 0.95*compressed/n.max(compressed)
print("Saving spectrally sparse signal to file compressed_signal
.wav")
sw.write("compressed_signal.wav",44100,n.array(20e3*compressed,
dtype=n.int16))
```

Listing 8.6: 010_audio_compression/audio_compression.py

The algorithm discards weak spectral components of the signal and stores the strong ones. This can result in significant savings in storage space. In this example, the amount of storage required is only 5% of the original audio signal!

Image Compression Example [Optional]

THE FOURIER SERIES CAN BE GENERALIZED TO TWO-DIMENSIONAL SIGNALS $I : \mathbb{R}^2 \rightarrow \mathbb{C}$. The two-dimensional Fourier series is often used in image processing and image compression.

The Fourier series representation for a two-dimensional periodic function with period T_x in the x direction and T_y in the y direction is:

$$I(x, y) = \sum_{n,m \in \mathbb{Z}} c_{n,m} e^{i\frac{2\pi}{T_x}nx} e^{i\frac{2\pi}{T_y}my}. \quad (8.93)$$

The analysis procedure to obtain the coefficients $c_{n,m} \in \mathbb{C}$ is:

$$c_{n,m} = \frac{1}{T_x T_y} \int_0^{T_x} \int_0^{T_y} I(x, y) e^{-i\frac{2\pi}{T_x}nx} e^{-i\frac{2\pi}{T_y}my} dx dy. \quad (8.94)$$

Many of the familiar results for 1d signals, such as the time shifting property, or differentiation, can be extended to 2d or higher dimensional periodic functions.

IN THIS PROGRAMMING EXAMPLE I'LL BRIEFLY DEMONSTRATE TO YOU THE CONCEPT OF SPECTRAL IMAGE COMPRESSION. It is possible to extend the idea of compression of a one dimensional signal, as demonstrated in the audio compression example, by only storing strong frequency components in 2D images. This is essentially the idea behind the JPEG image compression algorithm. The Python code in Listing 8.7 demonstrates image compression and produces the image shown in Figure 8.12. The program uses the 2D discrete-Fourier transform function, `numpy.fft.fft2`, to calculate the phases and amplitudes of the spectral components, $c_{n,m}$, of the image. It then uses the 2D inverse discrete Fourier transform to FFT synthesize an image from only 5% of the spectral components. The output of the image compression example is shown in Figure 8.12. It shows the original and compressed image that is formed using a sparse Fourier series representation of the image. The magnitudes of the spectral components for the compressed and original image are also shown.

The same words of warning that I gave with the audio compression example apply to this program example. If you are only beginning with Python, you may have a hard time with understanding this program. For now, you can think of the 2D FFT and inverse FFT as an analysis and synthesis step for a 2D periodic function. But don't let a warning or lack of experience prevent you from exploring!

```
import imageio.v3 as si
import matplotlib.pyplot as plt
import numpy as np
```

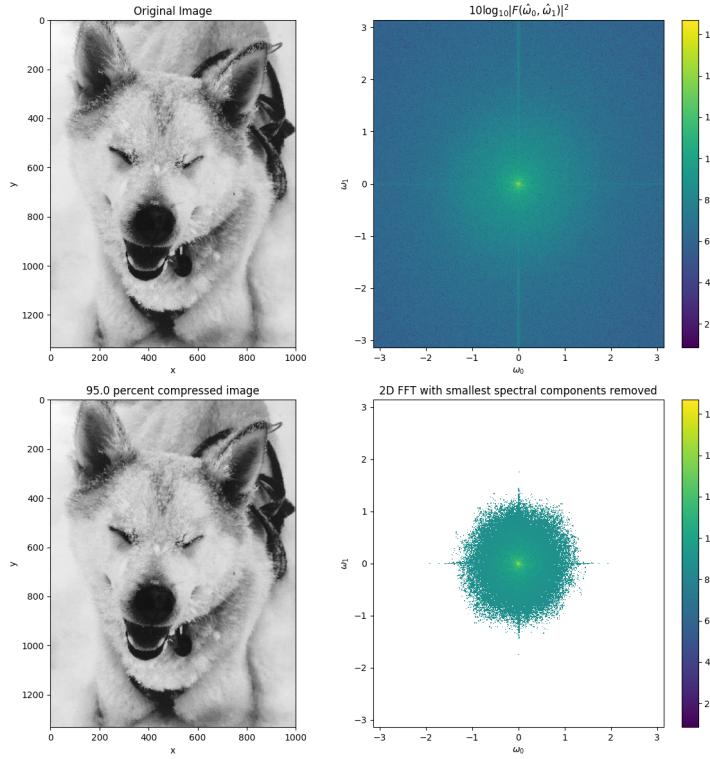


Figure 8.12: Image compression example. Top left: original image, Top right, the magnitude squared of the spectral components in dB scale. Bottom left: compressed image formed using only 5% of the strongest spectral components. Bottom right: the spectral magnitude squared of the spectral components used to form the image on the bottom left.

```

#
# 2d fft image, remove compression_ratio*(width*height)
# spectral components.
#
def compress_image(image, compression_ratio=0.95):
    # 2D DFT.
    image_fft = np.fft.fft2(image)
    L = image.shape[0]*image.shape[1]

    # Convert to 1d array,
    # sort spectral components by magnitude,
    # smallest spectral components first.
    image_fft = image_fft.flatten()
    idx = np.argsort(np.abs(image_fft).flatten())

    # Remove weakest spectral components, these don't
    # need to be stored, so this reduces data storage
    # requirement.
    # Ensure that at least two spectral components are left.
    max_idx = np.min([L-1, int(L*compression_ratio)])
    image_fft[idx[0:max_idx]] = 0.0
    image_fft.shape = image.shape

    # Inverse 2D DFT to get compressed image.
    # Take real component to ensure signal is real.
    comp_image = np.fft.ifft2(image_fft).real

    return ((comp_image, image_fft))

```

```

# Read image. Assume the image is perfect.
image = si.imread("husky.jpg", mode="F")

# How much do we compress the image. 0 is no compression,
# 1.0 is nearly 100% compression
# (we still have at least one spectral component.)
compression_ratio = 0.95

plt.figure(figsize=(12, 12))
# Show original image.
plt.subplot(221)
plt.imshow(image, cmap="gray", vmin=0, vmax=255)

plt.title("Original Image")

# Show 2D fourier transform of image.
image_fft = np.fft.fftshift(np.fft.fft2(image))
x_freq = np.linspace(-np.pi, np.pi, num=image_fft.shape[0])
y_freq = np.linspace(-np.pi, np.pi, num=image_fft.shape[1])
plt.subplot(222)
image_fft_DB = 10.0*np.log10(np.abs(image_fft)**2.0)
dB_min = np.min(image_fft_DB)
dB_max = np.max(image_fft_DB)
plt.pcolormesh(y_freq, x_freq, image_fft_DB)
plt.xlabel(r"\omega_0")
plt.ylabel(r"\omega_1")
plt.xlim([-np.pi, np.pi])
plt.ylim([-np.pi, np.pi])

plt.colorbar()
plt.title(r"$10 \log_{10} F(\hat{\omega}_0, \hat{\omega}_1)^2$")

# Compress image using 2D FFT.
plt.subplot(224)
comp_imag, comp_fft = compress_image(image, compression_ratio=
    compression_ratio)
plt.pcolormesh(y_freq, x_freq, 10.0*np.log10(np.abs(np.fft.
    fftshift(comp_fft) + 1e-9)**2.0), vmin=dB_min, vmax=dB_max)
plt.xlabel(r"\omega_0")
plt.ylabel(r"\omega_1")
plt.xlim([-np.pi, np.pi])
plt.ylim([-np.pi, np.pi])
plt.colorbar()
plt.title("2D FFT with smallest spectral components removed")

plt.subplot(223)
plt.imshow(comp_imag, cmap="gray", vmin=0, vmax=255)
plt.title("%1.1f percent compressed image" % (100.0*
    compression_ratio))
plt.tight_layout()
plt.savefig("image_compression.png")
plt.show()

```

Listing 8.7: 011_image_compression/image_compression.py

Variable star lightcurve fitting [Optional]

EXAMPLE: THE STUDY OF ASTRONOMICAL LIGHT CURVES OF VARIABLE STARS IS AN EXAMPLE OF A PRACTICAL APPLICATION OF THE FOURIER SERIES.

Variable stars, called Cepheids, have intensities $I(t)$ that vary periodically as a function of time. An example of a light curve is shown in Figure 8.13. Such variable stars pulsate in luminosity in a regularly repeating manner. The repeating luminosity waveform is often very non-sinusoidal, necessitating a Fourier series analysis.

Measurements of the intensity of a *Cepheid variable star* are made at periods of time when telescope time is available and astronomical seeing allows observations to be made. The time spacing between measurements is therefore by no means regular.

When analyzing the periodic waveform of this star, a search is made to determine the period T of the intensity waveform, and the Fourier series coefficients c_k . This is done by using an exhaustive search for plausible values of T . The model function is a Fourier series synthesis equation:

$$I_N(t) = \sum_{k=-N}^N c_k e^{i \frac{2\pi}{T} kt}. \quad (8.95)$$

The relationship between luminosity and the pulsation period is thought to follow a relatively well known relationship that can be linked to fundamental pulsation frequencies of rotating stars⁵. Longer period Cepheids are brighter, as show in Figure 8.14. This makes these types of variable stars useful for measuring the astronomical distances.

The Fourier series can be used to express periodic functions. One real life use case is estimating the functional form for a periodic star, which varies in brightness periodically. In this exercise, we will determine the Fourier series coefficients for brightness observations of a real Cepheid star. We will then plot the Fourier series for the periodic light curve of this star.

To estimate the periodic waveform for a Cepheid variable star brightness measurements $m(t)$, we will use something called the maximum likelihood method, which estimates the Fourier series coefficients. This involves expressing the Fourier series as a matrix

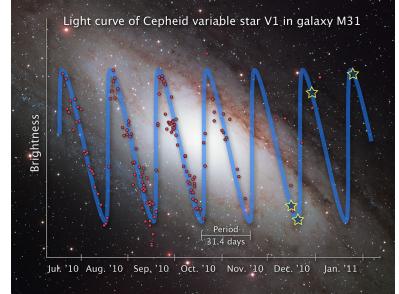


Figure 8.13: Measurements of the intensity of a variable star (red) and a Fourier series model of the intensity of the star as a function of time (blue). Illustration Credit: NASA, ESA and Z. Levay (STScI). Science Credit: NASA, ESA, the Hubble Heritage Team (STScI/AURA) and the American Association of Variable Star Observers.

⁵ RB Hindsley and RA Bell. The period-luminosity relation for cepheid variable stars. *Astrophysical Journal, Part 1* (ISSN 0004-637X), vol. 341, June 15, 1989, p. 1004–1019., 341:1004–1019, 1989

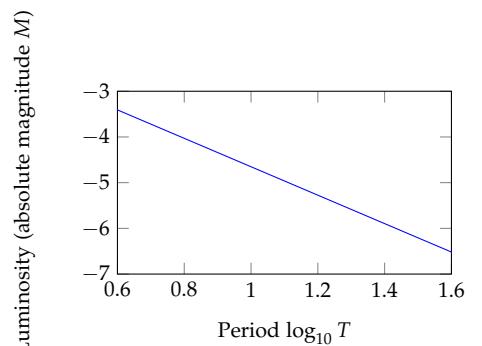


Figure 8.14: Relationship between Cepheid period and absolute magnitude, based on relationship published by Hindsley and Bell (1989).

vector operation:

$$m = Ax \quad (8.96)$$

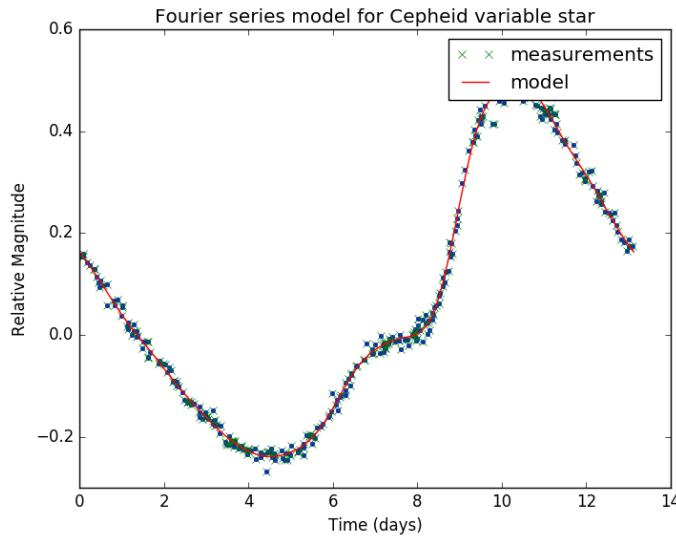
$$\begin{bmatrix} m(t_0) \\ m(t_1) \\ m(t_2) \\ \vdots \\ m(t_M) \end{bmatrix} = \begin{bmatrix} e^{i\frac{2\pi}{T}k_0t_0} & e^{i\frac{2\pi}{T}k_1t_0} & \dots & e^{i\frac{2\pi}{T}k_Nt_0} \\ e^{i\frac{2\pi}{T}k_0t_1} & e^{i\frac{2\pi}{T}k_1t_1} & \dots & e^{i\frac{2\pi}{T}k_Nt_1} \\ \vdots & \vdots & \ddots & \vdots \\ e^{i\frac{2\pi}{T}k_0t_M} & e^{i\frac{2\pi}{T}k_1t_M} & \dots & e^{i\frac{2\pi}{T}k_Nt_M} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_N \end{bmatrix}. \quad (8.97)$$

Here c_k is a Fourier series coefficient and T is the fundamental period. Because there are discrete measurements, the measurement function is only known at discrete points $m(t_\ell)$. In order to estimate the Fourier series coefficients, we use the linear least-squares estimator:

$$\hat{x} = (A^H A)^{-1} A^H m. \quad (8.98)$$

The vector \hat{x} will now contain the most probable Fourier series coefficients that explain the measurements $m(t_\ell)$ of the periodic function.

Because this course does not deal with statistics, we have implemented the code that figures out the Fourier series coefficients that fit the measurements. Your task is to edit the code below, and to evaluate the Fourier series model, given the coefficients c_k , which are in the array variable named `c_k`. Code is shown in Listing 8.8, which already does most of the work. Complete the last for-loop, and you should get the following plot:



```
import matplotlib.pyplot as plt
import numpy as np
```

```

# Load data, download data from:
# http://kaira.uit.no/fys2006/lcb1.dat
d = np.loadtxt("lcb1.dat")

# Measurement times.
m_error = d[:, 2]

# Select find measurements that don't have large errors.
good_idx = np.where(np.abs(m_error) < 1.0)[0]

# Measurement time (units of days).
m_t = d[good_idx, 0]

# Magnitude measurements (relative magnitude of star).
m_mag = d[good_idx, 1]

# Fundamental period.
T = 13.124349

# Time, modulo period.
m_modulo_t = np.mod(m_t, T)

plt.plot(m_modulo_t, m_mag, ".")
```

Number of Fourier series coefficients.

N = 10

N_meas = len(m_mag)

Frequency indices.

k_idx = np.arange(-N, N+1)

Theory matrix.

A = np.zeros([N_meas, len(k_idx)], dtype=np.complex64)

for ki, k in enumerate(k_idx):
 # Setup theory matrix row.
 A[:, ki] = np.exp(1j*(2.0*np.pi/T)*k*m_modulo_t)

Find maximum likelihood estimate for Fourier Series
coefficients c_k.

S = np.linalg.inv(np.dot(np.transpose(np.conj(A)), A))
c_k = np.dot(np.dot(S, np.transpose(np.conj(A))), m_mag)
print(c_k.shape)

Evaluate the Fourier Series for the maximum likelihood
estimate of
coefficients c_k.

N_model = 100
model_t = np.linspace(0, T, num=N_model)
model = np.zeros(N_model, dtype=np.complex64)

Sum together signal for all Fourier series coefficients a_k.

for ki, k in enumerate(k_idx):
 # Figure out what to put here.
 # array "model_t" contains time
 # array "c_k" contains the Fourier series coefficients.
 # We want the array "model" to contain the Fourier Series
 # vector
 model += # ... Figure out what to put here.

plt.plot(model_t, model.real, label="Model")

```
| plt.plot(m_modulo_t, m_mag, "x", label="Measurements")
| plt.title("Fourier series model for Cepheid variable star")
| plt.xlabel("Time (days)")
| plt.ylabel("Relative Magnitude")
| plt.legend()
| plt.show()
```

Listing 8.8: 012_variable_star/variable_star.py

9

Programming Assignment 1

Perform the following tasks. Write a report describing your results, which is **at most two pages long**. The report is free form, but it must be delivered in PDF format. Include your code and plots in the report. The report should be delivered by the deadline to Canvas (9th of September 12:00) if you want to receive feedback for your work. **Please note that you will need to submit the final version of your report at the end of course course on 11.11. as part of the portfolio to Wiseflow for final grading, so make sure that you store a copy of your report.**

The Dirac comb is a periodic signal, which is defined as follows:

$$x(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT) \quad (9.1)$$

The signal is shown in Figure 9.1.

- a) Implement a program that calculates a partial sum approximation

$$x_N(t) = \sum_{k=-N}^{N} c_k e^{j \frac{2\pi}{T} kt} \quad (9.2)$$

of a Dirac comb with a fundamental period of $T = 0.55$ seconds. Use $N = 50$ in Equation 9.2 to define the number of complex exponential signals to include in the sum. Evaluate the signal from $t = 0$ to $t = 4$ seconds at 10 kHz sample rate.

Here's some partial code, which almost does the job, but has several things wrong.

```
import matplotlib.pyplot as plt
import numpy as np
# Define the sample rate (Hz).
sample_rate = 10000.0
# Create time array 0 to 1 seconds.
t = np.arange(1.0 * sample_rate) / sample_rate
# Initialize empty vector to hold Fourier series.
sig = np.zeros(len(t), dtype=np.complex64)
```

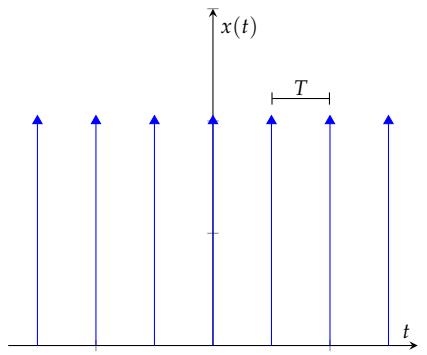


Figure 9.1: The Dirac comb signal with period T .

```
N = 10
# Add together complex sinusoids multiplied with Fourier
# series coefficients.
for k in range(-N, N):
    print(k)
    sig += ... # Complete this line.

plt.plot(t, sig.real)
plt.plot(t, sig.imag)
plt.xlabel("Time (s)")
plt.show()
```

If implemented correctly, the Fourier series should be real-valued. The imaginary part should only deviate from zero by only a very small number due to numerical errors in finite precision accuracy of floating point numbers.

- b) Figure out what modification you need to make to the Fourier series coefficients c_k in order to delay the signal by 0.2 seconds.
- c) Plot and verify that the coefficients obtained in b) produce the correct delay.

10

Fourier Transform

This chapter discusses the continuous-time Fourier transform. We'll cover the following topics:

- Important Fourier transform pairs
- Plancherel's theorem and Parseval's theorem
- Convolution theorem
- Impulse response and frequency response of LTI systems
- Fourier transform of a time shifted signal

The *forward Fourier transform* is defined as:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t} dt \quad (10.1)$$

and the *inverse Fourier transform* is defined as:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega)e^{i\omega t} d\omega . \quad (10.2)$$

We'll use the following notation to indicate Fourier transform pairs:

$$x(t) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) . \quad (10.3)$$

The left denotes the *time domain* representation of the signal and the right-hand side denotes the *frequency domain* representation of the same signal. I've used a hat to signify that the symbol $\hat{x}(\omega)$ is a frequency domain representation of a signal.

The existence or non-existence of a Fourier transform or an inverse Fourier transform is rarely something that is encountered in signal processing applications. Exploring this topic rigorously is outside the scope of this course.

Selected Fourier transforms

It is impossible to exhaustively cover all the possible Fourier transform pairs here. We'll only give examples of commonly encountered Fourier transforms and provide solution strategies for using these as basic building blocks to evaluate the Fourier transform for more complicated signals. Refer to a formula book^{1,2} for a more complete table of Fourier transform pairs.

Linear combination

The Fourier transform of a linear combination of signals in time domain is related to a linear combination of Fourier transforms in frequency domain:

$$y(t) = c_1 x_1(t) + c_2 x_2(t) \xleftrightarrow{\mathcal{F}} \hat{y}(\omega) = c_1 \hat{x}_1(\omega) + c_2 \hat{x}_2(\omega) . \quad (10.4)$$

This can be shown as follows:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} [c_1 x_1(t) + c_2 x_2(t)] e^{-i\omega t} dt \quad (10.5)$$

$$= c_1 \int_{-\infty}^{\infty} x_1(t) e^{-i\omega t} dt + c_2 \int_{-\infty}^{\infty} x_2(t) e^{-i\omega t} dt \quad (10.6)$$

$$= c_1 \hat{x}_1(\omega) + c_2 \hat{x}_2(\omega) \quad \square . \quad (10.7)$$

This may seem trivial, but keep this property in mind, as it can be used to Fourier transform more complicated signals that are a superposition of simple individual signals.

Fourier transform of $\delta(t + \tau)$

The time shifted unit impulse signal is a complex sinusoidal signal in frequency domain:

$$x(t) = \delta(t + \tau) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = e^{i\omega\tau} . \quad (10.8)$$

This is also relatively easy to show:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \quad (10.9)$$

$$= \int_{-\infty}^{\infty} \delta(t + \tau) e^{-i\omega t} dt \quad (10.10)$$

$$= e^{i\omega\tau} \quad \square . \quad (10.11)$$

Fourier transform of $e^{i\omega_0 t}$

The converse of the previous is a complex sinusoidal signal $e^{i\omega_0 t}$ with frequency ω_0 . This has the following Fourier transform pair:

$$x(t) = e^{i\omega_0 t} \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = 2\pi\delta(\omega - \omega_0) . \quad (10.12)$$

¹ The wikipedia article also contains a nice table of Fourier transform pairs:https://en.wikipedia.org/wiki/Fourier_transform.

² David W Kammler. *A first course in Fourier analysis*. Cambridge University Press, 2007

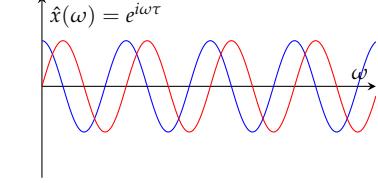
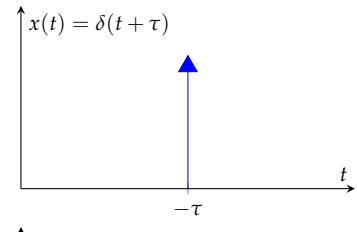


Figure 10.1: Unit impulse signal $\delta(t + \tau)$ is non-zero only when $t = -\tau$. In frequency domain, the Fourier transform is a complex sinusoidal signal with ω the independent variable.

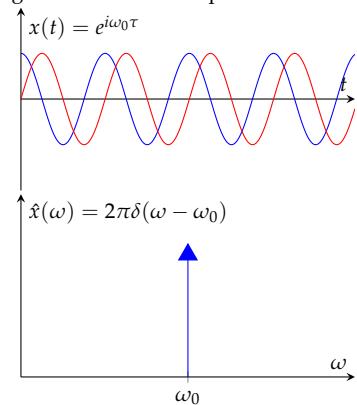


Figure 10.2: A complex sinusoidal signal of frequency ω_0 is a unit impulse signal $\delta(\omega - \omega_0)$ in frequency domain.

We show this by inverse Fourier transforming $\hat{x}(\omega)$:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega \quad (10.13)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} 2\pi\delta(\omega - \omega_0) e^{i\omega t} d\omega \quad (10.14)$$

$$= e^{i\omega_0 t} \quad \square. \quad (10.15)$$

Rectangular function

One often encountered Fourier transform pair is that of the rectangular function³ of length T :

³ Also often called the boxcar function.

$$\boxed{x(t) = u\left(t + \frac{T}{2}\right) - u\left(t - \frac{T}{2}\right) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = \frac{1}{i\omega} (e^{i\omega\frac{T}{2}} - e^{-i\omega\frac{T}{2}}).} \quad (10.16)$$

This can be derived as follows:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} \left[u\left(t + \frac{T}{2}\right) - u\left(t - \frac{T}{2}\right) \right] e^{-i\omega t} dt \quad (10.17)$$

$$= \int_{-T/2}^{T/2} e^{-i\omega t} dt \quad (10.18)$$

$$= -\frac{e^{-i\omega t}}{i\omega} \Big|_{t=-T/2}^{T/2} \quad (10.19)$$

$$= \frac{e^{i\omega\frac{T}{2}} - e^{-i\omega\frac{T}{2}}}{i\omega}. \quad (10.20)$$

We can use Euler $\sin(x) = \frac{1}{2i}(e^{ix} - e^{-ix})$ to express this as follows:

$$\boxed{\hat{x}(\omega) = \frac{2 \sin\left(\omega\frac{T}{2}\right)}{\omega}.} \quad (10.21)$$

The function $\text{sinc}(\omega) = \sin(\pi\omega)/(\pi\omega)$ is often used to express this function. This is convenient, as numerical implementations (such as `numpy.sinc`) of the function deal with the special case when $\omega = 0$, which can be investigated using L'Hôpital's rule:

$$\lim_{\omega \rightarrow c} \frac{f(\omega)}{g(\omega)} = \lim_{\omega \rightarrow c} \frac{f'(\omega)}{g'(\omega)}. \quad (10.22)$$

Which in this case is:

$$\lim_{\omega \rightarrow 0} \frac{2 \sin(\omega T/2)}{\omega} = \lim_{\omega \rightarrow 0} T \cos(\omega T/2) = T. \quad (10.23)$$

We'll be using L'Hôpital's rule elsewhere in signal processing to resolve similar situations.

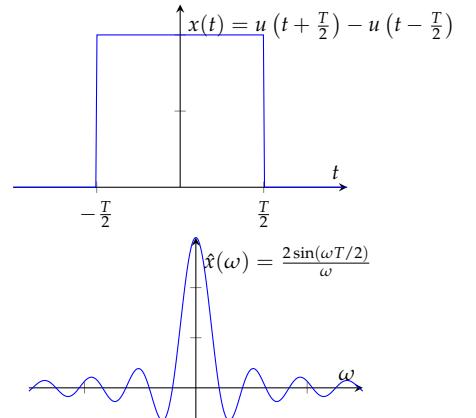


Figure 10.3: The Fourier transform of a boxcar function.

Sinc function

A related Fourier transform pair is the one where the frequency domain representation is a rectangular function. This is used, e.g., when calculating ideal filters. This result is also used in Shannon's sampling theorem to obtain an ideal reconstruction filter for band limited signals that only have spectral components in the band $|\omega| < \omega_b$.

$$\boxed{x(t) = \frac{\sin(\omega_b t)}{\pi t} \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = u(\omega + \omega_b) - u(\omega - \omega_b)} \quad (10.24)$$

This relationship can be shown by inverse Fourier transforming $\hat{x}(\omega)$:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} [u(\omega + \omega_b) - u(\omega - \omega_b)] e^{i\omega t} d\omega \quad (10.25)$$

$$= \frac{1}{2\pi} \int_{-\omega_b}^{\omega_b} e^{i\omega t} d\omega \quad (10.26)$$

$$= \frac{1}{2\pi} \left. \frac{e^{i\omega t}}{it} \right|_{\omega=-\omega_b}^{\omega_b} \quad (10.27)$$

$$= \frac{e^{it\omega_b} - e^{-it\omega_b}}{2\pi it} \quad (10.28)$$

$$= \frac{\sin(\omega_b t)}{\pi t} \quad \square. \quad (10.29)$$

Fourier transforming the sinc-function is not as trivial.

Gaussian

The Fourier transform of a Gaussian density function is another Gaussian density function:

$$\boxed{x(t) = e^{-\alpha t^2} \xleftrightarrow{\mathcal{F}} \sqrt{\frac{\pi}{\alpha}} e^{-\frac{\omega^2}{4\alpha}}}. \quad (10.30)$$

Note that the width of the time domain Gaussian is inversely proportional to the width of the frequency domain Gaussian.

If we rewrite the above into a form which resembles a Gaussian density function normalized to unity at zero, with a width parameter σ_t and σ_ω for the time domain and frequency domain width, we get:

$$x(t) \propto e^{-\frac{t^2}{2\sigma_t^2}} \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) \propto e^{-\frac{\omega^2}{2\sigma_\omega^2}}, \quad (10.31)$$

which implies that:

$$\frac{1}{\sigma_t} = \sigma_\omega. \quad (10.32)$$

This is a very nice demonstration of the inverse relationship between the width of a signal in time domain and frequency domain. When

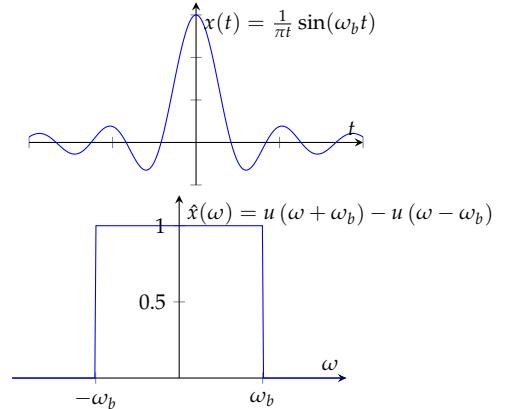


Figure 10.4: A sinc function in time domain is a boxcar function in frequency domain.

we discuss filters later on, we'll use this to show that the length of a filter in time domain is inversely proportional to the spectral resolution (width of the filter response).

This Fourier transform pair can be derived as follows:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} e^{-\alpha t^2} e^{-i\omega t} dt \quad (10.33)$$

$$= \int_{-\infty}^{\infty} e^{-\alpha t^2 - i\omega t} dt. \quad (10.34)$$

We perform a little trick and multiply with $1 = e^{c^2\omega^2}e^{-c^2\omega^2}$ using a procedure called completing the square, so that we can get to a variable substitution that allows us to easily integrate the function inside the integral:

$$\hat{x}(\omega) = e^{c^2\omega^2} \int_{-\infty}^{\infty} e^{-\alpha t^2 - i\omega t - c^2\omega^2} dt. \quad (10.35)$$

We now substitute a variable $\alpha' = \sqrt{\alpha}$, next we then need to find a constant c so that:

$$-t^2\alpha'^2 - i\omega t - c^2\omega^2 = -(t\alpha' + c\omega)^2. \quad (10.36)$$

Furthermore, we find that $c = \frac{i}{2\alpha'}$.

$$\hat{x}(\omega) = e^{-\frac{\omega^2}{4\alpha}} \int_{-\infty}^{\infty} e^{-(t\alpha' + \frac{i}{2\alpha'}\omega)^2} dt. \quad (10.37)$$

Now we perform a variable substitution: $u = t\alpha' + \frac{i}{2\alpha'}\omega$, which gives us: $\frac{1}{\alpha'}du = dt$.

We now need to refer to a table of integrals to obtain a solution to the integral:

$$\int_{-\infty}^{\infty} e^{-u^2} du = \sqrt{\pi} \quad (10.38)$$

and thus

$$\hat{x}(\omega) = \frac{1}{\sqrt{\alpha}} e^{-\frac{\omega^2}{4\alpha}} \int_{-\infty}^{\infty} e^{-u^2} du \quad (10.39)$$

$$= \sqrt{\frac{\pi}{\alpha}} e^{-\frac{\omega^2}{4\alpha}}. \quad (10.40)$$

The plots in Figure 10.5 show the time domain and frequency domain Fourier transform pairs for two different values of α , which demonstrates the inverse relationship between the width of the function in time and frequency domain.

Exponentially decaying signal $x(t) = e^{-\beta t}u(t)$

When $\beta \in \mathbb{R}_{>0}$ the signal $e^{-\beta t}u(t)$ decays exponentially. This type of signal is often encountered in electrical circuits (e.g., a low pass filter)

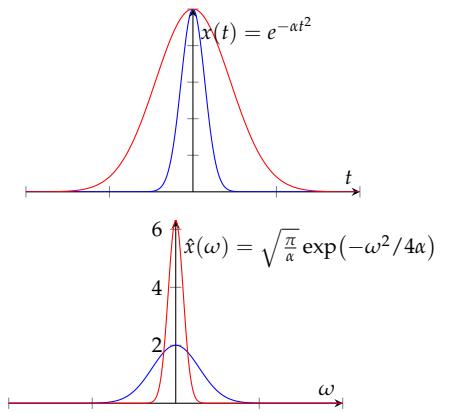


Figure 10.5: The Fourier transform of a Gaussian density function is another Gaussian density function. The width of the frequency domain and time domain density functions are inversely proportional.

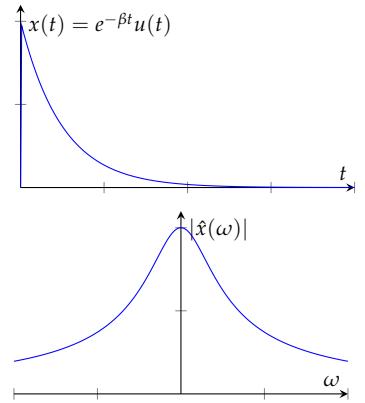


Figure 10.6: The Fourier transform of an exponentially decaying signal.

and in control systems:

$$x(t) = e^{-\beta t} u(t) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = \frac{1}{\beta + i\omega} . \quad (10.41)$$

This can be derived as follows:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} e^{-\beta t} u(t) e^{-i\omega t} dt \quad (10.42)$$

$$= \int_0^{\infty} e^{-(\beta+i\omega)t} dt \quad (10.43)$$

$$= -\frac{1}{\beta+i\omega} e^{-(\beta+i\omega)t} \Big|_{t=0}^{\infty} \quad (10.44)$$

$$= \frac{1}{\beta+i\omega} . \quad (10.45)$$

The above requires that $\beta \in \mathbb{R}_{>0}$.

Real-valued signals

There is a special conjugate symmetry for signals that are real-valued either in time domain or frequency domain.

If the time domain signal is real-valued, then the frequency domain representation is conjugate symmetric.

$$x(t) \in \mathbb{R} \xleftrightarrow{\mathcal{F}} \hat{x}(-\omega) = \hat{x}^*(\omega) . \quad (10.46)$$

The same also applies the other way around. If the spectral representation is real-valued, then the time domain representation is conjugate symmetric:

$$x(-t) = x^*(t) \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) \in \mathbb{R} . \quad (10.47)$$

This can be shown as follows:

Proof.

$$\hat{x}^*(\omega) = \left(\int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \right)^* \quad (10.48)$$

$$= \int_{-\infty}^{\infty} x^*(t) e^{i\omega t} dt \quad (10.49)$$

$$= \int_{-\infty}^{\infty} x(t) e^{i\omega t} dt \quad (10.50)$$

$$= \hat{x}(-\omega) . \quad (10.51)$$

□

Because $x(t) \in \mathbb{R}$, we have that $x^*(t) = x(t)$. A similar proof exists for the real-valued spectral representation.

Fourier series

A Fourier series can be used to represent a periodic function as a sum of frequency components. Because we know that the Fourier transform of $e^{i\omega't}$ is $2\pi\delta(\omega - \omega')$, we can easily determine the Fourier transform of an arbitrary periodic function by using its Fourier series:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{ik\omega_0 t} \xrightarrow{\mathcal{F}} \hat{x}(\omega) = 2\pi \sum_{k=-\infty}^{\infty} c_k \delta(\omega - k\omega_0). \quad (10.52)$$

The derivation is as follows:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_k e^{ik\omega_0 t} e^{-i\omega t} dt \quad (10.53)$$

$$= \sum_{k=-\infty}^{\infty} c_k \int_{-\infty}^{\infty} e^{ik\omega_0 t} e^{-i\omega t} dt \quad (10.54)$$

$$= 2\pi \sum_{k=-\infty}^{\infty} c_k \delta(\omega - k\omega_0). \quad (10.55)$$

This is simply the continuous-frequency representation of Fourier series spectral components. A periodic function has non-zero spectral components only at frequencies that are multiples of ω_0 .

The inverse Fourier transform of the spectrum $\hat{x}(\omega)$ is:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega \quad (10.56)$$

$$= \sum_{k=-\infty}^{\infty} c_k \int_{-\infty}^{\infty} \delta(\omega - k\omega_0) e^{i\omega t} d\omega \quad (10.57)$$

$$= \sum_{k=-\infty}^{\infty} c_k e^{ik\omega_0 t}. \quad (10.58)$$

The fact that the continuous-time Fourier transform and the Fourier series are related in this way shouldn't be too surprising, as we earlier derived the Fourier transform from the Fourier series representation for a periodic function with an infinitely long fundamental period.

Time scaling system

Let's look at a time scaling system:

$$y(t) = x(at). \quad (10.59)$$

What effect does this system have on the frequency domain representation of this signal?

We assume that a signal $x(t)$ has a Fourier transform:

$$x(t) \xrightarrow{\mathcal{F}} \hat{x}(\omega). \quad (10.60)$$

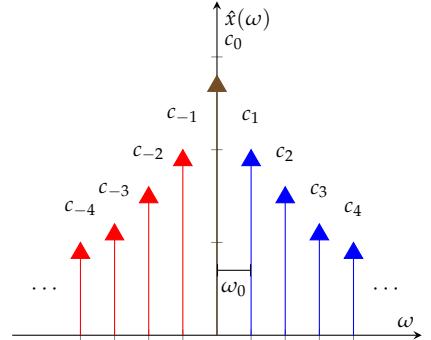


Figure 10.7: The Fourier transform of a Fourier series. The spectral representation is a set of spectral lines spaced apart by ω_0 , the fundamental angular frequency of the periodic signal. A set of spectral lines spaced apart by a constant value is a spectral signature of a time-periodic signal.

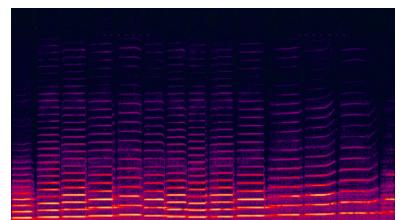


Figure 10.8: A time-frequency-power image of a violin being played. The x-axis represents time and the y-axis represents frequency. Only positive frequencies are shown. At any given moment, a violin is playing approximately a time-periodic signal (a single note). The spectrum is a forest of spectral lines spaced apart by the fundamental angular frequency of the signal. Credits: Wikipedia.

If we scale the time parameter $x(at)$, then the spectral representation is scaled inversely in frequency:

$$\boxed{y(t) = x(at) \xleftrightarrow{\mathcal{F}} \hat{y}(\omega) = \frac{1}{|a|} \hat{x}\left(\frac{\omega}{a}\right)} . \quad (10.61)$$

What does scaling in time mean?

- Compressing a signal in time with $|a| > 1$ will stretch the signal in frequency.
- Stretching a signal in time with $|a| < 1$ will compress the signal in frequency.

Proof. The proof is as follows. Let's investigate the case where $a > 0$ and Fourier transform $y(t)$:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} y(t) e^{-i\omega t} dt \quad (10.62)$$

$$= \int_{-\infty}^{\infty} x(at) e^{-i\omega t} dt . \quad (10.63)$$

We do a variable substitution $t' = at$, from which it follows that

$dt = \frac{1}{a} dt'$ and

$$\hat{y}(\omega) = \frac{1}{a} \int_{-\infty}^{\infty} x(t') e^{-i\frac{\omega}{a} t'} dt' \quad (10.64)$$

$$= \frac{1}{a} \hat{x}\left(\frac{\omega}{a}\right) . \quad (10.65)$$

□

Plancherel's theorem

Plancherel's theorem states that for any two signals $f(t)$ and $g(t)$ with Fourier transforms $\hat{F}(\omega)$ and $\hat{G}(\omega)$ the following integral is satisfied:

$$\boxed{\int_{-\infty}^{\infty} f(t) g^*(t) dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{F}(\omega) \hat{G}^*(\omega) d\omega} . \quad (10.66)$$

This is useful, as we can relate the amount of signal power in time domain to the total amount of power in frequency domain. This theorem has many applications in, e.g., the theory of random processes, radio astronomical signal processing, and radar signal processing.

Proof. The proof is as follows. Consider first the following signal, which is represented using the Dirac delta function in frequency domain:

$$\hat{x}(\omega) = \delta(\omega - \omega') . \quad (10.67)$$

This has a time domain-representation, which is obtained using the inverse Fourier transform:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \delta(\omega - \omega') e^{i\omega t} d\omega \quad (10.68)$$

$$= \frac{1}{2\pi} e^{i\omega' t}. \quad (10.69)$$

Let us now consider the Fourier transform representations for these two signals:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \quad (10.70)$$

$$g(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} G(\omega') e^{i\omega' t} d\omega'. \quad (10.71)$$

If we multiply the Fourier representations of these two and integrate over time:

$$\int_{-\infty}^{\infty} f(t) g^*(t) dt = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega \right] \left[\int_{-\infty}^{\infty} G^*(\omega') e^{-i\omega' t} d\omega' \right] dt \quad (10.72)$$

$$= \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} F(\omega) \int_{-\infty}^{\infty} G^*(\omega') \int_{-\infty}^{\infty} e^{i(\omega-\omega')t} dt d\omega' d\omega \quad (10.73)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) \int_{-\infty}^{\infty} G^*(\omega') \delta(\omega - \omega') d\omega' d\omega \quad (10.74)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) G^*(\omega) d\omega. \quad (10.75)$$

□

The key to this proof is that:

$$\int_{-\infty}^{\infty} e^{i(\omega-\omega')t} dt = \int_{-\infty}^{\infty} e^{-i\omega' t} e^{i\omega t} dt = 2\pi\delta(\omega - \omega'). \quad (10.76)$$

PARSEVAL'S THEOREM is simply a corollary of Plancherel's theorem. If we state that $f(t) = g(t) = x(t)$, then Plancherel's theorem becomes:

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\hat{x}(\omega)|^2 d\omega. \quad (10.77)$$

In physics this means that the total energy of the signal in time domain and frequency domain is equivalent, up to a predefined fixed scaling constant.

Convolution theorem

The convolution theorem states that a convolution operation in time domain is a multiplication operation in frequency domain. A

convolution sum of signals $x_1(t)$ and $x_2(t)$ is defined as:

$$x_1(t) * x_2(t) := \int_{-\infty}^{\infty} x_1(\tau)x_2(t - \tau)d\tau . \quad (10.78)$$

The star symbol $(*)$ is often used in shorthand to express this operation.

An often used property in signal processing is that convolution in time domain is multiplication in frequency domain.

$$y(t) = x_1(t) * x_2(t) \xrightarrow{\mathcal{F}} \hat{y}(\omega) = \hat{x}_1(\omega)\hat{x}_2(\omega) . \quad (10.79)$$

Proof.

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} y(t)e^{-i\omega t}dt \quad (10.80)$$

$$= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{\infty} x_1(\tau)x_2(t - \tau)d\tau \right) e^{-i\omega t}dt \quad (10.81)$$

$$= \int_{-\infty}^{\infty} x_1(\tau) \left(\int_{-\infty}^{\infty} x_2(t - \tau)e^{-i\omega t}dt \right) d\tau . \quad (10.82)$$

Next, we perform substitution of variables $m = t - \tau$. From this, it follows that $t = m + \tau$ and $dm = dt$. After substitution, we get:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} x_1(\tau) \left(\int_{-\infty}^{\infty} x_2(m)e^{-i\omega m}dm \right) e^{-i\omega\tau}d\tau \quad (10.83)$$

$$= \hat{x}_2(\omega) \int_{-\infty}^{\infty} x_1(\tau)e^{-i\omega\tau}d\tau \quad (10.84)$$

$$= \hat{x}_1(\omega)\hat{x}_2(\omega) . \quad (10.85)$$

□

THE CONVERSE OF THE PREVIOUS IS THAT MULTIPLICATION IN TIME DOMAIN CORRESPONDS TO CONVOLUTION IN FREQUENCY DOMAIN:

$$y(t) = x_1(t)x_2(t) \xrightarrow{\mathcal{F}} \hat{y}(\omega) = \frac{1}{2\pi}\hat{x}_1(\omega) * \hat{x}_2(\omega) . \quad (10.86)$$

Proof. We start by Fourier transforming $x_1(t)x_2(t)$:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} x_1(t)x_2(t)e^{-i\omega t}dt . \quad (10.87)$$

We then substitute $x_2(t)$ with its Fourier transform:

$$x_2(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_2(\omega')e^{i\omega't}d\omega' , \quad (10.88)$$

which gives us:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} x_1(t) \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_2(\omega')e^{i\omega't}d\omega' \right) e^{-i\omega t}dt . \quad (10.89)$$

We then swap the order of integration and combine all terms with t inside the innermost integral:

$$\hat{y}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_2(\omega') \left(\int_{-\infty}^{\infty} x_1(t) e^{i\omega' t} e^{-i\omega t} dt \right) d\omega' \quad (10.90)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_2(\omega') \left(\int_{-\infty}^{\infty} x_1(t) e^{-i(\omega-\omega')t} dt \right) d\omega' \quad (10.91)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_2(\omega') \hat{x}_1(\omega - \omega') d\omega' \quad (10.92)$$

$$= \frac{1}{2\pi} \hat{x}_1(\omega) * \hat{x}_2(\omega) . \quad (10.93)$$

□

This property is also very useful. We will use it next to inspect the spectrum of an amplitude modulated signal. This property will also be used when deriving Shannon's sampling theorem.

Time shifted signal

Assuming that the Fourier transform of $x(t)$ is $\hat{x}(\omega)$, the Fourier transform of a time shifted signal can be obtained using the following formula:

$$y(t) = x(t - t_0) \xrightarrow{\mathcal{F}} \hat{y}(\omega) = e^{-i\omega t_0} \hat{x}(\omega) . \quad (10.94)$$

Proof. Left as an exercise. □

Alternative definitions and notation

There are several notations and slightly different definitions of a Fourier transform that one may encounter in the literature. These mainly vary in how the frequency parameter of the spectral representation is scaled and what symbols are used to denote frequency and the imaginary number. The underlying mathematical concepts are the same.

The spectral representation is sometimes written with an explicit function parameter $i\omega$ instead of ω . Sometimes the hat-notation is not used:

$$\hat{x}(\omega) = \hat{x}(i\omega) = x(\omega) . \quad (10.95)$$

Commonly used symbols for frequency include ω , ν , ξ , λ , k , l , and f . Also, as usual, the imaginary number $i = \sqrt{-1}$ is sometimes denoted with j .

The most common alternative form is the one where the frequency parameter is given scaled to units of cycles per unit of t :

$$f = \frac{\omega}{2\pi} . \quad (10.96)$$

If t is in units of time in seconds, then the units of f would be $1/s$ or hertz. With this frequency parameterization, the transform is unitary (it is norm preserving):

$$\hat{x}_1(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft}dt \quad (10.97)$$

$$x(t) = \int_{-\infty}^{\infty} \hat{x}_1(f)e^{i2\pi ft}df. \quad (10.98)$$

Another commonly used definition for the Fourier transform is one where the forward and reverse transform are scaled symmetrically:

$$\hat{x}_2(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt \quad (10.99)$$

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{x}_2(\omega)e^{i\omega t}d\omega. \quad (10.100)$$

Sometimes, the sign convention for the forward and reverse transform is changed:

$$\hat{x}_3(\omega) = \int_{-\infty}^{\infty} x(t)e^{i\omega t}dt \quad (10.101)$$

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_3(\omega)e^{-i\omega t}d\omega. \quad (10.102)$$

All of these alternative Fourier transform definitions have the same general properties, but the exact equations can differ in scaling constants or in sign of the frequencies. For example, if we were to define the Fourier transform as in equation 10.99, the Fourier transform of $x(t) = \delta(t + \tau)$ would be $\hat{x}(\omega) = \frac{1}{\sqrt{2\pi}}e^{i\omega\tau}$, which is slightly different from the result we obtained in Equation 10.8.

Exercises: Fourier Transform

1. Find the Fourier transforms of these signals:

a) $x(t) = \delta(t+2) + \delta(t) + \delta(t-2)$

Hint: linearity and time shift properties.

b) $x(t) = \frac{\sin(100\pi|t-2|)}{\pi(t-2)}$

Hint: time shift property.

c) $x(t) = e^{-t}u(t) - e^{-t}u(t-4)$

Hint: look at the derivation of the Fourier transform of $e^{-\beta t}u(t)$.

Note that there are more than one way to solve the Fourier transforms!

2. Find the inverse Fourier transform for the following:

a) $\hat{x}(\omega) = 2 + \cos(\omega)$

b) $\hat{x}(\omega) = \frac{1}{1+i\omega} - \frac{1}{2+i\omega}$

c) $\hat{x}(\omega) = i\delta(\omega - 100\pi) - i\delta(\omega + 100\pi)$.

3. Use known Fourier transform pairs together with the properties of the Fourier transform to complete the following Fourier transform pairs:

a) $x(t) = u(t+3)u(3-t) \xleftrightarrow{\mathcal{F}} \hat{x} = ?$

b) $x(t) = \sin(4\pi t) \sin(50\pi t) \xleftrightarrow{\mathcal{F}} \hat{x} = ?$

c) $x(t) = \frac{\sin(4\pi t)}{\pi t} \sin(50\pi t) \xleftrightarrow{\mathcal{F}} \hat{x} = ?$

d) $x(t) = ? \xleftrightarrow{\mathcal{F}} \hat{x} = \cos^2(\omega)$

4. Let $\hat{x}(\omega)$ be the Fourier transform of $x(t)$. Prove that if $y(t) = x(t-\tau)$, then $\hat{y}(\omega) = e^{-i\omega\tau}\hat{x}(\omega)$.

Suggested solutions: Fourier Transform

1. a) If

$$x(t) = \delta(t+2) + \delta(t) + \delta(t-2),$$

then by linearity we have that:

$$\hat{x}(\omega) = \mathcal{F}\{x(t)\} = \mathcal{F}\{\delta(t+2)\} + \mathcal{F}\{\delta(t)\} + \mathcal{F}\{\delta(t-2)\},$$

now, by using time shift properties we get:

$$\hat{x}(\omega) = \mathcal{F}\{x(t)\} = \underline{\underline{e^{2i\omega}}} + 1 + \underline{\underline{e^{-2i\omega}}},$$

by Equation 10.8.

b) Can rewrite the signal using the sinc function as:

$$x(t) = \frac{\sin(100\pi(t-2))}{\pi(t-2)} = 100\text{sinc}(100(t-2)).$$

Then by Equation 10.94 and Equation 10.24 the Fourier transform is:

$$\hat{x}(\omega) = 100e^{-2i\omega} \mathcal{F}\{\text{sinc}(100t)\} = 100e^{-2i\omega} \frac{1}{100} [u(\omega + 100\pi) - u(\omega - 100\pi)],$$

giving:

$$\hat{x}(\omega) = \underline{\underline{e^{-2i\omega}[u(\omega + 100\pi) - u(\omega - 100\pi)]}}.$$

c) Take:

$$x(t) = e^{-t}u(t) - e^{-t}u(t-4),$$

then by linearity, the Fourier transform is:

$$\hat{x}(\omega) = \mathcal{F}\{x(t)\} = \mathcal{F}\{e^{-t}u(t)\} - \mathcal{F}\{e^{-t}u(t-4)\}.$$

The last term can be computed as follows:

$$\mathcal{F}\{e^{-t}u(t-4)\} = \int_{-\infty}^{\infty} e^{-t}u(t-4)e^{-i\omega t} dt = \int_4^{\infty} e^{-t}e^{-i\omega t} dt = \int_4^{\infty} e^{-(1+i\omega)t} dt,$$

Note that we cannot use the exponential decay formula here, as the term is $e^{-t}u(t-4)$ which is not the same as $e^{-(t-4)}u(t-4)$. This last expression can be solved with exponential decay, but that's not the case here, so we integrate manually, which gives:

$$\mathcal{F}\{e^{-t}u(t-4)\} = \left[-\frac{1}{1+i\omega} e^{-(1+i\omega)t} \right]_4^{\infty} = \frac{1}{1+i\omega} e^{-4(1+i\omega)}.$$

The first term is easily found as it is an exponential decay with $\beta = 1$, giving:

$$\hat{x}(\omega) = \frac{1}{1+i\omega} - e^{-4(1+i\omega)} \frac{1}{1+i\omega} = \underline{\underline{\frac{1}{1+i\omega} (1 - e^{-4(1+i\omega)})}}.$$

2. a) Let $\hat{x}(\omega) = 2 + \cos \omega$, then the inverse Fourier transform is:

$$\begin{aligned} x(t) &= \mathcal{F}^{-1}\{\hat{x}(\omega)\} = \mathcal{F}^{-1}\{2\} + \mathcal{F}^{-1}\{\cos \omega\}, \\ &= 2\delta(t) + \frac{1}{2}(\mathcal{F}^{-1}\{e^{i\omega}\} + \mathcal{F}^{-1}\{e^{-i\omega}\}), \\ &= 2\delta(t) + \frac{1}{2}(\delta(t+1) + \delta(t-1)), \end{aligned}$$

as $\mathcal{F}^{-1}\{e^{i\omega\tau}\} = \delta(t+\tau)$.

b) For $\hat{x}(\omega) = \frac{1}{1+i\omega} - \frac{1}{2+i\omega}$, then the inverse Fourier transform is:

$$\begin{aligned} x(t) &= \mathcal{F}^{-1}\{\hat{x}(\omega)\} = \mathcal{F}^{-1}\left\{\frac{1}{1+i\omega}\right\} - \mathcal{F}^{-1}\left\{\frac{1}{2+i\omega}\right\}, \\ &= \underline{\underline{e^{-t}u(t) - e^{-2t}u(t)}}, \end{aligned}$$

as $\mathcal{F}\{e^{-\beta t}u(t)\} = \frac{1}{\beta+i\omega}$.

c) Let $\hat{x}(\omega) = i\delta(\omega - 100\pi) - i\delta(\omega + 100\pi)$, then the inverse Fourier transform is:

$$\begin{aligned} x(t) &= \mathcal{F}^{-1}\{\hat{x}(\omega)\} = i\mathcal{F}^{-1}\{\delta(\omega - 100\pi)\} - i\mathcal{F}^{-1}\{\delta(\omega + 100\pi)\}, \\ &= \frac{i}{2\pi}e^{100\pi it} - \frac{i}{2\pi}e^{-100\pi it}, \\ &= \frac{1}{2\pi}(e^{i\pi/2}e^{100\pi it} + e^{-i\pi/2}e^{-100\pi it}), \\ &= \underline{\underline{\frac{1}{\pi}\cos\left(100\pi t + \frac{\pi}{2}\right)}}. \end{aligned}$$

3. a) Let $x(t) = u(t+3)u(3-t)$, then $x(t)$ is a rectangular function with corners at $t = \pm 3$ and height 1. Therefore, we have:

$$\hat{x}(\omega) = \mathcal{F}\{x(t)\} = \int_{-\infty}^{\infty} u(t+3)u(3-t)e^{-i\omega t}dt = \int_{-3}^3 e^{-i\omega t}dt = \left[\frac{1}{-i\omega}e^{-i\omega t} \right]_{t=-3}^3.$$

Evaluating this yields:

$$\hat{x}(\omega) = -\frac{1}{i\omega}(e^{-3i\omega} - e^{3i\omega}) = \frac{1}{i\omega}(e^{3i\omega} - e^{-3i\omega}) = \underline{\underline{\frac{2\sin(3\omega)}{\omega}}}.$$

b) Consider the signal $x(t)$ given as:

$$x(t) = \sin(4\pi t) \sin(50\pi t),$$

then:

$$\begin{aligned} \sin(4\pi t) &= \frac{1}{2i}(e^{4\pi it} - e^{-4\pi it}), \\ \sin(50\pi t) &= \frac{1}{2i}(e^{50\pi it} - e^{-50\pi it}), \end{aligned}$$

which gives:

$$x(t) = \frac{1}{2i} (e^{4\pi it} - e^{-4\pi it}) \frac{1}{2i} (e^{50\pi it} - e^{-50\pi it}) = -\frac{1}{4} (e^{i54\pi t} - e^{-i46\pi t} - e^{i46\pi t} + e^{-i54\pi t}),$$

Next, we evaluate the transform:

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt = -\frac{1}{4} \int_{-\infty}^{\infty} (e^{i54\pi t} - e^{-i46\pi t} - e^{i46\pi t} + e^{-i54\pi t}) e^{-i\omega t} dt,$$

and obtain:

$$\hat{x}(\omega) = -\frac{\pi}{2} [\delta(\omega - 54\pi) - \delta(\omega + 46\pi) - \delta(\omega - 46\pi) + \delta(\omega + 54\pi)].$$

c) Let

$$x(t) = \frac{\sin(4\pi t)}{\pi t} \sin(50\pi t),$$

then, to compute the Fourier transform we apply the convolution theorem, have that:

$$\mathcal{F}\left\{\frac{\sin(4\pi t)}{\pi t}\right\} = [u(\omega + 4\pi) - u(\omega - 4\pi)] = \hat{x}_1(\omega),$$

$$\mathcal{F}\{\sin(50\pi t)\} = -i\pi[\delta(\omega - 50\pi) - \delta(\omega + 50\pi)] = \hat{x}_2(\omega),$$

then in frequency domain we get:

$$\begin{aligned} \hat{x}_1(\omega) * \hat{x}_2(\omega) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}_1(\tau) \hat{x}_2(\omega - \tau) d\tau, \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} [u(\tau + 4\pi) - u(\tau - 4\pi)][-\pi i(\delta(\omega - \tau - 50\pi) - \delta(\omega - \tau + 50\pi))] d\tau, \\ &= \frac{1}{2i} \int_{-\infty}^{\infty} [u(\tau + 4\pi) - u(\tau - 4\pi)][\delta((\omega - 50\pi) - \tau) - \delta((\omega + 50\pi) - \tau)] d\tau, \end{aligned}$$

The integral consists of four terms, these being:

$$\int_{-\infty}^{\infty} u(\tau + 4\pi) \delta(\omega - 50\pi - \tau) d\tau = u(\omega - 50\pi + 4\pi) = u(\omega - 46\pi),$$

$$\int_{-\infty}^{\infty} u(\tau + 4\pi) \delta(\omega + 50\pi - \tau) d\tau = u(\omega + 50\pi + 4\pi) = u(\omega + 54\pi),$$

$$\int_{-\infty}^{\infty} u(\tau - 4\pi) \delta(\omega - 50\pi - \tau) d\tau = u(\omega - 50\pi - 4\pi) = u(\omega - 54\pi),$$

$$\int_{-\infty}^{\infty} u(\tau - 4\pi) \delta(\omega + 50\pi - \tau) d\tau = u(\omega + 50\pi - 4\pi) = u(\omega + 46\pi),$$

combining all these we get:

$$\hat{x}(\omega) = \frac{1}{2i} [u(\omega - 46\pi) - u(\omega + 54\pi) - u(\omega - 54\pi) + u(\omega + 46\pi)].$$

d) If $\hat{x}(\omega) = \cos^2(\omega)$, then we find $x(t)$ by using the inverse Fourier transform:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \cos^2(\omega) e^{i\omega t} d\omega.$$

Firstly, we rewrite our function as:

$$\cos^2(\omega) = \left(\frac{1}{2} (e^{i\omega} + e^{-i\omega}) \right)^2 = \frac{1}{4} (e^{2i\omega} + e^{-2i\omega} + 2),$$

Substituting this we obtain:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{1}{4} (e^{2i\omega} + e^{-2i\omega} + 2) e^{i\omega t} d\omega = \underline{\underline{\frac{1}{4} \delta(t+2) + 2\delta(t) + \frac{1}{4} \delta(t-2)}}.$$

4. Let $\hat{x}(\omega)$ be the Fourier transform of $x(t)$. Define $y(t) = x(t - \tau)$, then the Fourier transform of $y(t)$ is by definition:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} y(t) e^{-i\omega t} dt = \int_{-\infty}^{\infty} x(t - \tau) e^{-i\omega t} dt.$$

Let $u = t - \tau$, then $du = dt$, thus:

$$\hat{y}(\omega) = \int_{-\infty}^{\infty} x(u) e^{-i\omega(u+\tau)} du = e^{-i\omega\tau} \int_{-\infty}^{\infty} x(u) e^{-i\omega u} du = e^{-i\omega\tau} \hat{x}(\omega),$$

as $\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt$.

Amplitude modulated radio [Optional]

One application of the convolution theorem is explaining how amplitude modulated (AM) radio works. This is a nice and simple example, but it also demonstrates a more abstract concept of up- and downconversion of signals, which is used e.g., in radio hardware⁴.

Upconversion means that a signal is shifted in frequency up to a higher frequency range. Downconversion means the opposite of that.

The underlying principle behind amplitude modulation is very simple. An audio signal $x(t)$ is multiplied⁵ with a carrier signal $c(t) = \cos(\omega_c t)$, which is a high frequency sinusoidal signal:

$$y(t) = x(t)c(t) \quad (10.103)$$

$$= x(t) \cos(\omega_c t). \quad (10.104)$$

This is depicted in the block diagram shown in Figure 10.9.

Here $\omega_c = 2\pi f_c$ is the carrier angular frequency. A typical carrier frequency f_c for AM terrestrial radio transmissions is $f_c \in [0.1, 30]$ MHz.

These types of radio transmissions that provide global radio broadcasts still exist. Because signals bounce between the Earth's ionosphere and ground in this frequency range, signals transmitted from one location in the world can in optimal conditions travel to the other side of the Earth.

We'll now study what the spectrum $\hat{y}(\omega)$ of signal $y(t)$ looks like. The key to understanding what $\hat{y}(\omega)$ looks like is that the spectrum of the signal $x(t)$ is band-limited. This means that $\hat{x}(\omega) = 0$ when $|\omega| > \omega_{\max}$. For example, in the case of audio, we know that there are no audible spectral components at frequencies above 20 kHz. Therefore, in order to transfer an audio signal, we do not need to include higher frequency components.

In Figure 10.10, the magnitude spectrum of the real-valued audio signal $|\hat{x}(\omega)|$ is shown with a red line. The plot represents a magnitude spectrum for an arbitrary real-valued signal that occupies angular frequencies between $\pm\omega_{\max}$. Because the audio signal is real-valued $x(t) \in \mathbb{R}$, the magnitude spectrum is symmetric around the y-axis. The spectrum of the signal $c(t)$ can be determined by first representing the signal as a sum of two complex sinusoids (using Euler's formula):

$$c(t) = \cos(\omega_c t) = \frac{1}{2}e^{i\omega_c t} + \frac{1}{2}e^{-i\omega_c t}. \quad (10.105)$$

We know that the spectrum of a complex sinusoid is a frequency shifted Dirac-delta, and hence, the Fourier transform of $c(t)$ is

$$\hat{c}(\omega) = \pi\delta(\omega - \omega_c) + \pi\delta(\omega + \omega_c). \quad (10.106)$$

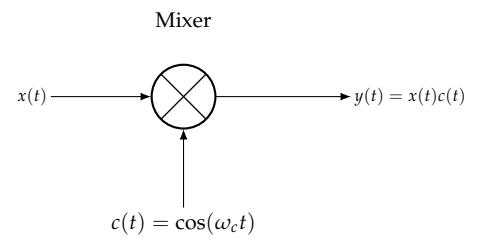


Figure 10.9: A simple block diagram representation of an AM radio signal generator system, which multiplies an audio signal $x(t)$ with a carrier wave $c(t) = \cos(\omega_c t)$.

⁴ This same principle is used by nearly all radio devices, such as the radios found in your smartphone. The only difference is that the signal $x(t)$ does not represent an audio waveform, but a scheme that encodes digital information into a 1d waveform.

⁵ In the early days of radio, this multiplication was implemented using vacuum tubes. It can also be implemented using a transistor or a diode circuit. Nowadays, it is more and more common to modulate signals with digital signal processing. Mathematically, the operation is the same for all cases.

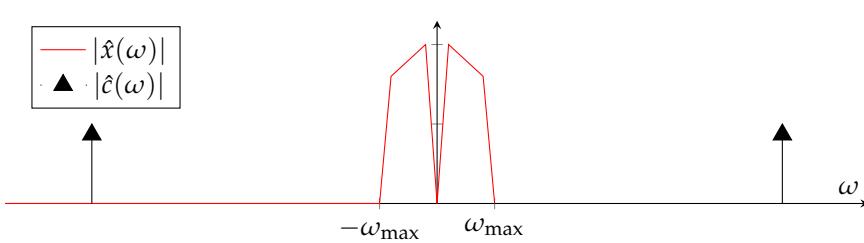


Figure 10.10: A spectral representation of signals $x(t)$ and $c(t)$.

We also know from the convolution theorem, that multiplying two signals in time domain results in convolution in frequency domain:

$$y(t) = x(t)c(t) \xleftrightarrow{\mathcal{F}} \hat{y}(\omega) = \frac{1}{2\pi} \hat{x}(\omega) * \hat{c}(\omega) . \quad (10.107)$$

We can now inspect what the spectrum $\hat{y}(\omega)$ looks like by evaluating the convolution:

$$\hat{y}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega') \hat{c}(\omega - \omega') d\omega' \quad (10.108)$$

$$= \frac{1}{2} \int_{-\infty}^{\infty} \hat{x}(\omega') [\delta(\omega - \omega_c - \omega') + \delta(\omega + \omega_c - \omega')] d\omega' \quad (10.109)$$

$$= \frac{1}{2} \hat{x}(\omega - \omega_c) + \frac{1}{2} \hat{x}(\omega + \omega_c) . \quad (10.110)$$

In this convolution, the Dirac delta functions act as frequency shifters, which shift the function $\hat{x}(\omega)$ up and down in frequency by ω_c .

Figure 10.11 shows a plot of the spectrum of the original signal $x(t)$, and the upconverted signal $y(t)$.

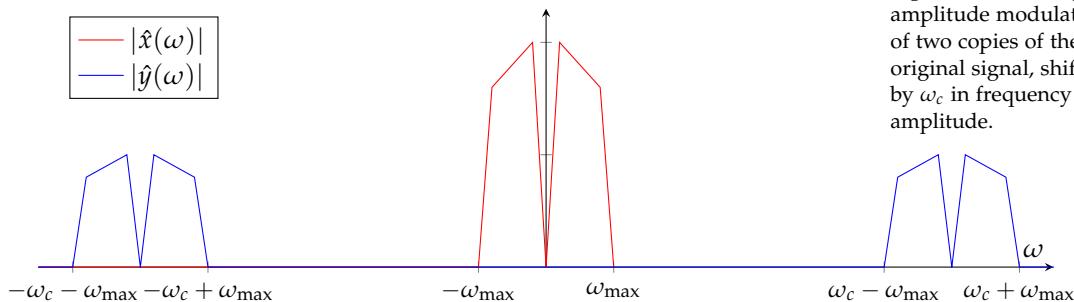


Figure 10.11: The spectrum of the amplitude modulated signal consists of two copies of the spectrum of the original signal, shifted up and down by ω_c in frequency and scaled by $\frac{1}{2}$ in amplitude.

IN ORDER TO RECONSTRUCT THE AUDIO SIGNAL $x(t)$ FROM THE MODULATED SIGNAL $y(t)$, the signal is first multiplied with the carrier signal:

$$w(t) = y(t)c(t) . \quad (10.111)$$

This operation again is a frequency shift, just like in the modulation step. We can again use the convolution theorem to obtain the spectrum of the intermediate signal $w(t)$:

$$\hat{w}(\omega) = \frac{1}{2\pi} \hat{y}(\omega) * \hat{c}(\omega) \quad (10.112)$$

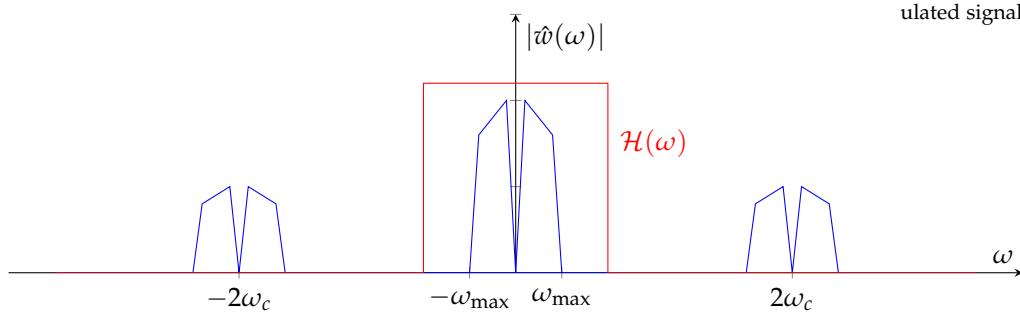
$$= \frac{1}{2} \hat{y}(\omega + \omega_c) + \frac{1}{2} \hat{y}(\omega - \omega_c) \quad (10.113)$$

$$= \frac{1}{4} [\hat{x}(\omega + \omega_c + \omega_c) + \hat{x}(\omega - \omega_c + \omega_c)] \quad (10.114)$$

$$+ \frac{1}{4} [\hat{x}(\omega + \omega_c - \omega_c) + \hat{x}(\omega - \omega_c - \omega_c)] \quad (10.115)$$

$$= \frac{1}{4} \hat{x}(\omega - 2\omega_c) + \frac{1}{4} \hat{x}(\omega + 2\omega_c) + \frac{1}{2} \hat{x}(\omega) . \quad (10.116)$$

There are now three copies of the spectrum $\hat{x}(\omega)$. One centered at $\omega = 0$ and two centered at twice the carrier frequency $\omega = \pm 2\omega_c$ as shown in Figure 10.13.



We almost have what we need, but we need to get rid of the high frequency components. This can be done with the help of a low-pass filter:

$$\mathcal{H}(\omega) = \begin{cases} 2, & |\omega| \leq \omega_{\max} \\ 0, & \text{otherwise} \end{cases} . \quad (10.117)$$

which also corrects the scale back to the original. The frequency response of such a filter is shown in both Figure 10.13 and 10.14 with red. The output signal spectrum is then:

$$\hat{v}(\omega) = \mathcal{H}(\omega) \hat{w}(\omega) \quad (10.118)$$

$$= \hat{x}(\omega) . \quad (10.119)$$

And therefore $v(t) = x(t)$. Multiplying with a cosine signal and low-pass filtering has shifted the signal back to DC and allowed us to reconstruct the original signal $x(t)$.

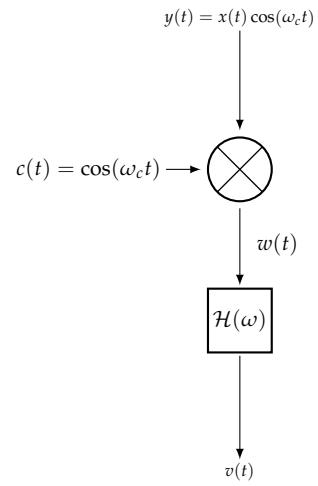


Figure 10.12: Demodulation or conversion of a high frequency signal $y(t)$ into an audio signal $v(t)$.

Figure 10.13: Spectra of an AM demodulated signal.

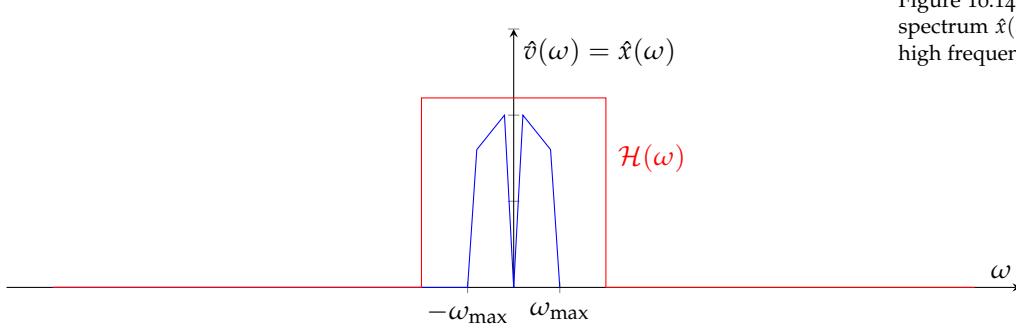
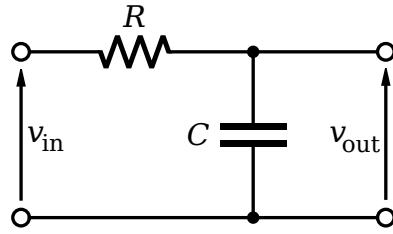


Figure 10.14: The original signal spectrum $\hat{x}(\omega)$ reconstructed from the high frequency signal.

RC-circuit [Optional]

Consider the following electrical circuit:



This is a very simple analogue filter implemented using a resistor and a capacitor. This type of circuit is often used in, e.g., audio electronics to remove high-pitched signals.

This system can be modeled as a linear time-invariant system. In this case, $v_{\text{in}}(t) = x(t)$ and $v_{\text{out}}(t) = y(t)$. This system has the following impulse response:

$$h(t) = \frac{1}{RC} e^{-\frac{1}{RC}t} u(t) = \alpha e^{-\alpha t} u(t) \quad (10.120)$$

where the real-valued constant $\alpha > 0$.

In order to determine the frequency response, we evaluate the Fourier transform of the impulse response:

$$\mathcal{H}(\omega) = \int_{-\infty}^{\infty} \alpha e^{-\alpha t} u(t) e^{-i\omega t} dt \quad (10.121)$$

$$= \alpha \int_0^{\infty} e^{-(\alpha+i\omega)t} dt \quad (10.122)$$

$$= \alpha \left. \frac{e^{-(\alpha+i\omega)t}}{-(\alpha+i\omega)} \right|_{t=0} \quad (10.123)$$

$$= \frac{\alpha}{\alpha + i\omega}. \quad (10.124)$$

In order to find the magnitude response, we'll need to establish that

$$\mathcal{H}^*(\omega) = \frac{\alpha}{\alpha - i\omega}, \quad (10.125)$$

which is relatively easy to show⁶.

The magnitude response is then:

$$|\mathcal{H}(\omega)| = \sqrt{\mathcal{H}(\omega)\mathcal{H}^*(\omega)} \quad (10.126)$$

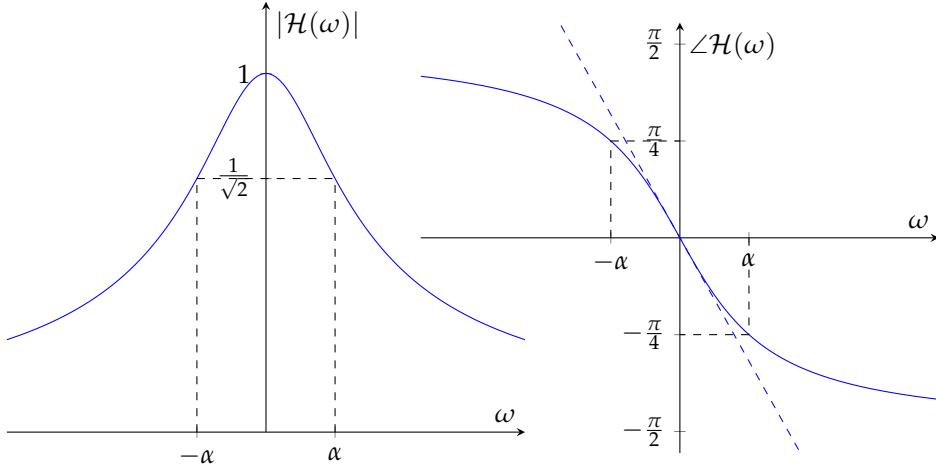
$$= \frac{\alpha}{\sqrt{\alpha^2 + \omega^2}}. \quad (10.127)$$

The phase response is⁷:

$$\angle \mathcal{H}(\omega) = \tan^{-1} \frac{\text{Im}(\mathcal{H}(\omega))}{\text{Re}(\mathcal{H}(\omega))} \quad (10.128)$$

$$= -\tan^{-1} \frac{\omega}{\alpha}. \quad (10.129)$$

The magnitude and phase responses are plotted below:



Note that the “half-power” point, where $|\mathcal{H}(\omega)|^2 = \frac{1}{2}$ corresponds to $\omega = \pm\alpha$. This half power width of a filter is often referred to as the full width half maximum (FWHM). At this “half-power” point, the phase is $-\tan^{-1}(\pm 1) = \mp \frac{\pi}{4}$. For our example filter, the output power has dropped to half when the frequency is:

$$\omega = 2\pi f = \frac{1}{RC} \quad (10.130)$$

or

$$f = \frac{1}{2\pi RC}. \quad (10.131)$$

For example, if $R = 1 \text{ k}\Omega$ and $C = 100 \text{ pF}$, the cutoff frequency of the filter is 1.6 MHz.

When looking at the Taylor series expansion of the phase around zero, recalling that

$$\frac{d}{d\omega} \tan^{-1}(-\omega/\alpha) = \frac{\alpha}{\omega^2 + \alpha^2}, \quad (10.132)$$

we get

$$\angle \mathcal{H}(\omega) \approx -\frac{1}{\alpha} \omega = -RC\omega. \quad (10.133)$$

⁶ For example using:

$$\frac{\alpha}{\alpha + i\omega} = \frac{\alpha(\alpha - i\omega)}{\alpha^2 + \omega^2} = \frac{\alpha^2}{\alpha^2 + \omega^2} - i \frac{\alpha\omega}{\alpha^2 + \omega^2}$$

⁷ recall that $\tan^{-1}(-x) = -\tan^{-1}(x)$

This is shown as a dashed blue line in the figure above.

If we compare this with the equation for delay of a sinusoid $x(t) = Ae^{i\omega t}$, which we can derive from

$$x(t - t_0) = Ae^{i\omega(t-t_0)} = Ae^{i\omega t}e^{-i\omega t_0} = Ae^{i\omega t}e^{i\varphi} \quad (10.134)$$

to be $-\omega t_0 = -RC\omega$, we see that the filter corresponds to approximately a $t_0 = RC$ second time delay. This approximation is valid for low values of $|\omega| < \frac{1}{2}\alpha$. In the case of $R = 1 \text{ k}\Omega$ and $C = 100 \text{ pF}$, the time delay is approximately: $0.1 \mu\text{s}$.

11

Discrete-time Signals

In this chapter, we introduce discrete-time signals. These are the type of signals that are used for digital signal processing. We'll cover the following topics:

- Discretization
- Aliasing
- Oversampling and undersampling
- Reconstruction
- Shannon-Nyquist sampling theorem

We'll need to know about complex sinusoidal signals and the frequency domain representation of signals (the Fourier transform).

Sampling

A continuous-time signal can be converted into a discrete-time signal by sampling it. This can be idealized as a system that performs the following operation:

$$x[n] = x(nT_s) . \quad (11.1)$$

In this process, a continuous function $x(t) \in \mathbb{C}$ is converted into an ordered sequence of numbers $x[n] \in \mathbb{C}$, where $n \in \mathbb{Z}$ indicates the order of the numbers.

The reason why we consider complex-valued signals in addition to real-valued signals is that there are many applications where the signals are naturally complex-valued. Radio signal processing is one example.

The sample-spacing T_s determines how densely samples of the continuous-time signal are taken. Sample-spacing is related with the sample-rate or sampling-rate f_s as follows:

$$\boxed{f_s = \frac{1}{T_s}} . \quad (11.2)$$

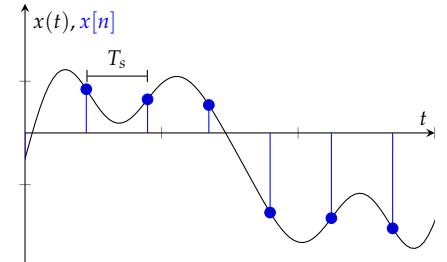


Figure 11.1: A continuous-time signal $x(t)$, and a discrete-time signal $x[n]$. Sample-spacing T_s is related to sample-rate as follows: $T_s = 1/f_s$.

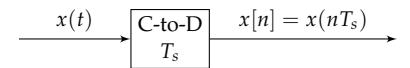


Figure 11.2: An ideal continuous-time to Discrete-time (C-to-D) converter.

Sample-rate typically is given in units of samples per second or hertz, but other units are also used in various applications. For example, the resolution of a digital image is often given in units of dots per inch (DPI).

Sampling a complex sinusoidal signal

In order to understand sampling, you need to understand how sampling affects a complex sinusoidal signal:

$$x(t) = Ae^{i\phi}e^{i\omega_0 t}. \quad (11.3)$$

Because we can use the Fourier transform to represent signals as a superposition of complex sinusoidal signals, knowledge of what happens to a fixed frequency signal will tell us how discretization affect signals more generally.

So what happens to a complex sinusoidal signal when we sample it?

$$x[n] = x(nT_s) \quad (11.4)$$

$$= Ae^{i\phi}e^{i\omega_0 n T_s} \quad (11.5)$$

$$= Ae^{i\phi}e^{i\hat{\omega}_0 n}. \quad (11.6)$$

What happens is that we get a discrete-time complex sinusoidal signal. This signal has a special type of frequency:

$$\hat{\omega}_0 = \omega_0 T_s. \quad (11.7)$$

The angular frequency $\hat{\omega}_0$ is called a *discrete-time angular frequency*. It indicates how many radians per sample the phase of the complex sinusoidal signal advances. With a discrete-time signal, time is only implied from the sample spacing that was used to discretize the signal, which is why no time unit is attached to $\hat{\omega}_0$. There's a hat on $\hat{\omega}_0$ to signify that this is not the same kind of frequency as the continuous-time angular frequency ω , which has units of radians per second.

Figure 11.3 shows an example of how the values of the discrete-time complex sinusoidal signal behave with increasing sample index, obtaining values on a circle of radius A in the complex plane. The amount of angle per sample that the signal advances for each sample on the circle is given by the discrete-time angular frequency $\hat{\omega}_0$.

Frequency aliases

What happens if $\hat{\omega}$ gets larger? What if it is so large that the phasor rotates around the circle on the complex plane one or more times every sample? This phenomenon is called aliasing.

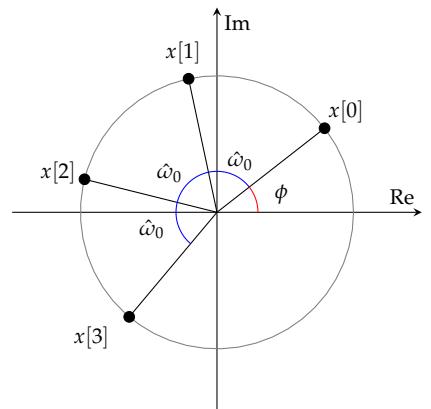
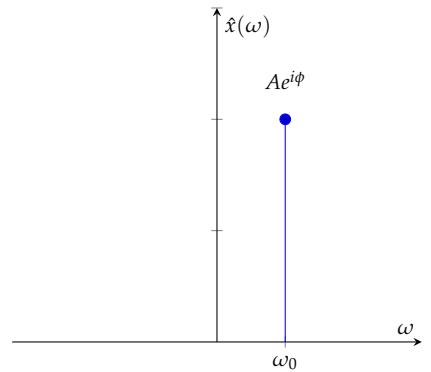


Figure 11.3: A discretized complex sinusoidal signal $x[n] = Ae^{i\phi}e^{i\hat{\omega}_0 n}$ with phase increments of $\hat{\omega}_0$ radians per sample. The initial phase for sample $n = 0$ is given by ϕ .

Here's a more formal way of defining aliasing. The complex sinusoidal signal $e^{i\omega_0 n}$ is not unique. There are infinitely many possible frequency increments that result in the same signal:

$$x[n] = Ae^{i\phi} e^{i(\hat{\omega}_0 + 2\pi k)n} \quad (11.8)$$

$$= Ae^{i\phi} e^{i\hat{\omega}_0 n} \quad (11.9)$$

where $k \in \mathbb{Z}$. What this means is that we can select any one of the discrete-time angular frequencies:

$$\hat{\omega}_k = \hat{\omega}_0 + 2\pi k \quad (11.10)$$

and our discrete-time complex sinusoidal signal will be identical. There is no way of telling just from the signal itself what the true discrete-time angular frequency is. Frequencies $\hat{\omega}_k$ are called *aliases*. There are infinitely many such aliases.

What continuous-time frequencies do these aliases correspond to? We can work this out by looking at the continuous-time signal of frequency ω_0 :

$$x(t) = Ae^{i\phi} e^{i\omega_0 t} \quad (11.11)$$

When discretized, each of the following signals are identical:

$$x[n] = Ae^{i\phi} e^{i\omega_0 T_s n} \quad (11.12)$$

$$= Ae^{i\phi} e^{i(\omega_0 T_s + 2\pi k)n} \quad (11.13)$$

$$= Ae^{i\phi} e^{i(\omega_0 + 2\pi k/T_s)T_s n} \quad (11.14)$$

$$= Ae^{i\phi} e^{i(\omega_0 + 2\pi k f_s)T_s n} \quad (11.15)$$

$$= Ae^{i\phi} e^{i\omega_k T_s n} \quad (11.16)$$

We've used the relationship between sample-rate $f_s = 1/T_s$ and sample spacing here. The aliases map to continuous-time angular frequency as follows:

$$\omega_k = \omega_0 + 2\pi k f_s \quad (11.17)$$

Any one of these continuous-time angular frequencies ω_k would result in an identical discrete-time complex sinusoidal signal when discretized with sample-rate f_s . Dividing by 2π , we get the aliasing behavior in units of hertz (samples per second)¹:

$$f_k = f_0 + k f_s \quad (11.18)$$

The phenomena of aliasing is problematic. We cannot uniquely distinguish a discrete-time complex sinusoidal signal $x[n] = Ae^{i\hat{\omega} n}$ which continuous-time signal it corresponds to. Conversely, there are infinitely many continuous-time sinusoidal signals $x(t) = Ae^{i\omega t}$ that will result in the same discrete-time signal. In order to rule out

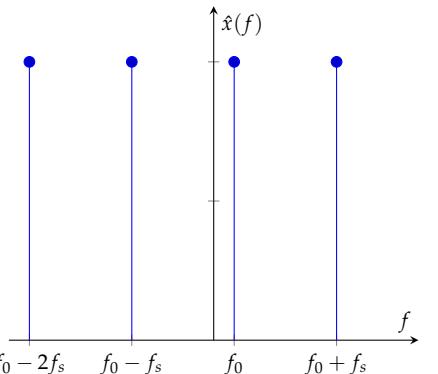


Figure 11.4: Each one of these signals $Ae^{i\phi} e^{i2\pi(f_0 + kf_s)t}$ would result in the same discrete-time complex sinusoidal signal when discretized with sample-rate f_s . Note that we're showing the spectrum with continuous-time frequency in units of hertz (cycles per second) instead of radians per second.

¹ remember that angular frequency (radians per second) and frequency (cycles per second) are related as follows $\omega = 2\pi f$.

the wrong solutions, we need a priori information about the spectral occupancy of the continuous-time signal.

Solving the problem of aliasing, by ensuring that there is a uniquely one-to-one mapping between continuous-time and discrete-time frequencies for all the spectral components of a signal, leads to the Shannon-Nyquist sampling theorem, which is introduced at the end of this chapter.

Real-valued signals

The concept of aliasing of complex sinusoidal signals allows us to inspect aliasing of real-valued sinusoidal signals, which come in pairs of positive and negative frequency spectral components. Understanding aliasing for real-valued signals is slightly more complicated than understanding complex-valued signals, because of the symmetric pairing of the positive and negative frequency components.

The Fourier transform of a real-valued signal $x(t) \in \mathbb{R}$ is conjugate symmetric

$$\hat{x}(\omega) = \hat{x}^*(-\omega). \quad (11.19)$$

This means that real-valued signals are composed of conjugate symmetric pairs of positive and negative frequency spectral components:

$$A \cos(\omega t + \phi) = \frac{A}{2} (e^{i\phi} e^{i\omega t} + e^{-i\phi} e^{-i\omega t}). \quad (11.20)$$

In order to understand how aliasing affects real-valued signals, we need to investigate how pairs of positive and negative frequency spectral components are affected.

A conjugate symmetric pairing of positive and negative frequencies means that there is no distinction between a positive and a negative frequency signal². For example, let's say we have a signal with angular frequency $\omega = -0.1$:

$$A \cos(-0.1t + \phi) = \frac{A}{2} (e^{i\phi} e^{-i0.1t} + e^{-i\phi} e^{i0.1t}). \quad (11.21)$$

This is the same thing as a cosine signal with frequency 0.1, but a sign flipped phase:

$$\frac{A}{2} (e^{-i\phi} e^{i0.1t} + e^{i\phi} e^{-i0.1t}) = A \cos(0.1t - \phi) \quad (11.22)$$

$$= A \cos(-0.1t + \phi). \quad (11.23)$$

This means that you can always represent a real-valued sinusoidal signal using only positive valued frequencies, but it does not apply for complex-valued signals.

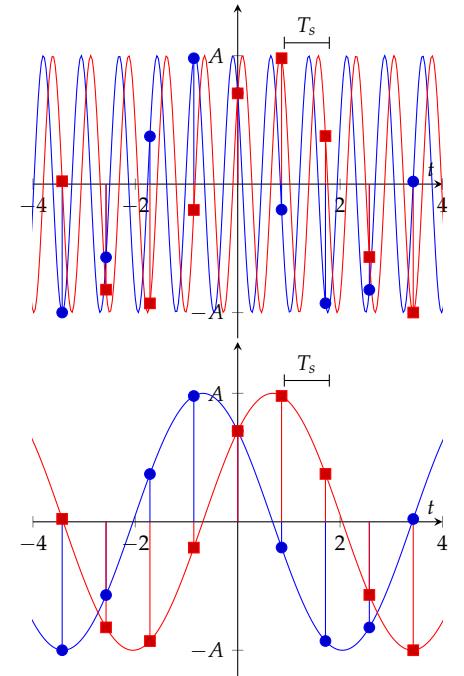


Figure 11.5: Two different complex sinusoidal signals that result in the same discrete-time signal.

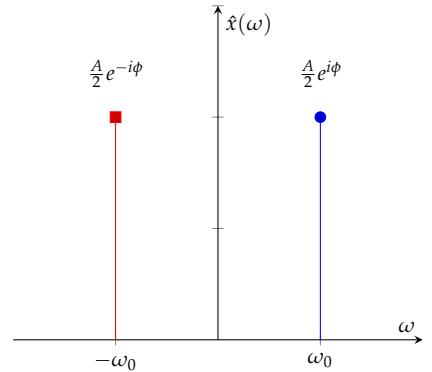


Figure 11.6: A real-valued signal consists of a positive and negative frequency spectral component, which are conjugate symmetric.

² For complex sinusoidal signals $e^{-i\omega t}$ and $e^{i\omega t}$ are two unique signals.

In the case of real-valued signals, the frequency alias corresponding to a negative frequency is called a *folded alias*. For complex sinusoidal signals, a folded alias does not exist, as positive and negative frequency components are not necessarily paired.

Example: Sinusoidal signal with $f_0 = 1 \text{ Hz}$ and $f_s = 10 \text{ Hz}$

Consider the following sinusoidal signal with a frequency of $f_0 = 1 \text{ Hz}$ and phase ϕ :

$$x(t) = A \cos(2\pi f_0 t + \phi) \quad (11.24)$$

$$x(t) = A \cos(2\pi t + \phi). \quad (11.25)$$

When discretized with a sample-rate $f_s = 10 \text{ Hz}$ or $T_s = 1/f_s = 0.1 \text{ s}$ this becomes:

$$x[n] = x(nT_s) \quad (11.26)$$

$$= A \cos(2\pi n T_s + \phi), \quad (11.27)$$

$$= A \cos(0.2\pi n + \phi). \quad (11.28)$$

The discrete-time angular frequency is $\hat{\omega}_0 = 0.2\pi$. For any $k \in \mathbb{Z}$ this signal is identical to:

$$x[n] = A \cos([0.2\pi + 2\pi k]n + \phi). \quad (11.29)$$

In discrete-time angular frequencies (radians per sample), the aliases are:

$$\hat{\omega}_k = (0.2\pi + 2\pi k) \quad (11.30)$$

$$= \dots, -3.8\pi, -1.8\pi, 0.2\pi, 2.2\pi, 4.2\pi, \dots \quad (11.31)$$

In continuous-time frequency (Hz), the aliases would be:

$$f_k = f_0 + f_s k \quad (11.32)$$

$$= \dots, -19, -9, 1, 11, 21, 31, \dots \quad (11.33)$$

In other words, a discretized cosine signal produced from any of the frequencies f_k would be identical to a 1 Hz signal, when using 10 Hz sample rate. The negative solutions are called folded aliases, which can be interpreted as cosine signals with positive frequencies 9, 19, ... but with a sign flipped phase.

Figure 11.9 shows part of the discrete-frequency spectrum where there are three aliases of the cosine signal. There are infinitely many aliases, but only three are shown here. It is customary to call the region of the spectrum where $|\hat{\omega}| < \pi$ or $|f| < f_s/2$, the *principal spectrum*. This region of the spectrums contains the smallest discrete-time angular frequency aliases.

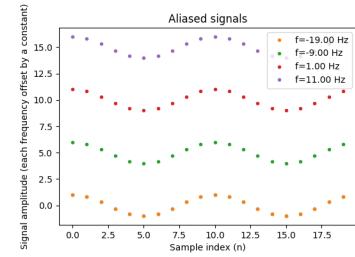


Figure 11.7: Example of signal aliasing for $f_0 \in \{-19, -9, 1, 11, 21\}$ Hz and $f_s = 10 \text{ Hz}$. All signals alias identically. Python code: 014_sampling/aliasing_example.py.

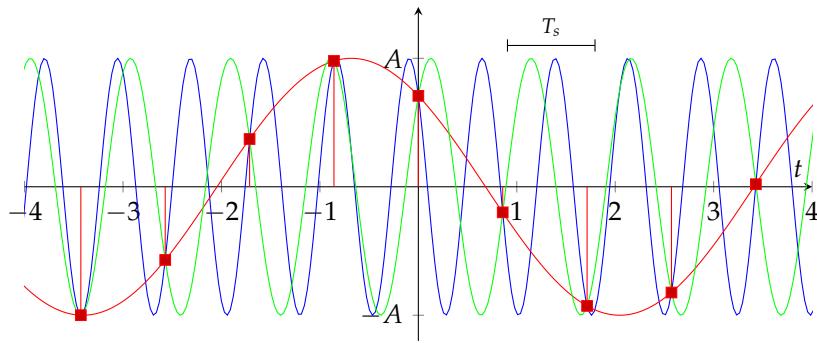


Figure 11.8: The following figure illustrates what the continuous-time and discrete-time signals look like. They are all identical in discrete-time, because the continuous-time signals intersect at the sampling points. The red line shows $\omega_0 = 0.2\pi f_s$ (1 Hz), the green line shows $\omega_1 = -1.8\pi f_s$ (-9 Hz), and the blue line shows $\omega_2 = 2.2\pi f_s$ (11 Hz). One can also see the phase-flip on the negative frequency folded alias, shown in green. The blue and red continuous-time signals are both sampled at identical phases, but the green not.

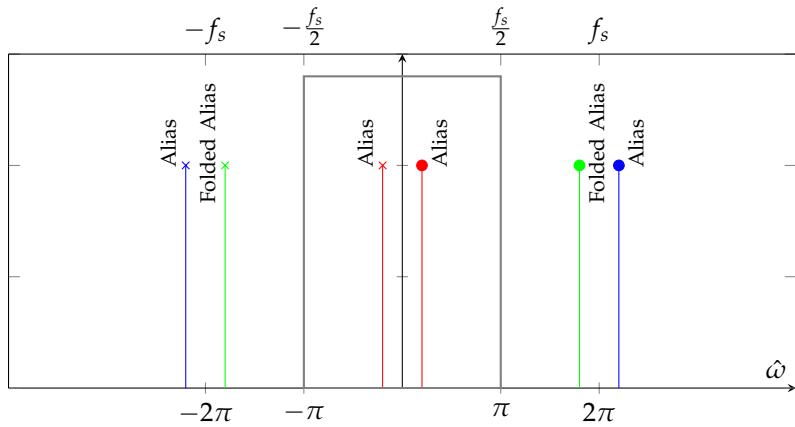


Figure 11.9: Spectral aliases of the components plotted as a function of discrete-time angular frequency $\hat{\omega}$ (radians per sample). The pairing of positive and negative frequencies indicates that this signal is real-valued, and that the $\hat{\omega}_{-1} = -1.8\pi$ alias corresponds to a cosine signal with a positive discrete-time angular frequency of 1.8π and a flipped phase. Frequency in units of hertz is shown on the top axis.

How could we be able to determine which one of the frequencies f_k is the true sinusoidal frequency component represented by the discrete-time signal? We can only do this if we know that the continuous-time frequencies of the spectral components of the signal lie within a certain range.

For example, if we knew that there are no spectral components with frequencies higher than 5 Hz present in the continuous-time signal ($|f| < 5$ Hz), then we could determine that the discrete-time signal is due to a 1 Hz continuous-time signal, because this is the only possible mapping between a discrete-time frequency and a continuous-time frequency.

Principal spectrum

The *principal spectrum* (also called normalized angular frequency) is the range of discrete-time angular frequencies between:

$$-\pi < \hat{\omega} < \pi . \quad (11.34)$$

It is the region for the lowest frequency discrete-time angular frequency representation for discrete-time spectral components that the signal consists of. Unaliased continuous-time frequencies between

$$-f_s/2 < f < f_s/2 \quad (11.35)$$

in units of hertz are mapped to this range of discrete-time angular frequencies.

By finding a suitable value for k , it is possible to find an alias of any discrete-time angular frequency $\hat{\omega}$, which lands in the range $-\pi < \hat{\omega} + 2\pi k < \pi$. This is called the *principal alias*. Every spectral component of a signal has an alias in the principal spectrum.

FOR EXAMPLE, consider a signal of frequency $f = 123$ Hz:

$$x(t) = \cos(2\pi ft) = \frac{1}{2}(e^{i2\pi ft} + e^{-i2\pi ft}) , \quad (11.36)$$

which is sampled using a sample-rate of $f_s = 100$. The discrete-time angular frequency is $\hat{\omega} = \pm 2\pi \cdot 123 \cdot \frac{1}{100} = \pm 2.46\pi$. This is outside the principal spectrum. If we shift this by $\mp 2\pi$, we obtain principal aliases: $\hat{\omega} = \pm 0.46\pi$.

Folding

Figure 11.10 shows an illustration by Texas Instruments, which nicely describes the concept of folding of spectral components.

Folding implies that at frequencies between $[f_s/2 + kf_s, f_s + kf_s]$ with $k = 0, 1, 2, \dots$, the order of the spectral components are flipped.

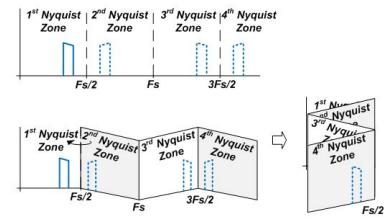


Figure 11.10: Folding.

Note that this only occurs if the signal is real-valued, as folding relies on a pairing of positive and negative frequency spectral components.

THE FOLLOWING EXAMPLE DEMONSTRATES FOLDING USING THREE REAL-VALUED SINUSOIDAL SIGNALS. Let us consider three sinusoidal signals:

$$x_1(t) = A \cos(2\pi f_0 t), \quad (11.37)$$

$$x_2(t) = A \cos(2\pi f_1 t), \quad (11.38)$$

$$x_3(t) = A \cos(2\pi f_2 t), \quad (11.39)$$

with frequencies:

$$f_0 = 0.1f_s, \quad (11.40)$$

$$f_1 = 0.2f_s, \quad (11.41)$$

$$f_2 = 0.3f_s. \quad (11.42)$$

When we discretize these signals, we obtain:

$$x_0[n] = A \cos(2\pi f_0 n / f_s) = A \cos(0.2\pi n), \quad (11.43)$$

$$x_1[n] = A \cos(2\pi f_1 n / f_s) = A \cos(0.4\pi n), \quad (11.44)$$

$$x_2[n] = A \cos(2\pi f_2 n / f_s) = A \cos(0.6\pi n). \quad (11.45)$$

These signals have spectral components with discrete-time angular frequencies within the principal spectrum:

$$\hat{\omega}_0 = \pm 0.2\pi, \quad (11.46)$$

$$\hat{\omega}_1 = \pm 0.4\pi, \quad (11.47)$$

$$\hat{\omega}_2 = \pm 0.6\pi. \quad (11.48)$$

If we now take three more sinusoidal signals, with frequencies:

$$f_3 = 0.7f_s, \quad (11.49)$$

$$f_4 = 0.8f_s, \quad (11.50)$$

$$f_5 = 0.9f_s. \quad (11.51)$$

When discretized we obtain the following principal aliases:

$$x_3[n] = A \cos(2\pi f_3 n / f_s) = A \cos(1.4\pi n) = A \cos(-0.6\pi n), \quad (11.52)$$

$$x_4[n] = A \cos(2\pi f_4 n / f_s) = A \cos(1.6\pi n) = A \cos(-0.4\pi n), \quad (11.53)$$

$$x_5[n] = A \cos(2\pi f_5 n / f_s) = A \cos(1.8\pi n) = A \cos(-0.2\pi n). \quad (11.54)$$

The frequencies f_3 , f_4 , and f_5 have the following aliases within the principal spectrum:

$$\hat{\omega}_3 = \mp 0.6\pi \quad (11.55)$$

$$\hat{\omega}_4 = \mp 0.4\pi \quad (11.56)$$

$$\hat{\omega}_5 = \mp 0.2\pi. \quad (11.57)$$

If we inspect the ordering of the frequencies, we can see that $|f_0| < |f_1| < |f_2|$. This agrees with $|\hat{\omega}_0| < |\hat{\omega}_1| < |\hat{\omega}_2|$. However, when we compare the ordering $|\hat{f}_3| < |\hat{f}_4| < |\hat{f}_5|$, we can see that the order is reversed with respect to $|\hat{\omega}_5| < |\hat{\omega}_4| < |\hat{\omega}_3|$. This is due to folding.

Nyquist oversampling criterion

Let us assume that our continuous-time signal has non-zero spectral components with frequencies only within the range $|f| < \frac{1}{2}f_s$. This means that it has a Fourier transform representation of the form:

$$x(t) = \frac{1}{2\pi} \int_{-\pi f_s}^{\pi f_s} \hat{x}(\omega) e^{i\omega t} d\omega. \quad (11.58)$$

When such a signal is discretized, all the discrete-time spectral components of the signal land on the principal spectrum between $-\pi < \hat{\omega} < \pi$.

The band limited nature of the continuous-time signal also guarantees that no high frequency components will have low frequency aliases in the principal spectrum. In this case, one can be certain that there is a one-to-one mapping between discrete-time frequency and continuous-time frequency. And therefore, no information is lost when discretizing the signal. In this case, the signal is said to be *oversampled*.

To avoid aliasing of spectral component frequencies $f > f_s/2$ into the principal spectrum, the sample rate f_s has to be at least twice the frequency of the highest frequency component of the signal:

$$f_s > 2f_{\max}. \quad (11.59)$$

It can be also expressed as

$$f_{\max} < \frac{1}{2}f_s. \quad (11.60)$$

This is called the *Nyquist oversampling criterion*. It is a special case of the more general Shannon-Nyquist sampling theorem. If this criterion is not satisfied, then the signal is said to be *undersampled*.

When deriving the Shannon-Nyquist sampling theorem, we will see that it is in some cases possible to retain information even when the signal is undersampled. Undersampling is a technique that is often used for sampling radio signals.

FOR EXAMPLE, the Nyquist frequency f_{\max} of a real-valued signal sampled at $f_s = 44.1$ kHz is $f_{\max} = 22.05$ kHz. The sample rate 44.1 kHz is often used when digitizing audio, because audio signals within the human hearing range are between $0 < f < 20$ kHz and signals conveying audio signals only contain spectral components within this range.

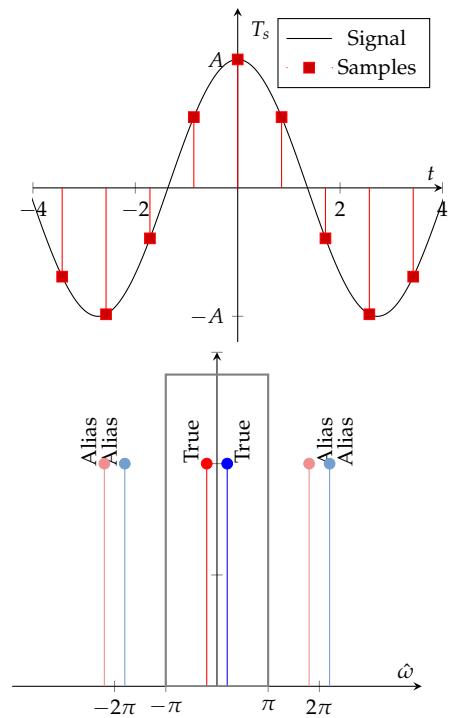


Figure 11.11: Oversampling $f_s > 2f_0$.

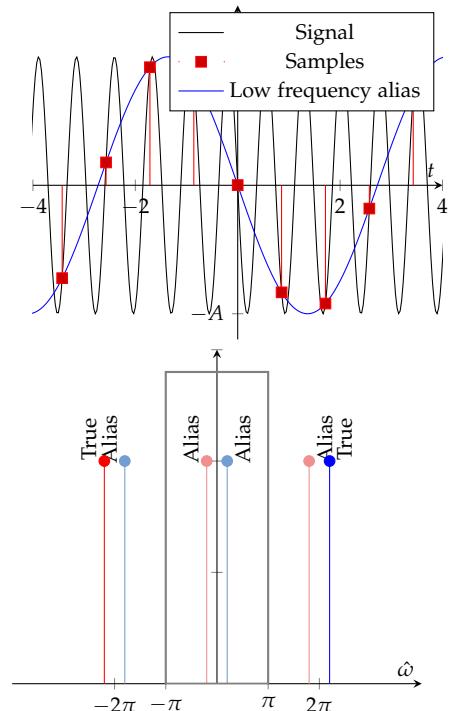


Figure 11.12: Undersampling $f_s < f_0$. When the sample rate is smaller than the frequency of the sinusoid there is less than one sample per sinusoid cycle. The discrete-time frequency $\hat{\omega} > 2\pi$ and a low-frequency alias of the high frequency sinusoid exists in the principal spectrum.

Nyquist zones

If spectral components of a real-valued continuous-time signal are only located between $k f_s / 2 \leq |f| < (k + 1) f_s / 2$ for only one value of k and if there are no non-zero spectral components elsewhere, then one can still recover the continuous-time signal from the discrete-time representation of the signal, using the a priori information that the signals are within this range of frequencies. This is the more general sampling criterion for real-valued signals.

These regions, which are shown in Figure 11.13, are called Nyquist zones. In each of these cases, all the low frequency aliases of the signal will be confined with the principal spectrum $\pm\pi$. Because no signals from outside the specific Nyquist zone are present, there is no chance of two continuous-time spectral components mapping to the same normalized angular frequency within the principal spectrum.

When $k = 0$, this case is called an *oversampled* signal. This is the special case that the Nyquist oversampling criterion applies to. When $k = 1$, the signal sampling is called *folded undersampled* signal. In this case, the aliased signal in the principal spectrum is flipped in frequency and phase relative to the continuous-time spectrum. When $k = 2$, the signal is *undersampled*. These different cases are shown in Figures 11.11, 11.14, and 11.12.

There are many cases, especially with modern software defined radios, where undersampling or folded undersampling at Nyquist zones $k > 0$ are used. This violates the simple $f_s > 0.5f_{\max}$ Nyquist oversampling criterion, but is still perfectly fine, as long as the original continuous-time signal has spectral components confined within a Nyquist zone.

Complex-valued signals

For complex-valued signals, the *principal spectrum* is the same as it is for real-valued signals. In discrete-time angular frequency it is $-\pi < \hat{\omega} < \pi$ and in continuous-time frequency $-\frac{1}{2}f_s < f < \frac{1}{2}f_s$. However, because there are no conjugate symmetric negative frequency pairs, both negative and positive frequencies can be used in this frequency band. Therefore, the effective bandwidth is twice that of real-valued signals sampled at the sample rate f_s , as both positive and negative frequency components can be used independently to encode information.

To ensure a one-to-one relationship between the discrete-time signal and the continuous-time signal, the sample rate f_s has to be at least the difference between the largest and smallest frequency component

$$f_s > f_{\max} - f_{\min} . \quad (11.61)$$

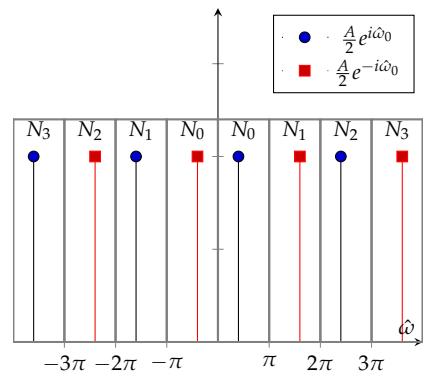


Figure 11.13: Nyquist zones.

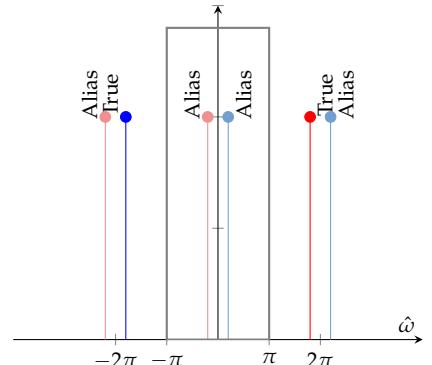
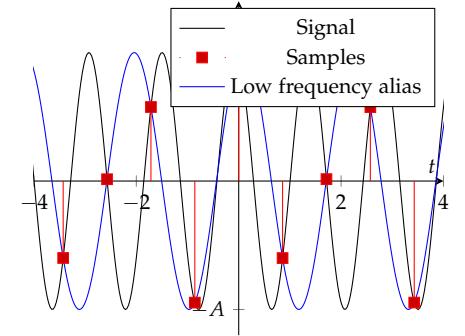


Figure 11.14: Folded undersampling $f_0 < f_s < 2f_0$. There are between 1 and 2 samples for each cycle of the continuous-time sinusoid. The low frequency alias is phase flipped.

if the frequency domain representation of the complex-valued signal is within a continuous range of frequencies between f_{\min} and f_{\max} . This avoids having two continuous-time spectral components with different frequencies aliasing onto the same discrete-time frequency within the principal spectrum³. This is another form of the Shannon-Nyquist sampling criterion, which applies only to complex-valued signals.

Figure 11.15 below illustrates aliasing with complex sinusoidal signals. Here, a complex sinusoidal signal with frequency $1.2\frac{f_s}{2}$ that is sampled with frequency f_s aliases to $-0.8\frac{f_s}{2}$ in the principal spectrum.

AN EXAMPLE OF ALIASING OF TWO COMPLEX SINUSOIDAL SIGNALS is shown below. If a continuous-time signal consists of two complex sinusoidal signals:

$$z(t) = a_1 e^{i2\pi f_1 t} + a_2 e^{i2\pi f_2 t}, \quad (11.62)$$

with frequencies $f_1 = 1.2f_s/2$ and $f_2 = 0.8f_s/2$.

When this signal is sampled with a sample-rate of f_s , the Nyquist oversampling criterion for real-valued signals is violated, because $f_1 > f_s/2$. However, this is a complex-valued signal, and we only need $f_{\max} - f_{\min} < f_s$ to hold in the more general case for complex-valued signals.

This condition is not violated. To see that everything is fine, we translate both of the sinusoidal signals into the principal spectrum:

$$z[n] = a_1 e^{i1.2\pi n} + a_2 e^{i0.8\pi n} \quad (11.63)$$

$$= a_1 e^{-i0.8\pi n} + a_2 e^{i0.8\pi n}. \quad (11.64)$$

We see that the signals land at $\hat{\omega} = \pm 0.8\pi$ in the principal spectrum. The two spectral components are distinct, and if we have sufficient information about the lowest and highest frequency component of the signal (i.e., we know f_{\min} and f_{\max}), we can still perfectly reconstruct the signal.

2D aliasing

Consider a two-dimensional signal $I(x, y) \in \mathbb{R}$, which is discretized as follows:

$$I[n, m] = I(nT_x, mT_y). \quad (11.65)$$

Here T_x and T_y are sample spacing in the x and y direction, and n , and m are sample indices in the two dimensions.

³ The most general sampling criterion involves investigating the relationship between the frequency domain representation of the continuous-time signal and the frequency domain representation of the discretized signal, which we will discuss later when deriving the Shannon sampling theorem.

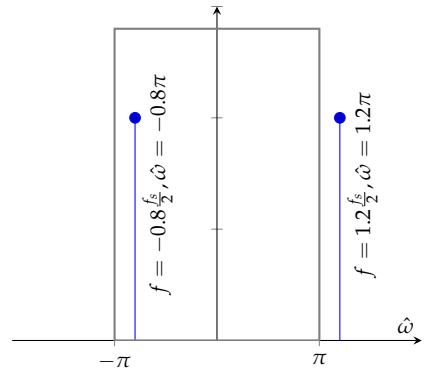


Figure 11.15: Aliasing of a complex sinusoidal spectral component into the principal spectrum. No conjugate symmetric negative frequency spectral components complicate the aliasing of the signal into the principal spectrum.

A 2D signal has a spectral representation, which is given by 2D complex exponential signals:

$$I(x, y) = \frac{1}{(2\pi)^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \hat{I}(\omega_1, \omega_2) e^{i(\omega_1 x + \omega_2 y)} d\omega_1 d\omega_2 . \quad (11.66)$$

To study aliasing behavior in 2D, it is sufficient to study aliasing effects on individual spectral components:

$$z(x, y) = ae^{i(\omega_1 x + \omega_2 y)} , \quad (11.67)$$

When discretizing the 2D complex exponential spectral component, one can obtain aliasing behavior when sampling through 2π -periodicity:

$$z[n, m] = ae^{i(\omega_1 T_x n + \omega_2 T_y m)} \quad (11.68)$$

$$= ae^{i([\omega_1 T_x + 2\pi k]n + [\omega_2 T_y + 2\pi \ell]m)} \quad (11.69)$$

$$= ae^{i([\hat{\omega}_1 + 2\pi k]n + [\hat{\omega}_2 + 2\pi \ell]m)} . \quad (11.70)$$

Here $k, l \in \mathbb{Z}$.

Figure 11.16 demonstrates aliasing in 2D. This is discretized in such a way that the high spatial frequency brick pattern on the side of the building becomes undersampled, and a low frequency alias is seen.

Reconstruction

Reconstruction involves transforming a discrete-time signal into a continuous-time signal. This type of operation is done, e.g., when translating an audio signal into air pressure variations to create sound.

In order to convert a discrete-time signal into a continuous-time signal, some form of interpolation is needed in order to fill in the space between the samples. This is the function of a reconstruction filter, or a discrete-time to continuous-time (D-to-C) system:

$$x(t) = \mathcal{R}\{x[n]\} . \quad (11.71)$$

The ingredients that are needed to reconstruct the signal are the samples $x[n]$, information about the sampling rate $T_s = 1/f_s$, and information about the Nyquist zone that the signal is within.

This is equivalent to knowing the true values of the discrete-time frequencies $\hat{\omega}$ of all the discretized spectral components, i.e., being able to rule out all the aliases using a priori knowledge of the frequency band that the signal lies in.

There are several ways to perform this reconstruction. If the analytical form of the signal is known, as in the case of a sinusoid,

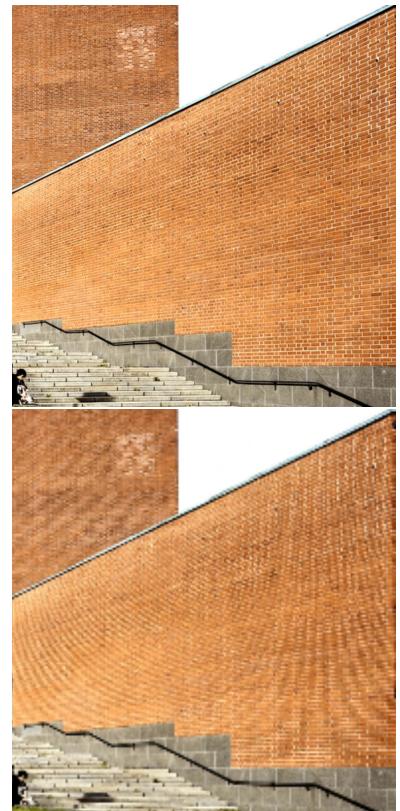


Figure 11.16: Example of aliasing behavior in a 2D image. Above: original, Below: aliased. When scaling an image (reducing its size), it is important to apply an anti-aliasing filter that removes high frequency spectral components from the high resolution image before it is scaled down to a lower resolution. Otherwise, there is a risk that high frequency periodic structures will appear as low frequency structures in the scaled image, as shown in the lower image. Anti-aliasing filtering is a standard feature of most image processing libraries, and one rarely sees the type of aliasing shown in the bottom figure in practice.

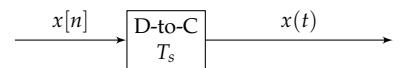


Figure 11.17: A Discrete-time to Continuous-time (D-to-C) converter.

the signal can be generated analytically based on the phase and amplitude of the signal. In other cases, some form of interpolation is needed.

Interpolation filters

In this section, we'll only cover reconstruction of oversampled signals⁴ that satisfy the Nyquist-Shannon sampling criterion $f_s > 2f_{\max}$, i.e., the spectral components of the signal are between $-f_s/2 < f < f_s/2$.

We can convert a discrete-time signal into a continuous-time signal using an interpolation filter of the form:

$$x(t) = \sum_{n=-\infty}^{\infty} x[n]p(t - nT_s). \quad (11.72)$$

Theoretically, this interpolation filter can use the discrete-time samples to obtain the continuous-time reconstruction.

THE ZERO-ORDER HOLD FILTER is the simplest interpolation filter. It obtains a constant value for the duration of a sample. The filter in this case has the following mathematical definition:

$$p(t) = \begin{cases} 1 & \text{when } -\frac{T_s}{2} < t \leq \frac{T_s}{2} \\ 0 & \text{otherwise} \end{cases}. \quad (11.73)$$

The generated continuous-time signal looks like the one shown in Figure 11.18.

The zero-order hold filter only needs one sample at a time. However, the continuous-time signal has large steps. The zero-order hold filter approaches the continuous-time signal only when $T_s \rightarrow 0$ – i.e., when the sample rate is infinitely large.

A **LINEAR INTERPOLATION FILTER** is defined as follows:

$$p(t) = \begin{cases} 1 - \frac{|t|}{T_s} & \text{when } -T_s < t < T_s \\ 0 & \text{otherwise} \end{cases}. \quad (11.74)$$

It applies a linear interpolation between two consecutive samples. The generated continuous-time signal is depicted in Figure 11.19.

THE IDEAL RECONSTRUCTION FILTER is based on the Shannon-Nyquist sampling theorem. It guarantees theoretically that a continuous-time real-valued signal can be perfectly reconstructed from discrete-time samples for signals occupying the band $-f_s/2 < f < f_s/2$, if $f_s > 2f_{\max}$. Here f_{\max} is the maximum frequency component within the signal. We'll prove this theorem at the end of this chapter.

⁴ It is possible to derive interpolation filters for reconstruction of undersampled signals in the same way, but this is beyond the scope of this course.

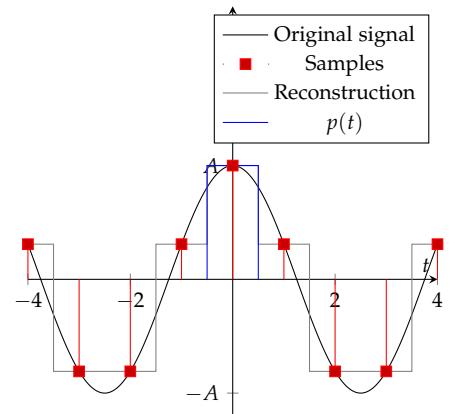


Figure 11.18: Zero-order hold interpolation.

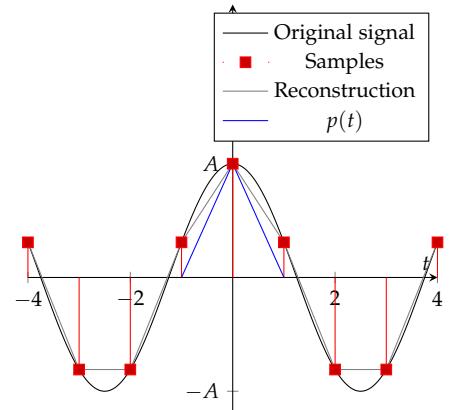


Figure 11.19: Linear interpolation filter.

In order to reconstruct the signal perfectly in a mathematical sense, an ideal reconstruction filter must be used. Something approximately resembling a truncated or tapered version of an ideal reconstruction filter is typically used in real world applications.

The form of this ideal reconstruction filter can be derived from the inverse Fourier transform of a rectangular shaped window that in frequency domain is an ideal low pass filter:

$$P(\omega) = \begin{cases} T_s & \text{when } |\omega| \leq \pi f_s \\ 0 & \text{otherwise} \end{cases}. \quad (11.75)$$

Here $P(\omega)$ is the Fourier transform of the reconstruction filter, ω is angular frequency. The resulting “ideal” reconstruction filter is:

$$p(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} P(\omega) e^{i\omega t} d\omega \quad (11.76)$$

$$= \frac{T_s}{\pi t} \sin\left(\frac{\pi}{T_s} t\right). \quad (11.77)$$

This filter is infinitely long, which implies that all samples need to be used in order for the reconstructed signal to be mathematically perfect.

Shannon's sampling theorem

Shannon's sampling theorem is a fundamental theorem in information theory and signal processing⁵. It relates the sample-rate of a discrete-time signal to the bandwidth of a continuous-time signal in frequency domain. The full form of the theory is not just: “use a sample-rate that is at least twice as large as the largest frequency component in the continuous-time signal”. There are other solutions, which also guarantee that no information about the signal is lost. The most general way to investigate if information is lost when discretizing a signal is to explore the relationship between the Fourier transform of the signal, and the Fourier transform of the discretized signal.

We will assume that information is contained in the shape of the signal $x(t)$. This signal is assumed to be band-limited in frequency domain. This means that the spectrum is zero valued outside a certain range of frequencies: $\hat{x}(\omega) = 0$ when $\omega < \omega_{\min}$ or $\omega > \omega_{\max}$.

Is it possible to form a discrete-time signal $x[n]$ from $x(t)$ in such a way that $x[n]$ contains sufficient information to allow us to perfectly reconstruct $x(t)$? In other words, can we perfectly reconstruct a continuous-time signal $x(t)$ from its discrete-time representation? This is the question that Shannon studied, and his sampling theorem provides an answer to.

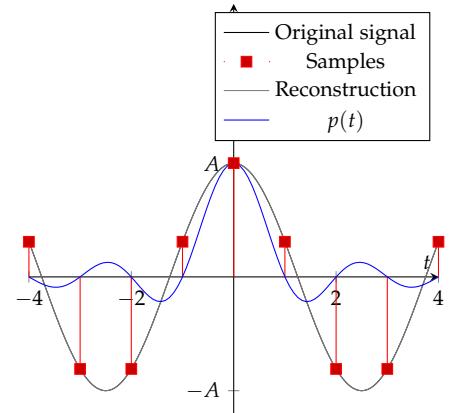


Figure 11.20: Ideal reconstruction filter.



Figure 11.21: Claude Elwood Shannon.
Photo: Bell Labs.

⁵ This theorem was discovered by Edmund Whittaker 33 years before Claude Shannon, and thus sometimes the sampling theory is called the Whittaker-Shannon sampling theorem.

Edmund Taylor Whittaker. On the functions which are represented by the expansions of the interpolation-theory. *Proceedings of the Royal Society of Edinburgh*, 35:181–194, 1915; and Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948

Sampling

In the idealized case, sampling of a signal is selection of samples from a continuous-time signal:

$$x[n] = x(nT_s) \quad (11.78)$$

where T_s is the sample spacing. This is the inverse of the sample-rate $T_s = f_s^{-1}$. We can express this sampling process using the Dirac delta function, where a delayed Dirac delta “samples” the value of the signal $x(t)$ at time nT_s :

$$x[n] = \int_{-\infty}^{\infty} x(t)\delta(t - nT_s)dt = x(nT_s). \quad (11.79)$$

We can use a train of Dirac delta functions (Dirac comb)

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s), \quad (11.80)$$

to represent the sampled signal in continuous-time:

$$x_s(t) = x(t)s(t) \quad (11.81)$$

$$= x(t) \sum_{n=-\infty}^{\infty} \delta(t - nT_s) \quad (11.82)$$

$$= \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_s) \quad (11.83)$$

The sampled signal $x_s(t)$ is a continuous-time signal, which has the same information content as the array of samples $x[n]$. This means that $x_s(t)$ can be formed just by knowledge of the discrete-time signal $x[n]$.

Fourier series representation for Dirac comb

The Dirac comb

$$s(t) = \sum_{k=-\infty}^{\infty} \delta(t - kT_s), \quad (11.84)$$

is a periodic function with period T_s . Therefore, it is possible to express it as a Fourier series:

$$s(t) = \sum_{k=-\infty}^{\infty} c_k e^{j\frac{2\pi}{T_s}kt}. \quad (11.85)$$

The coefficients of the Fourier series can be obtained using the Fourier series synthesis equation, by integrating over one period



Figure 11.22: Sir Eduard Whittaker.
Photo: National Portrait Gallery.

of the function:

$$c_k = \frac{1}{T_s} \int_{-T_s/2}^{T_s/2} \left[\sum_{n=-\infty}^{\infty} \delta(t - nT_s) \right] e^{-i\frac{2\pi}{T_s} kt} dt \quad (11.86)$$

$$= \frac{1}{T_s} \int_{-T_s/2}^{T_s/2} \delta(t) e^{-i\frac{2\pi}{T_s} kt} dt \quad (11.87)$$

$$= \frac{1}{T_s} e^{-i \cdot 0} \quad (11.88)$$

$$= \frac{1}{T_s} \quad (11.89)$$

We also note that $2\pi/T_s = \omega_s$, which is the sample-rate expressed as an angular frequency. The Fourier series representation of the Dirac-comb is therefore:

$$s(t) = \frac{1}{T_s} \sum_{k=-\infty}^{\infty} e^{ik\omega_s t}. \quad (11.90)$$

This Fourier series representation can now be substituted to represent the sampling operation as:

$$x_s(t) = x(t)s(t) \quad (11.91)$$

$$= x(t) \left(\frac{1}{T_s} \sum_{k=-\infty}^{\infty} e^{ik\omega_s t} \right) \quad (11.92)$$

Spectral representation of $x_s(t)$ and $x(t)$

We then Fourier transform $x_s(t)$ to study the relationship between the sample-rate and the bandwidth of the signal. We use the property that multiplication in time domain is convolution in frequency domain. The Fourier transform of $x(t)$ is $\hat{x}(\omega)$. The Fourier transform of $s(t)$ is $\hat{s}(\omega)$. We also make use of the fact that the Fourier transform of $e^{ik\omega_s t}$ is $2\pi\delta(\omega - k\omega_s)$:

$$\hat{x}_s(\omega) = \frac{1}{2\pi} \hat{x}(\omega) * \hat{s}(\omega) \quad (11.93)$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega') \hat{s}(\omega - \omega') d\omega' \quad (11.94)$$

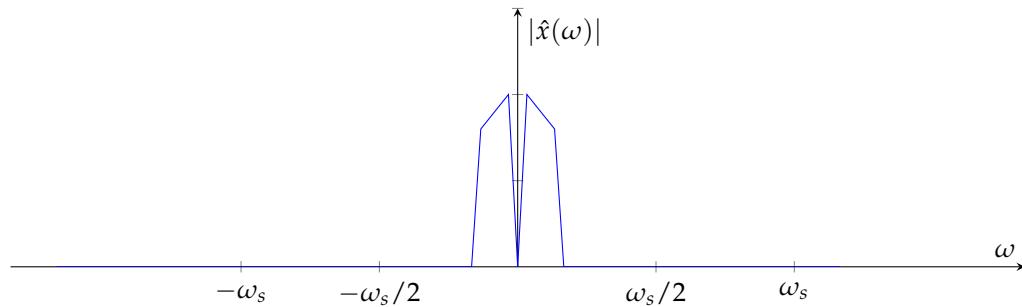
$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega') \left(\frac{1}{T_s} \sum_{k=-\infty}^{\infty} 2\pi\delta(\omega - k\omega_s - \omega') \right) d\omega' \quad (11.95)$$

$$= \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} \hat{x}(\omega') \delta(\omega - k\omega_s - \omega') d\omega' \quad (11.96)$$

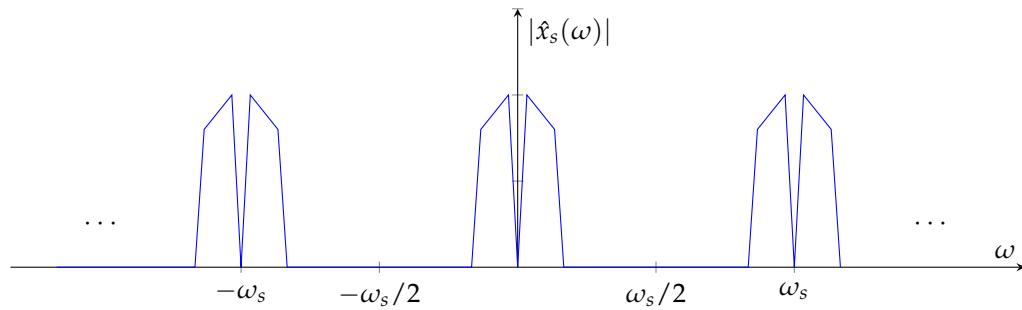
$$= \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \hat{x}(\omega - k\omega_s). \quad (11.97)$$

In other words, the Fourier transform of the sampled function $x_s(t)$ is infinitely many copies of $\hat{x}(\omega)$, each shifted by $k\omega_s$ and scaled in amplitude by $\frac{1}{T_s}$.

We illustrate this using the following two figures. The first one depicts the magnitude spectrum of the original continuous-time signal $x(t)$:



The sampled signal $x_s(t)$ has a periodic spectrum $\hat{x}_s(\omega) = \hat{x}_s(\omega + k\omega_s)$, which has a period ω_s :



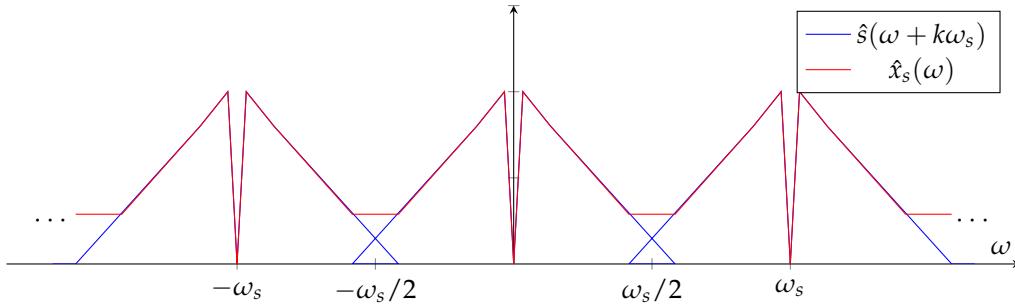
This probably already provides some idea of what the sample-rate should be, in order to avoid losing information. The most straightforward answer is that if the spectrum of the original signal $\hat{x}(\omega)$ is confined within the limits of $\pm\omega_s/2$, then no overlap between the different copies of $\hat{x}(\omega)$ occurs.

There are other solutions that avoid aliasing for real-valued signals. The original signal $\hat{x}(\omega)$ might also be confined in frequency to one of the intervals $n\omega_s/2 \leq |\omega| < (n+1)\omega_s/2$ with $n \in \mathbb{Z}$. This property is widely used when sampling radio signals that are at a much higher frequency than the sample rate. This type of sampling strategy is called undersampling.

Aliasing

If the different copies of $\hat{x}(\omega)$ in $\hat{x}_s(\omega)$ overlap, then aliasing occurs. For example, if the spectrum of $\hat{x}(\omega)$ extends over the limits $\pm\omega_s/2$, then there are regions where $\hat{x}_s(\omega)$ is occupied by two or more spectral components of $\hat{x}(\omega)$ at the same time.

If the frequency extent of the original signal $\hat{x}(\omega)$ is larger than ω_s , then aliasing occurs as shown below:

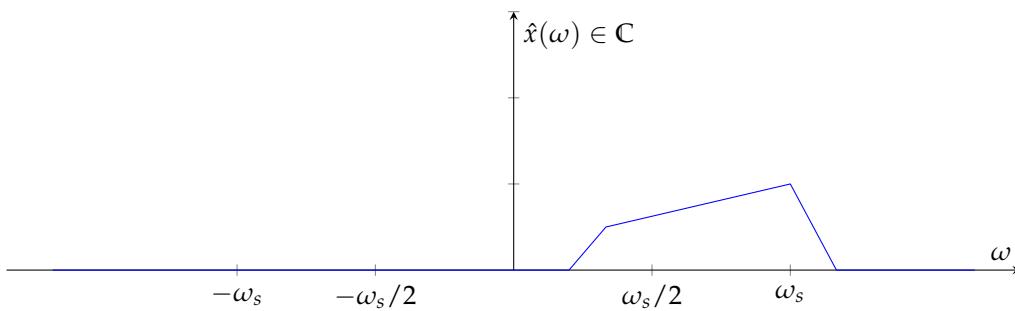


In this case, positive and negative frequency components of the original spectrum $\hat{x}_s(\omega)$ overlap and are added together when $|\omega| > \omega_s/2$, making it impossible to determine what the original value of $\hat{x}(\omega)$ is within the overlapping region.

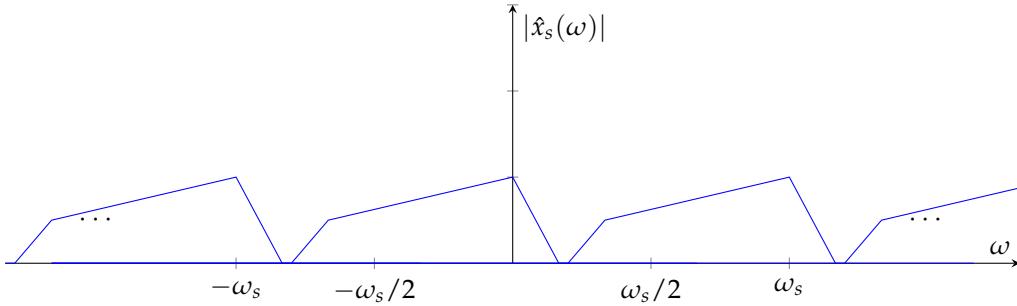
Complex-valued signal

For a complex-valued signal $x(t) \in \mathbb{C}$, a sufficient condition for avoiding $\hat{x}(\omega)$ overlapping with itself in $\hat{x}_s(\omega)$ (aliasing) is that $\omega_s > \omega_{\max} - \omega_{\min}$.

Aliasing with complex signals is illustrated with the following plots. Consider a signal, which spans between frequencies $0.11\omega_s$ and $1.1\omega_s$. In this case, the signal crosses $\omega_s/2$, which violates aliasing rules for real-valued signals. However, $\omega_s > \omega_{\max} - \omega_{\min}$.



The spectrum of the sampled signal $x_s(t)$ has a periodic spectrum $\hat{x}_s(\omega) = \hat{x}_s(\omega + k\omega_s)$. Notice that there is no overlap between the copies of the spectrum:



This means that the values of $\hat{x}_s(\omega)$ between, e.g., $0.11\omega_s$ and $1.1\omega_s$ fully determine the frequency domain representation of the original signal $\hat{x}(\omega)$.

Sampling criteria

The Shannon-Nyquist sampling theorem requires that it should be possible to reconstruct $\hat{x}(\omega)$ from $\hat{x}_s(\omega)$. The commonly described criterion that requires the sample-rate to be higher than twice the largest frequency component of the signal:

$$f_s > 2f_{\max} \quad (11.98)$$

is just one possible solution, which assumes that the spectral components within the original signal $\hat{x}(\omega)$ are confined to $|f| < f_s/2$. As we discussed above, complex-valued signals and under sampled signals result in different sampling criteria.

Reconstruction

This reconstruction strategy only applies to a real-valued signal $x(t) \in \mathbb{R}$ within the principal spectrum. It should be easy to determine how to create a reconstruction filter for, e.g., a complex-valued signal occupying the band $\omega \in [\omega_{\min}, \omega_{\max}]$ after following this derivation.

In order to reconstruct signal $x(t)$ from $x[n]$, we first form $x_s(t)$ using $x[n]$:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n] \delta(t - nT_s) \quad (11.99)$$

We then use an ideal low-pass filter specified in frequency domain as:

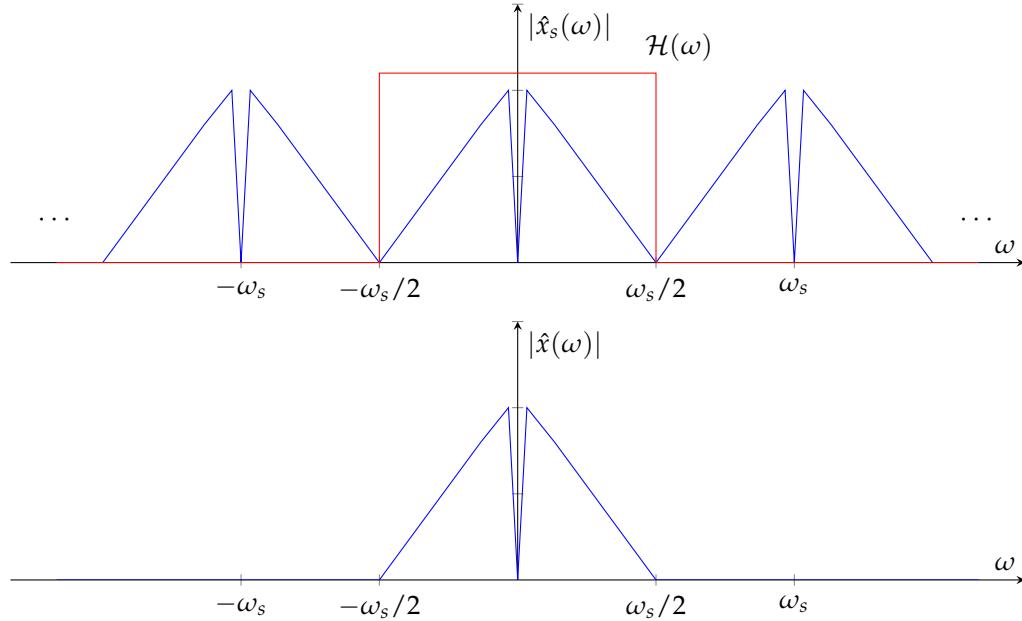
$$\mathcal{H}(\omega) = \begin{cases} T_s & |\omega| \leq \frac{\omega_s}{2} \\ 0 & \text{otherwise} \end{cases} \quad (11.100)$$

When we apply this filter $\hat{x}_s(\omega)$ to reconstruct $\hat{x}(\omega)$:

$$\mathcal{H}(\omega)\hat{x}_s(\omega) = \mathcal{H}(\omega)\frac{1}{T_s} \sum_{k=-\infty}^{\infty} \hat{x}(\omega - k\omega_s) \quad (11.101)$$

$$= \hat{x}(\omega) \quad \square \quad (11.102)$$

This operation is shown in the Figure below:



This is in essence the proof of Shannon's sampling theorem. To obtain $x(t)$, one can use an inverse Fourier transform:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega. \quad (11.103)$$

If $\hat{x}_s(\omega)$ contains overlapping copies of $\hat{x}(\omega)$, then the original spectrum $\hat{x}(\omega)$, and therefore the original signal $x(t)$, cannot be reconstructed.

For a complex-valued signal, we would need to apply an ideal band-pass filter, which only retains signals between ω_{\min} and ω_{\max} , instead of the ideal low-pass filter. The same applies to an undersampled signal.

Ideal reconstruction filter

Recall, from the beginning of this course, that we introduced the ideal reconstruction filter without deriving it. Now we can derive it. It is the inverse Fourier transform of the ideal low-pass filter

$$\mathcal{H}(\omega) = \begin{cases} T_s & |\omega| \leq \frac{\omega_s}{2} \\ 0 & \text{otherwise} \end{cases} \quad (11.104)$$

which is

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \mathcal{H}(\omega) e^{i\omega t} d\omega \quad (11.105)$$

$$= \frac{1}{2\pi} \int_{-\omega_s/2}^{\omega_s/2} T_s e^{i\omega t} d\omega \quad (11.106)$$

$$= \frac{T_s}{2\pi} \left. \frac{e^{i\omega t}}{it} \right|_{\omega=-\omega_s/2}^{\omega_s/2} \quad (11.107)$$

$$= \frac{T_s}{2t\pi i} \left(e^{i\frac{\omega_s}{2}t} - e^{-i\frac{\omega_s}{2}t} \right) \quad (11.108)$$

$$= \frac{T_s}{\pi t} \sin\left(\frac{\omega_s}{2}t\right) \quad (11.109)$$

Keeping in mind that $\omega_s/2 = \pi/T_s$ we obtain the familiar result:

$$h(t) = \frac{T_s}{\pi t} \sin\left(\frac{\pi}{T_s}t\right) \quad (11.110)$$

When applying this reconstruction filter in time domain, one convolves the sampled signal to obtain the reconstructed continuous-time signal:

$$x(t) = h(t) * x_s(t) \quad (11.111)$$

$$= \int_{-\infty}^{\infty} x_s(\tau) h(t - \tau) d\tau \quad (11.112)$$

$$= \int_{-\infty}^{\infty} \left(\sum_{n=-\infty}^{\infty} x[n] \delta(\tau - nT_s) \right) h(t - \tau) d\tau \quad (11.113)$$

$$= \sum_{n=-\infty}^{\infty} x[n] \int_{-\infty}^{\infty} \delta(\tau - nT_s) h(t - \tau) d\tau \quad (11.114)$$

$$= \sum_{n=-\infty}^{\infty} x[n] \frac{\sin\left(\frac{\pi}{T_s}(t - nT_s)\right)}{\frac{\pi}{T_s}(t - nT_s)}. \quad (11.115)$$

This reconstruction formula now perfectly reproduces $x(t)$ from samples $x[n]$, provided that $\omega_s \geq 2\omega_{\max}$.

For an undersampled real-valued signal, or a complex-valued band limited signal, the ideal reconstruction filter would be different. In this case, it would correspond to the impulse response of an ideal band-pass filter.

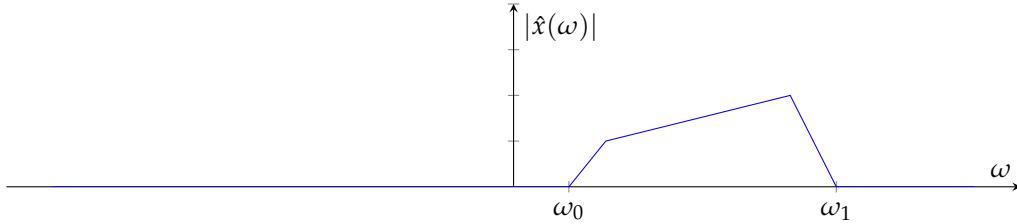
Exercises: Discrete-time Signals

1. The signal $x(t) = Ae^{i\omega_0 t}$ is discretized using an idealized continuous-to-discrete time converter $x[n] = x(nT_s) = Ae^{i\hat{\omega}_k n}$. Here $A = 1$ and $\omega_0 = 2\pi 10042$ radians per second. The sample rate is $f_s = 100$ samples per second and $T_s = 1/f_s$.
 - a) What are all the possible values of $\hat{\omega}_k$ that result in an identical discrete-time signal $x[n]$. Use normalized angular frequency with units of radians per sample. The possible values $\hat{\omega}_k$ are called frequency aliases. Here $k \in \mathbb{Z}$ is an integer index to the different solutions.
 - b) What are the principal aliases in normalized angular frequency with units radians per sample? Principal aliases are values of $\hat{\omega}_k$ that lie in the interval $-\pi < \hat{\omega}_k < \pi$.
 - c) Draw the spectrum of this signal with normalized angular frequency in units of radians per sample on the horizontal axis. Use the range $-3\pi < \hat{\omega} < 3\pi$. How many frequency components are there?
 - d) There are infinitely many continuous-time signals $x_k(t) = Ae^{i\omega_k t}$ that will result in the same $x[n]$. What are all the possible continuous-time angular frequencies ω_k ? What is the smallest possible absolute value of $|\omega_k|$?
 - e) What is the smallest possible sample-rate f_s at which the signal $x(t)$ can be sampled at, while still retaining enough information to allow the signal to be reconstructed?
2. Given the discrete-time sinusoid $y_1[n] = 2 \cos(0.67\pi n) + \cos(0.33\pi n)$.
 - a) Draw the spectrum for $y_1[n]$. Show aliases in the normalized angular frequency range $-3\pi < \hat{\omega} < 3\pi$ (radians per sample). Label the principal spectrum showing the phase, amplitude and normalized angular frequency of each frequency component.
 - b) Show how the spectrum changes if the signal changes to $y_2[n] = 2 \cos(2.67\pi n) + \cos(0.33\pi n)$. Use the same frequency range as in a).
3. If you've ever seen an old western movie with a stagecoach wheel seemingly rotating in the wrong direction, this exercise will help you understand how this optical illusion occurs. You have done a video recording of a disc with a red mark on the rim. The disc started a clockwise rotation with slowly increasing rotational speed $0 \leq v_{rot} \leq 2880$ rotations per minute (rpm). The sampling rate of your camera was $f_s = 24$ frames per second (fps). Assume

that each frame is captured instantaneously, i.e., the exposure time is infinitely short. On the recording, you can see that the red spot seems to rotate differently for certain speeds and speed ranges.

- a) At what rotational speed (in rpm) does the red spot appear to be standing still?
 - b) Around a certain speed, the red spot appears to start to rotate counter-clockwise, and for what speed range will this phenomenon occur?
 - c) Do you know another word for the counter-clockwise behavior?
 - d) What behavior does the red spot have beyond the speed found in a)?
4. The absolute values of the frequency domain representation of a signal is shown in the figure below. We know that $\omega_0 < \omega_1$ and:

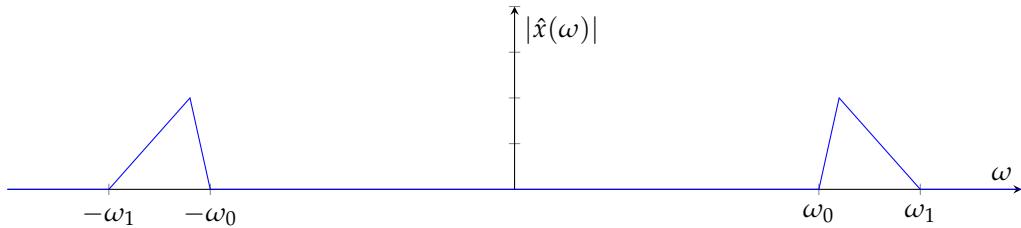
$$|\hat{x}(\omega)| = \begin{cases} 0 & \text{when } \omega \geq \omega_1 \\ 0 & \text{when } \omega \leq \omega_0 \\ > 0 & \text{otherwise} \end{cases}, \quad (11.116)$$



- a) $x(t) = (2\pi)^{-1} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega$. Why is $x(t)$ not a real-valued signal?
 - b) What is the minimum sample-rate (in units of samples per second) required to retain all information about the signal?
 - c) Sketch a plot of $|\hat{x}_s(\omega)|$ for the sample rate you found in b). $\hat{x}_s(\omega)$ is defined in Equation 11.93. Why don't the frequency shifted copies of $|\hat{x}(\omega)|$ overlap?
 - d) What reconstruction filter would be needed in order to perfectly reconstruct $x(t)$ from the discretized signal $x[n]$, using the sample-rate that you found?
5. The absolute values of the frequency domain representation of a signal is shown in the figure below. We know that $\hat{x}(\omega) = \hat{x}^*(-\omega)$ and that:

$$|\hat{x}(\omega)| = \begin{cases} > 0 & \text{when } \omega_0 < |\omega| < \omega_1 \\ 0 & \text{otherwise} \end{cases}, \quad (11.117)$$

We also know that $\omega_0 = 2\pi 30$ and $\omega_1 = 2\pi 40$ (radians per second). This signal is undersampled using a sample-rate of $f_s = 20$ hertz (samples per second).



- a) $x(t) = (2\pi)^{-1} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega$. Is the signal $x(t)$ a real-valued signal?
- b) Sketch a plot of $|\hat{x}_s(\omega)|$. Recall that $\hat{x}_s(\omega)$ is defined in Equation 11.93. Why don't the frequency shifted copies of $|\hat{x}(\omega)|$ overlap?
- c) The sample rate $f_s = 20$ hertz is sufficient for retaining all information about $x(t)$. Why?
- d) What is the frequency domain definition of the perfect reconstruction filter that reproduces the original continuous-time signal $x(t)$ from a sampled signal $x[n]$?

Suggested solutions: Discrete-time Signals

1. The signal $x(t) = Ae^{i\omega_0 t}$ is discretized using an ideal continuous-to-discrete time converter $x[n] = x(nT_s) = Ae^{i\hat{\omega}_k n}$. Let $A = 1$ and $\omega_0 = 2\pi 10042$ rad/s. Let the sample rate be $f_s = 100$ samples per second and $T_s = 1/f_s$.

- a) Have $\hat{\omega}_0 = \omega_0 T_s$ and $\hat{\omega}_k$ is:

$$\hat{\omega}_k = \hat{\omega}_0 + 2\pi k = \omega_0 \frac{1}{f_s} + 2\pi k = \frac{5021}{25}\pi + 2\pi k,$$

for $k \in \mathbb{Z}$.

- b) The principal aliases lie in $-\pi < \hat{\omega}_k < \pi$. Pick a $k \in \mathbb{Z}$ such that $-\pi < \hat{\omega}_k < \pi$. For instance, $k = -100$ works:

$$-\pi < \frac{5021}{25}\pi - 2\pi 100 < \pi,$$

giving a value of $\hat{\omega} \approx 2.6389$ rad/sample.

- c) To draw the spectrum for $x(t)$ we can use the code in Listing 11.1. The boundaries for -3π and 3π are shown as vertical lines in green. The principal alias is shown in blue while the other aliases are shown in red. There are a total of three spectral components on this the interval $(-3\pi, 3\pi)$.

```
import matplotlib.pyplot as plt
import numpy as np

fs = 100 # Sample rate in Hz.
omo = 2*np.pi*10042

def omk(k):
    """Function to compute the different aliases."""
    return omo/fs + 2*np.pi*k

# Plot on the interval (-3*np.pi, 3*np.pi).
om = np.linspace(-3*np.pi, 3*np.pi, num=1000)

# Plot the components.
plt.vlines(omk(-100), ymin=0, ymax=1, color="blue", label="Principal value")
plt.vlines(omk(-99), ymin=0, ymax=1, color="red", label="Alias")
plt.vlines(omk(-101), ymin=0, ymax=1, color="red")
plt.vlines(-3*np.pi, ymin=0, ymax=1, color="green", label=r"\$(-3\pi,3\pi)\$")
plt.vlines(3*np.pi, ymin=0, ymax=1, color="green")
plt.xlabel(r"\$\hat{\omega}\$")
plt.ylabel(r"\$x(\hat{\omega})\$")
plt.grid(True)
plt.legend()
plt.show()
```

Listing 11.1: Suggested solution to c)

The spectrum $\hat{x}(\omega)$ is shown in Figure 11.23.

- d) The possible angular frequencies that result in the same discretization is found by:

$$\omega_k = \omega_0 + 2\pi k f_s.$$

Choosing different values for k and search for a value that satisfy the requirement of being the smallest, $|\omega_k|$ is found to be $k = -100$. This value gives $\omega_k = 2\pi 42$.

- e) Here the signal is complex, so the signal can be sampled at $f_s > f_{\max} - f_{\min} = \frac{\omega_0}{2\pi}$ by the Shannon sampling theorem, which correspond to $f_s > 10042$ Hz.
2. Given the continuous-time sinusoid $y_1[n] = 2 \cos(0.67\pi n) + \cos(0.33\pi n)$.
- a) Using Euler's formula to first rewrite $y_1[n]$ on polar form:
- $$y_1[n] = e^{i0.67\pi n} + e^{-i0.67\pi n} + \frac{1}{2}e^{i0.33\pi n} + \frac{1}{2}e^{-i0.33\pi n}$$
- and then plot the spectrum as shown in figure on the side. The blue nodes are all aliases of each other, the same goes for the red ones. The light blue and scarlet colored nodes are folded aliases.
- b) For the changed signal $y_2[n] = 2 \cos(2.67\pi n) + \cos(0.33\pi n)$, the spectrum will stay unchanged because $\cos(2.67\pi n) = \cos(0.67\pi n)$.
3. A sampling rate of 24 fps equals 1440 frames per minute.
- a) When the disc speed equals the camera's sampling rate at 1440 rpm, the red spot will be recorded at the same position for every frame, and thus appears to be standing still.
- b) When the disc speed gets larger than the Shannon-Nyquist sampling theorem, $v_{rot} > f_s/2 > 720$ rpm, aliasing starts to occur. Within the speed range of $720 < v_{rot} < 1440$ rpm, the phase of the alias has the opposite sign than the phase of the real rotation, and so the red spot seems to rotate counter-clockwise with decreasing speed as v_{rot} increases.
- c) Aliasing in the interval with opposite phase sign is called *folding*.
- d) When $1440 < v_{rot} < 2880$, there is no folding, and the corresponding alias of the red spot will have speed $0 < v_{rot} < 1440$ rpm.
4. Let $x(t) = (2\pi)^{-1} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega$.

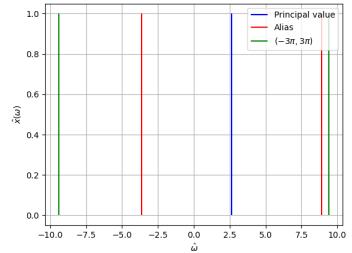
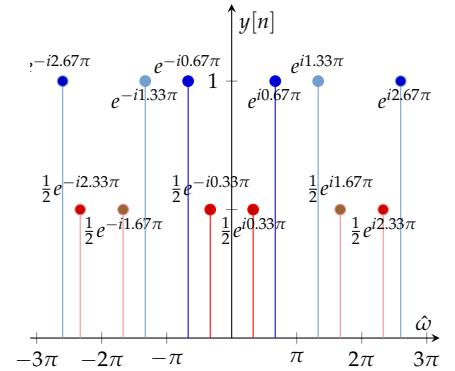
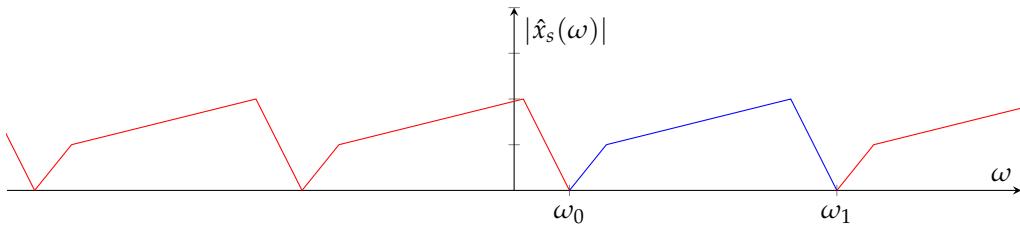


Figure 11.23: Spectrum of $x(t)$, we see 3 spectral components in the range $(-3\pi, 3\pi)$



- a) By the definition of $|\hat{x}(\omega)|$ and the Figure we see that there is no pairing for the frequencies, meaning that $\hat{x}^*(\omega) \neq \hat{x}(-\omega)$, so $x(t)$ can't be real.
- b) For complex signals, we must sample at a rate that satisfy $f_s > f_{\max} - f_{\min} = \frac{1}{2\pi}(\omega_1 - \omega_0)$ in units of Hz.
- c) By the Shannon sampling theorem the sample rate f_s from above is sufficient to retain all the information regarding the signal, thus the spectrum has no overlap. This is reflected in the drawing of $|\hat{x}_s(\omega)|$ where the copies are shown in red.



- d) In this case, the reconstruction filter would be an ideal band-pass filter of the form:

$$\mathcal{H}(\omega) = \begin{cases} T_s, & \omega_1 \leq \omega \leq \omega_2, \\ 0, & \text{otherwise.} \end{cases}$$

Have $x_s(t)$ as:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x[n] \delta(t - nT_s),$$

from the samples. To reconstruct the signal, simply multiply $\hat{x}(\omega)$ by the reconstruction filter in frequency domain. The frequency domain representation of $x_s(t)$ was shown to be:

$$\hat{x}_s(\omega) = \frac{1}{T_s} \sum_{n=-\infty}^{\infty} \hat{x}(\omega - n\omega_s).$$

Where $\omega_s = 2\pi f_s$. Thus, multiplying in frequency domain yields:

$$\mathcal{H}(\omega) \hat{x}_s(\omega) = T_s \frac{1}{T_s} \sum_{n=-\infty}^{\infty} \hat{x}(\omega - n\omega_s) = \hat{x}(\omega),$$

here the sum is killed by $\mathcal{H}(\omega)$ as it is 0 outside $\omega_1 \leq \omega \leq \omega_2$. Hence, we retain all the information necessary to reconstruct the signal as:

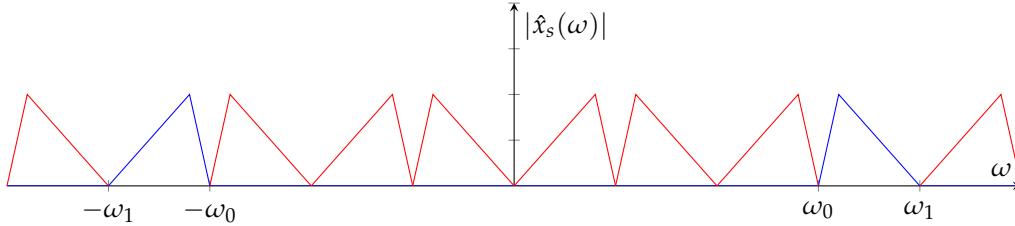
$$x(t) = (2\pi)^{-1} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega.$$

5. As before, let $x(t) = (2\pi)^{-1} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega$ and:

$$|\hat{x}(\omega)| = \begin{cases} > 0 & \text{when } \omega_0 < |\omega| < \omega_1, \\ 0, & \text{otherwise,} \end{cases}$$

with $\omega_0 = 2\pi 30$ and $\omega_1 = 2\pi 40$ radians per second. Here we use a sample rate of $f_s = 20$ Hz or $\omega_s = 2\pi 20$ radians per sample.

- a) We know that $\hat{x}(\omega) = \hat{x}^*(-\omega)$, so the signal is real-valued.
- b) Draw $|\hat{x}_s(\omega)|$ with the different aliases shown in red:



The frequency shifted copies do not overlap due to the sample-rate satisfying the Shannon-sampling theorem, but here the signal is undersampled, as it is confined in another interval than the typical $+/\omega_s/2$. Recall that the goal of sampling is exactly to avoid the shifted copies overlapping, which can be done with the provided sample-rate if we use a different area as to avoid aliasing.

- c) The condition $f_s > 2f_{\max}$ is violated in this case. This condition applies to real signals, but assumes the frequencies fall in the principal spectrum of $|f| < f_s/2$ which is not the case here. However, the sample rate satisfy:

$$n \frac{\omega_s}{2} \leq |\omega| \leq (n+1) \frac{\omega_s}{2},$$

or

$$n 2\pi 10 \leq |\omega| \leq (n+1) 2\pi 10,$$

which is satisfied for $n = 3$, as this gives:

$$2\pi 30 \leq |\omega| \leq 2\pi 40,$$

this bounds the spectrum and avoids aliasing, hence this is sufficient to retain all the information in the signal.

- d) Define a filter $\mathcal{H}(\omega)$ as an ideal band-pass filter:

$$\mathcal{H}(\omega) = \begin{cases} T_s, & \omega_0 \leq |\omega| \leq \omega_1, \\ 0, & \text{otherwise.} \end{cases}$$

Then:

$$\mathcal{H}(\omega) \hat{x}_s(\omega) = T_s \frac{1}{T_s} \sum_{k=-\infty}^{\infty} \hat{x}(\omega - k\omega_s) = \hat{x}(\omega).$$

Due to $\mathcal{H}(\omega)$ being 0 outside $\omega_0 \leq |\omega| \leq \omega_1$ the sum goes away giving back the original frequency representation of the signal. Note that the absolute value is needed here to get both the positive and negative frequencies as the signal is real, then applying the Fourier transform to this signal yields the original signal $x(t)$.

12

Linear Time-invariant Systems

Linear time-invariant systems (LTI) are an important class of systems that can be analyzed easily in frequency domain. An LTI system is equivalent to *convolution* of the *impulse response* of the system. The focus of this chapter is to learn more about what these concepts are.

Example: Running average filter

Here's an example of a discrete-time system that you might use to smooth a noisy signal:

$$y[n] = \frac{1}{15} \sum_{k=-7}^7 x[n - k]. \quad (12.1)$$

What does this system do? It averages together 15 neighboring values of the input signal $x[n]$. Figure 12.1 shows a demonstration of this system in action. The blue line indicates a noisy input signal $x[n]$, and the orange line depicts the output $y[n]$ of the running mean filter given in Equation 12.1. As you might expect, the output of the system is a smoother version of the input signal.

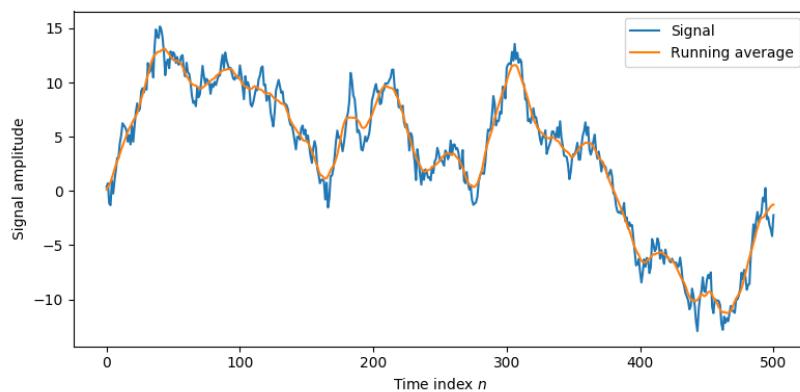


Figure 12.1: A running average filter is often used to smooth a noisy signal. You can find the Python code used to produce this example in `016_smoothing/smoothing.py`.

Finite impulse response filter

The previous system shown in Equation 12.1 is a special case of a more general type of discrete-time LTI system, known as a Finite Impulse Response (FIR) filter. This type of signal is often used in digital signal processing. An FIR filter is defined as follows:

$$y[n] = \sum_{k=-M}^N b_k x[n-k]. \quad (12.2)$$

The coefficients $b_k \in \mathbb{C}$ here are constant valued coefficients. As the name implies, there is a finite number of non-zero coefficients b_k . In the case of the running average filter in Equation 12.1, there would be 15 coefficients, which are all $b_k = 1/15$.

General discrete-time LTI system

What if we allow there to be infinitely many coefficients for the system given in Equation 12.2? We get the following:

$$y[n] = \sum_{k=-\infty}^{\infty} b_k x[n-k] = \sum_{k=-\infty}^{\infty} b[k] x[n-k]. \quad (12.3)$$

This turns out to be the general representation for arbitrary discrete-time LTI systems! In this case, it makes sense to think of the infinitely many coefficients b_k as a signal $b[k]$.

For discrete-time LTI systems in general, the output of a system is given by a discrete-time convolution sum of the input signal $x[n]$ with an impulse response $h[n]$:

$$y[n] = \mathcal{T}\{x[n]\} = \sum_{k=-\infty}^{\infty} h[k] x[n-k]. \quad (12.4)$$

The impulse response is defined as follows:

$$h[n] = \mathcal{T}\{\delta[n]\}. \quad (12.5)$$

It is hopefully easy to see that Equation 12.4 is valid for all LTI systems.

A linear system $\mathcal{T}\{\cdot\}$ ¹ must by definition satisfy the following relation:

$$\mathcal{T}\left\{\sum_{k=-\infty}^{\infty} \alpha_k s_k[n]\right\} = \sum_{k=-\infty}^{\infty} \alpha_k \mathcal{T}\{s_k[n]\}. \quad (12.6)$$

Here $\alpha_k \in \mathbb{C}$ are arbitrary constants and $s_k[n]$ are arbitrary signals.

Time-invariance of $\mathcal{T}\{\cdot\}$, on the other hand, implies that for any time shift k in the input, the output is correspondingly time shifted.

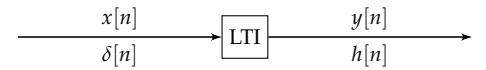


Figure 12.2: Discrete-time LTI systems are characterized by an impulse response $h[n]$, which is the response of the LTI system to a unit impulse signal.

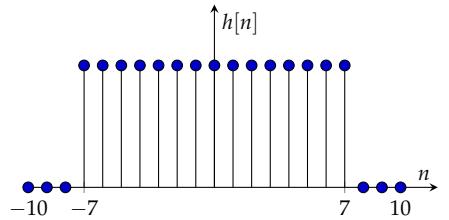


Figure 12.3: The impulse response of the 15 point running mean filter described in Equation 12.1.

¹ If linearity applies for two input signals $\mathcal{T}\{\alpha_1 s_1[n] + \alpha_2 s_2[n]\} = \alpha_1 \mathcal{T}\{s_1[n]\} + \alpha_2 \mathcal{T}\{s_2[n]\}$, it also applies for linear combinations of three or more signals.

This is valid for any input signal $x[n]$:

$$y[n - k] = \mathcal{T}\{x[n - k]\}, \quad (12.7)$$

if

$$y[n] = \mathcal{T}\{x[n]\}. \quad (12.8)$$

It is possible to represent any signal $x[n]$ with the help of time-shifted unit impulse signals²:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k] \delta[n - k]. \quad (12.10)$$

Linearity implies that:

$$\mathcal{T}\{x[n]\} = \mathcal{T}\left\{ \sum_{k=-\infty}^{\infty} x[k] \delta[n - k] \right\} = \sum_{k=-\infty}^{\infty} x[k] \mathcal{T}\{\delta[n - k]\}. \quad (12.11)$$

The equation above is the same as Equation 12.6 with $s_k[n] = \delta[n - k]$ and $\alpha_k = x[k]$. Due to time-invariance, we can relate the impulse response $h[n]$ delayed by k to the term on the right-hand side above. If a unit impulse fed into the system is

$$h[n] = \mathcal{T}\{\delta[n]\}, \quad (12.12)$$

then a time-shifted unit impulse corresponds to a time-shifted output:

$$h[n - k] = \mathcal{T}\{\delta[n - k]\}. \quad (12.13)$$

Therefore, the output of an LTI system for an arbitrary input signal $x[n]$ can be expressed using the impulse response $h[n]$ as follows:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] h[n - k]. \quad (12.14)$$

This type of equation is known as a discrete-time convolution sum. We have now shown that any discrete-time LTI system can be represented with such a convolution sum. Note that Equation 12.14 isn't quite yet the same as Equation 12.4. We will later show that the convolution sum is commutative, i.e., that:

$$\sum_{k=-\infty}^{\infty} x[k] h[n - k] = \sum_{k=-\infty}^{\infty} h[k] x[n - k]. \quad (12.15)$$

Which completes the proof.

Example: Impulse response of an FIR filter

An FIR filter (Equation 12.2) has the following impulse response:

$$h[n] = \sum_{k=-M}^N b_k \delta[n - k]. \quad (12.16)$$

² Recall that the discrete-time unit impulse is defined as:

$$\delta[n] = \begin{cases} 1 & \text{when } n = 0 \\ 0 & \text{otherwise} \end{cases}. \quad (12.9)$$

It is the discrete-time equivalent of a Dirac delta function. The unit impulse is shown in Figure 12.4.

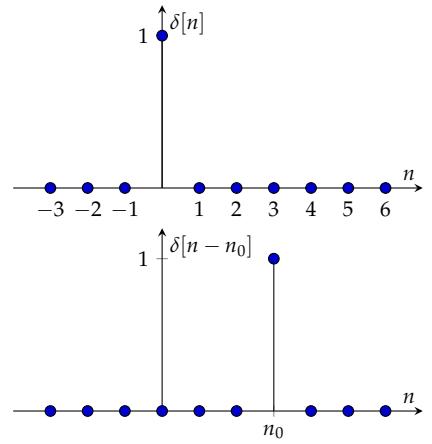


Figure 12.4: Discrete-time unit impulse signal $\delta[n]$ and a time-shifted version $\delta[n - n_0]$ centered at $n = n_0$.

The signal $h[n]$ contains the values of the filter coefficients $h[n] = b_n$. Since there are a finite number of coefficients b_k , the impulse response $h[n]$ has non-zero values only in a finite range of samples. This is also where the name “finite impulse response” comes from.

Impulse response

Linear time-invariant (LTI) systems are fully characterized by an impulse response $h(t)$. This impulse response is obtained by feeding a unit impulse into the LTI system:

$$h(t) = \mathcal{T}\{\delta(t)\} . \quad (12.17)$$

Using an impulse response, it is possible to represent the output of any LTI system as a convolution between the impulse response and the input signal.

$$y(t) = \mathcal{T}\{x(t)\} = h(t) * x(t) = \int_{-\infty}^{\infty} h(\tau)x(t - \tau)d\tau . \quad (12.18)$$

Let's prove this. We'll first need to represent an arbitrary signal as a sum of unit impulses:

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t - \tau)d\tau . \quad (12.19)$$

One way to think of this integral is that the unit impulse $\delta(t - \tau)$ “selects” the value of $x(\tau)$ where $\tau = t$. Another way to think of this is that the Dirac delta functions form a set of basis functions for representing the signal $x(t)$.

You may recall that linearity of the system $\mathcal{T}\{\cdot\}$ implies that:

$$\mathcal{T}\{c_1\delta(t - \tau_1) + c_2\delta(t - \tau_2)\} = c_1\mathcal{T}\{\delta(t - \tau_1)\} + c_2\mathcal{T}\{\delta(t - \tau_2)\} . \quad (12.20)$$

Here I've used $\delta(t - \tau_1)$ and $\delta(t - \tau_2)$ as two different input signals. The terms $c_1, c_2 \in \mathbb{C}$ are arbitrary complex-valued constants.

Linearity must therefore also apply for the linear combination of an arbitrary number of inputs:

$$\mathcal{T}\left\{\sum_n x_n\delta(t - \tau_n)\right\} = \sum_n x_n\mathcal{T}\{\delta(t - \tau_n)\} . \quad (12.21)$$

Linearity can be extended even further into a continuous linear combination:

$$\mathcal{T}\left\{\int_{-\infty}^{\infty} x(\tau)\delta(t - \tau)d\tau\right\} = \int_{-\infty}^{\infty} x(\tau)\mathcal{T}\{\delta(t - \tau)\}d\tau . \quad (12.22)$$

We can simplify the right-hand side and get:

$$\int_{-\infty}^{\infty} x(\tau)\mathcal{T}\{\delta(t - \tau)\}d\tau = \mathcal{T}\{x(t)\} . \quad (12.23)$$



Figure 12.5: The impulse response of an FIR filter is a signal that contains the coefficients. For a finite number of coefficients b_k , the length of the non-zero portion of the impulse response is finite, and hence the name FIR.

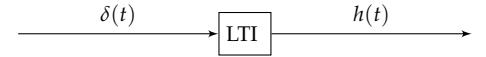


Figure 12.6: A linear time-invariant system is characterized by an impulse response.

In order to simplify the left-hand side, we have to rely on the property of *time-invariance*. That is:

$$h(t) = \mathcal{T}\{\delta(t)\} \Rightarrow h(t - \tau) = \mathcal{T}\{\delta(t - \tau)\}. \quad (12.24)$$

This now completes our proof:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t - \tau)d\tau = \mathcal{T}\{x(t)\} \quad \square. \quad (12.25)$$

The output of a linear time-invariant system $y(t) = \mathcal{T}\{x(t)\}$ is a convolution of the system's impulse response $h(t) = \mathcal{T}\{\delta(t)\}$ with the input signal $x(t)$ fed into the system.

Convolution

A convolution operation is defined for both continuous-time and discrete-time signals. As we just saw, a convolution operation can be interpreted as an LTI system applied to a signal.

The continuous-time convolution is defined as an integral:

$$a(t) * b(t) = \int_{-\infty}^{\infty} a(\tau)b(t - \tau)d\tau. \quad (12.26)$$

The discrete-time convolution is defined as a sum:

$$a[n] * b[n] = \sum_{k=-\infty}^{\infty} a[k]b[n - k]. \quad (12.27)$$

Properties of a convolution

The properties of the convolution operation are shown in Table 12.1. The same properties also exist for the continuous-time convolution operation.

Property	Equation
identity	$x[n] * \delta[n] = x[n]$
commutative	$a[n] * b[n] = b[n] * a[n]$
associative	$(a[n] * b[n]) * c[n] = a[n] * (b[n] * c[n])$
distributive	$a[n] * (b[n] + c[n]) = a[n] * b[n] + a[n] * c[n]$

Table 12.1: Table of convolution properties

The proof of the identity property is clear, and the commutative property is left as an exercise. The associative property takes a bit more work, and the proof is as follows:

Proof. Using the definition of the convolution operation twice:

$$(a[n] * b[n]) * c[n] = \sum_k \left(\underbrace{\sum_{\ell} a[\ell] b[k - \ell]}_{a[k]*b[k]} \right) c[n - k] \quad (12.28)$$

$$= \sum_k \sum_{\ell} a[\ell] b[k - \ell] c[n - k] \quad (12.29)$$

For the other ordering:

$$a[n] * (b[n] * c[n]) = \sum_{\ell} a[\ell] \left(\underbrace{\sum_m b[m] c[(n - m) - \ell]}_{b[n]*c[n]} \right) \quad (12.30)$$

$$= \sum_{\ell} \sum_m a[\ell] b[m] c[(n - m) - \ell]. \quad (12.31)$$

If we now substitute: $m = k - \ell$, then $n - m - \ell = n - k$, which gives

$$a[n] * (b[n] * c[n]) = \sum_k \left(\underbrace{\sum_{\ell} a[\ell] b[k - \ell]}_{a[k]*b[k]} \right) c[n - k], \quad (12.32)$$

which is the same as $(a[n] * b[n]) * c[n]$, which completes the proof. \square

A consequence of the associative property is that if we have two LTI systems that are applied to an input signal $x[n]$ in series, we can come up with a single LTI system that is equivalent to two chained LTI systems:

$$y[n] = (x[n] * h_1[n]) * h_2[n] = x[n] * h_3[n]. \quad (12.33)$$

Here $h_3[n] = h_1[n] * h_2[n]$. This is depicted in Figure 12.7. This property can be extended to an arbitrary number of systems that are connected together in series.

Distributive

The convolution is distributive:

$$a[n] * (b[n] + c[n]) = a[n] * b[n] + a[n] * c[n]. \quad (12.34)$$

Proof.

$$a[n] * (b[n] + c[n]) = \sum_{k=-\infty}^{\infty} a[k](b[n - k] + c[n - k]) \quad (12.35)$$

$$= \sum_{k=-\infty}^{\infty} a[k]b[n - k] + \sum_{k=-\infty}^{\infty} a[k]c[n - k]. \quad (12.36)$$

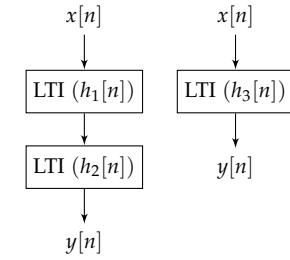


Figure 12.7: A consequence of the associative property of convolution is that two LTI systems characterized with $h_1[n]$ and $h_2[n]$ can be combined as a single LTI system with impulse response $h_3[n] = h_1[n] * h_2[n]$.

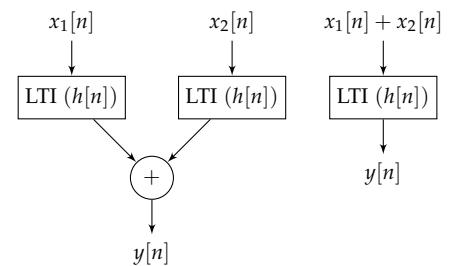


Figure 12.8: A consequence of the distributive property is linearity.

\square

An example application of this property is shown in Figure 12.8. Signals $x_1[n]$ and $x_2[n]$ fed into an identical LTI systems separately and then added together is equivalent to the sum of the signals fed into a single LTI system.

Example: Convolution animations

Animations of the convolution operation for various example systems can be found here: http://kaira.uit.no/juha/fir_animation/. The code for creating such animations can be obtained from GitHub: https://github.com/jvierine/signal_processing/tree/master/017_fir_animation.

Applications of convolution

This chapter only scratches the practical and theoretical uses of convolution and linear time-invariant systems. I'll use two examples to illustrate the application of a one-dimensional convolution equation. Be aware that these are by no means the only types of situations where you will encounter a convolution. Chances are quite high that a signal processing system you can think of is an LTI system, or can be approximated as one. Convolution can be found everywhere!

Example: Radar and sonar equation

The convolution equation is often used to model radar and sonar measurements. In this case, the impulse response signal $h[n]$ represents the signal that is scattered as a function of distance. The signal $x[n]$ represents what a radar or sonar transmits. The signal received by a radar or sonar receiver $m[n]$ is then modeled as:

$$m[n] = h[n] * x[n] \quad (12.37)$$

$$= \sum_{r=0}^M h[r]x[n-r]. \quad (12.38)$$

In this case, the index $r \in \mathbb{N}$ represents round-trip propagation time between the transmitter and the receiver. The larger the distance between the transmitter and the receiver, the larger the delay. Assuming that the sample rate is f_s in units of hertz or ($\frac{1}{s}$), and the group velocity of the transmitter wave v_g ($\frac{m}{s}$), the round-trip range is given by:

$$R = \frac{v_g r}{2f_s}. \quad (12.39)$$

In the case of radar, the group velocity for electromagnetic waves is $v_g \approx 3 \cdot 10^8$ ($\frac{m}{s}$). For sonar, it is the speed of acoustic waves within the medium. In the case of air, this is $v_g \approx 343$ ($\frac{m}{s}$).



Figure 12.9: EISCAT Svalbard Radar. The radar echo from the D-region of the ionosphere can be modeled using the equation shown in Equation 12.38. Photo: Craig Heinselman

To see how the convolution equation is the radar equation, we can use an illustration, as shown in Figure 12.10.

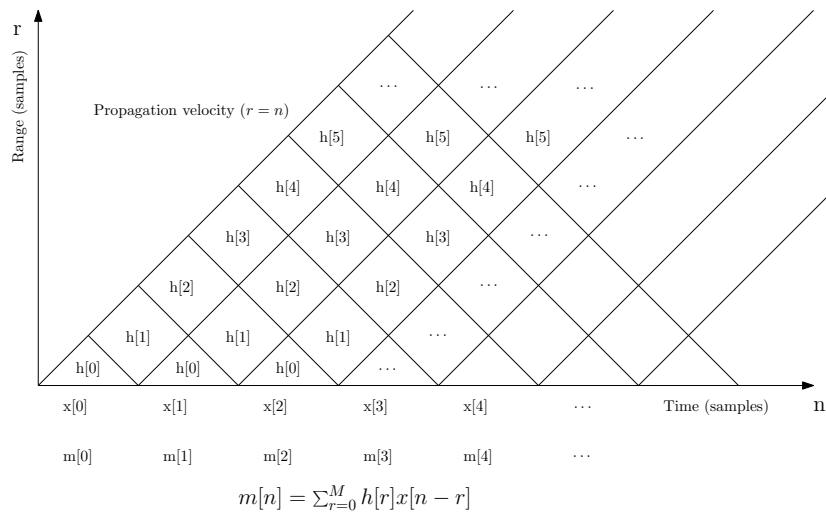


Figure 12.10: A range-time diagram depicting the relationship between a transmitted signal and a scattered signal.

If the probing signal is a unit impulse $x[n] = \delta[n]$ (a very short radar pulse), then the measurement directly provides the scattering amplitude as a function of range:

$$m[n] = \sum_{r=0}^M h[r]\delta[n-r] = h[n]. \quad (12.40)$$

This type of radar is called a pulsed radar. In terms of radar signal processing, this is the easiest case, as no signal processing is needed! A radar measurement in this case is equivalent to measuring the “impulse response” of the region that the radar is probing.

Here is a fascinating example of a blind child that has learned to click his tongue (emitting a $\delta[n]$ -like impulse sound) to probe the acoustic scattering of his surroundings <https://www.youtube.com/watch?v=fN7AIwhpik>. While this may seem like a superhuman feat, this is not unlike measuring the distance between yourself and a mountain side by shouting “echo” very loudly and counting how long it takes for you to hear a strong echo. I am sure that many of you have done this already.

If the transmitted waveform $x[n]$ is more complicated than a unit impulse, the measurements need to be filtered in some way to reconstruct the scattered signal as a function of range $h[n]$. This is achieved by designing a filter $\lambda[n]$, which has the following property $\lambda[n] * x[n] \approx \delta[n]$. After applying the filter $\lambda[n]$ to the measured signal, one obtains the echo as a function of range:

$$\lambda[n] * m[n] = \lambda[n] * h[n] * x[n] = (\lambda[n] * x[n]) * h[n] \approx h[n]. \quad (12.41)$$

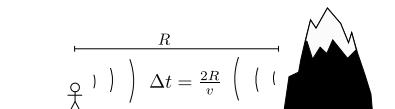


Figure 12.11: The acoustics of a space are determined by acoustic waves scattered from various obstacles at various propagation delays. This can be quite precisely modeled using a convolution, assuming that nothing is moving.

Design of pairs of signals $x[n]$ and filters $\lambda[n]$ is a topic of radar signal processing. An example of a good pairing of a radar transmit signal and a receiver filter is shown in the following convolution animation: http://kaira.uit.no/juha/fir_animation/ex12.gif. This is the so-called 13-bit Barker code and the inverse filter that corresponds to it. The convolution of these two signals is the unit impulse.

Example: Reverb effect

Convolution is encountered when modeling the acoustics of a room or a space in audio signal processing. The basic underlying principle is the same as for radar and sonar.

A sound source $x[n]$ is reflected or scattered from different surfaces within the room at various delays r . Each delay r corresponds to the round-trip propagation delay of the acoustic signal. This is what creates the “sound” of a room. This is also what is the cause for an acoustic “echo” from a mountain side that you may have experienced in real life!

How much signal is scattered from range r is determined by the impulse response $h[r]$. The resulting signal is a convolution $m[n] = h[n] * x[n]$. In this case, performing the convolution results in a reverb effect in acoustic signal processing.

By accurately modeling the impulse response of a room or a large concert hall with an impulse response $h[n]$, it is possible to modify an audio signal $x[n]$ to sound like it was played in that space.

One way to characterize $h[n]$ for a room is to simply measure it. One emits a very short discrete unit impulse $\delta[n]$ using a high quality speaker, and then measures the echoes for this pulse to obtain:

$$h[n] = \sum_{r=0}^M h[r]\delta[n-r]. \quad (12.42)$$

Another possibility is to model the impulse response $h[n]$ by making assumptions about the delays of the reflecting surfaces in a room or space.

The following snippet of code demonstrates modeling of the acoustics of a room using a modeled impulse response for a room.

```
# Reverb effect for sound
#
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sw
import scipy.signal as ss
```

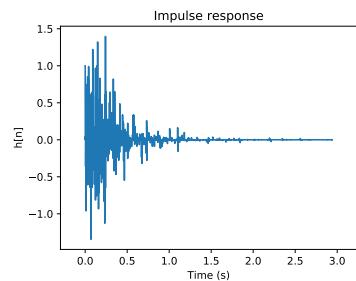


Figure 12.12: An impulse response that models the acoustics of a large space, with echoes at up to 3 seconds time delay.

```

def reverb_model(room_length_std=30.0, # Room wall to wall
                 distance standard deviation in meters.
                 echo_magnitude=0.5,      # How much is reflected
                 from a wall.
                 n_echoes=10,             # Number of echoes.
                 n_walls=10,              # Number of scattering
                 surfaces.
                 sr=44100,                # Sample-rate.
                 c_sound=340.0):          # Speed of sound (m/s).
"""
Simplified model of the acoustics of a room.
Model the reflections from reverberations between surfaces
in a room.
"""

echo_len = int(sr*2*room_length_std*n_echoes/c_sound)
h = np.zeros(echo_len, dtype=np.float32)
h[0] = 1.0

for _ in range(n_walls):
    wall_dist = np.abs(np.random.rand()*room_length_std)
    for i in range(n_echoes):
        idx = int(sr*(i+1)*wall_dist/c_sound)
        if idx < echo_len:
            h[idx] = np.random.rand()*echo_magnitude**(
+1.0)

return h

# Read audio file (.wav format).
ts = sw.read("7na.wav")
sr, clip = ts
if len(clip.shape) == 2: # If stereo, only use one channel.
    print("Using only one stereo channel. Read on.")
    clip = clip[:, 0]

# Make sure the clip is float32.
clip = np.array(clip, dtype=np.float32)

# This is the impulse response that determines
# the acoustics of a room.
h = reverb_model(room_length_std=15.0, n_walls=100,
                 echo_magnitude=0.5)

omhat = np.linspace(-np.pi, np.pi, num=10000)
H = np.zeros(len(omhat), dtype=np.complex64)

for i in range(len(h)):
    H += h[i]*np.exp(1j*omhat*i)

plt.plot(omhat, np.abs(H)**2.0)
plt.show()

# Plot the impulse response
# if the impulse response is short, use stem plot
# otherwise use normal plot (it is much faster).
time_vec = np.arange(len(h))/float(sr)

if len(h) < 100:

```

```

    plt.stem(time_vec, h)
else:
    plt.plot(time_vec, h)

plt.xlim([-0.1*np.max(time_vec), 1.1*np.max(time_vec)])
plt.title("Impulse response")
plt.xlabel("Time (s)")
plt.ylabel("h[n]")
plt.show()

# Plot the audio.
time_vec = np.arange(len(clip))
tidx = np.arange(0, int(np.min([44100, len(clip)])))
plt.subplot(211)
plt.plot(time_vec[tidx], clip[tidx])
plt.title("Original audio")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")

# Convolve the clip with the impulse response
# scipy.signal.convolve.
echo_clip = ss.convolve(clip, h, mode="full")

# Plot the audio.
plt.subplot(212)
plt.title("Convolution output")
# Scale numbers to 0..1 scale.
plt.plot(time_vec[tidx], echo_clip[tidx])
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()

# Normalize to unity.
echo_clip = echo_clip/(np.max(np.abs(echo_clip)))

print("Saving reverb.wav")
# Save as .wav file with 44.1 kHz sample rate.
sw.write("reverb.wav", sr, np.array(20e3*echo_clip, dtype=np.int16))

```

Listing 12.1: 018_reverb/reverb.py

Exercises: Linear Time-invariant Systems

1. Prove that the convolution operation is commutative. That is, show that $a[n] * b[n] = b[n] * a[n]$. In addition, show that this holds true for the continuous-time convolution operation.
2. A running average system is defined as:

$$y[n] = \mathcal{T}\{x[n]\} = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k] \quad (12.43)$$

- a) Show that the running average system is a linear time invariant system using the test for linearity and time invariance.
- b) What is the impulse response $h[n] = \mathcal{T}\{\delta[n]\}$ of the running average system?
- c) How many non-zero values does the impulse response $h[n]$ have?
- d) We feed a discrete-time complex sinusoidal signal $x[n] = e^{i\hat{\omega}_0 n}$ into the system. Show that the output is of the form $y[n] = Ae^{i\phi}e^{i\hat{\omega}_0 n}$, in other words, a discrete-time complex sinusoidal signal with the same frequency as the input signal.
- e) Continue with task d). Let us assume that $L = 4$. What is the amplitude of the output signal A when we have a normalized angular frequency of:
 - i. $\hat{\omega}_0 = 0$ radians per sample?
 - ii. $\hat{\omega}_0 = \pi$ radians per sample?
 - iii. $\hat{\omega}_0 = 0.5\pi$ radians per sample?
- f) Show that system $y_2[n] = \mathcal{T}\{\mathcal{T}\{x[n]\}\}$ is also an LTI system.
- g) What is the impulse response $h_2[n] = \mathcal{T}\{\mathcal{T}\{\delta[n]\}\}$? Sketch a plot of the non-zero values $h_2[n]$.

3. Consider the following running average filter:

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k]. \quad (12.44)$$

- a) Let $x[n]$ be the signal:

$$x[n] = \begin{cases} 1.2, & n = 0, \\ 4.3, & n = 1, \\ 4.7, & n = 2, \\ 3.3, & n = 3, \\ 2.9, & n = 4. \end{cases}$$

Compute the resulting signal $y[n]$ from using the filter in 12.44 by hand, using $L = 3$. Values of n for which $n \notin \{0, 1, 2, 3, 4\}$ are treated as 0.

- b) Implement a function `running_average_filter(L, x)` that computes the running average filter for a given L and signal x . Apply the filter on the signal x from the previous exercise to verify that you get the same answer (up to some rounding errors).
- c) The code shown in Listing 12.2 makes plots of a noisy signal along with the filtered signal, using the running average filter from Equation 12.44. Add the function you wrote from the previous exercise to the code, and verify that the noisy signal has been smoothed by the filter. It should look similar to Figure 12.1.

```
import sys

import matplotlib.pyplot as plt
import numpy as np

def running_average_filter(L: int, x: np.ndarray) -> np.ndarray:
    """Simple implementation of a running average filter."""
    y = np.zeros_like(x)

    # TODO: implement me!

    return y

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print(f"Usage: {sys.argv[0]} L")
        exit(-1)

    L = int(sys.argv[1]) if sys.argv[1].isdecimal() else None

    if L is None:
        print("You need to specify the L value as an integer!")
        exit(-1)

    N = 500 # Length of the example signals.
    t = np.arange(N)

    # Random noise signal, will change with each run of the program.
    x = np.cumsum(np.random.randn(N))

    # Apply the running average filter.
    y = running_average_filter(L=3, x=x)
```

```

plt.plot(t, x, label="Original", color="red")
plt.plot(t, y, label="Filter", color="blue")
plt.legend()
plt.show()

```

Listing 12.2: Given code for exercise 3c

- d) Download `7na.wav` from the course GitHub examples, https://github.com/jvierine/signal_processing/tree/master/018_reverb. Apply the running average filter in Equation 12.44 to the audio using $L = 4, 25, 50, 100$. Can you detect with your ear what happens to the low and high frequency components of the signal as a result of the operation?
4. Obtain the demonstration code that implements a reverb effect using a convolution operation:

$$y[n] = h[n] * x[n], \quad (12.45)$$

where $x[n]$ is the input signal, $h[n]$ is the impulse response of a room, and $y[n]$ is the output signal with the reverb effect. Download `7na.wav` from the course GitHub examples, https://github.com/jvierine/signal_processing/tree/master/018_reverb or alternatively use your own audio file.

- a) Run the example code and verify that the filtered signal indeed sounds like it is played in a large room by playing `7na.wav` and the file `reverb.wav` produced by the script.
- b) Find the part in the code where a convolution between the audio signal and the impulse response is evaluated. Plot the impulse response of the FIR filter applied to the input signal.
- c) Figure out how to increase and reduce the amount of reverb, i.e., to make it sound like the audio signal is played in a large or small room with many surfaces that reflect sound waves. What does the impulse response for a large room and a small room look like?
- d) Now try to create an impulse response of an echo from a distance of 1000 meters using the following impulse response $h[n] = \delta[n] + 0.5\delta[n - n_0]$. Assuming that the propagation velocity of sound is $343 \frac{\text{m}}{\text{s}}$, determine a suitable value for n_0 .

Suggested solutions: Linear Time-invariant Systems

1. Claim: $a[n] * b[n] = b[n] * a[n]$.

Proof. By definition, the convolution of two discrete-time signals is defined as:

$$a[n] * b[n] = \sum_{k=-\infty}^{\infty} a[k]b[n-k].$$

Change variables by setting $l = n - k$, so that $k = n - l$, then:

$$a[n] * b[n] = \sum_{l=-\infty}^{\infty} a[n-l]b[l] = \sum_{l=-\infty}^{\infty} b[l]a[n-l].$$

The last sum is the definition of $b[n] * a[n]$, just with the sum index named l , hence $a[n] * b[n] = b[n] * a[n]$. \square

The same approach can be used to prove the commutativity of the continuous-time convolution. Again, substitution of variables, but with integrals. Here is the proof:

Proof.

$$\begin{aligned} a(t) * b(t) &= \int_{-\infty}^{\infty} a(\tau)b(t-\tau)d\tau, \\ &= - \int_{\infty}^{-\infty} a(t-u)b(u)du, \\ &= \int_{-\infty}^{\infty} b(u)a(t-u)du, \\ &= b(t) * a(t), \end{aligned}$$

\square

where $u = t - \tau$, giving $du = -d\tau$. Note that the minus sign is used to swap the limits.

2. Consider the running average system, defined as:

$$y[n] = \mathcal{T}\{x[n]\} = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k].$$

- Consider two discrete-time signals $x_1[n]$ and $x_2[n]$ with arbitrary constants c_1, c_2 , then:

$$\begin{aligned} \mathcal{T}\{c_1x_1[n] + c_2x_2[n]\} &= \frac{1}{L} \sum_{j=0}^{L-1} [c_1x_1[n-j] + c_2x_2[n-j]], \\ c_1\mathcal{T}\{x_1[n]\} + c_2\mathcal{T}\{x_2[n]\} &= c_1 \frac{1}{L} \sum_{k=0}^{L-1} x_1[n-k] + c_2 \frac{1}{L} \sum_{l=0}^{L-1} x_2[n-l], \end{aligned}$$

which are equal.

For time-invariance we have:

$$\begin{aligned}\mathcal{T}\{\mathcal{D}\{x[n]\}\} &= \mathcal{T}\{x[n - \tau]\} = \frac{1}{L} \sum_{k=0}^{L-1} x[n - \tau - k], \\ \mathcal{D}\{\mathcal{T}\{x[n]\}\} &= \mathcal{D}\left\{\frac{1}{L} \sum_{k=0}^{L-1} x[n - k]\right\} = \frac{1}{L} \sum_{k=0}^{L-1} x[n - \tau - k],\end{aligned}$$

both are equal, so the system is time-invariant.

- b) The impulse response can be determined by $h[n] = \mathcal{T}\{\delta[n]\}$, doing this yields:

$$h[n] = \frac{1}{L} \sum_{k=0}^{L-1} \delta[n - k].$$

The impulse response function is then:

$$h[n] = \begin{cases} \frac{1}{L}, & n = 0, \dots, L-1, \\ 0, & \text{otherwise.} \end{cases}$$

- c) The impulse response has L nonzero values, all of them being $1/L$.
- d) Let $x[n] = e^{i\hat{\omega}_0 n}$, then if we feed this into our system we get (using the formula for a geometric sum):

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} e^{i\hat{\omega}_0(n-k)} = \frac{1}{L} e^{i\hat{\omega}_0 n} \sum_{k=0}^{L-1} e^{-i\hat{\omega}_0 k} = \frac{1}{L} e^{i\hat{\omega}_0 n} \left(\frac{1 - (e^{-i\hat{\omega}_0})^L}{1 - e^{-i\hat{\omega}_0}} \right).$$

The output signal can then be rewritten as:

$$y[n] = \frac{1}{L} e^{i\hat{\omega}_0 n} \frac{e^{-i\hat{\omega}_0 L/2}}{e^{-i\hat{\omega}_0/2}} \left(\frac{e^{i\hat{\omega}_0 L/2} - e^{-i\hat{\omega}_0 L/2}}{e^{i\hat{\omega}_0/2} - e^{-i\hat{\omega}_0/2}} \right),$$

then use the definition of $\sin(\theta)$ for complex values to obtain:

$$y[n] = \frac{1}{L} e^{i\hat{\omega}_0 n} e^{-i\hat{\omega}_0(L-1)/2} \frac{\sin(\hat{\omega}_0 L/2)}{\sin(\hat{\omega}_0/2)}.$$

Later, we'll introduce this as a filter that contains two parts, one part being a Dirichlet kernel which dictates how the magnitude of the frequencies are filtered, the other part is a time-delay, which tells us how the filter delays the signal. We can see this somewhat already, as we have:

$$y[n] = \left(\frac{1}{L} e^{-i\hat{\omega}_0(L-1)/2} \frac{\sin(\hat{\omega}_0 L/2)}{\sin(\hat{\omega}_0/2)} \right) x[n].$$

Here we see that:

$$D_L(\hat{\omega}_0) = \frac{1}{L} \frac{\sin(\hat{\omega}_0 L/2)}{\sin(\hat{\omega}_0/2)},$$

changes the magnitude, while:

$$e^{-i\hat{\omega}_0(L-1)/2}$$

delays the signal.

- e) Take $L = 4$, then:

$$y[n] = \frac{1}{4} e^{-i\hat{\omega}_0 3/2} \frac{\sin(2\hat{\omega}_0)}{\sin(\hat{\omega}_0/2)} x[n].$$

Next, consider the cases of input frequencies:

$$\begin{aligned}\hat{\omega} &= 0, \\ \hat{\omega} &= \pi, \\ \hat{\omega} &= 0.5\pi,\end{aligned}$$

from our discussion above, we know that the amplitude is dependent on the $D_L(\hat{\omega}_0)$ function only. In these cases the amplitude of the output is:

$$\begin{aligned}A &= \lim_{\hat{\omega}_0 \rightarrow 0} \frac{1}{4} \frac{\sin(2\hat{\omega}_0)}{\sin(\hat{\omega}_0/2)} = \lim_{\hat{\omega}_0 \rightarrow 0} \frac{1}{4} \frac{2\cos(2\hat{\omega}_0)}{\frac{1}{2}\cos(\hat{\omega}_0/2)} = 1, \\ A &= \frac{1}{4} \frac{\sin(2\pi)}{\sin(\pi/2)} = 0, \\ A &= \frac{1}{4} \frac{\sin(2\pi/2)}{\sin(\pi/4)} = 0.\end{aligned}$$

We have used L'Hôpital's rule to determine the behavior of the filter at $\omega_0 = 0$, otherwise, just plug in the value for ω_0 . The filter seems to behave as a low-pass filter, passing low frequencies and removing high frequencies.

- f) The system $y[n] = \mathcal{T}\{x[n]\}$ is an LTI system and can be written as $y[n] = h[n] * x[n]$, where $h[n]$ is the impulse response function as defined above. Then the system can be described as:

$$y_2[n] = \mathcal{T}\{\mathcal{T}\{x[n]\}\} = \mathcal{T}\{h[n] * x[n]\} = h[n] * (h[n] * x[n]) = (h[n] * h[n]) * x[n].$$

We've used associativity of convolution in the final step. Then the system can be described as $y_2[n] = h_2[n] * x[n]$, where $h_2[n] = h[n] * h[n]$. Hence, $y_2[n]$ is an LTI system, as all LTI systems are fully determined by convolution with an impulse response.

- g) Have that $h_2[n] = \mathcal{T}\{\mathcal{T}\{\delta[n]\}\} = h[n] * h[n]$, so:

$$h_2[n] = \sum_{k=-\infty}^{\infty} h[k]h[n-k] = h[0]h[n] + h[1]h[n-1] + h[2]h[n-2] + h[3]h[n-3] + h[4]h[n-4].$$

Terms with $n < 0$ are dropped as these will be 0 due to $h[n] = 0$ for $n < 0$. Let's evaluate the values of $h_2[n]$ with $L = 4$:

$$\begin{aligned} h_2[0] &= h[0]h[0] + h[1]h[-1] + h[2]h[-2] + h[3]h[-3] + h[4]h[-4] = \frac{1}{L^2}, \\ h_2[1] &= h[0]h[1] + h[1]h[0] + h[2]h[-1] + h[3]h[-2] + h[4]h[-3] = \frac{2}{L^2}, \\ h_2[2] &= h[0]h[2] + h[1]h[1] + h[2]h[0] + h[3]h[-1] + h[4]h[-2] = \frac{3}{L^2}, \\ h_2[3] &= h[0]h[3] + h[1]h[2] + h[2]h[1] + h[3]h[0] + h[4]h[-1] = \frac{4}{L^2}, \\ h_2[4] &= h[0]h[4] + h[1]h[3] + h[2]h[2] + h[3]h[1] + h[4]h[0] = \frac{3}{L^2}, \\ h_2[5] &= h[0]h[5] + h[1]h[4] + h[2]h[3] + h[3]h[2] + h[4]h[1] = \frac{2}{L^2}, \\ h_2[6] &= h[0]h[6] + h[1]h[5] + h[2]h[4] + h[3]h[3] + h[4]h[2] = \frac{1}{L^2}, \end{aligned}$$

while the rest are 0. Thus:

$$h_2[n] = \begin{cases} \frac{1}{16}, & n = 0, 6, \\ \frac{2}{16}, & n = 1, 5, \\ \frac{3}{16}, & n = 2, 4, \\ \frac{4}{16}, & n = 3, \\ 0, & \text{otherwise.} \end{cases}$$

To draw the impulse response function, we write a little program to do it. The program is shown in Listing 12.3. The impulse response is 0 for all values of n not shown.

```
import matplotlib.pyplot as plt
import numpy as np

L = 4
h = np.repeat(1/L, L) # Add 1/L into an array L times.

# Compute the convolution.
h2 = np.convolve(h, h, mode="full")

plt.stem(h2)
plt.xlabel("Samples $(n)$")
plt.ylabel("$h_{\{2\}}[n]$")
plt.title("Impulse response")
# Call this if needed.
```

Listing 12.3: Simple convolution

The output of the program is shown in Figure 12.13.

3. a) The output signal $y[n]$ will have the same length as $x[n]$, thus,

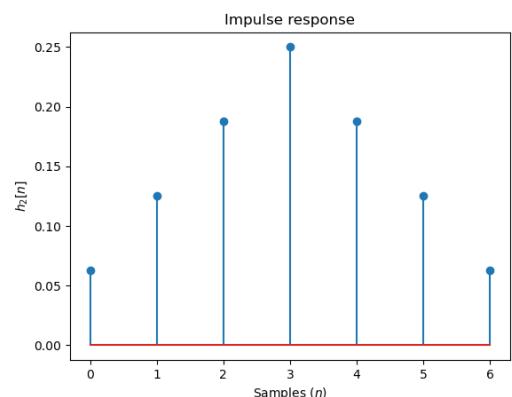


Figure 12.13: Impulse response for $y_2[n]$

computing each value is done as follows:

$$\begin{aligned}y[0] &= \frac{1}{1}(x[0-0] + x[0-1] + x[0-2]), \\y[1] &= \frac{1}{2}(x[1-0] + x[1-1] + x[1-2]), \\y[2] &= \frac{1}{3}(x[2-0] + x[2-1] + x[2-2]), \\y[3] &= \frac{1}{3}(x[3-0] + x[3-1] + x[3-2]), \\y[4] &= \frac{1}{3}(x[4-0] + x[4-1] + x[4-2]),\end{aligned}$$

Note that the values of factor in front varies. The reason is that the operation is to average, so we must divide by the total number of terms that are counted. Remove all the negative index terms (which are treated as 0), we get:

$$\begin{aligned}y[0] &= \frac{1}{1}(x[0-0]) = x[0] = 1.2, \\y[1] &= \frac{1}{2}(x[1-0] + x[1-1]) = \frac{1}{2}(x[1] + x[0]) = 2.75, \\y[2] &= \frac{1}{3}(x[2-0] + x[2-1] + x[2-2]) = \frac{1}{3}(x[2] + x[1] + x[0]) = 3.40, \\y[3] &= \frac{1}{3}(x[3-0] + x[3-1] + x[3-2]) = \frac{1}{3}(x[3] + x[2] + x[1]) = 4.10, \\y[4] &= \frac{1}{3}(x[4-0] + x[4-1] + x[4-2]) = \frac{1}{3}(x[4] + x[3] + x[2]) = 3.63,\end{aligned}$$

so the filtered signal $y[n]$ is then:

$$y[n] = \begin{cases} 1.2, & n = 0, \\ 2.75, & n = 1, \\ 3.40, & n = 2, \\ 4.10, & n = 3, \\ 3.63, & n = 4. \end{cases}$$

- b) Listing 12.4 shows a Python implementation that implements the running average filter from Equation 12.44. Running this code on the signal $x[n]$ gives the same result as the computation done in the previous exercise.

```
import numpy as np

def running_average_filter(L: int, x: np.ndarray) -> np.ndarray:
    """Simple implementation of a running average filter."""
    y = np.zeros_like(x)
```

```

for n in range(len(y)):

    # Determine if the index is negative.
    # If yes, pick 0 as the starting point, otherwise
    # use n - L + 1. Add 1 to avoid the n = L case.
    start = max(0, n - L + 1)

    # Use +1 since n is never actually the final index
    # of y.
    y[n] = np.mean(x[start:n+1])

return y

if __name__ == "__main__":
    x = np.array([1.2, 4.3, 4.7, 3.3, 2.9], dtype=np.
float32)
    print(running_average_filter(3, x))

```

Listing 12.4: Solution for exercise 3b

- c) Listing 12.5 shows the previous function added to the code from Listing 12.2

```

import sys

import matplotlib.pyplot as plt
import numpy as np

def running_average_filter(L: int, x: np.ndarray) -> np.
ndarray:
    """Simple implementation of a running average
    filter."""
    y = np.zeros_like(x)

    for n in range(len(y)):

        # Determine if the index is negative.
        # If yes, pick 0 as the starting point, otherwise
        # use n - L + 1. Add 1 to avoid the n = L case.
        start = max(0, n - L + 1)

        y[n] = np.mean(x[start:n+1])  # Use +1 since n is
        never actually the final index of y.

    return y

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: <L>")
        exit(1)

    L = int(sys.argv[1]) if sys.argv[1].isdecimal() else
None

```

```

if L is None:
    print("L should be an integer!")
    exit(1)

N = 500 # Length of the example signals.
t = np.arange(N)

# Random noise signal, will change with each run of the
# program.
x = np.cumsum(np.random.randn(N))

# Apply the running average filter.
y = running_average_filter(L=L, x=x)

plt.plot(t, x, label="Original", color="red")
plt.plot(t, y, label="Running average filtered", color=
"blue")
plt.xlabel("Samples $n$")
plt.legend()
# # plt.show() # Call this if needed.

```

Listing 12.5: Solution for exercise 3c

The plot generated is shown in Figure 12.14. Note that the plot is random, so yours will look different when the code is run.

- d) The code in Listing 12.6 implements the running average filter and applies the running average filter to the input file. The code is written so that the user can run the code by providing the filename and L on the command line.

```

import sys

import numpy as np
from scipy.io.wavfile import read, write

def read_wav_file(filename: str) -> tuple[float, np.ndarray]:
    """Function to read a .wav file. Returns the sample
    rate and a single channel with the audio.
    """

    # Read the file, then extract the necessary information
    wav_file = read(filename)
    sr, signal = wav_file

    # Check if the audio is stereo or mono.
    if len(signal.shape) == 2:
        # Use only 1 channel if stereo.
        signal = signal[:, 0]

    return sr, signal

def write_wav_file(filename: str, sr: float, signal: np.
ndarray) -> None:
    """Function to write to a .wav file. Arguments consists

```

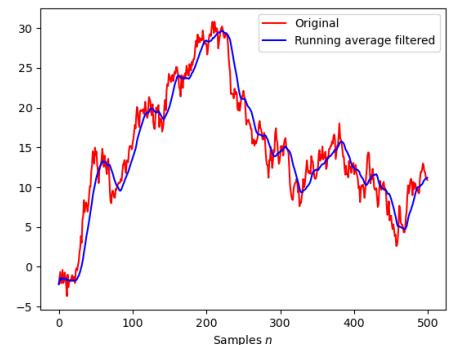


Figure 12.14: Noisy signal that has been smoothed with a running average filter using $L = 25$.

```

of the filename to write to, the sample rate and the
actual
data to write into the file."""

# Use the floating point file format for .wav files.
# This requires the data to be cropped between -1 and
1.
# clipped_signal = np.clip(signal, a_min=-1.0, a_max
=1.0, dtype=np.float32)
clipped_signal = 0.9*signal/np.max(np.abs(signal))

write(filename, sr, clipped_signal.astype(np.float32))

def running_average_filter(L: int, x: np.ndarray) -> np.
ndarray:
    """Simple implementation of a running average
filter."""

y = np.zeros_like(x)

for n in range(len(y)):
    start = max(0, n - L + 1)

    y[n] = np.mean(x[start:n+1])

return y

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: <filename> <L>")
        exit(1)

    _, filename, l = sys.argv
    print(f"Filtering file: '{filename}' using L = {l}")

    # Verify that the input for the program is okay.
    L = int(l) if l.isdecimal() else None

    if L is None:
        print("Invalid size of running average filter.")
        exit(1)

    # Read the file from disk.
    sr, signal = read_wav_file(filename)

    # Filter the audio signal using a running average
    # low-pass filter with the user specified L.
    filtered_signal = running_average_filter(L, signal)

    # Write the result to disk.
    write_wav_file(f"{filename.replace('.wav', '')}_{L}
_filtered.wav", sr, filtered_signal)

```

Listing 12.6: Suggested solution for exercise 3d

The effect on the audio can be heard between the original and for large L . The higher frequencies have been reduced, so they

are less prevalent. This is not surprising, as the filter we are using is a low-pass filter.

4. The code for this exercise is based on the reverb demonstration code.

- By running `reverb.py` and listening to `reverb.wav`, one can hear the reverb effect having been applied to the audio-signal.
- Running the code in Listing 12.7 yields the figure shown in Figure 12.15. Each peak shown represents an impulse that has been reflected by some wall, except the first peak, which represents the original signal.

```
import sys
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.wavfile import read, write

# This code is based on the reverb example.

def read_wav_file(filename: str) -> tuple[int, np.ndarray]:
    """Function to read a .wav file. Returns the sample rate and a single channel with the audio."""
    # Read the file, then extract the necessary information
    wav_file = read(filename)
    sr, signal = wav_file

    # Check if the audio is stereo or mono.
    if len(signal.shape) == 2:
        # Use only 1 channel if stereo.
        signal = signal[:, 0]

    return sr, signal

def write_wav_file(filename: str, sr: float, signal: np.ndarray) -> None:
    """Function to write to a .wav file. Arguments consists of the filename to write to, the sample rate and the actual data to write into the file."""
    # Use the floating point file format for .wav files.
    # This requires the data to be cropped between -1 and 1.
    # clipped_signal = np.clip(signal, a_min=-1.0, a_max=1.0, dtype=np.float32)
    clipped_signal = 0.9 * signal / np.max(np.abs(signal))

    write(filename, sr, clipped_signal.astype(np.float32))

def reverb_model(room_length_std=30.0, # Room wall to wall distance standard deviation in meters.
```

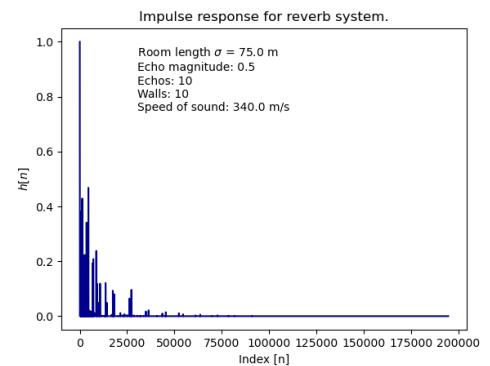


Figure 12.15: Impulse response for the reverb LTI system.

```

        echo_magnitude=0.5,      # How much is
reflected from a wall.
        n_echoes=10,            # Number of echoes.
        n_walls=10,            # Number of
scattering surfaces.
        sr=44100,              # Sample-rate.
        c_sound=340.0):         # Speed of sound (m
/s).
"""Simplified model of the acoustics of a room.
Model the reflections from reverberations between
surfaces in a room.
"""

echo_len = int(2*sr*room_length_std*n_echoes/c_sound)
h = np.zeros(echo_len, dtype=np.float32)
h[0] = 1.0 # This represents the initial impulse (the
original signal).

# Add each echo for the given number of walls.
for _ in range(n_walls):
    # Generate random wall distance.
    wall_dist = np.abs(np.random.rand()*room_length_std
)

    for i in range(n_echoes):
        idx = int(sr*(i + 1)*wall_dist/c_sound)

        if idx < echo_len:
            # Echoes decay. This model makes each echo
decay as x^(i + 1), where x < 1
            # and i is the index for the ith echo.
            h[idx] = np.random.rand()*echo_magnitude**(
i + 1.0)

return h

if __name__ == "__main__":
    room_length_std = 75.0 # Standard deviation for the
room length.
    echo_magnitude = 0.5 # Must be less than 1.
    n_echoes = 10 # Must be greater than 0.
    n_walls = 10 # Number of walls for the echo to bounce
off.
    c_sound = 340.0 # Speed of sound.

    if len(sys.argv) != 2:
        print("Usage: <filename>")
        exit(1)

    _, filename = sys.argv

    print(f"Reading .wav file named {filename}")
    sr, x = read_wav_file(filename)

    print("Computing the reverb model...")
    h = reverb_model(room_length_std, echo_magnitude,
n_echoes, n_walls, sr, c_sound)

    # Convolution happens here. The impulse response is

```

```

convolved with
# the audio signal (called x here), to spit out the
reverb effect signal.
print("Computing the convolution...")
y = np.convolve(x, h, mode="full")

print("Saving reverb.wav file.")
write_wav_file("reverb.wav", sr, y)

# Plot the impulse response.
plt.plot(h, color="darkblue")
plt.xlabel("Index [n]")
plt.ylabel("$h[n]$")
plt.title(rf"Impulse response for reverb system.")
plt.text(9.0, 0.7,
rf"""
Room length $\sigma$ = {room_length_std} m
Echo magnitude: {echo_magnitude}
Echos: {n_echoes}
Walls: {n_walls}
Speed of sound: {c_sound} m/s
""")
# plt.show() # If needed

```

Listing 12.7: Suggested solution for exercise 4b

- c) By changing `room_length_std` one can increase the reverb effect. Making this parameter larger means the sound will travel further and reflect at a later time. This gives the effect of a larger room. The impulse responses are similar, but the impulse response for a larger room has more peaks at higher indices. This makes sense, as the smaller room will not have any reflections on these indices as they do not exist for the small room. That is, there are no walls to reflect off of at such distances. Running Listing 12.8 gives Figure 12.16, where we can see that the larger room has more peaks at higher indices than the smaller room. You can also see that the smaller room has more peaks at lower indices, which is as expected as these reflections come earlier, due to the signal having less travel distance, than in the large room.

```

import numpy as np
import matplotlib.pyplot as plt

def reverb_model(room_length_std=30.0, # Room wall to wall
                 distance standard deviation in meters.
                 echo_magnitude=0.5,      # How much is
                 reflected from a wall.
                 n_echoes=10,             # Number of echoes.
                 n_walls=10,              # Number of
                 scattering surfaces.
                 sr=44100,                # Sample-rate.
                 c_sound=340.0):          # Speed of sound (m
                 /s).

```

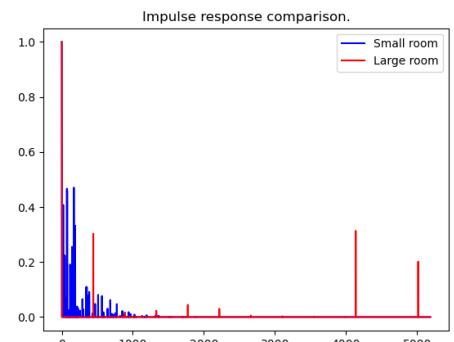


Figure 12.16: Impulse response for the reverb LTI system.

```

""" Simplified model of the acoustics of a room.
Model the reflections from reverberations between
surfaces in a room.
"""

echo_len = int(2*sr*room_length_std*n_echoes/c_sound)
h = np.zeros(echo_len, dtype=np.float32)
h[0] = 1.0 # This represents the initial impulse (the
           # original signal).

# Add each echo for the given number of walls.
for _ in range(n_walls):
    # Generate random wall distance.
    wall_dist = np.abs(np.random.rand() * room_length_std
)

    for i in range(n_echoes):
        idx = int(sr*(i + 1)*wall_dist/c_sound)

        if idx < echo_len:
            # Echoes decay. This model makes each echo
            # decay as x^(i + 1), where x < 1
            # and i is the index for the ith echo.
            h[idx] = np.random.rand() * echo_magnitude**(
                i + 1.0)

return h

h1 = reverb_model(2.0, 0.5, 10, 10, 44100, 340)
h2 = reverb_model(75.0, 0.5, 10, 10, 44100, 340)

# h1 will be shorter than h2 when the room is larger.
length = min(len(h1), len(h2))

t = np.arange(length)

plt.plot(t, h1[:length], label="Small room", color="blue")
plt.plot(t, h2[:length], label="Large room", color="red")
plt.xlabel("Samples $n$")
plt.legend()
plt.title("Impulse response comparison.")
# plt.show() # Run if needed.

```

Listing 12.8: Suggested solution for exercise 4b

- d) The impulse response $h[n] = \delta[n] + 0.5\delta[n - n_0]$ represents a system where the original signal is unchanged, this can be seen from the $\delta[n]$ term. In addition, there is the $0.5\delta[n - n_0]$ term, which represents the original signal being delayed and reduced by a factor of 0.5 in amplitude. We want this to represent a delay of an echo from a 1000 meters distance. To do this, we must find n_0 so that the distance traveled by the echo is 1000 meters. Assuming a constant velocity for the audio signal, the total time it takes for the signal to travel, bounce off a wall and come back is then:

$$t = \frac{2s}{v}$$

where $s = 1000$ m and $v = 343$ m/s (the assumed speed of sound). The relation between time in seconds and samples for a discretized signal is then:

$$t = T_s n.$$

Solving for n_0 , we obtain:

$$n_0 = \frac{2s}{T_s v} = \frac{2s f_s}{v},$$

where f_s is the sample rate. Plugging in the numbers yields:

$$n_0 = \frac{2s f_s}{v} = \frac{(2)(1000 \text{ m})(44100) \text{ Hz}}{343 \text{ m/s}} \approx 257142.85,$$

or $n_0 = 257143$ to round to the nearest integer. The implementation of this reverb effect is shown in Listing 12.9. The delay should occur at

$$t = T_s n = n / f_s = 257143 / 44100 \text{ Hz} = 5.83 \text{ seconds.}$$

If you listen to the audio produced by the code, you should be able to hear the delay coming in at around 5.83 seconds, as hoped.

```
import sys

import numpy as np
from scipy.io.wavfile import read, write

# This code is based on the reverb example .

def read_wav_file(filename: str) -> tuple[int, np.ndarray]:
    """Function to read a .wav file. Returns the sample
    rate and a single channel with the audio.
    """

    # Read the file, then extract the necessary information
    #
    wav_file = read(filename)
    sr, signal = wav_file

    # Check if the audio is stereo or mono.
    if len(signal.shape) == 2:
        # Use only 1 channel if stereo.
        signal = signal[:, 0]

    return sr, signal


def write_wav_file(filename: str, sr: float, signal: np.
ndarray) -> None:
    """Function to write to a .wav file. Arguments consists
    of the filename to write to, the sample rate and
    the actual data to write into the file."""

```

```

# Use the floating point file format for .wav files.
# This requires the data to be cropped between -1 and
# 1.
# clipped_signal = np.clip(signal, a_min=-1.0, a_max
#=1.0, dtype=np.float32)
clipped_signal = 0.9*signal/np.max(np.abs(signal))

write(filename, sr, clipped_signal.astype(np.float32))

def reverb_1000_meters_model(room_length_std=1000.0, # Room wall to wall distance standard deviation in meters
                            echo_magnitude=0.5,           # How much is reflected from a wall.
                            sr=44100,                   # Sample-rate.
                            c_sound=343.0):             # Speed of sound (m/s).
    """Implementation of the delta[n] + 0.5delta[n - no]
    system.
    """

    # Formula for computing no, see pdf for the computation
    no = int((2*sr*room_length_std)/c_sound)
    h = np.zeros(2*no, dtype=np.float32)

    h[0] = 1.0 # This represents the initial impulse (the original signal).

    # This is the reflection coming from a wall at a 1000 meter distance.
    # Note that the amplitude is halved.
    h[no] = echo_magnitude

    return h

if __name__ == "__main__":
    room_length_std = 1000.0 # Standard deviation for the room length.
    echo_magnitude = 0.5 # Must be less than 1.
    c_sound = 343.0 # Speed of sound.

    if len(sys.argv) != 2:
        print("Usage: <filename>")
        exit(1)

    _, filename = sys.argv

    print(f"Reading .wav file named {filename}")
    sr, x = read_wav_file(filename)

    print("Computing the reverb model...")
    h = reverb_1000_meters_model(room_length_std,
                                echo_magnitude, sr, c_sound)

    # Convolution happens here. The impulse response is

```

```
convolved with
# the audio signal (called x here), to spit out the
reverb effect signal.
print("Computing the convolution... ")
y = np.convolve(x, h, mode="full")

print("Saving 1000m_reverb.wav file.")
write_wav_file("1000m_reverb.wav", sr, y)
```

Listing 12.9: Solution for exercise 4

13

Frequency Response

In this chapter, we will investigate the response of an LTI system to a complex sinusoidal input. This investigation leads to the concept of *frequency response* of an LTI system. The frequency response allows you to determine how an LTI system modifies each frequency component of a signal fed into the system.

Every LTI system $\mathcal{T}\{\cdot\}$ can be expressed as a convolution of an input signal $x(t)$ with the impulse response $h(t) = \mathcal{T}\{\delta(t)\}$ of the LTI system:

$$y(t) = \mathcal{T}\{x(t)\} \quad (13.1)$$

$$= x(t) * h(t) \quad (13.2)$$

$$= \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau. \quad (13.3)$$

If we feed a complex sinusoidal signal $x(t) = Ae^{i\phi}e^{i\omega t}$ to an LTI system, we get¹:

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \quad (13.5)$$

$$= \int_{-\infty}^{\infty} h(\tau)Ae^{i\phi}e^{i\omega(t-\tau)}d\tau \quad (13.6)$$

$$= \int_{-\infty}^{\infty} h(\tau)e^{-i\omega\tau}Ae^{i\phi}e^{i\omega t}d\tau \quad (13.7)$$

$$= \underbrace{\left(\int_{-\infty}^{\infty} h(\tau)e^{-i\omega\tau}d\tau \right)}_{\mathcal{H}(\omega)} \underbrace{\left(Ae^{i\phi}e^{i\omega t} \right)}_{x(t)} \quad (13.8)$$

$$= \mathcal{H}(\omega)x(t). \quad (13.9)$$

The output of the system is the input signal, multiplied by a complex function $\mathcal{H}(\omega) \in \mathbb{C}$, which depends on the angular frequency ω of the input signal. The term $\mathcal{H}(\omega)$ is called the frequency response of the LTI system:

$$\boxed{\mathcal{H}(\omega) = \int_{-\infty}^{\infty} h(\tau)e^{-i\omega\tau}d\tau}. \quad (13.10)$$

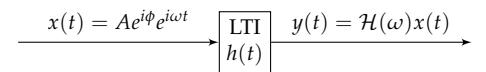


Figure 13.1: The frequency response of a continuous-time LTI system is obtained by investigating what the system does to a complex sinusoidal signal.

¹ An *eigenfunction* $x(t)$ of an operator $\mathcal{T}\{\cdot\}$ is defined as a function that has the following property:

$$\mathcal{T}\{x(t)\} = \lambda x(t), \quad (13.4)$$

where $\lambda \in \mathbb{C}$ is a constant. This means that applying an operator on an eigenfunction of that operator is equivalent to multiplication of this eigenfunction by a constant.

The derivation leading up to Equation 13.9 shows that complex sinusoidal signals $x(t) = Ae^{i\phi}e^{i\omega t}$ are eigenfunctions of LTI systems. The same property also applies to discrete-time LTI systems and discrete-time complex sinusoidal signals.

From the definition, we can see that it is the same thing as a Fourier transform of the impulse response $h(t)$.

Arbitrary signals

Let's consider an arbitrary continuous-time signal that is represented as a sum² of complex sinusoidal signals using the inverse Fourier transform:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega . \quad (13.11)$$

Here $\hat{x}(\omega)$ is the Fourier transform of $x(t)$ – it represents the complex amplitude of spectral components of the signal as a function of angular frequency ω .

The convolution theorem³ states that convolution in time domain is multiplication in frequency domain:

$$\boxed{y(t) = h(t) * x(t) \xleftrightarrow{\mathcal{F}} \hat{y}(\omega) = \mathcal{H}(\omega) \hat{x}(\omega)} . \quad (13.12)$$

The frequency response therefore tells us how each frequency component of the input signal is modified by the LTI system. Therefore, we can view an LTI system as a filter. How each frequency component of the input signal is modified, is determined by the Fourier transform of the impulse response of the LTI system, which we will call frequency response $\mathcal{H}(\omega)$.

Discrete-time frequency response

A discrete-time LTI system $\mathcal{T}\{\cdot\}$ is also described by a convolution of the input signal with the impulse response $h[n]$ of the LTI system:

$$y[n] = \mathcal{T}\{x[n]\} \quad (13.13)$$

$$= x[n] * h[n] \quad (13.14)$$

$$= \sum_{k=-\infty}^{\infty} h[k] x[n-k] . \quad (13.15)$$

When inspecting the case where the input is a complex sinusoidal signal $x[n] = A e^{i\phi} e^{i\hat{\omega}n}$, with a normalized angular frequency $\hat{\omega}$ and phase ϕ , we get⁴:

² You can think of the inverse Fourier transform as stating that the signal $x(t)$ consists of a linear combination of infinitely many complex sinusoidal signals.

³ the convolution theorem was discussed in the continuous-time Fourier transform chapter

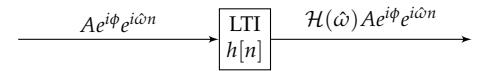


Figure 13.2: The frequency response of a discrete-time LTI system is obtained by investigating what the system does to a complex sinusoidal signal.

⁴ The normalized angular frequency is defined as $\hat{\omega} = \omega T_s$ (radians per sample), where T_s (seconds) is the sample spacing, and ω is the continuous-time angular frequency (radians per second). Remember that it is possible to convert normalized angular frequency to frequency in units of cycles per second using the formula $\omega = 2\pi f$. Which results in $f = \hat{\omega}f_s/2\pi$, where the sample-rate is $f_s = 1/T_s$. For example, this means that $\hat{\omega} = \pi$ corresponds to a frequency $f = f_s/2$.

$$y[n] = \sum_k h[k]x[n-k] \quad (13.16)$$

$$= \sum_k h[k]Ae^{i\phi}e^{i\hat{\omega}(n-k)} \quad (13.17)$$

$$= \sum_k h[k]e^{-i\hat{\omega}k}Ae^{i\phi}e^{i\hat{\omega}n} \quad (13.18)$$

$$= \underbrace{\left(\sum_k h[k]e^{-i\hat{\omega}k} \right)}_{\mathcal{H}(\hat{\omega})} \underbrace{\left(Ae^{i\phi}e^{i\hat{\omega}n} \right)}_{x[n]} \quad (13.19)$$

$$= \mathcal{H}(\hat{\omega})x[n]. \quad (13.20)$$

Here the complex-valued function $\mathcal{H}(\hat{\omega}) \in \mathbb{C}$, denotes the *discrete-time frequency response*, which is defined as:

$$\boxed{\mathcal{H}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} h[k]e^{-i\hat{\omega}k}}. \quad (13.21)$$

We'll later on show that this is a Fourier transform, the *discrete-time Fourier transform* of a discrete-time signal $h[n]$.

Magnitude and phase response

The frequency response $\mathcal{H}(\hat{\omega}) \in \mathbb{C}$ is a complex-valued function. It can be expressed in either Cartesian or polar form using Euler's formula:

$$\mathcal{H}(\hat{\omega}) = \operatorname{Re}\{\mathcal{H}(\hat{\omega})\} + i\operatorname{Im}\{\mathcal{H}(\hat{\omega})\} = |\mathcal{H}(\hat{\omega})| e^{i\angle\mathcal{H}(\hat{\omega})}. \quad (13.22)$$

This is shown in Figure 13.3.

When an LTI system is applied to a complex sinusoidal signal $x[n] = Ae^{i\phi}e^{i\hat{\omega}n}$, the effect is scaling the absolute value and shifting the phase of this signal:

$$y[n] = x[n] * h[n] \quad (13.23)$$

$$= \mathcal{H}(\hat{\omega})Ae^{i\phi}e^{i\hat{\omega}n} \quad (13.24)$$

$$= \underbrace{(A|\mathcal{H}(\hat{\omega})|)}_{A'} \underbrace{e^{i(\phi+\angle\mathcal{H}(\hat{\omega}))}}_{e^{i\phi'}} e^{i\hat{\omega}n} \quad (13.25)$$

$$= A'e^{i\phi'}e^{i\hat{\omega}n}. \quad (13.26)$$

The output signal is still a complex sinusoidal signal with the same frequency, but the amplitude A is scaled by a factor of $|\mathcal{H}(\hat{\omega})|$, and the phase is shifted by a factor $\angle\mathcal{H}(\hat{\omega})$. The magnitude of the frequency response $|\mathcal{H}(\hat{\omega})|$ is called the *magnitude response*:

$$|\mathcal{H}(\hat{\omega})| = \sqrt{\operatorname{Re}\{\mathcal{H}(\hat{\omega})\}^2 + \operatorname{Im}\{\mathcal{H}(\hat{\omega})\}^2}, \quad (13.27)$$

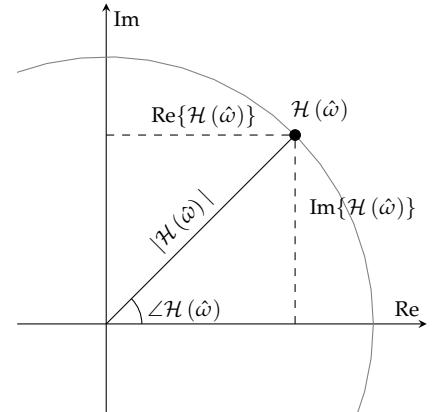


Figure 13.3: The frequency response is complex-valued, and it represents a phase shift $\angle\mathcal{H}(\hat{\omega})$ and a magnitude scaling $|\mathcal{H}(\hat{\omega})|$ introduced by the LTI system to a complex sinusoidal signal of normalized angular frequency $\hat{\omega}$.

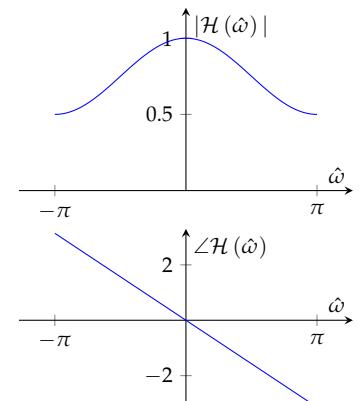


Figure 13.4: An example frequency response. Top: The magnitude response of the LTI system $|\mathcal{H}(\hat{\omega})|$. Bottom: The phase response of the system $\angle\mathcal{H}(\hat{\omega})$.

and the phase $\angle \mathcal{H}(\hat{\omega})$ is called the *phase response*

$$\angle \mathcal{H}(\hat{\omega}) = \tan^{-1} \left(\frac{\text{Im}\{\mathcal{H}(\hat{\omega})\}}{\text{Re}\{\mathcal{H}(\hat{\omega})\}} \right). \quad (13.28)$$

of the LTI system.

Figure 13.4 shows an example frequency response for a low-pass filter, which reduces the absolute value of complex sinusoidal signals with high frequencies near $\hat{\omega} = \pm\pi$. The filter also introduces a phase shift, which is described by a linear slope. We'll later see that a linear slope means that the system applies a time-shift to the input signal.

Example: Low-pass filter

Let's find the frequency response of the finite impulse response (FIR) filter, which has an impulse response defined as:

$$h[n] = \begin{cases} 1 & n = 0 \\ 2 & n = 1 \\ 1 & n = 2 \\ 0 & \text{otherwise} \end{cases}. \quad (13.29)$$

Another way we can specify the impulse response is as a sum of unit impulses:

$$h[n] = \delta[n] + 2\delta[n - 1] + \delta[n - 2] \quad (13.30)$$

The impulse response of the filter is shown in Figure 13.5.

The frequency response can be determined using Equation 13.21:

$$\mathcal{H}(\hat{\omega}) = \sum_{k=0}^2 h[k]e^{-i\hat{\omega}k} \quad (13.31)$$

$$= h[0]e^{-i\hat{\omega}0} + h[1]e^{-i\hat{\omega}1} + h[2]e^{-i\hat{\omega}2} \quad (13.32)$$

$$= 1 + 2e^{-i\hat{\omega}} + e^{-i2\hat{\omega}}. \quad (13.33)$$

We can tidy this up a bit by multiplying the equation with $1 = e^{-i\hat{\omega}}e^{i\hat{\omega}}$ so that we get a conjugate symmetric pairing of complex exponential functions, which allows us to represent $\mathcal{H}(\hat{\omega})$ conveniently in polar form:

$$\mathcal{H}(\hat{\omega}) = e^{-i\hat{\omega}} \left(e^{i\hat{\omega}} + 2 + e^{-i\hat{\omega}} \right) \quad (13.34)$$

$$= \underbrace{[2 + 2\cos(\hat{\omega})]}_{\text{magnitude}} \underbrace{e^{-i\hat{\omega}}}_{\text{phase}}. \quad (13.35)$$

The magnitude response is $|\mathcal{H}(\hat{\omega})| = 2 + 2\cos(\hat{\omega})$ and the phase response is $\angle \mathcal{H}(\hat{\omega}) = -\hat{\omega}$. These are shown in Figure 13.6.

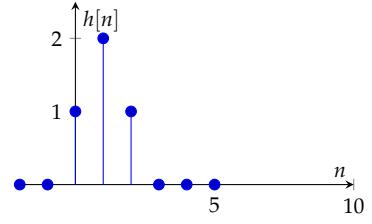


Figure 13.5: The impulse response $h[n]$ of a discrete-time low-pass filter.

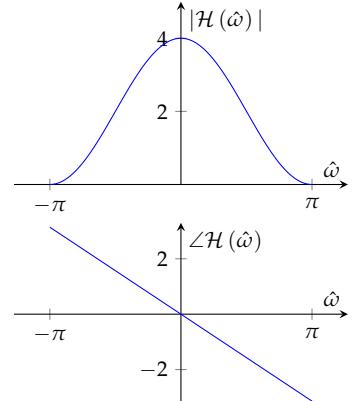


Figure 13.6: Magnitude and phase response of low pass filter $\{h[n]\}_{n=0}^2 = \{1, 2, 1\}$.

Based on the magnitude response, we can see that the filter is a *low-pass filter*. It reduces the magnitude of high frequency spectral components of the input signal. At frequency $\hat{\omega} = \pm\pi$, the output signal is completely attenuated, as $|\mathcal{H}(\pm\pi)| = 0$.

The phase response has a linear slope $\angle \mathcal{H}(\hat{\omega}) = -\hat{\omega}$. This implies that the output signal is delayed relative to the input by a constant value. Recall that a delay to a complex sinusoidal signal $x[n] = Ae^{i\phi}e^{i\hat{\omega}n}$ is:

$$x[n - n_0] = Ae^{i\phi}e^{i\hat{\omega}(n-n_0)} = e^{-i\hat{\omega}n_0}x[n]. \quad (13.36)$$

This means that a delay by one sample $n_0 = 1$ will correspond to the phase response of $\angle \mathcal{H}(\hat{\omega}) = -\hat{\omega}$.

Example: High-pass filter

A first difference system is defined as follows:

$$y[n] = x[n] - x[n - 1]. \quad (13.37)$$

This is equivalent to a FIR filter with an impulse response $h[n] = \delta[n] - \delta[n - 1]$. This is shown in Figure 13.7.

Using Equation 13.21, we obtain the frequency response:

$$\mathcal{H}(\hat{\omega}) = \sum_k h[k]e^{-i\hat{\omega}k} \quad (13.38)$$

$$= 1 - e^{-i\hat{\omega}}. \quad (13.39)$$

If we multiply this with $1 = e^{-i\frac{\hat{\omega}}{2}}e^{i\frac{\hat{\omega}}{2}}$, we obtain:

$$\mathcal{H}(\hat{\omega}) = e^{-i\frac{\hat{\omega}}{2}} \left(e^{i\frac{\hat{\omega}}{2}} - e^{-i\frac{\hat{\omega}}{2}} \right). \quad (13.40)$$

We know that $\frac{1}{2i}(e^{i\alpha} - e^{-i\alpha}) = \sin(\alpha)$, therefore:

$$\mathcal{H}(\hat{\omega}) = e^{-i\frac{\hat{\omega}}{2}} 2i \sin\left(\frac{\hat{\omega}}{2}\right). \quad (13.41)$$

We also know that $i = e^{i\frac{\pi}{2}}$, so

$$\mathcal{H}(\hat{\omega}) = 2 \sin\left(\frac{\hat{\omega}}{2}\right) e^{-i\frac{1}{2}(\hat{\omega}-\pi)}. \quad (13.42)$$

In order to inspect the magnitude and phase response, we need to obtain the frequency response in the following form:

$$\mathcal{H}(\hat{\omega}) = |\mathcal{H}(\hat{\omega})| e^{i\angle \mathcal{H}(\hat{\omega})}. \quad (13.43)$$

We can do this using a phase shift by π to flip the phase $-1 = e^{-i\pi}$. This allows us to take into account the fact that $\sin(\hat{\omega}/2)$ is negative when $-\pi < \hat{\omega} < 0$. The frequency response can now be written as:

$$\mathcal{H}(\hat{\omega}) = \begin{cases} 2 \left| \sin\left(\frac{\hat{\omega}}{2}\right) \right| e^{-i\frac{1}{2}(\hat{\omega}-\pi)} & \text{when } 0 < \hat{\omega} < \pi \\ 2 \left| \sin\left(\frac{\hat{\omega}}{2}\right) \right| e^{-i\frac{1}{2}(\hat{\omega}+\pi)} & \text{when } -\pi < \hat{\omega} \leq 0 \end{cases}. \quad (13.44)$$

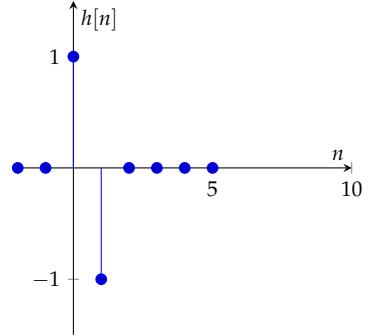


Figure 13.7: The impulse response of a high-pass filter $h[n] = \delta[n] - \delta[n - 1]$.

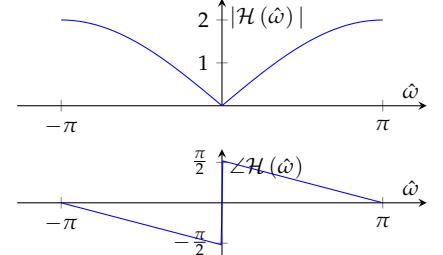


Figure 13.8: Top: The magnitude response of the high-pass filter. Bottom: The phase response of the same filter.

The magnitude response is thus:

$$|\mathcal{H}(\hat{\omega})| = 2 \left| \sin\left(\frac{\hat{\omega}}{2}\right) \right|, \quad (13.45)$$

and the phase response is:

$$\angle \mathcal{H}(\hat{\omega}) = \begin{cases} -\frac{1}{2}\hat{\omega} + \frac{\pi}{2} & \text{when } 0 < \hat{\omega} < \pi \\ -\frac{1}{2}\hat{\omega} - \frac{\pi}{2} & \text{when } -\pi < \hat{\omega} \leq 0 \end{cases}. \quad (13.46)$$

The magnitude and phase response is shown in Figure 13.8. It completely removes the zero-frequency component of the signal, and reduces the amplitudes of low frequency spectral components. High frequency spectral components of the input signal will be amplified. For input signal spectral components with frequencies at $\hat{\omega} = \pm\pi$, the filter output signal spectral components will have twice the original magnitude.

This type of filter is called a *high-pass filter*, as it attenuates low frequency spectral components relative to high frequency spectral components.

Example: band-pass filter

Let's now investigate the frequency response of the following filter:

$$y[n] = 0.5x[n+1] - 0.5x[n-1]. \quad (13.47)$$

The impulse response $h[n] = 0.5\delta[n+1] - 0.5\delta[n-1]$ of this system is shown in Figure 13.9.

Using Equation 13.21, we obtain the frequency response:

$$\mathcal{H}(\hat{\omega}) = \sum_k h[k]e^{-i\hat{\omega}k} \quad (13.48)$$

$$= 0.5e^{i\hat{\omega}} - 0.5e^{-i\hat{\omega}} \quad (13.49)$$

$$= i \sin(\hat{\omega}). \quad (13.50)$$

If we wanted to investigate this analytically, we could use the type of analysis that did in the previous example. We'll write a little Python script instead to plot the magnitude and phase response. This is shown in Listing 13.1.

The plot produced is shown in Figure 13.10. From the magnitude response plot, we can see that this filter reduces the magnitudes of high and low frequency spectral components of the input signal near $\hat{\omega} = 0$ and $\hat{\omega} = \pm\pi$. However, the spectral components with frequencies near $\hat{\omega} = \pm\frac{\pi}{2}$ are unchanged in magnitude. The phase response is flat, except for a discontinuity at 0, where the phase switches from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ to account for the sign of the sin function.

This type of filter that only lets through spectral components of the signal within a range of frequencies is called a *band-pass filter*.

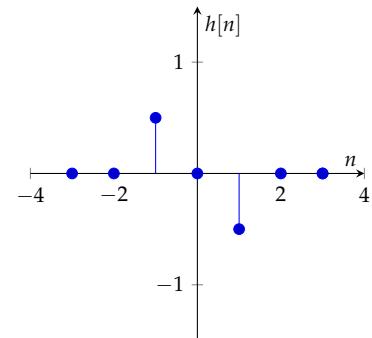


Figure 13.9: The impulse response of a simple band-pass filter.

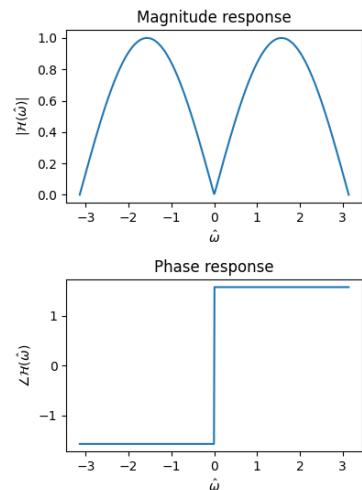


Figure 13.10: The frequency response of the band-pass filter $h[n] = 0.5\delta[n+1] - 0.5\delta[n-1]$. Top: the magnitude response. Bottom: The phase response.

```

import matplotlib.pyplot as plt
import numpy as np

# A sweep of -\pi to \pi in normalized angular frequency.
omhat = np.linspace(-np.pi, np.pi, num=500)
H = 0.5*np.exp(1j*omhat) - 0.5*np.exp(-1j*omhat)

plt.figure(figsize=(0.7*6, 0.7*8))
plt.subplot(211)
plt.plot(omhat, np.abs(H))
plt.title("Magnitude response")
plt.ylabel(r"\|\mathcal{H}(\hat{\omega})\|")
plt.xlabel(r"\hat{\omega}")
plt.subplot(212)
plt.title("Phase response")
plt.plot(omhat, np.angle(H))
plt.xlabel(r"\hat{\omega}")
plt.ylabel(r"\angle \mathcal{H}(\hat{\omega})")
plt.tight_layout()
plt.savefig("bpf_fresp.png")
plt.show()

```

Listing 13.1: 019_frequency_response/example_bpf.py

Example: Delay filter

A delay system is defined as:

$$y[n] = x[n - n_0]. \quad (13.51)$$

This has a unit impulse $h[n] = \delta[n - n_0]$ (shown in Figure 13.11).

Using Equation 13.21, we obtain the following frequency response:

$$\mathcal{H}(\hat{\omega}) = \sum_k h[k] e^{-i\hat{\omega}k} \quad (13.52)$$

$$= \sum_k \delta[k - n_0] e^{-i\hat{\omega}k} \quad (13.53)$$

$$= e^{-i\hat{\omega}n_0}. \quad (13.54)$$

The magnitude response is $|\mathcal{H}(\hat{\omega})| = 1$ and the phase response is $\angle \mathcal{H}(\hat{\omega}) = -\hat{\omega}n_0$. This is shown in Figure 13.12. This filter does not modify the magnitude of any spectral components. However, it applies a phase shift to spectral components, which results in a time delay.

This may seem trivial now, but the frequency domain representation for a time-shift will later be useful; when defining more advanced, fractional sample, time-shift filters for discrete-time signals. In this case, $n_0 \in \mathbb{R}$ is not an integer. We'll need to define the inverse discrete-time Fourier transform in order to derive the filter impulse response in this case.

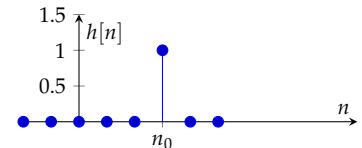


Figure 13.11: The impulse response of a time-shift system.

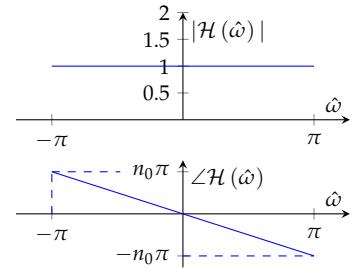


Figure 13.12: The frequency response of a time-delay filter.

Example: Continuous-time low-pass filter

Consider a continuous-time LTI system, which uniformly averages the input signal around a region of length T :

$$y(t) = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} x(t - \tau) d\tau. \quad (13.55)$$

This has the following an impulse response:

$$h(t) = \frac{1}{T} [u(t + T/2) - u(t - T/2)]. \quad (13.56)$$

This is the familiar rectangular function. Using Equation 13.10, we obtain the following frequency response⁵:

$$\mathcal{H}(\omega) = \frac{2 \sin(\omega T/2)}{\omega T}. \quad (13.57)$$

Note that this is a continuous-time frequency response. Our frequency variable is angular frequency (radians per second). The frequency response is a sinc-function, as shown in Figure 13.13. This is a low-pass filter, as this filter reduces the magnitude of spectral components of the input signal $x(t)$ with higher frequencies.

Example: Time symmetric running average filter

Let's now look at a discrete-time version of the previous continuous-time filter, the time symmetric running average system. This system averages $M = 2N + 1$ values of the input signal $x[n]$:

$$y[n] = \frac{1}{M} \sum_{k=-N}^N x[n - k]. \quad (13.58)$$

We can easily see that this is an LTI system with an impulse response:

$$h[n] = \frac{1}{M} \sum_{k=-N}^N \delta[n - k]. \quad (13.59)$$

The impulse response $h[n]$ of the system is shown in Figure 13.14.

This type of filter is often used to smooth noisy signals. It will also later on play an important role in determining the frequency response of a discrete Fourier transform.

Using Equation 13.21, we can find the frequency response of this filter:

$$\mathcal{H}(\hat{\omega}) = \sum_{k=-N}^N \frac{1}{M} e^{-i\hat{\omega}k}. \quad (13.60)$$

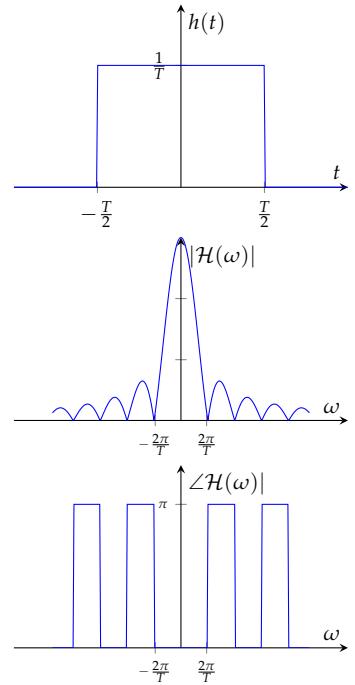


Figure 13.13: Top: the impulse response of an averaging filter. Middle: The magnitude response. Bottom: The phase response.

⁵ See the Fourier transform chapter for derivation of Equation 13.57.

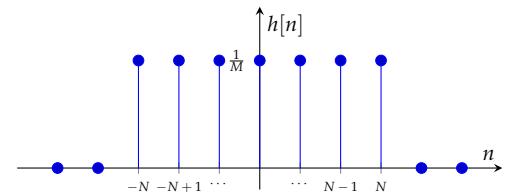


Figure 13.14: The impulse response of a discrete-time running average filter.

Let's set $c = 1/M$, $\alpha = e^{i\hat{\omega}}$, and $S = \mathcal{H}(\hat{\omega})$. By doing some algebraic manipulations, we can find a closed form solution:

$$S = c\alpha^{-N} + c\alpha^{-N+1} + \cdots + c + \cdots + c\alpha^{N-1} + c\alpha^N \quad (13.61)$$

$$\alpha S = c\alpha^{-N+1} + c\alpha^{-N+2} + \cdots + c + \cdots + c\alpha^N + c\alpha^{N+1} \quad (13.62)$$

$$S - \alpha S = c\alpha^{-N} - c\alpha^{N+1} \quad (13.63)$$

$$S(1 - \alpha) = c(\alpha^{-N} - \alpha^{N+1}) \quad (13.64)$$

$$S = c \frac{\alpha^{-N} - \alpha^{N+1}}{1 - \alpha}. \quad (13.65)$$

This is known as the closed form solution to a *geometric series*. When substituting back the values of c , α , and S we obtain:

$$\mathcal{H}(\hat{\omega}) = \frac{1}{M} \frac{e^{-i\hat{\omega}N} - e^{i\hat{\omega}(N+1)}}{1 - e^{i\hat{\omega}}}. \quad (13.66)$$

We can once again use the trick of multiplying the complex exponentials with $1 = e^{i\hat{\omega}/2}e^{-i\hat{\omega}/2}$ on the numerator and denominator.

This allows us to create conjugate pairings of complex exponentials:

$$\mathcal{H}(\hat{\omega}) = \frac{1}{M} \frac{e^{-i\hat{\omega}N} - e^{i\hat{\omega}(N+1)}}{1 - e^{i\hat{\omega}}} \quad (13.67)$$

$$= \frac{1}{M} \frac{(e^{-i\hat{\omega}N} - e^{i\hat{\omega}(N+1)})[e^{i\hat{\omega}/2}e^{-i\hat{\omega}/2}]}{(1 - e^{i\hat{\omega}})[e^{i\hat{\omega}/2}e^{-i\hat{\omega}/2}]} \quad (13.68)$$

$$= \frac{1}{M} \frac{e^{i\hat{\omega}/2} (e^{-i\hat{\omega}(N+1/2)} - e^{i\hat{\omega}(N+1/2)})}{e^{i\hat{\omega}/2} (e^{-i\hat{\omega}/2} - e^{i\hat{\omega}/2})} \quad (13.69)$$

$$= \frac{1}{M} \frac{e^{-i\hat{\omega}(N+1/2)} - e^{i\hat{\omega}(N+1/2)}}{e^{-i\hat{\omega}/2} - e^{i\hat{\omega}/2}}, \quad (13.70)$$

which then allows us to use $\sin(\theta) = \frac{1}{2i}(e^{i\theta} - e^{-i\theta})$ to relate the symmetric pairs to sin functions:

$$\mathcal{H}(\hat{\omega}) = \frac{1}{M} \frac{2i \sin(\hat{\omega}(N+1/2))}{\sin(\hat{\omega}/2)} = D_M(\hat{\omega}). \quad (13.71)$$

The function $D_M(\hat{\omega})$ is defined as (recall that $M = 2N + 1$):

$$D_M(\hat{\omega}) = \frac{1}{M} \frac{\sin(\hat{\omega}M/2)}{\sin(\hat{\omega}/2)}. \quad (13.72)$$

It is sometimes referred to as the *periodic sinc* function, and sometimes as the *Dirichlet kernel*⁶.

Equation 13.72 may at first appear problematic at $\hat{\omega} = 0$, which results in $D_M(0) = 0/0$. This can be resolved using L'Hôpital's rule:

$$\lim_{x \rightarrow c} \frac{f(x)}{g(x)} = \lim_{x \rightarrow c} \frac{f'(x)}{g'(x)}. \quad (13.73)$$

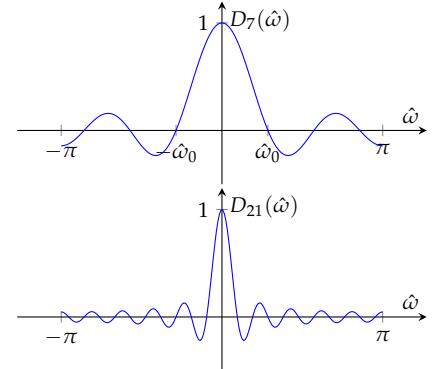


Figure 13.15: The Dirichlet kernel $D_7(\hat{\omega})$ and $D_{21}(\hat{\omega})$. The first zero-crossings $\pm\hat{\omega}_0$ are depicted for $D_7(\hat{\omega})$, which indicate the bandwidth of the filter.

⁶ Note that there are several conflicting definitions of the Dirichlet kernel. I am using the one consistent with the Scipy implementation `scipy.special.diric(x,n)`.

From this it follows that:

$$D_M(0) = 1. \quad (13.74)$$

Figure 13.15 shows a plot of the Dirichlet kernel for $M = 7$ and $M = 21$, which shows that the width of the main lobe becomes more narrow as M is increased. The location of the first zero-crossing of $D_M(\hat{\omega}_0) = 0$ is at $\hat{\omega}_0 = 2\pi/M$. This is a demonstration of *time-frequency ambiguity*, i.e., the inverse relationship between the width of a signal in time domain and frequency domain.

It can be shown that the Dirichlet kernel also arises from the Fourier series representation of a Dirac comb signal with period 2π :

$$D'(\hat{\omega}) = \sum_{k=-\infty}^{\infty} \delta(\hat{\omega} - 2\pi k), \quad (13.75)$$

which has the following Fourier series representation using $M = 2N + 1$ terms:

$$D'_M(\hat{\omega}) = \frac{1}{2\pi} \sum_{k=-N}^N e^{i\hat{\omega}k} = \frac{1}{2\pi} \frac{\sin(\hat{\omega}M/2)}{\sin(\hat{\omega}/2)}. \quad (13.76)$$

Note that $D'_M(\hat{\omega})$ and $D_M(\hat{\omega})$ are up to a constant the same. The proof of Equation 13.76 is essentially the same as the derivation of the running mean filter frequency response shown above⁷. As $N \rightarrow \infty$, the location of the first zero crossing approaches $\hat{\omega} = 0$. This means that the bandwidth of the filter approaches zero and passes through only signals with frequencies $\hat{\omega} = 2\pi k$.

From Equation 13.75 it is easy to see that the Dirichlet kernel is 2π -periodic. This is a general property that applies to the frequency response of any discrete-time LTI system, and is due to discretization of the signal.

There is a short Python program in Listing 13.2, which can be used to plot the magnitude and phase response of the running average filter. The output of this program is shown in Figure 13.17.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as s

def frequency_response(h, n_value, omhat):
    H = np.zeros(len(omhat), dtype=np.complex64)
    for idx in range(len(h)):
        H += h[idx]*np.exp(-1j*n_value[idx]*omhat)
    return (H)

# Impulse response of an averaging filter.
# M = 2N + 1
h = np.repeat(1.0/21.0, 21)
```

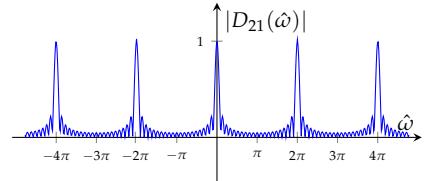


Figure 13.16: The Dirichlet kernel is 2π -periodic, which is a general property of discrete-time frequency responses.

⁷ The Fourier series representation for a Dirac comb signal was discussed in the Fourier series chapter.

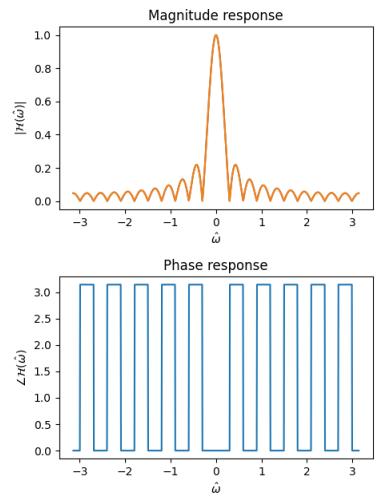


Figure 13.17: The output of the program in Listing 13.2, which shows the magnitude and phase response of the 21-point running average filter. As one can see of the plot, the two methods have equivalent magnitude response.

```
print(len(h))
omhat = np.linspace(-np.pi, np.pi, num=1000)
# Integers -N to N.
H = frequency_response(h, np.arange(-10, 10+1), omhat)

# Alternative way using the dirichlet kernel:
# D_21(\omega)
D = s.diric(omhat, 21)

plt.figure(figsize=(0.8*6, 0.8*8))
plt.subplot(211)
plt.plot(omhat, np.abs(D))
plt.plot(omhat, np.abs(H))
plt.title("Magnitude response")
plt.ylabel(r"\|\mathcal{H}(\hat{\omega})\|")
plt.xlabel(r"\hat{\omega}")
plt.subplot(212)
plt.title("Phase response")
plt.plot(omhat, np.angle(D))
plt.xlabel(r"\hat{\omega}")
plt.ylabel(r"\angle\mathcal{H}(\hat{\omega})")
plt.tight_layout()
plt.savefig("dirichlet21.png")
plt.show()
```

Listing 13.2: 019_frequency_response/dirichlet.py

Exercises: Frequency Response

1. The impulse response of an LTI system is defined as:

$$h(t) = \frac{42 \sin(\omega_c t)}{\pi t}. \quad (13.77)$$

The input to the LTI system is a periodic signal with a fundamental period $T = 1$:

$$x(t) = \sum_{n=-\infty}^{\infty} \delta(t - n). \quad (13.78)$$

When the signal $x(t)$ is fed into an LTI system, the output signal is given by a convolution of the input signal with the output signal:

$$y(t) = \int_{-\infty}^{\infty} h(t - \tau)x(\tau)d\tau. \quad (13.79)$$

- a) Determine the Fourier transform $\hat{x}(\omega)$ of the input signal. Plot $\hat{x}(\omega)$ over the range of angular frequencies $-7\pi < \omega < 7\pi$.

Hint: It might help if you start by expressing the signal $x(t)$ using a Fourier series representation first, and then apply Equation 10.52.

- b) Determine the frequency response $\mathcal{H}(\omega)$ of the LTI system. In other words, Fourier transform the impulse response signal given in Equation 13.77. It should be a fairly simple function. Make a plot of $|\mathcal{H}(\omega)|$ on the same graph as $\hat{x}(\omega)$ using $\omega_c = 5\pi$.
- c) Use your plot in b) to determine $y(t)$, the output of the LTI system characterized using $h(t)$ when $\omega_c = 5.5\pi$. Hint: use property that convolution in time domain is multiplication in frequency domain.
- d) What values of ω_c will result in a non-zero constant output $y(t) = c$. What is the constant c ?

2. A running average filter is defined as:

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k]. \quad (13.80)$$

This filter only depends on the current and past values of the input, and can thus be implemented in a real-time system. Assume that M is an odd number.

- a) What is the impulse response $h_M[n]$ of the system described in Equation 13.80?
- b) What is the frequency response $\mathcal{H}_M(\omega)$ of the system described in Equation 13.80?

- c) What is the frequency response $\mathcal{H}_\tau(\hat{\omega})$ of a time-shift system
 $h_\tau[n] = \delta[n - \tau]$?
- d) I recommend that you do this task on a computer. Plot the squared magnitude response $|\mathcal{H}_M(\hat{\omega})|^2$ between $-\pi < \hat{\omega} < \pi$. Assume that $M = 11$. Compare this with a plot of the squared magnitude response of the time-symmetric running average filter $|D_M(\hat{\omega})|^2$ of the same length. It is defined in Equation 13.72. They should be the same. If you are not sure why, carry on with the next tasks.
- e) Show that the frequency response $\mathcal{H}_M(\hat{\omega})$ be obtained by multiplying the frequency response of the time symmetric running average filter given in Equation 13.72 with the frequency response of the time shift system with a suitable value of τ :

$$\mathcal{H}_M(\hat{\omega}) = D_M(\hat{\omega})\mathcal{H}_\tau(\hat{\omega})? \quad (13.81)$$

Hint: Look at the derivation of $D_M(\hat{\omega})$.

- f) Explain why $|\mathcal{H}_M(\hat{\omega})|^2 = |D_M(\hat{\omega})|^2$. Hint: $|z|^2 = zz^*$.

3. It is possible to numerically estimate the second time derivative of a discretized signal using an FIR filter with the following filter coefficients:

$$h[n] = T_s^{-2}\delta[n+1] - 2T_s^{-2}\delta[n] + T_s^{-2}\delta[n-1] \quad (13.82)$$

where T_s is the sample spacing. This can be motivated by the following definition of a second time-derivative:

$$\frac{d^2x}{dt^2} = \lim_{T_s \rightarrow 0} \frac{x(t+T_s) - 2x(t) + x(t-T_s)}{T_s^2} \quad (13.83)$$

which is numerically evaluated with a finite difference T_s .

If a signal of the form $x[n] = Ae^{i\phi}e^{i\hat{\omega}n}$ is fed into the FIR filter described by an impulse response $h[n]$, the output signal will be of the form $y[n] = A'e^{i\phi'}e^{i\hat{\omega}n}$. Here $A, A' \in \mathbb{R}_{\geq 0}$ and $\hat{\omega}, \phi, \phi' \in \mathbb{R}$.

- a) Show that the frequency response of this filter is

$$\mathcal{H}(\hat{\omega}) = \frac{2}{T_s^2}[\cos(\hat{\omega}) - 1] \quad (13.84)$$

- b) Make a plot of the magnitude response $|\mathcal{H}(\hat{\omega})|$ between normalized angular frequencies $-\pi < \hat{\omega} < \pi$ (radians per sample).
- c) What is the phase response $\angle \mathcal{H}(\hat{\omega})$ of this system?
- d) Is this filter a high-pass or a low-pass filter?

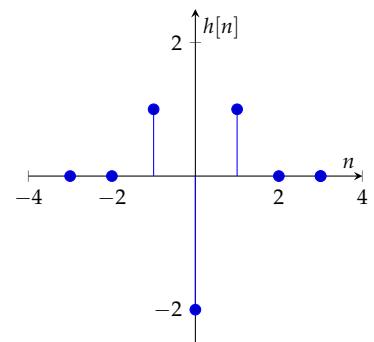


Figure 13.18: The impulse response of a second order derivative estimator filter with $T_s = 1$.

- e) What is A' when $\hat{\omega} = 0$?
- f) What is A' and ϕ' when $\hat{\omega} = \pi/2$ and $\hat{\omega} = \pi$?
- g) Show that for continuous-time signals, the second time-derivative operation $\mathcal{T}\{x(t)\} = \frac{d^2}{dt^2}x(t)$ has a frequency response of the form $\mathcal{H}(\omega) = -\omega^2$.
- h) A plot of the continuous-time frequency response found in g) and the discrete-time frequency response found in a) is shown in Figure 13.19. The sample spacing is assumed to be $T_s = 1$. With this sample spacing, both ω and $\hat{\omega}$ will have the same scale ($\omega = 1$ rad/s will correspond to $\hat{\omega} = 1$ rad/sample). What does the plot tell you about how well the FIR filter in Equation 13.82 approximates the second time derivative operation for continuous-time complex sinusoidal signals as a function of frequency?

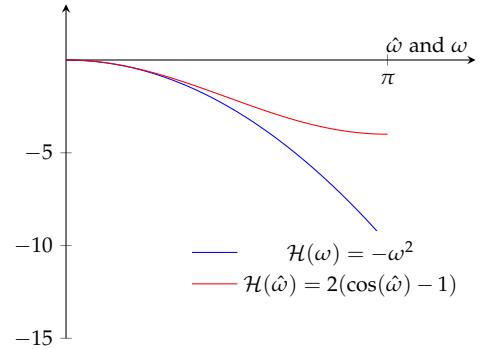


Figure 13.19: The frequency response of a continuous-time second time-derivative operation and a numerical estimator of the second time-derivative in discrete-time.

Suggested solutions: Frequency Response

1. The impulse response for an LTI system is:

$$h(t) = \frac{42 \sin(\omega_c t)}{\pi t}.$$

Let the input to the LTI system be a periodic signal with period $T = 1$ of the form:

$$x(t) = \sum_{n=-\infty}^{\infty} \delta(t - n).$$

- a) Given $x(t)$ we find the Fourier transform as follows, firstly, we know that the Dirac comb can be expressed as a Fourier series of the form:

$$x(t) = \frac{1}{T} \sum_{n=-\infty}^{\infty} e^{i \frac{2\pi n t}{T}}.$$

In this case, we have $T = 1$, hence:

$$x(t) = \sum_{n=-\infty}^{\infty} e^{i 2\pi n t}.$$

Thus, the Fourier series coefficients are $c_k = 1$ for all $k \in \mathbb{Z}$.

Then the Fourier transform is:

$$\hat{x}(\omega) = 2\pi \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n).$$

A plot of the spectrum for $\hat{x}(\omega)$ is shown in Figure 13.20 in red.

- b) The frequency response function is defined by:

$$\mathcal{H}(\omega) = \mathcal{F}\{h(t)\} = \int_{-\infty}^{\infty} \frac{42 \sin(\omega_c t)}{\pi t} e^{-i\omega t} dt = 42[u(\omega + \omega_c) - u(\omega - \omega_c)].$$

Figure 13.20 shows the frequency response in green.

- c) The output of the LTI system can be found by convolution in time domain, that is $y(t) = h(t) * x(t)$, thus, in frequency domain we have multiplication by the convolution theorem. By Figure 13.20 the frequencies inside $(-5.5\pi, 5.5\pi)$ remain after multiplication, but the frequencies outside $(-5.5\pi, 5.5\pi)$ gets mapped to 0.
- d) If $y(t) = c$ then $\hat{y}(\omega) = 2\pi c \delta(\omega)$ so the frequency response must be such that:

$$2\pi c \delta(\omega) = \hat{x}(\omega) \mathcal{H}(\omega) = 2\pi \sum_{n=-\infty}^{\infty} \delta(\omega - 2\pi n) 42[u(\omega + \omega_c) - u(\omega - \omega_c)] = 2\pi 42 \delta(\omega),$$

the only way for this to work is that $|\omega_c| \leq \pi$ giving $c = 42$.

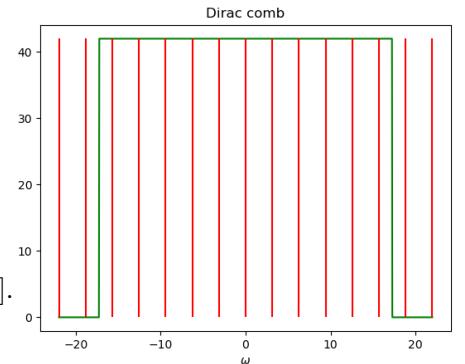


Figure 13.20: Spectrum for the Dirac comb between $-7\pi < \omega < 7\pi$.

2. A running average filter is defined as:

$$y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n-k].$$

We'll assume that M is an odd number.

- a) The impulse response function is obtained by feeding a Dirac delta function into the LTI system, so if $y[n] = \mathcal{T}\{x[n]\}$, then $h[n] = \mathcal{T}\{\delta[n]\}$. Doing this, we find that:

$$h[n] = \frac{1}{M} \sum_{k=0}^{M-1} \delta[n-k].$$

- b) The frequency response is the DTFT of $h[n]$, which gives:

$$\begin{aligned} \mathcal{H}(\hat{\omega}) &= \sum_{k=0}^{M-1} \frac{1}{M} e^{-i\hat{\omega}k} = \frac{1}{M} \frac{1 - e^{-i\hat{\omega}M}}{1 - e^{-i\hat{\omega}}} = \frac{1}{M} \left(\frac{e^{-i\hat{\omega}M/2}(e^{i\hat{\omega}M/2} - e^{-i\hat{\omega}M/2})}{e^{-i\hat{\omega}/2}(e^{i\hat{\omega}/2} - e^{-i\hat{\omega}/2})} \right), \\ &= \underbrace{\frac{1}{M} \frac{\sin(\hat{\omega}M/2)}{\sin(\hat{\omega}/2)}}_{D_M(\hat{\omega})} e^{-i\hat{\omega}(M-1)/2}. \end{aligned}$$

The sum, is evaluated using known formulas for geometric sums.

- c) If we have a system with impulse response $h_\tau[n] = \delta[n - \tau]$, then the frequency response is:

$$\mathcal{H}_\tau(\hat{\omega}) = \sum_{k=-\infty}^{\infty} h[k]e^{-i\hat{\omega}k} = \sum_{k=-\infty}^{\infty} \delta[k - \tau]e^{-i\hat{\omega}k} = e^{-i\hat{\omega}\tau}.$$

- d) The following code can be used to make the plot shown in

Figure 13.21:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as dd

# Partition the interval (-pi,pi) into num points.
om = np.linspace(-np.pi, np.pi, num=1000)

# Use M = 11 for the Dirichlet kernel plot.
M = 11

# Plot the two functions in the same coordinate system.
plt.plot(om, np.abs(dd.diric(om, n=M))**2, label=r"\$|D_{11}(\hat{\omega})|^2\$")
plt.plot(om, np.abs(np.exp(1j*om*(M-1)/2)*dd.diric(om, n=M))**2, label=r"\$|H_{11}(\hat{\omega})|^2\$")
plt.xlabel(r"\$\hat{\omega}\$")
plt.legend()
# Call this if needed.
# plt.plot()
```

Listing 13.3: Plotting the frequency response code

As expected, the functions are identical.

- e) The frequency response function was derived in b), and it has the form:

$$\mathcal{H}(\hat{\omega}) = D_M(\hat{\omega})\mathcal{H}_\tau(\hat{\omega}),$$

where $\mathcal{H}_\tau(\hat{\omega})$ is a time-shift system with $\tau = (M - 1)/2$.

- f) Claim: $|\mathcal{H}(\hat{\omega})|^2 = |D_M(\hat{\omega})|^2$.

Proof. A simple computation gives:

$$\begin{aligned} |\mathcal{H}(\hat{\omega})|^2 &= \mathcal{H}(\hat{\omega})\mathcal{H}^*(\hat{\omega}) = (D_M(\hat{\omega})\mathcal{H}_\tau(\hat{\omega}))(D_M(\hat{\omega})\mathcal{H}_\tau(\hat{\omega}))^*, \\ &= |D_M(\hat{\omega})|^2 \underbrace{e^{-i\hat{\omega}\tau} e^{i\hat{\omega}\tau}}_1 = |D_M(\hat{\omega})|^2, \end{aligned}$$

as claimed. \square

In conclusion, both running average filters produce the same magnitude response, but one of them induces a time-shift while the other doesn't.

3. Consider an FIR filter approximation to the second derivative of the form:

$$h[n] = T_s^{-2}\delta[n+1] - 2T_s^{-2}\delta[n] + T_s^{-2}\delta[n-1].$$

- a) The frequency response can be found by the DTFT (Equation 13.10) as:

$$\mathcal{H}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} h[k]e^{-i\hat{\omega}k},$$

for this LTI system, we get:

$$\mathcal{H}(\hat{\omega}) = T_s^{-2}e^{i\hat{\omega}} - 2T_s^{-2} + T_s^{-2}e^{-i\hat{\omega}} = -2T_s^{-2} + T_s^{-2}2\cos(\hat{\omega}) = \frac{2}{T_s^2}(\cos(\hat{\omega}) - 1).$$

- b) The code shown in Listing 13.4 can be used to plot the spectral response.

```
import matplotlib.pyplot as plt
import numpy as np

# Plot in the principal spectrum (-pi, pi).
om = np.linspace(-np.pi, np.pi, num=1000)

# For simplicity take Ts as 1.
Ts = 1

def freq_resp(om: np.ndarray) -> np.ndarray:
    """Definition of the frequency response function."""
    return 2/(Ts**2)*(np.cos(om)-1)

# Plot the frequency response function over (-pi, pi).
```

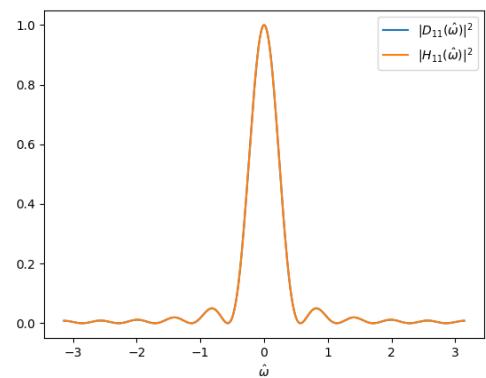


Figure 13.21: Comparison of frequency responses.

```

plt.plot(om, np.abs(freq_resp(om)), label=r"\mathcal{H}(\hat{\omega})")
plt.legend()
# Call this if needed.
# plt.plot()

```

Listing 13.4: Frequency response for finite difference

Output of the code is shown in Figure 13.22.

- c) In this case, the frequency response function is real, so the phase response is 0. Meaning, there is no delay applied to the signal by this filter.
- d) By looking at the plot of $|H(\hat{\omega})|$, we see that the filter is a high-pass filter, as the frequencies around 0 are greatly reduced.
- e) Inspecting the plot, we see that for $\hat{\omega} = 0$, the amplitude of the output signal will be $A' = 0$. The reason is that in frequency domain we have:

$$\hat{y}(\hat{\omega}) = \mathcal{H}(\hat{\omega})\hat{x}(\hat{\omega}).$$

- f) The phase is not changed by the filter, so $\phi' = \phi$. For $\hat{\omega} = \pi/2$ the amplitude is doubled, so $A' = 2A$, while for $\hat{\omega} = \pi$ the amplitude is scaled by 4, giving $A' = 4A$. This can be seen from the plot in Figure 13.22 or by inserting the values into the frequency response function.
- g) The system $y(t) = \mathcal{T}\{x(t)\} = \frac{d^2}{dt^2}x(t)$ is a continuous-time LTI system, so:

$$y(t) = \mathcal{H}(\omega)x(t).$$

In particular, if we take $x(t) = Ae^{i\phi}e^{i\omega t}$, then:

$$y(t) = \frac{d^2}{dt^2}(Ae^{i\phi}e^{i\omega t}) = Ae^{i\phi}(i\omega)^2 e^{i\omega t} = -Ae^{i\phi}\omega^2 e^{i\omega t} = -\omega^2(Ae^{i\phi}e^{i\omega t}) = \mathcal{H}(\omega)x(t),$$

thus $\mathcal{H}(\omega) = -\omega^2$.

- h) Comparing the plots, we conclude that the approximation is only good for small angular frequencies in the range $0 < \omega < \pi/2$.

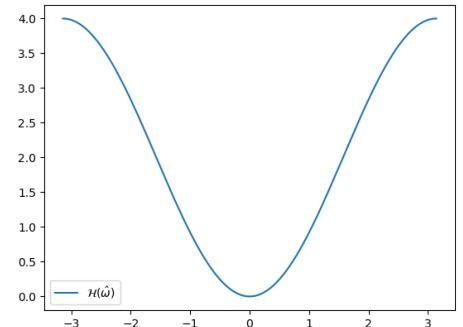


Figure 13.22: Output of Listing 13.4

14

Programming Assignment 2

This programming assignment deals with deconvolution of transmit signals for an *acoustic sounder* or a *sonar*. A sonar is a device that measures the amplitude of sound waves that are scattered from objects at different propagation distances between the acoustic wave transmitter and receiver. These devices are used, e.g., in cars to assist with parking, in ships to measure the water depth, and for non-invasive medical examinations. Sonars are also closely related with radars, as they share many of the same signal processing concepts. The main difference is that radars use electromagnetic waves instead of acoustic waves.

In order to implement a simple sonar, I have used a loudspeaker and a microphone connected to a sound card to transmit and receive acoustic signals. The sampling rate of the audio recording is $f_s = 44.1 \cdot 10^3$ hertz, which results in a sample-spacing of $T_s = 2.2676 \cdot 10^{-5}$ s. The loudspeaker and the microphone are spaced apart by approximately 30 cm.

If we ignore measurement errors, the acoustic signal can be modeled using a discrete-time convolution equation:

$$m[n] = \sum_{r=0}^{R-1} h[r]x[n-r] = h[n] * x[n] \quad (14.1)$$

In this equation $m[n]$ is the signal measured with a microphone, $x[n]$ is the signal transmitted by the loudspeaker, and $h[n]$ is the amplitude of the scattered signal as a function of range. The signal $h[n]$ is also the impulse response of the space that the sonar is measuring. The convolution equation can be motivated with a range-time diagram that models the propagation of acoustic signals as a function of range and time. Such a diagram is shown in Figure 14.2.

Modern sonars often use long waveforms in order to compress more energy into the transmitted pulse without increasing the amount of peak power. Longer waveforms also make it easier to separate multiple different transmitters from one another, reducing



Figure 14.1: Bats use chirp-like ultrasound signals to sense their surroundings. Image: David Dennis.

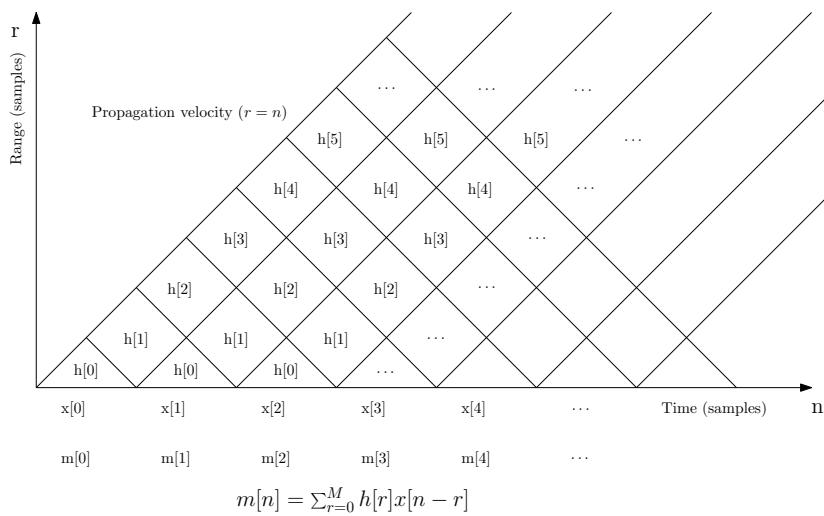


Figure 14.2: A range-time diagram depicting the relationship between a transmitted signal and a scattered signal.

interference.

One of the main objectives of a sonar measurement is to recover the impulse response $h[n]$ from measurements $m[n]$. This is trivial when the transmit pulse is a unit impulse $x[n] = A\delta[n]$. With longer transmit signals, we need to deconvolve the transmitted signal. This is typically achieved by convolving the received signal $m[n]$ with a deconvolution filter $\lambda[n]$:

$$\lambda[n] * m[n] = \lambda[n] * x[n] * h[n]. \quad (14.2)$$

The deconvolution filter needs to have the following property¹:

$$\lambda[n] * x[n] \approx A\delta[n], \quad (14.3)$$

where A is a constant. In the case of chirp-like signals, an often used deconvolution filter is the time reversed chirp signal:

$$\lambda[n] = x[-n + n_0], \quad (14.4)$$

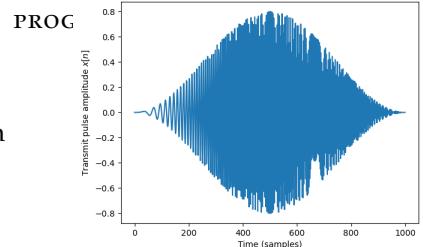
where n_0 is a suitable time shift.

For this assignment, I have used an amplitude tapered chirp-like waveform with a linearly increasing frequency of the following form:

$$x[n] = \begin{cases} \sin^2\left(\frac{\pi n}{N}\right) \sin(2\pi\beta T_s^2 n^2) & \text{when } 0 \leq n < N \\ 0 & \text{otherwise} \end{cases} \quad (14.5)$$

Here N is the integer length of the chirp pulse in samples and $\beta = \frac{1}{2NT_s}f_{\max}$ is the chirp-rate, which determines how fast the instantaneous frequency of the signal increases as a function of time, with f_{\max} the maximum frequency used of the chirp (in hertz) at

¹ One way to determine $\lambda[n]$ is to solve for it in frequency domain, but this is beyond the scope of this exercise.



the end of the transmit pulse. Figure 14.3 shows this waveform with $N = 1000$ and $\beta = 486202.5 \text{ s}^{-2}$.

You will need to download two data files for this task:

- <http://kaira.uit.no/juha/chirp.bin>
- http://kaira.uit.no/juha/sonar_meas.bin

The first one contains the transmitted chirp waveform $x[n]$ and the second one contains a microphone recording during a sonar experiment $m[n]$. Note that if you do not want to use the provided recordings, feel free to use your own sonar setup. You will need a microphone and a loudspeaker, and some software that will transmit the chirp waveform in a loop. You can also borrow my equipment in order to make your own measurements.

You can read these signals into a computer as follows:

```
import matplotlib.pyplot as plt
import numpy as np

# Read chirp waveform.
chirp = np.fromfile("chirp.bin", dtype=np.float32)
# Read microphone recording.
m = np.fromfile("sonar_meas.bin", dtype=np.float32)

plt.plot(chirp)
plt.xlabel("Time (samples)")
plt.ylabel("Transmitted signal $x[n]$")
plt.show()

# Plot the first three interpulse periods.
plt.plot(m[30000])
plt.xlabel("Time (samples)")
plt.ylabel("Received signal $m[n]$")
plt.show()
```

These acoustic pulses are transmitted every $M = 10000$ samples. This means that the sonar is repeating a measurement of $h[n]$ every 10000 samples and the transmitted signal actually looks like this:

$$\epsilon[n] = \sum_{k=0}^{N_p-1} x[n - Mk] \quad (14.6)$$

In this task, you should treat each segment of 10000 samples as an independent sonar measurement, which will allow you to study how the impulse response evolves over time. You can do this as follows:

```
import matplotlib.pyplot as plt
import numpy as np

m = np.fromfile("sonar_meas.bin", dtype=np.float32)

interpulse_period = 10000
N_ipps = int(np.floor(len(m)/interpulse_period))
for i in range(N_ipps):
```

Figure 14.3: The chirp transmit pulse signal $x[n]$ used for this exercise. In this case, the pulse is $N = 1000$ samples long, sweeping in frequency between 0 and 22.05 kHz.

```

echoes = m[(i * interpulse_period):(i * interpulse_period +
interpulse_period)]
plt.plot(echoes)
plt.show()

```

For this assignment, you are to perform the following tasks. Write a short report describing what you did and what results you got. The report is otherwise free form, as long as it is in PDF format. Include your code and plots in the report.

You will find a lot of help for this task in the lecture notes that discusses LTI systems, convolution and frequency response. You may help each other on Perusall. It is fine to give hints, but please try not to give away the *exact solution*.

- a) How many meters of distance² does an acoustic pulse travel during one sample duration $T_s = f_s^{-1}$? Assume sound speed in a typical lecture room at UiT (343 m/s).
- b) How long is the transmitted pulse read from file `chirp.bin` in seconds?
- c) How long is the measurement in seconds? You can figure this out by looking at the length of the signal `sonar_meas.bin` and using the known sample-rate of $44.1 \cdot 10^3$ samples per second.
- d) How many transmit pulses are contained in the measurement?
One pulse is transmitted every 10000 samples.
- e) Why does the deconvolution filter $\lambda[n]$ need to have the property:
 $\lambda[n] * x[n] \approx A\delta[n]$, where A is a constant?
- f) Evaluate the autocorrelation function of the transmitted signal
 $c[n] = x[n] * x[-n]$. Use the signal that you read from `chirp.bin`. You can time-reverse a signal in Python using `deconvolution_filter = x[::-1]`. Make a plot of $c[n]$ zooming into the portion of the signal that has significantly non-zero values.
- g) How close is $c[n]$ in your opinion to a scaled unit impulse $A\delta[n]$? How will deviations from the unit impulse affect deconvolution quality when using $\lambda[n] = x[-n]$ as a deconvolution filter?
- h) Make a plot of the scattered power as a function of time and range. Use one pulse for each column of the array that you are plotting. You should get something similar to the plot in Figure 14.4. Use decibel scale for power. Use time in seconds on the x-axis and total propagation distance in meters on the y-axis. Total propagation distance means the total travelled distance between the loudspeaker and the microphone. You can use, e.g., the `pcolormesh` command from Python's Matplotlib to make this plot.

² I mean total distance, *not half the total distance*, which is often used in sonar and radar to take into account the fact that the wave has to travel to the target and back.

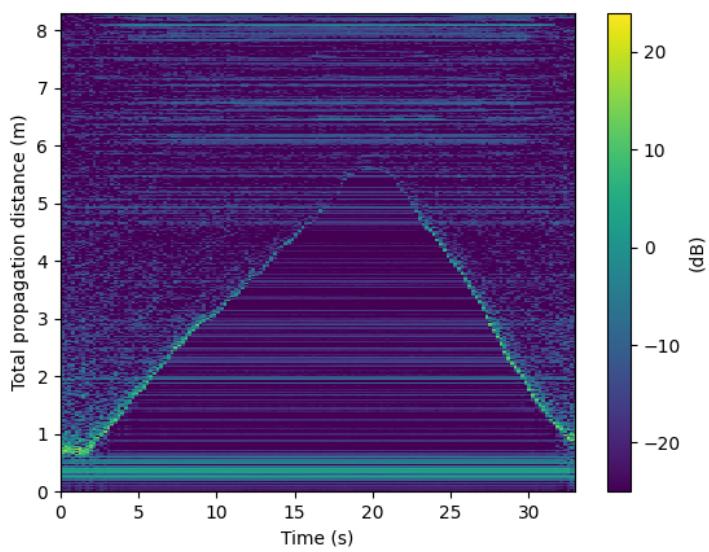


Figure 14.4: An example range-time intensity plot showing how much power in decibel scale is scattered back from the surroundings of the sonar system. The distance between the microphone and the loudspeaker is about 30 cm, which can be seen as a constant horizontal line at the propagation distance. The plot also shows an acoustic scatterer that is first moving away and then moving back towards the transmitter.

15

Discrete-time Fourier Transform (DTFT)

The equation for frequency response of a discrete-time linear time-invariant system (Equation 13.21) is also known as the *Discrete-Time Fourier Transform* (DTFT). We can obtain a DTFT for an arbitrary discrete-time signal $x[n]$ as follows¹:

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\hat{\omega}n}. \quad (15.1)$$

¹ Hat on top of \hat{x} to signify that it is a frequency domain representation of signal $x[n]$, and a hat on top of $\hat{\omega}$ to signify that the frequency has units of radians per sample.

Here $\hat{x}(\hat{\omega})$ is the DTFT of a discrete-time signal $x[n]$. We'll use the following notation for denoting DTFT pairs:

$$x[n] \xleftrightarrow{\mathcal{F}} \hat{x}(\hat{\omega}) \quad (15.2)$$

like we did in the Fourier transform chapter. The left-hand side denotes the time domain representation, and the right-hand side denotes the frequency domain representation.

Let's show that Equation 15.1 is a special case of the continuous-time Fourier transform for discretized signals. Remember that the formula for discretizing a continuous-time signal is:

$$x[n] = x(nT_s). \quad (15.3)$$

Here T_s is the sample spacing, which is related to sample-rate $f_s = 1/T_s$. The continuous-time representation of the discretized signal is defined as:

$$x_d(t) = \sum_{n=-\infty}^{\infty} x[n]\delta(t - nT_s). \quad (15.4)$$

You might recall this from the derivation of the Whittaker-Shannon sampling theorem. Figure 15.1 depicts a continuous-time signal and the continuous-time representation of the discretized signal $x_d(t)$, which is a sequence of unit impulses multiplied by the value of each discrete-time signal sample $x[n]$ at the position in time, where it was sampled.

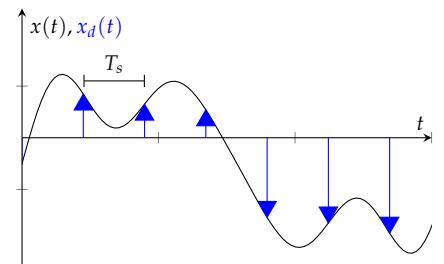


Figure 15.1: A continuous-time signal $x(t)$ and a continuous-time representation of a discretized signal $x_d(t)$. Sample spacing T_s is related to sample rate as follows: $T_s = 1/f_s$.

The Fourier transform of $x_d(t)$ is

$$\hat{x}_d(\omega) = \int_{-\infty}^{\infty} x_d(t) e^{-i\omega t} dt . \quad (15.5)$$

This can be simplified as follows:

$$\hat{x}_d(\omega) = \int_{-\infty}^{\infty} \left(\sum_{n=-\infty}^{\infty} \delta(t - nT_s) x[n] \right) e^{-i\omega t} dt \quad (15.6)$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} \delta(t - nT_s) x[n] e^{-i\omega t} dt \quad (15.7)$$

$$= \sum_{n=-\infty}^{\infty} x[n] e^{-i\omega n T_s} . \quad (15.8)$$

If we substitute $\hat{\omega} = \omega T_s$, we get²:

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-i\hat{\omega}n} \quad \square . \quad (15.9)$$

This shows that the discrete-time Fourier transform is a special case of the continuous-time Fourier transform for discretized signals, and that $\hat{x}(\hat{\omega})$ is the frequency domain representation of the discrete-time signal $x[n]$.

The fact that the discrete-time Fourier transform is a Fourier transform means that all the properties of a Fourier transform also apply to the discrete-time Fourier transform. However, there are some special properties of the discrete-time Fourier transform, which we will go through.

Periodicity

The DTFT $\hat{x}(\hat{\omega})$ is a 2π -periodic function:

$$\boxed{\hat{x}(\hat{\omega}) = \hat{x}(\hat{\omega} + 2\pi k)} \quad (15.10)$$

with $k \in \mathbb{Z}$. This is relatively easy to show:

$$\hat{x}(\hat{\omega} + 2\pi k) = \sum_{n=-\infty}^{\infty} x[n] e^{-i(\hat{\omega} + 2\pi k)n} \quad (15.11)$$

$$= \sum_{n=-\infty}^{\infty} x[n] e^{-i\hat{\omega}n} \underbrace{e^{-i2\pi nk}}_{=1} \quad (15.12)$$

$$= \hat{x}(\hat{\omega}) \quad \square . \quad (15.13)$$

The periodicity property is the same one that we encountered when investigating aliasing of complex sinusoidal discrete-time signals, which are essentially frequency components of a discrete-time signal. Figure 15.2 shows one example of a discrete-time Fourier transform, the Dirichlet kernel, that we encountered when investigating the frequency response of a running average filter.

² This also implies that $\hat{x}(\hat{\omega}) = \hat{x}_d(\hat{\omega}/T_s)$

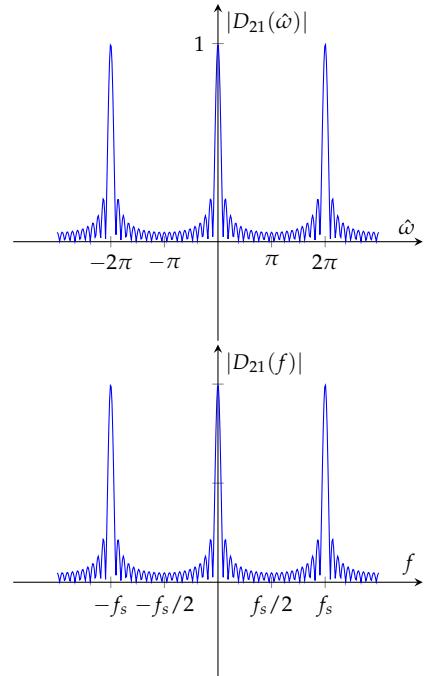


Figure 15.2: A discrete-time Fourier transform is 2π -periodic in frequency domain. The top figure shows the magnitude response of the running average filter $|D_{21}(\hat{\omega})|$ with frequency in units of radians per sample on the x-axis. Bottom: The same as the above, but with x-axis with frequency in units of cycles per second. The sampling-rate $f = \pm f_s/2$ corresponds to $\hat{\omega} = \pm\pi$.

Inverse transform

The reverse operation is the inverse discrete-time Fourier transform, which is defined as:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} . \quad (15.14)$$

This allows transforming a discrete-time frequency domain representation $\hat{x}(\hat{\omega})$ to a discrete-time signal $x[n]$.

It is quite easy to show that this inverse formula recovers $x[n]$ for a DTFT $\hat{x}(\hat{\omega})$ that is defined as³:

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-i\hat{\omega}n} , \quad (15.15)$$

we can recover $x[n]$ using the inverse transform formula:

$$\mathcal{F}^{-1}\{\hat{x}(\hat{\omega})\} = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} \quad (15.16)$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[\sum_{m=-\infty}^{\infty} x[m] e^{-i\hat{\omega}m} \right] e^{i\hat{\omega}n} d\hat{\omega} \quad (15.17)$$

$$= \frac{1}{2\pi} \sum_{m=-\infty}^{\infty} \int_{-\pi}^{\pi} x[m] e^{i\hat{\omega}(n-m)} d\hat{\omega} \quad (15.18)$$

$$= \frac{1}{2\pi} \sum_{m=-\infty}^{\infty} 2\pi \delta[m-n] x[m] \quad (15.19)$$

$$= x[n] \quad \square \quad (15.20)$$

Here $\mathcal{F}^{-1}\{\cdot\}$ refers to the inverse DTFT.

We now have an inverse and forward discrete-time Fourier transform pair:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} \xleftrightarrow{\mathcal{F}} \hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-i\hat{\omega}n} . \quad (15.21)$$

We'll now list several discrete-time Fourier transform pairs. Note that many of these have in fact already been discussed in the chapter on the continuous-time Fourier transform.

Frequency response

The frequency response of a discrete-time LTI system (shown earlier in Equation 13.21)

$$\mathcal{H}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} h[k] e^{-i\hat{\omega}k} \quad (15.22)$$

³ The inverse transform is nearly the same as the Fourier series analysis equation with $x[n]$ being the Fourier series coefficients and $\hat{x}(\hat{\omega})$ being a 2π -periodic function.

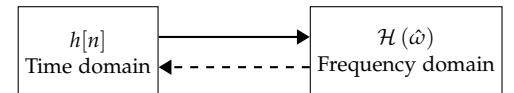


Figure 15.3: The frequency response $\mathcal{H}(\hat{\omega})$ of an LTI system is the Fourier transform of the impulse response $h[n]$.

is a discrete-time Fourier transform of the impulse response $h[n]$. This means that all the properties of a discrete-time Fourier transform will apply also to a frequency response. Using our short hand notation, we can define this as a DTFT pair:

$$h[n] \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) . \quad (15.23)$$

The inverse DTFT formula will come in handy when, e.g., defining LTI systems based on their frequency response characteristics, and then obtaining the impulse response using the inverse transform. This leads to, e.g., ideal filters, ideal frequency selective filters, and arbitrary time shift filters. These are LTI systems that would be difficult or impossible to derive in time domain.

Time-shifted unit impulse

DTFT for unit impulse is:

$$x[n] = \delta[n - n_0] \xleftrightarrow{\mathcal{F}} \hat{x}(\hat{\omega}) = e^{-i\hat{\omega}n_0} . \quad (15.24)$$

The proof is trivial.

Frequency shift $e^{i\hat{\omega}_0 n} x[n]$

A shift in frequency domain corresponds to multiplication by a complex exponential signal in time domain. If

$$x[n] \xleftrightarrow{\mathcal{F}} \hat{x}(\hat{\omega}) , \quad (15.25)$$

then

$$e^{i\hat{\omega}_0 n} x[n] \xleftrightarrow{\mathcal{F}} \hat{x}(\hat{\omega} - \hat{\omega}_0) . \quad (15.26)$$

The proof is as follows. If we multiply the signal $x[n]$ with a complex exponential signal with frequency $\hat{\omega}_0$, we obtain a new signal $y[n]$:

$$y[n] = e^{i\hat{\omega}_0 n} x[n] . \quad (15.27)$$

If we DTFT this, we obtain:

$$\hat{y}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} e^{i\hat{\omega}_0 k} x[k] e^{-i\hat{\omega}k} \quad (15.28)$$

$$= \sum_{k=-\infty}^{\infty} x[k] e^{-i(\hat{\omega} - \hat{\omega}_0)k} = \hat{x}(\hat{\omega} - \hat{\omega}_0) \quad \square . \quad (15.29)$$

Multiplying a time domain signal by a complex sinusoid with frequency $\hat{\omega}_0$ results in a frequency shift of the signal in frequency domain.

This property is useful for creating a filter with a specific center frequency. This property can also be used to shift the center frequency of a signal in frequency domain, e.g., in digital up or down conversion systems that are found in digital radios.

Convolution theorem

An often used theorem of signal processing is the convolution theorem. It states that convolution in time domain is multiplication in frequency domain. This also applies to discrete-time Fourier transforms.

$$a[n] = b[n] * c[n] \xleftrightarrow{\mathcal{F}} \hat{a}(\hat{\omega}) = \hat{b}(\hat{\omega})\hat{c}(\hat{\omega}) . \quad (15.30)$$

The proof is identical to the proof shown in the chapter on Fourier transforms.

Example: cascaded filters

A consequence of the convolution theorem is that we can analyze the combined effect of a cascaded system, simply by multiplying together the frequency responses.

Let's assume that a signal is first convolved with a filter that has an impulse response $h_1[n]$ and then with a filter that has an impulse response $h_2[n]$:

$$y[n] = h_2[n] * (h_1[n] * x[n]) \quad (15.31)$$

$$= (h_1[n] * h_2[n]) * x[n] \quad (15.32)$$

$$= h_3[n] * x[n] . \quad (15.33)$$

The combined frequency response $\mathcal{H}_3(\hat{\omega})$ of the cascaded system is $\mathcal{H}_1(\hat{\omega})\mathcal{H}_2(\hat{\omega})$:

$$h_3[n] = h_1[n] * h_2[n] \xleftrightarrow{\mathcal{F}} \mathcal{H}_3(\hat{\omega}) = \mathcal{H}_1(\hat{\omega})\mathcal{H}_2(\hat{\omega}) \quad (15.34)$$

where $\mathcal{H}_1(\hat{\omega})$ and $\mathcal{H}_2(\hat{\omega})$ are the frequency responses corresponding to impulse responses $h_1[n]$ and $h_2[n]$.

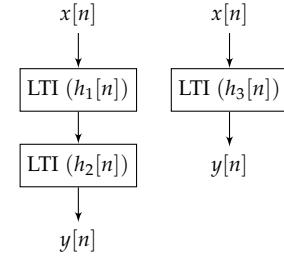


Figure 15.4: A consequence of the associative property of convolution is that two LTI systems characterized with $h_1[n]$ and $h_2[n]$ can be combined as a single LTI system with impulse response $h_3[n] = h_1[n] * h_2[n]$. The convolution theorem allows us to investigate the combined frequency response of the system using the following formula: $\mathcal{H}_3(\hat{\omega}) = \mathcal{H}_1(\hat{\omega})\mathcal{H}_2(\hat{\omega})$.

Exercises: Discrete-time Fourier Transform (DTFT)

1. Show that if $x[n] \in \mathbb{R}$, then $\hat{x}(\hat{\omega}) = \hat{x}^*(-\hat{\omega})$.
2. Show that if $\hat{x}(\hat{\omega}) \in \mathbb{R}$, then $x[n] = x^*[-n]$.
3. Show that $\mathcal{F}^{-1}\{42i\} = 42i\delta[n]$ using Equation 15.14. Here \mathcal{F}^{-1} is the inverse discrete-time Fourier transform.
4. Consider three systems: $\mathcal{T}_1\{x[n]\} = x[n] - x[n-1]$, $\mathcal{T}_2\{x[n]\} = x[n] + x[n-2]$, and $\mathcal{T}_3\{x[n]\} = x[n-1] + x[n-2]$.
 - a) What are the impulse responses for these three LTI systems: $h_1[n] = \mathcal{T}_1\{\delta[n]\}$, $h_2[n] = \mathcal{T}_2\{\delta[n]\}$, and $h_3[n] = \mathcal{T}_3\{\delta[n]\}$?
 - b) Show that the frequency response of the system $y[n] = \mathcal{T}_1\{\mathcal{T}_2\{\mathcal{T}_3\{x[n]\}\}\}$ is $e^{-i\hat{\omega}} - e^{-5i\hat{\omega}}$.
5. Inverse DTFT the following functions:
 - a) $\hat{x}(\hat{\omega}) = 1 - 2e^{3i\hat{\omega}}$
 - b) $\hat{x}(\hat{\omega}) = 2e^{-3i\hat{\omega}} \cos(\hat{\omega})$
 - c) $\hat{x}(\hat{\omega}) = e^{-42i\hat{\omega}} \frac{\sin(10.5\hat{\omega})}{\sin(\hat{\omega}/2)}$.

Hint: Look at the derivation for the Dirichlet kernel.

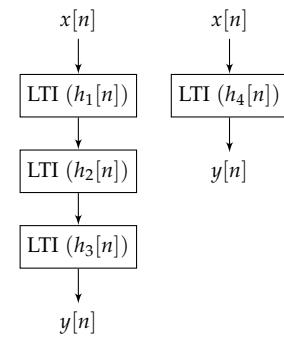


Figure 15.5: A cascade of LTI systems defined by impulse responses $h_1[n]$, $h_2[n]$, and $h_3[n]$ is equivalent to a single LTI system defined by an impulse response $h_4[n]$.

Suggested solutions: Discrete-time Fourier Transform (DTFT)

1. Suppose $x[n]$ is a real discrete-time signal, then by definition:

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\hat{\omega}n}.$$

Apply complex conjugation to obtain:

$$\hat{x}^*(\hat{\omega}) = \sum_{n=-\infty}^{\infty} (x[n]e^{-i\hat{\omega}n})^* = \sum_{n=-\infty}^{\infty} x^*[n]e^{i\hat{\omega}n}.$$

Since the signal $x[n]$ is real: $x^*[n] = x[n]$. Then evaluating at $\hat{x}^*(\hat{\omega})$ at $-\hat{\omega}$:

$$\hat{x}^*(-\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\hat{\omega}n},$$

which is equal to $\hat{x}(\hat{\omega})$.

2. Suppose $\hat{x}(\hat{\omega})$ is a real signal, then consider:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega})e^{i\hat{\omega}n} d\hat{\omega}.$$

Conjugating:

$$\hat{x}^*[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} (\hat{x}(\hat{\omega})e^{i\hat{\omega}n})^* d\hat{\omega} = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}^*(\hat{\omega})e^{-i\hat{\omega}n} d\hat{\omega},$$

and since $\hat{x}(\hat{\omega})$ is real, we obtain $\hat{x}^*(\hat{\omega}) = \hat{x}(\hat{\omega})$, then evaluating $\hat{x}^*[n]$ at $-n$ gives:

$$\hat{x}^*[-n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega})e^{i\hat{\omega}n} d\hat{\omega},$$

which is equal to $x[n]$.

3. Let $\hat{x}(\hat{\omega}) = 1$, then the inverse discrete-time Fourier transform is:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \hat{x}(\hat{\omega})e^{i\hat{\omega}n} d\hat{\omega} = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\hat{\omega}n} d\hat{\omega} = \frac{1}{2\pi} \left[\frac{1}{in} e^{i\hat{\omega}n} \right]_{-\pi}^{\pi},$$

which gives:

$$x[n] = \frac{1}{2\pi in} (e^{i\pi n} - e^{-i\pi n}) = \frac{1}{\pi n} \sin(\pi n).$$

If $n \neq 0$, then this is 0. If $n = 0$ we get, using L'Hôpital's rule:

$$\lim_{n \rightarrow 0} \frac{\sin(\pi n)}{\pi n} = 1.$$

Hence, this corresponds to a Dirac-delta $\delta[n]$, so by linearity we get $42i\delta[n]$.

4. Consider three systems:

$$\begin{aligned}\mathcal{T}_1\{x[n]\} &= x[n] - x[n-1], \\ \mathcal{T}_2\{x[n]\} &= x[n] + x[n-2], \\ \mathcal{T}_3\{x[n]\} &= x[n-1] + x[n-2].\end{aligned}$$

- a) The impulse responses are given as $h_i[n] = \mathcal{T}_i\{\delta[n]\}$ for $i = 1, 2, 3$. We get:

$$\begin{aligned}h_1[n] &= \delta[n] - \delta[n-1], \\ h_2[n] &= \delta[n] + \delta[n-2], \\ h_3[n] &= \delta[n-1] + \delta[n-2].\end{aligned}$$

b) In time domain we have:

$$y[n] = \mathcal{T}_1\{\mathcal{T}_2\{\mathcal{T}_3\{x[n]\}\}\},$$

which can be written as:

$$y[n] = h[n] * x[n],$$

since this an LTI system, where:

$$h[n] = h_1[n] * h_2[n] * h_3[n].$$

In frequency domain, we have:

$$\mathcal{H}(\hat{\omega}) = \mathcal{H}_1(\hat{\omega})\mathcal{H}_2(\hat{\omega})\mathcal{H}_3(\hat{\omega}).$$

Each \mathcal{H}_i is found by the discrete-time Fourier transform of $h_i[n]$:

$$\begin{aligned}\mathcal{H}_1(\hat{\omega}) &= 1 - e^{-i\hat{\omega}}, \\ \mathcal{H}_2(\hat{\omega}) &= 1 + e^{-2i\hat{\omega}}, \\ \mathcal{H}_3(\hat{\omega}) &= e^{-i\hat{\omega}} + e^{-2i\hat{\omega}}.\end{aligned}$$

Then:

$$\mathcal{H}(\hat{\omega}) = (1 - e^{-i\hat{\omega}})(1 + e^{-2i\hat{\omega}})(e^{-i\hat{\omega}} + e^{-2i\hat{\omega}}) = e^{-i\hat{\omega}} - e^{-5i\hat{\omega}},$$

as we wanted to show.

5. a) If $\hat{x}(\hat{\omega}) = 1 - 2e^{3i\hat{\omega}}$, then the IDTFT is:

$$\underline{\underline{x[n] = \delta[n] - 2\delta[n+3]}}.$$

- b) If $\hat{x}(\hat{\omega}) = 2e^{-3i\hat{\omega}} \cos(\hat{\omega})$ then by rewriting the function as follows:

$$\hat{x}(\hat{\omega}) = 2e^{-3i\hat{\omega}} \frac{1}{2}(e^{i\hat{\omega}} + e^{-i\hat{\omega}}) = e^{-3i\hat{\omega}}(e^{i\hat{\omega}} + e^{-i\hat{\omega}}) = e^{-2i\hat{\omega}} + e^{-4i\hat{\omega}},$$

which gives that:

$$\underline{\underline{x[n] = \delta[n-2] + \delta[n-4]}}.$$

- c) For the function $\hat{x}(\hat{\omega}) = e^{-42i\hat{\omega}} \frac{\sin(10.5\hat{\omega})}{\sin(\hat{\omega}/2)}$, we notice the similarity to the Dirichlet kernel. Therefore, we start by rewriting the function and do the Dirichlet calculation backwards.

$$\hat{x}(\hat{\omega}) = e^{-42i\hat{\omega}} \left[\frac{e^{-10.5i\hat{\omega}} - e^{10.5i\hat{\omega}}}{e^{-i\hat{\omega}/2} - e^{i\hat{\omega}/2}} \right].$$

We've flipped the sign in the numerator and denominator and canceled the common $2i$. Next, multiply by $e^{i\hat{\omega}/2}$.

$$\begin{aligned}\hat{x}(\hat{\omega}) &= e^{-42i\hat{\omega}} \left[\frac{e^{i\hat{\omega}/2} e^{-10.5i\hat{\omega}} - e^{10.5i\hat{\omega}}}{e^{i\hat{\omega}/2} e^{-i\hat{\omega}/2} - e^{i\hat{\omega}/2}} \right], \\ &= e^{-42i\hat{\omega}} \left[\frac{e^{-10i\hat{\omega}} - e^{11i\hat{\omega}}}{1 - e^{i\hat{\omega}}} \right],\end{aligned}$$

This last expression is a geometric sum, with $N = 10$, giving:

$$\hat{x}(\hat{\omega}) = e^{-42i\hat{\omega}} \sum_{k=-10}^{10} e^{-i\hat{\omega}k} = \sum_{k=-10}^{10} e^{-i\hat{\omega}(k+42)}.$$

The IDTFT is then:

$$\underline{\underline{x}[n] = \sum_{k=-10}^{10} \delta[n - k - 42].}$$

16

Ideal and Tapered Filters

The inverse discrete-time Fourier transform can be used for filter design purposes. It is possible to specify a filter in frequency domain $\mathcal{H}(\hat{\omega})$, and then use an inverse Fourier transform to obtain the impulse response $h[n]$ of the filter with these properties. However, this often leads to filters that are infinitely long and therefore impossible to realize. In practice, the issue of infinitely long filters can be overcome by truncating the impulse response into one that has a finite length.

This chapter will only discuss ideal and tapered windows in the context of discrete-time signals. We will first go through several important ideal filters and derive their impulse responses. We will then discuss various strategies for truncating and tapering these into finite length filters.

Ideal filters

It is possible to specify a filter in frequency domain. In many cases, one knows what sort of frequency response is needed. The basic recipe is to inverse DTFT the desired frequency response to obtain a time domain impulse response. In this section, we'll go through some basic ideal filters.

Example: Ideal low-pass filter

An ideal low-pass filter has the following spectral response:

$$\mathcal{H}_{LP}(\hat{\omega}) = \begin{cases} 1 & |\hat{\omega}| < \hat{\omega}_0 \\ 0 & \hat{\omega}_0 \leq |\hat{\omega}| \leq \pi \end{cases}. \quad (16.1)$$

In order to determine the filter coefficients, one needs to inverse

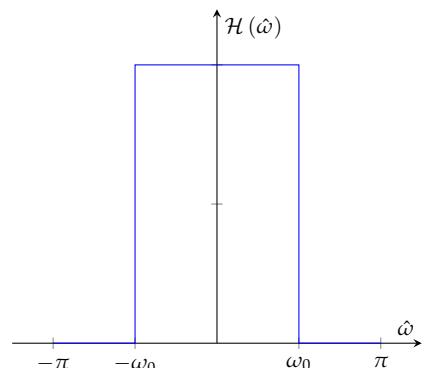


Figure 16.1: The frequency response of an ideal low pass filter, which filters out signals with normalized frequencies larger than $|\hat{\omega}| > \hat{\omega}_0$

DTFT the frequency response:

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathcal{H}_{LP}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} \quad (16.2)$$

$$= \frac{1}{2\pi} \int_{-\hat{\omega}_0}^{\hat{\omega}_0} e^{i\hat{\omega}n} d\hat{\omega} \quad (16.3)$$

$$= \frac{1}{2\pi} \left. \frac{e^{i\hat{\omega}n}}{in} \right|_{-\hat{\omega}_0}^{\hat{\omega}_0} \quad (16.4)$$

$$= \frac{1}{\pi n} \frac{1}{2i} (e^{i\hat{\omega}_0 n} - e^{-i\hat{\omega}_0 n}) \quad (16.5)$$

$$= \frac{\sin(\hat{\omega}_0 n)}{\pi n}. \quad (16.6)$$

The impulse response for this ideal filter is infinitely long, but it is damped by a factor of $1/n$. There is also an indeterminate value at $n = 0$, which can be determined using L'Hôpital's rule.

$$\lim_{x \rightarrow c} \frac{f(x)}{g(x)} = \lim_{x \rightarrow c} \frac{f'(x)}{g'(x)} \quad (16.7)$$

$$\lim_{n \rightarrow 0} \frac{\sin(\hat{\omega}_0 n)}{\pi n} = \lim_{n \rightarrow 0} \frac{\hat{\omega}_0 \cos(\hat{\omega}_0 n)}{\pi} \quad (16.8)$$

$$= \frac{\hat{\omega}_0}{\pi}. \quad (16.9)$$

Therefore, the ideal low-pass filter is defined as:

$$h[n] = \frac{\sin(\hat{\omega}_0 n)}{\pi n} \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) = u(\hat{\omega} + \hat{\omega}_0) - u(\hat{\omega} - \hat{\omega}_0). \quad (16.10)$$

The impulse response for an ideal low-pass filter with $\hat{\omega}_0 = 0.4\pi$ is shown in Figure 16.2.

The function $\sin(\pi x)/\pi x$ is often referenced to as the sinc-function:

$$\text{sinc}(\theta) = \frac{\sin(\pi\theta)}{\pi\theta}. \quad (16.11)$$

Sometimes it is useful to express the ideal low-pass using a sinc function, as numerical implementations of this function take into account the limiting behavior at $\hat{\omega} = 0$ automatically.

Example: Ideal high-pass filter

An ideal high-pass filter with a cutoff at $\hat{\omega}_0$ is defined as:

$$\mathcal{H}_{HP}(\hat{\omega}) = \begin{cases} 0 & |\hat{\omega}| < \hat{\omega}_0 \\ 1 & \hat{\omega}_0 \leq |\hat{\omega}| \leq \pi \end{cases}. \quad (16.12)$$

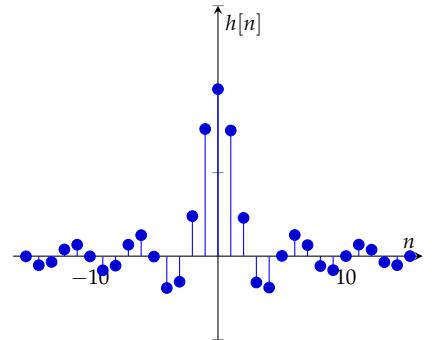


Figure 16.2: The impulse response of an ideal low-pass filter.

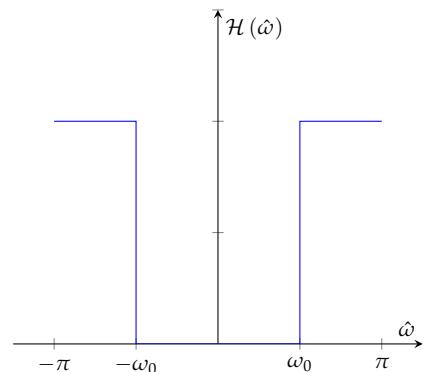


Figure 16.3: The frequency response of an ideal high-pass filter.

To obtain the time domain impulse response $h[n]$, we perform an inverse DTFT, much in the same way as we did for the ideal low pass filter:

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathcal{H}_{\text{HP}}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} \quad (16.13)$$

$$= \frac{1}{2\pi} \left[\int_{-\pi}^{-\omega_0} e^{i\hat{\omega}n} d\hat{\omega} + \int_{\omega_0}^{\pi} e^{i\hat{\omega}n} d\hat{\omega} \right] \quad (16.14)$$

$$= \frac{1}{2\pi ni} \left[e^{-i\hat{\omega}_0 n} - e^{-i\pi n} + e^{i\pi n} - e^{i\hat{\omega}_0 n} \right] \quad (16.15)$$

$$= \frac{1}{\pi n} [\sin(\pi n) - \sin(\hat{\omega}_0 n)] . \quad (16.16)$$

Here we need to be careful when $n = 0$. While $\sin(\pi n)/\pi n = 0$ when $n \neq 0$, when $n = 0$, using L'Hôpital's rule, we get:

$$\lim_{n \rightarrow 0} \frac{\sin(\pi n)}{\pi n} = 1 . \quad (16.17)$$

Therefore, the impulse response of the ideal high-pass filter is:

$$h[n] = \delta[n] - \frac{\sin(\hat{\omega}_0 n)}{\pi n} \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) = 1 - [u(\hat{\omega} + \omega_0) - u(\hat{\omega} - \omega_0)] \quad (16.18)$$

which is another way of saying all but the ideally low pass filtered signal.

Example: Ideal band-pass filter

An ideal band-pass filter has a non-zero frequency response only on a narrow frequency range (passband). For a real-valued passband filter, one needs to include the positive and negative frequency passbands. The specification for an ideal band-pass filter that only lets through real-valued signals with frequencies between ω_0 and ω_1 is:

$$\mathcal{H}_{\text{BP}}(\hat{\omega}) = \begin{cases} 1 & \omega_0 < |\hat{\omega}| < \omega_1 \\ 0 & \text{otherwise} \end{cases} . \quad (16.19)$$

The frequency response is shown in Figure 16.4.

This has the impulse response given by:

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathcal{H}_{\text{BP}}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega} \quad (16.20)$$

$$= \frac{1}{2\pi} \left[\int_{-\hat{\omega}_1}^{-\hat{\omega}_0} e^{i\hat{\omega}n} d\hat{\omega} + \int_{\hat{\omega}_0}^{\hat{\omega}_1} e^{i\hat{\omega}n} d\hat{\omega} \right] \quad (16.21)$$

$$= \frac{1}{2\pi ni} \left[e^{-i\hat{\omega}_0 n} - e^{-i\hat{\omega}_1 n} + e^{i\hat{\omega}_1 n} - e^{i\hat{\omega}_0 n} \right] \quad (16.22)$$

$$= \frac{\sin(\hat{\omega}_1 n)}{\pi n} - \frac{\sin(\hat{\omega}_0 n)}{\pi n} . \quad (16.23)$$

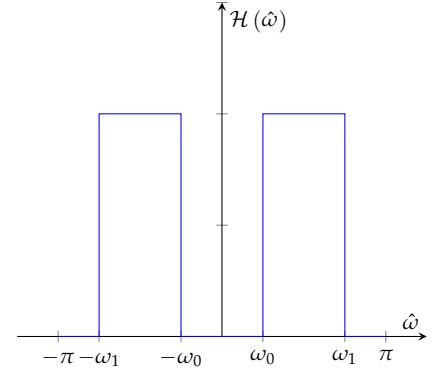


Figure 16.4: The frequency response of an ideal band-pass filter.

Again, using L'Hôpital's rule, we obtain the behavior at $n = 0$:

$$h[0] = (\hat{\omega}_1 - \hat{\omega}_0)/\pi . \quad (16.24)$$

Similarly, it is possible to determine the impulse response for an ideal band-stop filter, which has zero response between ω_0 and ω_1 , but lets through signals at all other frequencies. We leave this as an exercise to the reader.

Example: Ideal point frequency filter

A filter, which is selective to only one frequency $\hat{\omega}_0$:

$$h[n] = \frac{1}{2\pi} e^{i\hat{\omega}_0 n} \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) = \delta(\hat{\omega} - \hat{\omega}_0) . \quad (16.25)$$

Here $\delta(x)$ is the Dirac delta function. The impulse response of such a filter can be obtained using the IDTFT (for $|\omega_0| < \pi$):

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \delta(\hat{\omega} - \hat{\omega}_0) e^{i\hat{\omega} n} d\hat{\omega} \quad (16.26)$$

$$= \frac{1}{2\pi} e^{i\hat{\omega}_0 n} . \quad (16.27)$$

This filter is constant in magnitude, i.e., $|h[n]| = (2\pi)^{-1}$, and infinitely long. The filter is a complex sinusoid, with frequency matched to the signal frequency $\hat{\omega}_0$.

Example: Real-valued point frequency filter

The real-valued single frequency filter is actually a filter that lets through two frequencies. The positive and negative frequency component $\hat{\omega}_0$ and $-\hat{\omega}_0$.

$$h[n] = \frac{1}{\pi} \cos(\hat{\omega}_0 n) \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) = \delta(\hat{\omega} - \hat{\omega}_0) + \delta(\hat{\omega} + \hat{\omega}_0) . \quad (16.28)$$

Proof.

$$h[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \delta(\hat{\omega} - \hat{\omega}_0) e^{i\hat{\omega} n} d\hat{\omega} + \frac{1}{2\pi} \int_{-\pi}^{\pi} \delta(\hat{\omega} + \hat{\omega}_0) e^{i\hat{\omega} n} d\hat{\omega} \quad (16.29)$$

$$= \frac{1}{2\pi} (e^{i\hat{\omega}_0 n} + e^{-i\hat{\omega}_0 n}) \quad (16.30)$$

$$= \frac{1}{\pi} \cos(\hat{\omega}_0 n) . \quad (16.31)$$

□

The ideal point-frequency impulse response will be later revisited when dealing with discrete Fourier transforms as filter banks.

Fractional sample delay

An important elementary filter is the time-shift filter, which has an impulse response

$$h[n] = \delta[n - n_0]. \quad (16.32)$$

This filter delays the signal by n_0 samples. We already determined earlier that time-delay is equivalent to a linear phase slope in frequency domain:

$$\mathcal{H}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} \delta[n - n_0] e^{-i\hat{\omega}n} \quad (16.33)$$

$$= e^{-i\hat{\omega}n_0}. \quad (16.34)$$

We can now use this to derive the impulse response of a fractional sample delay filter, which delays the signal by a real-valued number of samples $n_0 \in \mathbb{R}$:

$$h[n] = \frac{\sin(\pi[n - n_0])}{\pi(n - n_0)} \xleftrightarrow{\mathcal{F}} \mathcal{H}(\hat{\omega}) = e^{-i\hat{\omega}n_0}. \quad (16.35)$$

For the proof, we need to evaluate the inverse DTFT of the frequency domain specification of a time-shift $\mathcal{H}(\hat{\omega}) = e^{-i\hat{\omega}n_0}$:

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{-i\hat{\omega}n_0} e^{i\hat{\omega}n} d\hat{\omega} \quad (16.36)$$

$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{i\hat{\omega}(n-n_0)} d\hat{\omega} \quad (16.37)$$

$$= \frac{1}{2\pi} \frac{e^{i\hat{\omega}(n-n_0)}}{i(n - n_0)} \Big|_{\hat{\omega}=-\pi}^{\pi} \quad (16.38)$$

$$= \frac{1}{\pi(n - n_0)} \frac{1}{2i} (e^{i\pi(n - n_0)} - e^{-i\pi(n - n_0)}) \quad (16.39)$$

$$= \frac{\sin(\pi[n - n_0])}{\pi(n - n_0)}. \quad (16.40)$$

Figure 16.5 shows the impulse response of a filter that delays a signal by $n_0 = 0.5$, that is by half a sample. It would be difficult to define this filter in time domain, but relatively straightforward using the frequency domain specification, which is inverse discrete Fourier transformed into time domain to obtain an impulse response.

It can be shown that for integer values $n_0 \in \mathbb{Z}$, the impulse response simplifies to

$$\frac{\sin(\pi[n - n_0])}{\pi(n - n_0)} = \delta[n - n_0], \quad (16.41)$$

but this would be easy to define already in time domain.

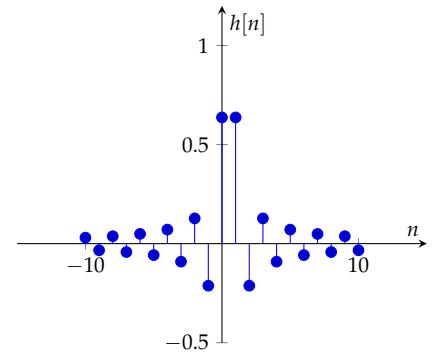


Figure 16.5: Fractional sample delay filter impulse response $h[n]$, which delays the signal by half a sample. This filter is infinitely long.

Tapered ideal filters

Ideal filters for discrete-time signals, as discussed in the discrete-time Fourier transform chapter, are impossible to realize. This is because ideal filters require an infinitely long impulse response $h[n]$.

In practical applications, finite length filters have to be used. One method for designing filters is to take these infinitely long ideal filters and truncate these to get a filter with finite length. Approximating an ideal filter with an FIR filter can be accomplished by using *window functions* or *tapering windows*. A window function is a function that is 0 outside an interval, so the effect of multiplying with a window function is that the window function truncates the filter, making it finite.

If $h[n]$ is an ideal filter centered at $n = 0$, a windowed version of this filter is obtained by multiplying the ideal filter with a window function $w(n)$ of length N :

$$h_w[n] = w(n)h[n - N/2], \quad (16.42)$$

After this, the non-zero amplitude portion of the impulse response $h_w[n]$ is finite length and the filter $h_w[n]$ is a finite impulse response filter that can be applied in practice to a signal.

There are two main effects that you need to be aware of when using Equation 16.42 to create a truncated finite length filter. Firstly, there is a time delay of $N/2$ introduced by the windowing operation. This results in a time shift when applying this filter. Secondly, due to time-frequency ambiguity, this type of practical filter will have finite spectral resolution. It is impossible to achieve a perfectly sharp filter cutoff using a practical filters.

Rectangular window

The most basic type of window function is the rectangular window. If the length of the filter is N , the window function evaluates to:

$$w_R[n] = \begin{cases} 1 & 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases} \quad (16.43)$$

This window function just selects N central points of an infinitely long impulse response $h[n]$. The top panel of Figure 16.6 shows a rectangular window function.

Tapered window

The window function can also be tapered in such a way that the amplitude of the window is smoothly reduced to zero near the edges of

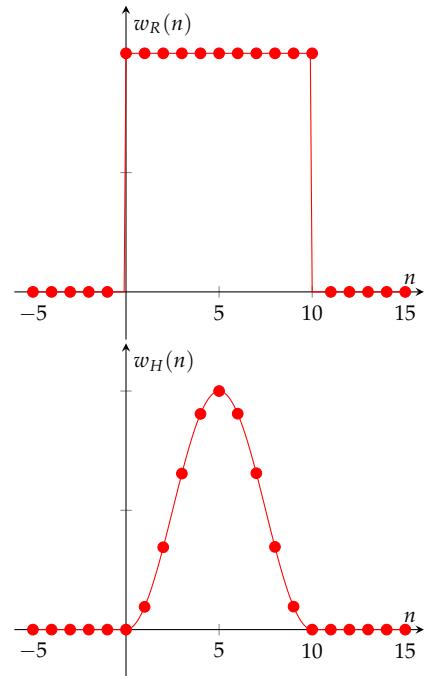


Figure 16.6: Top: Rectangular window. Bottom: Hann window. In both cases, the length of the window function is $N = 10$.

the window. This results in better rejection of out of band signals, at the expense of slightly broadening sharp frequency domain features.

One example of a good tapered window function is the Hann window. A Hann window of length N is defined as:

$$w_H[n] = \begin{cases} \sin^2(\pi n/N) & 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases} \quad (16.44)$$

A Hann window is shown on the bottom panel of Figure 16.6.

A selection of commonly used window function implementations can be found in the `scipy.signal` module. This includes an implementation of the Hann window function¹.

Example: Frequency selective filter

The ideal point-frequency filter for a real-valued signal with frequency $\hat{\omega}_0$ was shown in the discrete-time Fourier transform chapter to be:

$$h[n] = \cos(\hat{\omega}_0 n) \quad (16.45)$$

This is an infinitely long filter, and therefore it is not possible to implement this filter in practice. The windowed version of this filter of length N in the case of a rectangular window function would be:

$$h_{R,\hat{\omega}_0}[n] = \begin{cases} \cos(\hat{\omega}_0(n - N/2)) & 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases} \quad (16.46)$$

and using the Hann window:

$$h_{H,\hat{\omega}_0}[n] = \begin{cases} \sin^2(\pi n/N) \cos(\hat{\omega}_0(n - N/2)) & 0 \leq n \leq N-1 \\ 0 & \text{otherwise.} \end{cases} \quad (16.47)$$

The Python code in Listing 16.1 shows an implementation of a frequency selective filter using a rectangular and Hann window. The plot produced is shown in Figure 16.8.

The key difference between the rectangular windowed and the Hann windowed filter is the rejection of signals outside the frequencies that we are trying to filter. A rectangular filter has significantly worse rejection of out-of-band frequencies compared to the Hann windowed filter. This is why usually tapered window functions are used.

```
import matplotlib.pyplot as plt
import numpy as np
# Commonly used window functions are found in
# the scipy.signal.windows module.
from scipy.signal.windows import hann
```

¹ The Wikipedia page on Window functions https://en.wikipedia.org/wiki/Window_function contains a relatively comprehensive visual catalog of the frequency responses of different tapering window functions.

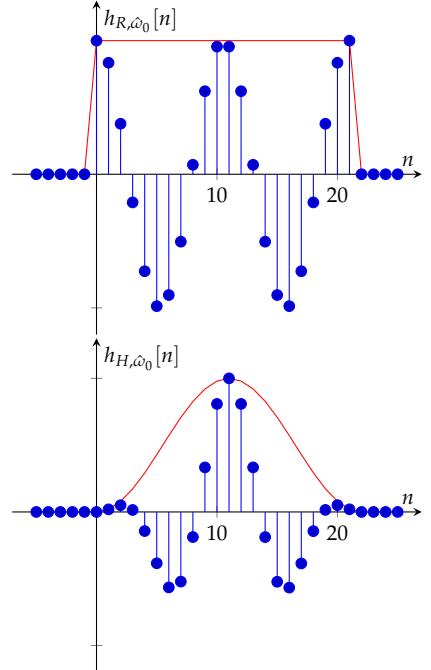


Figure 16.7: Top: Frequency selective filter impulse response using a rectangular window. Bottom: Frequency selective filter impulse response using a Hann window. In both cases, the length of the window function is $N = 22$. The window functions $w(n)$ are shown in red, and the filter impulse response in blue.

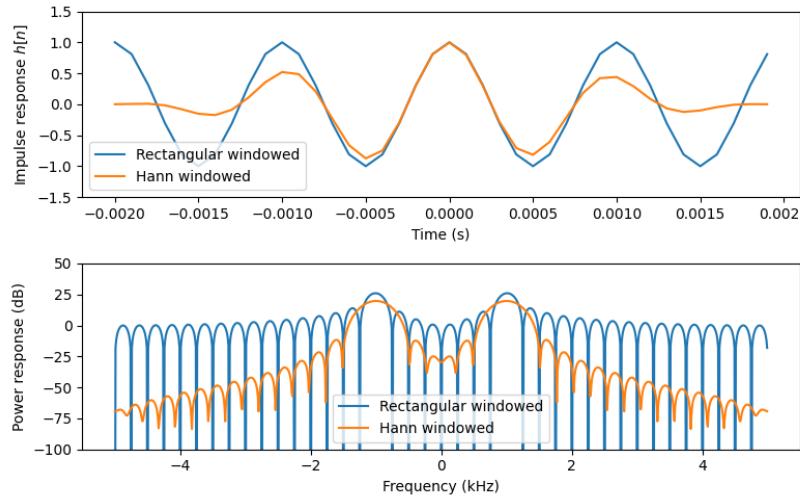


Figure 16.8: The impulse response and magnitude response of a frequency selective filter, which selects real-valued signals with frequencies $f_0 = 1 \text{ kHz}$. Top: The impulse response of the rectangular and Hann windowed impulse response. Bottom: the power response of the same filters. The Hann tapered filter has significantly better rejection of spectral components with frequencies outside $f_0 = 1\text{e}3$. However, the rectangular windowed filter has a slightly narrower main lobe near the 1 kHz frequency that the filter is designed to pass through. Due to time-frequency uncertainty, the filter has a finite width.

```
# Calculate the DIFT of signal x using zero-padded FFT
# at N evenly spaced points between  $-\pi$  and  $\pi$ .
def dft_dtft(x, N, sample_rate=1e3):
    # fftshift shifts the frequencies so that we obtain
    # normalized angular frequency between  $-\pi$  to  $\pi$ ,
    # instead of 0 to  $2\pi$ .
    X = np.fft.fftshift(np.fft.fft(x, N))

    # Normalized angular frequencies.
    freqs = np.fft.fftshift(np.fft.fftfreq(N, d=1.0/sample_rate))
    )

    return X, freqs

# Signal (FIR filter coefficients).
sample_rate = 10e3
t = np.arange(-20, 20)/sample_rate
# Make a filter that selects 1 kHz signals.
fo = 1e3

# Rectangular windowed frequency selective filter.
h_R = np.cos(2.0*np.pi*fo*t)

# Hann windowed frequency selective filter.
h_H = hann(40)*np.cos(2.0*np.pi*fo*t)

# Frequency response of both filters.
H_R, freqs = dft_dtft(h_R, 1000, sample_rate)
H_H, freqs = dft_dtft(h_H, 1000, sample_rate)
plt.figure(figsize=(8, 5))
plt.subplot(211)
plt.plot(t, h_R, label="Rectangular windowed")
plt.plot(t, h_H, label="Hann windowed")
plt.ylim([-1.5, 1.5])
plt.xlabel("Time (s)")
plt.ylabel("Impulse response $h[n]$")
plt.legend()
```

```

plt.subplot(212)
plt.plot(freqs/1e3, 10.0*np.log10(np.abs(H_R)**2.0), label="Rectangular windowed")
plt.plot(freqs/1e3, 10.0*np.log10(np.abs(H_H)**2.0), label="Hann windowed")
plt.xlabel("Frequency (kHz)")
plt.ylabel("Power response (dB)")
plt.legend()
plt.ylim([-100, 50])
plt.tight_layout()
plt.savefig("freq_filter.png")
plt.show()

```

Listing 16.1: 022_window_functions/frequency_selective.py

Example: Low-pass filter

In the discrete-time Fourier transform chapter, we derived the impulse response for an ideal low-pass filter:

$$h[n] = \frac{\sin(\hat{\omega}_0 n)}{\pi n}. \quad (16.48)$$

This is an infinitely long filter, so the only way to implement this in practice is to use a windowed version of the ideal impulse response:

$$h_w[n] = w(n) \frac{\sin(\hat{\omega}_0[n - N/2])}{\pi[n - N/2]}. \quad (16.49)$$

The Python code in Listing 16.2 shows how this is done in practice. The program evaluates the impulse response of a Hann windowed ideal low-pass filter and shows the magnitude response in dB scale. The output of this program is shown in Figure 16.9. While this filter does not perfectly remove spectral components with frequencies outside the passband, it does a pretty decent job. For example, frequencies $\hat{\omega} = \pm\pi$ are reduced in power by a factor larger than 10^{12} .

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.signal.windows import hann

om = np.linspace(-np.pi, np.pi, num=1000)

# Low pass filter cutoff frequency.
omc = 0.1*np.pi

# Sample indices for filter impulse response.
nn = np.arange(-100, 100)

# Ideal low-pass filter impulse response with cutoff at omc.
sinc = np.sin(omc*nn)/(np.pi*nn)
# Use L'Hopital's rule to determine value at nn=0.
sinc[nn == 0] = omc/np.pi

# Window function and length.

```

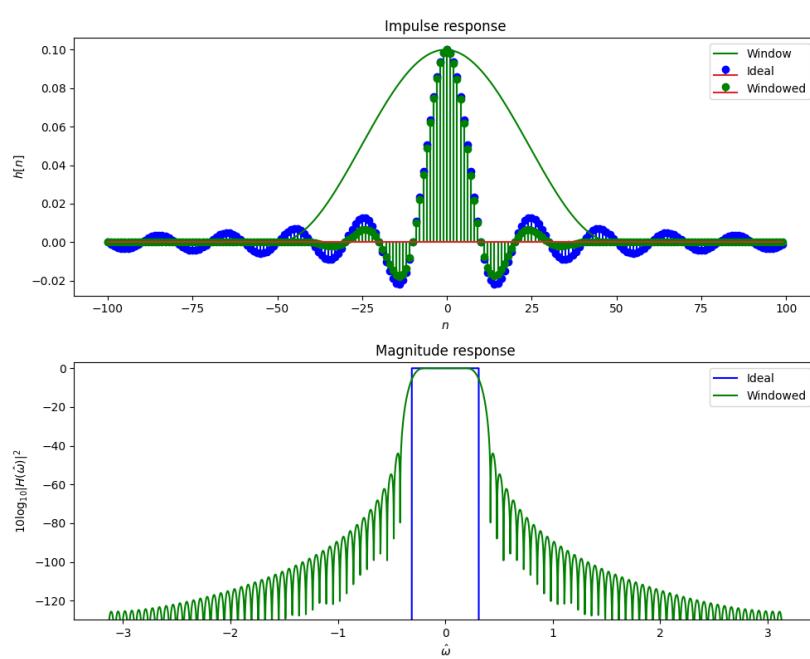


Figure 16.9: The impulse response and power response of a windowed low-pass filter. Top: the impulse response of the ideal filter is shown in blue, and the Hann windowed filter impulse response is shown in green. Bottom: the power response of the filter.

```

wf = np.zeros(len(nn))
wl = 100

# Hann window.
W = hann(wl)
# Center window function around 0.

wf[int(len(wf) / 2) - int(wl/2) + np.arange(wl)] = W

# Use a zero-padded DFT to approximate the frequency response
# of the windowed low-pass filter.
WX = np.fft.fftshift(np.fft.fft(wf*sinc, 4096))
# Vector of frequencies for DFT.
omw = np.linspace(-np.pi, np.pi, num=len(WX))

# Ideal frequency response (boxcar).
X = np.zeros(len(omw))
X[:] = 1e-120 # Very small number to avoid taking a log of 0.
X[np.abs(omw) < omc] = 1.0

# Plot power spectral response in dB scale for
# ideal and windowed filters.
plt.figure(figsize=(10, 8))
plt.subplot(212)
plt.plot(omw, 10.0*np.log10(np.abs(X)**2.0), label="Ideal", color="blue")
plt.plot(omw, 10.0*np.log10(np.abs(WX)**2.0), label="Windowed",
         color="green")
plt.ylim([-130, 3])
plt.legend()
plt.xlabel(r"\hat{\omega}")

```

```
plt.ylabel(r"$\log_{10}|H(\hat{\omega})|^2$")
plt.title("Magnitude response")

# Plot the impulse response of ideal and windowed filter.
plt.subplot(211)
plt.stem(nn, sinc, "b", markerfmt="bo", label="Ideal")
plt.plot(nn, np.max(sinc)*wf, label="Window", color="green")
plt.stem(nn, wf*sinc, "g", markerfmt="go", label="Windowed")
plt.legend()
plt.xlabel("$n$")
plt.ylabel("$h[n]$")
plt.title("Impulse response")
plt.tight_layout()
plt.savefig("windowed_lpf.png")
plt.show()
```

Listing 16.2: 022_window_functions/windowed_lpf.py

Exercises: Ideal and Tapered Filters

1. Define the ideal band-stop filter as:²

² That is $\mathcal{H}_{\text{BS}}(\hat{\omega}) = 1 - \mathcal{H}_{\text{BP}}(\hat{\omega})$.

$$\mathcal{H}_{\text{BS}}(\hat{\omega}) = \begin{cases} 0 & \hat{\omega}_0 < |\hat{\omega}| < \hat{\omega}_1 \\ 1 & \text{otherwise} \end{cases}. \quad (16.50)$$

Derive the impulse response for the band-stop filter.

2. Consider the band-stop filter you found in the previous exercise.

The goal of this exercise is to implement this filter in Python and apply it to filter an audio file.

- a) Let $w[n]$ be a tapered window function, like the Hann window of length N . Show that the windowed filter can be written as:

$$h_w[n] = \delta[n - N/2]w[n] + \frac{\sin(\hat{\omega}_0(n - N/2))}{(n - N/2)\pi}w[n] - \frac{\sin(\hat{\omega}_1(n - N/2))}{(n - N/2)\pi}w[n].$$

- b) Download the `crappy.wav` file from the link:

<https://bit.ly/3CvE89s>

The audio file contains a guitar playing, but it is impossible to hear due to noise. Your task is to filter out this noise using a band-stop filter. The noise in the signal is contained in a frequency band between $f_0 = 3$ kHz and $f_1 = 5$ kHz, but try to experiment with these to see what happens. Implement a band-stop filter using Python to remove the noise and recover the original audio. You can use the code shown in Listing 16.3 as a starting point.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sw
import scipy.signal as ss

# Read the noisy signal.
ts = sw.read("crappy.wav")
sr, crap = ts

# Filter length.
N = 4000

# Frequencies of noise (units of Hz).
fo = 3e3
f1 = 5e3

# Choose a window function.
w = ...
# ^ See the scipy.signal documentation for a list of window
# functions.

# Implement the filter and apply it here.
uncrap = ... # Complete this.
```

```
# Scale to 0.9, because 1.0 is the maximum allowed by the .wav
# file format.
uncrap = 0.9*uncrap/np.max(np.abs(uncrap))

# Save the filtered audio file.
sw.write("test_uncrappy.wav", sr, np.array(uncrap, dtype=np.
    float32))
```

Listing 16.3: Band-stop filter starting point

Suggested solutions: Ideal and Tapered Filters

- Let the ideal band-stop filter be given as:

$$\mathcal{H}_{BS}(\hat{\omega}) = \begin{cases} 0 & \hat{\omega}_0 < |\hat{\omega}| < \hat{\omega}_1 \\ 1 & \text{otherwise} \end{cases}.$$

The impulse response can then be computed using the inverse DTFT as follows (see Figure 16.10):

$$\begin{aligned} h[n] &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathcal{H}_{BS}(\hat{\omega}) e^{i\hat{\omega}n} d\hat{\omega}, \\ &= \frac{1}{2\pi} \int_{-\pi}^{-\hat{\omega}_1} e^{i\hat{\omega}n} d\hat{\omega} + \frac{1}{2\pi} \int_{-\hat{\omega}_0}^{\hat{\omega}_0} e^{i\hat{\omega}n} d\hat{\omega} + \frac{1}{2\pi} \int_{\hat{\omega}_1}^{\pi} e^{i\hat{\omega}n} d\hat{\omega}, \\ &= \frac{1}{2\pi} \left[\frac{1}{in} e^{i\hat{\omega}n} \right]_{-\pi}^{-\hat{\omega}_1} + \frac{1}{2\pi} \left[\frac{1}{in} e^{i\hat{\omega}n} \right]_{-\hat{\omega}_0}^{\hat{\omega}_0} + \frac{1}{2\pi} \left[\frac{1}{in} e^{i\hat{\omega}n} \right]_{\hat{\omega}_1}^{\pi}, \\ &= \frac{1}{2in\pi} (e^{-i\hat{\omega}_1 n} - e^{-i\pi n}) + \frac{1}{2in\pi} (e^{i\hat{\omega}_0 n} - e^{-i\hat{\omega}_0 n}) + \frac{1}{2in\pi} (e^{i\pi n} - e^{i\hat{\omega}_1 n}), \\ &= \frac{1}{n\pi} \sin(\pi n) + \frac{1}{n\pi} \sin(\hat{\omega}_0 n) - \frac{1}{n\pi} \sin(\hat{\omega}_1 n), \\ &= \delta[n] + \frac{\sin(\hat{\omega}_0 n)}{n\pi} - \frac{\sin(\hat{\omega}_1 n)}{n\pi}. \end{aligned}$$

As expected, the ideal stop-pass is just the opposite of an ideal band-pass filter.

- Continuing with the filter in the previous exercise.

- The impulse response of the band-stop filter was shown in the previous exercise to be:

$$h[n] = \delta[n] + \frac{\sin(\hat{\omega}_0 n)}{n\pi} - \frac{\sin(\hat{\omega}_1 n)}{n\pi}.$$

A tapered version of this filter is then found by:

$$h_w[n] = w(n)h[n - N/2],$$

where N is the length of the filter and the window function.

Putting all of this together, we obtain:

$$h_w[n] = \delta[n - N/2]w[n] + \frac{\sin(\hat{\omega}_0(n - N/2))}{(n - N/2)\pi}w[n] - \frac{\sin(\hat{\omega}_1(n - N/2))}{(n - N/2)\pi}w[n].$$

- An implementation of the band-stop filter is shown in Listing 16.4.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile as sw
from scipy.signal.windows import hann
```

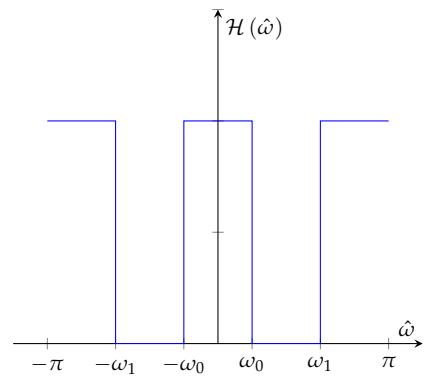


Figure 16.10: The frequency response of an ideal stop-pass filter.

```

# Read the noisy signal.
ts = sw.read("crappy.wav")
sr, crap = ts

# Filter length.
N = 4000

# Frequencies of noise (units of Hz).
fo = 3e3
f1 = 5e3

# Make the band a bit wider to hopefully remove some
# boundary effects.
sfo = fo - 0.2e3
sf1 = f1 + 0.2e3

# Calculate the discrete-time frequencies (units of rad /
# sample).
omo = 2.0*np.pi*sfo/sr
om1 = 2.0*np.pi*sf1/sr

# Declare the filter as 0.
h = np.zeros(N)

# Use a Hann window.
w = hann(N)

n = np.arange(N)

# Create a mask to compute the filter. Otherwise a zero-
# division occurs.
mask = (n != N//2)

# Compute the impulse response. Add the N//2 value
# afterwards.
h[mask] = (np.sin(omo*(n[mask] - N//2)) / ((n[mask] - N//2)
    *np.pi) - np.sin(om1*(n[mask] - N//2)) / ((n[mask] - N
    //2)*np.pi))*w[mask]
h[N//2] = (1 + (omo - om1) / np.pi) * w[N//2]

# In frequency domain, we multiply the filter with the
# spectral representation.
# Multiplication in frequency domain correspond to
# convolution in time domain
# therefore, convolve the band-stop filter and the signal.
uncrap = np.convolve(crap, h, mode="valid")
# ^ mode = valid will remove some boundary effects.

# Scale to 0.9, because 1.0 is the maximum allowed by the .
# wav file format.
uncrap = 0.9*uncrap/np.max(np.abs(uncrap))

# Save the filtered audio file.
sw.write("test_uncrappy.wav", sr, np.array(uncrap, dtype=np
    .float32))

```

Listing 16.4: Band-stop filter

In Figure 16.11 a plot of the impulse response for the band-stop filter is shown, both in time and frequency domain. In time

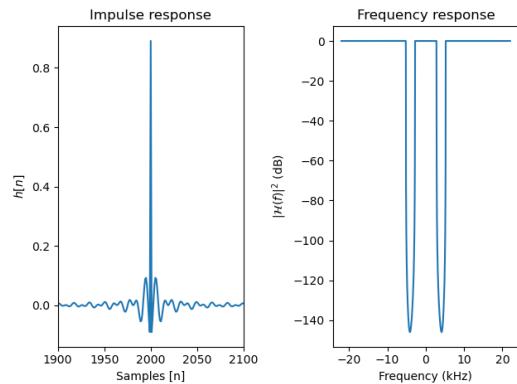


Figure 16.11: Impulse response for the band-stop filter

domain we see the Dirac delta in the middle and also the sinc function behavior. For the frequency domain plot the filter is close to what we would expect. Figure 16.12 shows that the frequency components corresponding to the band between f_0 and f_1 have been reduced in power. This is what one would except from a band-stop filter.

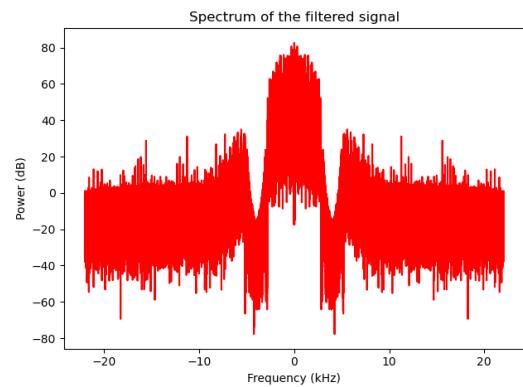


Figure 16.12: Frequency spectrum of the filtered signal

17

Time-frequency Uncertainty Principle

A fundamental property of the Fourier transform is that the width of a signal in time domain and in frequency domain are inversely related to one another. This relationship is known as the *time-frequency uncertainty principle*:

$$\Delta\omega\Delta t = \gamma . \quad (17.1)$$

Here $\gamma \in \mathbb{R}$ is a real-valued constant, which is dependent on how the “width” of a signal is defined in the time and frequency domain, Δt is the width of the signal in time domain, and $\Delta\omega$ is the width of the same signal in frequency domain.

Time-frequency uncertainty is a general relationship that applies to Fourier transform pairs. The units don't have to be time and frequency. One such uncertainty rule in quantum physics is the *Heisenberg uncertainty principle* which states that the width of the momentum wave function is inversely proportional to the width of the position wave function.

Example: Gaussian signal

Figure 17.1 shows a graphical representation of what Equation 17.1 means using a Gaussian signal, which has the following Fourier transform pair:

$$x(t) = e^{-\alpha t^2} \xleftrightarrow{\mathcal{F}} \hat{x}(\omega) = \sqrt{\frac{\pi}{\alpha}} e^{-\frac{\omega^2}{4\alpha}} . \quad (17.2)$$

If we define the width of a Gaussian to be $x(\Delta t) = x(0)e^{-1}$ in time domain and $\hat{x}(\Delta\omega) = \hat{x}(0)e^{-1}$ in frequency domain, we find that $\Delta t = \alpha^{-1/2}$ and $\Delta\omega = 2\alpha^{1/2}$. In this case, our time-frequency uncertainty rule would be:

$$\Delta\omega\Delta t = 2 . \quad (17.3)$$

What this relation means in practice can be seen in Figure 17.1. A Gaussian signal that is narrow in time domain is wide in frequency

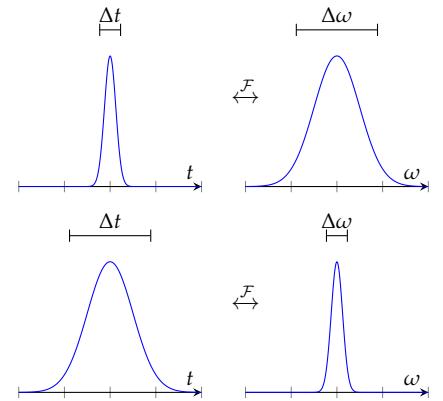


Figure 17.1: Above: A narrow signal in time domain is a wide signal in frequency domain. Below: A wide signal in time domain is a narrow signal in frequency domain.

domain. Conversely, a signal that is wide in time domain is narrow in frequency domain.

Time scaling

Recall the time scaling system from the chapter on Fourier transforms, which adjusts the width of a signal using the constant α :

$$y(t) = x(\alpha t). \quad (17.4)$$

The Fourier transform of a time scaled signal $\hat{y}(\omega)$ is scaled inversely in frequency:

$$\boxed{y(t) = x(\alpha t) \xrightarrow{\mathcal{F}} \hat{y}(\omega) = \frac{1}{|\alpha|} \hat{x}\left(\frac{\omega}{\alpha}\right)}. \quad (17.5)$$

What does scaling in time mean?

- compressing a signal in time with $|\alpha| > 1$ will stretch the signal in frequency.
- stretching a signal in time with $|\alpha| < 1$ will compress the signal in frequency.

Example: Diffraction limit

One example of the time-frequency uncertainty is the *diffraction limit*

$$\Delta\varphi\Delta D = \lambda, \quad (17.6)$$

which is encountered, e.g., in optics or in radio antenna theory. It provides the theoretical limit for angular resolution $\Delta\varphi$ of an aperture with diameter ΔD that collects electromagnetic waves. Here λ is the wavelength of the electromagnetic waves observed by the aperture. In this case, there is a Fourier transform relationship between $\Delta\varphi$ and ΔD . A derivation of the diffraction limit using a basic Fourier transform pair was shown in this video: <https://youtu.be/gKw46e4Ks4k>.

Example: minimum length of a low-pass filter impulse response

What is the minimum length required for a low-pass filter, in order to achieve a certain pass bandwidth?

Let's say that we wanted to low-pass filter a discrete-time signal with a cutoff frequency $\hat{\omega}_0$. Figure 17.3 shows the frequency response of an ideal low-pass filter. In this case, $\Delta\hat{\omega} = 2\hat{\omega}_0$ is a measure of how wide the filter is in frequency domain.

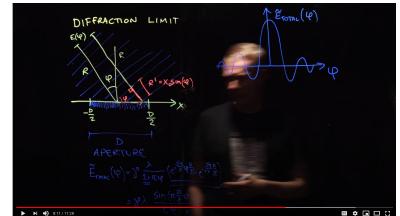


Figure 17.2: A video deriving the diffraction limit from the Fourier transform relationship between an aperture diameter D and the angular resolution $\Delta\varphi$ <https://youtu.be/gKw46e4Ks4k>.

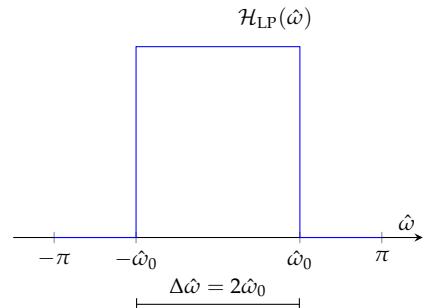


Figure 17.3: Ideal low-pass filter with cutoff $\hat{\omega}_0$ specified in discrete-time normalized angular frequency (radians per sample). The width of this filter in frequency domain is $\Delta\hat{\omega} = 2\hat{\omega}_0$.

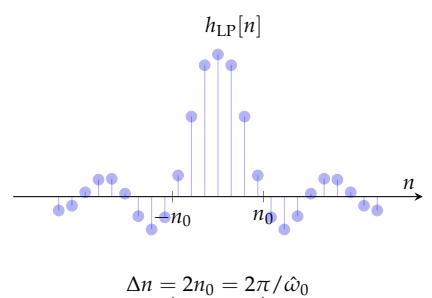


Figure 17.4: The impulse response of an ideal discrete-time low-pass filter $h[n]$. The width of this filter in time domain is $\Delta n = 2\pi/\hat{\omega}_0$ (samples).

In the discrete-time Fourier transform chapter, we derived the impulse response for the ideal low-pass filter to be:

$$h_{LP}[n] = \frac{\sin(\hat{\omega}_0 n)}{\pi n} . \quad (17.7)$$

This impulse response is shown in Figure 17.4.

We'll define the time domain width using the approximate first zero crossings of $h[n]$. This is denoted with $\pm n_0$ in Figure 17.4. We find that $n_0 = \pm\pi/\hat{\omega}_0$. Note that because this is a discrete-time signal, there might not be an actual sample that is zero valued and the value n_0 is the interpolated value where the impulse response crosses zero. The total effective width of the filter in time domain is then $\Delta n = 2\pi/\hat{\omega}_0$.

Now that we know the width in time and frequency domain, we can obtain the product:

$$\Delta\hat{\omega}\Delta n = 4\pi . \quad (17.8)$$

This is the time-frequency uncertainty principle applied to filter design. This tells us that in order to make a low pass filter with bandwidth $\Delta\hat{\omega}$, the impulse response of the filter (the length of the FIR filter) needs to be at least $\Delta n \geq 4\pi/\Delta\hat{\omega}$ samples long.

Note that the value of this constant depends on the definition of the width of the filter in time and frequency domain, and isn't necessarily exactly 4π . The only way to exhaustively characterize the time and frequency domain properties of a signal is to investigate the signal in time and frequency domain.

Exercises: Time-frequency Uncertainty Principle

1. The Hann window in continuous-time is defined as:

$$h_1(t) = \begin{cases} \sin^2(\pi t/T) & \text{when } 0 \leq t \leq T \\ 0 & \text{otherwise} \end{cases} \quad (17.9)$$

Let's define the time-domain width of this filter to be T . The Hann window is shown in Figure 17.5.

By using $\sin(\theta) = \frac{1}{2i}(e^{i\theta} - e^{-i\theta})$, it is possible to show that the Fourier transform of the Hann window is:

$$\mathcal{H}_1(\omega) = \left[\frac{\sin\left(\frac{T}{2}\omega\right)}{\omega} + \frac{1}{2} \frac{\sin\left(\pi - \frac{T}{2}\omega\right)}{\frac{2\pi}{T} - \omega} + \frac{1}{2} \frac{\sin\left(\pi + \frac{T}{2}\omega\right)}{\frac{2\pi}{T} + \omega} \right] e^{-i\frac{T}{2}\omega} \quad (17.10)$$

A rectangular window $h_2(t)$ of length T is also shown in Figure 17.5. The Fourier transform of a rectangular window of length T is:

$$\mathcal{H}_2(\omega) = \frac{2 \sin\left(\frac{T}{2}\omega\right)}{\omega} e^{-i\frac{T}{2}\omega} \quad (17.11)$$

Both of these filters can be considered to be low pass filters. It is impossible to unambiguously say that one of these has better frequency domain properties. The rectangular window has a more narrow pass bandwidth in frequency domain, while the Hann window has significantly better rejection of high frequency signals outside the pass band. Which one you should use will depend on your application.

- a) Show that Equation 17.10 is the Fourier transform of the signal in Equation 17.9.
- b) The largest value (peak) of the Hann window $h_1(t)$ as shown in Figure 17.5 is centered at time $\frac{T}{2}$. You can easily modify $\mathcal{H}_1(\omega)$ to obtain the frequency response of a Hann window, with the peak of the impulse response centered at $t = 0$ in time. Show how this can be done.
- c) Study the magnitude responses for the two filters in decibel scale by writing a program to plot $10 \log_{10}(|\mathcal{H}_1(\omega)|^2)$ and $10 \log_{10}(|\mathcal{H}_2(\omega)|^2)$. Use the frequency range $-10\pi < \omega < 10\pi$ and assume that $T = 1$.
- d) Define the widths of the two filters in frequency domain by inspecting the smallest value of ω for which $|\mathcal{H}(\omega)| = 0$, i.e., the first null of the magnitude response. You can also use Figure 17.6 to help you out. In the case of $\mathcal{H}_2(\omega)$, the first null is at $\omega_2 = 2\pi/T$ and the null-to-null width of the rectangular

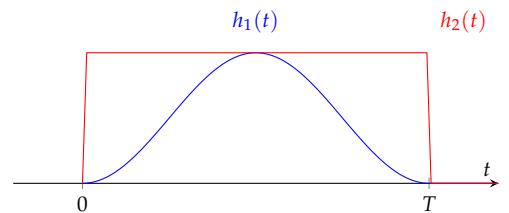


Figure 17.5: Two impulse responses: 1) a Hann window $h_1(t)$ of length T (blue), and 2) a rectangular window $h_2(t)$ of length T (red).

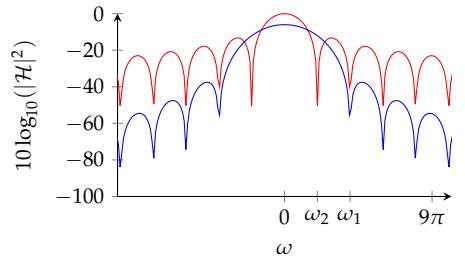


Figure 17.6: The magnitude responses for the Hann window (blue) and the rectangular window (red). In both cases, the length of the filter is $T = 1$. Both are shown in decibel scale.

filter is thus $\Delta\omega_2 = 2\omega_2 = 4\pi/T$. From Figure 17.6, it appears that the null-to-null width of the Hann filter in frequency domain is twice that of the rectangular window $\Delta\omega_1 = 2\Delta\omega_2$. Show that this is the case¹.

- e) How long do the two filters need (in seconds) to be in order to achieve a null-to-null pass bandwidth of $\Delta\omega = 2\pi$ rad/s?
- f) Recall that if a complex sinusoidal signal

$$x(t) = e^{i\omega' t} \quad (17.13)$$

is fed into an LTI system with impulse response $h(t)$, the output signal would be of the form:

$$y(t) = |\mathcal{H}(\omega')| e^{i\angle\mathcal{H}(\omega')} e^{i\omega' t} \quad (17.14)$$

Assuming that $T = 1$ s and $\omega' = 9\pi$ rad/s, what would the amplitudes of the complex sinusoidal signals coming out of the Hann and rectangular filters ($|\mathcal{H}_1(9\pi)|$ and $|\mathcal{H}_2(9\pi)|$) be? You don't necessarily need to obtain an exact number, you can look at Figure 17.6 and estimate it from there.

- g) While the rectangular filter has a more narrow filter width in frequency domain than a Hann window of the same length, the Hann window is in most cases significantly better when it comes to filtering out frequency components with frequencies outside the passband. Explain why by comparing the magnitude responses of the rectangular and Hann windows in decibel scale, shown in Figure 17.6.
2. Consider the following function:

$$\Psi(x) = \frac{1}{\sqrt{2T}} [u(x+T) - u(x-T)]. \quad (17.15)$$

The function $|\Psi(x)|^2$ describes the probability density of finding a particle at position x .

- a) Show that:

$$\begin{aligned} \hat{\Psi}(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \Psi(x) e^{-ikx} dx, \\ &= \frac{1}{\sqrt{T\pi}} \frac{\sin(kT)}{k}. \end{aligned}$$

Here $\hat{\Psi}(k)$ is the unitary Fourier transform² of $\Psi(x)$. The function $|\hat{\Psi}(k)|^2$ is the probability density function for the particle having the wavenumber k . If x has units of meters, then k has units of rad/m. In this case, the particle momentum and wavenumber are related as follows: $p = \hbar k$ with units of kg·m/s.

¹ Note that the width of the passband of a frequency symmetric low-pass filter in frequency domain is often given in terms of half-power width, not null-to-null width. In most cases, this needs to be numerically evaluated, so we haven't done this here. The half-power width is defined based on the following property:

$$|\mathcal{H}(\omega')|^2 = \frac{1}{2} |\mathcal{H}(0)|^2 \quad (17.12)$$

with the half-power width being $\Delta\omega_{1/2} = 2\omega'$.

² The unitary Fourier transform pair is defined as:

$$\begin{aligned} \hat{f}(\omega) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \\ f(t) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega t} d\omega \end{aligned}$$

This version of the Fourier transform results in the following variant of Plancherel's theorem:

$$\int_{-\infty}^{\infty} \hat{f}(\omega) \hat{g}^*(\omega) d\omega = \int_{-\infty}^{\infty} f(t) g^*(t) dt.$$

This theorem was derived in the Fourier transform chapter (see Equation 10.66).

- b) Define the width of the wave functions $\Psi(x)$ and $\hat{\Psi}(k)$.³ We'll define the width of the wavefunctions in the following way:

$$\Delta x = \min\{x : |\Psi(x)|^2 = 0, x > 0\},$$

and

$$\Delta k = \min\{k : |\hat{\Psi}(k)|^2 = 0, k > 0\},$$

or, in other words, Δx and Δk are the smallest positive values for which the functions $|\Psi(x)|^2$ and $|\hat{\Psi}(k)|^2$ are 0. Determine Δx and Δk for the functions $\Psi(x)$ and $\hat{\Psi}(k)$ and come up with a relation between Δx and Δk .

- c) Sketch the functions $\Psi(x)$ and $\hat{\Psi}(k)$ and indicate Δx and Δk in your drawing.
d) Show that:

$$\int_{-\infty}^{\infty} |\hat{\Psi}(k)|^2 dk = 1.$$

The meaning of this integral is that we are 100% certain that a particle will have a wavenumber in the range $-\infty < k < \infty$.

- e) Now consider another wavefunction that describes the position of a particle:

$$\Psi(x) = \delta(x - x_0), \quad (17.16)$$

where x_0 is a constant and x is the position of the particle.

In this case, you cannot say that one value of momentum or wavenumber is more probable than another. Explain why, by calculating the probability density of the particle wavenumber $|\hat{\Psi}(k)|^2$ corresponding to Equation 17.16. Here $\hat{\Psi}(k)$ is the unitary Fourier transform of $\Psi(x)$.

³ A solution to the Schrödinger equation is called a wave function. If you aren't familiar with quantum mechanics, just think of it as an ordinary function.

Suggested solutions: Time-frequency Uncertainty Principle

1. The Hann window in continuous-time is defined as:

$$h_1(t) = \begin{cases} \sin^2(\pi t/T), & 0 \leq t \leq T, \\ 0, & \text{otherwise.} \end{cases}$$

Define the length of the Hann filter to be T in time-domain. The Fourier transform of $h_1(t)$ can be shown to be:

$$\mathcal{H}_1(\omega) = \left[\frac{\sin\left(\frac{T}{2}\omega\right)}{\omega} + \frac{1}{2} \frac{\sin\left(\pi - \frac{T}{2}\omega\right)}{\frac{2\pi}{T} - \omega} + \frac{1}{2} \frac{\sin\left(\pi + \frac{T}{2}\omega\right)}{\frac{2\pi}{T} + \omega} \right] e^{-i\frac{T}{2}\omega}.$$

Let $h_2(t)$ denote the rectangular window of length T , for which the Fourier transform is:

$$\mathcal{H}_2(\omega) = \frac{2 \sin\left(\frac{T}{2}\omega\right)}{\omega} e^{-i\frac{T}{2}\omega},$$

which also has length taken to be T .

a) Using the Fourier transform we have:

$$\begin{aligned} \mathcal{H}_1(\omega) &= \int_{-\infty}^{\infty} \sin^2(\pi t/T) e^{-i\omega t} dt = \frac{1}{(2i)^2} \int_0^T \left(e^{i\pi t/T} - e^{-i\pi t/T} \right)^2 e^{-i\omega t} dt, \\ &= -\frac{1}{4} \int_0^T \left[e^{-i(\omega-2\pi/T)t} - 2e^{-i\omega t} + e^{-i(\omega+2\pi/T)t} \right] dt. \end{aligned}$$

Last handle each term on its own, the first term is computed as:

$$\begin{aligned} \int_0^T e^{-i(\omega-2\pi/T)t} dt &= \left[-\frac{1}{i(\omega - \frac{2\pi}{T})} e^{-i(\omega-2\pi/T)t} \right]_0^T = \frac{1}{i(\omega - \frac{2\pi}{T})} (1 - e^{-i(\omega T - 2\pi)}), \\ &= \frac{1}{i(\omega - \frac{2\pi}{T})} (e^{i(\omega T/2 - \pi)} - e^{-i(\omega T/2 - \pi)}) e^{-i(\omega T/2 - \pi)}, \\ &= \frac{2}{\omega - \frac{2\pi}{T}} \sin\left(\omega \frac{T}{2} - \pi\right) e^{-i\omega \frac{T}{2}} \overbrace{e^{i\pi}}^{-1}, \\ &= -\frac{2}{\frac{2\pi}{T} - \omega} \sin\left(-\left(\pi - \omega \frac{T}{2}\right)\right) e^{-i\omega \frac{T}{2}} (-1), \\ &= -\frac{2}{\frac{2\pi}{T} - \omega} \sin\left(\pi - \omega \frac{T}{2}\right) e^{-i\omega \frac{T}{2}}, \end{aligned}$$

the last step uses $\sin(-\theta) = -\sin(\theta)$. The second term is:

$$\begin{aligned} \int_0^T e^{-i\omega t} dt &= \left[-\frac{1}{i\omega} e^{-i\omega t} \right]_0^T = \frac{1}{i\omega} (1 - e^{-i\omega T}), \\ &= \frac{1}{i\omega} (e^{i\omega T/2} - e^{-i\omega T/2}) e^{-i\omega T/2} = \frac{2}{\omega} \sin\left(\omega \frac{T}{2}\right) e^{-i\omega \frac{T}{2}}, \end{aligned}$$

and finally, the same approach as the first gives:

$$\begin{aligned}
 \int_0^T e^{-i(\omega+2\pi/T)t} dt &= \left[-\frac{1}{i(\omega + \frac{2\pi}{T})} e^{-i(\omega+2\pi/T)t} \right]_0^T = \frac{1}{i(\omega + \frac{2\pi}{T})} (1 - e^{-i(\omega T + 2\pi)}), \\
 &= \frac{1}{i(\omega + \frac{2\pi}{T})} (e^{i(\omega T/2 + \pi)} - e^{-i(\omega T/2 + \pi)}) e^{-i(\omega T/2 + \pi)}, \\
 &= \frac{2}{\omega + \frac{2\pi}{T}} \sin\left(\omega \frac{T}{2} + \pi\right) e^{-i\omega \frac{T}{2}} e^{-i\pi}, \\
 &= -\frac{2}{\frac{2\pi}{T} + \omega} \sin\left(\pi + \omega \frac{T}{2}\right) e^{-i\omega \frac{T}{2}}.
 \end{aligned}$$

Putting everything together with the correct signs and factors we obtain:

$$H_1(\omega) = \left[\frac{\sin\left(\omega \frac{T}{2}\right)}{\omega} + \frac{1}{2} \frac{\sin\left(\pi - \omega \frac{T}{2}\right)}{\frac{2\pi}{T} - \omega} + \frac{1}{2} \frac{\sin\left(\pi + \omega \frac{T}{2}\right)}{\frac{2\pi}{T} + \omega} \right] e^{-i\frac{T}{2}\omega}$$

as desired.

- b) If the peak of the Hann window is at $t = T/2$, then we can shift the peak to be at $t = 0$ as follows:

$$h(t) = h_1\left(t - \frac{T}{2}\right) = \sin^2\left(\frac{\pi}{T}\left(t - \frac{T}{2}\right)\right) = \sin^2\left(\frac{\pi}{T}t - \frac{\pi}{2}\right) = \cos^2\left(\frac{\pi}{T}t\right),$$

as $\sin(\theta - \frac{\pi}{2}) = -\cos(\theta)$.

- c) To plot the spectral responses, we can use Listing 17.1.

```

import matplotlib.pyplot as plt
import numpy as np

def H1(omega: np.ndarray) -> np.ndarray:
    """Frequency response for the Hann window with T = 1.
    """
    return (np.sin(omega/2)/omega + (1/2)*np.sin(np.pi -
(1/2)*omega)/(2*np.pi - omega) + (1/2)*np.sin(np.pi +
(1/2)*omega)/(2*np.pi + omega))*np.exp(-1j*1/2*omega)

def H2(omega: np.ndarray) -> np.ndarray:
    """Frequency response for the rectangular window with T
    = 1.
    """
    return 2*np.sin(omega/2)/omega*np.exp(-1j*1/2*omega)

def convert_to_decibel(x: np.ndarray) -> np.ndarray:
    return 10*np.log10(np.power(np.abs(x), 2))

# Partition the interval (-10pi, 10pi) into 1000 equally
# spaced points.
x = np.linspace(-10*np.pi, 10*np.pi, num=1000)

```

```

# Plot the window functions to compare
# radians per sample on the x-axis and dB on the y-axis.
plt.plot(x, convert_to_decibel(H1(x)), label="Hann window")
plt.plot(x, convert_to_decibel(H2(x)), label="Rectangular
    window")
plt.xlabel(r"\hat{\omega}")
plt.ylim(-120, 10) # Limit the y-axis to (-120,10).
plt.legend()
# Call this if needed.
# plt.show()

```

Listing 17.1: Filter spectral response

The magnitude response is shown in Figure 17.7. The Figure shows that the Hann window manages the rejection of frequencies out of the band-pass better than the rectangular window.

- d) By inspecting Figure 17.6 we conclude that the rectangular window (in red) has its first zero at $\omega = \omega_2$, while the Hann window (in blue) has the first zero at $\omega = \omega_1$. By investigating the plot it appears that $\omega_1 = 2\omega_2$, where $\omega_2 = 2\pi/T$ as given. The width of the rectangular window is then $\Delta\omega_2 = 2\omega_2 = 4\pi/T$, meaning that the width of the Hann window is $\Delta\omega_1 = 2\omega_1 = 2(4\pi/T) = 2\Delta\omega_2 = 8\pi/T$.

- e) We've shown that:

$$\begin{aligned}\Delta\omega_1 &= 4\pi/T, \\ \Delta\omega_2 &= 8\pi/T,\end{aligned}$$

so the rectangular window needs to be $T = 2$ seconds long, while the Hann window needs to be $T = 4$ seconds long.

- f) By Figure 17.6: the Hann window cuts the power to approximately -60 dB, which is around 10^{-6} , while the rectangular window only cuts around -20 dB which corresponds to 10^{-2} , so not that much reduction. The Hann window does a far better job in reducing the edges than the rectangular window.
- g) As discussed in f) the Hann window is much better at handling frequencies outside the band-pass. This can be seen from the plot of the power of the magnitude response. Thus, the Hann window can be used much more efficiently to reduce spectral leakage which in many cases can be a huge problem, so the trade-off when comparing filter width to spectral leakage, reducing the leakage is preferred over reducing the filter width.

2. Let $\Psi(x)$ be given as:

$$\Psi(x) = \frac{1}{\sqrt{2T}}[u(x+T) - u(x-T)].$$

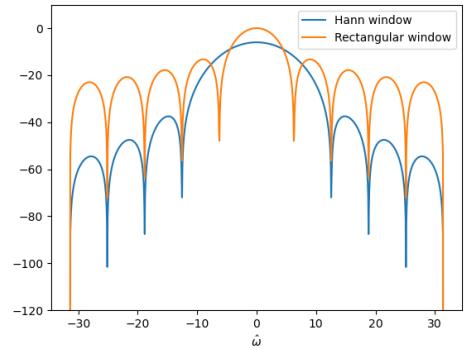


Figure 17.7: Comparison of the spectral responses of two filters

- a) Using the unitary Fourier transform, we can directly compute it as:

$$\begin{aligned}
 \hat{\Psi}(k) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \Psi(x) e^{-ikx} dx, \\
 &= \frac{1}{\sqrt{2\pi}\sqrt{2T}} \int_{-\infty}^{\infty} [u(x+T) - u(x-T)] e^{-ikx} dx, \\
 &= \frac{1}{\sqrt{4\pi T}} \int_{-T}^{T} e^{-ikx} dx, \\
 &= \frac{1}{2ik\sqrt{\pi T}} [e^{ikT} - e^{-ikT}], \\
 &= \frac{1}{\sqrt{T\pi}} \frac{\sin(kT)}{k},
 \end{aligned}$$

where we've used that $\sin \theta = \frac{1}{2i}(e^{i\theta} - e^{-i\theta})$.

- b) Have that:

$$\begin{aligned}
 |\Psi(x)|^2 &= \frac{1}{2T} [u(x+T) - u(x-T)], \\
 |\hat{\Psi}(k)|^2 &= \frac{1}{T\pi} \frac{\sin^2(kT)}{k^2}.
 \end{aligned}$$

For $|\Psi(x)|^2$, we see that $\Delta x = T$ and for $|\hat{\Psi}(k)|^2$, we get $\Delta k = \pi/T$. The time-frequency uncertainty can then be expressed as

$$\Delta x \Delta k = (T) \left(\frac{\pi}{T} \right) = \pi.$$

- c) The wave functions $\Psi(x)$ and $\hat{\Psi}(k)$ are shown in Figure 17.8 and Figure 17.9, respectively.

- d) By Parseval's theorem, we have, in the unitary case:

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |\hat{x}(\omega)|^2 d\omega.$$

Using this, we get:

$$\begin{aligned}
 \int_{-\infty}^{\infty} |\Psi(x)|^2 dx &= \int_{-\infty}^{\infty} \frac{1}{2T} [u(x+T) - u(x-T)] dx, \\
 &= \frac{1}{2T} \int_{-T}^{T} dx = 1,
 \end{aligned}$$

thus, by the unitary form of Parseval's theorem, we get:

$$\int_{-\infty}^{\infty} |\hat{\Psi}(k)|^2 dk = \int_{-\infty}^{\infty} |\Psi(x)|^2 dx = 1.$$

- e) If the wavefunction for position is given as $\delta(x - x_0)$, then the wavefunction for the wavenumber (or momentum) is given by the unitary Fourier transform as:

$$\hat{\Psi}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \delta(x - x_0) e^{-ikx} dx = \frac{1}{\sqrt{2\pi}} e^{-ikx_0}.$$

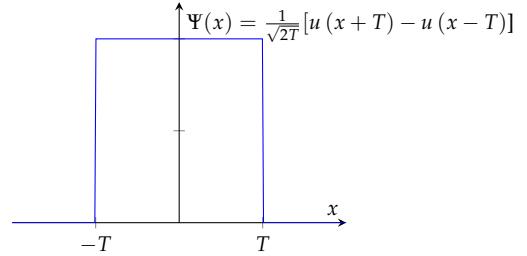


Figure 17.8: The wave function $\Psi(x)$ in position space, with $\Delta x = T$.

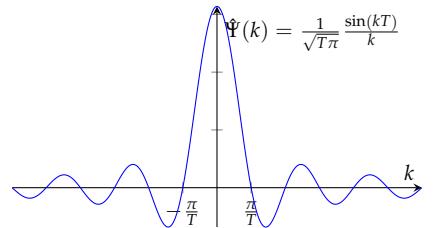


Figure 17.9: The wave function $\hat{\Psi}(k)$ in wavenumber space, with $\Delta k = \frac{\pi}{T}$.

Then the modulus of the wavefunction is proportional to the probability distribution for the wavenumber, so that:

$$|\hat{\Psi}(k)|^2 = \frac{1}{\sqrt{2\pi}} e^{-ikx_0} \frac{1}{\sqrt{2\pi}} e^{ikx_0} = \frac{1}{2\pi}.$$

This shows that all wavenumbers are equally likely, so no, we cannot say that any wavenumber/momentum is more probable than any other.

18

Discrete Fourier Transform

In this chapter, we will introduce the *discrete Fourier transform* (DFT). We will also briefly discuss the *fast Fourier transform* (FFT) algorithm, which is a very efficient numerical algorithm that can be used to evaluate a DFT.

The importance of the DFT and the FFT algorithm cannot be sufficiently stressed. Its use is ubiquitous in modern signal processing. Much of the technology that you use in your everyday life is dependent on this algorithm. Whenever you are watching a video or displaying an image on your computer screen, listening to digital music, or using a wireless internet connection, you are with a high likelihood using technology that relies on the DFT implemented using the FFT algorithm.

The DFT is widely used in practical signal processing applications, which are impossible to exhaustively list here. For example, the DFT can be used:

- to approximate a Fourier transform
- to approximate a discrete-time Fourier transform,
- for spectral analysis of signals,
- to efficiently implement a convolution of two signals,
- in spectral compression algorithms (e.g., the MP3 and JPEG sound and image formats),
- to interpolate and re-sample signals,
- for filtering signals,
- for modulation and demodulation of radio signals in telecommunications, or
- to solve differential equations (e.g., Poisson's equation).

This is just a small fraction of the applications of the DFT and the FFT algorithm.

Fourier Series

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{i \frac{2\pi k}{T} t}$$

$$c_k = \frac{1}{T} \int_T x(t) e^{-i \frac{2\pi k}{T} t} dt$$

Periodic in time:

$$x(t+T) = x(t)$$

Fourier Transform

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{x}(\omega) e^{i\omega t} d\omega$$

$$\hat{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt$$

Discrete-time Fourier Transform

$$x[n] = \frac{1}{2\pi} \int_{2\pi}^{\infty} \hat{x}(\hat{\omega}) e^{i\hat{\omega} n} d\hat{\omega}$$

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-i\hat{\omega} n}$$

Periodic in frequency:

$$\hat{x}(\hat{\omega}) = \hat{x}(\hat{\omega} + 2\pi)$$

Discrete Fourier Transform

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}[k] e^{i \frac{2\pi}{N} kn}$$

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} kn}$$

Periodic in time and frequency:

$$x[n+N] = x[n]$$

$$\hat{x}[k+N] = \hat{x}[k]$$

Figure 18.1: A summary of all the frequency domain representations of a signal covered so far in this course. The discrete Fourier transform will be covered in this chapter.

Derivation of the discrete Fourier transform

The DFT is a frequency domain representation for periodic discrete-time signals $x[n] = x[n + N]$. Because the time domain signal is discrete-time and periodic, it follows that the frequency domain representation is also a periodic discretized signal $\hat{x}[k] = \hat{x}[k + N]$. Let's see why.

Consider the following continuous-time signal, which has a period of NT_s :

$$x_d(t) = \sum_{p=-\infty}^{\infty} \sum_{n=0}^{N-1} x[n] \delta(t - nT_s - pNT_s). \quad (18.1)$$

This is a continuous-time representation of a periodic discrete-time signal $x[n]$ with period N , sampled with sample-spacing $T_s = 1/f_s$. Here f_s is sample-rate. The index p is used to create infinitely many copies of the signal with time offsets of pNT_s . I've made an example plot of this type of signal in Figure 18.2. In the example $N = 3$, which means that $x_d(t) = x_d(t + 3T_s)$.

Because this signal is periodic, we can use the Fourier series to represent this signal as a sum of complex sinusoidal signals:

$$x_d(t) = \sum_{k=-\infty}^{\infty} c_k e^{i \frac{2\pi}{NT_s} kt} = \sum_{k=-\infty}^{\infty} c_k e^{i \omega_k t}. \quad (18.2)$$

We can use the Fourier series analysis formula to determine the coefficients c_k :

$$c_k = \frac{1}{NT_s} \int_0^{NT_s} x_d(t) e^{-i \frac{2\pi}{NT_s} kt} dt \quad (18.3)$$

$$= \frac{1}{NT_s} \int_0^{NT_s} \left(\sum_{p=-\infty}^{\infty} \sum_{n=0}^{N-1} x[n] \delta(t - nT_s - pNT_s) \right) e^{-i \frac{2\pi}{NT_s} kt} dt \quad (18.4)$$

$$= \frac{1}{NT_s} \sum_{n=0}^{N-1} x[n] \int_0^{NT_s} \delta(t - nT_s) e^{-i \frac{2\pi}{NT_s} knT_s} dt \quad (18.5)$$

$$= \frac{1}{NT_s} \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{NT_s} knT_s} \quad (18.6)$$

$$= \frac{1}{NT_s} \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} kn} \quad (18.7)$$

$$= \frac{1}{NT_s} \hat{x}[k]. \quad (18.8)$$

The last term

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} kn}, \quad (18.9)$$

is the discrete Fourier transform. It is up to a constant $1/NT_s$, the Fourier series coefficient that corresponds to the spectral component

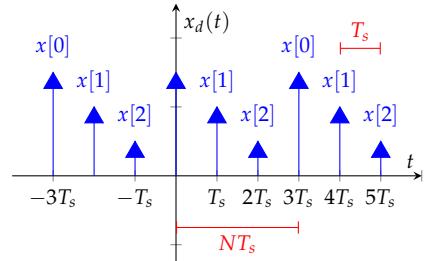


Figure 18.2: A continuous-time representation of a periodic discretized signal. In this example, the period of the discrete-time signal is $N = 3$, which means that $x[n] = x[n + 3]$. The sample-spacing is T_s , which implies that the period of this signal in continuous-time is NT_s .

with angular frequency

$$\omega_k = \frac{2\pi}{NT_s} k, \quad (18.10)$$

in a Fourier series representation of $x_d(t)$. This equation can be used to relate the k th spectral component to continuous-time angular frequency.

By inspecting Equation 18.9, it is easy to see that $\hat{x}[k] = \hat{x}[k + N]$. This means that the Fourier series coefficients repeat every N points.

We can Fourier transform $x_d(t)$ to obtain the frequency domain representation:

$$\hat{x}_d(\omega) = \int_{-\infty}^{\infty} x_d(t) e^{-i\omega t} d\omega \quad (18.11)$$

$$= \int_{-\infty}^{\infty} \left(\sum_{k=-\infty}^{\infty} c_k e^{i\omega_k t} \right) e^{-i\omega t} d\omega \quad (18.12)$$

$$= \sum_{k=-\infty}^{\infty} c_k \int_{-\infty}^{\infty} e^{i(\omega_k - \omega)t} d\omega \quad (18.13)$$

$$= \frac{2\pi}{NT_s} \sum_{k=-\infty}^{\infty} \hat{x}[k] \delta(\omega_k - \omega) \quad (18.14)$$

$$= \frac{2\pi}{NT_s} \sum_{p=-\infty}^{\infty} \sum_{k=0}^{N-1} \hat{x}[k] \delta(\omega_k - \omega - p\omega_s). \quad (18.15)$$

In the last line, I have emphasized the fact that $\hat{x}[n]$ is a periodic signal with period N . Periodicity of $\hat{x}[k]$ implies that $\hat{x}_d(\omega)$ is also periodic $\hat{x}_d(\omega) = \hat{x}_d(\omega + \omega_s)$ with period $\omega_s = 2\pi f_s$.

An example plot of $\hat{x}_d(\omega)$ scaled by $NT_s/2\pi$ is shown in Figure 18.3. In this example $N = 3$, which means that $\hat{x}[k] = \hat{x}[k + 3]$.

By comparing equations 18.15 and 18.1, we can see that both are similar continuous-time representations of a periodic discrete signal. One is a periodic discrete-time signal $x[n] = x[n + N]$ and the other is a periodic discrete-frequency signal $\hat{x}[k] = \hat{x}[k + N]$. The period of both signals is N .

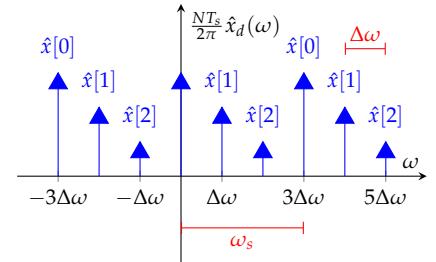


Figure 18.3: The spectral representation of a periodic discretized signal $\hat{x}_d(\omega) = \hat{x}_d(\omega + \omega_s)$, multiplied by $NT_s/2\pi$. In this example, $N = 3$. The frequency step between unit impulses is $\Delta\omega = \frac{\omega_s}{N}$.

Discrete Fourier Transform

The discrete Fourier transform (DFT) is defined as:

$$\hat{x}[k] = \mathcal{F}_D\{x[n]\} = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}nk}. \quad (18.16)$$

It is used to obtain the complex amplitude of the spectral components for normalized angular frequencies¹:

$$\hat{\omega}_k = \frac{2\pi}{N}k. \quad (18.17)$$

We use the notation $\hat{x}[k]$ to denote the discrete nature of the spectral representation.

The inverse operation, i.e., the operation that converts the frequency domain representation $\hat{x}[k]$ into a time domain representation $x[n]$ is the inverse discrete Fourier transform (IDFT), which is defined as:

$$x[n] = \mathcal{F}_D^{-1}\{\hat{x}[k]\} = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}[k] e^{i\frac{2\pi}{N}nk}. \quad (18.18)$$

The term $1/N$ is a required normalization constant, which is by convention typically attached to the inverse transform.

We'll prove the reversibility of the transform $x[n] = \mathcal{F}_D^{-1}\{\mathcal{F}_D\{x[n]\}\}$ by expressing the DFT using orthogonal basis vectors ψ_k .

Orthogonal basis vectors

Let's first define a phase constant $\phi = e^{-i\frac{2\pi}{N}}$ in \mathbb{C} , which implies that

$$\phi^{kn} = e^{-i\frac{2\pi}{N}kn}, \quad (18.19)$$

we can use this to define basis vectors $\psi_k \in \mathbb{C}^{N \times 1}$ as:

$$\psi_k = [\phi^{k \cdot 0} \quad \phi^{k \cdot 1} \quad \phi^{k \cdot 2} \quad \dots \quad \phi^{k \cdot (N-1)}]^T. \quad (18.20)$$

All N unique basis vectors orthogonal with one another, i.e., the inner product is:

$$\psi_k^H \psi_\ell = \sum_{n=0}^{N-1} e^{i\frac{2\pi}{N}(k-\ell)n} \quad (18.21)$$

$$= N \delta_{k,\ell}. \quad (18.22)$$

I've used the Kronecker delta $\delta_{k,\ell}$ here, which is defined as:

$$\delta_{k,\ell} = \begin{cases} 1 & k = \ell \\ 0 & k \neq \ell \end{cases}. \quad (18.23)$$

¹ With Python, you can use the `numpy.fft.freq` command to calculate the continuous-time frequency f_k corresponding to each frequency component $\hat{x}[k]$.

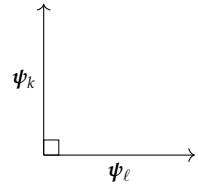


Figure 18.4: The discrete Fourier transform basis vectors are orthogonal $\psi_k \perp \psi_\ell$ for all $\ell \neq k$. This implies that $\psi_k^H \psi_\ell = 0$ when $\ell \neq k$.

The proof is divided into two cases. First, when $k = \ell$:

$$\psi_k^H \psi_k = \sum_{n=0}^{N-1} e^{i \frac{2\pi}{N} (k-k)n} = \sum_{n=0}^{N-1} e^0 = N . \quad (18.24)$$

and when $k \neq \ell$

$$\psi_k^H \psi_\ell = \sum_{n=0}^{N-1} e^{i \frac{2\pi}{N} n(k-\ell)} = S . \quad (18.25)$$

We can use a geometric series to solve for S :

$$\begin{aligned} S &= \alpha^0 + \cdots + \alpha^{N-1} \\ \alpha S &= \alpha^1 + \cdots + \alpha^N \\ S - \alpha S &= 1 - \alpha^N \\ S &= \frac{1 - \alpha^N}{1 - \alpha} . \end{aligned}$$

With $\alpha = e^{i \frac{2\pi}{N} (k-\ell)}$, we obtain:

$$\sum_{n=0}^{N-1} e^{i \frac{2\pi}{N} (k-\ell)n} = \frac{1 - 1}{1 - e^{i \frac{2\pi}{N} (k-\ell)}} = 0 . \quad (18.26)$$

This proves that the discrete Fourier transform basis vectors ψ_k are orthogonal.

Reversibility of the transform

Let's define a vector \hat{x} that contains the N spectral components $\hat{x}[k]$ obtained using a DFT:

$$\hat{x} = [\hat{x}[0] \quad \hat{x}[1] \quad \hat{x}[2] \quad \cdots \hat{x}[N-1]]^T \in \mathbb{C}^{N \times 1} . \quad (18.27)$$

We also define a vector x , which contains the N elements of the periodic signal $x[n]$:

$$x = [x[0] \quad x[1] \quad x[2] \quad \cdots \quad x[N-1]]^T \in \mathbb{C}^{N \times 1} . \quad (18.28)$$

Finally, let's define a matrix $F \in \mathbb{C}^{N \times N}$ by stacking the N basis vectors ψ_k :

$$F = \begin{bmatrix} \psi_0^T \\ \psi_1^T \\ \psi_2^T \\ \vdots \\ \psi_{N-1}^T \end{bmatrix} = \begin{bmatrix} \phi^{0,0} & \phi^{0,1} & \phi^{0,2} & \cdots & \phi^{0,(N-1)} \\ \phi^{1,0} & \phi^{1,1} & \phi^{1,2} & \cdots & \phi^{1,(N-1)} \\ \phi^{2,0} & \phi^{2,1} & \phi^{2,2} & \cdots & \phi^{2,(N-1)} \\ \vdots & & & \ddots & \vdots \\ \phi^{(N-1),0} & \phi^{(N-1),1} & \phi^{(N-1),2} & \cdots & \phi^{(N-1),(N-1)} \end{bmatrix} .$$

Using this matrix, we can express the forward and reverse discrete Fourier transforms as matrix-vector products:

$$x[n] = \mathcal{F}_D^{-1}\{\hat{x}[k]\} \xrightarrow{\mathcal{F}_D} \hat{x}[k] = \mathcal{F}_D\{x[n]\} \quad (18.29)$$

$$\mathbf{x} = \frac{1}{N} \mathbf{F}^H \hat{\mathbf{x}} \xrightarrow{\mathcal{F}_D} \hat{\mathbf{x}} = \mathbf{F} \mathbf{x}. \quad (18.30)$$

We can now prove reversibility. A combination of a forward and reverse transform $x[n] = \mathcal{F}_D^{-1}\{\mathcal{F}_D\{x[n]\}\}$ is equivalent to:

$$\mathbf{x} = \frac{1}{N} \mathbf{F}^H \mathbf{F} \mathbf{x}. \quad (18.31)$$

We can see that $\mathbf{F}^H \mathbf{F} = N \mathbf{I}$ is an identity matrix multiplied with N :

$$\mathbf{F}^H \mathbf{F} = \begin{bmatrix} \psi_0^H \psi_0 & \psi_0^H \psi_1 & \psi_0^H \psi_2 & \cdots & \psi_0^H \psi_{N-1} \\ \psi_1^H \psi_0 & \psi_1^H \psi_1 & \psi_1^H \psi_2 & \cdots & \psi_1^H \psi_{N-1} \\ \psi_2^H \psi_0 & \psi_2^H \psi_1 & \psi_2^H \psi_2 & \cdots & \psi_2^H \psi_{N-1} \\ \vdots & & \ddots & & \vdots \\ \psi_{N-1}^H \psi_0 & \psi_{N-1}^H \psi_1 & \psi_{N-1}^H \psi_2 & \cdots & \psi_{N-1}^H \psi_{N-1} \end{bmatrix} = N \mathbf{I}. \quad (18.32)$$

This means that a combination of a forward and inverse transform results in the original discrete-time signal \mathbf{x} :

$$\mathbf{x} = \frac{1}{N} \mathbf{F}^H \mathbf{F} \mathbf{x} \quad (18.33)$$

$$= \mathbf{x} \quad \square. \quad (18.34)$$

This proves the reversibility of the transform.

Example: Periodic unit impulse

Let us consider a periodic discrete-time signal $\{x[n]\}_{n=0}^{N-1} = \{1, 0, 0, 0\}$ of length $N = 4$. This signal is periodic, so $x[n+N] = x[n]$. The signal is shown in Figure 18.5.

The DFT of this signal is:

$$\hat{x}[k] = \sum_{n=0}^3 x[n] e^{-i \frac{2\pi}{4} kn} \quad (18.35)$$

$$= 1. \quad (18.36)$$

The spectrum $\hat{x}[k]$ is plotted in the bottom half of Figure 18.5. If we now take the inverse DFT of signal $\hat{x}[k]$, we get:

$$x[n] = \frac{1}{4} \sum_{k=0}^3 e^{i \frac{2\pi}{4} kn}. \quad (18.37)$$

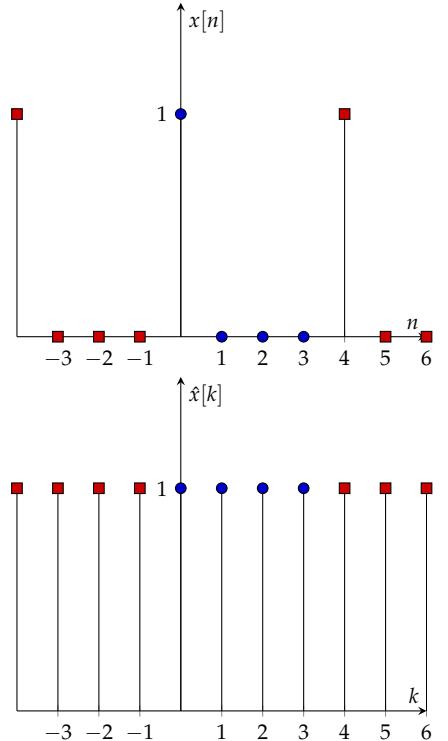


Figure 18.5: A periodic unit impulse in time domain results in a constant valued frequency domain representation.

For different values of n , we get:

$$\begin{aligned}x[0] &= \frac{1}{4}(e^{i\frac{2\pi}{4}0\cdot0} + e^{i\frac{2\pi}{4}1\cdot0} + e^{i\frac{2\pi}{4}2\cdot0} + e^{i\frac{2\pi}{4}3\cdot0}) = \frac{1}{4}(1+1+1+1) = 1, \\x[1] &= \frac{1}{4}(e^{i\frac{2\pi}{4}0\cdot1} + e^{i\frac{2\pi}{4}1\cdot1} + e^{i\frac{2\pi}{4}2\cdot1} + e^{i\frac{2\pi}{4}3\cdot1}) = \frac{1}{4}(1+i-1-i) = 0, \\x[2] &= \frac{1}{4}(e^{i\frac{2\pi}{4}0\cdot2} + e^{i\frac{2\pi}{4}1\cdot2} + e^{i\frac{2\pi}{4}2\cdot2} + e^{i\frac{2\pi}{4}3\cdot2}) = \frac{1}{4}(1-1+1-1) = 0, \\x[3] &= \frac{1}{4}(e^{i\frac{2\pi}{4}0\cdot3} + e^{i\frac{2\pi}{4}1\cdot3} + e^{i\frac{2\pi}{4}2\cdot3} + e^{i\frac{2\pi}{4}3\cdot3}) = \frac{1}{4}(1-i-1+i) = 0.\end{aligned}$$

Which is the original signal $x[n]$.

Relationship to the discrete-time Fourier transform

The DTFT of a discrete-time signal is:

$$\hat{x}(\hat{\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\hat{\omega}n}. \quad (18.38)$$

This is not a DFT, as $\hat{x}(\hat{\omega})$ is a continuous function of $\hat{\omega}$, whereas $\hat{x}[k]$ is discrete in frequency.

For a finite-length signal $x[n] = 0$ when $n > N - 1$ and $x[n] = 0$ when $n < 0$, the discrete-time Fourier transform is:

$$\hat{x}(\hat{\omega}) = \sum_{n=0}^{N-1} x[n]e^{-i\hat{\omega}n}. \quad (18.39)$$

Recall now the definition of the DFT:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n]e^{-ik\frac{2\pi}{N}n} \quad (18.40)$$

$$= \hat{x}(\hat{\omega}_k). \quad (18.41)$$

This means that the DFT is the DTFT evaluated at discrete points:

$\hat{\omega}_k = \frac{2\pi}{N}k$

(18.42)

The DFT can be seen as a special case of the DTFT. Note that nothing about periodicity of signal $x[n]$ was assumed. We just assumed that it was finite in length.

Zero-padded DFT

This immediately suggests a procedure for evaluating a DTFT $\hat{x}(\hat{\omega})$ on M evenly spaced frequencies $\{0, \frac{2\pi}{M}, \frac{4\pi}{M}, \dots, (M-1)\frac{2\pi}{M}\}$ using a DFT, where $M > N$ is an integer larger than N , the length of the finite length discrete-time signal $x[n]$.

We can do this with a zero-padded DFT. Zero padding means that we define a zero-padded signal $x_{\text{zp}}[n]$ of length M , which has values

$$x_{\text{zp}}[n] = \begin{cases} x[n] & 0 \leq n < N \\ 0 & N \leq n < M \end{cases}. \quad (18.43)$$

We then perform an M -point DFT:

$$\hat{x}_{\text{zp}}[k] = \hat{x}(\hat{\omega}_k) = \sum_{n=0}^{M-1} x_{\text{zp}}[n] e^{-i \frac{2\pi}{M} kn}. \quad (18.44)$$

This means that we can evaluate numerically the DTFT of a finite length signal at M evenly spaced points along the frequency axis. We can select M freely, to allow for arbitrary spacing of frequencies at which the DTFT is evaluated.

Example: Zero-padded DTFT estimate using DFT

The Python code in Listing 18.1 demonstrates how to estimate the DTFT of a signal using a DFT. In this example, we are estimating the DTFT of the running average filter,

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k], \quad (18.45)$$

which is an FIR filter with filter coefficients:

$$h[n] = \frac{1}{L} \sum_{k=0}^{L-1} \delta[n-k]. \quad (18.46)$$

The filter has an analytic DTFT that was found to be:

$$\mathcal{H}(\hat{\omega}) = \frac{1}{L} \frac{1 - e^{-i\hat{\omega}L}}{1 - e^{-i\hat{\omega}}}. \quad (18.47)$$

The Python program will first calculate the DTFT of the filter $h[n]$ using the analytic formula. It will then calculate the values of the DTFT at discrete points by using a zero-padded DFT (this is done using the `numpy.fft.fft` function). The output of the program is shown in Figure 18.6. The solid line shows the analytic magnitude response, and the step plot shows the values estimated using a DFT. By increasing the value of N , it is possible to increase the number of points that are sampled.

```
import matplotlib.pyplot as plt
import numpy as np

# Calculate the DTFT of signal x using zero-padded DFT
# at N evenly spaced points between -\pi and \pi.
```

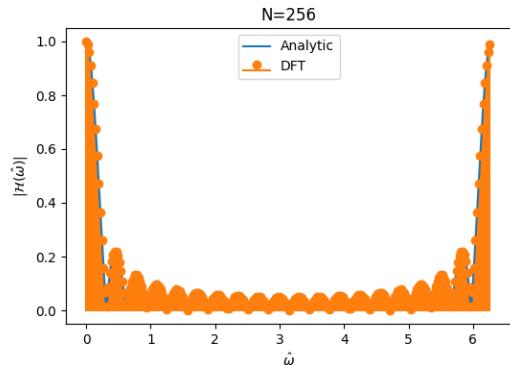


Figure 18.6: An estimate of the discrete-time Fourier transform evaluated using a DFT.

```

def dft_dtft(x, N):
    # The N parameter determines the length of the DFT.
    # If N > len(x), then the signal x is zero padded.
    X = np.fft.fft(x, N)
    # Normalized angular frequency step.
    dom = 2.0*np.pi/N
    # Normalized angular frequencies.
    om_dft = np.arange(N)*dom
    return (X, om_dft)

def he(om, L=20):
    # Analytic DTFT of a causal running average filter.
    return ((1.0/L)*(1 - np.exp(-1j*om*L))/(1 - np.exp(-1j*om)))

# Calculate using analytic formula the DTFT of the running
# average filter.
om = np.linspace(0, 2*np.pi, num=1000)

# DTFT of running average filter by evaluating using analytic
# formula
H = he(om, L=20)

# Signal (FIR filter coefficients).
x = np.repeat(1.0/20.0, 20)

X, om_dft = dft_dtft(x, 256)

# Zero padded signal.
x_zp = np.zeros(256)
x_zp[:20] = x
plt.stem(x_zp)
plt.show()

plt.figure(figsize=(6, 4))
plt.plot(om, np.abs(H), color="C0", label="Analytic")
plt.stem(om_dft, np.abs(X), linefmt="C1-", markerfmt="C1o",
         basefmt="C1", label="DFT")
plt.legend()
plt.title("N=256")
plt.xlabel(r"\hat{\omega}")
plt.ylabel(r"\hat{H}(\hat{\omega})")
plt.savefig("dtft_estimate.png")
plt.show()

```

Listing 18.1: 020_dtft_estimate/dtft_estimate.py

Fast Fourier Transform

In practical applications, one should rarely naively follow Equation 18.48 to implement a DFT operation, because it is not a very efficient way to numerically evaluate this operation for N frequencies. There exists a much faster algorithm to calculate this operation called the Fast Fourier Transform (FFT), which is mathematically equivalent to Equation 18.48. The algorithm is fast, because it subdivides a

long Fourier transform into shorter discrete Fourier transforms in a tree-like fashion.

The DFT as implemented using the mathematical formula (here $\phi_N = e^{-i\frac{2\pi}{N}}$):

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n]\phi_N^{nk}, \quad (18.48)$$

would require $\mathcal{O}(N^2)$ mathematical operations. The FFT algorithm manages to perform mathematically the same operation by using only $\mathcal{O}(N \log_2 N)$ operations².

The FFT algorithm makes use of the fact that you can divide a length N DFT into two length $N/2$ DFTs:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n]\phi_N^{nk} \quad (18.49)$$

$$= \sum_{n=0}^{N/2-1} x[2n]\phi_{N/2}^{nk} + \phi_N \sum_{n=0}^{N/2-1} x[2n+1]\phi_{N/2}^{nk}. \quad (18.50)$$

A simple implementation of the first line takes N^2 operations, as there are N operations are needed for each of the N frequencies. The second line is mathematically equivalent to the first line, but it takes only $N^2/2$ operations, as there are two sums that have $N/2$ unique frequencies and sum over $N/2$ terms ($2N^2/4 = N^2/2$ operations).

This subdivision of signals can be repeated until the length of each DFT is 2. Each time the vectors are split in half, there is a reduction by a factor of two in the number of operations that are needed. A length N signal can thus be split in half, approximately $\log_2 N$ times. Without going into further details, the end result is a $N \log_2 N$ computational complexity.

Because the DFT is such a widely used and important algorithm, much effort has been spent to optimizing it. Every scientific computing environment includes one or more implementations of the FFT algorithm. One of the most popular libraries is FFTW. In Python, the PyFFTW module provides access to this library. There also exist GPU optimized libraries, such as CuFFT, which allows NVidia graphics acceleration cards to be used to calculate FFTs. In Python, one can also use the `numpy.fft` and `scipy.fftpack` modules, although neither of them is as efficient numerically as PyFFTW.

Convolution

A periodic convolution is defined as:

$$a[n] = \sum_{k=0}^{N-1} b[k]c[(n-k) \bmod N]. \quad (18.51)$$

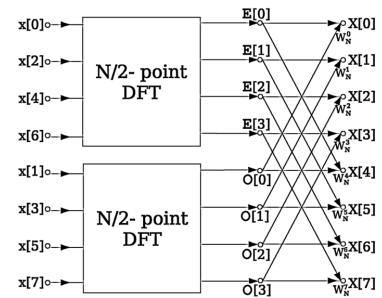


Figure 18.7: The FFT speeds up the calculation by recursively subdividing an N length vector in half until the length of a vector has length 2. Credit: Virens (Wikimedia).

² This is big \mathcal{O} notation from computer science, which is used to describe how the number of computations grows as a function of input size. $\mathcal{O}(N^2)$ means that when the input size grows by a factor of 10, the number of computations required will grow by a factor of 100.



Figure 18.8: The FFTW is included in many programming environments. In many situations, it is the fastest implementation of the FFT algorithm.

In the case of DFT, multiplication in frequency domain is equivalent to periodic convolution (denoted with symbol \circledast):

$$a[n] = b[n] \circledast c[n] \xleftrightarrow{\mathcal{F}_D} \hat{a}[k] = \hat{b}[k]\hat{c}[k]. \quad (18.52)$$

In order to implement a non-periodic convolution of two finite signals of length N and M , zero padding to length $N + M$ is needed. This procedure is widely used to allow FFT to be used to compute a convolution. The Scipy function `scipy.signal.fftconvolve` can be used to convolve two signals using the FFT algorithm. It first calculates $\hat{b}[k]$ and $\hat{c}[k]$, then multiplies them together, and finally performs an inverse Fourier transform to obtain the convolved signals.

Example: Convolution using zero-padded FFT

The Python code in Listing 18.2 provides an example of how to implement a convolution in frequency domain using a zero-padded FFT. Because the FFT algorithm is so efficient numerically, it is often advantageous to implement the convolution operation using FFT. Figure 18.10 shows the output of the Python script.

```
# Use of FFT to implement a linear convolution
# zero-padding used to avoid periodicity.
import matplotlib.pyplot as plt
import numpy as np

# Input signals to be convolved.
a = np.flip(np.arange(10))
b = np.ones(20)

# Zero-padded FFT (10 + 20 = 30)
A = np.fft.fft(a, 30)
B = np.fft.fft(b, 30)
ab = np.fft.ifft(A*B)

# This function also does the same as the three lines
# above ab = s.convolve(a,b)
plt.figure(figsize=(4, 6))
plt.subplot(311)
plt.stem(a)
plt.ylim([0, 12])
plt.xlim([0, 30])
plt.title("$a[n]$")
plt.subplot(312)
plt.stem(b)
plt.ylim([0, 1.2])
plt.xlim([0, 30])
plt.title("$b[n]$")
plt.subplot(313)
plt.title("$a[n]*b[n]$")
plt.stem(ab)
plt.ylim([0, 50])
plt.tight_layout()
plt.savefig("convolution.png")
plt.show()
```

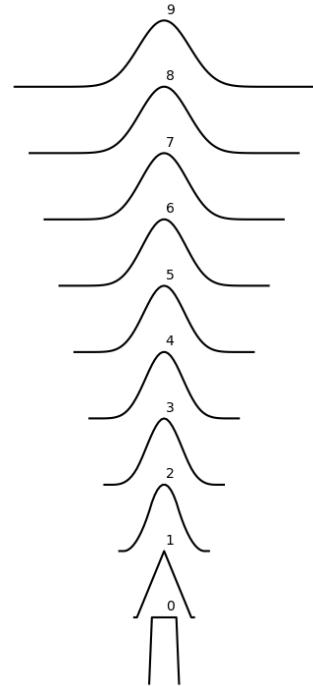


Figure 18.9: Convolving a rectangular signal with itself over and over again will result in a Gaussian. See example: `021_fft_convolution/central_limit.py`.


 Listing 18.2: 021_fft_convolution/fft_convolution.py

Example: 2D FFT

There also exists higher dimensional discrete Fourier transforms. For example, the 2D DFT is defined as:

$$\hat{x}[k, \ell] = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x[n, m] e^{-i \frac{2\pi}{N} nk} e^{-i \frac{2\pi}{N} ml}. \quad (18.53)$$

There exists a 2D FFT algorithm that implements this efficiently, and it is often used in image processing and image compression. There was an example of the use of a 2D FFT for image compression in the Fourier series chapter.

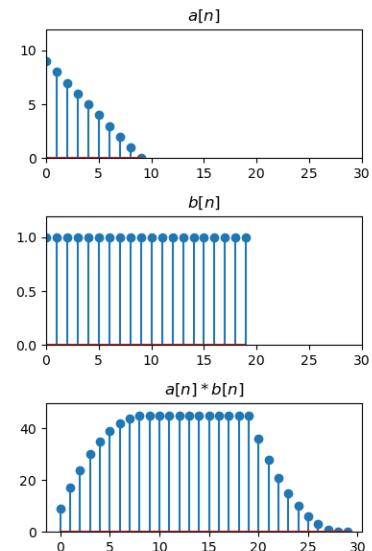


Figure 18.10: An example of a convolution of signals $a[n]$ and $b[n]$ evaluated using an FFT.

Exercises: Discrete Fourier Transform

1. Let $x(t)$ be some continuous-time signal that is sampled. The signal is discretized with a sampling rate of $f_s = 17 \text{ kHz}$, giving a discrete-time signal $x[n]$ of length N . We apply a discrete Fourier transform

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} kn} \quad (18.54)$$

to the discrete-time signal $x[n]$, giving the signal $\hat{x}[k]$. This allows us to represent the signal $x[n]$ as a sum of N unique complex sinusoidal terms (spectral components):

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{x}[k] e^{i \frac{2\pi}{N} kn}. \quad (18.55)$$

Here n and k are integers.

- a) What are the N frequencies of the complex sinusoids that make up signal $x[n]$ in units of radians per sample? What are these in units of hertz?
 - b) When using a DFT for spectral analysis, the value of N influences the spectral resolution, i.e., the spacing of frequencies of the spectral components that are used to represent the signal. What value of N gives a frequency resolution of 0.1 Hz when analyzing the signal $x[n]$ using a DFT?
2. Show that $|\hat{x}[k]| = |\hat{x}[-k]|$ when $x[n] \in \mathbb{R}$. Here $\hat{x}[k]$ is the DFT of signal $x[n]$. Hint: investigate $\hat{x}^*[k]$.
3. The code in Listing 18.3 creates a discrete-time signal consisting of three sinusoidal signals:

$$x[n] = \cos(\hat{\omega}_0 n) + 2 \cos(\hat{\omega}_1 n) + 3 \cos(\hat{\omega}_2 n) \quad (18.56)$$

and calculates the magnitude spectrum $\hat{x}[k]$ of this signal. The spectrum is shown in Figure 18.11.

```
import matplotlib.pyplot as plt
import numpy as np

# Sample indices for 10000 samples.
m = np.arange(10000)

# Sample period.
Ts = 1e-4

# Create a signal consisting of three sinusoids.
x = np.cos(2.0*np.pi*4000.0*Ts*m) + 2*np.cos(2.0*np.pi*1000.0*
    Ts*m) + 3*np.cos(2.0*np.pi*2500.0*Ts*m)
```

Listing 18.3: Create a signal consisting of three sinusoids and analyze the magnitude spectrum calculated using FFT.

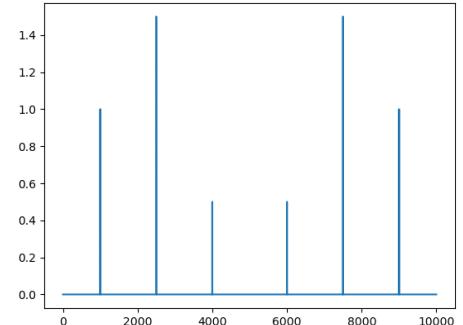


Figure 18.11: The magnitudes of six spectral components of the signal defined in Equation 18.56

- a) What are the numerical values of $\hat{\omega}_0$, $\hat{\omega}_1$, and $\hat{\omega}_2$ in units of radians per sample? Show that they correspond to frequencies $f_0 = 4000$, $f_1 = 1000$, and $f_2 = 2500$ in units of hertz.
- b) Show that signal $x[n]$ is periodic ($x[n] = x[n + N]$). Determine the fundamental period of this signal in samples and in units of seconds.
- c) Modify the code so that the figure produced by the code, shown in Figure 18.11, displays frequency on the horizontal axis in units of hertz. Make sure that you plot both the positive and negative frequency components of the signal, recalling that a real-valued cosine signal consists of a positive and negative frequency complex sinusoidal signal. Verify that you get the correct frequency axis by looking at the relative magnitudes of the six non-zero frequency components of the signal.
Hint: you may find Python functions `numpy.fft.fftshift` and `numpy.fft.freq` useful for this exercise.

Suggested solutions: Discrete Fourier Transform

1. Let $x[n]$ be a discrete-time signal of length N .

- a) The spectral components in the DFT is given as $\hat{\omega}_k = \frac{2\pi k}{N}$ for $k = 0, 1, \dots, N - 1$, in units of radians per sample. In units of hertz one has:

$$f_k = \frac{\omega_k}{2\pi} = \frac{k}{NT_s} = \frac{kf_s}{N},$$

where f_s is the sample rate.

- b) The DFT yields evenly spaced points in frequency domain corresponding to normalized angular frequencies of:

$$\hat{\omega}_k = \frac{2\pi k}{N}, \quad \text{for } k = 0, 1, \dots, N - 1$$

In other words, the resolution is:

$$\Delta\hat{\omega} = \frac{2\pi(k+1)}{N} - \frac{2\pi k}{N} = \frac{2\pi}{N}.$$

In units of hertz, we get:

$$\Delta f = \frac{\Delta\omega_k}{2\pi} = \frac{\Delta\hat{\omega}}{2\pi T_s} = \frac{2\pi}{N} \frac{1}{2\pi T_s} = \frac{1}{NT_s} = \frac{f_s}{N}.$$

Having used that $\hat{\omega} = \omega_k T_s$. If we want to have a resolution of $\Delta f = 0.1$ Hz, then we must choose N , so that:

$$N = \frac{f_s}{\Delta f} = \frac{17 \cdot 10^3 \text{ Hz}}{0.1 \text{ Hz}} = \underline{\underline{170\,000}}.$$

2. Suppose $x[n] \in \mathbb{R}$ for all $n \in \mathbb{Z}$. By definition, the DFT of $x[n]$ is:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}.$$

If we take the complex conjugate (recall that the complex conjugate is distributive over addition and multiplication), we get:

$$\hat{x}^*[k] = \sum_{n=0}^{N-1} x^*[n] e^{i\frac{2\pi}{N}kn}.$$

By assumption $x[n]$ is real, so $x^*[n] = x[n]$, hence:

$$\hat{x}^*[k] = \sum_{n=0}^{N-1} x[n] e^{i\frac{2\pi}{N}kn}.$$

Consider $\hat{x}^*[-k]$, which is:

$$\hat{x}^*[-k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}.$$

Therefore, $\hat{x}[k] = \hat{x}^*[-k]$, so $|\hat{x}[k]| = |\hat{x}^*[-k]|$, as desired.

3. Let $x[n]$ be a discrete-time signal of the form:

$$x[n] = \cos(\hat{\omega}_0 n) + 2 \cos(\hat{\omega}_1 n) + 3 \cos(\hat{\omega}_2 n).$$

- a) Have that $\hat{\omega} = \omega T_s$, where ω is in units of radians per second and $\hat{\omega}$ is in units of radians per sample. Furthermore, have that $\omega = 2\pi f$, where f is in units of hertz, so that $\hat{\omega} = 2\pi f T_s$. In other words:

$$\begin{aligned}f_0 &= \frac{\hat{\omega}_0}{2\pi T_s} = \frac{2\pi(4000)T_s}{2\pi T_s} = \underline{\underline{4000 \text{ Hz}}}, \\f_1 &= \frac{\hat{\omega}_1}{2\pi T_s} = \frac{2\pi(1000)T_s}{2\pi T_s} = \underline{\underline{1000 \text{ Hz}}}, \\f_2 &= \frac{T_s}{2\pi T_s} = \frac{2\pi(2500)T_s}{2\pi T_s} = \underline{\underline{2500 \text{ Hz}}},\end{aligned}$$

where the values obtained for the normalized frequency is read from the Python code.

- b) Let $x[n]$ be as above, then the signal is periodic if: for all $\hat{\omega}_i, \hat{\omega}_j$ one has $\hat{\omega}_i/\hat{\omega}_j \in \mathbb{Q}$. This is the case here, as:

$$\begin{aligned}\hat{\omega}_0/\hat{\omega}_1 &= 4, \\ \hat{\omega}_1/\hat{\omega}_2 &= 2/5, \\ \hat{\omega}_2/\hat{\omega}_0 &= 8/5,\end{aligned}$$

for which all are rational numbers, hence $x[n]$ is periodic.

We've only checked some choices of indices, as the reciprocal of a rational number is also a rational number.

To compute the period of the signal, use the fact that the fundamental period is related to the greatest common divisor (GCD) of the angular frequencies. To compute the GCD of three terms, one can use the fact that: $\hat{\omega} = \text{gcd}(\hat{\omega}_0, \hat{\omega}_1, \hat{\omega}_2) = \text{gcd}(\hat{\omega}_0, \text{gcd}(\hat{\omega}_1, \hat{\omega}_2))$. You can compute the gcd in Python if you ignore the $2\pi T_s$ part, using the `math` module with the function `gcd`. Doing this gives $\hat{\omega} = 2\pi 500 T_s$ in units of radians per sample. To obtain the period in units of samples:

$$N = \frac{2\pi}{\hat{\omega}} = \frac{2\pi}{2\pi 500 T_s} = 20 \text{ samples.}$$

The period in units of seconds, is then:

$$T = \frac{2\pi}{\omega} = \frac{2\pi T_s}{\hat{\omega}} = \frac{2\pi T_s}{2\pi 500 T_s} = \frac{1}{500} = 0.002 \text{ s,}$$

as $\hat{\omega} = \omega T_s$, so the signal is periodic with a period of 20 samples, or 0.0002 seconds.

c) Listing 18.4 shows a way to obtain units of hertz on the x-axis.

The output of Listing 18.4 is shown in Figure 18.12. The original signal was a cosine with frequencies of 4000 Hz, 2500 Hz and 1000 Hz, which is present in the plot, as it should be. The cosine with frequency of 2500 Hz has the largest amplitude, followed by the 1000 Hz signal, which is reflected in the plot too.

```
import matplotlib.pyplot as plt
import numpy as np

# Sample indices for 10000 samples.
m = np.arange(10000)

# Sample period.
Ts = 1e-4

# Create a signal consisting of three sinusoids.
x = np.cos(2*np.pi*4000.0*Ts*m) + 2*np.cos(2*np.pi*1000.0*
    Ts*m) + 3*np.cos(2*np.pi*2500.0*Ts*m)

# Call fftfreq to compute the frequencies in units of hertz
# The function takes in the first argument, which is the
# length of the window and second argument for the
# stepsize (Ts),
# use fftshift to shift the frequencies to have zero in the
# middle; this includes both positive and negative
# frequencies.
freq = np.fft.fftshift(np.fft.fftfreq(len(m), d=Ts))

# Plot the magnitude of sinusoids using fft.
plt.plot(freq, np.abs(np.fft.fft(x))/len(m))
plt.xlabel("Frequency (Hz)")
# Call this if needed.
# plt.show()
```

Listing 18.4: Plot with units of hertz on the x-axis

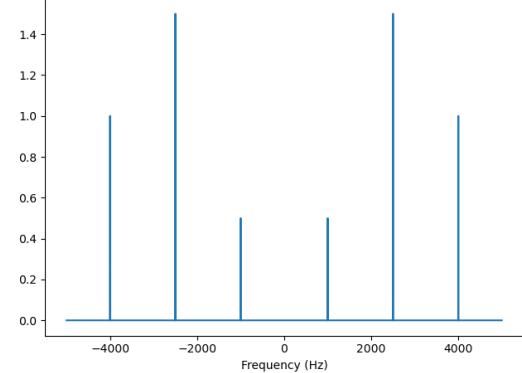


Figure 18.12: The magnitudes of six spectral components, but units of hertz on the x-axis

19

Spectral Analysis

The DFT can be used for spectral analysis of signals. One can think of the DFT as a filter bank of frequency selective filters. This concept allows us to use the FFT algorithm to efficiently analyze the spectral contents of a discrete-time signal $x[n]$.

Discrete Fourier transform as a filter bank

The impulse response of an ideal point-frequency filter that filters only some frequency $\hat{\omega}_0$ was derived in the discrete-time Fourier transform chapter. Dropping the constant $(2\pi)^{-1}$, the impulse response was:

$$h[n] = e^{i\hat{\omega}_0 n}. \quad (19.1)$$

If we apply an N sample long rectangular window $w_R[n]$ of the following form:

$$w_R[n] = \begin{cases} 1 & \text{when } -(N-1) \leq n \leq 0 \\ 0 & \text{otherwise} \end{cases}, \quad (19.2)$$

we have a windowed filter impulse response¹:

$$h_R[n] = w_R[n]h[n] = \begin{cases} e^{i\hat{\omega}_0 n} & \text{when } -(N-1) \leq n \leq 0 \\ 0 & \text{otherwise} \end{cases}. \quad (19.3)$$

To apply this filter to a signal, we convolve $h_R[n]$ with the signal $x[n]$ and inspect the output of this filter at sample $n = 0$. Recall that a convolution is:

$$y[n] = \sum_{\ell=-\infty}^{\infty} x[\ell]h_R[n-\ell] \quad (19.4)$$

and at $n = 0$ this would be:

$$y[0] = \sum_{\ell=-\infty}^{\infty} x[\ell]h_R[0-\ell] \quad (19.5)$$

$$= \sum_{\ell=0}^{N-1} x[\ell]e^{-i\hat{\omega}_0 \ell} \quad (19.6)$$

¹ Note that this is a truncated point-frequency filter.

We'll now say that the center frequency of the filter is $\hat{\omega}_0 = 2\pi k/N$. The output of the filter $y[n]$ at $n = 0$ can be written as:

$$y[0] = \sum_{\ell=0}^{N-1} x[\ell] e^{-i\frac{2\pi}{N}k\ell}. \quad (19.7)$$

Now recall that the DFT is defined as:

$$\hat{x}[k] = \sum_{\ell=0}^{N-1} x[\ell] e^{-i\frac{2\pi}{N}k\ell} \quad (19.8)$$

They are the same! This means that a DFT evaluates the output of frequency selective filters with frequencies:

$$\hat{\omega}_k = \frac{2\pi}{N}k.$$

(19.9)

This is at first counter-intuitive. The output of the DFT should be a frequency domain quantity, and so it is. However, it also happens to be the time domain filter output at time 0 for a frequency selective filter with a rectangular window:

$$h_{R,k}[n] = \begin{cases} e^{i\frac{2\pi}{N}kn} & -(N-1) \leq n \leq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (19.10)$$

Frequency response

We can investigate the frequency response $\mathcal{H}_k(\hat{\omega})$ of each filter $h_{R,k}[n]$ in the filter bank implemented with a discrete-time Fourier transform:

$$\mathcal{H}_k(\hat{\omega}) = \sum_{\ell=-\infty}^{\infty} h_{R,k}[\ell] e^{-i\hat{\omega}\ell} \quad (19.11)$$

$$\sum_{\ell=-(N-1)}^0 e^{i(\hat{\omega}_k - \hat{\omega})\ell} \quad (19.12)$$

$$= \sum_{\ell=0}^{N-1} e^{-i(\hat{\omega}_k - \hat{\omega})\ell} \quad (19.13)$$

$$= \frac{1 - e^{-i(\hat{\omega}_k - \hat{\omega})N}}{1 - e^{-i(\hat{\omega}_k - \hat{\omega})}} \quad (19.14)$$

$$= Ne^{-i(\hat{\omega}_k - \hat{\omega})(N-1)/2} D_N(\hat{\omega}_k - \hat{\omega}) \quad (19.15)$$

In the second last step, I've used the geometric sum closed solution formula $S = (1 - \alpha^{N+1})/(1 - \alpha)$ with $\alpha = e^{-i(\hat{\omega}_k - \hat{\omega})}$. The function $D_N(\hat{\omega})$ is the familiar Dirichlet function we encountered when investigating the frequency response of the running average filter. In this case, it is frequency shifted so that the peak is at $\hat{\omega} = \hat{\omega}_k$.

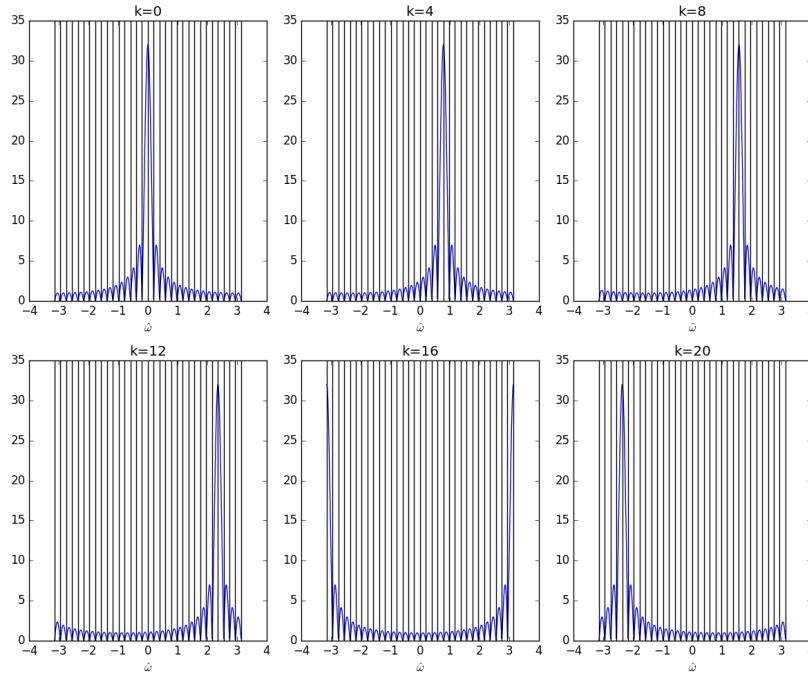


Figure 19.1: The magnitude response $|\mathcal{H}_k(\hat{\omega})|$ for several values of k . Each DFT output bin $\hat{x}[k]$ can be thought of as the output of a frequency selective filter with this magnitude response. For example, $k = 0$ corresponds to a low-pass filter that passes mainly spectral components of the signal $x[n]$ that have a frequency close to zero.

The magnitude response $|\mathcal{H}_k(\hat{\omega})|$ for $N = 32$ using several values of k are shown in Figure 19.1.

From this plot, it is obvious that complex sinusoidal signals at nearly all frequencies, except for frequencies $\hat{\omega}_k$, will leak into the filter output with fairly significant amplitudes. This is often unwanted.

Windowed DFT

In order to reduce the spectral sidelobes seen in Figure 19.1, a tapered window function can be used. The procedure is relatively simple. One applies a more selective window function to the signal before evaluating the DFT. This is equivalent to the tapered frequency selective filter shown in the chapter on practical filters. The impulse response of each filter in the filter bank will now be:

$$h_{R,k}[n] = \begin{cases} w(N-1+n)e^{j\frac{2\pi}{N}kn} & -(N-1) \leq n \leq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (19.16)$$

Here $w(n)$ is a tapered window function of length N .

The filter bank can now be implemented using a DFT as follows:

$$\hat{x}_w[k] = \sum_{n=0}^{N-1} x[n]w_N(n)e^{-i\frac{2\pi}{N}nk}.$$

(19.17)

This results in a wider central peak at each frequency, but less spectral leakage from signals outside frequency $\hat{\omega}_k$.

Figure 19.2 shows the magnitude response for several DFT frequency components below, when a Hann window (Equation 16.44) of length $N = 32$ is applied to the signal:

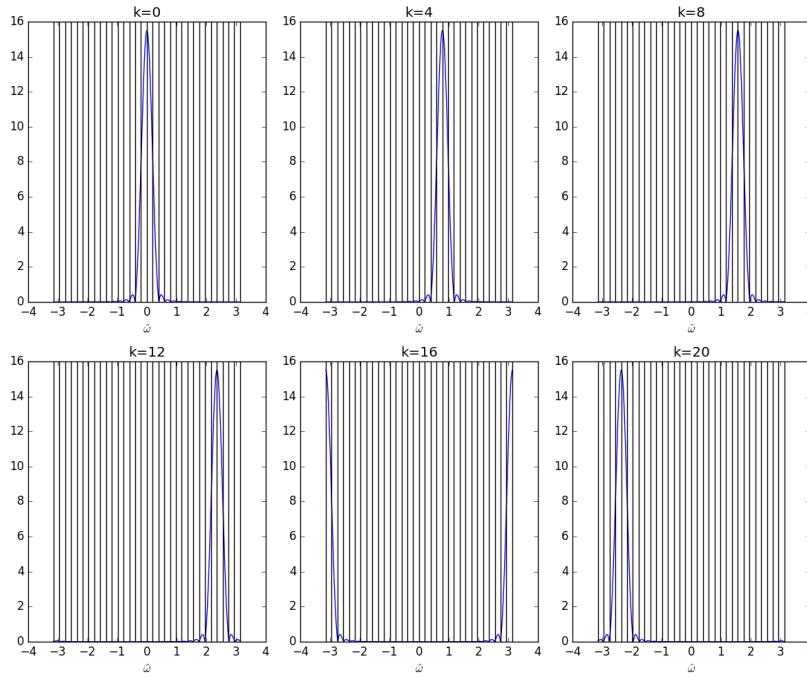


Figure 19.2: Windowed DFT filter bank $|\mathcal{H}_k(\hat{\omega})|$ magnitude responses for several values of k . Compared to rectangular window function, the magnitude response of the Hann windowed DFT filter bank has significantly lowered sidelobes. There is less out-of-band signal leaking into the pass band of each frequency.

In nearly all spectral analysis applications, it is beneficial to apply a window function to the signal that is being analyzed for spectral content, to avoid spectral leakage.

Example: Spectral analysis using windowed DFT

The following example illustrates the advantages of using a window function with low spectral sidelobes when using an FFT for spectral analysis.

Let's generate a synthetic signal, which consists of a large and a small amplitude sinusoidal signal.

$$y[n] = A_1 \sin(\hat{\omega}_1 n) + A_2 \sin(\hat{\omega}_2 n), \quad (19.18)$$

where $A_1 \gg A_2$ and $\hat{\omega}_1 \neq \hat{\omega}_2$. This signal is shown in Figure 19.3. When looking at the plot of the signal $y[n]$ in time domain, it is very difficult, if not impossible, to identify the weak signal. The strong signal overpowers it.

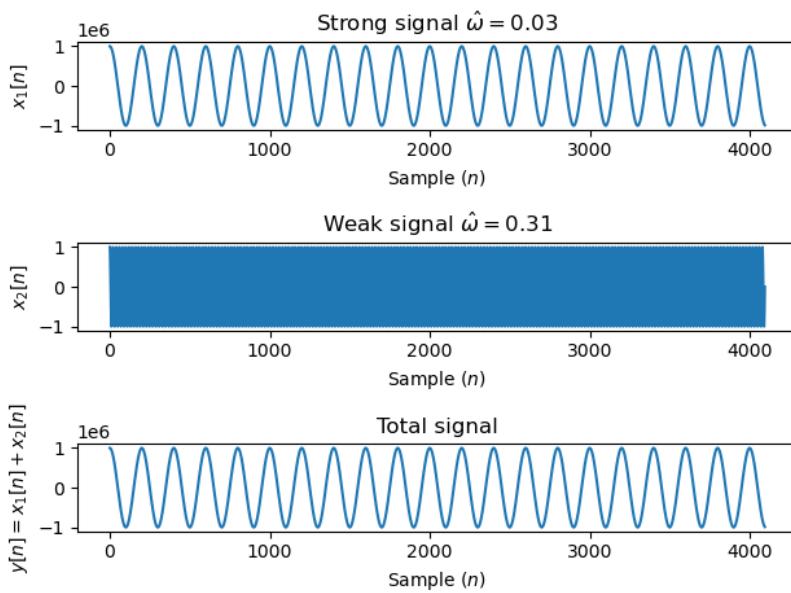


Figure 19.3: Top: a large amplitude sinusoidal signal, Middle: a small amplitude sinusoidal signal. Bottom: Sum signal $y[n]$. When added together, the large amplitude signal dominates and the small amplitude signal cannot be visually detected.

When analyzing the signal for spectral content using an unwindowed DFT, we get:

$$\hat{y}[k] = \sum_{n=0}^{N-1} y[n] e^{-i\frac{2\pi}{N}nk}. \quad (19.19)$$

The strong sinusoidal signal $A_1 \sin(\hat{\omega}_1 n)$ will result in a non-zero output for all spectral components $\hat{y}[k]$. This “spectral” leakage can easily overpower the weaker spectral component. Power in decibel scale ($10 \log_{10} |\hat{y}[k]|^2$) is shown with a blue line in Figure 19.4. It is easy to identify one strong spectral peak corresponding to the large amplitude sinusoidal signal with frequency $\hat{\omega}_1$.

In order to detect the sinusoidal signal with the small amplitude, we can use a windowed DFT:

$$\hat{y}_w[k] = \sum_{n=0}^{N-1} y[n] w(n) e^{-i\frac{2\pi}{N}nk}. \quad (19.20)$$

Power in decibel scale is shown with an orange line in Figure 19.4. In this case, spectral leakage is reduced significantly, and it is possible to identify the weak spectral component.

The Python program used to create the plots for this example is shown in Listing 19.1

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal.windows import hann
```

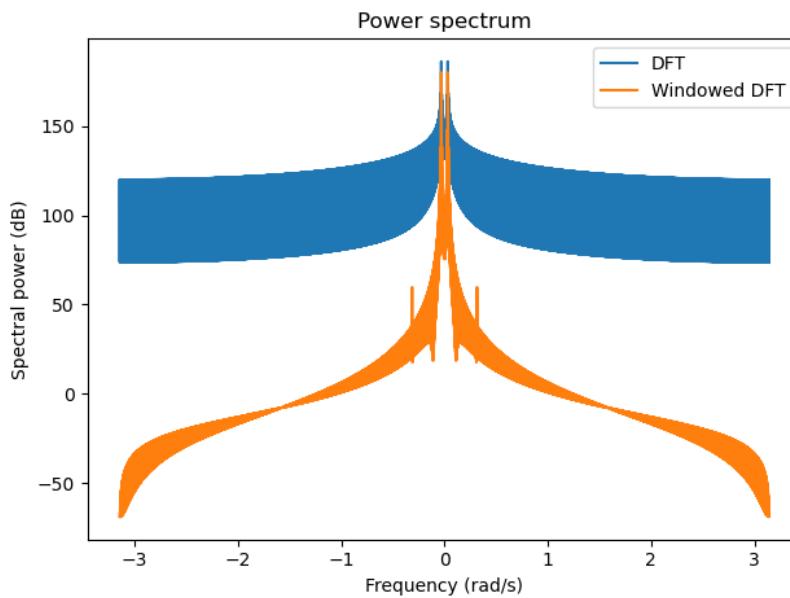


Figure 19.4: Spectral power with (orange) and without (blue) a tapered window function. Without a tapering window, it is impossible to identify the sinusoidal signal with a very small amplitude, as spectral leakage from the large amplitude sinusoidal signal overpowers the weak signal.

```
# Using windowed DFT for spectral analysis of signals.
# Demonstrate that a window function allows weak
# signals to be found better, due to less spectral
# leakage than if a rectangular window is used.

N = 4096
n = np.arange(N)

freq1 = 0.01*np.pi
freq2 = 0.1*np.pi

# Strong low-frequency signal signal.
strong_signal = 1e6*np.cos(freq1*n)

# Weak signal at higher frequency.
weak_signal = np.cos(freq2*n)

y = strong_signal + weak_signal

plt.subplot(311)
plt.plot(n, strong_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_1[n]$")
plt.title(r"Strong signal $\hat{\omega}=%1.2f\pi$ (% freq1))")
plt.subplot(312)
plt.plot(n, weak_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_2[n]$")
plt.title(r"Weak signal $\hat{\omega}=%1.2f\pi$ (% freq2))")
plt.subplot(313)
# Weak signal is impossible to see,
# because it has a 1e6 smaller amplitude.
```

```

plt.plot(n, y)
plt.xlabel("Sample ($n$)")
plt.ylabel("$y[n]=x_1[n]+x_2[n]$")
plt.title("Total signal")
plt.tight_layout()
plt.savefig("windowed_signals.png")
plt.show()

# Analyze spectrum zero padded DFT to be of length 2*N.
Y = np.fft.fft(y, 2*N)
# Windowed zero-padded DFT.
w = hann(N)
# Zero padded DFT to be of length 2*N.
WY = np.fft.fft(w*y, 2*N)
# Frequencies in radians per sample.
om = np.fft.fftfreq(len(Y), d=1)*2.0*np.pi
# Reorder frequencies so that we go from -pi to pi instead of 0
# to 2\pi.
om = np.fft.fftshift(om)
Y = np.fft.fftshift(Y)
WY = np.fft.fftshift(WY)

plt.plot(om, 10.0*np.log10(np.abs(Y)**2.0), label="DFT")
plt.plot(om, 10.0*np.log10(np.abs(WY)**2.0), label="Windowed DFT")
plt.ylabel("Spectral power (dB)")
plt.xlabel("Frequency (rad/s)")
plt.title("Power spectrum")
plt.legend()
plt.tight_layout()
plt.savefig("windowed_spec.png")
plt.show()

```

Listing 19.1: 022_window_functions/windowed_dft.py

Example: Dynamic spectrum

Signals are rarely unchanging in their spectral content. In many applications, it is advantageous to analyze the time-frequency evolution of signals. This can be done with the help of a windowed DFT and selecting portions of the signal $x[n]$ of length N . This results in a 2d signal, containing time and frequency:

$$\hat{x}[t, k] = \sum_{n=0}^{M-1} x[n + t\Delta n] w_N(n) e^{-i\frac{2\pi}{M} kn}. \quad (19.21)$$

Here Δn is the number of samples between spectral estimates. This type of time-frequency spectrum is sometimes called a *dynamic spectrum* or a *spectrogram*²

Note that in this case, N samples in time are used for each spectrum, so the spectral estimate is not instantaneous, but an average over N samples. This is yet again a consequence of time-frequency ambiguity.

² There is an implementation of a spectrogram in `scipy.signal.spectrogram`.

Zero padding of the DFT to length M can also be used to produce a higher resolution DTFT estimate. In this case, M denotes the length of the DFT, and $w_N(n)$ indicates a window function of length N , which will zero-pad samples of $x[n]$ at values of n beyond sample $N - 1$.

An example of a dynamic spectrum is shown in Figure 19.5. The signal that is analyzed in time and frequency is a chirp signal:

$$x[n] = \sin(an^2), \quad (19.22)$$

which has a frequency that increases as a function of time.

The figure is produced with the Python program shown in Listing 19.2 shown below, which implements a dynamic spectrum using the FFT function in Python.

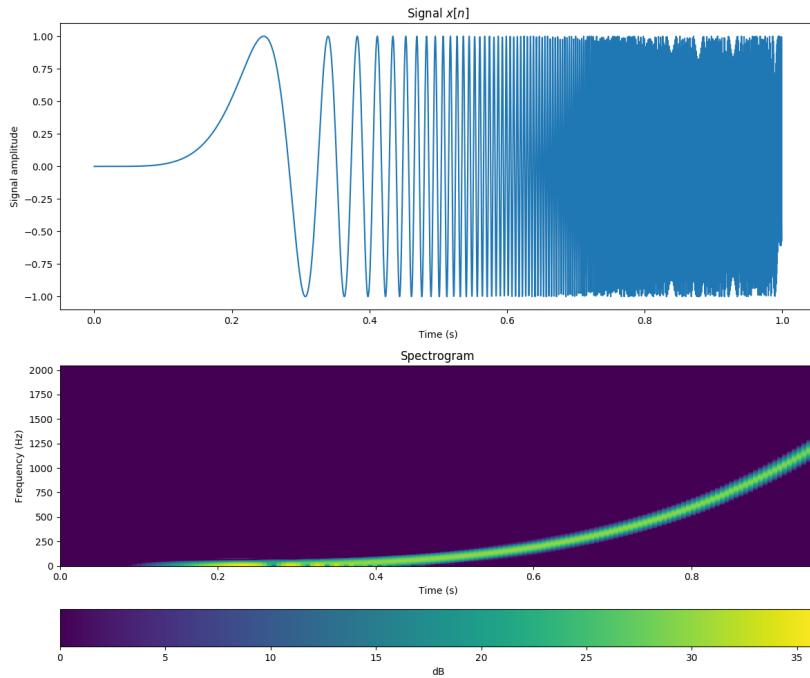


Figure 19.5: Time-frequency power spectrum $10 \log_{10} |X[t, k]|^2$ for a chirp signal, which increases in frequency.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal.windows import hann

def spectrogram(x, M=1024, N=128, delta_n=100):
    """
    x = signal
    M = FFT length
    N = window function length
    delta_n = step step
    """
    # ... (rest of the code is omitted)
```

```

max_t = int(np.floor((len(x)-N)/delta_n))
X = np.zeros([max_t, M], dtype=np.complex64)
w = hann(N)
xin = np.zeros(N)

for i in range(max_t):
    # Zero padded windowed FFT.
    xin[0:N] = x[i*delta_n + np.arange(N)]
    X[i, :] = np.fft.fft(w*xin, M)

return (X)

# Sample rate (Hz).
fs = 4096.0

# Sample indices (one second of signal).
n = np.arange(4096)
# Generate a chirp signal.
x = np.sin(0.15e-14*n**5.0)

# Time step.
delta_n = 25
M = 2048
# Create dynamic spectrum.
S = spectrogram(x, M=M, N=128, delta_n=delta_n)

freqs = np.fft.freq(2048, d=1.0/fs)
time = delta_n*np.arange(S.shape[0])/fs

plt.figure(figsize=(12, 10))
plt.subplot(211)
plt.plot(n/fs, x)
plt.title("Signal $x[n]$")
plt.xlabel("Time (s)")
plt.ylabel("Signal amplitude")
plt.subplot(212)

plt.title("Spectrogram")
plt.pcolormesh(time, freqs[:int(M/2)], np.transpose(10.0*np.
    log10(np.abs(S[:, :int(M/2)])**2.0)), vmin=0)
plt.xlim([0, np.max(time)])
plt.ylim([0, fs/2.0])
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")
cb = plt.colorbar(orientation="horizontal")
cb.set_label("dB")
plt.tight_layout()
plt.savefig("dyncspec.png")
plt.show()

```

Listing 19.2: 023_dynamic_spectrum/dynamic_spectrum.py

Exercises: Spectral Analysis

1. This task is a programming assignment. Your task will be to analyze the time-frequency contents of a musical recording. The aim is to determine what sequence of musical notes are being played.

Download the audio file from:

<https://bit.ly/3EAhn26>

You can listen to the audio file to get an idea of the time-frequency contents of the signal.

Read the audio signal using Python as follows:

```
import scipy.io.wavfile
audio = scipy.io.wavfile.read("b.wav")
sample_rate = audio[0]
# read only one channel of the stereo signal
signal = audio[1][:,0]
```

Complete the following tasks and justify your result:

- a) What is the sample-rate? What is the lowest and highest frequency spectral component that can be represented using this sample-rate?
- b) Make a plot of the signal as a function of time. Use time in seconds on the horizontal axis. How long is the signal?
- c) Write a short program to analyze the time-frequency contents of this signal. There are several Python library functions for implementing a spectrogram. For example: `scipy.signal.spectrogram` or `matplotlib.pyplot.specgram`. For this exercise, I ask that you do not use these functions, but implement your own function instead, so that you can gain a good understanding of how these functions are implemented. You can use the example program in Listing 19.2 to help you out. Use a Hann window in the spectral analysis.
- d) Make a plot of the spectrogram produced with your program. Limit the frequency axis to range 0 to 1500 Hz. Use an appropriate time and frequency resolution that allows you to identify the positions of individual notes being played in time and frequency.
- e) In the time-frequency spectrum of the musical instrument playing an individual note, there are spectral lines with multiple harmonics $f = nf_0$, where $n \in \mathbb{N}$ is a positive integer, and f_0 is



Figure 19.6: Ludwig van Beethoven, a well known musical composer active around the turn of the 18th and 19th century.

the base frequency. For example, if $f_0 = 220$ Hz, you will also see spectral lines at 440, 660, 880, and 1100 Hz. Why are there multiple spectral lines?

- f) Using the spectrogram plot, determine what are the nine musical notes that are played in the recording? Make sure that your plot allows you to clearly identify each note in time and frequency. Use 150 to 400 Hz on the frequency axis as the axis limits. You will most likely need to adjust the FFT length, overlap, and zero padding settings to be able to easily determine the notes. You might also need to adjust the color scale. I'll give you a hint. The first note is E.

Assume an equal tempered scale is used, which results in the following mapping of base frequencies and musical notes:

Note	Frequency (Hz)
A	220.0
A#	233.0819
B	246.9417
C	261.6256
C#	277.1826
D	293.6648
D#	311.127
E	329.6276
F	349.2282
F#	369.9944
G	391.9954

Suggested solution: Spectral Analysis

1. Read the audio file as follows:

```
import scipy.io.wavfile
audio = scipy.io.wavfile.read("b.wav")
sample_rate = audio[0]
# Read only one channel of the stereo signal.
signal = audio[1][:,0]
```

- a) Using the code given, one can simply print the sample rate.

The sample rate is $f_s = 44100$ Hz, so the highest and lowest frequencies that can be represented are $f_s/2 = \pm 22050$ Hz.

- b) The following code will import the audio file and plot the signal with seconds on the x -axis.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile

audio = scipy.io.wavfile.read("b.wav")
sample_rate = audio[0]

# Read only one channel of the stereo signal.
signal = audio[1][:,0]

# The sample rate is:
print(sample_rate) # output is 44100 Hz.

# Partition the interval such that the
# units become seconds.
t = np.arange(len(signal))/(sample_rate)

plt.plot(t, signal)
plt.xlabel("Time (s)")
# Call this if needed.
```

Listing 19.3: Code to plot audio signal

The output of Listing 19.3 is shown in Figure 19.7.

The length in seconds can be computed as time in seconds = (number of samples)(sample spacing) which in this case is $t = 215678/44100 \approx 4.89$, so around a 5 seconds long signal. From Figure 19.7 we see that the signal is around 5 seconds long.

- c) The following code shown in Listing 19.4 will compute and print the spectrogram for the audio signal.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.io.wavfile
from scipy.signal.windows import hann
```

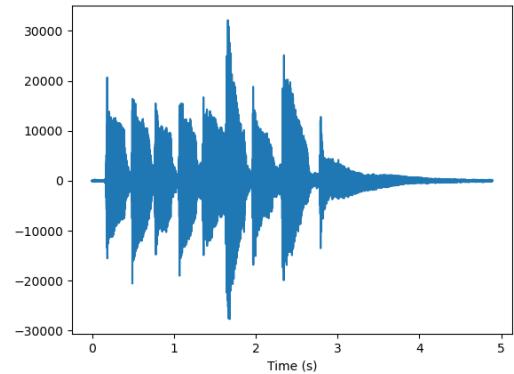


Figure 19.7: Audio signal

```

def convert_to_decibel(x):
    """Function to convert to dB."""
    return 10*np.log10(np.abs(x)**2)

audio = scipy.io.wavfile.read("b.wav")
sample_rate = audio[0]

# Read only one channel of the stereo signal.
signal = audio[1][:, 0]

def spectrogram(signal, delta_n, N, M):
    """Have a signal x of length L.
    We divide the signal into sub arrays of length N, the
    step size is then delta_n
    the maximum time units are then L/delta_n

    M - length of FFT.
    N - length of window.
    delta_n - step size in time.
    """

    # Window function (Hann window).
    w = hann(N)

    # Length of signal.
    L = len(signal)

    # Compute the maximum number of time steps.
    t_max = (L - N) // delta_n

    # Allocate space for the spectrogram and sub_arrays.
    H = np.zeros([t_max, M], dtype=np.complex64)
    sub_array = np.zeros(N)

    # Step through the signal.
    for i in range(t_max):
        # Get a sub_array and then fft it with the window
        # and store it in H.
        sub_array[:N] = signal[i*delta_n + np.arange(N)]
        H[i, :] = np.fft.fft(sub_array*w, M)

    return H

M = 10480
N = 2000
delta_n = 40

# Compute the spectrogram.
spect = spectrogram(signal, delta_n, N, M)

# Partition the axes correctly with
# units of Hertz and seconds.
freqs = np.fft.fftfreq(M, d=1.0/sample_rate)
time = delta_n*np.arange(spect.shape[0])/sample_rate

# Create the spectrogram plot, limiting frequency to (0,
# 1500) Hz.
plt.pcolormesh(time, freqs[:M//2], np.transpose(

```

```

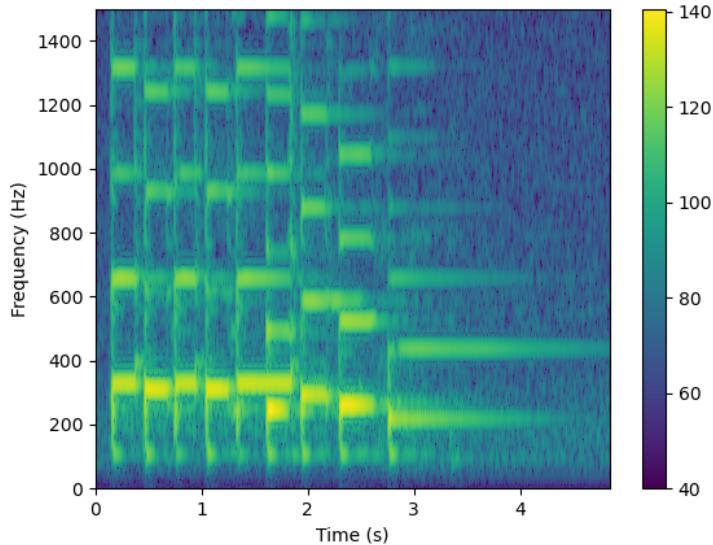
    convert_to_decibel(spect[:, :M//2]), vmin=40)
plt.xlabel("Time (s)")
plt.ylabel("Frequency (Hz)")
plt.ylim(0, 1500)
plt.colorbar()
# Call this if needed.

```

Listing 19.4: Spectrogram code

- d) The output of Listing 19.4 is shown in Figure 19.8.

Figure 19.8: Für Elise spectrogram



- e) Instruments don't produce pure frequency tones, so there will be harmonics, which can be seen in Figure 19.8. Different instruments emphasize different harmonics, giving each instrument its respective sound among other factors.
- f) Comparing the spectrogram with the frequency table, one can read that the musical phrase is E D# E D# E B D C A. You can compare this with sheet music for the piece, which is the correct phrase.

Arbitrary Frequency Response Filters

The discrete Fourier transform and the FFT algorithm allow us to efficiently implement filters with an arbitrary frequency response.

This is because the frequency domain representation of the filter impulse response $h[n]$ can be specified as in frequency domain $\hat{h}[k]$.

Convolution (i.e., filtering) is multiplication in frequency domain:

$$y[n] = h[n] \circledast x[n] \xleftrightarrow{\mathcal{F}_D} \hat{y}[k] = \hat{h}[k] \hat{x}[k]. \quad (20.1)$$

This means that we don't even have to determine the filter impulse response in time domain $h[n]$. One can FFT the signal $x[n]$, specify and apply the filter $\hat{h}[k]$ directly in frequency domain through multiplication, and finally inverse FFT the signal to obtain a filtered signal $y[n]$.

Example: Filtering out spectrally narrow frequency components

In this example, we will create a filter that removes strong spectral components from a signal using a band-stop filter.

Consider the following signal:

$$x_1[n] = A_1 \cos(\omega_1 n + \phi_1) + A_2 \cos(\omega_2 n + \phi_2) \quad (20.2)$$

$$+ A_3 \cos(\omega_3 n + \phi_3), \quad (20.3)$$

and another signal

$$x_2[n] = a_1 \delta[n - n_1] + a_2 \delta[n - n_2] + a_3 \delta[n - n_3], \quad (20.4)$$

overlaid on top of it:

$$x[n] = x_1[n] + x_2[n]. \quad (20.5)$$

We'll assume that signal $x_1[n]$ is the problematic signal that we want to filter out, so that we are able to detect the signal $x_2[n]$. We'll assume that $|A_i| \gg |a_j|$ for all i, j . This means that the signal $x_1[n]$

greatly overpowers the signal $x_2[n]$ in amplitude. Figure 20.1 shows $x_1[n]$, $x_2[n]$, and $x[n] = x_1[n] + x_2[n]$ for this example. It is virtually impossible to visually detect the weak signal $x_2[n]$ from the plot.

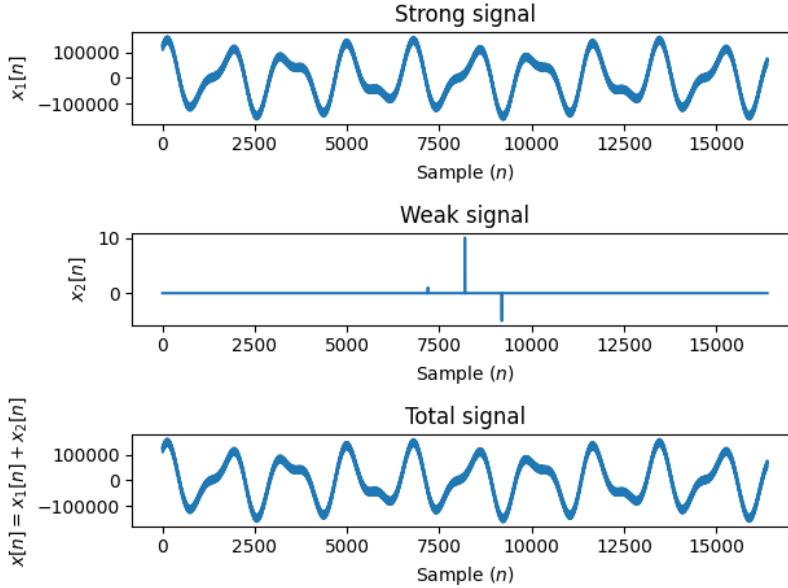


Figure 20.1: A plot of the strong narrowband signal $x_1[n]$, the weak wideband signal $x_2[n]$, and the summed signal $x_1[n] + x_2[n]$. It is difficult to observe the weak signal $x_2[n]$ just by visually inspecting $x[n]$.

In order to detect signal $x_2[n]$, we need to filter out signal $x_1[n]$. We can do this by creating a filter, which will remove the strong spectral components, allowing the weak broad band signal to be recovered. Forming such a filter in frequency domain is relatively easy. We specify that:

$$\hat{h}[k] = \begin{cases} \frac{\alpha}{|\hat{x}[k]|} & \text{for strong spectral components } k \\ 1 & \text{otherwise} \end{cases} \quad (20.6)$$

The strong spectral components can be identified by inspecting the spectrum of the signal $x[n]$, that is $|\hat{x}[k]|$. When this filter is applied in frequency domain, we obtain a frequency domain representation of the filtered signal:

$$\hat{y}[k] = \hat{h}[k]\hat{x}[k]. \quad (20.7)$$

The strong spectral components will be attenuated to a constant magnitude α , while the rest of the spectral components are unaffected. Figure 20.2 shows $\hat{y}[k]$ and $\hat{x}[k]$. Note that the spectrally narrow signals have approximately 10^{12} more power than the weak signals.

In order to obtain the filtered signal, we simply inverse DFT:

$$y[n] = \mathcal{F}_D^{-1}\{\hat{y}[k]\} \quad (20.8)$$

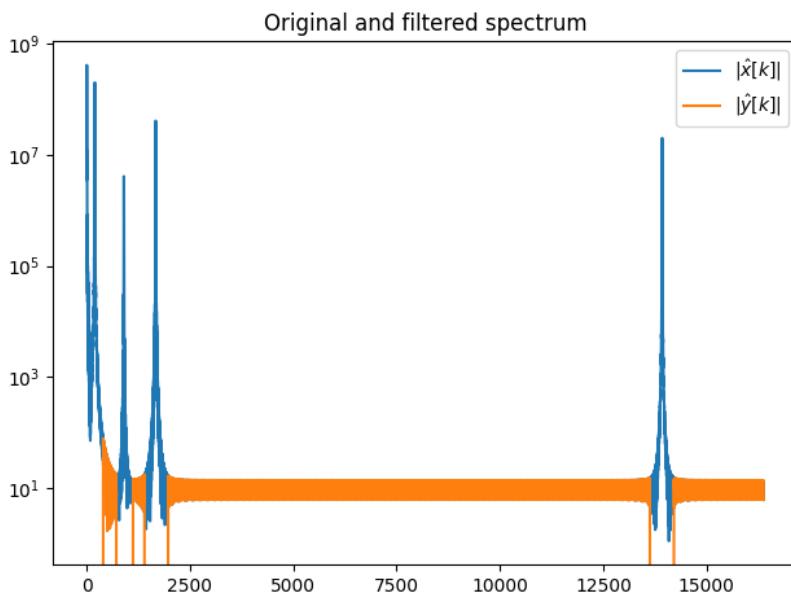


Figure 20.2: The magnitude spectrum of signal $|\hat{x}[k]|$ and the magnitude spectrum of the filtered signal $|\hat{y}[k]|$. In this case, a logarithmic scale is used. A Hann window was applied to the signal when calculating the spectrum to reduce spectral leakage.

The filtered signal is shown in Figure 20.3. The plot shows that when strong spectrally narrow signal components are filtered out, the weak signal becomes visible. There are some artifacts caused by filtering, because some spectral components of the weak signal have also been reduced in amplitude.

In most cases, it is advantageous to use a tapered window on the signal $x[n]$ when estimating the discrete Fourier transform, in order to reduce spectral leakage:

$$\hat{x}[k] = \sum_{n=0}^{N-1} w[n]x[n]e^{-\frac{2\pi}{N}nk} \quad (20.9)$$

Here $w[n]$ would be a tapered window with better suppression of spectral sidelobes than a rectangular window. The Hann window is one good example.

The Python program used to create the plots in this example is shown in Listing 20.1.

```
#  
# Using DFT to create a filter that  
# attenuate spectral components to allow  
# weak signals buried under the strong but  
# spectrally narrow signals to be found.  
#  
import matplotlib.pyplot as plt  
import numpy as np  
from scipy.signal.windows import hann
```

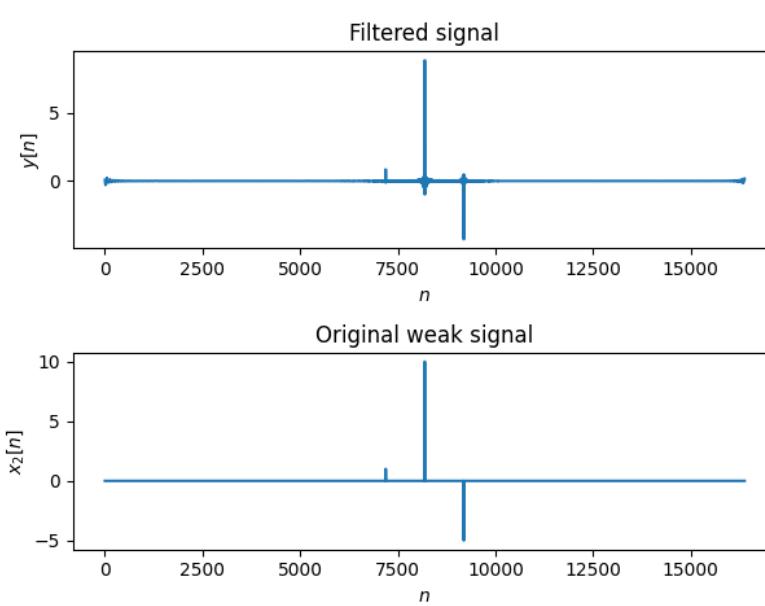


Figure 20.3: A comparison of the filtered estimate $y[n]$ of the weak signal $x_2[n]$ and the original signal.

```

N = 4096*2*2
n = np.arange(N)
freqs = [0.0012, 0.0021, 0.0032, 0.1, 0.9]
A = [1e5, 0.5e5, 1e3, 1e4, 0.5e4]
strong_signal = np.zeros(N)

for i, f in enumerate(freqs):
    strong_signal += A[i]*np.cos(np.pi*f*n + np.random.randn(1))

# Weak signal at the middle of the signal.
weak_signal = np.zeros(N)
weak_signal[int(N/2)] = 10.0
weak_signal[int(N/2)+1000] = -5.0
weak_signal[int(N/2)-1000] = 1.0

x = strong_signal+weak_signal

plt.subplot(311)
plt.plot(n, strong_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_1[n]$")
plt.title("Strong signal")
plt.subplot(312)
plt.plot(n, weak_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_2[n]$")
plt.title("Weak signal")
plt.subplot(313)
plt.plot(n, x)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x[n]=x_1[n]+x_2[n]$")
plt.title("Total signal")
plt.tight_layout()

```

```

plt.savefig("filter_signals.png")
plt.show()

# Windowed zero-padded DFT.
w = hann(N)
WX = np.fft.rfft(w*x, 2*N)

# Create a filter than filters out strong spectral components
# the signal.
WH = np.zeros(len(WX))
WH[:] = 1.0 # Pass all frequencies.
WH[0:400] = 1.0/np.abs(WX[0:400]) # Attenuate strong
# frequencies.
WH[1400:1800] = 1.0/np.abs(WX[1400:1800]) # Attenuate strong
# frequencies.
WH[14000:15000] = 1.0/np.abs(WX[14000:15000]) # Attenuate
# strong frequencies.

plt.semilogy(np.abs(WX), label=r"$|\hat{x}[k]|$")
plt.semilogy(np.abs(WX*WH), label=r"$|\hat{y}[k]|$")
plt.legend()
plt.title("Original and filtered spectrum")
plt.tight_layout()
plt.savefig("filter_spec.png")
plt.show()

# Apply filter in frequency domain.
wy = np.fft.irfft(WH*WX)[0:N]

# Plot filtered signal and compare with original weak
# signal buried under stronger signals
plt.subplot(211)
plt.plot(n, wy)
plt.title("Filtered signal")
plt.xlabel("$n$")
plt.ylabel("$y[n]$")

plt.subplot(212)
plt.plot(n, weak_signal)
plt.title("Original weak signal")
plt.xlabel("$n$")
plt.ylabel("$x_2[n]$")
plt.tight_layout()
plt.savefig("filter_filtered.png")
plt.show()

```

Listing 20.1: 024_fft_filter/fft_filter.py

Example: Whitening filter

It is also possible to approach the previous example in a more automatic fashion. We can also simply form a filter that whitens the spectrum, i.e., ensures that all the spectral components are unity magnitude after at the filter output. This is a useful filter, especially if the signal also contains random noise.

In this example, we'll use the same signals as we used in the previous example. Instead of reducing the amplitudes of spectral

components within a certain range, we'll simply set the amplitudes of all spectral components to unity:

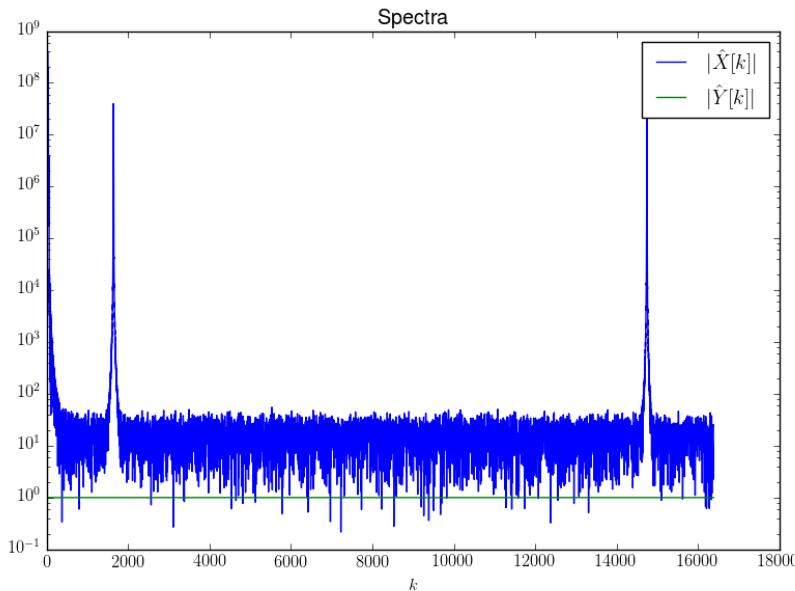
$$\hat{h}[k] = \frac{1}{|\hat{x}[k]|} \quad (20.10)$$

This type of filter is called a whitening filter¹

When this filter is applied in frequency domain:

$$\hat{y}[k] = \hat{h}[k]\hat{x}[k], \quad (20.11)$$

the resulting signal spectrum $|\hat{y}[k]| = 1$ will have a unity magnitude at all frequencies. Figure 20.4 shows the spectrum of the original signal $\hat{x}[k]$ (this is the same as in the previous example), as well as $|\hat{y}[k]| = 1$.



¹ In statistical signal processing, $|\hat{x}[k]|$ would be replaced with an estimate of the mean squared spectral power $\sqrt{\mathbb{E}|\hat{x}[k]|^2}$. Here \mathbb{E} is the statistical expectation operator.

Figure 20.4: Windowed magnitude spectrum of the original signal $|\hat{x}[k]|$ and the magnitude spectrum of the whitened signal $|\hat{y}[k]|$.

When we now inverse Fourier transform $\hat{y}[k]$, we obtain the filtered signal:

$$y[n] = \mathcal{F}_D^{-1}\{\hat{y}[k]\}. \quad (20.12)$$

The filtered signal is shown in Figure 20.5. Note that while $|\hat{y}[k]| = 1$, there is still information in the phase $\angle\hat{y}[k]$, which allows us to detect the weak signal $x_2[n]$.

The Python code used to produce this example is shown in Listing 20.2.

```
# 
# Using DFT to create a filter that
# attenuate spectral components to allow
# weak signals buried under the strong but
```

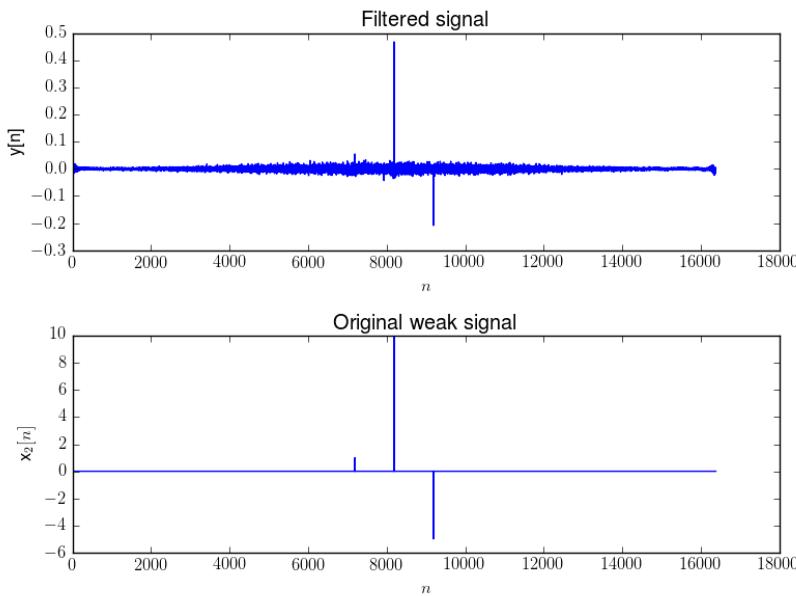


Figure 20.5: The output of the whitening filter $y[n]$. It is possible to identify the weak signal $x_2[n]$.

```

# spectrally narrow signals to be found.
#
import numpy as n
import matplotlib.pyplot as plt
import scipy.signal as s

N=4096*2*2
nn=n.arange(N)
freqs=[0.0012,0.0021,0.0032,0.1,0.9]
A=[1e5,0.5e5,1e3,1e4,0.5e4]
strong_signal=n.zeros(N)
for i,f in enumerate(freqs):
    strong_signal+=A[i]*n.cos(n.pi*freqs[i]*nn+n.random.randn(1))

# weak signal at the middle of the signal
weak_signal=n.zeros(N)
weak_signal[int(N/2)]=10.0
weak_signal[int(N/2)+1000]=-5.0
weak_signal[int(N/2)-1000]=2.0

# add noise
noise=n.random.randn(N)*0.2
x=strong_signal+weak_signal+noise

plt.subplot(311)
plt.plot(nn,strong_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_1[n]$")
plt.title("Strong signal")
plt.subplot(312)
plt.plot(nn,weak_signal)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x_2[n]$")

```

```

plt.title("Weak signal")
plt.subplot(313)
plt.plot(nn,x)
plt.xlabel("Sample ($n$)")
plt.ylabel("$x[n]=x_1[n]+x_2[n]+\backslash\backslash i[n]$")
plt.title("Total signal")
plt.tight_layout()
plt.savefig("whiten_signals.png")
plt.show()

# windowed zero-padded DFT
w=s.hann(N)
WX=n.fft.rfft(w*x,2*N)

# create a filter than sets filters all spectral components to
# unity magnitude
H=1.0/n.abs(WX)

plt.semilogy(n.abs(WX),label="$|\hat{x}[k]|$")
plt.semilogy(n.abs(H*WX),label="$|\hat{y}[k]|$")
plt.legend()
plt.xlabel("$k$")
plt.title("Spectra")
plt.tight_layout()
plt.savefig("whiten_spec.png")
plt.show()

# apply filter in frequency domain and
# IFFT the signal into time domain.
wy=n.fft.irfft(H*WX)[0:N]

# plot filtered signal and compare with original weak signal
# buried under
# stronger signals
plt.subplot(211)
plt.plot(nn,wy)
plt.title("Filtered signal")
plt.xlabel("$n$")
plt.ylabel("$y[n]$")

plt.subplot(212)
plt.plot(nn,weak_signal)
plt.title("Original weak signal")
plt.xlabel("$n$")
plt.ylabel("$x_2[n]$")
plt.tight_layout()
plt.savefig("whiten_filtered.png")
plt.show()

```

Listing 20.2: 024_fft_filter/whitening_filter.py

Exercises: Arbitrary Frequency Response Filters

1. The Python code in Listing 20.3 creates a signal consisting of narrowband sinusoids with large amplitudes ($x_1[n]$), and three time shifted unit impulses with weak amplitudes ($x_2[n]$):

$$x[n] = x_1[n] + x_2[n] \quad (20.13)$$

```
import numpy as np

N = 16384
n = np.arange(N)
freqs = [0.0003, 0.012, 0.055, 0.102, 0.85]
A = [1e5, 0.5e5, 1e3, 1e4, 0.5e4]
x1 = np.zeros(N)

for i, f in enumerate(freqs):
    x1 += A[i]*np.cos(np.pi*f*n + np.random.randn(1))

# Weak signal at the middle of the signal.
x2 = np.zeros(N)
x2[int(N/2)] = 10.0
x2[int(N/2)+1000] = -5.0
x2[int(N/2)-1000] = 1.0

x = x1 + x2
```

Listing 20.3: fftex.py

- a) Estimate the spectrum of the signal $x[n]$ using a Hann window $w[n]$ of length $N = 16384$ to reduce spectral leakage.

$$\hat{x}_w[k] = \sum_{n=0}^{N-1} w[n]x[n]e^{-i\frac{2\pi}{N}nk} \quad (20.14)$$

Use FFT to evaluate the DFT.

Make a plot of the magnitude spectrum with power in dB scale $10\log_{10} |\hat{x}_w[\hat{\omega}_k]|^2$. Only plot the positive frequencies between 0 and π with units of radians per sample. Identify the frequency ranges that are occupied by strong narrowband spectral components.

- b) Filter out the large magnitude frequency components in frequency domain and inverse DFT to obtain a time domain representation of the signal.

$$y[n] = \mathcal{F}_D^{-1} \left\{ \hat{h}[k] \hat{x}_w[k] \right\} \quad (20.15)$$

Use FFT to evaluate the inverse DFT. Your plot should look like Figure 20.3.

- c) Explain why the filtered signal $y[n]$ still looks like the original weak signal $x_2[n]$ consisting of unit impulses, even though it is missing some frequency components.

Suggested solutions: Arbitrary Frequency Response Filters

1. Consider two signals of the form:

$$x_1[n] = \sum_{i=1}^5 A_i \cos(2\pi f_i i) + w_n,$$

$$x_2[n] = a_1\delta[n - 8192] + a_2\delta[n - 9192] + a_3\delta[n - 7192],$$

where $|A_i| > |a_j|$ for every pair i, j , here w_n is white noise.² In Listing 20.4 is code to generate these signals and plot them.

```
import matplotlib.pyplot as plt
import numpy as np

N = 16384
n = np.arange(N)
freqs = [0.0003, 0.012, 0.055, 0.102, 0.85] # x1[n]
        frequencies
A = [1e5, 5e5, 1e3, 1e4, 0.5e4]                 # x1[n]
        amplitudes
x1 = np.zeros(N)

# Create the first signal.
for i, f in enumerate(freqs):
    x1 += A[i]*np.cos(np.pi*f*n + np.random.randn(1))

# Create the second signal (the weak signal).
x2 = np.zeros(N)
x2[N//2] = 10.0
x2[N//2 + 1000] = -5.0
x2[N//2 - 1000] = 1.0

x = x1 + x2

# Plot the signals.
plt.plot(n, x)
plt.xlabel("Samples")
plt.ylabel("$x[n]$")
# Call this if needed:
# plt.show()
```

Listing 20.4: Example signal code

- To estimate the spectrum, we use the Hann window to avoid spectral leakage. The code for computing the spectrum using the Hann window is shown in Listing 20.5.

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.signal.windows import hann

def convert_to_decibel(x: np.ndarray) -> np.ndarray:
    """Function to convert to dB."""
    return 10*np.log10(np.abs(x)**2)
```

² White noise is considered as a random variable with a normal distribution with $\mu = 0$ and variance $\sigma^2 = 1$ here

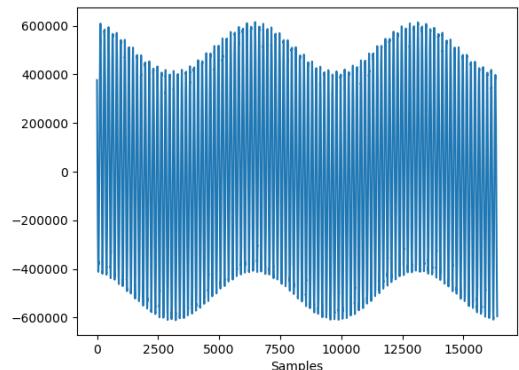


Figure 20.6: Noisy signal we want to filter

```

N = 16384
nn = np.arange(N)
freqs = [0.0003, 0.012, 0.055, 0.102, 0.85] # x1[n]
    frequencies
A = [1e5, 5e5, 1e3, 1e4, 0.5e4]           # x1[n]
    amplitudes
x1 = np.zeros(N)

# Create the first signal.
for i, f in enumerate(freqs):
    x1 += A[i]*np.cos(np.pi*freqs[i]*nn + np.random.randn(1))

# Create the second signal (the weak signal).
x2 = np.zeros(N)
x2[N//2] = 10.0
x2[N//2 + 1000] = -5.0
x2[N//2 - 1000] = 1.0

x = x1 + x2

# Hann window to reduce spectral leakage.
w = hann(N)

xw = np.fft.rfft(w*x, 2*N)      # Use FFT to compute the
                                spectrum.
om_freqs = np.linspace(0, np.pi, num=len(xw)) # Partition
                                                the interval (0,pi).

plt.plot(om_freqs, convert_to_decibel(xw))
plt.xlabel(r"\hat{\omega} (rad / sample)")
plt.ylabel(r"\hat{x}[k]^2 (dB)")
plt.title("Spectrum of $x[n]$")
# Call this if needed:
# plt.show()

```

Listing 20.5: Spectrum for noisy signal in Figure 20.6

- b) To filter out the noise, we use a filter that will remove strong spectral components and keep the weak components constant at 1.0. That is, our filter will be:

$$\hat{h}[k] = \begin{cases} \frac{1}{|\hat{x}_w[k]|}, & \text{for strong spectral components,} \\ 1, & \text{otherwise.} \end{cases}$$

Where $\hat{x}_w[k]$ is the Hann windowed tapered signal. Looking at the spectral power, we can determine which components need to be lowered. After doing this we apply the inverse DFT to obtain our filtered signal,

$$x[k] = \mathcal{F}_D^{-1}\{\hat{x}_w[k]\hat{h}[k]\}.$$

The implementation of this is shown in Listing 20.6.

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.signal.windows import hann

```

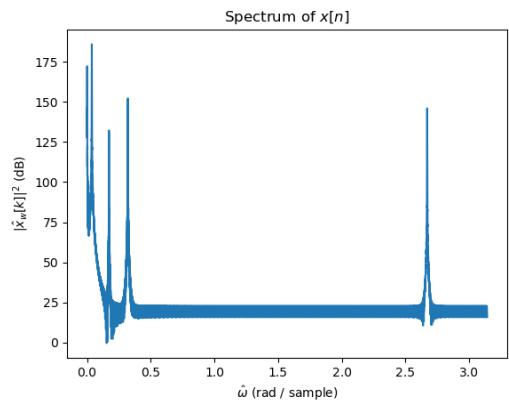


Figure 20.7: Spectrum in dB for the signal shown in Figure 20.6

```

def convert_to_decibel(x: np.ndarray) -> np.ndarray:
    """Function to convert to dB."""
    return 10*np.log10(np.abs(x)**2)

N = 16384
n = np.arange(N)
freqs = [0.0003, 0.012, 0.055, 0.102, 0.85] # x1[n]
        frequencies
A = [1e5, 5e5, 1e3, 1e4, 0.5e4]                 # x1[n]
        amplitudes
x1 = np.zeros(N)

# Create the first signal.
for i, f in enumerate(freqs):
    x1 += A[i]*np.cos(np.pi*f*n + np.random.randn(1))

# Create the second signal (the weak signal).
x2 = np.zeros(N)
x2[N//2] = 10.0
x2[N//2 + 1000] = -5.0
x2[N//2 - 1000] = 1.0

x = x1 + x2

# Hann window to reduce spectral leakage.
w = hann(N)

xw = np.fft.rfft(w*x, 2*N)                         # Use FFT to
    compute the spectrum.
om_freqs = np.linspace(0, np.pi, num=len(xw)) # Partition
    the interval (0, pi).

# Define a filter to reduce strong spectral components.
h = np.ones(len(xw)) # Initialize each entry to 1.

# Lower the strong spectral components.
# Look at the previous exercise output plot to determine
    the intervals
# alternatively, plot the spectrum with samples on x-axis
    instead of \hat{\omega}.
h[:1050] = 1.0/np.abs(xw[:1050])
h[1500:1860] = 1.0/np.abs(xw[1500:1860])
h[13500:14200] = 1.0/np.abs(xw[13500:14200])

# Plot the spectral power to compare the two signals.
plt.plot(om_freqs, convert_to_decibel(xw), label="Original
    spectrum")
plt.plot(om_freqs, convert_to_decibel(xw*h), label="Windowed
    spectrum")
plt.xlabel(r"\hat{\omega} (rad / sample)")
plt.ylabel("Power of spectrum (dB)")
plt.legend()
# Call this if needed:
# plt.show()

# Finally, inverse DFT to obtain the filtered signal.
filter_signal = np.fft.irfft(h*xw)
plt.plot(filter_signal)

```

```

plt.xlabel("Samples")
plt.title("Filtered signal")
# Call this if needed:
# plt.show()

```

Listing 20.6: Filtering of the signal

Running this code will generate the plots shown in Figure 20.8 and 20.9.

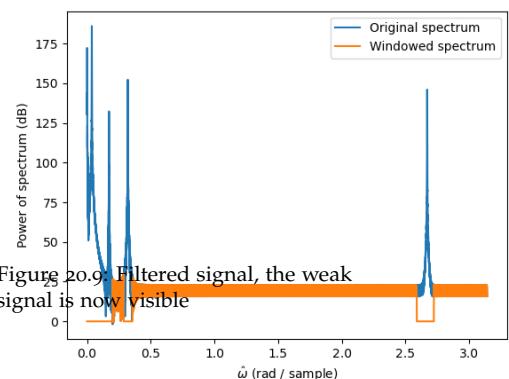
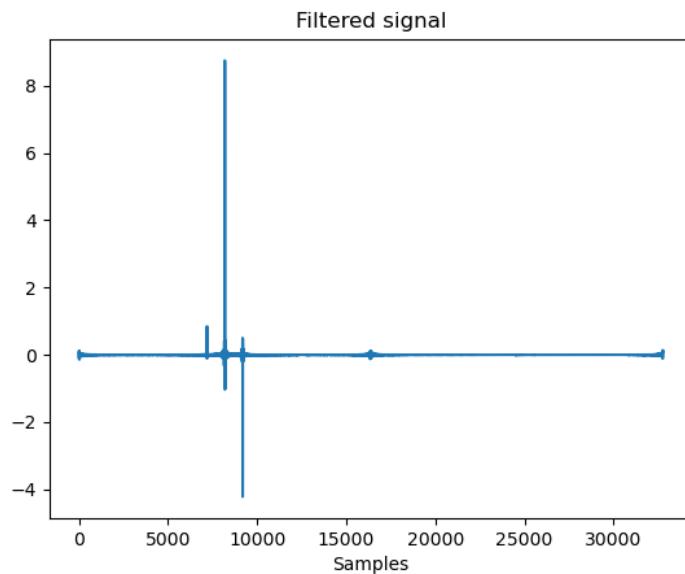


Figure 20.8: The comparison of the spectral power

- c) The filter in this case reduces the strong frequencies while at the same time keeping lower frequencies fixed. The effect is that the original signal which had a lot of noise coming from $x_1[n]$ will have that noise significantly reduced. The remaining information in the signal is the other part, being $x_2[n]$, which consists of unit impulses for which the frequency components were kept at 1. Therefore, the filtered signal looks like $x_2[n]$ even though a lot of frequency components have been removed.

21

Programming Assignment 3

In 2017, the Nobel Prize in physics was awarded to Rainer Weiss, Barry Barish and Kip Thorne for the discovery of gravitational waves. Only two years earlier, on September 14, th 2015, 09:50:45 UTC, the two Laser Interferometer Gravitational-Wave Observatory (LIGO) instruments detected a gravitational wave for the first time in history. The existence of gravitational waves had been predicted by Einstein's general theory of relativity, but never before detected in situ.

The first gravitational wave event detected by LIGO is thought to be created from a collision of two black holes, which sends out a localized chirp like pulse in the space-time. LIGO utilizes two detectors, which are spaced 3000 km apart. These detectors measure *strain* ($\Delta L/L$) as a function of time. Here ΔL is the variation in length and L is the total length in which the variation is measured. In other words, strain is the normalized variation in length L of the interferometer line due to gravitational-wave. Figure 21.5 provides a high level overview of the measurement.

LIGO uses two geographically separated stations to measure gravitational waves: Hanford (H_1), and Livingston (L_1). The gravitational wave propagates at the speed of light $c \approx 3 \cdot 10^8$ m/s. If the same signal is detected at two different places with a time difference less than or equal to the speed of light propagation time between the sensors, then this provides more confidence that the event is in fact real, and not caused by for example local seismic activity. A third sensor would allow determining the direction of arrival based on time of arrival.

The LIGO data is severely corrupted with instrumental noise. This noise is much larger in amplitude than the gravitational wave signal. However, this noise is very narrowband in nature, and it can be filtered out using relatively basic signal processing techniques without affecting the relatively broad band gravitational signal very much. Such filtering is routinely used by LIGO to improve the sensitivity of the instrument.

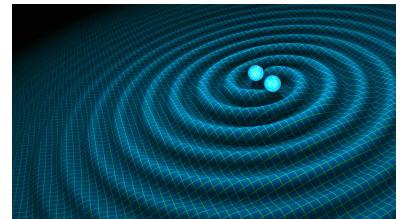


Figure 21.1: Artist's depiction of gravitational waves created by a merger of two neutron stars. Credits: US National Science Foundation.



Figure 21.2: The Hanford and Livingston interferometers. Credits: LIGO.

Verification of scientific results is an important part of science. UiT does not have the resources for building a giant interferometer, and this course really isn't about solving Einstein's field equations, so we won't be able to reproduce all the results. However, we can verify the signal processing part. In this programming assignment, your task is to independently develop signal processing software to verify that you can also find the gravitational wave signature in the LIGO measurements.

Instructions

We expect you to complete the listed signal processing tasks. The submission form is a written report, which answers the questions given in each part of this assignment. The report should include the code with comments that indicate what each part of the program does.

You will only need to know about signal processing concepts taught in FYS-2006. You do not need to know anything about gravitational waves or the LIGO instrument in order to complete the assignment. The lecture notes for the course contain several helpful signal processing examples. Take a look at lecture notes for Week 43 on *spectral analysis*, and *arbitrary frequency response filters*.

Instruction for reading the data

To begin, download the data files and a simple program that demonstrates reading these files from this location: <https://bit.ly/3bIcR1B>. These data files contain real measurements from LIGO starting at 2015-09-14T09:50:30 UTC. After downloading the files, the next step is to read the strain signal from the data files. In Python, this can be done using the h5py module¹.

```
import h5py
h = h5py.File("file.hdf5", "r")
data = h["strain/Strain"][:]()
```

You will need to read two data vectors. One for the Livingston station (L_1) and one for the Hanford station (H_1). We will use the symbol $x_H[n]$ of the Hanford signal and $x_L[n]$ for the Livingston signal.

¹ Make sure you have the h5py module installed on your computer!

1. Data

The sample rate of both of the signals is $f_s = 4096$ Hz. The samples in both signals are synchronized in time, i.e., sample n in signal $x_H[n]$

and $x_L[n]$ occur at the same time.

- a) Write code to read the Hanford and Livingston signals from the data file.
- b) How many samples are in each of the signals: $x_H[n]$ and $x_L[n]$?
- c) How many seconds of signal does each of the two data vectors $x_H[n]$ and $x_L[n]$ represent?
- d) The gravitational wave signal is a one dimensional real-valued signal that is spectrally confined to the frequency range $[-300, 300]$ Hz. Is the signal sampled at a sufficiently high sample rate to retain all the gravitational wave information?
- e) What is the sample spacing in units of seconds?

2. Plotting the data

In order to see what the signals look like, you will need to plot the data.

- a) Plot the signals $x_H[n]$ and $x_L[n]$, with time in seconds on the horizontal axis and strain on the vertical axis. Assume that time at the beginning of the signal array starts at 0 seconds. Label the axes of your plot. Use separate plots for $x_H[n]$ and $x_L[n]$ signals. Hint: you can use the `plt.plot(t, signal)` command found in Matplotlib. Use an array `t` to denote the seconds of each sample of the array `signal`.
- b) What are the minimum, maximum, and mean values of the $x_H[n]$ and $x_L[n]$ signals?

3. Selecting a tapered window function

You will need to apply a discrete Fourier transform to analyze the spectral content of the signal. You will need to select a suitable tapered window function in order to obtain a good rejection of out of band signals. In this task, we'll use a synthetic narrowband signal to compare the performance of a discrete Fourier transform (DFT) based spectral analysis using a tapered window to spectral analysis done without a tapered window.

In order to calculate the magnitude spectrum of a signal $x[n]$ of length N , you need to evaluate a DFT on the signal:

$$\hat{x}[k] = \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} kn}. \quad (21.1)$$

and the windowed signal:

$$\hat{x}_w[k] = \sum_{n=0}^{N-1} w[n]x[n]e^{-i\frac{2\pi}{N}kn}. \quad (21.2)$$

Hint: Use the `fft` function to evaluate the DFT. This function is available in Python as `numpy.fft.fft`.

- a) Choose a tapered window function and implement it. Hint: you can use window functions available in the `scipy.signal` module.
- b) In order to get an idea of how your window function behaves, apply it to a sinusoidal signal

$$x[n] = \cos(2\pi f n / f_s) \quad (21.3)$$

with frequency $f = 31.5$ Hz, which is sampled at $f_s = 4096$ Hz. Sample signal at $n \in [0, 1, 2, \dots, 4095]$. Make a plot of the signal $x[n]$ and the windowed signal $w[n]x[n]$. Use a window of the same length as your signal $N = 4096$. Hint: You can use `numpy.arange(N)` to create a sequence of integers between 0 and $N - 1$.

- c) The FFT algorithm will evaluate $\hat{x}[k]$ at integer values of k between 0 and $N - 1$. What frequencies f_k in hertz do frequencies $\hat{\omega}_k = 2\pi k / N$ in radians per sample correspond to on the principal spectrum ($-f_s/2 < f_k < f_s/2$)?
- d) Which values of k correspond to a frequency f_k that is nearest to 31.5 and -31.5 hertz?
- e) Estimate the power spectrum of the windowed signal $w[n]x[n]$ (with tapering) and the signal $x[n]$ (without tapering). Plot the power spectrum in decibel scale (power) for both. Use frequency in Hz on the horizontal axis and magnitude squared $10 \log_{10}(|\hat{x}[k]|^2)$ (decibels) on the vertical axis. Plot both positive and negative frequencies. Hint: You can use `numpy.fft.freq` and `numpy.fft.fftshift` to determine what frequency (in hertz) each FFT bin k corresponds to, and to order the frequencies in ascending order.
- f) Mark the locations of 31.5 and -31.5 Hz on the plot of the power spectrum.
- g) The frequency corresponding to $\hat{\omega} = \pi$ radians per second or $f = 2048$ Hz is non-zero for both $\hat{x}[k]$ and $\hat{x}_w[k]$. However, your test signal has a frequency of 31.5 Hz. Why is there a non-zero frequency component anywhere else than at 31.5 Hz when you analyze the signal?

- h) How many decibels is the frequency response better with the tapered window function ($10 \log_{10}(|\hat{x}_w[k]|^2)$) than without ($10 \log_{10}(|\hat{x}[k]|^2)$) at frequency $\hat{\omega} = \pi$ (radians per second)?

4. Estimating the spectrum of the LIGO signal

- a) Calculate the power spectrum of the LIGO signals $|\hat{x}_L[k]|^2$ and $|\hat{x}_H[k]|^2$. Here $\hat{x}_L[k]$ is the windowed DFT of the Livingston signal $x_L[n]$ and $\hat{x}_H[k]$ is the windowed DFT of the Hanford signal $x_H[n]$. The windowed DFT is obtained using

$$\hat{x}[k] = \sum_{n=0}^{N-1} w[n]x[n]e^{-i\frac{2\pi}{N}kn} \quad (21.4)$$

Perform the DFT over the whole dataset, i.e., N is the number of samples in the whole signal vector. Use the window function that you have chosen in the previous exercise, but make sure that use a window of length N , where N is the LIGO data vector length.

Hint: Use FFT.

- b) Plot the results with frequency in Hz in the horizontal axis and power using decibel scale on the vertical axis. Make one plot for the Hanford and one plot for the Livingston. Plot only the positive frequencies.
- c) On what frequencies are there strong spectral components in the Hanford and Livingston power spectra? Use Hz as the unit of frequency. Identify up to 12 frequency bands that contain narrowband interference. Label these regions on the plot of the power spectrum.

5. Whitening filter

In order to remove instrumental noise, you will next implement a whitening filter and apply it to the LIGO signal. A whitening filter is a filter that modifies the amplitudes of each spectral component in such a way that the magnitude spectrum of the filter output is constant-valued.

A whitening filter can be implemented as an FIR filter, which in frequency domain can be implemented as a multiplication:

$$\hat{y}[k] = \hat{h}[k]\hat{x}[k]. \quad (21.5)$$

Here $\hat{y}[k]$ is the DFT of the output of the filter, $\hat{h}[k]$ is the DFT of the whitening filter, and $\hat{x}[k]$ is the windowed DFT of the input signal $x[n]$.

The purpose of a whitening filter is to filter the signal in such a way that the magnitude of the output signal is unity: $|\hat{y}[k]| = 1$. This can be obtained using a filter of the form:

$$\hat{h}[k] = \frac{1}{|\hat{x}[k]|}. \quad (21.6)$$

- a) Show that $\hat{h}[k]$ as defined in equation 21.6 will filter signal $x[n]$ in such a way that $|\hat{y}[k]| = 1$.
- b) Implement a whitening filter in frequency domain $\hat{h}[k]$ for the LIGO data. Implement a separate filter for the Hanford and Livingston signals. Use all the signal as input to the windowed DFT when calculating $\hat{x}[k]$. Do not filter the signal in smaller blocks!
- c) Use an inverse discrete Fourier transform to transform the whitened signal $\hat{y}[k]$ into time-domain. Do this for both Livingston and Hanford signals separately (obtaining $y_L[n]$ and $y_H[n]$).
- d) Plot the whitened signals $y_H[n]$ and $y_L[n]$ for both Hanford and Livingston. Use x-axis for time in seconds ($t \in [0, t_{\max}]$) and y-axis for whitened strain $y[n]$. The gravitational wave signal is in the middle of the signal, between 16.2 and 16.5 seconds. If you've done everything correctly, you should see the gravitational wave signal. It looks like a chirp (see plot in Figure 21.3). However, because we are not done yet, the signal will look noisy.

6. Low-pass filtering

The gravitational wave signal is at frequencies below 300 Hz. Design a simple running mean averaging low-pass filter

$$y[n] = \frac{1}{L} \sum_{k=0}^{L-1} x[n-k] \quad (21.7)$$

that will attenuate spectral components with frequencies above 300 Hz.

- a) Find an integer value of L such that the filter will reduce the power of frequency components at $f = 300$ Hz by approximately -6 dB compared to the filter output for a $f = 0$ Hz signal.
- b) Plot the power spectral response of the filter in dB scale ($10 \log_{10} |\mathcal{H}(\omega)|^2$), where $\mathcal{H}(\omega)$ is the discrete-time Fourier transform of the FIR filter coefficients of the averaging filter. Use frequency on the horizontal axis in Hz. Label the -6 dB point in frequency and in

power spectral response, verifying that the -6 dB point is close to 300 Hz.

- c) What is the time delay τ to the signal introduced by the filter, in seconds?
- d) Apply the running mean average low-pass filter on the whitened Hanford and Livingston signals ($y_H[n]$ and $y_L[n]$).
- e) Undo the effects of the filter time-delay by shifting the signal in time. You can do this by adjusting the time variable $t' = t - \tau$, instead of filtering the signal.
- f) Plot the low pass filtered whitened Hanford and Livingston signals. You should see the gravitational wave signal more clearly now. Use the horizontal axis for time and the vertical axis for the signal amplitude. Plot only the time interval between 16.1 and 16.6 seconds where the gravitational wave signal is located. Compare your plot with Figure 21.3. Your plot should be similar, but not necessarily exactly the same, as your filter is not exactly the same as the one used for Figure 21.3.

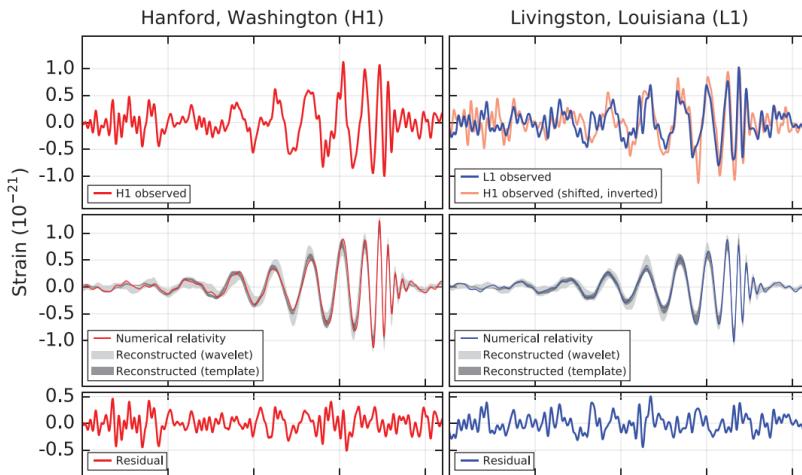


Figure 21.3: The figure is from: B. P. Abbott et al. (LIGO Scientific Collaboration and Virgo Collaboration) Phys. Rev. Lett. 116, 061102 – Published 11 February 2016

7. Time delay

Gravitational waves are expected to propagate at the speed of light. The Hanford and Livingston detectors are separated by about 3000 km. A gravitational wave will propagate this distance in about 10 ms.

The gravitational wave angle of arrival is not known, but the relative time delay between the signal detected at Hanford and

Livingston is expected to be $-10 < \tau < 10$ ms, if it is moving at the speed of light in vacuum.

- Determine the time separation between the two signals *by plotting the magnitudes* of the filtered gravitational wave signals ($|y_L[n]|$ and $|y_H[n + n_0]|$) with different delays n_0 on the same plot. Try different values of n_0 until the signals are approximately aligned in time. The magnitude of the signal is used to avoid phase-time ambiguities. Hint: Use magnitude, not amplitude. Magnitude can be obtained using `numpy.abs`.
- What value do you obtain for the sample delay n_0 ?
- What time delay τ in seconds does the sample delay n_0 correspond to?
- Is the time delay τ in agreement with gravitational-wave propagation speed (i.e., that $-10 < \tau < 10$ ms)?

8. Dynamic spectrum

Study the time-frequency behavior of the gravitational wave signal.

- Calculate the dynamic spectrum (spectrogram) of the low-pass filtered whitened signal $|\hat{x}[t, k]|$, where t is time and k is frequency.
- Plot the dynamic power spectrum $|\hat{x}[t, k]|^2$ in dB scale. Perform this in such a way that you can see the time-frequency response behavior of the gravitational wave signal clearly. Plot your result at between $t = 15.5$ s and $t = 17$ s. Hint: You will need to use zero-padding, a tapered window, and overlapping windows. An example of the time-frequency domain behavior of the gravitational wave signal is shown in Figure 21.4.

9. Extra task

For earning an extra point, improve any part of the signal processing in a way that you see fit. Document your improvements. You can, e.g., try to create a filter that removes only the strong frequency components from the signal, or you can use a better low-pass filter.

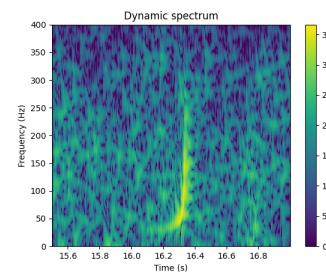


Figure 21.4: A dynamic spectrum plot of the gravitational wave signal.

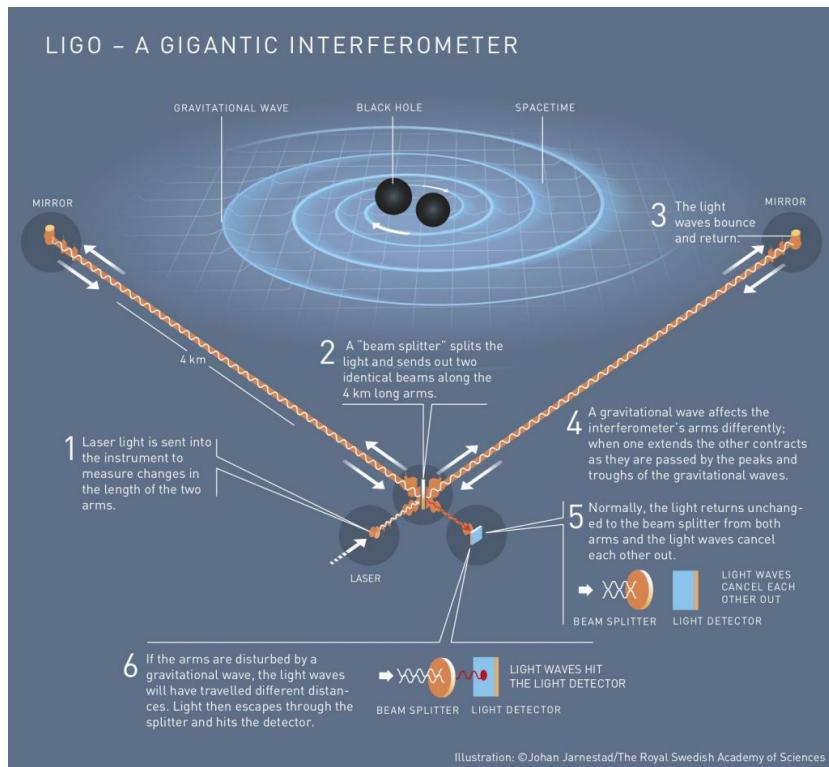


Figure 21.5: The gravitational wave measurement using a Michelson-Morley interferometer. Credits: Johan Jarnestad, The Royal Swedish Academy of Sciences.

22

Z-transform

The *z-transform* is a general frequency domain representation of discrete-time signals. A special case of the *z-transform* is the discrete-time Fourier transform.

The main benefit of the *z-transform* is that it allows us to use algebraic manipulations of complex-valued polynomials to study the frequency domain behavior of discrete-time signals. This can simplify calculations and provide further mathematical intuition. The *z-transform* also provides a framework that can be used to derive numerically efficient low-order filters¹. The *z-transform* also has applications, e.g., in statistics and statistical signal processing.

¹ filters that require only a few computations to apply on signals

System function $\mathcal{H}(z)$

The *system function* $\mathcal{H}(z)$ provides an alternative frequency domain representation of the frequency response of LTI systems. We'll start by defining what this is.

Recall that a discrete-time LTI system can be represented as a convolution of an input signal $x[n]$ with the impulse response $h[n]$ of the LTI system:

$$y[n] = \mathcal{T}\{x[n]\} \quad (22.1)$$

$$= h[n] * x[n] \quad (22.2)$$

$$= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \quad (22.3)$$

The system function arises from investigating what is the output of this LTI system when using an input signal of the form $x[n] = z^n$, where $z \in \mathbb{C}$ is an arbitrary complex number. What happens to this

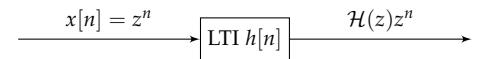


Figure 22.1: The motivation for the *z-transform* is studying how an LTI system modifies a signal z^n with $z \in \mathbb{C}$.

signal when it is fed into an LTI system? Let's find out:

$$y[n] = \mathcal{T}\{x[n]\} \quad (22.4)$$

$$= \sum_{k=-\infty}^{\infty} h[k]x[n-k] \quad (22.5)$$

$$= \sum_{k=-\infty}^{\infty} h[k]z^{n-k} \quad (22.6)$$

$$= \underbrace{\left(\sum_{k=-\infty}^{\infty} h[k]z^{-k} \right)}_{\mathcal{H}(z)} z^n \quad (22.7)$$

$$= \mathcal{H}(z)z^n. \quad (22.8)$$

The output signal is the original input signal z^n multiplied with $\mathcal{H}(z)$. This term is called the *system function* of the LTI system:

$$\boxed{\mathcal{H}(z) = \sum_{k=-\infty}^{\infty} h[k]z^{-k}} \quad (22.9)$$

Sometimes the term *transfer function* is used instead of the system function ². They both refer to $\mathcal{H}(z)$.

The system function also happens to be a transformation called the *z-transform* of the signal $h[n]$. Recall that this is similar to the frequency response of an LTI system, which we investigated earlier. There we investigated the response of a signal of the form $x[n] = Ae^{i\hat{\omega}n}$ for a discrete-time LTI system.

² The term transfer function is used e.g., by the SciPy z-transform functions.

What is the meaning of z^n ?

What does a signal of the form $x[n] = z^n$ look like? The variable $z \in \mathbb{C}$ is a complex number. We can express a complex number in polar form $z = Ae^{i\hat{\omega}}$, where $A = |z|$ and $\hat{\omega} = \angle z$. This means that:

$$\boxed{z^n = A^n e^{i\hat{\omega}n}} \quad (22.10)$$

This signal is exponentially decaying when $A < 1$, exponentially growing when $A > 1$, or constant in amplitude when $A = 1$. The meaning of $\hat{\omega}$ is the familiar normalized angular frequency in units of radians per sample.

Figure 22.2 shows several examples of a signal of the form $z^n = A^n e^{i\hat{\omega}n} u[n]$. The unit step function $u[n]$ is used to restrict the values of z^n to be non-zero only for positive values of n .

The signal z^n is more general than just a complex sinusoidal signal $e^{i\hat{\omega}n}$. It allows the amplitude of the signal to exponentially grow, decay, or stay constant.

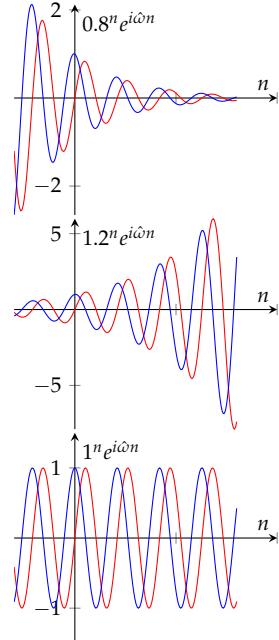


Figure 22.2: Several examples of the signal $z^n = A^n e^{i\hat{\omega}n}$ with different values of A .

Z-transform

Forward transform

The forward z-transform $X(z)$ of an arbitrary discrete-time signal $x[n]$ is defined as:

$$X(z) = \mathcal{Z}\{x[n]\} = \sum_{k=-\infty}^{\infty} x[k]z^{-k}. \quad (22.11)$$

The z-transform is not necessarily defined everywhere, as the sum may not converge for some values of z . The set of points S_{ROC} on the complex plane where the sum $X(z)$ converges to a finite value is known as the *region of convergence*:

$$S_{\text{ROC}} = \{z \in \mathbb{C} : |X(z)| < \infty\} \quad (22.12)$$

In terms of filter design, the region of convergence indicates the set of signals for which a filter provides a finite output.

Reverse transform

The inverse z-transform is defined as a contour integral:

$$x[n] = \mathcal{Z}^{-1}\{X(z)\} = \frac{1}{2\pi i} \oint_C X(z)z^{n-1} dz. \quad (22.13)$$

Here C is a closed loop that encircles the origin within the region of convergence.

Solving the inverse z-transform directly using this formula requires the use of contour integration³. However, one typically does not need to use this formula, because with practical signal processing applications, it is nearly always possible to algebraically manipulate the z-transform polynomial $X(z)$ in such a way that it is possible to use a table of elementary z-transform pairs to obtain an inverse z-transform symbolically.

Relationship to the discrete-time Fourier transform

It is possible to determine the DTFT of a signal using a z-transform of the signal $X(z)$. This is done by substituting $z = e^{i\hat{\omega}}$. In other words, evaluating the z-transform along the unit circle.

Recall the definition of the z-transform:

$$X(z) = \sum_{k=-\infty}^{\infty} x[k]z^{-k}. \quad (22.14)$$

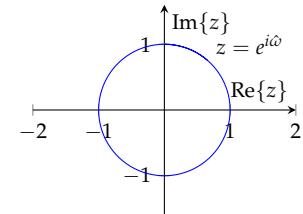


Figure 22.3: The function $z = e^{i\hat{\omega}}$ is a parametric curve for a circle on the complex plane.

³ This topic is beyond the scope of this course. You will not need to know how to solve Equation 22.13 in the exam.

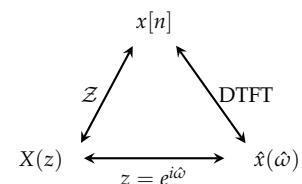


Figure 22.4: The z-transform $X(z)$ evaluated on the unit circle $z = e^{i\hat{\omega}}$ corresponds to the discrete-time Fourier transform.

If we replace $z = e^{i\hat{\omega}}$, we obtain the discrete-time Fourier transform:

$$\hat{x}(\hat{\omega}) = \sum_{k=-\infty}^{\infty} x[k]e^{-i\hat{\omega}k}. \quad (22.15)$$

In a sense, the z -transform is a more general frequency domain representation of a discrete-time signal than a discrete-time Fourier transform, as a subset of the z -transform is the discrete-time Fourier transform.

One application is that the z -transform of the impulse response $h[n]$ of an LTI system can be used to obtain the frequency response of an LTI system:

$$\boxed{\mathcal{H}(\hat{\omega}) = \mathcal{H}(z)|_{z=e^{i\hat{\omega}}}} \quad (22.16)$$

In other words, if one knows the system function, it is possible to evaluate the frequency response.

Linearity

The z -transform is a linear transformation:

$$\boxed{\alpha_1 x_1[n] + \alpha_2 x_2[n] \xrightarrow{Z} \alpha_1 X_1(z) + \alpha_2 X_2(z)}. \quad (22.17)$$

To show this, let's investigate the following linear combination of signals $x[n] = \alpha_1 x_1[n] + \alpha_2 x_2[n]$. The z -transform for $x[n]$ is:

$$X(z) = \sum_n (\alpha_1 x_1[n] + \alpha_2 x_2[n]) z^{-n} \quad (22.18)$$

$$= \alpha_1 \sum_n x_1[n] z^{-n} + \alpha_2 \sum_n x_2[n] z^{-n} \quad (22.19)$$

$$= \alpha_1 X_1(z) + \alpha_2 X_2(z). \quad \square \quad (22.20)$$

The result is a linear combination of the z -transforms of the two individual signals that form the signal $x[n]$.

Time-shifted unit impulse $\delta[n - n_0]$

One of the elementary z -transform pairs is a time-shifted unit impulse:

$$\boxed{x[n] = \delta[n - n_0] \xrightarrow{Z} X(z) = z^{-n_0}} \quad (22.21)$$

A time-shifted unit impulse is a basic building block of discrete-time signals, which have a finite number of non-zero elements.

This can be easily obtained using Equation 22.11 for the forward

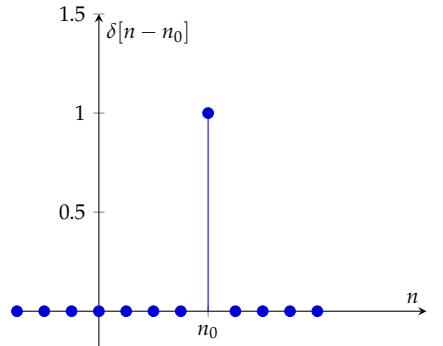


Figure 22.5: A time-shifted unit impulse is a basic building block of arbitrary signals. If you know the z -transform of this signal, you can determine the z -transform of an arbitrary finite-length discrete-time signal, as it will be a linear combination of time-shifted unit impulses.

z -transform:

$$X(z) = \mathcal{Z}\{\delta[n - n_0]\} \quad (22.22)$$

$$= \sum_{k=-\infty}^{\infty} \delta[k - n_0] z^{-k} \quad (22.23)$$

$$= z^{-n_0} \quad \square \quad (22.24)$$

The special case of this is the unit impulse $\delta[n]$, which results in $\mathcal{Z}\{\delta[n]\} = 1$.

It is also possible to evaluate the reverse transform of:

$$X(z) = z^{-n_0}, \quad (22.25)$$

by using Equation 22.13. We select the unit circle $|z| = 1$ as the contour of integration C . It is easy to see that the unit circle is in the region of convergence, as the system function is of constant magnitude along the unit circle $|e^{-i\hat{\omega}n_0}| = 1$.

Integration along the unit circle is achieved with variable substitution $z = e^{i\hat{\omega}}$. By sweeping the parameter $\hat{\omega}$ between 0 and 2π , the variable z draws a unit circle on the complex plane. This is the closed curve C .

With the chosen variable substitution, we find that $dz/d\hat{\omega} = ie^{i\hat{\omega}}$ and we can substitute the infinitesimal dz with $ie^{i\hat{\omega}}d\hat{\omega}$.

The limits of integration, which result in a closed counterclockwise loop around the contour of integration, are: $[0, 2\pi]$. We now have everything that we need to inverse z -transform $X(z) = z^{-n_0}$:

$$x[n] = \mathcal{Z}^{-1}\{z^{-n_0}\} \quad (22.26)$$

$$= \frac{1}{2\pi i} \oint_C z^{-n_0} z^{n-1} dz \quad (22.27)$$

$$= \frac{1}{2\pi i} \oint_C z^{n-n_0-1} dz \quad (22.28)$$

Let's now insert $dz = ie^{i\hat{\omega}}d\hat{\omega}$, $z = e^{i\hat{\omega}}$ and use the limits $0, 2\pi$ for $\hat{\omega}$:

$$x[n] = \frac{1}{2\pi i} \int_0^{2\pi} e^{i\hat{\omega}(n-n_0-1)} ie^{i\hat{\omega}} d\hat{\omega} \quad (22.29)$$

$$= \frac{1}{2\pi} \int_0^{2\pi} e^{i\hat{\omega}(n-n_0)} d\hat{\omega}. \quad (22.30)$$

Let's investigate two cases. In the first case $n \neq n_0$. It is easy to see that this integral is zero when $n \neq n_0$. This is because $n - n_0$ is an integer. The second case is when $n = n_0$, we then get:

$$x[n_0] = \frac{1}{2\pi} \int_0^{2\pi} d\hat{\omega} \quad (22.31)$$

$$= 1. \quad (22.32)$$

Combining these two results, we get:

$$x[n] = \delta[n - n_0] \quad \square \quad (22.33)$$

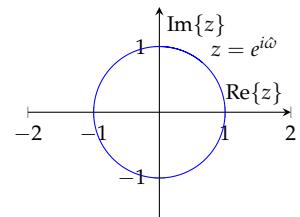


Figure 22.6: The function $z = e^{i\hat{\omega}}$ is a parametric curve for a circle on the complex plane.

Arbitrary signals of finite length

By using Equation 22.21 and linearity, you can forward and reverse z-transform an arbitrary signal $x[n]$ with a finite number of non-zero values:

$$x[n] = \sum_{k=0}^N b_k \delta[n - n_k] \xrightarrow{\mathcal{Z}} X(z) = \sum_{k=0}^N b_k z^{-n_k} \quad (22.34)$$

Here n_k denotes the time-shift of the k th non-zero element $x[n_k] = b_k$. A large fraction of practical signals are of this form, so this is a very useful formula.

Example: System function of an FIR filter

A discrete-time LTI system $y[n] = \mathcal{T}\{x[n]\}$ is defined using the following difference equation:

$$y[n] = x[n - 3] - 2x[n - 5] + 3x[n - 7]. \quad (22.35)$$

This is a finite impulse response filter (shown in Figure 22.7), with the following impulse response:

$$h[n] = \delta[n - 3] - 2\delta[n - 5] + 3\delta[n - 7]. \quad (22.36)$$

Using Equation 22.34 we get the following system function (z -transform of the LTI system impulse response):

$$\mathcal{H}(z) = z^{-3} - 2z^{-5} + 3z^{-7}. \quad (22.37)$$

We can also multiply this with z^7/z^7 and obtain a polynomial fraction with more familiar positive valued exponents of the following form:

$$\mathcal{H}(z) = \frac{z^4 - 2z^2 + 3}{z^7} \quad (22.38)$$

This means that the system function in this case is a polynomial fraction, with a fourth order polynomial on the numerator and a seventh order polynomial on the denominator.

Example: Algebraic inverse z-transform

Given a system function $\mathcal{H}(z)$, it is possible to go back into time domain and to determine the impulse response $h[n]$ of an FIR filter using simple algebraic manipulations, instead of the more involved contour integration discussed earlier. From the impulse response, it is then possible to reconstruct a difference equation describing the output of the discrete-time LTI system as a function of the input signal.

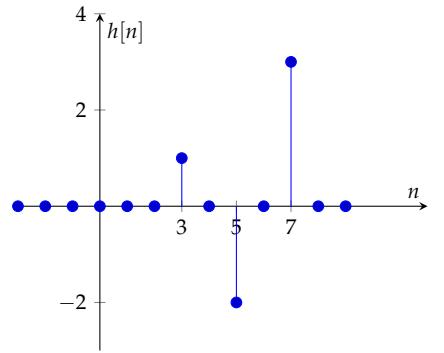


Figure 22.7: A finite impulse response $h[n]$.

For example, let's say that we have a system function of the following form:

$$\mathcal{H}(z) = \frac{3z^3 - 2z + 1}{z^2} \quad (22.39)$$

We can find the inverse z -transform of $\mathcal{H}(z)$ purely by algebraic manipulation. First, let's write Equation 22.39 as:

$$\mathcal{H}(z) = 3z^1 - 2z^{-1} + z^{-2}. \quad (22.40)$$

We can now use Equation 22.34, to obtain the impulse response, which is the inverse z -transform of $\mathcal{H}(z)$:

$$h[n] = 3\delta[n+1] - 2\delta[n-1] + \delta[n-2]. \quad (22.41)$$

This is shown in Figure 22.8. We can see that the difference equation for this LTI system can be written as

$$y[n] = 3x[n+1] - 2x[n-1] + x[n-2]. \quad (22.42)$$

Time-shift theorem

The time-shift theorem allows us to obtain the z -transform of a time delayed signal using polynomial multiplication:

$$y[n] = x[n - n_0] \xrightarrow{\mathcal{Z}} Y(z) = z^{-n_0} X(z) \quad (22.43)$$

Here $X(z)$ is the z -transform of signal $x[n]$, and $Y(z)$ is the z -transform of signal $x[n - n_0]$. We'll use this property later to prove the convolution theorem for the z -transform.

It is easy to see that the time-shift property follows from the definition of the z -transform. Let us assume that the z -transform of the signal $x[n]$ is $X(z)$:

$$X(z) = \sum_{k=-\infty}^{\infty} x[k]z^{-k}. \quad (22.44)$$

If we now delay $x[k]$ by n_0 samples, and look at the z -transform of the delayed signal $y[n] = x[n - n_0]$, we get:

$$Y(z) = \sum_{k=-\infty}^{\infty} y[k]z^{-k}, \quad (22.45)$$

$$= \sum_{k=-\infty}^{\infty} x[k - n_0]z^{-k}, \quad (22.46)$$

$$= z^{-n_0} \sum_{\ell=-\infty}^{\infty} x[\ell]z^{-\ell}, \quad (22.47)$$

$$= z^{-n_0} X(z) \quad \square \quad (22.48)$$

We used variable substitution: $k - n_0 = \ell$.

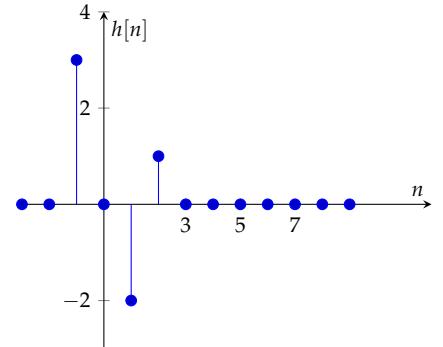


Figure 22.8: A finite impulse response $h[n]$.

Convolution theorem

The convolution theorem is of fundamental importance for theoretical and practical applications in signal processing. This theorem states that *convolution in time domain is multiplication in frequency domain*. We've already encountered this with Fourier transforms. Now we'll introduce the z -transform equivalent.

If the signal $a[n]$ is a convolution of signals $b[n]$ and $c[n]$ and the z -transforms of these signals are $A(z)$, $B(z)$, and $C(z)$, we have the following z -transform rule:

$$a[n] = b[n] * c[n] \xleftrightarrow{z} A(z) = B(z)C(z). \quad (22.49)$$

One application of this is that we can use the z -transform to investigate the z -domain representation of the output of an LTI system:

$$y[n] = h[n] * x[n] \xleftrightarrow{z} Y(z) = \mathcal{H}(z)X(z) \quad (22.50)$$

Here $\mathcal{H}(z)$ is the system function, which is the z -transform of the impulse response $h[n]$.

Proof. The proof of the z -transform convolution theorem is as follows. We start with the definition of a convolution operation:

$$a[n] = b[n] * c[n] = \sum_k b[k]c[n - k]. \quad (22.51)$$

z -transforming $a[n]$, one obtains:

$$A(z) = \sum_n (b[n] * c[n])z^{-n} \quad (22.52)$$

$$= \sum_n \left(\sum_k b[k]c[n - k] \right) z^{-n}. \quad (22.53)$$

$$= \sum_k b[k] \underbrace{\left(\sum_n c[n - k]z^{-n} \right)}_{z^{-k}C(z)} \quad (22.54)$$

$$= \sum_k b[k]z^{-k}C(z) \quad (22.55)$$

$$= \underbrace{\left(\sum_k b[k]z^{-k} \right)}_{B(z)} C(z) \quad (22.56)$$

$$= B(z)C(z) \quad (22.57)$$

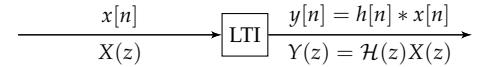


Figure 22.9: One consequence of the convolution theorem is that the z -transform of the output of an LTI system $Y(z)$ is the z -transform of the input signal $X(z)$ multiplied with the system function $\mathcal{H}(z)$.

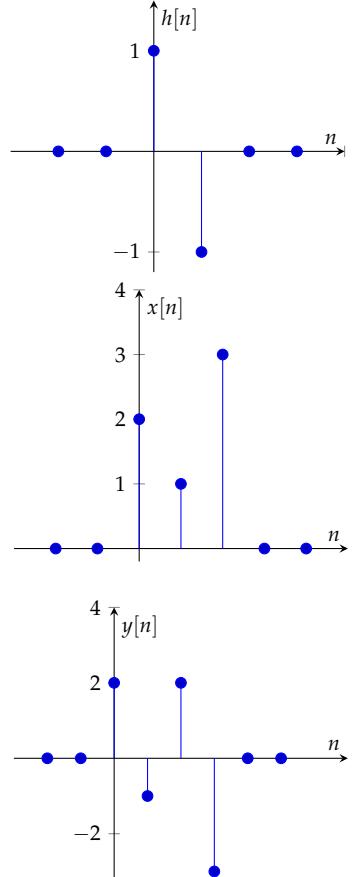


Figure 22.10: A convolution of signals $h[n]$ (top) and $x[n]$ (middle) is $y[n] = x[n] * h[n]$ (bottom).

We rely on the time-shift theorem to obtain the z -transform of the delayed signal $b[n - k]$. \square

Example: convolution in z-domain

Let $h[n] = \delta[n] - \delta[n - 1]$ and $x[n] = 2\delta[n] + \delta[n - 1] + 3\delta[n - 2]$.

These signals are shown in Figure 22.10.

In this case, $\mathcal{H}(z) = 1 - z^{-1}$ and $X(z) = 2 + z^{-1} + 3z^{-2}$. What is $y[n] = h[n] * x[n]$? We can of course directly solve this in time domain, but we can often more efficiently solve this in frequency domain.

We can obtain the result of the convolution using polynomial calculations and z-transforms. The z-transform of $y[n]$ is:

$$Y(z) = \mathcal{H}(z)X(z) \quad (22.58)$$

$$= (1 - z^{-1})(2 + z^{-1} + 3z^{-2}) \quad (22.59)$$

$$= 2 - z^{-1} + 2z^{-2} - 3z^{-3} \quad (22.60)$$

$$= y[0]z^0 + y[1]z^{-1} + y[2]z^{-2} + y[3]z^{-3}. \quad (22.61)$$

Since $Y(z) = \sum_n y[n]z^{-n}$, it follows that:

$$y[n] = 2\delta[n] - \delta[n - 1] + 2\delta[n - 2] - 3\delta[n - 3]. \quad (22.62)$$

The signal $y[n]$ is shown on the bottom panel of Figure 22.10.

Example: Two filter cascade

When two LTI systems are cascaded as shown in Figure 22.11, the system is mathematically described by

$$w[n] = \sum_{\ell} h_1[\ell]x[n - \ell] = h_1[n] * x[n] \quad (22.63)$$

and

$$y[n] = \sum_k h_2[k]w[n - k] \quad (22.64)$$

$$= h_2[n] * w[n] \quad (22.65)$$

$$= h_2[n] * (h_1[n] * x[n]). \quad (22.66)$$

Because convolution is associative, $a * (b * c) = (a * b) * c$, the two convolutions can be combined as follows:

$$y[n] = h_2[n] * (h_1[n] * x[n]) \quad (22.67)$$

$$= \underbrace{(h_1[n] * h_2[n])}_{h[n]} * x[n]. \quad (22.68)$$

Which is equivalent to a single convolution by a combined LTI system with an impulse response that is a convolution of two impulse responses $h[n] = h_1[n] * h_2[n]$. In z-domain, the equivalent system function that combines the two systems is:

$$h[n] = h_1[n] * h_2[n] \xrightarrow{\mathcal{Z}} \mathcal{H}(z) = \mathcal{H}_1(z)\mathcal{H}_2(z), \quad (22.69)$$

and the z-transform of the output $y[n]$ is:

$$Y(z) = \mathcal{H}_1(z)\mathcal{H}_2(z)X(z) = \mathcal{H}(z)X(z). \quad (22.70)$$

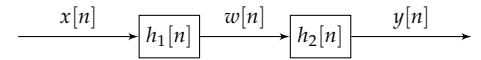
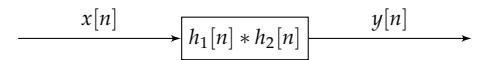


Figure 22.11: A system that consists of two LTI systems connected in series (cascade).



FIR filters as cascaded first order systems

It is possible to view an arbitrary finite impulse response filter as a cascade of first order filters.

Let's assume that all the non-zero elements of the impulse response of a FIR filter lie in the range $n \in \{0, \dots, N\}$. If this is not the case, we can always shift the impulse response in time so that the first non-zero element of the impulse response is at sample index $n = 0$ (time-shift theorem).

The impulse response is then:

$$h[n] = \sum_{k=0}^N b_k \delta[n - k]. \quad (22.71)$$

And the z -transform is the following polynomial fraction:

$$\mathcal{H}(z) = b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N} \quad (22.72)$$

$$= z^{-N} (b_0 z^N + b_1 z^{N-1} + b_2 z^{N-2} + \dots + b_N) \quad (22.73)$$

The *fundamental theorem of algebra* states that a polynomial of degree N has N roots when counting with multiplicity. This means that we can represent the polynomial on the right-hand side of $\mathcal{H}(z)$ in the following form:

$$\mathcal{H}(z) = z^{-N} \prod_{k=1}^N (z - \alpha_k) \quad (22.74)$$

$$= \prod_{k=1}^N (1 - \alpha_k z^{-1}) \quad (22.75)$$

$$= \prod_{k=1}^N \mathcal{H}_k(z). \quad (22.76)$$

In the second line, we have distributed z^{-N} into the product. Here α_k is a root of the N th order polynomial $\mathcal{H}(\alpha_k) = 0$.

We can see that the FIR filter is a product of elementary polynomials:

$$\mathcal{H}_k(z) = 1 - \alpha_k z^{-1}. \quad (22.77)$$

The convolution theorem says that convolution in time domain is equivalent to multiplication in frequency domain. This means that $\mathcal{H}(z)$ is the result of a convolution of N first order system functions $\mathcal{H}_k(z) = 1 - \alpha_k z^{-1}$.

We can use the inverse z -transform on these systems to obtain the impulse response of each first order filter:

$$h_k[n] = \mathcal{Z}^{-1}\{\mathcal{H}_k(z)\} = \delta[n] - \alpha_k \delta[n - 1] \quad (22.78)$$

The full impulse response $h[n]$ is the result of these first order filters convolved together:

$$h[n] = h_1[n] * h_2[n] * h_3[n] * \dots * h_N[n]. \quad (22.79)$$

This means that an FIR filter of arbitrary length can be decomposed into elementary first order FIR filters. It is often easy to analytically study how the individual first order filters behave (e.g., which signals each filter attenuates).

Zeros of the system function of an FIR filter

The system function of an FIR filter $\mathcal{H}(z)$ is a polynomial. The *zeros* of the system function $\mathcal{H}(z)$ are values $z = \alpha_k$, where the system function evaluates to zero: $\mathcal{H}(\alpha_k) = 0$.

The location of the zeros determine the frequency domain behavior of an FIR system. This allows us to use analysis of complex-valued polynomials to investigate the frequency domain behavior of LTI systems.

A zero of the system function means that the LTI system will completely eliminate a signal of the form $x[n] = \alpha_k^n$, i.e., the amplitude of the output of the LTI system will be $|\mathcal{T}\{\alpha_k^n\}| = 0$.

Example: First order FIR filter

Consider the first order LTI system:

$$y[n] = x[n] - x[n - 1] \quad (22.80)$$

This has an impulse response:

$$h[n] = \delta[n] - \delta[n - 1]. \quad (22.81)$$

And the system function is a first order polynomial:

$$\mathcal{H}(z) = 1 - z^{-1}. \quad (22.82)$$

This is a first order polynomial with one zero at $z^{-1} = 1$. In other words, $\alpha_1 = 1$.

This means that if we feed in a constant (DC) signal $x[n] = \alpha_1^n = 1$, the system will completely filter out this signal. This makes sense, as this system is a simple high-pass filter, which we have already studied before in the discrete-time Fourier transform chapter.

Example: Third order FIR filter

Let's look at a more complicated system. Consider the following system:

$$y[n] = x[n] - 2x[n - 1] + 2x[n - 2] - x[n - 3]. \quad (22.83)$$

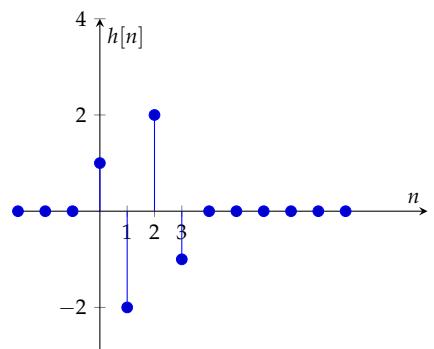


Figure 22.12: A finite impulse response $h[n]$ of a third order system.

This is an FIR filter with the following impulse response:

$$h[n] = \delta[n] - 2\delta[n-1] + 2\delta[n-2] - \delta[n-3]. \quad (22.84)$$

This has a z -transform of the form

$$\mathcal{H}(z) = 1 - 2z^{-1} + 2z^{-2} - z^{-3}. \quad (22.85)$$

This is a third order polynomial. Based on the fundamental theorem of algebra, this polynomial has three zeros. These zeros are at $\alpha_1 = 1$, $\alpha_2 = e^{i\frac{\pi}{3}}$, and $\alpha_3 = e^{-i\frac{\pi}{3}}$. You can find this using one of the many solution strategies for finding the roots of a polynomial⁴ Figure 22.13 shows the locations of these zeros, marked with blue circles, on the complex plane.

With the help of the roots of the polynomial, we can now write the polynomial in the following form:

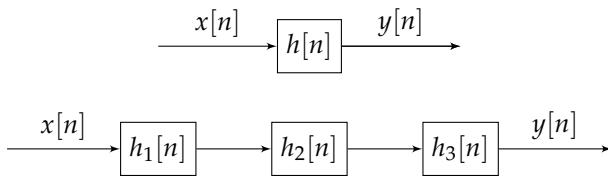
$$\mathcal{H}(z) = (1 - z^{-1})(1 - e^{i\frac{\pi}{3}}z^{-1})(1 - e^{-i\frac{\pi}{3}}z^{-1}). \quad (22.86)$$

This can also be viewed as a cascade of three different first order FIR filters:

$$\mathcal{H}(z) = \mathcal{H}_1(z)\mathcal{H}_2(z)\mathcal{H}_3(z) \quad (22.87)$$

$$= (1 - z^{-1})(1 - e^{i\frac{\pi}{3}}z^{-1})(1 - e^{-i\frac{\pi}{3}}z^{-1}) \quad (22.88)$$

Using the z -transform, we can find the impulse responses of the individual filters: $h_1[n] = \delta[n] - \delta[n-1]$, $h_2[n] = \delta[n] - e^{i\frac{\pi}{3}}\delta[n-1]$, $h_3[n] = \delta[n] - e^{-i\frac{\pi}{3}}\delta[n-1]$. The cascaded filter equivalent to $h[n]$ is shown below.



The first filter $h_1[n]$ will completely remove a signal of the form $x[n] = \alpha_1^n = 1$. The second will completely remove signals of the form $x[n] = \alpha_2^n = e^{i\frac{\pi}{3}n}$, and the third filter will remove signals of the form $x[n] = \alpha_3^n = e^{-i\frac{\pi}{3}n}$. This filter completely removes three spectral components of the input signal with normalized angular frequencies $\hat{\omega}_1 = 0$, $\hat{\omega}_2 = \frac{\pi}{3}$, and $\hat{\omega}_3 = -\frac{\pi}{3}$.

Plotting $\mathcal{H}(\hat{\omega})$ on the complex plane

In order to investigate what the system function $\mathcal{H}(z)$ looks like, I've written some Python code to plot the magnitude and phase of the

⁴ It may be helpful to convert the polynomial into a form with positive exponents: $\mathcal{H}(z) = z^{-3}(z^3 - 2z^2 + 2z^1 - 1)$ and to solve for $z^3 - 2z^2 + 2z^1 - 1 = 0$ using a symbolic calculator such as Wolfram Alpha. You don't need to make it a priority to learn how to solve higher order polynomials using pencil and paper. We have computers for that.

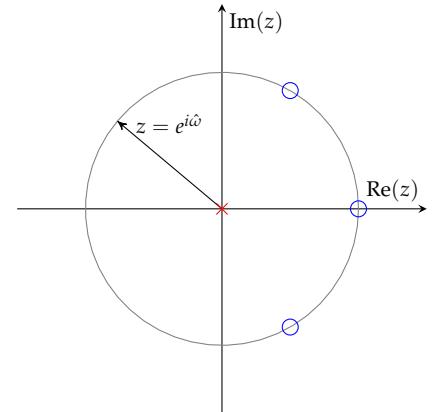


Figure 22.13: Locations of the three zeros ($\alpha_k = \{e^{i\frac{\pi}{3}}, e^{-i\frac{\pi}{3}}, 1\}$) of $\mathcal{H}(z)$ in the complex plane are marked with blue circles. In this case, they all coincide with the unit circle.

system function $\mathcal{H}(z)$ on the complex plane for various values of z . This code is shown in Listing 22.1. The plot is shown in Figure 22.14.

From this plot, one can see that $|\mathcal{H}(z)|$ obtains very low values near the locations of the zeros of $\mathcal{H}(z)$. This means that signals with normalized frequencies near $\hat{\omega}_1 = 0$, $\hat{\omega}_2 = \frac{\pi}{3}$, and $\hat{\omega}_3 = -\frac{\pi}{3}$ are also reduced in amplitude as a result of the filtering operation.

The magnitude of $\mathcal{H}(z)$ is larger than unity on the left-hand side of the plot. This means that complex sinusoidal signals with sufficiently high normalized angular frequencies are amplified by the system.

```
import matplotlib.pyplot as plt
import numpy as np

# Define system function.
def h(z):
    return ((1-z**(-1))*(1-np.exp(1j*np.pi/3.0)*z**(-1))*(1-np.
    exp(-1j*(np.pi/3)*z**(-1)))

xx = np.linspace(-3, 3, num=1000)
yy = np.linspace(-3, 3, num=1000)
x, y = np.meshgrid(xx, yy)

fig, axs = plt.subplots(2, 1, figsize=(3, 6))

# Plot magnitude of system function.
c = axs[0].pcolormesh(xx, yy, 10.0*np.log10(np.abs(h(x+1j*y)))
    **2.0), vmin=-20, vmax=20, cmap="jet")
fig.colorbar(c, ax=axs[0])
om = np.linspace(0, 2.0*np.pi, num=1000)
axs[0].plot(np.cos(om), np.sin(om), color="white")
axs[0].set_aspect('equal')
axs[0].set_ylabel("Imaginary part of z")
axs[0].set_xlabel("Real part of z")
axs[0].set_xlim([-1.5, 1.5])
axs[0].set_ylim([-1.5, 1.5])
axs[0].set_title("$|\mathcal{H}(z)|^2$ (dB)")

# Plot phase angle of system function.
c = axs[1].pcolormesh(xx, yy, np.angle(h(x+1j*y)), cmap="jet")
fig.colorbar(c, ax=axs[1])
axs[1].set_aspect('equal')
axs[1].set_ylabel("Imaginary part of z")
axs[1].set_xlabel("Real part of z")
axs[1].set_xlim([-1.5, 1.5])
axs[1].set_ylim([-1.5, 1.5])

axs[1].set_title("$\angle\mathcal{H}(z)$")

om = np.linspace(0, 2.0*np.pi, num=1000)
axs[1].plot(np.cos(om), np.sin(om), color="white")
plt.tight_layout()
plt.savefig("z_mag_angle.png")
plt.show()
```

Listing 22.1: 025_system_function/third_order.py

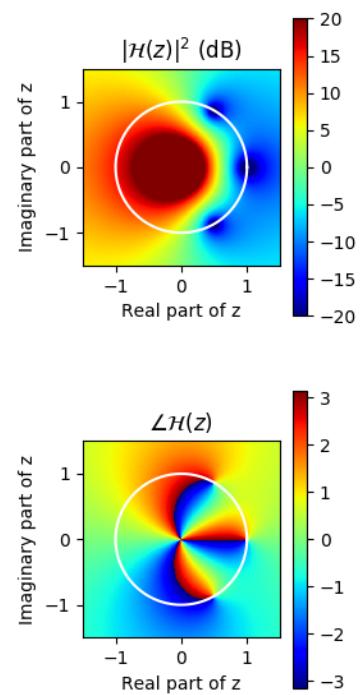


Figure 22.14: The magnitude and phase angle of the system function plotted on the complex plane. The unit circle is shown in white.

Frequency response $\mathcal{H}(\hat{\omega})$

In order to obtain the frequency response, evaluate $\mathcal{H}(z)$ on the unit circle $z = e^{i\hat{\omega}}$ for $\hat{\omega} \in [-\pi, \pi]$:

$$\mathcal{H}(\hat{\omega}) = (1 - e^{-i\hat{\omega}})(1 - e^{i(\frac{\pi}{3}-\hat{\omega})})(1 - e^{-i(\frac{\pi}{3}+\hat{\omega})}) \quad (22.89)$$

The magnitude response $|\mathcal{H}(\hat{\omega})|$ is shown in Figure 22.15. This is a high-pass filter. Compare $|\mathcal{H}(\hat{\omega})|$ with the values of $\mathcal{H}(z)$ on the unit circle, shown in the top panel of Figure 22.14.

The zeros of the system function, that lie on the unit circle $z = e^{i\hat{\omega}}$, correspond to the frequencies at which the gain of the system is zero. Thus, complex sinusoids at these frequencies are blocked.

Blocking filters

The factorization of a filter can be used to design a filter that blocks certain frequencies. A filter of the form:

$$\mathcal{H}(z) = z^{-n_0} \prod_{k=1}^N (1 - \alpha_k z^{-1}) \quad (22.90)$$

blocks all signals of form α_k^n . By selecting values of $\alpha_k = e^{i\hat{\omega}_k}$, we can make a filter that blocks complex sinusoidal signals $x[n] = e^{i\hat{\omega}_k n}$ with frequencies $\hat{\omega}_k$, as we saw in the previous example. The term z^{-n_0} is an arbitrary time shift for this filter, which doesn't affect the locations of the zeros of the system function.

Example: blocking filter for a real-valued sinusoidal signal

Let's design a filter that blocks a 50 Hz power line signal. We'll assume that this signal is sinusoidal:

$$x[n] = \cos(\hat{\omega}_0 n) = \frac{1}{2}(e^{i\hat{\omega}_0 n} + e^{-i\hat{\omega}_0 n}) \quad (22.91)$$

If our sampling rate is $f_s = 10^3$ Hz, a 50 Hz frequency would correspond to a normalized angular frequency of $\hat{\omega}_0 = 2\pi 50/f_s = 0.1\pi$ radians per sample.

Because our signal has two spectral components, we need to filter out normalized angular frequencies $\hat{\omega} = \pm 0.1\pi$. This would correspond to zeros of the system function at $\alpha_1 = e^{i0.1\pi}$ and $\alpha_2 = e^{-i0.1\pi}$. We now have a filter description:

$$\mathcal{H}(z) = (1 - e^{-i0.1\pi} z^{-1})(1 - e^{i0.1\pi} z^{-1}) \quad (22.92)$$

$$= 1 - (e^{-i0.1\pi} + e^{i0.1\pi})z^{-1} + (e^{-i0.1\pi} e^{i0.1\pi})z^{-2} \quad (22.93)$$

$$= 1 - 2\cos(0.1\pi)z^{-1} + z^{-2} \quad (22.94)$$

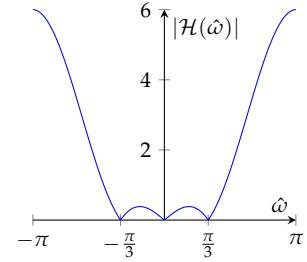


Figure 22.15: The magnitude response $|\mathcal{H}(\hat{\omega})|$ for the third order FIR filter.

This filter has an impulse response:

$$h[n] = \delta[n] - 2 \cos(0.1\pi)\delta[n-1] + \delta[n-2] \quad (22.95)$$

and system function description:

$$y[n] = x[n] - 2 \cos(0.1\pi)x[n-1] + x[n-2]. \quad (22.96)$$

This is a simple filter that will completely remove a pure 50 Hz sinusoidal signal. You can expand on this to, e.g., build a filter that blocks harmonics of 50 Hz.

Running average filter

In the frequency response chapter, we analyzed the frequency response of the running average filter. Now we will analyze the z-transform of the same filter.

This running average filter with $M = 2N + 1$ is defined using the following system definition:

$$y[n] = \frac{1}{M} \sum_{k=-N}^N x[n-k]. \quad (22.97)$$

This system has the following impulse response:

$$h[n] = \frac{1}{M} \sum_{k=-N}^N \delta[n-k]. \quad (22.98)$$

The z-transform of this filter is:

$$\mathcal{H}(z) = \sum_{k=-\infty}^{\infty} h[k]z^{-k}, \quad (22.99)$$

$$= \frac{1}{M} \sum_{k=-N}^N z^{-k}. \quad (22.100)$$

This is a *geometric series*, which we can solve by our usual method.

We obtain:

$$\mathcal{H}(z) = \frac{1}{M} \frac{z^N - z^{-(N+1)}}{1 - z^{-1}}. \quad (22.101)$$

Multiplying the right-hand side with z^{N+1}/z^{N+1} gives:

$$\mathcal{H}(z) = \frac{1}{M} \frac{z^{2N+1} - 1}{z^N(z-1)} \quad (22.102)$$

$$= \frac{1}{M} \frac{z^M - 1}{z^N(z-1)} \quad (22.103)$$

The system function is zero when $z^M - 1 = 0$, or $z^M = 1$. Since $1 = e^{i2\pi k}$ for $k \in \mathbb{Z}$, it follows that the roots of the Mth order polynomial are $\alpha_k = e^{i\frac{2\pi}{M}k}$. This has M unique values when $k \in [0, 1, 2, \dots, M-1]$.

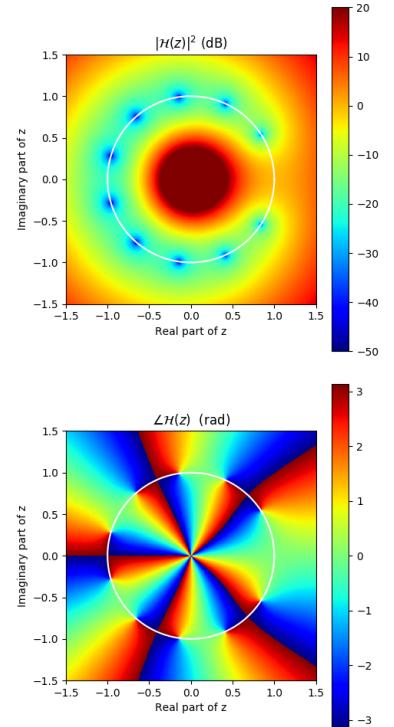


Figure 22.16: Above: The running average system function magnitude $|\mathcal{H}(z)|$ plotted on the complex plane. The $M - 1$ zeros of the system function are evenly distributed $\alpha_k = e^{i\frac{2\pi}{M}k}$ with $k \in 1, 2, \dots, M - 1$. These are the frequencies that are perfectly removed by the filter. The unit circle $e^{i\omega}$ is denoted with a white line. Below: The phase angle of the system function corresponding to a running average filter.

We can now express the system function in factorized form:

$$\mathcal{H}(z) = \frac{1}{M} \frac{1}{(z-1)z^N} \prod_{k=0}^{M-1} (z - e^{i\frac{2\pi}{M}k}) \quad (22.104)$$

$$= \frac{1}{M} \cancel{\frac{1}{(z-1)z^N}} \cancel{(z-1)} \prod_{k=1}^{M-1} (z - e^{i\frac{2\pi}{M}k}) \quad (22.105)$$

$$= \frac{1}{M} \frac{1}{z^N} \prod_{k=1}^{M-1} (z - e^{i\frac{2\pi}{M}k}) \quad (22.106)$$

Notice that one of the zeros (for $k = 0$) of the denominator polynomial coincides with a zero of the numerator. This means that this root of the system function polynomial cancels out.

Finally, we multiply with $z^{-(M-1)}/z^{-(M-1)}$ to obtain:

$$\mathcal{H}(z) = \frac{1}{M} z^N \prod_{k=1}^{M-1} (1 - e^{i\frac{2\pi}{M}k} z^{-1}). \quad (22.107)$$

We can see that this is the same form as Equation 22.90. This means that the running average filter is actually a set of blocking filters which blocks complex sinusoidal signals $x[n] = e^{i\frac{2\pi}{M}kn}$ with normalized angular frequencies $\hat{\omega}_k = \frac{2\pi}{M}k$ with $k = [1, 2, \dots, M-1]$. The term z^N means that there is a time-shift (negative delay) by N samples.

The magnitude of the running average system function $|\mathcal{H}(z)|$ is shown in Figure 22.16 for $N = 5$ ($M = 2N + 1 = 11$). This system function has 10 nulls at complex-valued positions $\alpha_k = e^{i\frac{2\pi}{11}k}$ with $k \in [1, 2, \dots, 10]$. They are all along the unit circle (shown with a white circle). Only the region near $z = 1$ doesn't have a null. This corresponds to the pass-band region of this filter. Figure 22.16 also shows the phase angle of the same system function.

If we evaluate the value of the system function on the unit circle $z = e^{i\hat{\omega}}$, we obtain the magnitude and phase response of the system. The magnitude and phase response shown in Figure 22.17 looks exactly like the one we obtained earlier using a discrete-time Fourier transform for the same running average filter.

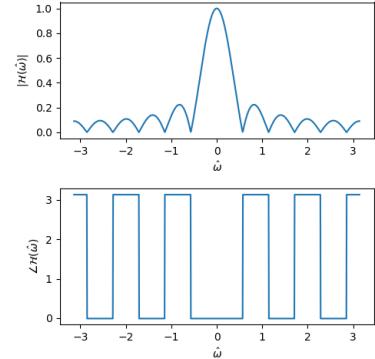


Figure 22.17: The magnitude and phase response of the running average filter. Within the main lobe of the low-pass filter pass band, the filter introduces no phase shift. The phase then alternates between 0 and π within the sidelobes.

Exercises: Z-transform

1. Find the z-transforms for the following signals:

- a) $h[n] = \delta[n]$
- b) $h[n] = 2\delta[n + 4]$
- c) $h[n] = -41\delta[n - 42]$
- d) $h[n] = \delta[n + 1] - 2\delta[n - 1] + 4\delta[n - 4]$

2. The system function $\mathcal{H}(z)$ of a discrete-time LTI system is defined as:

$$\mathcal{H}(z) = z^1 + 1 + 3z^{-1} - 0.5z^{-2} + 4z^{-10} \quad (22.108)$$

- a) What is the difference equation of the form:

$$y[n] = \sum_k \alpha_k x[n - n_k] \quad (22.109)$$

that describes the relationship between the input signal $x[n]$ and the output signal $y[n]$ for the discrete-time LTI system, uniquely determined by Equation 22.108?

- b) Sketch a plot of the impulse response $h[n] = \mathcal{T}\{\delta[n]\}$ of this LTI system.
c) Modify $\mathcal{H}(z)$ in such a way that a delay by 2 samples ($y_2[n] = y[n - 2]$) will be applied to the signal $y[n]$ defined in a).
3. The system function $\mathcal{H}(z)$ of a discrete-time LTI system is defined as:

$$\mathcal{H}(z) = (z - e^{i\pi/4})(z - e^{-i\pi})z^{-2} \quad (22.110)$$

You can view this system as a cascade of three elementary systems:

$$\mathcal{H}(z) = \mathcal{H}_1(z)\mathcal{H}_2(z)\mathcal{H}_3(z) \quad (22.111)$$

Two of these systems will null out a single frequency and one of these systems will apply time-shift to the signal.

- a) Define $\mathcal{H}_1(z)$, $\mathcal{H}_2(z)$, and $\mathcal{H}_3(z)$ and describe what each of these elementary LTI systems does to an input signal. It will be one of two options: 1) null out the frequency component with frequency $\hat{\omega}_0$, or 2) time-shift the signal by n_0 samples.
- b) Sketch the locations of the poles and zeroes of the system function on the complex plane. Include the unit circle on the plot.
- c) Sketch a plot of the magnitude response $|\mathcal{H}(\hat{\omega})|$ of this system. Mark the nulls on the plot.

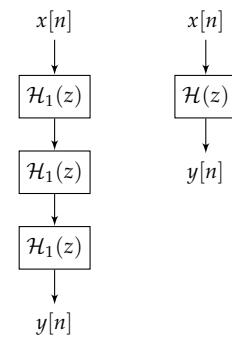


Figure 22.18: A cascade of three elementary LTI systems defined by system functions $\mathcal{H}_1(z)$, $\mathcal{H}_2(z)$ and $\mathcal{H}_3(z)$ is equivalent to a single LTI system $\mathcal{H}(z) = \mathcal{H}_1(z)\mathcal{H}_2(z)\mathcal{H}_3(z)$.

- d) List the frequencies of the frequency components of the input signal that will be completely removed by this system (nulled out). Use radians per sample as the units for frequency.
- e) Write the system function on the form:

$$\mathcal{H}(z) = \sum_k \beta_k z^{-k} \quad (22.112)$$

What are the non-zero values of β_k ?

- f) What is the impulse response $h[n]$ of the LTI system?
- g) If a real-valued signal will be fed into the system, will the output always be real-valued? Justify your answer.
4. You have a real-valued discrete-time signal that is sampled with $f_s = 44.1 \cdot 10^3$ samples per second. Come up with a simple LTI system that will remove a 1500 and 5000 hertz frequency component of the input signal. This means that the magnitude response of the LTI needs to have a null at frequencies -1500, 1500, -5000 and 5000 hertz. Use a suitably selected system function (z-transform of the impulse response) to derive this filter⁵.
- a) What is the system function that you came up with?
 - b) Draw a pole-zero diagram of your system function.
 - c) Write a computer program to verify that the magnitude response has nulls on the correct frequencies. Hint: you can use the code in Listing 22.2 to help you out. An example output of this program is shown in Figure 22.19.
 - d) What is the impulse response of $h[n]$ of your LTI system?

```
import matplotlib.pyplot as plt
import numpy as np

# Frequency, in units of radians per sample.
omhat = np.linspace(-np.pi, np.pi, num=100000)

def h(z):
    # System function.
    omo = 2.0*np.pi*5000.0/(44.1e3)
    om1 = 2.0*np.pi*1500.0/(44.1e3)
    return (z**(-4)*(z**4 - 2*z**3*(np.cos(om1)+np.cos(omo)) +
        2*z**2*(1+2*np.cos(om1)*np.cos(omo)) -
        2*z*(np.cos(omo)+np.cos(om1)) + 1))

# Magnitude response.
magresp = np.abs(h(np.exp(1j*omhat)))
# Plot the magnitude response in dB scale.
plt.plot(omhat, 10.0*np.log10(magresp**2.0))
plt.xlabel(r"Frequency $\hat{\omega}$ (rad/sample)")
plt.ylabel(r"Magnitude response $10\log_{10}(|\mathcal{H}(\hat{\omega})|^2)$ dB")
```

⁵ Remember that you can relate frequency in units of radians per sample to radians per second using $\hat{\omega} = \omega T_s$, where $T_s = 1/f_s$. Also remember that you can relate frequency in units of radians per second to frequency in cycles per second (hertz) using $\omega = 2\pi f$

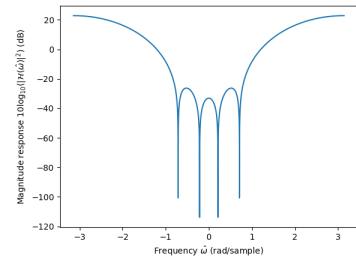


Figure 22.19: Magnitude response of a filter that notches out two frequencies.

```
| plt.show()
```

Listing 22.2: 030_z_fir_magresp/plot_mag_resp.py

Suggested solutions: Z-transform

1. Given a discrete time signal $x[n]$, then the z-transform is defined as:

$$X(z) = \sum_{k=-\infty}^{\infty} x[k]z^{-k}.$$

a) Using the definition, we have:

$$X(z) = \mathcal{Z}\{\delta[n]\} = \sum_{k=-\infty}^{\infty} \delta[k]z^{-k} = \delta[0]z^{-0} = 1,$$

since the impulse function is only non-zero for $n = 0$, for which the value is 1.

b) The z-transform is linear and for a time-shift, we then have:

$$\begin{aligned} X(z) &= \mathcal{Z}\{2\delta[n+4]\} = 2\mathcal{Z}\{\delta[n+4]\}, \\ &= 2 \sum_{k=-\infty}^{\infty} \delta[k+4]z^{-k}, \\ &= 2z^4 \mathcal{Z}\{\delta[n]\}, \\ &= 2z^4, \end{aligned}$$

since by the previous exercise $\mathcal{Z}\{\delta[n]\} = 1$.

- c) The z-transform of $h[n] = -41\delta[n-42]$ is done in the same way as the previous problem. Skipping the details, we obtain:

$$X(z) = \mathcal{Z}\{-41\delta[n-42]\} = -41z^{-42},$$

again by linearity and the time-shift theorem.

- d) For $h[n] = \delta[n+1] - 2\delta[n-1] + 4\delta[n-4]$ we obtain again by linearity and the time-shift property:

$$X(z) = \mathcal{Z}\{h[n]\} = z - 2z^{-1} + 4z^{-4}.$$

2. Let the system function for an LTI system be $\mathcal{H}(z) = z^1 + 1 + 3z^{-1} - 0.5z^{-2} + 4z^{-10}$.

- a) If the system function is defined as above, then the impulse response is found by taking the inverse z-transform of $\mathcal{H}(z)$. We get:

$$h[n] = \delta[n+1] + \delta[n] + 3\delta[n-1] - 0.5\delta[n-2] + 4\delta[n-10],$$

from this we conclude that the difference equation for the system is:

$$y[n] = x[n+1] + x[n] + 3x[n-1] - 0.5x[n-2] + 4x[n-10],$$

since every LTI system can be expressed as a convolution of the form:

$$y[n] = h[n] * x[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k],$$

with $h[n] = \mathcal{T}\{\delta[n]\}$.

- b) In Listing 22.3 is a simple program to plot the impulse response for this system.

```
import matplotlib.pyplot as plt
import numpy as np

# Sample values we want to plot on.
nn = np.array([-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Values for the impulse response function.
h = np.array([0, 1, 1, 3, -0.5, 0, 0, 0, 0, 0, 0, 0, 4])

# Plot as a stem plot to emphasize the
# discrete nature of the function.
plt.stem(nn, h)
plt.xlabel("samples [n]")
plt.ylabel("$h[n]$")
plt.title("Impulse response")
# Call this if needed:
# plt.show()
```

Listing 22.3: Code to plot the impulse response function

The impulse response plot is shown in Figure 22.20.

- c) If we define a new signal $y_2[n] = y[n-2]$, then by the time-shift theorem we have that the new system function will be:

$$Y_2(z) = z^{-2}Y(z) = z^{-2}\mathcal{H}(z)X(z)$$

which gives:

$$\mathcal{H}_2(z) = z^{-2}\mathcal{H}(z) = z^{-1} + z^{-2} + 3z^{-3} - 0.5z^{-4} + 4z^{-12},$$

this new system function will describe a system that is delayed by 2 samples compared to $\mathcal{H}(z)$.

3. Let the system function of a discrete-time LTI system be defined as:

$$\mathcal{H}(z) = (z - e^{i\pi/4})(z - e^{-i\pi})z^{-2}.$$

- a) One can view the system function as a cascade of three systems of the form:

$$\begin{aligned}\mathcal{H}_1(z) &= z - e^{i\pi/4}, \\ \mathcal{H}_2(z) &= z - e^{-i\pi}, \\ \mathcal{H}_3(z) &= z^{-2},\end{aligned}$$

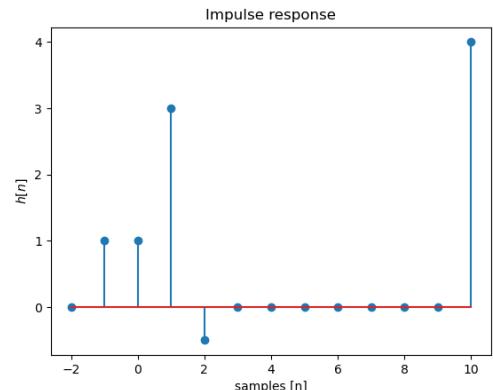


Figure 22.20: Output of Listing 22.3

where the effect is to null-out a frequency for \mathcal{H}_1 and \mathcal{H}_2 , the frequencies being $\pi/4$ and π , respectively. The last elementary system \mathcal{H}_3 will delay by 2 samples. To determine the overall effect of this system, factor it:

$$\mathcal{H}(z) = z(1 - e^{i\pi}z^{-1})z(1 - e^{-i\pi}z^{-1})z^{-2} = (1 - e^{i\pi/4}z^{-1})(1 - e^{-i\pi}z^{-1}).$$

Then we can view the system function as a cascade of two elementary systems of the form:

$$\begin{aligned}\mathcal{H}_1(z) &= 1 - e^{i\pi/4}z^{-1}, \\ \mathcal{H}_2(z) &= 1 - e^{-i\pi}z^{-1}.\end{aligned}$$

This filter blocks sinusoidal signals where $\alpha_1 = e^{i\pi/4}$ and $\alpha_2 = e^{-i\pi}$, which correspond to normalized angular frequencies of $\hat{\omega} = \pi/4$ and $\hat{\omega} = \pi$.

- b) The poles and zeros of the system function are shown in Figure 22.21.
- c) The following code snippet will plot the magnitude response corresponding to $\mathcal{H}(\hat{\omega}) = \mathcal{H}(z)|_{z=e^{j\hat{\omega}}}$:

```
import matplotlib.pyplot as plt
import numpy as np

# Partition the interval (-pi, pi).
om = np.linspace(-np.pi, np.pi, num=100)

def system_function(z):
    return (1 - np.exp(1j * np.pi / 4) * z**(-1)) * (1 - np.exp(-1j * np.pi) * z**(-1))

def frequency_response(om):
    return system_function(np.exp(1j * om))

# Marking the zeros on the graph.
x_markers = [np.pi/4, -np.pi]
y_markers = [np.abs(frequency_response(np.pi/4)), np.abs(frequency_response(-np.pi))]

# Plot the system function with the zeros marked.
plt.plot(om, np.abs(frequency_response(om)))
plt.plot(x_markers, y_markers, "rx", label="zeros")
plt.xlabel(r"\$\hat{\omega}\$ (rad / sample)")
plt.ylabel(r"\$|\mathcal{H}(\hat{\omega})|\$")
plt.title("Magnitude response plot")
plt.legend()
# Call this if needed.
# plt.show()
```

Listing 22.4: Script to plot the magnitude response

Output from Listing 22.4 is shown in Figure 22.22.

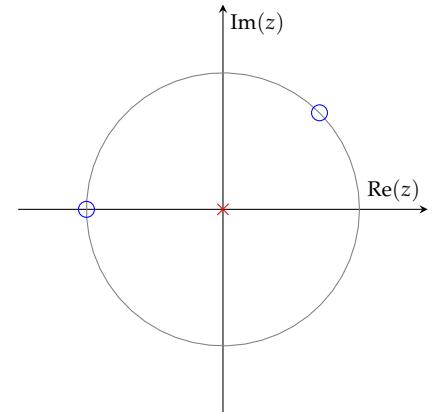


Figure 22.21: The zeros of the system function $\mathcal{H}(z)$, have $\alpha_k = \{e^{i\pi/4}, e^{-i\pi}\}$. Zeros are marked with blue circles and poles are marked with red crosses.

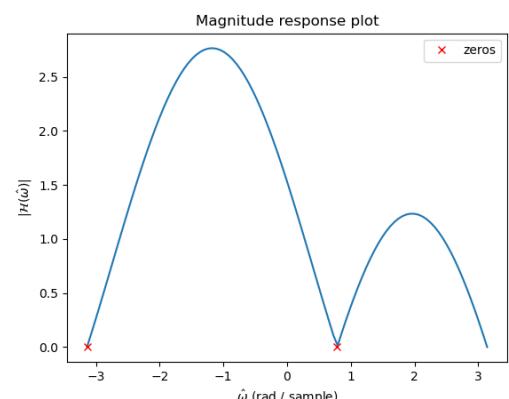


Figure 22.22: Plot of $|\mathcal{H}(\hat{\omega})|$, which has zeros at $\hat{\omega} = \pi/4$ and $\hat{\omega} = -\pi$.

- d) The zeros of the magnitude response will get completely removed by the system. The original system has a system function $\mathcal{H}(z)$ which has two zeros, these being $\alpha_1 = e^{i\pi/4}$ and $\alpha_2 = e^{-i\pi}$. These correspond to $\hat{\omega} = \pi/4$ and $\hat{\omega} = -\pi$ in units of radians per sample. These discrete-time angular frequencies will be completely removed by the system.
- e) From the original definition of the system, we have:

$$\begin{aligned}\mathcal{H}(z) &= (z - e^{i\pi/4})(z - e^{-i\pi})z^{-2}, \\ &= 1 - e^{i\pi/4}z^{-1} - e^{-i\pi}z^{-1} + e^{-i3\pi/4}z^{-2}, \\ &= 1 - (e^{i\pi/4} + e^{-i\pi})z^{-1} + (e^{-i3\pi/4})z^{-2}\end{aligned}$$

The non-zero values for β_k are $\beta_1 = \alpha_1 + \alpha_2$ and $\beta_2 = \alpha_1\alpha_2$.

- f) To find the impulse response, we find the inverse z-transform of the system function. The impulse response $h[n]$ is then:

$$h[n] = \delta[n] - (e^{i\pi/4} + e^{-i\pi})\delta[n-1] + e^{-i3\pi/4}\delta[n-2].$$

- g) For any discrete-time LTI system, we have:

$$y[n] = h[n] * x[n].$$

If the impulse response is real-valued, then the output signal will also be real-valued. If $h[n]$ is real-valued, it must satisfy: $h[n]^* = h[n]$. We can easily see that the impulse response doesn't satisfy this property, therefore a real-valued signal is not necessarily real-valued after applying this system. This can also be seen from the pole-zero diagram. The positive and negative frequencies are not treated the same, as the frequency $\hat{\omega} = \pi/4$ gets filtered out, while $\hat{\omega} = -\pi/4$ passed through.

4. Let $f_s = 44.1 \cdot 10^3$ Hz.

- a) To filter out the frequencies, we require that the system function has four zeros. These being at frequencies corresponding to ± 5000 and ± 1500 Hz. Thus, the system function will be a polynomial of degree four with four distinct roots. That is, the system function looks like:

$$\mathcal{H}(z) = (z - \alpha_1)(z - \alpha_2)(z - \alpha_3)(z - \alpha_4)$$

The frequencies in hertz corresponds to discrete-time angular frequencies of $\hat{\omega}_k = \omega T_s = 2\pi f_k / f_s$. This gives:

$$\begin{aligned}\hat{\omega}_0 &= 2\pi 5000 / 44100 = 100\pi / 441, \\ \hat{\omega}_1 &= -2\pi 5000 / 44100 = -100\pi / 441, \\ \hat{\omega}_2 &= 2\pi 1500 / 44100 = 10\pi / 147, \\ \hat{\omega}_3 &= -2\pi 1500 / 44100 = -10\pi / 147.\end{aligned}$$

Finally, the system function is therefore of the form:

$$\begin{aligned}\mathcal{H}(z) &= (z - e^{10i\pi/441})(z - e^{-10i\pi/441})(z - e^{10i\pi/147})(z - e^{-10i\pi/147}), \\ &= (1 - e^{\frac{10\pi}{441}i}z^{-1})(1 - e^{-\frac{10\pi}{441}i}z^{-1})(1 - e^{\frac{10\pi}{147}i}z^{-1})(1 - e^{-\frac{10\pi}{147}i}z^{-1})z^4, \\ &= (1 - 2\cos\left(\frac{10\pi}{441}\right)z^{-1} + z^{-2})(1 - 2\cos\left(\frac{10\pi}{147}\right)z^{-1} + z^{-2})z^4.\end{aligned}$$

- b) A pole-zero diagram for the system function $\mathcal{H}(z)$ is shown in Figure 22.23.
- c) Listing 22.5 provides code to plot the magnitude response for this system.

```
import matplotlib.pyplot as plt
import numpy as np

# Function to convert to dB.
def convert_to_decibel(x):
    return 10*np.log10(np.abs(x)**2)

# Partition the interval (-pi, pi).
om = np.linspace(-np.pi, np.pi, num=10000)

def system_function(z):
    return (1 - np.exp(1j*100*np.pi/441)*z**(-1))*(1 - np.exp(-1j*100*np.pi/441)*z**(-1))*(1 - np.exp(20j*np.pi/441)*z**(-1))*(1 - np.exp(-20j*np.pi/441)*z**(-1))*z**4

def frequency_response(om):
    return system_function(np.exp(1j*om))

# Plot the system function with the zeros marked.
plt.plot(om, convert_to_decibel(frequency_response(om)),
         label=r"$|\mathcal{H}(\hat{\omega})|$")
plt.xlabel(r"$\hat{\omega}$ (rad / sample)")
plt.ylabel(r"$|\mathcal{H}(\hat{\omega})|$")
plt.title("Magnitude response plot")
plt.legend()
# Call this if needed:
# plt.show()
```

Listing 22.5: Code to plot the magnitude response

Output of Listing 22.5 is shown in Figure 22.24.

- d) Having found the system function, one can easily determine the impulse response function by using the inverse z -transform. Have that:

$$\mathcal{H}(z) = (1 - 2\cos\left(\frac{10\pi}{441}\right)z^{-1} + z^{-2})(1 - 2\cos\left(\frac{10\pi}{147}\right)z^{-1} + z^{-2})z^4.$$

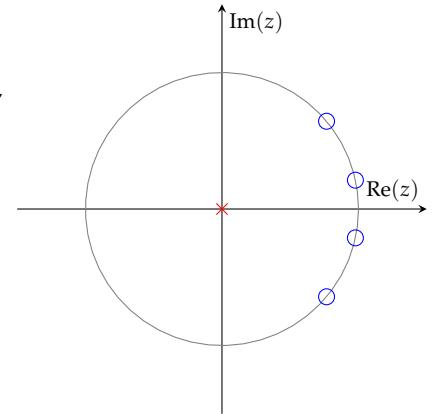


Figure 22.23: The zeros of the system function $\mathcal{H}(z)$, have $\alpha_k = \{e^{10i\pi/441}, e^{-10i\pi/441}, e^{20i\pi/441}, e^{-20i\pi/441}\}$. Zeros are marked with blue circles and poles are marked with red crosses.

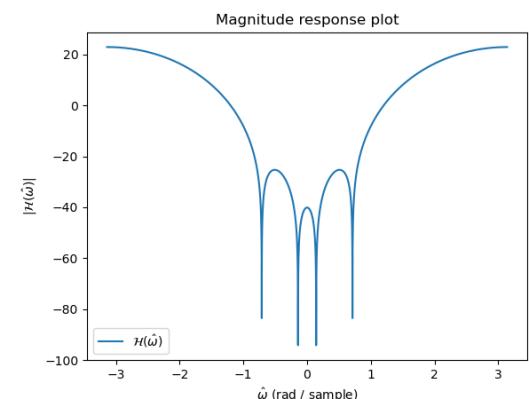


Figure 22.24: Plot of the magnitude response in dB

Expanding all of this out, one obtains:

$$\mathcal{H}(z) = z^4 - 2(\cos(\theta_1) + \cos(\theta_2))z^3 + 2(2\cos\theta_1\cos\theta_2 + 1)z^2 - 2(\cos\theta_1 + \cos\theta_2)z + 1,$$

where $\theta_1 = \frac{10\pi}{441}$ and $\theta_2 = \frac{10\pi}{147}$. The impulse response is then the inverse z -transform:

$$h[n] = \delta[n] - 2(\cos\theta_1 + \cos\theta_2)\delta[n+1] + 2(2\cos\theta_1\cos\theta_2 + 1)\delta[n+2] - 2(\cos\theta_1 + \cos\theta_2)\delta[n+3] + \delta[n+4].$$

23

Infinite Impulse Response Filters

In this chapter, we will introduce infinite impulse response (IIR) filters. Just like finite impulse response filters (FIR), they are also linear time-invariant systems. The main difference is that infinite impulse response filters include feedback from the samples of the output signal $y[n]$ of the system, and as the name already implies, this feedback makes the impulse response of the system infinitely long.

The advantage of IIR filters compared to FIR filters is that it is often possible to design a lower order filter, which meets design requirements for a frequency response. The lower the order of the filter, the fewer computations per output sample are needed. This is important in some digital signal processing applications, especially ones where the sampling rate of the signal is very large and there are strict requirements for real-time processing speeds.

A drawback of IIR filters is that the feedback terms can result in an unstable filter output. These types of filters are also often not as accurate numerically as FIR filters.

Infinite impulse response filter

Consider the following system:

$$y[n] = \underbrace{\sum_{\ell=1}^N a_\ell y[n - \ell]}_{\text{feedback terms}} + \underbrace{\sum_{k=0}^M b_k x[n - k]}_{\text{feed forward terms}} \quad (23.1)$$

The filter output depends on the current and past values of the input signal, as well as past values of the *system output* $y[n]$. This is a causal system¹, because the system output does not depend on future values of the input or output signal.

The system defined in Equation 23.1 is called an infinite impulse response filter. A total of $N + M + 1$ filter coefficients a_ℓ and b_k are

¹ a system is causal if it only depends on present and past values.

needed to specify this filter.

System function

The system function, i.e., the z -transform of the impulse response for an IIR filter, is:

$$\mathcal{H}(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 - \sum_{\ell=1}^N a_\ell z^{-\ell}} \quad (23.2)$$

Proof. Apply the z -transform to the definition of the IIR, and use linearity and time-shift properties:

$$y[n] = \sum_{\ell=1}^N a_\ell y[n-\ell] + \sum_{k=0}^M b_k x[n-k] \quad \Big| \mathcal{Z}\{\cdot\} \quad (23.3)$$

$$Y(z) = \sum_{\ell=1}^N a_\ell z^{-\ell} Y(z) + \sum_{k=0}^M b_k z^{-k} X(z) \quad (23.4)$$

$$\left(1 - \sum_{\ell=1}^N a_\ell z^{-\ell}\right) Y(z) = \left(\sum_{k=0}^M b_k z^{-k}\right) X(z) \quad (23.5)$$

$$Y(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 - \sum_{\ell=1}^N a_\ell z^{-\ell}} X(z) \quad (23.6)$$

$$Y(z) = \mathcal{H}(z)X(z). \quad (23.7)$$

□

The system function of an IIR filter can be represented as a block diagram, as shown in Figure 23.1. In this diagram, a delay by one sample is indicated with z^{-1} , as this corresponds to the z -transform of a time delay by one sample. The triangle in this diagram indicates multiplying the signal with a constant coefficient.

Impulse response

An IIR filter is a linear time-invariant system. This means that the impulse response $h[n]$ completely characterizes this system.

The impulse response of an LTI system is obtained by feeding a unit impulse $\delta[n]$ into the system. Using the system definition for an IIR filter, the impulse response is defined as:

$$h[n] = \sum_{\ell=1}^N a_\ell h[n-\ell] + \sum_{k=0}^M b_k \delta[n-k]. \quad (23.8)$$

The impulse response $h[n]$ must satisfy this difference equation for all values of n .

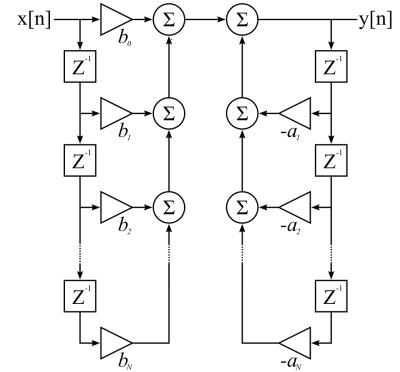


Figure 23.1: A block diagram representation of an infinite impulse response filter system function.

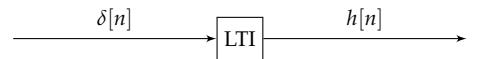


Figure 23.2: The impulse response $h[n]$ of an LTI system is obtained by feeding a unit impulse into the system.

Elementary IIR system

An important type of IIR system is one of the following form:

$$y[n] = a_1 y[n-1] + b_0 x[n] \quad (23.9)$$

In most cases, a more complicated IIR filter can be expressed as a superposition of such elementary filters. Note that Equation 23.9 is the same as the general system described in Equation 23.1 using $N = 1$ and $M = 0$. This system is described using two coefficients a_1 and b_0 .

The system defined in Equation 23.9 has the following impulse response and system function:

$$h[n] = b_0 a_1^n u[n] \xleftrightarrow{z} \mathcal{H}(z) = \frac{b_0}{1 - a_1 z^{-1}}. \quad (23.10)$$

The impulse response $h[n]$ is shown in Figure 23.3.

The derivation is as follows. We start with the impulse response in the form obtainable from Equation 23.9:

$$h[n] = a_1 h[n-1] + b_0 \delta[n]. \quad (23.11)$$

In order to solve this impulse response, we need to know the initial value of the impulse response at $h[-\infty]$. We need to specify this *boundary condition* somehow. We'll assume that the impulse response is zero valued initially at $h[-\infty] = 0$.

It is now possible to calculate the values of the impulse response. The unit impulse arrives at $n = 0$, at which time we get the first non-zero value of the impulse response $h[n]$. It is then possible to evaluate the output recursively using the previous output and find that a pattern emerges:

$$h[-\infty] = 0 \quad (23.12)$$

$$\vdots = 0 \quad (23.13)$$

$$h[-1] = 0 \quad (23.14)$$

$$h[0] = a_1 h[-1] + b_0 = b_0 \quad (23.15)$$

$$h[1] = a_1 h[0] = b_0 a_1 \quad (23.16)$$

$$h[2] = a_1 h[1] = b_0 a_1^2 \quad (23.17)$$

$$h[3] = a_1 h[2] = b_0 a_1^3 \quad (23.18)$$

$$h[4] = a_1 h[3] = b_0 a_1^4 \quad (23.19)$$

$$\vdots = \vdots \quad (23.20)$$

$$h[n] = a_1 h[n-1] = b_0 a_1^n \quad (23.21)$$

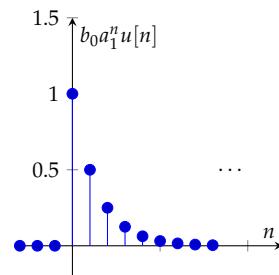


Figure 23.3: The impulse response of an elementary first order IIR system. In this example, $b_0 = 1$ and $a_1 = 0.5$.

The impulse response is therefore:

$$h[n] = b_0 a_1^n u[n] \quad \square, \quad (23.22)$$

where $u[n]$ is the unit step function²

While Equation 23.2 already allows us to determine the system function for this elementary system, it is also possible to directly z -transform the impulse response and obtain the same result. Applying the z -transform on the impulse response gives us:

$$\mathcal{H}(z) = \sum_{n=-\infty}^{\infty} h[n] z^{-n} = \sum_{n=-\infty}^{\infty} b_0 a_1^n u[n] z^{-n} \quad (23.24)$$

$$= b_0 \sum_{n=0}^{\infty} (a_1 z^{-1})^n. \quad (23.25)$$

Using the formula for geometric series, assuming that $|a_1 z^{-1}| < 1$, we obtain³:

$$\mathcal{H}(z) = \frac{b_0}{1 - a_1 z^{-1}} \quad \square. \quad (23.27)$$

This elementary z -transform pair will be used later in analysis of more complicated impulse responses, when applying partial fraction expansion to decompose an N th order IIR system into a superposition of 1st order systems like the one we just derived. One can think of this as an elementary building block for IIR system functions.

Example: IIR system function

Consider an impulse response of the following form:

$$h[n] = (-0.9)^n u[n] - 0.5(-0.9)^{n-1} u[n-1] \quad (23.28)$$

What is the system function $\mathcal{H}(z)$? What is the system description for this system? What is the frequency response of this system? Let's find out.

We can use linearity and the two z -transform pairs that we have learned about. First, a time delay is multiplication by z^{-n_0} :

$$y[n] = x[n - n_0] \xrightarrow{Z} Y(z) = z^{-n_0} X(z), \quad (23.29)$$

and

$$x[n] = b_0 a_1^n u[n] \xrightarrow{Z} X(z) = \frac{b_0}{1 - a_1 z^{-1}}. \quad (23.30)$$

Let's apply this in practice:

$$\mathcal{H}(z) = \frac{1}{1 + 0.9z^{-1}} - z^{-1} \frac{0.5}{1 + 0.9z^{-1}}, \quad (23.31)$$

$$= \frac{1 - 0.5z^{-1}}{1 + 0.9z^{-1}}. \quad (23.32)$$

$$u[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0. \end{cases} \quad (23.23)$$

³ Geometric series closed form solution formula:

$$\sum_{k=0}^L \alpha^k = \frac{1 - \alpha^{L+1}}{1 - \alpha}, \quad (23.26)$$

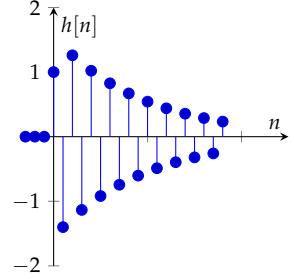


Figure 23.4: The impulse response shown in Equation 23.28.

By comparing the system function in Equation 23.32 with Equations 23.1 and 23.2, we can see that the system is described using the following difference equation:

$$y[n] = -0.9y[n-1] + x[n] - 0.5x[n-1] \quad (23.33)$$

The impulse response is a polynomial fraction. We can multiply this by z/z to obtain:

$$\mathcal{H}(z) = \frac{z - 0.5}{z + 0.9} = \frac{P(z)}{Q(z)}. \quad (23.34)$$

The numerator polynomial $P(z)$ has a zero at $z = 0.5$, and the denominator polynomial $Q(z)$ has a zero at $z = -0.9$. This means that $\mathcal{H}(0.5) = 0$ and $\mathcal{H}(-0.9) \rightarrow \infty$. The zeros of $P(z)$ and $Q(z)$ have the opposite effect on the signal.

Zeros of the denominator polynomial $Q(z)$ are called *poles* of the system function. They indicate signals $\mathcal{T}\{z^n\} \rightarrow \infty$ for which the output of the system grows exponentially.

Figure 23.5 shows the magnitude of the system function $|\mathcal{H}(z)|$ on the complex plane. The pole at $z = -0.9$ is indicated with a white cross, and the zero at $z = 0.5$ is indicated with a white circle. We can see the implication of the zeros and the poles of the system function. The system function has a large magnitude near the poles and low values in the region near the zeros.

A plot like the one shown in Figure 23.5 is known as a *pole-zero diagram*. Typically, such a diagram only depicts the locations of the poles and zeros with crosses and circles. These diagrams are useful for analyzing the properties of discrete-time LTI systems.

We can obtain the frequency response of the system by evaluating $\mathcal{H}(z)$ on the unit circle:

$$\mathcal{H}(\hat{\omega}) = \frac{e^{i\hat{\omega}} - 0.5}{e^{i\hat{\omega}} + 0.9} \quad (23.35)$$

This is shown in Figure 23.6. This filter is a high-pass filter.

Stability for an arbitrary IIR system

Because of the feedback terms in IIR systems, these systems are not necessarily stable. Instability means that the output of the system for some inputs can become indeterminate. This is one of the main drawbacks of IIR systems.

Bounded Input Bounded Output (BIBO) stability is used to determine the stability of an IIR system. BIBO stability implies that for a bounded input:

$$|x[n]| \leq M_x < \infty. \quad (23.36)$$

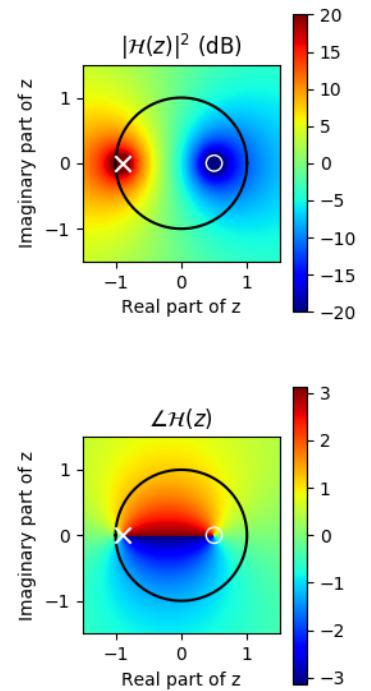


Figure 23.5: The magnitude of the system function given in Equation 23.34. The white cross indicates the pole at $z = -0.9$ and the white circle indicates the zero at $z = 0.5$. The black circle indicates the unit circle.

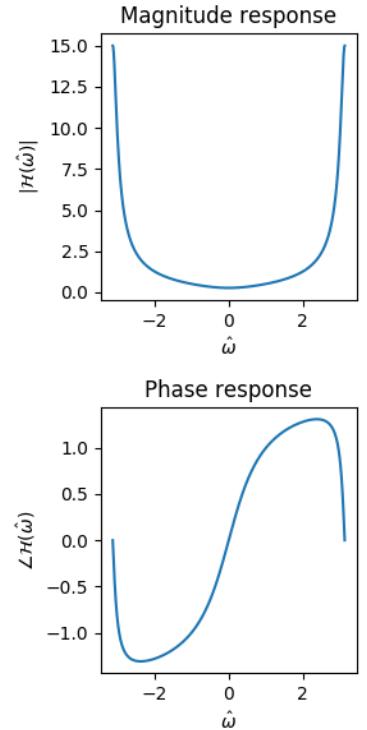


Figure 23.6: The magnitude and frequency response of the system given in Equation 23.34. This filter is a high-pass filter.

The output is also bounded:

$$|y[n]| \leq M_y < \infty. \quad (23.37)$$

The output of a filter is a convolution sum of the input signal $x[n]$ convolved with the impulse response $h[n]$ of the system. In the case of an LTI system, the magnitude of the output is:

$$|y[n]| = \left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right| \quad (23.38)$$

Using the triangle inequality ($c = a + b \Rightarrow |c| \leq |a| + |b|$), we can obtain the following bounds:

$$\left| \sum_{k=-\infty}^{\infty} h[k]x[n-k] \right| \leq \sum_{k=-\infty}^{\infty} |h[k]| \underbrace{|x[n-k]|}_{\leq M_x} \leq M_x \sum_{k=-\infty}^{\infty} |h[k]|. \quad (23.39)$$

Thus, a sufficient condition for bounded output $|y[n]| \leq M_y < \infty$ given a bounded input $|x[n]| \leq M_x < \infty$ is that the impulse response $h[n]$ is bounded to some finite value M_h :

$$\boxed{\sum_{k=-\infty}^{\infty} |h[k]| \leq M_h < \infty.} \quad (23.40)$$

Example: Stability of a first order IIR system

A first order system has the following system function:

$$\mathcal{H}(z) = \frac{b}{1 - az^{-1}} \quad (23.41)$$

When is this system stable?

The impulse response of this system is:

$$h[n] = ba^n u[n]. \quad (23.42)$$

Using the BIBO stability criterion, the output of this system is stable if:

$$\sum_{k=-\infty}^{\infty} |h[k]| \leq M_h < \infty. \quad (23.43)$$

Using Equation 23.42, we obtain:

$$\sum_{k=-N}^N |h[k]| = \sum_{k=-N}^N |ba^k u[k]| \quad (23.44)$$

$$= |b| \sum_{k=0}^N |a|^k \quad (23.45)$$

If $|a| \geq 1$, then

$$\sum_{k=-\infty}^{\infty} |h[k]| = \lim_{N \rightarrow \infty} |b| \sum_{k=0}^N |a|^k \rightarrow \infty \quad (23.46)$$

The system is BIBO unstable if $|a| \geq 1$.

We can use the geometric series solution formula for the case $|a| < 1$:

$$|b| \sum_{k=0}^N |a|^k = |b| \frac{1 - |a|^{N+1}}{1 - |a|}. \quad (23.47)$$

A limit of the infinite sum is thus a limit of the finite sum:

$$\sum_{k=-\infty}^{\infty} |h[k]| = \lim_{N \rightarrow \infty} |b| \frac{1 - |a|^{N+1}}{1 - |a|}. \quad (23.48)$$

When $|a| < 1$, we get:

$$\sum_{k=-\infty}^{\infty} |h[k]| = \frac{|b|}{1 - |a|} < \infty. \quad (23.49)$$

The system is BIBO stable when $|a| < 1$.

Poles and zeros

An arbitrary order IIR system has a system function, which is a polynomial fraction:

$$\mathcal{H}(z) = \frac{\sum_{k=0}^M b_k z^{-k}}{1 - \sum_{\ell=1}^N a_\ell z^{-\ell}} = \frac{P(z)}{Q(z)}. \quad (23.50)$$

The fundamental theorem of algebra says that because the numerator polynomial $P(z)$ is an M th order polynomial, it has M values of $z = \alpha_k$ for which $P(\alpha_k) = 0$.

Similarly, the denominator polynomial $Q(z)$ is an N th order polynomial, which has N roots $Q(\beta_\ell) = 0$.

We will assume that all the zeros of the numerator and denominator are distinct: $\alpha_k \neq \alpha_\ell$, $\alpha_k \neq \beta_\ell$, and $\beta_k \neq \beta_\ell$ for all $k \neq \ell$. We'll also assume that $M > 0$, that there is at least one zero of the numerator polynomial.

One can then express the system function of an IIR filter in the following form:

$$\mathcal{H}(z) = \frac{\prod_{k=1}^M (z - \alpha_k)}{\prod_{\ell=1}^N (z - \beta_\ell)} = \frac{P(z)}{Q(z)} \quad (23.51)$$

The terms β_ℓ are the *poles* of the system function. They indicate regions of the complex plane where the system function magnitude approaches infinity $\mathcal{H}(\beta_\ell) \rightarrow \infty$.

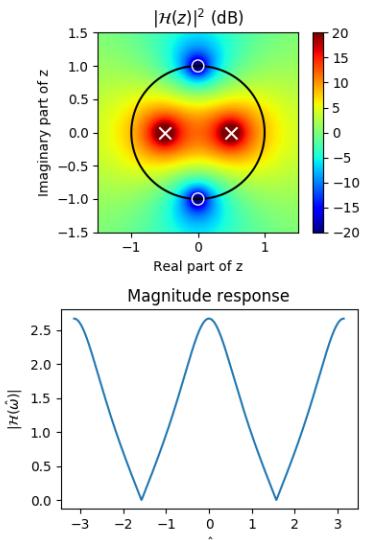


Figure 23.7: A band-stop filter.

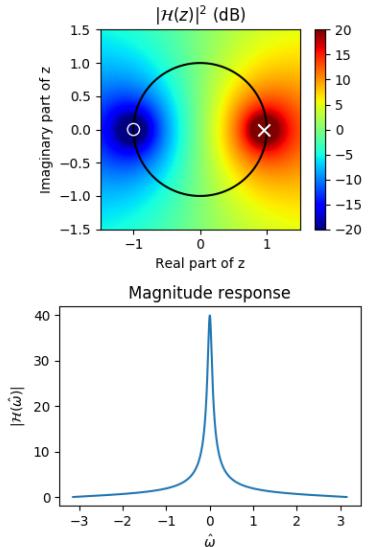


Figure 23.8: A low-pass filter.

The terms α_k are the *zeros* of the system function. They indicate regions of the complex plane where the system function magnitude becomes zero valued $\mathcal{H}(\alpha_k) = 0$.

Example: Pole-zero diagrams

By knowing the poles and zeros of the system function, it is possible to analytically say quite a lot about the behavior of the filter. Figures 23.7, 23.8, 23.9 and 23.10 show some examples of pole-zero diagrams and the magnitude responses associated with these filters. It is possible to tell from the positions of the poles and zeros what type of magnitude response the filter has.

Partial fraction decomposition

Using *partial fraction decomposition*, we can refactor the system function of an IIR filter in such a way that we can easily calculate an inverse z-transform of the system function using the elementary first order IIR filter z-transform pair that we covered earlier. I won't provide the general formulation here. Instead, I'll demonstrate this with an example, which can be generalized for higher order IIR systems.

Example: Partial fraction decomposition $N = 2$ and $M = 2$

Consider the case where the system function of an IIR filter is defined as:

$$\mathcal{H}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{(1 - \beta_1 z^{-1})(1 - \beta_2 z^{-1})}. \quad (23.52)$$

What is the impulse response of this system? Is this filter BIBO stable?

Note that the denominator of Equation 23.52 is already expressed in a form where β_1 and β_2 correspond to the poles of the system function.

The order of the numerator and denominator is $N = M = 2$. In this case, the partial fraction decomposition results in the following form:

$$\mathcal{H}(z) = \frac{A}{1 - \beta_1 z^{-1}} + \frac{B}{1 - \beta_2 z^{-1}} + C, \quad (23.53)$$

where A , B , and C are coefficients that need to be determined.

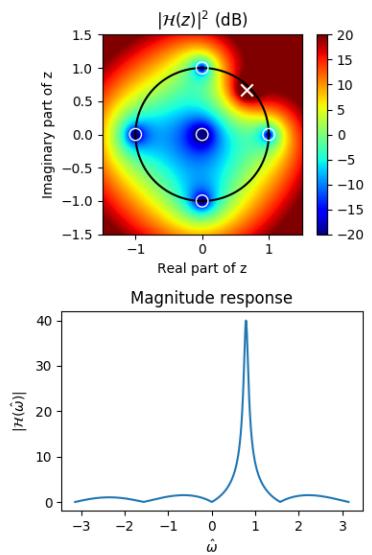


Figure 23.9: A band-pass filter that passes through positive frequencies. The resulting signal will be complex-valued, even if a real-valued signal is fed into the system.

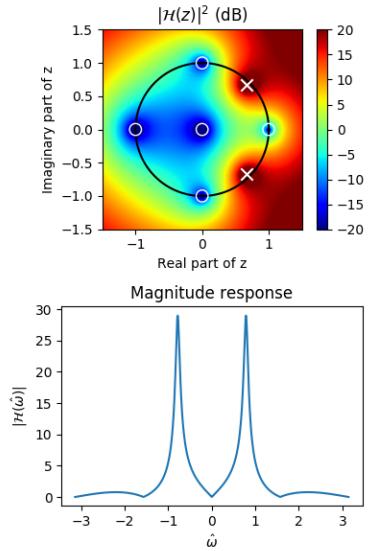


Figure 23.10: A band-pass filter

We can determine what these coefficients are:

$$\mathcal{H}(z) = \frac{A(1 - \beta_2 z^{-1}) + B(1 - \beta_1 z^{-1}) + C(1 - \beta_1 z^{-1})(1 - \beta_2 z^{-1})}{(1 - \beta_1 z^{-1})(1 - \beta_2 z^{-1})} \quad (23.54)$$

$$= \frac{(A + B + C) + (-A\beta_2 - B\beta_1 - C(\beta_1 + \beta_2))z^{-1} + (C\beta_1\beta_2)z^{-2}}{(1 - \beta_1 z^{-1})(1 - \beta_2 z^{-1})}. \quad (23.55)$$

By comparing Equation 23.52 with Equation 23.55, we get a set of linear equations, which can be used to solve for coefficients A , B , and C :

$$\begin{cases} A + B + C &= b_0 \\ -A\beta_2 - B\beta_1 - C(\beta_1 + \beta_2) &= b_1 \\ C\beta_1\beta_2 &= b_2 \end{cases} \quad (23.56)$$

The inverse z -transform can then be done using known z -transform pairs:

$$a\delta[n - n_0] \xleftrightarrow{Z} az^{-n_0} \quad (23.57)$$

and

$$ba^n u[n] \xleftrightarrow{Z} \frac{b}{1 - az^{-1}}. \quad (23.58)$$

The time-domain impulse response $h[n]$ is therefore:

$$h[n] = A\beta_1^n u[n] + B\beta_2^n u[n] + C\delta[n]. \quad (23.59)$$

In order to address the question of filter stability, we can now apply our previous BIBO stability result for first order IIR filter impulse response. The filter is stable as long as $|\beta_1| < 1$ and $|\beta_2| < 1$. In other words, the filter is stable if all the poles of the system function are inside the unit circle. This will hold for any system function that can be split using partial fraction decomposition.

IIR filtering with Scipy

It is possible to implement an IIR filter using Scipy. There is a function `scipy.signal.lfilter`, which implements an IIR filtering operation.

The function `lfilter` expects the filter coefficients to be specified in the following format:

$$\mathcal{H}(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \dots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \dots + a_M z^{-M}} \quad (23.60)$$

The example filter that is defined in Equation 23.32 is as follows:

$$\mathcal{H}(z) = \frac{1 - 0.5z^{-1}}{1 + 0.9z^{-1}} \quad (23.61)$$

In Python, we would form two vectors: $b=[1, -0.5]$ and $a=[1, 0.9]$ and pass these to `lfilter(b, a, signal)` to IIR filter a vector signal.

The Python code in Listing 23.1 demonstrates implementing the IIR filter that has the system function defined in Equation 23.61. It filters a Gaussian random noise signal, which has a flat power spectrum. How the filter modifies this spectrum allows us to inspect what the filter does to the signal. The code also has a simple routine that demonstrates how the same IIR filter could be naively implemented without resorting to a library function.

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as s

def simple_hpfilter(signal):
    # Implement IIR filter:
    #  $y[n] = -0.9*y[n-1] + x[n] - 0.5*x[n-1]$ 
    out = np.zeros(len(signal))

    for i in range(len(signal)):
        if i > 1:
            out[i] = signal[i]-0.9*out[i-1]-0.5*signal[i-1]

    return out

# Implement this system function
# 
$$H(z) = \frac{1 - 0.5 z^{-1}}{1 + 0.9 z^{-1}}$$

b = [1.0, -0.5]
a = [1, 0.9]

# Create a white noise signal, which has a flat power
# spectrum
signal = np.random.randn(100000)

# Filter the noise signal.
iir_filtered = s.lfilter(b, a, signal)
# The same, but using a slower direct routine.
iir_filtered_slow = simple_hpfilter(signal)

sample_rate = 1000.0
# Plot the power spectrum estimate.
plt.psd(iir_filtered, NFFT=1024, Fs=sample_rate)
plt.savefig("ex_psd_hpfilter.png")
plt.show()
```

Listing 23.1: 026_iir/simple_hpfilter.py

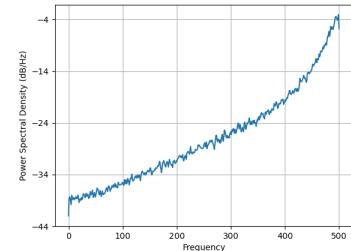


Figure 23.11: The power spectrum estimate of the filtered white noise signal using the simple high-pass filter specified in Equation 23.61. Compare this with the analytical magnitude response shown in Figure 23.6

IIR filter design with Scipy

Scipy includes an IIR filter design routines `scipy.signal.iirfilter` and `scipy.signal.iirdesign`. These functions can be used to design

bandpass, lowpass, highpass, and bandstop filters that meet design specifications.

Scipy functions `scipy.signal.lfilter` and `scipy.signal.sosfilt` can be used to filter signals using an IIR filter.

The Python code in Listing 23.2 shows an example of designing an IIR low-pass filter. The filter cutoff is set to be 100 Hz and the filter band-stop is set to be 200 Hz. The minimum attenuation for signals with frequencies higher than 200 Hz is set to be 100 dB.

I then generate a Gaussian random signal and filter this signal using the low-pass filter that was designed. A Gaussian random noise has a flat power spectrum, which allows us to investigate how the filter shapes this spectrum.

Finally, I use the `matplotlib.pyplot.psd` function to estimate the power spectrum of the filtered signal.

```
import matplotlib.pyplot as plt
import numpy as np
# You'll find this on:
# github.com/jvierine/signal_processing/o26_iir/plot_h.py
import plot_h
import scipy.signal as s

# Design an IIR band-pass filter with:
# Sample rate = 1000 Hz.
# Pass-band = 100 to 200 Hz.
# Maximum pass band ripple 1 dB.
# Minimum attenuation of signals out of band = 100 dB.
sample_rate = 1000.
# Return zeros and poles of the filter.
zeros, poles, k = s.iirdesign(100, 200, gpass=3.0, gstop=100,
                               ftype='ellip', output="zpk", fs=
                               sample_rate)
# Same filter, but return filter coefficients.
b, a = s.iirdesign(100, 200, gpass=3.0, gstop=100,
                   ftype='ellip', output="ba", fs=sample_rate)

# Create a white noise signal.
signal = np.random.randn(100000)

# Filter signal with the IIR filter specified with
# coefficients b and a.
filtered_signal = s.lfilter(b, a, signal)
plt.plot(filtered_signal)
plt.show()

# Plot the power spectrum estimate.
plt.psd(filtered_signal, NFFT=1024, Fs=sample_rate)
plt.savefig("ex_psd_lpf.png")
plt.show()

# Plot the system function and the magnitude response of the IIR
# filter.
plot_h.plot_hmag("ex_design_lpf.png",
                  zeros=zeros, poles=poles, vmin=-150,
                  fs=sample_rate)
```

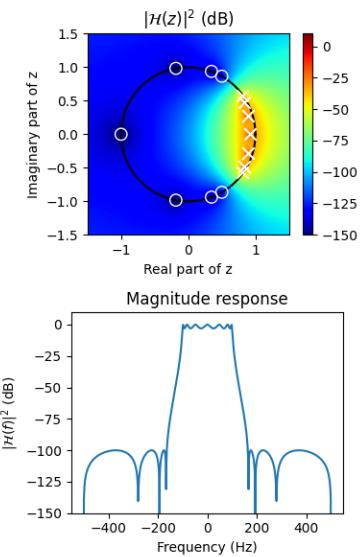


Figure 23.12: The pole-zero diagram and magnitude response of the designed IIR low-pass filter. The bandwidth is specified to be 100 Hz, with the stop-band starting at 200 Hz. Spectral components of the signal in the stop-band are attenuated by at least 100 dB.

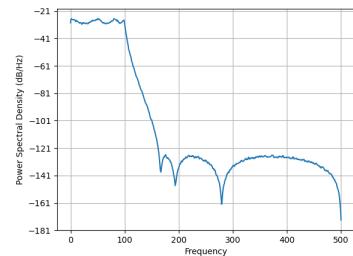


Figure 23.13: A power spectrum estimate of the filtered signal. The spectrum looks nearly identical to the analytic magnitude response of the filter shown in Figure 23.12.

Listing 23.2: 026_iir/iir_design.py

The inverse z-transform [Optional]

This section covers some details on the inverse z -transform. Although the definition looks complicated, it is pretty straight forward to compute. To do this, we require some tools from complex analysis. Recall that the inverse z -transform is defined as:

$$x[n] = \mathcal{Z}^{-1}\{X(z)\} = \frac{1}{2\pi i} \oint_C X(z)z^{n-1} dz. \quad (23.62)$$

The contour integral on the right can easily be computed by the residue theorem, which states:

$$\oint_C f(z) dz = 2\pi i \sum \text{Res}(f, z_0). \quad (23.63)$$

The $\text{Res}(f, z_0)$ means the residue of f at z_0 , where z_0 is a pole for f . The proper definition of the residue of a complex function f doesn't make sense if you haven't seen the concept of a Laurent series, so we'll ignore a proper definition and just state how this quantity is computed. The basic idea is to find this quantity called the residue of f at a pole z_0 with multiplicity m . The formula is given here as:

$$\text{Res}(f, z_0) = \lim_{z \rightarrow z_0} \frac{1}{(m-1)!} \frac{d^{m-1}}{dz^{m-1}} [(z - z_0)^m f(z)]. \quad (23.64)$$

Example: Pole without multiplicity

As an example, consider $X(z) = \frac{b}{1-az^{-1}}$. The inverse z -transform is then:

$$x[n] = \frac{1}{2\pi i} \oint_C \frac{bz^{n-1}}{1-az^{-1}} dz = \sum \text{Res}(f, z_0),$$

where C is a loop going counterclockwise around all the poles of $X(z)$. In this case $X(z)$ has one pole, being $z = a$. We get:

$$\text{Res}(f, a) = \lim_{z \rightarrow a} \frac{1}{0!} (z - a)^1 \frac{bz^{n-1}}{1-az^{-1}} = \lim_{z \rightarrow a} bz^n = ba^n.$$

We assume that the sequence $x[n]$ is only valid for $n \geq 0$, so the resulting discrete-time signal is then:

$$x[n] = ba^n u[n], \quad (23.65)$$

as was shown previously.

Example: Pole with multiplicity

Consider a more complicated example. Take $X(z)$ as:

$$X(z) = \frac{1}{(1 - az^{-1})^m}.$$

This can be rewritten as:

$$X(z) = \frac{z^m}{(z - a)^m}, \quad (23.66)$$

so we have a pole of order m . The inverse z -transform is then:

$$x[n] = \frac{1}{2\pi i} \oint_C \frac{z^m}{(z - a)^m} z^{n-1} dz, \quad (23.67)$$

where C is a loop going counterclockwise around the pole $z_0 = a$.

Here we use the residue theorem (Equation 23.63) and Equation 23.64:

$$\text{Res}(f, a) = \lim_{z \rightarrow a} \frac{1}{(m-1)!} \frac{d^{m-1}}{dz^{m-1}} \left[(z-a)^m \frac{z^m}{(z-a)^m} z^{n-1} \right], \quad (23.68)$$

$$= \lim_{z \rightarrow a} \frac{1}{(m-1)!} \frac{d^{m-1}}{dz^{m-1}} z^{m+n-1}, \quad (23.69)$$

$$= \lim_{z \rightarrow a} \frac{1}{(m-1)!} \frac{(m+n-1)!}{(m+n-1-(m-1))!} z^{m+n-1-(m-1)}, \quad (23.70)$$

$$= \lim_{z \rightarrow a} \frac{1}{(m-1)!} \frac{(m+n-1)!}{n!} z^n, \quad (23.71)$$

$$= \binom{m+n-1}{m-1} a^n, \quad (23.72)$$

where we've used the fact that:

$$\frac{d^k}{dz^k} z^n = \frac{n!}{(n-k)!k!} z^{n-k}, \quad (23.73)$$

and that $\binom{n}{k} = \frac{n!}{(n-k)!k!}$. Therefore, the signal is:

$$x[n] = \binom{m+n-1}{m-1} a^n u[n], \quad (23.74)$$

assuming $x[n] = 0$ for all $n < 0$.

Exercises: Infinite Impulse Response Filters

1. Consider the following linear time-invariant system $y[n] = \mathcal{T}\{x[n]\}$, which is defined as:

$$y[n] = y[n-1] + y[n-2] + x[n-1] \quad (23.75)$$

- a) Assuming that $h[n] = 0$ when $n < 0$, show that the first six values of the impulse response $h[n] = \mathcal{T}\{\delta[n]\}$ of the system defined in Equation 23.75 are:

$$h[0] = 0 \quad h[1] = 1 \quad h[2] = 1 \quad h[3] = 2 \quad h[4] = 3 \quad h[5] = 5$$

- b) Would this filter be classified in signal processing jargon as a *finite impulse response* filter (FIR), or an *infinite impulse response* (IIR) filter?
- c) z -transform Equation 23.75. Use $Y(z) = \mathcal{Z}\{y[n]\}$ and $X(z) = \mathcal{Z}\{x[n]\}$.
- d) Show that the system function $\mathcal{H}(z)$ that corresponds to the system defined in Equation 23.75 can be written as:

$$\mathcal{H}(z) = \frac{z^{-1}}{(1 - \varphi_1 z^{-1})(1 - \varphi_2 z^{-1})}$$

Remember that the output of a linear time invariant system in frequency domain is: $Y(z) = \mathcal{H}(z)X(z)$.

- e) Determine the values of φ_1 and φ_2 .
- f) The impulse response $h[n]$ of the system defined in Equation 23.75 corresponds to the *Fibonacci sequence*. Provide a closed form formula for the Fibonacci sequence without using recursion. Hint: $h[n] = \mathcal{Z}^{-1}\{\mathcal{H}(z)\}$. You can check your result by comparing the first few values of the sequence $h[0] = 0$, $h[1] = 1$, $h[2] = 1$, \dots .
- g) Is the system described in Equation 23.75 bounded-input bounded-output (BIBO) stable? In other words, does the system provide a bounded output for every bounded input? Justify your answer.
2. A time-shifted unit impulse signal is given by: $x[n] = \delta[n - n_0]$. The z -transform of this signal is $X(z) = z^{-n_0}$. What is the discrete-time Fourier transform of $x[n]$?
3. Determine the inverse z -transforms of the following:

a) $A(z) = \frac{1+2z^{-2}}{1-0.25z^{-1}}$

b) $B(z) = \frac{3}{1-0.3z^{-1}} - \frac{2}{1+0.7z^{-1}}$

- c) $C(z) = \frac{1+2z^{-2}}{1-0.4z^{-1}-0.32z^{-2}}$
4. The output of a filter is given by the first-order difference equation:
- $$y[n] = 0.8y[n-1] + 5x[n] \quad (23.76)$$
- What type of filter is this (IIR or FIR)?
 - Find the system function $\mathcal{H}(z)$ of the filter described by equation 23.76.
 - Provide a pole-zero plot of $\mathcal{H}(z)$. Is this system stable or not? Explain why.
 - Find the corresponding impulse response $h[n]$.
 - Is this filter a high-pass filter, band-pass filter, or a low-pass filter? Briefly explain why.
 - Let the input signal fed into the system be given by:

$$x[n] = u[n] \quad (23.77)$$

Find the output $y[n]$.

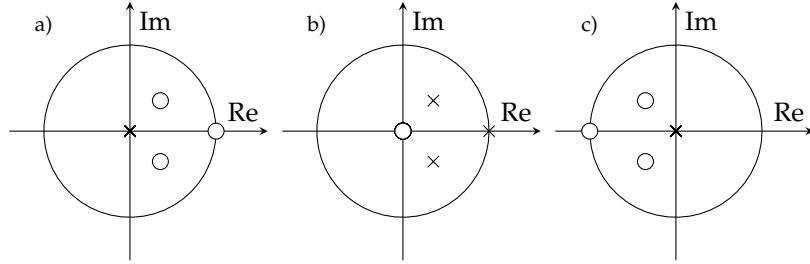


Figure 23.14: Three different pole-zero diagrams. The unit circle $z = e^{j\hat{\omega}}$ is marked in each diagram with a solid black line. The zeros of the system function $\mathcal{H}(z)$ are marked with circles (o) and the locations of the poles are marked with crosses (x). The real and imaginary components of poles and zeros only consist of the following numbers: 0, ± 1 , or $\pm(2\sqrt{2})^{-1}$.

5. A system function $\mathcal{H}(z)$ is the z -transform of the impulse response of a discrete-time LTI system. Figure 23.14 shows pole-zero diagrams that characterize three system functions: 23.14a), 23.14b) and 23.14c). Answer the following questions related to these three pole-zero diagrams. Justify your answers.

- Sketch an approximate magnitude response for the three LTI systems shown in Figure 23.14. Use normalized angular frequency $\hat{\omega}$ in units of radians per sample on the x-axis and the magnitude response $|\mathcal{H}(\hat{\omega})|$ of the y-axis.
- Which one of the pole-zero diagrams corresponds to:

$$\mathcal{H}(z) = \frac{(z+1)(z-\alpha)(z-\alpha^*)}{z^3}. \quad (23.78)$$

Here $\alpha = i(2\sqrt{2})^{-1} - (2\sqrt{2})^{-1}$.

- c) A low pass filter is a filter that reduces the absolute amplitude of high frequency spectral components relative to low frequency spectral components. Which one of the diagrams represents a **stable** low-pass filter?
- d) Which one of the diagrams represents a **stable** high-pass filter?
- e) The filter described in Figure 23.14c) completely filters out one spectral component of the input signal. What is the frequency of this spectral component in units of radians per sample.
- f) Which of the filters have a finite length impulse response?

Suggested solutions: Infinite Impulse Response Filters

1. Let $y[n] = \mathcal{T}\{x[n]\}$ be a discrete LTI system, defined as:

$$y[n] = y[n-1] + y[n-2] + x[n-1].$$

- a) The impulse response is simply $h[n] = \mathcal{T}\{\delta[n]\}$, giving that $h[n]$ must satisfy the following difference equation:

$$h[n] = h[n-1] + h[n-2] + \delta[n-1].$$

Assume $h[n] = 0$ for $n < 0$, then by iterating, we have:

$$h[0] = h[-1] + h[-2] + \delta[0-1] = 0,$$

$$h[1] = h[0] + h[-1] + \delta[0] = 1,$$

$$h[2] = h[1] + h[0] + \delta[1] = 1,$$

$$h[3] = h[2] + h[1] + \delta[2] = 2,$$

$$h[4] = h[3] + h[2] + \delta[3] = 3,$$

$$h[5] = h[4] + h[3] + \delta[4] = 5,$$

as we wanted to show.

- b) This is an infinite impulse response filter as it can be written as:

$$y[n] = \sum_{l=1}^2 a_l y[n-l] + \sum_{k=0}^1 b_k x[n-k],$$

with $a_1 = a_2 = 1$, $b_0 = 0$ and $b_1 = 1$.

- c) Calculating the z-transform, setting $Y(z) = \mathcal{Z}\{y[n]\}$, one gets:

$$Y(z) = \mathcal{Z}\{y[n-1]\} + \mathcal{Z}\{y[n-2]\} + \mathcal{Z}\{x[n-1]\},$$

$$Y(z) = z^{-1} \mathcal{Z}\{y[n]\} + z^{-2} \mathcal{Z}\{y[n]\} + z^{-1} \mathcal{Z}\{x[n]\},$$

$$Y(z) = z^{-1} Y(z) + z^{-2} Y(z) + z^{-1} X(z),$$

where the final equation can be solved to obtain:

$$Y(z) = \frac{z^{-1}}{1 - z^{-1} - z^{-2}} X(z).$$

- d) The system function is, by definition, the function $\mathcal{H}(z)$, such that:

$$Y(z) = \mathcal{H}(z) X(z).$$

Clearly, we must have:

$$\mathcal{H}(z) = \frac{z^{-1}}{1 - z^{-1} - z^{-2}} = \frac{z}{z^2 - z - 1} = \frac{z}{(z - \varphi_1)(z - \varphi_2)}$$

This can be factored as:

$$\mathcal{H}(z) = \frac{z^{-1}}{(1 - \varphi_1 z^{-1})(1 - \varphi_2 z^{-1})},$$

where φ_1 and φ_2 are the roots of the quadratic equation:

$$z^2 - z - 1 = 0.$$

- e) The values of φ_1 and φ_2 are the roots of the equation $z^2 - z - 1$. You can easily find the roots by solving the quadratic equation by the standard formula. On the other, the equation is the second degree polynomial that defined the golden ratio, therefore, we know the roots are:

$$\varphi_1 = \frac{1 + \sqrt{5}}{2},$$

$$\varphi_2 = \frac{1 - \sqrt{5}}{2}.$$

- f) Apply partial fraction decomposition to the system function:

$$\mathcal{H}(z) = \frac{z^{-1}}{(1 - \varphi_1 z^{-1})(1 - \varphi_2 z^{-1})} = \frac{A}{1 - \varphi_1 z^{-1}} + \frac{B}{1 - \varphi_2 z^{-1}},$$

from which we get:

$$z^{-1} = A(1 - \varphi_2 z^{-1}) + B(1 - \varphi_1 z^{-1}).$$

Collecting terms gives the following relations:

$$A + B = 0,$$

$$-A\varphi_2 - B\varphi_1 = 1.$$

Solving this linear system, we conclude that:

$$A = \frac{1}{\varphi_1 - \varphi_2},$$

$$B = -\frac{1}{\varphi_1 - \varphi_2}.$$

The system function can then be written as:

$$\mathcal{H}(z) = \frac{1}{\varphi_1 - \varphi_2} \frac{1}{1 - \varphi_1 z^{-1}} - \frac{1}{\varphi_1 - \varphi_2} \frac{1}{1 - \varphi_2 z^{-1}}.$$

Apply the inverse z -transform:

$$\begin{aligned} h[n] &= \mathcal{Z}^{-1}\{\mathcal{H}(z)\} = \frac{1}{\varphi_1 - \varphi_2} \mathcal{Z}^{-1} \left\{ \frac{1}{1 - \varphi_1 z^{-1}} \right\} - \frac{1}{\varphi_1 - \varphi_2} \mathcal{Z}^{-1} \left\{ \frac{1}{1 - \varphi_2 z^{-1}} \right\}, \\ &= \frac{1}{\varphi_1 - \varphi_2} \varphi_1^n u[n] - \frac{1}{\varphi_1 - \varphi_2} \varphi_2^n u[n]. \end{aligned}$$

You can verify that this is indeed a function that generates the Fibonacci sequence (do it!).

- g) By looking at the system function \mathcal{H} , we can see that there are one zero at $z = 0$ and two poles having values of φ_1 and φ_2 , as defined above. Here φ_1 is outside the unit circle, hence the system is BIBO unstable.
2. If a time-shifted unit impulse: $x[n] = \delta[n - n_0]$ satisfy $X(z) = z^{-n_0}$. Then the discrete-time Fourier transform is the z -transform evaluated on the unit circle. Thus:

$$\mathcal{F}\{x[n]\} = \mathcal{Z}\{x[n]\}|_{z=e^{i\hat{\omega}}} = (e^{i\hat{\omega}})^{-n_0} = e^{-i\hat{\omega}n_0}.$$

You can compare this with previous results and see that it is correct.

3. Recall the following inverse z -transforms:

$$\begin{aligned}\mathcal{Z}^{-1}\{z^{-k}\} &= \delta[n - k], \\ \mathcal{Z}^{-1}\left\{\frac{b}{1 - az^{-1}}\right\} &= ba^n u[n].\end{aligned}$$

- a) For the complex function:

$$A(z) = \frac{1 + 2z^{-2}}{1 - 0.25z^{-1}},$$

we can apply polynomial division to obtain:

$$A(z) = -8z^{-1} - 32 + \frac{33}{1 - 0.25z^{-1}}.$$

Now $A(z)$ consists of known inverse z -transforms. The inverse z -transform of $A(z)$ is then:

$$a[n] = \mathcal{Z}^{-1}\{A(z)\} = \underline{-8\delta[n - 1] - 32\delta[n] + 33(0.25)^n u[n]}.$$

- b) For the complex function:

$$B(z) = \frac{3}{1 - 0.3z^{-1}} - \frac{2}{1 + 0.7z^{-1}},$$

we can apply the known inverse z -transforms immediately:

$$b[n] = \mathcal{Z}^{-1}\{B(z)\} = \underline{3(0.3)^n u[n] - 2(-0.7)^n u[n]}.$$

- c) For the complex function:

$$C(z) = \frac{1 + 2z^{-2}}{1 - 0.4z^{-1} - 0.32z^{-2}},$$

we apply a polynomial division:

$$C(z) = -6.25 + \frac{7.25 - 2.5z^{-1}}{1 - 0.4z^{-1} - 0.32z^{-2}}.$$

Factor the denominator:

$$C(z) = -6.25 + \frac{7.25 - 2.5z^{-1}}{0.32(2.5 + z^{-1})(1.25 - z^{-1})}.$$

Apply a partial fraction decomposition on the last term:

$$\begin{aligned} \frac{7.25 - 2.5z^{-1}}{0.32(2.5 + z^{-1})(1.25 - z^{-1})} &= \frac{A}{2.5 + z^{-1}} + \frac{B}{1.25 - z^{-1}}, \\ 7.25 - 2.5z^{-1} &= 0.32A(1.25 - z^{-1}) + 0.32B(2.5 + z^{-1}). \end{aligned}$$

Get the following linear relations:

$$\begin{aligned} (0.32)(1.25)A + (0.32)(2.5)B &= 7.25, \\ -0.32A + 0.32B &= -2.5, \end{aligned}$$

so $A = 11.25$ and $B = 3.4375$. Then $C(z)$ can be written as:

$$\begin{aligned} C(z) &= -6.25 + \frac{11.25}{2.5 + z^{-1}} + \frac{3.4375}{1.25 - z^{-1}}, \\ &= -6.25 + \frac{4.5}{1 + 0.4z^{-1}} + \frac{2.75}{1 - 0.8z^{-1}}. \end{aligned}$$

Now, we can apply our known inverse z -transform pairs to obtain:

$$\underline{c[n] = -6.25\delta[n] + 4.5(-0.4)^n u[n] + 2.75(0.8)^n u[n]}.$$

4. Let $y[n]$ be given by the difference equation:

$$y[n] = 0.8y[n - 1] + 5x[n].$$

- a) The equation contains $y[n - 1]$ depending on one past value, hence the filter is an IIR.
- b) To find the system function, apply the z -transform:

$$Y(z) = 0.8z^{-1}Y(z) + 5X(z),$$

where $Y(z) = \mathcal{Z}\{y[n]\}$ and $X(z) = \mathcal{Z}\{x[n]\}$. Rewriting it on the form $Y(z) = H(z)X(z)$ one gets:

$$Y(z) = \frac{5}{1 - 0.8z^{-1}}X(z),$$

so the system function is:

$$H(z) = \frac{5}{1 - 0.8z^{-1}} = \frac{5z}{z - 0.8}.$$

- c) A pole-zero diagram for this system is shown in Figure 23.15. The poles for this system function all lie inside the unit circle, so the system is stable.

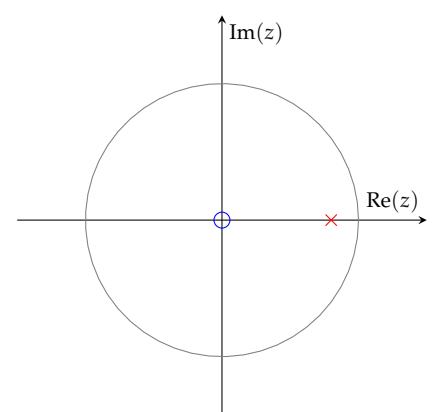


Figure 23.15: The system function has one zero and one pole in this case. There is a zero at $z = 0$ and a pole at $z = 0.8$. The zero is displayed as a blue circle and the pole as a red cross.

- d) The impulse response function can be found by the inverse z -transform of the system function. In this case, we get:

$$h[n] = \mathcal{Z}^{-1}\{\mathcal{H}(z)\} = \mathcal{Z}^{-1}\left\{\frac{5}{1 - 0.8z^{-1}}\right\} = 5(0.8)^n u[n].$$

- e) Any discrete LTI system can be written on the form:

$$y[n] = |\mathcal{H}(\hat{\omega})| \angle \mathcal{H}(\hat{\omega}) x[n],$$

so the frequencies are multiplied with $|\mathcal{H}(\hat{\omega})|$. In this case, the filter is a low-pass filter. This can be seen by drawing the magnitude response function, $|\mathcal{H}(\hat{\omega})|$. Low frequencies are attenuated while high frequencies are reduced, signifying a low-pass filter. On the other hand, this can be seen from the pole-zero diagram, as there is a pole close to $z = 1$, which corresponds to the frequency $\hat{\omega} = 0$. The pole will increase the magnitude of the frequencies close to the frequency $\hat{\omega} = 0$, meaning they will get increased, which is the defining property of a low-pass filter.

- f) The filter is LTI, so the system can be written as a convolution:

$$y[n] = h[n] * x[n].$$

Taking $x[n] = u[n]$ gives:

$$y[n] = h[n] * u[n] = \sum_{k=-\infty}^{\infty} u[k]h[n-k] = \sum_{k=0}^{\infty} h[n-k].$$

Set $m = n - k$, then the series can be rewritten as:

$$\begin{aligned} y[n] &= \sum_{m=-\infty}^n h[m], \\ &= \sum_{m=-\infty}^n 5(0.8)^m u[m], \end{aligned}$$

now $u[m]$ kills the negative terms as $u[m] = 0$ for $m < 0$ and $u[m] = 1$ for $m \geq 1$, thus the series simplifies to a regular sum, which can be found by the formula for a geometric sum:

$$\begin{aligned} y[n] &= 5 \sum_{m=0}^n (0.8)^m, \\ &= 5 \frac{1 - 0.8^{n+1}}{1 - 0.8} = 25(1 - 0.8^{n+1}). \end{aligned}$$

5. Consider the diagrams in Figure 23.14.

- a) The first diagram has a pole at 0 and three zeros. There is a zero at $z = 1$, which corresponds to $\hat{\omega} = 0$, thus this frequency

is filtered out. The other zeros and poles are barely noticeable, but the function will dip close to the zeros. For diagram b) there is a pole at $z = 1$, which is on the unit circle. Therefore, the function will tend towards infinity here. The other poles don't do much in comparison, but some small peaks would occur close to these values. The last diagram filters out $\hat{\omega} = \pm\pi$ as there is a zero for $z = -1$. A plot of the magnitude responses is shown in Figure 23.16.

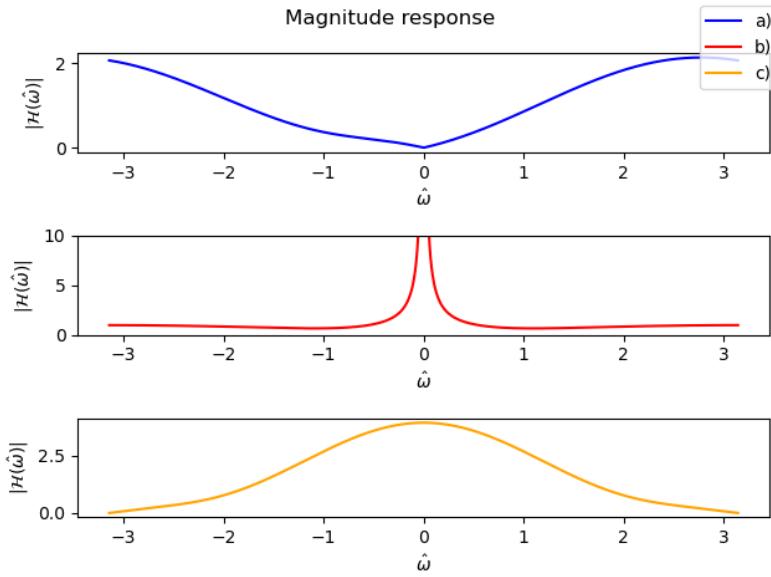


Figure 23.16: Magnitude response for the three pole-zero diagrams. Note that the limits of b) has been set to 0-10.

b) The system function:

$$\mathcal{H}(z) = \frac{(z+1)(z-\alpha)(z-\alpha^*)}{z^3},$$

has zeros corresponding to $z = -1, \alpha, \alpha^*$, thus this is the system function corresponding to diagram c) as $\alpha = i(2\sqrt{2})^{-1} - (2\sqrt{2})^{-1}$, which has a negative real part.

- c) Diagram b) and c) both reduce high frequency components relative to low frequency components, meaning both of these are low-pass filters. However, only diagram c) is stable. This can be seen from Figure 23.16 as the magnitude response becomes unbounded at $\hat{\omega} = 0$, but we can also see this from the pole-zero diagram, as diagram b) has a pole on the unit circle. Recall that a system function corresponds to a stable BIBO LTI system if all the poles lie inside the unit circle. If any pole is on the unit circle or outside the unit circle, the system is unstable.
- d) The only diagram left is a), as b) and c) are both low-pass filters. Diagram a) is indeed a high-pass filter as $\hat{\omega} = 0$ gets

sent to 0 and the high frequencies pass through. The filter is also stable as all the poles lie inside the unit circle.

- e) The filter corresponding to diagram c) filters out $\hat{\omega} = \pi$ completely. The filter has a zero at $z = -1$, which correspond to $e^{j\hat{\omega}} = -1$, meaning $\hat{\omega} = \pi$.
- f) Diagram a) and diagram c) represents FIR filters, as these only have a pole at 0. Recall that the system function for a FIR filter can be found by feeding a $\delta[n]$ into the LTI system. Observe:

$$y[n] = \sum_k b_k x[n - k],$$

which yield a system function of the form:

$$\mathcal{H}(z) = \sum b_k z^{-k}.$$

Bibliography

RB Hindsley and RA Bell. The period-luminosity relation for cepheid variable stars. *Astrophysical Journal, Part 1 (ISSN 0004-637X)*, vol. 341, June 15, 1989, p. 1004–1019., 341:1004–1019, 1989.

David W Kammler. *A first course in Fourier analysis*. Cambridge University Press, 2007.

Andrey Kolmogoroff. Une série de Fourier-Lebesgue divergente presque partout. *Fundamenta mathematicae*, 1(4):324–328, 1923.

Andreas Maier, Christopher Syben, Tobias Lasser, and Christian Riess. A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik*, 29(2):86–101, 2019.

Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Edmund Taylor Whittaker. On the functions which are represented by the expansions of the interpolation-theory. *Proceedings of the Royal Society of Edinburgh*, 35:181–194, 1915.

Index

- cos, 57
- sin, 57
- z-transform, 316

- aliases, 139
- aliasing, 23
- amplitude, 57
- Anaconda, 15
- analysis, 77, 80
- angular frequency, 57
- argument, 23
- array functions, 18

- band-pass filter, 202
- bandpass, 351
- bandstop, 351
- basis vectors, 262
- blocking filter, 328
- boundary condition, 343
- Bounded Input Bounded Output, 345

- cascade of first order filters, 324
- Cepheid variable star, 106
- chained LTI system, 172
- commensurable, 79
- complex conjugate, 24
- complex number, 23
- complex sinusoidal signal, 57
- compression, 9
- compression algorithm, 9
- continuous-time, 12

- continuous-time Fourier transform, 77, 89
- contour integral, 317
- convolution, 38, 167
- convolution theorem, 322

- dependent variable, 33
- difference equation, 320
- diffraction limit, 248
- Dirac delta function, 51
- Dirichlet kernel, 205
- discrete Fourier transform, 259, 262
- discrete Fourier transform basis vectors, 263
- discrete-time, 12
- discrete-time angular frequency, 138
- Discrete-Time Fourier Transform, 221
- dynamic spectrum, 283

- eigenfunction, 58, 197
- electromagnetic waves, 64
- Euclid's algorithm, 79
- Euler's formula, 10, 23
- exponentially decaying, 316
- exponentially growing, 316

- Fast Fourier Transform, 267
- fast Fourier transform, 259
- Fibonacci sequence, 354
- filter bank, 278

- filters, 291
- folded alias, 141
- forward z-transform, 317
- forward Fourier transform, 113
- Fourier series, 9, 77
- Fourier transform, 11
- frequency, 58
- frequency domain, 113
- frequency response, 197, 198
- fundamental angular frequency, 78
- fundamental frequency, 77
- fundamental period, 59, 77, 78
- fundamental theorem of algebra, 324
- Gaussian density function, 51
- geometric series, 205, 263, 329
- greatest common divisor, 79
- Heaviside-function, 53
- Heisenberg uncertainty principle, 247
- high-pass filter, 202
- highpass, 351
- IIR, 341
- impulse response, 170
- independent variable, 33
- infinite impulse response, 341
- inverse z-transform, 317
- inverse discrete Fourier transform, 262
- inverse Euler, 25
- inverse Fourier transform, 113
- JPEG, 10
- Kronecker delta, 262
- Laplacian operator, 65
- Linear system, 39
- linear time-invariant, 38
- Linear time-invariant systems, 167
- linearity, 38
- low-pass filtering, 88
- lowpass, 351
- LTI, 38
- magnitude, 23
- magnitude response, 199
- Mandelbrot set, 18
- Matplotlib, 15, 18
- Maxwell's equations, 64
- modulus, 23
- MP3, 10
- natural number, 23
- NumPy, 15, 18
- Nyquist oversampling, 145
- operator., 37
- orthogonality of basis functions, 82
- oversampled, 145
- partial fraction decomposition, 348
- periodic discrete-time signals, 260
- periodic convolution, 268
- periodic function, 59
- periodic sinc, 205
- phase, 57
- phase response, 200
- phasor, 57
- phasor summation, 60
- polar representation, 23
- pole-zero diagrams, 348
- pole-zero diagram, 345
- poles, 347
- principal alias, 143
- principal spectrum, 143
- proof of Euler's formula, 23
- Python, 15, 18

quantized, 36
radians per sample, 138
rectangular function, 53
region of convergence, 317
reversibility, 264
Richard Feynman, 10
running average, 329

sample-rate, 137
sample-spacing, 137
sampling-rate, 137
SciPy, 15
sifting, 52
signal, 33
signal processing, 9
signals and systems, 9
simple algebraic manipulations, 320
spectral analysis, 277
spectrogram, 283
stability, 345
synthesis, 77, 80
system, 33, 37
system function, 315, 316

tapering windows, 236
time domain, 113

time shifted signal, 123
time-frequency ambiguity, 206
time-frequency uncertainty principle, 247
time-invariance, 38
Time-invariant, 39
time-shift theorem, 321
time-shifted unit impulse, 318
transfer function, 316
two dimensional complex sinusoidal signal, 61
two-dimensional complex sinusoidal signal, 64

Ubuntu, 15
unit impulse, 51
unit step, 51

Visual Studio Code, 15

wave equation, 65
wavenumber, 59
whitening filter, 296
window functions, 236

z-transform, 315
zero-padded DFT, 266
zeros, 348