Lecture Notes for ES3C5: Signal Processing
Version 1.3

Dr Adam Noel
adam.noel@warwick.ac.uk

School of Engineering
University of Warwick
Autumn 2022

Contents

Version History

These notes are Version 1.3 for the 2022-2023 academic year.

- 1.3: Clarified in Lesson 4 End-of-Lesson Problem 5 that $0\,\text{dB} = 1$, since dB-to-linear version is not introduced until the following Lesson. Corrected intermediate calculation for Lesson 14 Example 14.2. Corrected reasoning in solution to Lesson 19 End-of-Lesson Problem 1.

- 1.2: Added Lesson 12 End-of-Lesson Problem 8.

- 1.1: Corrected answer listed for Lesson 2 End-of-Lesson Problem 6(ii). Corrected wording of filter band specification in Lesson 5 End-of-Lesson Problem 4. Corrected label on Appendix Example A.2 and corresponding introduction to state that it is an example of partial fractions with repeated roots and not complex roots.

- 1.0: Added EXTRA comment in Lesson 3 on LTI systems being real. Added comment on filter characteristics providing constraints in Lesson 5. Added clarification in Lesson 7 that order argument in the `butter` function is only half the total order for band pass and band stop filters. Extended sequence indices displayed for Lesson 7 Example 9.4. Shifted labels in Fig. 12.1 for visibility. Designated Lesson 15 Section 15.2.2 as EXTRA content since it is not recommended to use the symbolic math toolbox for digital systems. Updated narrative in Lesson 15 Section 15.2.3 to specify that polynomial coefficient vectors should be written for negative powers of z and where the leading denominator coefficient is 1. Designated the paragraph in Lesson 18 on sequential least squares and constrained least squares to be EXTRA content. Added section on interval limits to Mathematical Background Appendix A.

- 0.0: Initial version copied from notes for 2021-2022 academic year.

Acknowledgements

These notes were developed with the support of materials from previous offerings of this and similar modules in the School of Engineering at the University of Warwick:

- Lecture slides for ES3C5 by Prof Weisi Guo and Dr Christos Mias. These were last updated for the 2018-2019 academic year, when the module was co-taught by Dr Mias and Dr Adam Noel.
- Module textbook "Systems and Control: an introduction to systems, signal processing and linear control," published in 2004 by Dr Tardi Tjahjadi. This text is included in the "Further Reading" list for this module and several copies can be borrowed from the University of Warwick Library.

Preface

0.1 Learning Outcomes

By the end of this lesson, you should ...

1. **Understand** the structure and context of ES3C5: Signal Processing.
2. Be able to **Recall** the background mathematical knowledge that is required and assumed for ES3C5: Signal Processing.

0.2 Welcome

Welcome to ES3C5: Signal Processing. This document is the official set of notes to supplement the lectures (in-person and video lectures). It is written as a series of lessons that correspond to the lectures. Each lesson will align with one to several video lectures. The video lectures focus on highlighting the main theoretical ideas and presenting example problems that are worked on in the videos. These notes are intended to provide a precise, *self-contained* presentation of all technical material needed for ES3C5, including formal definitions and descriptions of the theoretical ideas. These notes also present additional applications, example problems, code snippets, etc.

This raises two important questions:

1. Do you need to read these notes if you watch the video lectures and attend the in-person lectures?
2. Do you need to attend the in-person lectures and watch the video lectures if you read these notes?

In practice, depending on your personal approach to learning, you may find the lectures and videos more helpful than the notes, or vice versa. Nevertheless, it is recommended that you use ***all of them***.

A few reasons to attend the in-person lectures:

- **Opportunity to ask and hear questions**. The benefits of live discussion in a classroom setting are difficult to replicate in any other format.
- **Live examples and software demonstrations**. The coursework is primarily software-based (MATLAB). There will be several sessions focusing on live MATLAB demonstrations.
- **Lecture capture isn't perfect**. Lecture capture is a very useful tool and we will be using it for all in-person lectures, but it is not 100% guaranteed. Previous limitations have included not capturing all of the projected screens and not always capturing the audio.

A few reasons to watch the video lectures:

- **Keep on top of the material**. While you may have been exposed to many of the mathematical concepts in previous modules, ES3C5 covers a lot of ground and ties together a lot of ideas from a design perspective. The video lectures will help you keep up.
- **Emphasize what's most important**. The video lectures won't cover all of the material in the same detail as the notes, but they will highlight the most important content that you will be assessed on.
- **Be guided through problems**. We will work through a lot of examples and they can be easier to follow in the video lectures than to read through solutions yourself.

A few reasons to use the notes:

- **Preview lecture material**. The notes can help set the stage for what will be covered in the lectures.
- **Clarify details about what was covered in lectures**. After an in-person or video lecture, the notes might help you to resolve lingering questions.
- **The notes are self-contained**. There will be no concept that you need to know for the exam that isn't in the notes (though the coursework may require you to do some additional research). This can make the notes very useful for exam revision.
- **More accessible than the textbooks**. Although the notes are self-contained, they are written to be more concise and accessible than the module textbooks. Furthermore, the scope of the module is larger than what can be found in any one of the module textbooks.
- **Additional study aides**. The notes include many example problems; many but not all of these will be covered during video and in-person lectures.

0.2.1 Structure of These Notes

The notes are divided into a series of lessons. Each lesson has its own learning outcomes and sample problems. Some lesson content (including problems and applications) is provided as advanced material that is beyond the scope of ES3C5; any such content is clearly labelled as per below. The video lectures follow a similar structure, except that most lessons are taught over multiple video lectures.

Example 0.1

Examples will be clearly labelled as such and will appear inside a rounded box.

EXTRA: Demonstration of the Extra Material Box

Extra content will be clearly labelled as such and will appear inside a blue box. Any content inside of a blue box is beyond the scope of the module. Content might include advanced topics, extended applications, or just tangential comments that are provided for insight or reference.

Example 0.2: Example of Extra Material

If an example is inside of a blue box, then it is also beyond the scope of the module, i.e., is harder than what is expected to be asked in any of the assessments.

These notes were originally written for the 2019-2020 academic year and have been updated for the 2022-2023 academic year. Feedback on the content and suggestions for future years are welcome.

0.3 ES3C5 Overview

ES3C5 has several formal learning outcomes. By the end of the module you should be able to ...

1. **Apply** mathematics to analyse deterministic and random signals and to analyse processing systems.

2. **Apply** signal processing systems to classify signals and extract information.
3. **Critique** practical issues behind signal processing and information retrieval.
4. **Design** signal processing systems.
5. **Model** signals, filters and processes using computer packages.
6. **Evaluate** signals and systems using laboratory test and measurement equipment.

The learning objectives mention signals but not what kinds of signals (besides deterministic vs random). This module and these notes are organized according to the signal type. We will consider deterministic analog signals, deterministic digital signals, and random digital signals. Although most practical signals have a random component, we first consider deterministic signals because they are simpler to analyse. We also focus on signal processing systems that are filters, which are broadly applicable to a very wide range of signal processing applications.

This module will not teach you everything there is or everything you will ever need to know about signal processing. But it will help you to develop a skill set to understand signal processing, design (relatively) simple signal processing systems, and be aware of some more advanced signal processing techniques.

0.3.1 Module in Context

ES3C5: Signal Processing is a core module for the Systems, Biomedical, and EE/EEE streams, and optional for students in the General engineering program. It can also be taken by MSc students who do not have a background in signal processing. It builds most directly on material that is covered in ES2C7 (Engineering Mathematics and Technical Computing) and is the foundation for all of the subsequent signal processing modules in the School.

The broad applicability of signal processing is reflected in the diverse modules that build on this one, including ES335 (Communications Systems), ES4A4 (Biomedical Signal Processing), ES4E9 (Affective Computing), etc.

You will also find that many 3rd and 4th year projects include a signal processing component, so you may pick up some skills or discover methods that you can apply in your project work. Of course, you should also find this module relevant as a practising engineer.

0.3.2 Textbooks

The following textbooks are the most relevant for this module:

- "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014. Lathi has authored several popular textbooks on signals and systems. This recent text is quite accessible and features strong integration with MATLAB.

- "Essential MATLAB for engineers and scientists," B. Hahn and D. Valentine, Academic Press, 7th Edition, 2019. There are many excellent free resources for MATLAB, including the official software documentation (go to the help browser within the software or visit `https://uk.mathworks.com/help/matlab/index.html`). While this book has only a very brief chapter on signal processing, it is good for a broad overview of MATLAB if you are seeking a general reference. It is also up to date as of MATLAB release 2018b.

- "Discrete-Time Signal Processing," Oppenheim and Schafer, Pearson, 3rd Edition, 2013. Every signal processing textbook will have its relative strengths and weaknesses. This book serves as an alternative to Lathi and Green's "Essentials of Digital Signal Processing." While MATLAB is used for some of the examples, it is not thoroughly integrated, but overall this book has greater depth and breadth of topics. For example, it provides better coverage of random processes and signals.

We will refer to several other textbooks and resources throughout the module, but they will only be relevant for 1 or 2 lessons each. Please see "Further Reading" at the end of each lesson for details.

0.3.3 Administrative Details

Please refer to the module syllabus and announcements for current information about the lecture schedule, contacting the instructors, support and feedback arrangements, and the assessment schedule.

0.4 Mathematical Knowledge

ES3C5 is quite mathematically-focussed and it is assumed that you have mathematical background from previous modules in your degree program. In particular, for successful completion of this module it is essential that you are comfortable with the following topics:

- Partial fractions

- Integration (including integration by parts)

- Complex numbers (including calculation of magnitude and phase)
- Curve sketching
- Elementary algebra, including the quadratic equation
- Laplace transform

We also assume that you have some familiarity with calculating frequency responses, but we cover these extensively in the module.

Previously, it was also useful to be familiar with basic electrical circuits (e.g., passive elements, ideal op-amps). However, since this module is now taken by students from non-engineering backgrounds, you will **not** be required to analyse a circuit and any corresponding example problems are labelled as Extra material.

Some mathematical background is provided in Appendix A. Furthermore, we will recall some concepts as needed when we go through sample problems.

Lesson 1

Introduction

This lesson establishes some basic definitions and provides examples of signals and signal processing systems. The distinction between analogue vs digital signals will divide the first two parts of this module.

1.1 Learning Outcomes

By the end of this lesson, you should ...

1. **Understand** that there are different classes of signals.
2. **Understand** the meaning of signal processing and be aware of its applications.

1.2 Signals and Signal Classification

So what are signals? A **signal** is a quantity that can be varied in order to convey information. If a signal does not contain useful information (at least not in the current context), then the signal is regarded as **noise**. You may have a useful audio signal for your neighbour in a lecture, but this could be noise to anyone nearby that is trying to listen to the instructor!

Practically any physical phenomena can be understood as a signal (e.g., temperature, pressure, concentration, voltage, current, impedance, velocity, displacement, vibrations, colour). Immaterial quantities can also be signals (e.g., words, stock prices, module marks). Signals are usually described over time, frequency, and/or spatial domains. Time and frequency will be the most common in the context of this module, but our brief introduction to image processing will treat images as two-dimensional signals.

There are several ways of classifying signals. We will classify according to how they are defined over time and in amplitude. Over *time* we have:

1. **Continuous-time signals** – signals that are specified for every value of time t (e.g., sound level in a classroom).

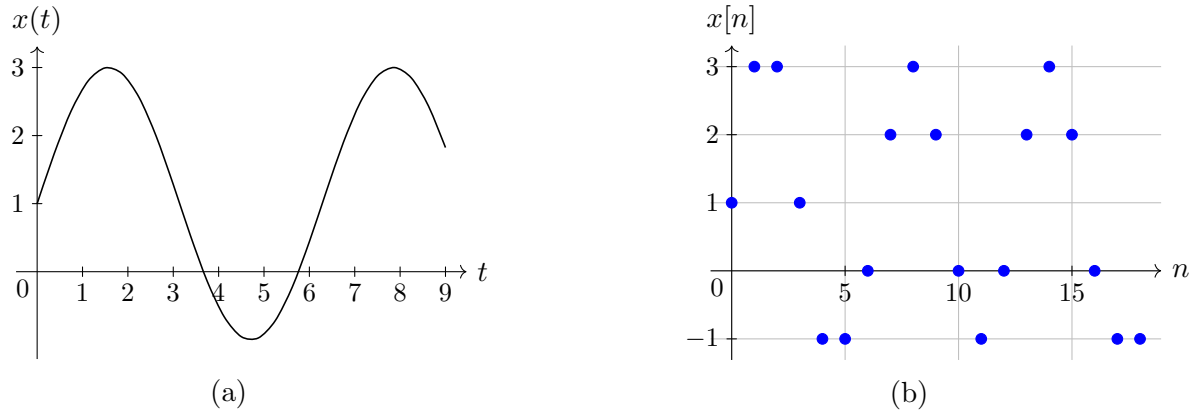
2. **Discrete-time signals** – signals that are specified at discrete values of time (e.g., the average daily temperature). The times are usually denoted by the integer n.

In *amplitude* we have:

1. **Analogue signals** – signals can have any value over a continuous range (e.g., body temperature).

2. **Digital signals** – signals whose amplitude is restricted to a finite number of values (e.g., the result of rolling a die).

While we can mix and match these classes of signals, in practice we most often see **continuous-time analogue signals** (i.e., many physical phenomena) and **discrete-time digital signals** (i.e., how signals are most easily represented in a computer); see Fig. 1.1. However, digital representations of data are often difficult to analyse mathematically, so we will usually treat them as if they were analogue. Thus, the *key distinction is actually continuous-time versus discrete-time*, even though for convenience we will refer to these as analogue and digital. The corresponding mathematics for continuous-time and discrete-time signals are distinct, and so they also impose the structure of this module.



(a) (b)

Figure 1.1: Comparison of classes of signals. (a) A continuous-time analogue signal $x(t)$. (b) A discrete-time digital signal $x[n]$ where amplitudes must be a whole integer.

Although many signals are not naturally in an electrical form, we can convert them to electrical form using **transducers**. We do this because most physical signal processing is done electrically (whether in analogue electronics or digital electronics).

1.3 What is Signal Processing?

In short, **signal processing** applies mathematical operations to a signal. Signal processing is applied in many disciplines in practice. Here are some top-level examples:

(a) **Image and video processing**. Used in industrial machine vision, target tracking, media compression, social media photo filters, etc.

(b) **Communication systems**. Used to package information for transmission over a noisy channel (wired or wireless) and recover at a destination.

(c) **Audio mixing**. Used to amplify sounds at different frequencies, noise cancellation, karaoke, introduce effects such as reverb, distortion, and delay, etc.

(d) **Biomedical systems**. Used to monitor vital signs, diagnose diseases, guide surgical procedures, etc.

(e) **Artificial intelligence**. Self-driving cars, speech/pattern recognition, smart homes (heating, appliances), video games, etc.

(f) **Financial markets**. Predict future prices (of currencies, stocks, options, houses, etc.) and optimise portfolio asset allocations.

We will not be covering all of these applications in this module, particularly as some of them rely on more advanced methods than what we will learn about. But we will use a diverse range of applications for our examples.

1.4 Summary

- **Signals** are quantities that vary to convey information.
- This module will focus on **continuous-time analogue signals**, which are continuous in both time and amplitude, and **discrete-time analogue signals**, which are discrete in both time. For convenience we will usually refer to these as just **analogue signals** and **digital signals**.
- **Signal processing** applies mathematical operations to a signal. There are many engineering fields that make use of it.

1.5 Further Reading

- Section 1.1 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

1.6 End-of-Lesson Problems

We haven't covered enough material yet to provide meaningful problems, but consider the following reflective questions to review and prepare for the module:

1. Refer to the top-level examples of signal processing in Section 1.3. Can you infer what the signals might be and how they would be classified?

2. Refer to the background mathematical material in Appendix A. Are you comfortable with the skills needed for this module?

Part I

Analogue Signals and Systems

Lesson 2

Laplace Transforms and LTI Systems

The first part of this module is the study of **Analogue systems and signals**. As discussed in Lesson 1, most practical physical phenomena can be represented as continuous-time analogue signals. In this lesson, we present the notion of Laplace transforms (which should be a review from previous modules) and how they can be applied to linear time invariant (LTI) systems. This forms the basis for all of the analysis of analogue signal processing that we will perform in this part of the module.

2.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Apply** the Laplace transform and the inverse Laplace transform.
2. **Apply** properties of the Laplace transform and the inverse Laplace transform.
3. **Understand** what a Linear Time-Invariant (LTI) system is.
4. **Understand** the definitions of a transfer function and convolution.
5. **Understand** the common simple input signals.
6. **Apply** the Laplace transform and inverse Laplace transform to find the system dynamic response to an input.
7. **Analyse** common electrical circuits with RLC components.

2.2 Linear Time Invariant Systems

This is a module on signal processing, and in this context we perform signal processing through **systems**, which take a signal as an input and then return a signal as an output. We will focus on systems that we can design as engineers, i.e., with particular system processing goals in mind. For example, in communication problems, there is a natural system that distorts our communication signal, and we design a receiver system to help us recover the original signal.

We will focus our study of analogue systems in this part of the module on a particular class of systems: those that are **Linear Time Invariant** (LTI). LTI systems have particular properties when acting on input signals. Given an LTI system that is defined by the functional (i.e., function of a function) $\mathcal{F}\{\cdot\}$ acting on time-varying input signals $x_1(t)$ and $x_2(t)$, where t is time, the properties are as follows:

1. The system is **linear**, meaning that:
   (a) The system is **additive**, i.e.,
   $$\mathcal{F}\{x_1(t) + x_2(t)\} = \mathcal{F}\{x_1(t)\} + \mathcal{F}\{x_2(t)\} \tag{2.1}$$
   (b) The system is **scalable** (or **homogeneous**), i.e.,
   $$\mathcal{F}\{ax_1(t)\} = a\mathcal{F}\{x_1(t)\} \tag{2.2}$$
   for any real or complex constant a.
2. The system is **time-invariant**, i.e., if output $y(t) = \mathcal{F}\{x_1(t)\}$, then
   $$y(t-\tau) = \mathcal{F}\{x_1(t-\tau)\}. \tag{2.3}$$
   In other words, delaying the input by some constant time τ will delay the output and make no other changes.

Part of the convenience of working with LTI systems is that we can derive the output $y(t)$ given the input $x(t)$, if we know the system's **impulse response** $h(t)$. The impulse response is the system output when the input is a Dirac delta, i.e.,

$$h(t) = \mathcal{F}\{\delta(t)\}. \tag{2.4}$$

Given the impulse response $h(t)$ of a system, the output is the **convolution** of the input signal with the impulse response, i.e.,

$$y(t) = \int_0^t x(\tau)\, h(t-\tau)\, d\tau = x(t) * h(t) = h(t) * x(t). \tag{2.5}$$

Convolution can be described as "**flip-and-shift**", because in effect it flips one function about the horizontal time axis (around $t = 0$) and then measures the area as that function is shifted past the other one.

While it is nice that the output of convolution is still a time-domain signal, it is often cumbersome to use in practice and it doesn't offer much intuition to help us design $h(t)$ or interpret it for a particular system. To help us out, we will find it helpful to take advantage of the **Laplace transform**.

2.3 The Laplace Transform

2.3.1 Laplace Transform Definition

The **Laplace transform** $\mathcal{L}\{\cdot\}$ is a very widely used signal transformation in the physical sciences and engineering. It converts a time-domain function $f(t)$, which we will assume is causal, i.e., $f(t) = 0, \forall t < 0$, into a complex domain function $F(s)$ that we say is in the Laplace domain or the s-domain. It is defined as follows:

$$\mathcal{L}\{f(t)\} = F(s) = \int_{t=0}^{\infty} f(t)\, e^{-st} dt, \tag{2.6}$$

where $s = \sigma + j\omega$ is a complex independent parameter.

EXTRA 2.1: The Bilateral Laplace Transform

The definition above for the Laplace transform is formally for the *unilateral* Laplace transform and is specifically for causal signals. But what if our function $f(t)$ is not causal, i.e., what if it is non-zero for some negative values of t? In this case, we should use the *bilateral* Laplace transform, which is defined as

$$F(s) = \int_{t=-\infty}^{\infty} f(t)\, e^{-st} dt. \tag{2.7}$$

Why would we not always use this form, since it is more general? The bilateral transform is more involved than the unilateral transform, particularly when evaluating its inverse, because we need to consider its **Region of Convergence**. Since we are generally more interested in causal signals, and your Engineering data book only provides a table for unilateral Laplace transforms anyway, we will just assume that all Laplace transforms in this module are unilateral. For additional details, refer to Section 1.10 of Lathi and Green.

You will likely have already seen the Laplace transform in previous modules (hopefully!). The notation may have been slightly different, but we will present it consistently here. While we will formally write the transform as $F(s) = \mathcal{L}\{f(t)\}$, we will also show conversion between the time and Laplace domains with a double arrow, i.e., by writing

$$f(t) \Longleftrightarrow F(s). \tag{2.8}$$

The double arrow suggests that we can also reverse the transform to go from the Laplace domain back to the time domain, and this is true. The **inverse Laplace transform** $\mathcal{L}^{-1}\{\cdot\}$ is defined as follows:

$$\mathcal{L}^{-1}\{F(s)\} = f(t) = \frac{1}{2\pi j}\lim_{T\to\infty}\int_{s=\gamma-jT}^{\gamma+jT} F(s)\,e^{st}ds, \tag{2.9}$$

which converts the complex function $F(s)$ into the causal ($t > 0$) time-domain signal $f(t)$.

Example 2.1: Laplace Transform of the Step Function

Find the Laplace transform of the step function

$$f(t) = u(t) = \begin{cases} 1, & t \geq 0, \\ 0, & t < 0. \end{cases} \tag{2.10}$$

Using the definition of the Laplace transform, we can write

$$\begin{aligned} F(s) &= \int_{t=0}^{\infty} f(t)\,e^{-st}dt \\ &= \int_{t=0}^{\infty} e^{-st}dt \\ &= -\frac{e^{-st}}{s}\Big|_0^{\infty} = \boxed{\frac{1}{s}}, s \neq 0. \end{aligned} \tag{2.11}$$

Example 2.2: Laplace Transform of an Exponential Function

Find the Laplace transform of the exponential function

$$f(t) = e^{at}, \tag{2.12}$$

for real constant a. Using the definition of the Laplace transform, we can write

$$\begin{aligned} F(s) &= \int_{t=0}^{\infty} f(t)\, e^{-st} dt \\ &= \int_{t=0}^{\infty} e^{at} e^{-st} dt = \int_{t=0}^{\infty} e^{(a-s)t} dt \\ &= \frac{1}{a-s} e^{(a-s)t}\Big|_0^{\infty} = \boxed{\frac{1}{s-a}}, s \neq a. \end{aligned} \tag{2.13}$$

So far, it is not easy to appreciate the benefit of using the Laplace transform over a convolution to study systems. However, the Laplace transform is almost entirely *bijective* (and for this module we will always assume that it is bijective), which means there is a one-to-one mapping between the time-domain function $f(t)$ and its transform $F(s)$, i.e., a time-domain function has a unique Laplace domain function and vice versa. Common function pairs are published in tables and there is a Laplace transform table in the Engineering Data Book; a brief list of common transform pairs is presented here in Example 2.3.

Example 2.3: Sample Laplace Transform Pairs

$$\begin{array}{cc} f(t) & F(s) \\ 1 & \Longleftrightarrow \quad \frac{1}{s} \\ e^{at} & \Longleftrightarrow \quad \frac{1}{s-a} \\ t^n, n \in \{1,2,3,\ldots\} & \Longleftrightarrow \quad \frac{n!}{s^{n+1}} \\ \cos(at) & \Longleftrightarrow \quad \frac{s}{s^2+a^2} \\ \sin(at) & \Longleftrightarrow \quad \frac{a}{s^2+a^2} \end{array}$$

We can now use a Laplace transform table to simplify an example.

Example 2.4: Inverse Laplace Example

Find the inverse Laplace transform of the following equation:

$$F(s) = \frac{s+4}{s(s+2)}. \tag{2.14}$$

Before applying the transform table, we need to decompose $F(s)$ using partial fractions.

$$F(s) = \frac{2}{s} - \frac{1}{s+2}$$
$$\implies f(t) = \boxed{u(t)(2 - e^{-2t})}$$

In fact, our standard approach to find an inverse Laplace transform is to apply partial fractions and then use a table to invert each of the components.

2.3.2 Laplace Transform Properties

The Laplace transform has several properties that we will find relevant in this module. They are as follows:

1. **Linearity** - the Laplace transform of a sum of scaled functions is equal to the sum of scaled Laplace transforms of the individual functions, i.e.,

$$af_1(t) + bf_2(t) \Longleftrightarrow aF_1(s) + bF_2(s), \tag{2.15}$$

   which we have already applied in Example 2.4.

2. **Derivatives** - the Laplace transform of the derivative of a smooth function (i.e., no discontinuities) is as follows:

$$\mathcal{L}\{f'(t)\} = s\mathcal{L}\{f(t)\} - f(0) = \boxed{sF(s) - f(0)}. \tag{2.16}$$

   We have a proof in the following example.

Example 2.5: Proof of the Laplace Transform of the Derivative

Use integration-by-parts to find the Laplace transform of the derivative

of a function $f(t)$.

$$\begin{aligned}\mathcal{L}\{f'(t)\} &= \int_{t=0}^{\infty} f'(t)\, e^{-st} dt \\ &= e^{-st} f(t) \Big|_0^{\infty} + \int_{t=0}^{\infty} s e^{-st} f(t)\, dt \\ &= 0 - f(0) + s \int_{t=0}^{\infty} e^{-st} f(t)\, dt \\ &= sF(s) - f(0)\end{aligned}$$

We can also use recursion to find higher order derivatives, e.g.,

$$\mathcal{L}\{f''(t)\} = \mathcal{L}\left\{\frac{df'(t)}{dt}\right\} = \boxed{s^2 F(s) - sf(0) - f'(0)}$$

In general and ignoring initial conditions (i.e., at time $t = 0$), the Laplace transform of the nth order derivative is

$$f^n(t) \Longleftrightarrow \boxed{s^n F(s)}. \tag{2.17}$$

3. **Integrals** - the Laplace transform of the integral of a smooth function is as follows:

$$\int_{t=0}^{\infty} f(t)\, dt \Longleftrightarrow \boxed{\frac{F(s)}{s}}. \tag{2.18}$$

We omit the proof because we will generally find the Laplace transform of the derivative to be more useful than that of the integral.

4. **Delay (in both domains)** - delaying a signal in one domain corresponds to multiplying by an exponential in the other domain. More precisely, we have the following two transforms:

$$f(t - T) \Longleftrightarrow e^{-sT} F(s) \tag{2.19}$$

$$e^{at} f(t) \Longleftrightarrow F(s - a), \tag{2.20}$$

for constant time shift T or Laplace shift a.

2.4 Transfer Functions

Now that we are caught up on Laplace transforms, we can proceed to apply them to LTI systems. This is because there is another very important property about the

Laplace transform. The Laplace transform of the convolution of two functions is the product of the Laplace transforms of the two functions, i.e.,

$$\mathcal{L}\{x(t) * h(t)\} = \mathcal{L}\{x(t)\}\,\mathcal{L}\{h(t)\} = X(s)\,H(s). \tag{2.21}$$

In other words, **convolution in the time domain is equal to multiplication in the Laplace domain**. Recalling our convention that $x(t)$ is the system input and $h(t)$ is the system impulse response, we can combine this property with our understanding of LTI systems by writing the Laplace transform of the system output as

$$\mathcal{L}\{y(t)\} = Y(s) = X(s)\,H(s). \tag{2.22}$$

So, we can find the time-domain output $y(t)$ of an LTI system by

1. Transforming $x(t)$ and $h(t)$ into the Laplace domain.
2. Finding the product $Y(s) = X(s)\,H(s)$.
3. Taking the inverse Laplace transform of $Y(s)$.

Although this series of steps may seem difficult, it is often easier than doing convolution in the time domain. Furthermore, as we will see in the following lectures, we can gain a lot of insight about a system by analysing it in the Laplace domain. We refer to the Laplace transform $H(s)$ of the system impulse response as the system's **transfer function**.

Example 2.6: System Responses with Simple Inputs

There are two special cases of system response that we will regularly see.

1. When the input is a delta function, i.e., $x(t) = \delta(t)$, then we know that $X(s) = 1$ and so we immediately have $Y(s) = H(s)$. This can also be proven in the time domain, i.e., $y(t) = h(t)$, using convolution.
2. When the input is a step function, i.e., $x(t) = u(t)$, then we know that $X(s) = 1/s$ and so $Y(s) = H(s)/s$. This can be proven in the time domain from convolution and the integral property of the Laplace transform.

EXTRA 2.2: Proof that Convolution in the Time Domain is Equivalent to Multiplication in the Laplace Domain

We want to prove that $\mathcal{L}\{x(t) * h(t)\} = X(s)\,H(s)$. We will start with the definition of the Laplace transform of $y(t)$ and apply a change of variables.

$$\begin{aligned}
\mathcal{L}\{x(t) * h(t)\} = Y(s) &= \int_{t=0}^{\infty} e^{-st} \left(\int_{\tau=0}^{t} x(\tau)\, h(t-\tau)\, d\tau \right) dt \\
&= \int_{\tau=0}^{\infty} \int_{t=\tau}^{\infty} e^{-st} x(\tau)\, h(t-\tau)\, dt d\tau \\
&= \int_{\tau=0}^{\infty} \int_{\alpha=0}^{\infty} e^{-s(\alpha+\tau)} x(\tau)\, h(\alpha)\, d\alpha d\tau \\
&= \left(\int_{\tau=0}^{\infty} e^{-s\tau} x(\tau)\, d\tau \right) \left(\int_{\alpha=0}^{\infty} e^{-s\alpha} h(\alpha)\, d\alpha \right) \\
&= \boxed{X(s)\,H(s)},
\end{aligned}$$

where the swapping of integrals in the second line requires application of Fubini's theorem, and in the third line we define $\alpha = t - \tau$.

We can also re-arrange the product $Y(s) = X(s)\,H(s)$ to write the definition of the system transfer function $H(s)$ from the time-domain input and output, i.e.,

$$H(s) = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{x(t)\}}, \tag{2.23}$$

which is valid for a linear system when the initial conditions are zero. We commonly see this kind of formulation in circuit analysis, where the input and output are currents or voltages associated with the circuit. Deriving the system response of a circuit is outside the scope of this module, but you could be presented with a circuit and its corresponding system response and then asked to analyse it.

EXTRA 2.3: First Example of Circuit System Response

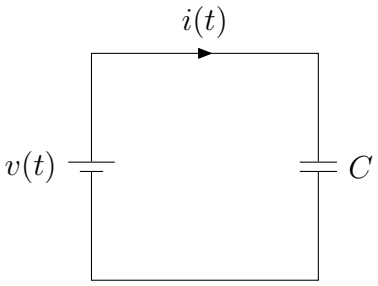


Figure 2.1: Simple capacitive circuit for Example 2.7.

Example 2.7: System Response of Simple Circuit

Consider the capacitive circuit in Fig. 2.1. What is the transfer function of this circuit? Assume that the input is the voltage $v(t)$ and the output is the current $i(t)$.

From circuit theory we (should) know that:

$$v(t) = \frac{1}{C}\int_0^t i(\tau)d\tau$$

$$\implies V(s) = \frac{I(s)}{sC}$$

$$\implies H(s) = \frac{I(s)}{V(s)} = \boxed{sC}$$

EXTRA 2.4: Second Example of Circuit System Response

Generally, we will find it helpful to know the voltage potential drops across the 3 common passive circuit elements in the Laplace domain. Given the current $I(s)$ through the element, the potential drop $V(s)$ across the element is:

1. $V(s) = RI(s)$ across resistance R.
2. $V(s) = sLI(s)$ across inductance L.

3. $V(s) = \frac{I(s)}{sC}$ across capacitance C.

Example 2.8: System Response of More Complex Circuit

Consider the capacitive circuit in Fig. 2.2. What is the transfer function of this circuit? Assume that the input is the voltage $v_i(t)$ and the output is the voltage $v_o(t)$.

Using result of the previous example we already have

$$V_o(s) = \frac{I(s)}{sC}.$$

From Kirchoff's circuit law we can also write

$$\begin{aligned} V_i(s) &= RI(s) + V_o(s) \\ &= RI(s) + \frac{I(s)}{sC} \\ \implies H(s) = \frac{V_o(s)}{V_i(s)} &= \frac{\frac{I(s)}{sC}}{RI(s) + \frac{I(s)}{sC}} \\ &= \boxed{\frac{1}{sRC+1}} \end{aligned}$$

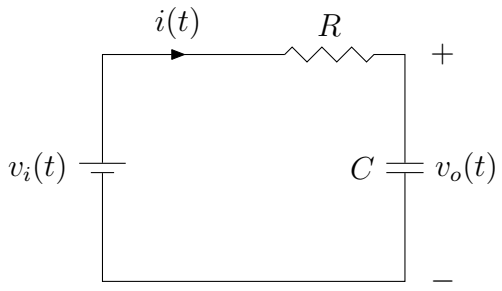


Figure 2.2: More complex capacitive circuit for Example 2.8.

2.5 Summary

- The output of a **linear time invariant** (LTI) system can be found by convolving the input with the system **impulse response**.
- We can use the Laplace transform to convert an LTI system into the Laplace domain, where the impulse response is known as the **transfer function**. Convolution in the time domain is equivalent to multiplication in the Laplace domain, so we can more readily find the output of a system in the Laplace domain.
- Laplace domain analysis is particularly helpful when analysing circuits.

2.6 Further Reading

- Sections 1.5, 1.10 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

2.7 End-of-Lesson Problems

1. Find the Laplace transform of the following functions:
   (a) $f(t) = 4t^2$
   (b) $f(t) = t\sin(2t)$
   (c) $f(t) = 2te^{2t} + t^2$
   (d) $f(t) = e^{3t}\cos(2t) + e^{3t}\sin(2t)$
   (e) $f(t) = e^{-3t}t\sin(2t)$
2. Find the inverse Laplace transform of the following functions:
   (a) $F(s) = \frac{1}{(s-2)^2+9}$
   (b) $F(s) = \frac{5s+10}{s^2+3s-4}$
3. Solve the differential equation $f''(t) - 3f'(t) + 2f(t) = 0$, given that $f(0) = 0$ and $f'(0) = 1$.
4. Use Euler's formula and the transform $e^{at} \Longleftrightarrow \frac{1}{s-a}$ to prove that the Laplace transform of $\cos(\omega t)$ is $\frac{s}{s^2+\omega^2}$

5. Use the Laplace transform of the first order derivative of a function to prove that the Laplace transform of the second order derivative is $\mathcal{L}\{f''(t)\} = s^2F(s) - sf(0) - f'(0)$

6. Find the time-domain output $y(t)$ of an LTI system defined by transfer function
$$H(s) = \frac{1}{1+\tau s}$$
when the input $x(t)$ is i) A step function, and ii) A ramp function, i.e., $x(t) = t$.

7. **(EXTRA Problem)** Find the transfer function $H(s) = \frac{V_o(s)}{V_i(s)}$ for the circuit in Figure 2.3

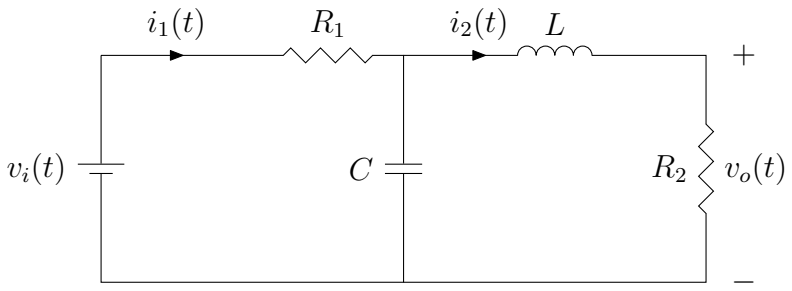


Figure 2.3: A two-loop circuit.

8. **(EXTRA Problem)** Find the transfer function $H(s) = \frac{V_o(s)}{V_i(s)}$ for the circuit in Figure 2.4

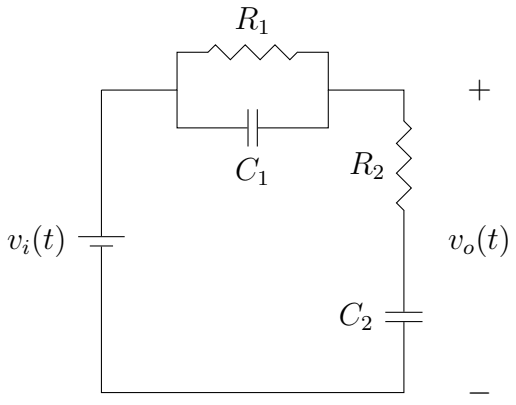


Figure 2.4: A second order filter circuit.

9. **(EXTRA Problem)** Assume that the operational amplifier in Fig. 2.5 is perfect. Find the transfer function $H(s) = \frac{V_o(s)}{V_i(s)}$ for the circuit.

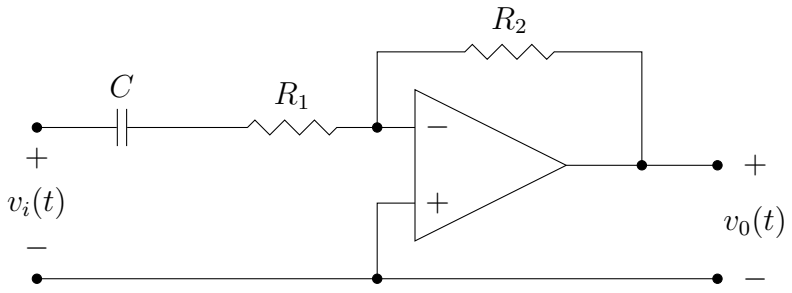


Figure 2.5: Op-amp circuit.

Lesson 3

Poles, Zeros, and Stability of Analogue Systems

System stability is a very important consideration in system design, as we will see in future lessons throughout this module. In this lesson, we consider the stability of analogue systems. We will see that Laplace domain analysis makes this possible by looking at the roots of the system's transfer function.

3.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Use** an analogue system's transfer function to identify its poles and zeroes.
2. **Apply** the criterion for analogue system stability.
3. **Analyse** the dynamic response of an analogue system with real poles.
4. **Analyse** the dynamic response of an analogue system with complex poles.

3.2 Poles and Zeros

Generally, a transfer function $H(s)$ in the Laplace domain can be written as a fraction of two polynomials, i.e.,

$$H(s) = \frac{b_0 s^M + b_1 s^{M-1} + \ldots + b_{M-1} s + b_M}{a_0 s^N + a_1 s^{N-1} + \ldots + a_{N-1} s + a_N}, \tag{3.1}$$

where the numerator is a Mth order polynomial with coefficients bs, and the denominator is a Nth order polynomial with coefficients as. For a system to be real,

then the order of the numerator polynomial must be no greater than the order of the denominator polynomial, i.e., $M \leq N$.

EXTRA 3.1: Border Case of Being "Real"

If the order of the numerator polynomial is greater than the order of the denominator polynomial, i.e., if $M > N$, then the LTI system output will grow unbounded with increasing frequency (we can prove this with what we cover in Lesson 4). This is not physically possible, which is why we consider an LTI system to not be real *in this specific context*, since a system can also be unreal due to anti-causality. Furthermore, *strictly speaking*, an LTI system with transfer function polynomials of equal order, i.e., where $M = N$, is *also* not real, because LTI system output cannot be constant with arbitrary increasing frequency. However, this special case would force us to refer to some ideal filters (which we cover in Lesson 5), such as high pass and band stop filters, as unreal, even though they have (ideal) circuit implementations available.

We are not going to focus on whether LTI systems are real or unreal because of their transfer function polynomials, but this discussion is provided to note the ambiguity.

Eq. 3.1 may look intimidating, but it is general. If we factorize Eq. 3.1 into its roots, then we can re-write $H(s)$ as a ratio of products, i.e.,

$$H(s) = K\frac{(s-z_1)(s-z_2)\dots(s-z_M)}{(s-p_1)(s-p_2)\dots(s-p_N)} \tag{3.2}$$

$$= K\frac{\prod_{i=1}^{M} s - z_i}{\prod_{i=1}^{N} s - p_i} \tag{3.3}$$

where we emphasise that the coefficient on each s is 1. Eq. 3.1 has several key definitions:

- The roots z of the numerator are called the **zeros**. When s is equal to any z_i, the transfer function $H(s) = 0$.
- The roots p of the denominator are called the **poles**. When s is equal to any p_i, the transfer function $H(s)$ will be infinite (and we will soon see that this relates to stability...)
- K is the overall transfer function **gain**.

Note: later in this module we will see the z-transform. It will be important to not confuse it with the zeros of a transfer function. Unfortunately, due to the names of the terms, re-use of variables is unavoidable here.

We will often find it useful to plot the poles and zeros of a transfer function on the complex domain of s, which we call a **pole-zero plot**. The convention is to plot zeros as circles (o for 0, i.e., don't fall in the hole!) and poles as "x"s (i.e., stay away from these!).

Example 3.1: First Pole-Zero Plot

Plot the pole-zero plot for the transfer function

$$H(s) = \frac{s+1}{s}$$

First we need to figure out the poles and zeros, i.e., the roots of the transfer function. This transfer function is already in a form where the roots are factorized. We can see that the numerator is zero when $s = -1$, so $z - 1$ and the denominator is zero when $s = 0$, so $p = 0$. We also see that the gain is $K = 1$.

We need to draw a 2D axes, where the real components of s lie along the horizontal axis and the imaginary components of s lie along the vertical axis. The plot is shown in Fig. 3.1.

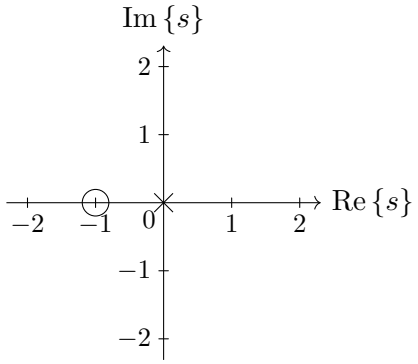


Figure 3.1: Pole-zero plot for Example 3.1.

More generally, poles and zeros are complex, i.e., they do not need to lie on the horizontal axis of the s-plane. In the following, we focus on the features of analogue systems with real and complex poles.

3.3 Poles and Dynamic Responses

3.3.1 Real Poles

We will see that the poles of a system determine the general form of its response. First, consider a system where all of the poles are *real*. If we consider a step input to the system, then from the previous lesson we know what we can write the output in the Laplace domain as

$$Y(s) = \frac{1}{s} H(s) \tag{3.4}$$

$$= \frac{1}{s} \times \frac{K \times \text{Zero Polynomial}}{(s-p_1)(s-p_2)(s-p_3)\dots} \tag{3.5}$$

$$= \frac{\text{some constant}}{s} + \frac{\text{some constant}}{s-p_1} + \frac{\text{some constant}}{s-p_2} + \dots, \tag{3.6}$$

where the numerators will all be constants that depend on the form of the numerator polynomial (i.e., zero polynomial). By partial fraction expansion, the output $Y(s)$ is a sum of fractions where each denominator is a linear function of s. Since we've assumed that each pole is real, we can apply the transforms $1 \iff \frac{1}{s}$ and $e^{at} \iff \frac{1}{s-a}$ and see that the time-domain output signal $y(t)$ is scaled constant plus a sum of (hopefully decaying!) exponentials. In this particular example, the scaled constant came from the input; more generally, we will see that the input dominates the first term of the output.

Thus, if we take the inverse Laplace transform of *only* the terms associated with the poles, then we have

$$y_{\text{poles}}(t) = [\text{some constant}]e^{p_1 t} + [\text{some constant}]e^{p_2 t} + \dots \tag{3.7}$$

The main takeaways for systems with real poles are as follows:

- The system input will (hopefully) dominate the output.
- Each real pole corresponds to an exponential component in the time domain with a rate that depends on that pole.
- The magnitude of each exponential component depends on the zeros.

Example 3.2

Consider an LTI system with the impulse response

$$h(t) = \frac{1}{2} e^{-t}(e^{2t} - 1),$$

which we will assume is causal (i.e., $h(t) = 0$ for $t < 0$). Answer the following:

1. Find the system's poles and zeros and sketch a pole-zero plot.
2. Find the dynamic response in the time domain, subject to a step input.
3. Sketch the dynamic response and comment on the role of the poles and zeros.

We can re-write the impulse response as

$$h(t) = \frac{e^t}{2} - \frac{e^{-t}}{2}.$$

The Laplace transform gives the transfer function:

$$H(s) = \frac{1}{2(s-1)} - \frac{1}{2(s+1)} = \frac{1}{(s-1)(s+1)}.$$

Thus, there are no zeros and 2 poles at $p = \pm 1$. Also, the gain is $K = 1$. See the pole-zero plot in Fig. 3.2.

If there is a step input, then we know $X(s) = \frac{1}{s}$ and hence the output is

$$\begin{aligned} Y(s) &= \frac{1}{s} H(s) \\ &= \frac{1}{s} \frac{1}{(s-1)(s+1)} \\ &= \frac{1}{2(s-1)} - \frac{1}{s} + \frac{1}{2(s+1)} \\ \implies y(t) &= \boxed{-1 + \frac{e^{-t}}{2} + \frac{e^t}{2}} \end{aligned}$$

A plot of this output is shown in Fig. 3.3. We see that the $+1$ exponent dominates for time $t > 0$ (in fact this system is unstable; we will soon see why), and the -1 exponent dominates for $t < 0$ (but the system is causal so $y(t) = 0$ before an input is applied).

3.3.2 Complex Poles

Now let us consider a system with complex poles. Rather than write the most general form, let us jump right to a more specific example and consider the transfer

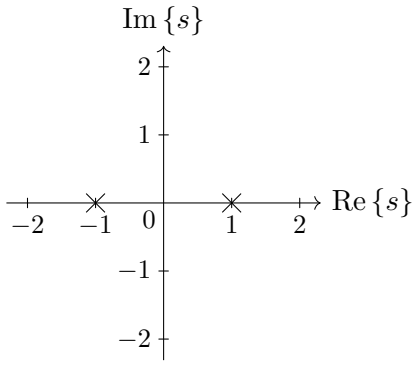


Figure 3.2: Pole-zero plot for Example 3.2.

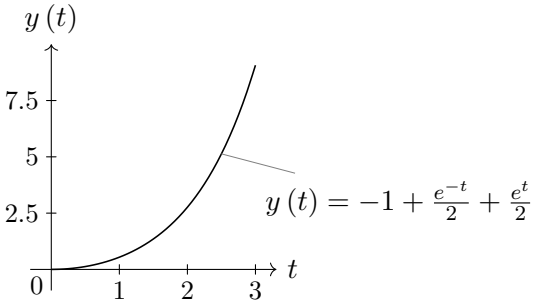


Figure 3.3: Dynamic response for Example 3.2.

function

$$H(s) = \frac{As + B}{(s+\alpha)^2 + \beta^2}, \tag{3.8}$$

for real constants α and β. In this case, the poles exist on the complex s-plane at $s = -\alpha \pm j\beta$. We see that there are two poles, and in fact they are a **conjugate pair**. In practice, any poles that we will see with an imaginary component will be part of a conjugate pair. What is more interesting, however, is the impact of complex poles on the system response. Let us find the inverse Laplace transform of $Y(s)$ by first finding suitable numerators:

$$H(s) = \frac{As + B}{(s+\alpha)^2 + \beta^2} = \frac{A(s+\alpha)}{(s+\alpha)^2 + \beta^2} + \frac{C\beta}{(s+\alpha)^2 + \beta^2}, \tag{3.9}$$

where we have introduced the constant $C = \frac{B - A\alpha}{\beta}$. We can now take the inverse Laplace transform as

$$h(t) = Ae^{-\alpha t}\cos(\beta t) + Ce^{-\alpha t}\sin(\beta t) = De^{-\alpha t}\sin(\beta t + \phi), \tag{3.10}$$

where $D = \sqrt{A^2 + C^2}$ and $\tan\phi = \frac{A}{C}$. The important result here is that *complex poles lead to an oscillatory response* in the time domain. The real component of the pole $(-\alpha)$ corresponds to the power of the exponent, and the imaginary component of the pole (β) corresponds to the frequency of oscillation. A critical parameter, then, is the value of α, or more specifically the *sign*. If $\alpha > 0$, then the exponent $e^{-\alpha t}$ will decay with time. If $\alpha < 0$, then the exponent $e^{-\alpha t}$ will grow with time. At the transition is where $\alpha = 0$ and the response stays the same. We visualize these three cases in Fig. 3.4.

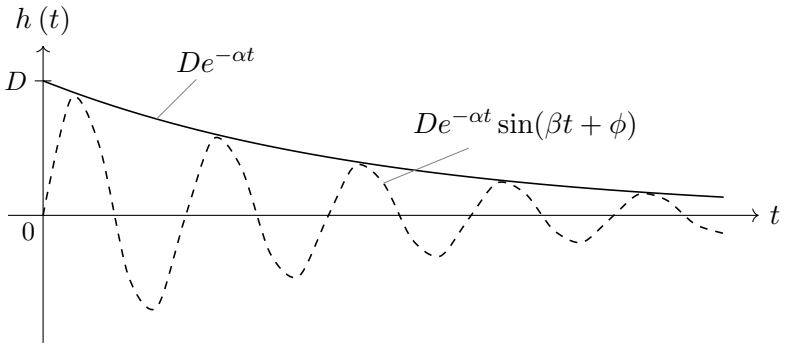
3.3.3 Stability Criterion for Analogue Systems

The preceding discussion about poles brought us very close to discussing stability. In fact, we should now define it. A system is considered to be **stable** if its impulse response tends to zero in the time domain. If the impulse response does not tend to zero or some finite value, then the system is **unstable**. We have seen that this depends on the locations of the poles, so we can use the poles to define a stability criterion for a system. In particular, if all of a system's poles are on the left half of the complex s-plane (i.e., to the left of the imaginary s-axis), then the system is stable. We see this criterion visually in Fig. 3.5. Any pole satisfying this condition corresponds to a decaying exponential in the time domain.

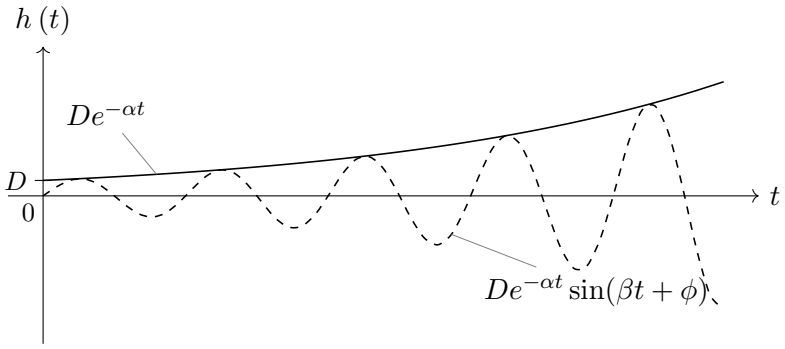
It is worth mentioning what to do when a system has a pole that is *on* the imaginary s-axis. Is such a system stable? From Fig. 3.5, we might be inclined to say yes. However, as we can see in Fig. 3.4(c), such a system does not satisfy our definition of stability, as the impulse response does not tend to zero over time. Such a system is said to be **marginally stable**.

EXTRA 3.2: Alternative Definitions of Stability

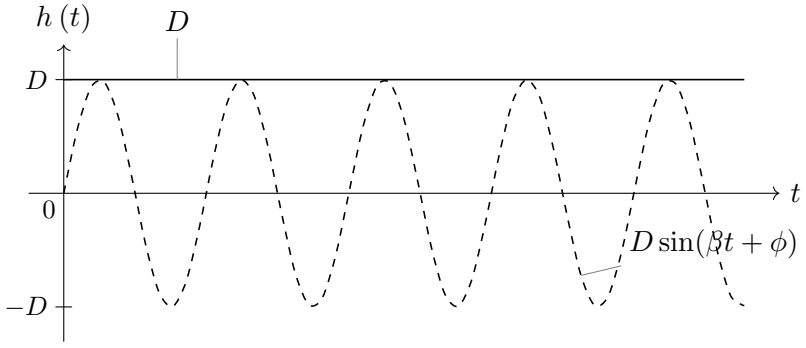
Lathi and Green define stability from a slightly different perspective. They call a system **bounded-input bounded-output (BIBO) stable** if and only if every bounded (finite) input $x(t)$ results in bounded output $y(t)$. A system is **asymptotically stable** if bounded initial conditions result in bounded output. In practice for most systems, these two definitions are *equivalent* and also equal to our definition based on the impulse response. Under all 3 definitions, a system whose transfer function's poles are all to the left of the imaginary s-axis is stable and having poles to the right of the imaginary s-axis makes the system unstable. The details vary for poles *on* the imaginary s-axis.



(a) $\alpha > 0$



(b) $\alpha < 0$



(c) $\alpha = 0$

Figure 3.4: Impulse response of system with complex poles for different values of α.

3.4 Summary

- The **zeros** are the roots to the transfer function's numerator in the Laplace domain.
- The **poles** are the roots to the transfer function's denominator in the Laplace domain.

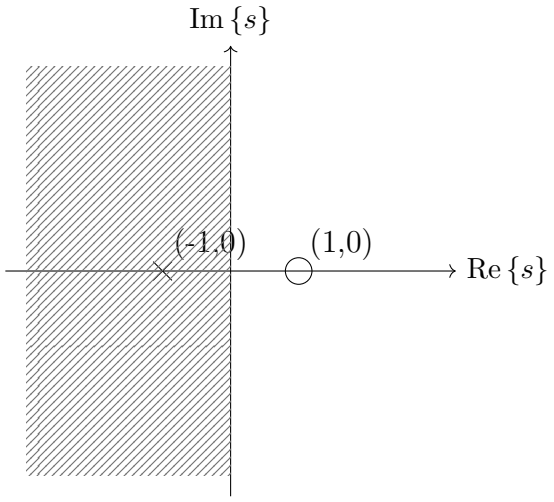


Figure 3.5: Pole-zero plot showing the stability region for poles with shaded cross hatches.

- A system is **stable** if the impulse response tends to zero with increasing time.
- Real components of poles correspond to exponential responses.
- Imaginary components of poles correspond to the angular frequency of oscillating responses.

3.5 Further Reading

- Sections 1.5.5 and 2.1.1 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014. Note that Lathi and Green use a slightly different definition of stability than what we use in this module. Our definition for stability in analogue systems is more convenient to use in consideration of time-domain behaviour.

3.6 End-of-Lesson Problems

1. Determine the zeros and poles of the system with transfer function
$$H(s) = \frac{(1+0.04s)(1+11s)}{0.44s^2 + 13.04s + 1}.$$
2. Consider a system with transfer function
$$H(s) = \frac{20}{0.03s^2 + 1.4s + 20}.$$

Determine the poles of this system. Sketch the time-domain output of the system in response to a step input. Is this system stable?

3. Determine the s-domain transfer function $H(s)$ of the system with the pole-zero plot shown in Fig. 3.6. Is this system stable?

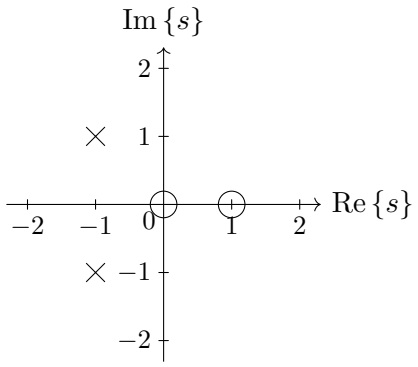


Figure 3.6: Pole-zero plot for Question 3.

4. Determine the poles of each of the following transfer functions and state whether they represent a stable system:

   (a) $H(s) = \frac{1}{s-1}$

   (b) $H(s) = \frac{\omega}{s^2+\omega^2}$

   (c) $H(s) = \frac{3}{s^2+2s+10}$

Lesson 4

Analog Frequency Response

In Lesson 3, we saw how a system's pole-zero plot enabled us to describe time-domain characteristics of the system, including its stability. In this lesson, we will focus on the behaviour of systems in response to periodic inputs of a particular frequency. To do so, we will find it convenient to translate between the Laplace transform and the Fourier transform. Fortunately, this translation is straightforward and we can also use the Laplace-domain pole-zero plot to determine the frequency response of a system. Being able to analyse the frequency response of a system is a prerequisite for studying and designing analogue filters, as we will see in Lesson 5.

4.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Apply** knowledge of the Laplace transform to find the frequency response of an input to an analogue system.
2. **Understand** the Fourier transform and its relation to the Laplace transform.

4.2 Frequency Response

The **frequency response** of a system is its output in response to a sinusoid input of unit magnitude and some specified frequency. The frequency response is measured as 2 components: a **magnitude** and a **phase angle**. Typically, we show the frequency response in two plots (1 for magnitude; 1 for phase) as a function of the input frequency.

Generally, a system may be given inputs at different frequencies, but it is *common when we design a system to do so for a particular target frequency or range of frequencies*. Some examples:

- Telephone technologies are designed for processing the frequency ranges commonly used by a human voice.
- Household appliances are designed to operate with a mains frequency of about 50Hz (in Europe; 60Hz in North America, etc.)
- Different radio frequency bands are licensed (or open) to different technologies. Wifi commonly operates at 2.4GHz and 5GHz; other bands are reserved for specific use such as television transmissions or the military.

To find the frequency response, recall the definition of the Laplace transform:

$$F(s) = \int_{t=0}^{\infty} f(t)\, e^{-st} dt, \tag{4.1}$$

for $s = \sigma + j\omega$. Let us *ignore the real component* of s and only consider the Laplace transform on the imaginary s-axis, such that $s = j\omega$. We can then re-write the Laplace transform as

$$F(j\omega) = \int_{t=0}^{\infty} f(t)\, e^{-j\omega t} dt. \tag{4.2}$$

We emphasise that ω is the radial frequency, measured in $\frac{\text{rad}}{\text{s}}$, and we can convert to and from the frequency f in Hz via $\omega = 2\pi f$.

The special case of the Laplace transform in Eq. 4.2 is better known as the **continuous Fourier transform** and the result $F(j\omega)$ is referred to as the **spectrum** of $f(t)$ or as the **frequency response**. Similarly, we can also write the inverse of the continuous Fourier transform as

$$f(t) = \frac{1}{2\pi} \int_{\omega=-\infty}^{\infty} F(j\omega)\, e^{j\omega t} d\omega. \tag{4.3}$$

We recall from Lesson 3 that we wrote a system's transfer function as a ratio of products that included its poles and zeros. We can re-write with $s = j\omega$ as

$$H(j\omega) = K \frac{\prod_{i=1}^{M} j\omega - z_i}{\prod_{i=1}^{N} j\omega - p_i}. \tag{4.4}$$

This form of $H(j\omega)$ (which we may also simply write as $H(\omega)$) is incredibly useful for determining the frequency response of a system. We can treat $H(j\omega)$ as a function of vectors from the system's poles and zeros to the imaginary $j\omega$-axis (i.e., each pole and zero corresponds to 1 vector; effectively a line from that pole or zero to the $j\omega$-axis at frequency ω). Thus, from a pole-zero plot, we can *geometrically* determine the magnitude and phase of the frequency response.

Recall that the phasor form of a complex number separates it into its magnitude and phase, i.e.,

$$H(j\omega) = |H(j\omega)|e^{j\angle H(j\omega)}, \tag{4.5}$$

such that the magnitude component has a phase of 0 and the phase component has a magnitude of 1. First we consider the magnitude. We can take the magnitude of both sides of Eq. 4.4 as follows:

$$|H(j\omega)| = |K|\frac{\prod_{i=1}^{M}|j\omega - z_i|}{\prod_{i=1}^{N}|j\omega - p_i|}. \tag{4.6}$$

In words, the **magnitude of the frequency response** (MFR) $|H(j\omega)|$ is equal to *the gain multiplied by the magnitudes of the vectors corresponding to the zeros, divided by the magnitudes of the vectors corresponding to the poles.*

The phase depends on the sign of K. If $K > 0$, then

$$\angle H(j\omega) = \sum_{i=1}^{M}\angle(j\omega - z_i) - \sum_{i=1}^{N}\angle(j\omega - p_i). \tag{4.7}$$

If $K < 0$, then it effectively adds a phase of π and we have

$$\angle H(j\omega) = \sum_{i=1}^{M}\angle(j\omega - z_i) - \sum_{i=1}^{N}\angle(j\omega - p_i) + \pi. \tag{4.8}$$

In words, the **phase angle of the frequency response** (PAFR) $\angle H(j\omega)$ is equal to *the sum of the phases of the vectors corresponding to the zeros, minus the sum of the phases of the vectors correspond to the poles, plus π if the gain is negative.* Each phase vector is measured from the positive real s-axis (or a line parallel to the real s-axis if the pole or zero is not on the real s-axis).

We will now look at examples of frequency responses.

4.3 Frequency Response Examples

Example 4.1: First Order Low Pass Filter

We will consider filters in greater detail in Lesson 5, but for now consider this transfer function of a simple low pass filter:

$$H(s) = \frac{1}{1+\tau s}.$$

We first re-write $H(s)$ to match the form of Eq. 4.4 so that the coefficients

of all roots are equal to 1:

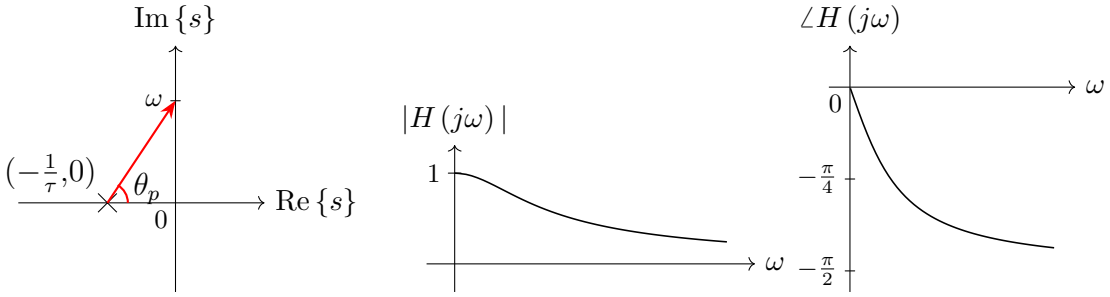
$$H(s) = \frac{1/\tau}{1/\tau + s},$$

so the gain is $K = \frac{1}{\tau}$ and there is a pole at $s = -\frac{1}{\tau}$. We sketch the pole-zero plot and draw a vector from the pole to an arbitrary (usually positive) ω in Fig. 4.1(a).

Consider what happens to the vector as we vary ω. The vector has a minimum magnitude when $\omega = 0$ and rises *monotonically* (i.e., sign doesn't change) with increasing (or decreasing) ω. So, the magnitude of this vector *increases* with ω, but since it is a pole the magnitude of the *system decreases* with ω.

The magnitude of the frequency response is shown in Fig. 4.1(b). Note that the magnitude of the pole vector at frequency $\omega = 0$ is $1/\tau$, but $K = \frac{1}{\tau}$, so the overall system magnitude is 1. This system is a **low pass filter** because it "*passes*" a strong signal (i.e., higher magnitude) at lower frequencies and passes a weak signal (i.e., *attenuates*) at higher frequencies.

The phase of the pole vector also increases monotonically with the frequency, but it converges to a maximum of $\pi/2$. Thus, the phase of the system is a negative monotonic function and is shown in Fig. 4.1(c).



(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

Figure 4.1: Plots for Example 4.1

Example 4.2: Second Order Low Pass Filter

Consider a system with the transfer function

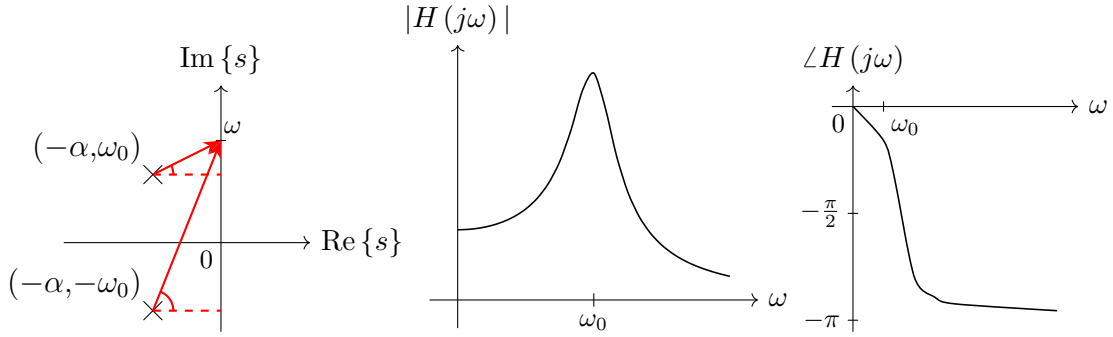
$$H(s) = \frac{1}{s^2 + 2\alpha s + (\alpha^2 + \omega_0^2)},$$

for positive real constants α and ω_0. This transfer function has no zeros, but a pair of complex conjugate poles at $s = -\alpha \pm j\omega_0$. There are now two pole vectors to deal with and they do not lie on the real s-axis. The corresponding pole-zero plot is in Fig. 4.2(a).

The magnitude of the 2 pole vectors has a minimum value at $\omega = \pm\omega_0$ (or just $\omega = \omega_0$ when assuming $\omega > 0$). The pole magnitude rises monotonically as the frequency moves away from ω_0. The frequency response magnitude of the system will do the opposite and decrease as we move away from ω_0, as shown in Fig. 4.2(b). However, this is *still* a **low pass filter** because the frequency response magnitude does not tend to 0 as $\omega \to 0$ whereas it does tend to 0 as $\omega \to \infty$. The filter is second order because the denominator polynomial is second order.

By symmetry, the phase of the frequency response is 0 at $\omega = 0$. As ω increases, the angle associated with one pole vector increases while the other decreases, which results in a slow net decrease of the system phase, until both vector angles increase and the system phase decreases to $-\pi$, as shown in Fig. 4.2(c).

We have not used actual numbers for the pole coordinates in this example, but you will see if you try substituting different values of α that a smaller α results in a more pronounced peak in the magnitude of the frequency response.



(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

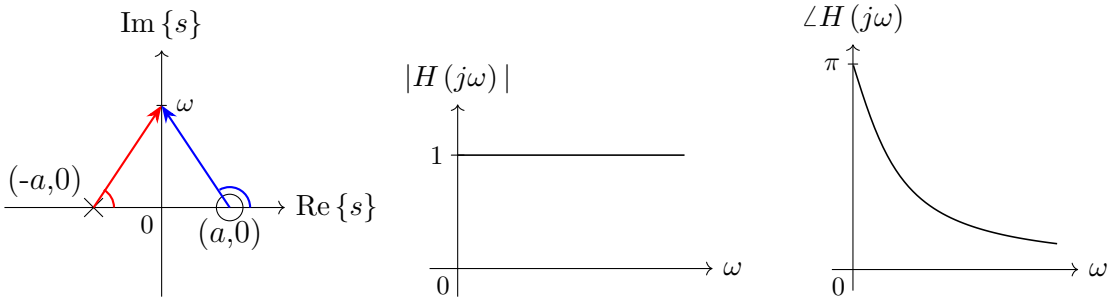
Figure 4.2: Plots for Example 4.2

Example 4.3: All Pass Filter

Consider the following filter transfer function:

$$H(s) = \frac{s-a}{s+a},$$

for positive real constant a. The pole-zero plot is shown in Fig. 4.3(a). By symmetry of the pole and zero, the pole vector will always have the same magnitude as the zero vector for any ω. Thus, the magnitude of the frequency response is always 1, as shown in Fig. 4.3(b). This is known as an **all pass filter**. While the magnitude does not change, the phase does; the zero vector has a phase of π when $\omega = 0$ whereas the pole vector has a phase of 0, then the phases of both vectors tend to $\pi/2$ with increasing ω such that their difference tends to 0. This is shown in Fig. 4.3(c).



(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

Figure 4.3: Plots for Example 4.3

4.4 Summary

- The **frequency response** of a system is its response to an input with unit magnitude and a fixed frequency. The response is given by the magnitude and phase of the system output as a function of the input frequency.
- The **continuous Fourier transform** is the Laplace transform evaluated on the imaginary s-axis at some frequency $s = j\omega$.
- The frequency response can be found geometrically by assessing the vectors made by the poles and zeros of the system's transfer function to the imaginary

s-axis.

4.5 Further Reading

- Section 2.1 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

4.6 End-of-Lesson Problems

Note: For additional practice, you can try plotting the frequency response for any Laplace-domain transfer function in this part of the notes. As we will see in Lesson 7, all results can be readily verified in MATLAB.

1. Sketch the pole-zero plot, magnitude of the frequency response, and phase of the frequency response of the system with the transfer function:
$$H(s) = \frac{s}{s+1}.$$

2. Sketch the pole-zero plot, magnitude of the frequency response, and phase of the frequency response of the system with the transfer function:
$$H(s) = \frac{1}{s^2+3s+2}.$$

3. Consider the all pass filter from Example 4.3. The transfer function is
$$H(s) = \frac{s-a}{s+a}.$$
for real positive constant a. Prove mathematically that the magnitude of the frequency response is always 1, and that the phase of the frequency response is given by
$$\angle H(j\omega) = \pi - 2\tan^{-1}\frac{\omega}{a}.$$

4. Sketch the pole-zero plot, magnitude of the frequency response, and phase of the frequency response of the system with the transfer function:
$$H(s) = \frac{s+0.5}{s^2+4s+8}.$$

5. You are asked to design a stable LTI system that has a transfer function gain of $K = 2$ and a gain of $0\,\text{dB} = 1$ at $\omega = 0$ (Note: conversion between dB and linear gains will be covered in the next Lesson). The system should have one pole and one zero. If there is a zero placed at $s = 1$, then where should the pole be placed?

Lesson 5

Analogue Filter Design

In Lesson 4 we started to talk about some types of simple filters and their frequency responses. Filtering is a very common use of signal processing and practically any signal processing system will include some form of filtering. In this lesson, we focus on some of the practical details of filter design and how to design an analogue filter to meet target specifications (i.e., how to *engineer* a filter ... did you wonder when engineering would appear in this module?).

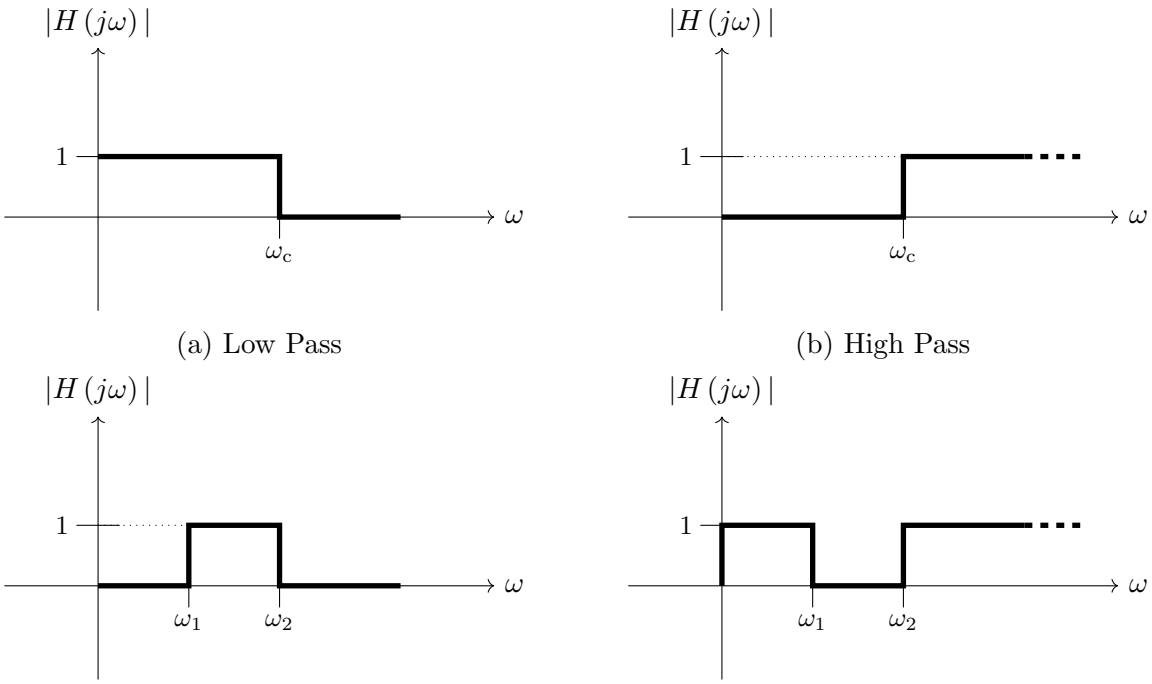
5.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the basic classifications of analogue filters and common filter designs.
2. **Understand** the constraints on filter realisability.
3. **Design** analogue filters to meet specified performance criteria.

5.2 Ideal Filter Responses

Our discussion of analogue filter design begins with ideal filter responses, as we show in Fig. 5.1. This enables us to define the different types of filters and helps us to establish the targets that we will try to realise with practical filters. Each ideal filter has unambiguous **pass bands**, which are ranges of frequencies that pass through the system without distortion, and **stop bands**, which are ranges of frequencies that are rejected and do not pass through the system without significant loss of signal strength. The **transition band** between stop and pass bands in ideal filters has a size of 0; transitions occur at single frequencies.



(a) Low Pass

(b) High Pass

(c) Band Pass

(d) Band Stop

Figure 5.1: Frequency response magnitudes of common ideal analogue filters.

The four main types of filter magnitude responses are as follows:

- **Low pass** filters pass frequencies less than cutoff frequency ω_c and reject frequencies greater than ω_c.
- **High pass** filters reject frequencies less than cutoff frequency ω_c and pass frequencies greater than ω_c.
- **Band pass** filters pass frequencies that are within a specified range, i.e., between ω_1 and ω_2, and reject frequencies that are either below or above the band.
- **Band stop** filters reject frequencies that are within a specified range, i.e., between ω_1 and ω_2, and pass all other frequencies.

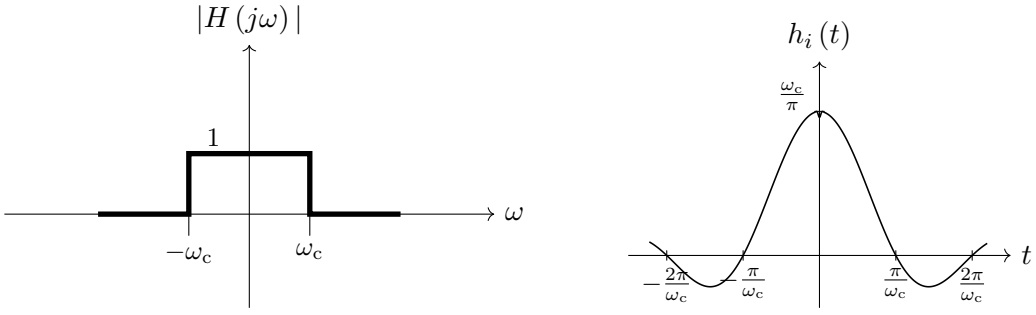
Unfortunately, none of these filters are realisable. We can show this mathematically in a ***very*** important example.

Example 5.1: Realisability of an Ideal Filter

Consider an ideal low pass filter, which acts as a rectangular pulse in the Laplace domain. We will focus on its double-sided frequency response, so we consider that it passes signals within the range $-\omega_c < \omega < \omega_c$. Is it realisable? Let's apply the inverse Fourier transform to determine the impulse response.

$$\begin{aligned} h(t) &= \frac{1}{2\pi}\int_{\omega=-\infty}^{\infty} H(j\omega)\, e^{j\omega t} d\omega \\ &= \frac{1}{2\pi}\int_{\omega=-\omega_c}^{\omega_c} e^{j\omega t} d\omega \\ &= \frac{1}{2\pi jt} e^{j\omega t}\Big|_{\omega=-\omega_c}^{\omega=\omega_c} \\ &= \frac{1}{2\pi jt}\left[e^{j\omega_c t} - e^{-j\omega_c t}\right] \\ &= \frac{1}{\pi t}\sin(\omega_c t)\frac{\omega_c t}{\omega_c t} \\ &= \frac{\omega_c}{\pi}\,\mathrm{sinc}(\omega_c t). \end{aligned}$$

Thus the impulse response is a scaled sinc function, which has non-zero values for $t < 0$ as shown in Fig. 5.2. This means that the system starts to respond to an input *before* that input is applied, so this filter is $\boxed{\textit{unrealisable}}$. In fact, none of the ideal filters are realisable.



(a) Frequency response magnitude. (b) Impulse Response.

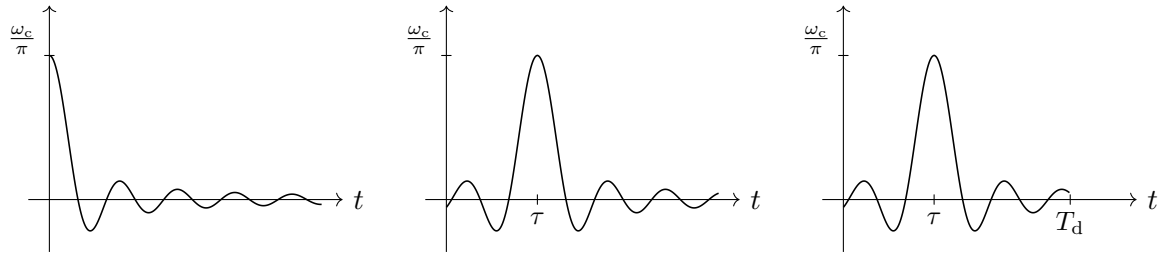
Figure 5.2: Responses of ideal low pass filter in Example 5.1.

5.3 Realising Filters

So, how to realise a practical filter? Even if we cannot implement an ideal filter, we seek smooth behaviour in the pass bands and steep transitions to the stop bands. We have seen simple realisations of filters in Lesson 4, but their frequency responses deviated quite significantly from the ideal filter behaviour. It turns out that we can do much better with filters that are not only (theoretically) realisable but also practical to implement with circuits.

First, let's consider what we could do to the impulse response in Fig. 5.2(b) to make it realisable. If we just dropped the portion of $h_i(t)$ for $t < 0$, i.e., used $h_i(t)\,u(t)$ then we would not get suitable behaviour in the frequency domain because we have discarded 50% of the system energy. However, if we are able to tolerate delays, then we can shift the sinc function to the right so that more of the energy is in causal time, i.e., use $h_i(t-\tau)\,u(t)$. As we recall from Laplace transforms, a shift in the time domain corresponds to scaling by a complex exponential in the Laplace domain. This is also true for the Fourier transform. So, *a delay in time maintains the magnitude of the frequency response* but it changes the phase.

Is waiting sufficient to realise a filter? If we can wait forever for our filter output, then yes. But in practice we will have a **delay tolerance** T_d that we are willing to wait for signal output, which means that we need to further **truncate** our impulse response $h_i(t)$ so that it is finite, i.e., $h_i(t-\tau)\,u(t) = 0$ for $t > T_\mathrm{d}$. We show the combined effects of causality, shifting, and truncating in Fig. 5.3.



(a) Causal response $h_i(t)\,u(t)$

(b) Shifted response $h_i(t-\tau)\,u(t)$.

(c) Truncation of shifted response.

Figure 5.3: Realising a filter by transforming unrealisable behaviour.

Are we done? Not quite. Truncation can cause unexpected problems in the frequency domain, so we also usually apply **windowing**, which scales the non-truncated impulse response to improve the frequency response behaviour. We won't worry about windowing for analogue systems, but we will re-visit this idea for digital systems. A more serious issue is the actual physical implementation of a filter with a truncated impulse response, which do not generally produce a *rational* transfer function $H(s)$. However, fortunately, there are families of practical filters that have

rational transfer functions, which make them implementable with standard electric circuit components. We will see some examples of these in the following section.

5.4 Practical Filter Families

Before we discuss specific practical filters, we need to define terminology that is used to set and describe practical filter performance. For the rest of this lesson, we focus on low pass filters, but the same ideas apply to the other types of filter as well. A practical filter tends to have a frequency response magnitude in the form of that in Fig. 5.4. The main characteristics are the following:

- There tends to not be a true stop band, as there is always some non-zero magnitude for any finite frequency. So, we define a small maximum **stop band gain** G_s, such that the gain in a stop band is less than or equal to this value. The threshold frequency that needs to satisfy this constraint is the **stop band frequency** ω_s.

- The magnitude of the frequency response is not actually flat over the entire pass band. So, we define a minimum **pass band gain** G_p such that the gain in a pass band is greater than or equal to this value. The threshold frequency that needs to satisfy this constraint is the **pass band frequency** ω_p.

- There is a non-zero gap between the pass band frequency and the stop band frequency that we call the **transition band**. We typically want this to be as steep as possible, but in generally to do so also increases the complexity of the filter and its implementation.

The gain and frequency characteristics above are often used as design **constraints**, i.e., to form a specification, that a given filter design must satisfy. It would be uncommon for all constraints to be met exactly, i.e., for the gain to be precisely the stop band gain at the stop band frequency while also precisely the pass band gain at the pass band frequency. A designer might "over-deliver" at one band in order to satisfy the requirements at another band. This kind of intuition about filter design is best developed through practice.

Typically, the gains G_p and G_s are defined on a logarithmic scale in decibels (dB). We convert between a linear gain and dB gain as follows:

$$G_\mathrm{dB} = 20 \log_{10} G_\mathrm{linear} \tag{5.1}$$

$$G_\mathrm{linear} = 10^{\frac{G_\mathrm{dB}}{20}}. \tag{5.2}$$

Note: the transfer function gain K that appears in the pole-zero factorization form of $H(s)$ is *different* from the filter gain G that refers to the magnitude of the

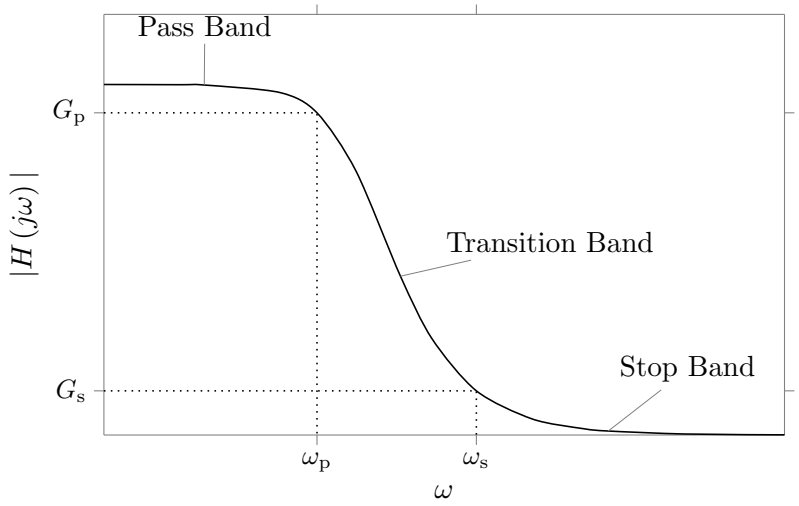


Figure 5.4: Frequency response magnitude of realizable analogue low pass filter.

frequency response at a particular frequency. Unfortunately, naming conventions mean that we need to refer to both of them as gains.

5.4.1 Butterworth Filters

A common filter design is the **Butterworth filter**. Butterworth filters are *maximally flat* in the pass band, i.e., their frequency response magnitudes are as flat as possible for the given order of the filter. The transfer function of an Nth order Butterworth low pass filter is

$$H(s) = \frac{\omega_\mathrm{c}^N}{\prod_{n=1}^{N}(s - p_n)}, \tag{5.3}$$

where p_n is the nth pole and ω_c is the **half-power cutoff frequency**, i.e., the frequency at which the filter gain is $G_\mathrm{linear} = 1/\sqrt{2}$ or $G_\mathrm{dB} = -3dB$. The poles are located at

$$p_n = j\omega_\mathrm{c} e^{\frac{j\pi}{2N}(2n-1)} \tag{5.4}$$

$$= -\omega_\mathrm{c} \sin\left(\frac{\pi(2n-1)}{2N}\right) + j\omega_\mathrm{c} \cos\left(\frac{\pi(2n-1)}{2N}\right), \tag{5.5}$$

for $n = \{1, 2, \ldots, N\}$. So, given a desired filter order N and cutoff frequency ω_c, we can use Eq. 5.5 to find the poles and then Eq. 5.3 to determine the transfer function. You would find that the poles form a semi-circle to the left of the imaginary s-axis.

In practice the magnitude of the Butterworth filter frequency response is more common. For a low pass Butterworth filter, it is

$$|H(j\omega)| = \frac{1}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2N}}}. \tag{5.6}$$

To standardise design, we usually assume a normalised cutoff frequency $\omega_c = 1$ radian per second. The frequency response magnitude is then just

$$|H(j\omega)| = \frac{1}{\sqrt{1 + \omega^{2N}}}. \tag{5.7}$$

To convert the normalised frequency as used in Eq. 5.7 to non-normalised form, we just multiply ω by the actual ω_c.

A sample of frequency response magnitudes for Butterworth filters with normalised frequency is shown in Fig. 5.5. We see that increasing the order improves the approximation of ideal behaviour.

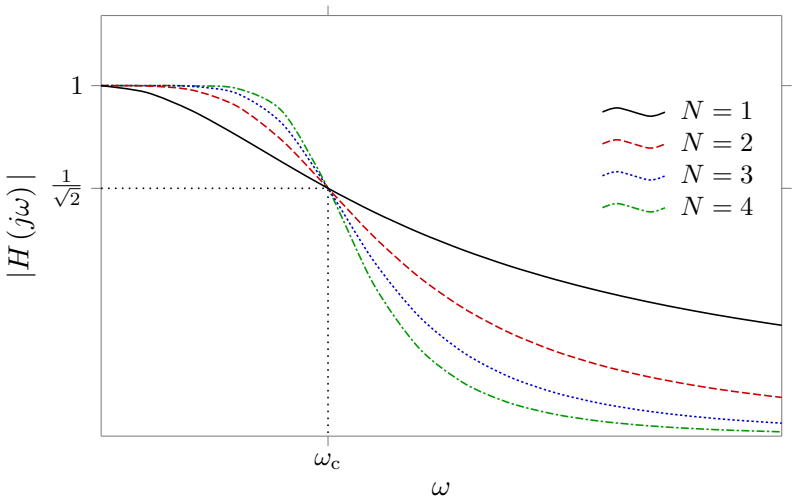


Figure 5.5: Frequency response magnitude of Butterworth low pass filter for different orders.

EXTRA 5.1: Converting Filter Designs

Our design of practical filters has focused on low pass filters and converting between different filter specifications. This is not an inherent limitation because we can also convert between different types of filters. For example, we

can replace s with $1/s$ to convert a low pass filter transfer function into a corresponding high pass filter transfer function. More complex conversions exist for band pass and band stop filters. More details can be found in Section 2.6 of Lathi and Green.

For a given low pass Butterworth filter specification, we have the following steps to find a corresponding filter that meets the specification:

1. Translating pass band and stop band requirements (via $G_\mathrm{p}, \omega_\mathrm{p}, G_\mathrm{s}, \omega_\mathrm{s}$) to a suitable order N.
2. Determine the cut-off frequency ω_c
3. Scaling the normalised frequency $\omega_\mathrm{c} = 1$.

It can be shown that the minimum order N is the following:

$$N = \left\lceil \frac{\log\left(\frac{10^{-G_\mathrm{s}/10}-1}{10^{-G_\mathrm{p}/10}-1}\right)}{2\log\left(\frac{\omega_\mathrm{s}}{\omega_\mathrm{p}}\right)} \right\rceil, \tag{5.8}$$

where $\lceil \cdot \rceil$ mean that we have to round up (i.e., so that we over-satisfy and not under-satisfy the specification) and the gains are in dB. Given the order, we have two ways to determine the low pass cut-off frequency ω_c:

$$\omega_\mathrm{c} = \frac{\omega_\mathrm{p}}{(10^{-G_\mathrm{p}/10}-1)^{\frac{1}{2N}}} \tag{5.9}$$

$$\omega_\mathrm{c} = \frac{\omega_\mathrm{s}}{(10^{-G_\mathrm{s}/10}-1)^{\frac{1}{2N}}}, \tag{5.10}$$

such that we meet the pass band specification exactly in the first case and we meet the stop band specification exactly in the second case. Due to component imperfections, we may choose a cut-off frequency that is in between these two cases.

EXTRA 5.2: Proof of Butterworth Filter Specification Equations

Let us demonstrate the validity of Eqs. 5.8, 5.9, and 5.10. We can write G_p (in dB) as a function of ω_p by converting to linear gain and using the definition

of the Butterworth filter frequency response, i.e.,

$$\begin{aligned} G_{\mathrm{p}} &= 20\log_{10}\left|H\left(j\omega_{\mathrm{p}}\right)\right| \\ &= 20\log_{10}\left(\frac{1}{\sqrt{1+\left(\frac{\omega_{\mathrm{p}}}{\omega_{\mathrm{c}}}\right)^{2N}}}\right) \\ &= 10\log_{10}\left(1+\left(\frac{\omega_{\mathrm{p}}}{\omega_{\mathrm{c}}}\right)^{2N}\right) \end{aligned}$$

which we can re-arrange to solve for ω_{c} and arrive at Eq. 5.9. Similarly, we can write G_{s} (in dB) as a function of ω_{s} and re-arrange to solve for ω_{c} and arrive at Eq. 5.10. By setting Eqs. 5.9 and 5.10 equal to each other, we can re-arrange and arrive at Eq. 5.8. The ceiling function in Eq. 5.8 comes from the fact that we need an integer order and rounding down will make the filter unable to meet the specification.

Given Eqs. 5.8, 5.9, and 5.10 (which you would *not* be expected to memorise), we can design Butterworth filters to meet target specifications. We see this in the following example.

Example 5.2: Butterworth Filter Design

Design a Butterworth filter with a pass band gain no less than $-2\,\mathrm{dB}$ over $0 \leq \omega < 10\,\frac{\mathrm{rad}}{\mathrm{s}}$, and a stop band gain no greater than $-20\,\mathrm{dB}$ for $\omega > 20\,\frac{\mathrm{rad}}{\mathrm{s}}$.

From the specification, we have $G_{\mathrm{p}} = -2\,\mathrm{dB}$, $\omega_{\mathrm{p}} = 10\,\frac{\mathrm{rad}}{\mathrm{s}}$ $G_{\mathrm{s}} = -20\,\mathrm{dB}$, and $\omega_{\mathrm{s}} = 20\,\frac{\mathrm{rad}}{\mathrm{s}}$. From Eq. 5.8, we find that $N = 3.701$. We need an integer order, so we choose $N = 4$. We are then free to choose whether to exactly meet the pass band or stop band specification. If we choose to satisfy the pass band specification, then we solve Eq. 5.9 and get $\omega_{\mathrm{c}} = 10.69\,\frac{\mathrm{rad}}{\mathrm{s}}$. If we choose to satisfy the stop band specification, then we solve Eq. 5.10 and get $\omega_{\mathrm{c}} = 11.26\,\frac{\mathrm{rad}}{\mathrm{s}}$. We could leave a "buffer" on both bands and choose $10.69\,\frac{\mathrm{rad}}{\mathrm{s}} < \omega_{\mathrm{c}} < 11.26\,\frac{\mathrm{rad}}{\mathrm{s}}$.

EXTRA 5.3: Other Filter Families

We could do a whole module on practical analogue filters, but we've just covered the Butterworth filter to give you some sense of practical filter design. Here we briefly summarise a few other common filter families to give you an idea of some of the trade-offs available. You can find more details in Lathi and Green.

The **Chebyshev** filter has "ripples" in the pass band (i.e., there are oscillations instead of being smooth like a Butterworth filter), but the transition band behaviour is steeper than the Butterworth filter. In practice, a lower order N is needed to meet a specification with Chebyshev filter than a Butterworth one.

While the pass band of a Chebyshev filter has ripples, its stop band is smooth. Since pass band behaviour is usually more important than the stop band, the **inverse Chebyshev** filter does the reverse and can be derived from a Chebyshev response.

Elliptic filters have ripples in both the pass band and stop band, in exchange for an even sharper transition band.

We haven't considered the phase of the frequency response in our discussion, but some filters are designed specifically for phase characteristics. **Bessel-Thomson** filters are designed for a maximally flat time delay over a specified frequency band, such that the phase response is close to linear over that band.

We are not covering time domain circuits implementations of practical analogue filters in this module, but you should be aware that they exist and commonly include op-amps. We will focus more on time-domain implementations when we consider digital filters.

5.5 Summary

- The common classifications of filters are **low pass**, **high pass**, **band pass**, and **band stop**. They are defined according to ideal transitions between **pass bands** and **stop bands**.

- Ideal filter magnitude responses are not realisable but we can obtain practical filter designs to approximate ideal responses.

- Butterworth filters can be readily designed to meet specified performance criteria.

5.6 Further Reading

- Chapter 2 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

5.7 End-of-Lesson Problems

1. Explain in words why the ideal low pass filter frequency response is unrealisable.

2. If a low pass Butterworth filter is 12th order with a 3 dB cut-off frequency at 500 Hz, calculate the gain of the filter at 750 Hz.

3. Design the lowest order low pass Butterworth filter that meets the specification $\omega_\mathrm{p} \geq 10\,\frac{\mathrm{rad}}{\mathrm{s}}$, $G_\mathrm{p} \geq -2\,\mathrm{dB}$, $\omega_\mathrm{s} \leq 30\,\frac{\mathrm{rad}}{\mathrm{s}}$, $G_\mathrm{s} \leq -20\,\mathrm{dB}$. Find the order and feasible range of cut-off frequencies.

4. A subwoofer is a loudspeaker designed for low-pitch audio frequencies. Professional subwoofers typically produce sounds below 100 Hz. Design an audio filter for a speaker input that produces $G_\mathrm{p} \geq -1\,\mathrm{dB}$ for frequencies below 100 Hz and $G_\mathrm{s} \leq -30\,\mathrm{dB}$ for frequencies above 250 Hz. Find a suitable order and cut-off frequency (in Hz) for a Butterworth filter that meets these requirements.

Lesson 6

Periodic Analogue Functions

As we saw in Lesson 4, the Fourier transform is used to determine the spectra of signals. This is fine when we focus on the response of a system to a particular input frequency, but it isn't convenient when the input signal of interest has multiple components with different frequencies. We complete the theoretical material on analogue systems and signals with a brief look at the Fourier Series, which is useful for representing more complicated periodic signals.

6.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** and **Apply** the Fourier Series to model complex periodic waveforms in the frequency domain.
2. **Understand** how periodic signals are affected by linear systems.

6.2 Representing Periodic Analogue Signals

There are multiple ways to represent periodic signals. We have been relying on Euler's formula to convert between an exponential representation and a trigonometric representation:

$$e^{jx} = \cos x + j \sin x \tag{6.1}$$

From Eq. 6.1, we can convert trigonometric functions into exponentials:

$$\cos x = \mathrm{Re}\left\{e^{jx}\right\} = \frac{e^{jx} + e^{-jx}}{2} \tag{6.2}$$

$$\sin x = \mathrm{Im}\left\{e^{jx}\right\} = \frac{e^{jx} - e^{-jx}}{2j} \tag{6.3}$$

A key point here is that a trigonometric function has an exponential form, and vice versa. Generally, the exponential form is more compact, and we have seen from Laplace and Fourier transforms that system responses to exponential signals are much easier to determine and manipulate than responses to trigonometric signals. However, there is a trade-off; complex exponential signals are more difficult to visualise. Thus, it is often more convenient to use exponential forms for mathematical manipulations and trigonometric forms for plots and intuition.

The exponential form of the **Fourier series** represents the period signal $x\left(t\right)$ as a sum of complex exponentials. There is an underlying fundamental frequency f_0, such that *all* frequencies contained in the signal are multiples of f_0. The corresponding fundamental period is $T_0 = 1/f_0$. The Fourier series is written as

$$x\left(t\right) = \sum_{k=-\infty}^{\infty} X_k e^{jk\omega_0 t}, \tag{6.4}$$

where $\omega_0 = 2\pi f_0 = 2\pi/T_0$ and the **Fourier coefficients** are

$$X_k = \frac{1}{T_0}\int_{T_0} x\left(t\right) e^{-jk\omega_0 t} dt, \tag{6.5}$$

which integrates the signal over its fundamental period.

One important property of the Fourier series is how it represents signals $x\left(t\right)$ that are real. A real $x\left(t\right)$ has an even magnitude spectrum and an odd phase spectrum, i.e.,

$$|X_k| = |X_{-k}| \quad \text{and} \quad \angle X_k = -\angle X_{-k} \tag{6.6}$$

Example 6.1: Fourier Series of Periodic Square Wave

A very important Fourier series is that for the periodic square wave. Consider that shown in Fig. 6.1, which is centred around $t = 0$, has amplitude A, and within a period remains high for τ seconds. We wish to write the function as a Fourier series and plots its magnitude. The period of the wave is by definition the fundamental T_0. We can then find the Fourier coefficients as

follows:

$$\begin{aligned}
X_k &= \frac{1}{T_0}\int_{T_0} x\left(t\right)e^{-jk\omega_0 t}dt \\
&= \frac{1}{T_0}\int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} Ae^{-jk\omega_0 t}dt \\
&= \frac{A}{T_0}\left[\frac{-e^{-jk\omega_0 t}}{jk\omega_0}\right]_{-\frac{\tau}{2}}^{\frac{\tau}{2}} \\
&= \frac{A}{T_0}\left(\frac{e^{jk\omega_0\frac{\tau}{2}} - e^{-jk\omega_0\frac{\tau}{2}}}{jk\omega_0}\right) \\
&= \frac{2A}{T_0}\frac{\sin\left(k\omega_0\frac{\tau}{2}\right)}{k\omega_0}\frac{\frac{\tau}{2}}{\frac{\tau}{2}} \\
&= \frac{A\tau}{T_0}\,\mathrm{sinc}\left(k\omega_0\frac{\tau}{2}\right),
\end{aligned}$$

which we note is an even function as expected for a real $x\left(t\right)$. In this particular case, the Fourier coefficients are also real (though generally they can be complex).

The signal $x\left(t\right)$ as a Fourier series is then

$$x\left(t\right) = \sum_{k=-\infty}^{\infty}\frac{A\tau}{T_0}\,\mathrm{sinc}\left(k\omega_0\frac{\tau}{2}\right)e^{jk\omega_0 t},$$

which we could also equivalently write by converting the sinc back to exponential form or converting the complex exponential to trigonometric form. More importantly, let us consider the magnitudes of the Fourier coefficients for different values of T_0 relative to τ, i.e., for different fundamental frequencies. For $T_0 = 2\tau$, we note that $\omega_0 = \pi/\tau$, and thus

$$X_k = \frac{A\tau}{2\tau}\,\mathrm{sinc}\left(k\frac{\pi}{\tau}\frac{\tau}{2}\right) = \frac{A}{2}\,\mathrm{sinc}\left(\frac{k\pi}{2}\right),$$

so we are sampling a sinc function at multiples of $\pi/2$. Similarly, for $T_0 = 5\tau$, we are sampling a (different) sinc function at multiples of $\pi/5$. The Fourier coefficients in both cases are plotted in Fig. 6.2.

It is important to emphasise that Fourier spectra only exist at the **harmonic frequencies**, i.e., at integer multiples of the fundamental frequency. The overall shape made the data points at these frequencies, i.e., the envelope, depends on the

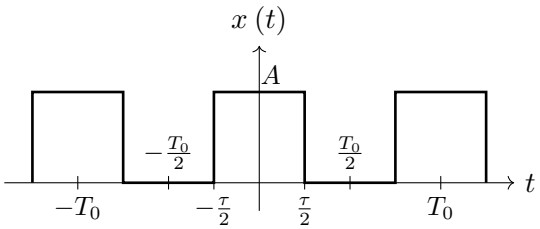
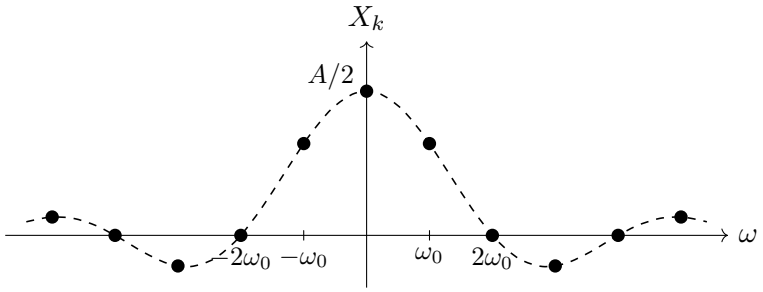
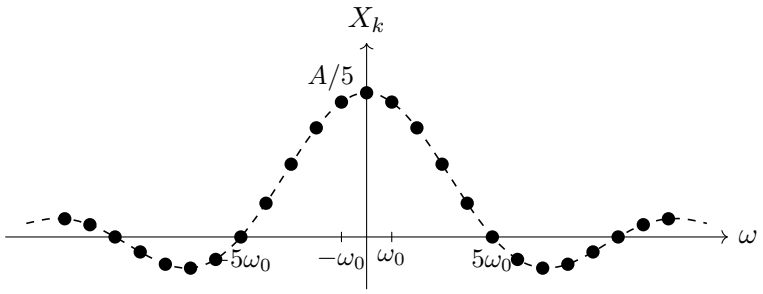


Figure 6.1: Periodic square wave in Example 6.1.



(a) $T_0 = 2\tau$.



(b) $T_0 = 5\tau$.

Figure 6.2: Fourier coefficients for periodic square wave in Example 6.1.

shape of the time-domain signal. It is as if we are sampling in the frequency domain, which results in a repeated signal in the time domain. In some sense, this is the opposite of what happens in analogue-to-digital conversion, where a time-domain signal is sampled and this results in repeated spectra. We will see this in more detail in the digital part of these notes.

6.3 Processing Periodic Signals in Linear Systems

We recall that for the Fourier transform we considered a single test sinusoid as the input to an LTI system. With the Fourier series, we have a (potentially infinite) sum

of sinusoids for the system input. Fortunately, besides the increase in complexity, there is no fundamental difference in the analysis. Due to the superposition property of LTI systems, the system will produce an output for each corresponding input, based on the frequency response at the corresponding frequency. In other words, the system will change the amplitude and the phase of each frequency in the input. Thus, we can write

$$y(t) = \sum_{k=-\infty}^{\infty} H(jk\omega_0) X_k e^{jk\omega_0 t}, \tag{6.7}$$

or in other words

$$Y_k = H(jk\omega_0) X_k. \tag{6.8}$$

Example 6.2: Filtering a Periodic Signal

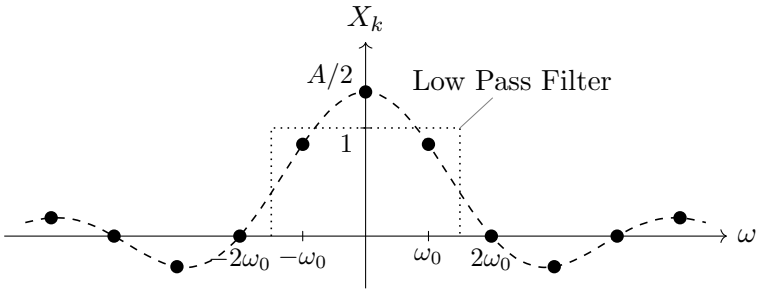
Consider passing the periodic square wave in Fig. 6.1, with $T_0 = 2\tau$, as the input into an ideal low pass filter with gain 1 for $0 \leq \omega < 1.5\frac{\pi}{\tau}$. What is the output of this filter in trigonometric form?

If $T_0 = 2\tau$, then the fundamental frequency is $\omega_0 = \pi/\tau$. The filter will attenuate any sinusoids with frequencies outside of $-1.5\pi/\tau < \omega_c < 1.5\pi/\tau$, so the only terms remaining in the summation are $k \in \{-1, 0, 1\}$. Therefore, the output is

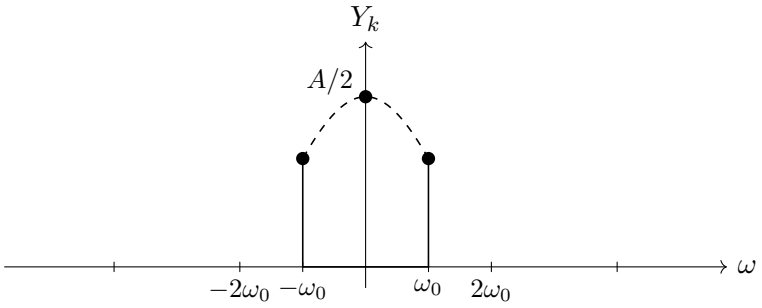
$$\begin{aligned}
y(t) &= \sum_{k=-1}^{1} H(jk\omega_0) X_k e^{jk\omega_0 t} \\
&= X_0 + X_{-1} e^{-jk\frac{\pi}{\tau}t} + X_1 e^{jk\frac{\pi}{\tau}t} \\
&= \frac{A}{2}\left[1 + \operatorname{sinc}\left(\frac{-\pi}{2}\right) e^{-\frac{j\pi t}{\tau}} + \operatorname{sinc}\left(\frac{\pi}{2}\right) e^{\frac{j\pi t}{\tau}}\right] \\
&= \left[1 + \frac{2}{\pi} e^{-\frac{j\pi t}{\tau}} + \frac{2}{\pi} e^{\frac{j\pi t}{\tau}}\right] \\
&= \frac{A}{2}\left[1 + \frac{4}{\pi}\cos\left(\frac{\pi t}{\tau}\right)\right].
\end{aligned}$$

We determined the output in trigonometric form because this is easier to plot (see Fig. 6.3). We also see that the output has 2 components: a constant plus a sinusoid with frequency π/τ. These match the frequencies of the input

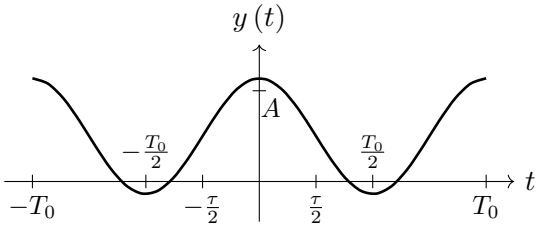
signal components that passed through the filter without being attenuated.



(a) Input spectra with $T_0 = 2\tau$. Here $\omega_0 = \pi/\tau$.



(b) Output spectra with $T_0 = 2\tau$. Here $\omega_0 = \pi/\tau$.



(c) Time-domain filter output $y\,(t)$.

Figure 6.3: Filter input and output in Example 6.2.

EXTRA 6.1: Fourier Series Properties

We can show that the Fourier series has many properties that are similar to those for the Fourier transform. However, since we will place a bigger emphasis on digital analysis, we will not elaborate on these here. You can find more discussion of these in Section 1.9 of Lathi and Green.

6.4 Summary

- The **Fourier Series** is used to model signals with multiple frequency components in the frequency domain.
- The output of an LTI system due to a signal with multiple frequency components can be found by superposition of the outputs due to the individual frequency components.

6.5 Further Reading

- Section 1.7 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

6.6 End-of-Lesson Problems

1. Consider a periodic square wave with period $T_0 = 0.3\,\text{s}$ that is passed into an ideal low pass filter that attenuates components with frequencies above 8 Hz. What are the frequencies of the components that remain in the signal at the output of the filter?
2. Consider passing the periodic square wave in Fig. 6.1, with $T_0 = 5\tau$, as the input into an ideal band pass filter with gain 1 for $1.5\frac{\pi}{\tau} < \omega < 2.5\frac{\pi}{\tau}$ (and gain 0 elsewhere). How many sinusoidal terms will there be at the output? What will the frequencies of those terms be?
3. Consider passing the periodic square wave in Fig. 6.1, with $T_0 = 2\tau$, as the input into a low pass Butterworth filter of the 3rd order and cut-off frequency $\omega_\text{c} = 3/\tau$. What is the output amplitude of the complex exponential component of frequency π/τ?

Lesson 7

Computing with Analogue Signals

This lesson is the end of Part 1 of these notes. Our time to focus on purely analogue signals and systems is coming to a close. Here we shift our attention from primarily mathematical analysis to computing and applications. Analogue systems are by definition not digital systems, but we still find it useful to represent them on computers to help with design and analysis. In this lesson, we will see how MATLAB can be used as a computing tool for analogue systems. We should emphasise that MATLAB is not the only tool available for such purposes, as there are comparable features available in other platforms such as Mathematica, and also in programming languages. We use MATLAB since you may have used it in other modules and it has a relatively low learning curve for programming.

The computing examples are implementations and demonstrations of tasks that we've done "by hand" in previous lessons, so they should be helpful as a learning tool, e.g., to help verify your calculations and plots, and to investigate systems that are too complex to readily do so manually. Furthermore, the MATLAB functions used for digital systems are very similar (or sometimes *the same*) as those for analogue systems, so understanding the code here will offer some additional return later.

7.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** how analogue signals and systems are represented in digital hardware.

2. **Analyse** analogue signals and systems with a computing package.

3. **Design** and **Implement** analogue filters with a computing package.

7.2 Analogue Signals in MATLAB

Computers are inherently digital systems. Information is stored in discrete containers such as arrays and matrices, and individual numbers are restricted to take on a finite number of values. Thus, they are ideal for implementing discrete-time digital signals and systems, and in fact they are why we have digital signal processing to begin with. So how can computers effectively model analogue systems?

While it is not possible to *precisely* implement an analogue system in a computer, we are able to store precise information about an analogue system via **symbolic math** (also known as **computer algebra**). In other words, we can store the *functions* that describe the analogue system, e.g., the impulse response, the transfer function, the input signal, and the output signal. For example, we have seen that we can write the Laplace-domain transfer function $H(s)$ of an LTI system as a ratio of polynomials. Therefore, to represent this system as a computer, we can use arrays to store the polynomial coefficients. This is the approach used in MATLAB.

Even though digitally-represented numbers have a finite number of values, the number of values is sufficiently large that granularity is not a problem for most practical systems. For example, a double-precision floating point number, which is the standard precision for most modern computing systems, uses 64 bits and has 15-17 decimal digits of precision over its range. This is *far more precision than what we usually ever need*. Thus, we can approximate double-precision numbers as being over a continuous range. When we want to know the behaviour of an analogue system at a particular time, we **sample** at the time of interest within the resolution of a double-precision number, and behaviour of the system at that time is calculated and returned within the resolution of a double-precision number.

EXTRA 7.1: Aside on Double-Precision Granularity

Strictly speaking, as we perform more and more calculations with double-precision numbers, we lose more and more accuracy as the precision errors accumulate. This is usually not an issue in practice, but it can be useful to keep in mind if you get unexpected behaviour from code where you expect two calculated values to be identical. For example, it is often helpful to round values to the nearest integer to mitigate such effects.

Let's now discuss how to create and study analogue systems in MATLAB. These is not a definitive discussion, as there is often more than one way to do something in MATLAB, but this will give you some ways to get started. Some particularly relevant toolboxes are the **Symbolic Math Toolbox** the **Signal Processing Tool-**

box, and the **Control Systems Toolbox**, although they can be used independently from each other.

7.2.1 Analogue Systems in the Symbolic Math Toolbox

With the Symbolic Math Toolbox, you can create symbolic variables. These are not variables in the traditional programming sense, but in the more abstract mathematical sense as they can be used to define functions and solve them analytically. Symbolic variables can be created using the `syms` function, and for an analogue system we usually find it convenient to create symbolic variables for time t and for s, i.e., enter

```
syms t s
```

We can now write expressions using s or t and they will be stored as symbolic functions that can be evaluated analytically. For example, we can create a time-domain input signal as a function of t and a Laplace-domain transfer function as a function of s. We can translate between the two domains using the `laplace` and `ilaplace` functions. Similar, we can use the `fourier` and `ifourier` functions for the Fourier and inverse Fourier transforms, respectively. There are a large number of pre-built functions available, such as `rectangularPulse`, `dirac`, `heaviside` (step function), `cos` (and other trigonometric functions as you might expect), and `sign`. You can also write your own functions.

Example 7.1: Finding Expression for System Output

Find the output of a simple low pass filter with transfer function

$$H(s) = \frac{1}{s+1},$$

when the input is a cosine $x(t) = \cos(\omega t)$. Using our knowledge of LTI systems, we can find the answer in a short function using symbolic math.

```
function y = lowpassOutput(omega)
% Find output to my low pass filter

% Create symbolic variables
syms t s
% Create symbolic functions for input and system
x = cos(omega*t);
H = 1/(s+1);
```

```
% Convert input to Laplace, solve, and convert output to time
X = laplace(x); Y = H*X; y = ilaplace(Y);
```

If you call this function with frequency $\omega = 5$, then the output will be `y = cos(5*t)/26 - exp(-t)/26 + (5*sin(5*t))/26`.

We can visualise a symbolic function using `fplot`, where an optional second argument specifies the range of coordinates of the domain, e.g.,

```
y = lowpassOutput(5);
figure; % New figure
fplot(y,[0,10])
```

This works for either the time or s-domain, but only for real values of s.

Finally, we can substitute the variables in a symbolic expression with a numerical value using the `subs` function, and then convert the expression to a double-precision number using the `double` function, e.g.,

```
syms s
y = s^2 + 2*s;
yEval = subs(y,'s',5); % Substitute value
yDbl = double(yEval); % Convert to double-precision
```

7.2.2 Analogue Systems in the Signal Processing Toolbox

The Signal Processing toolbox provides the `freqs` function for plotting frequency responses for analogue transfer functions (the "s" in `freqs` stands for s-domain). The first two input arguments for this function are two arrays, one describing the coefficients of the numerator polynomial and one describing the coefficients of the denominator polynomial. The standard naming convention for these arrays matches the form that we used in Lesson 3; a `b` vector lists the numerator coefficients in decreasing order and an `a` vector lists the denominator coefficients in decreasing order. For the system in Example 7.1, we have `b=1` and `a=[1,1]`. We can then plot the frequency response of this filter:

```
b = 1; a = [1, 1];
figure; % New figure
w = 0.1:0.1:10;
freqs(b,a,w)
```

The figure is shown in Fig. 7.1. The argument `w` is a vector of (radial) frequencies, but if you make it scalar instead then it defines the number of frequency points are

determined automatically. Importantly, `freqs` plots the magnitude and phase of the frequency response on a logarithmic scale (for both the frequencies and for the magnitude). If you want to evenly space the data points along the domain of each plot, then you can use the `logspace` function to create these for you (e.g., `logspace(1,2,10)` will create 10 logarithmically spaced points between 10^1 and 10^2).

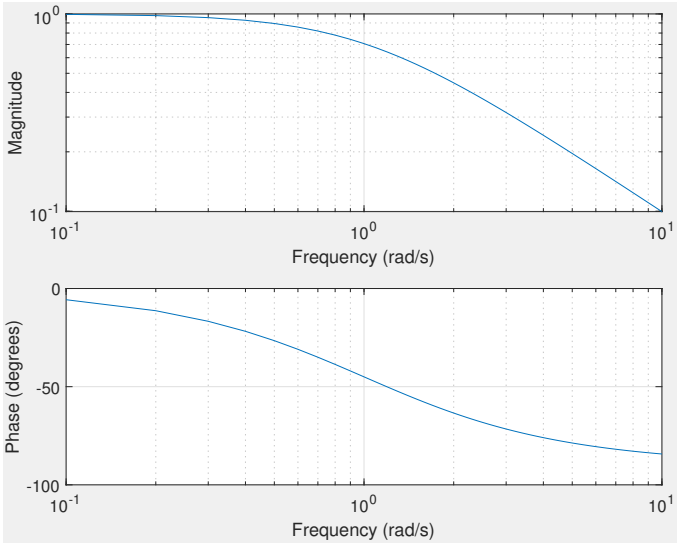


Figure 7.1: Frequency response of the system in Example 7.1.

If you assign `freqs` to an output argument, then it will store the complex frequency response values for each of the frequencies. If you append a semi-colon `;` after the function then it will omit the plot.

7.2.3 Analogue Systems in the Control Systems Toolbox

If you also have the Control Systems Toolbox then you can also have functions that can automatically generate pole-zero plots from a transfer function. With this toolbox, transfer functions are defined using the `tf` function, where the input is of the form `sys = tf(b,a)`, where `b` and `a` are once again vectors that describe the descending coefficients of the numerator and denominator polynomials. You can then create a pole-zero plot using the `pzplot` function:

```
b = 1; a = [1, 1];
figure; % New figure
H = tf(b,a);
pzplot(H)
```

The figure is shown in Fig. 7.2. For this particular system the result is rather boring as there is a single pole and no zeros.

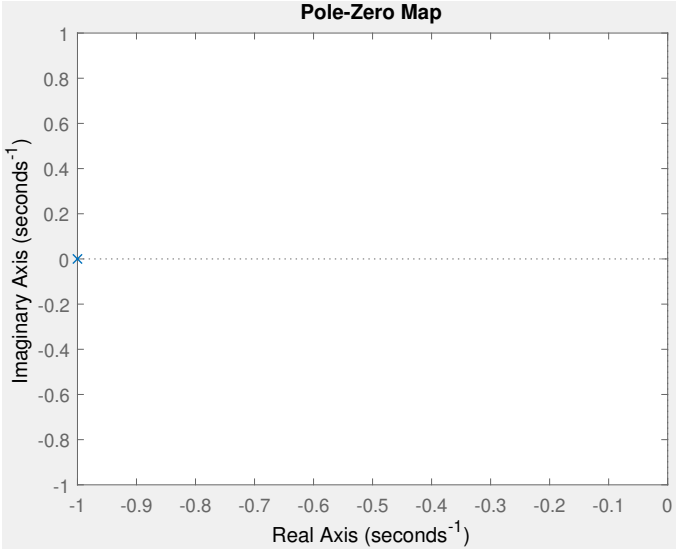


Figure 7.2: Frequency response of the system in Example 7.1.

7.3 Analogue Filter Design

A particular strength of the Signal Processing Toolbox is that it has practical filter families included, even for analogue systems. In fact, all of the filter families described in Lesson 5 are available, i.e., Butterworth, Chebyshev, elliptic, Bessel, and more. The corresponding function names are somewhat intuitive but we list them here:

- Butterworth: `butter`
- Chebyshev: `cheby1` (i.e., type 1)
- Inverse Chebyshev: `cheby2` (i.e., type 2)
- Elliptic: `ellip`
- Bessel (i.e., Bessel-Thomson): `besself`

The syntax and use of these functions are consistent, so we will focus on using the `butter` function. The default input arguments are the order of the filter and the cut-off frequency, *where the order is* ***half*** *the total order for a band pass or band*

stop filter. By default the filter will be low pass, but a third optional argument can indicate whether the filter is to be a different type, i.e., one of `'low'`, `'high'`, `'bandpass'`, or `'stop'`. To actually make this an *analogue* filter instead of a digital one is to make the final argument `'s'`. For the output, we will consider one of 2 formats. If you define two input arguments then they will be of the form `[b,a]`, i.e., the coefficients of the transfer function's numerator and denominator polynomials, respectively. If you define three input arguments then they will be of the form `[z,p,k]`, i.e., you will get the poles, zeros, and the transfer function gain K. From our knowledge of transfer functions, either of these two forms are sufficient to fully define the transfer function. You can also convert between the two forms using the `zp2tf` or `tf2zp` functions.

Example 7.2: Analgoue Butterworth Filter in MATLAB

Design a 4th order analogue low pass Butterworth filter with cutoff frequency $\omega_c = 50\,\frac{\text{rad}}{\text{s}}$. What are its poles and zeros? Plot its frequency response and a pole-zero plot.

We can complete all of these tasks directly in a script.

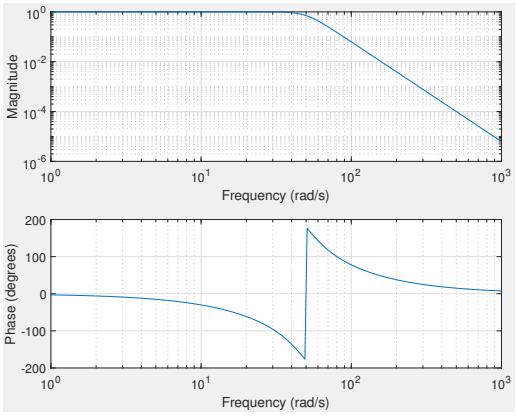
```
[z,p,k] = butter(4,50,'low','s') % Omitting semi-colon will
    print output
[b,a] = zp2tf(z,p,k); % Convert to [b,a] form for freqs
figure; freqs(b,a) % Plot frequency response
H = tf(b,a);
figure; pzplot(H) % Pole-zero plot
```

The output states that there are no zeros and 4 poles. The generated plots are shown in Fig. 7.3.

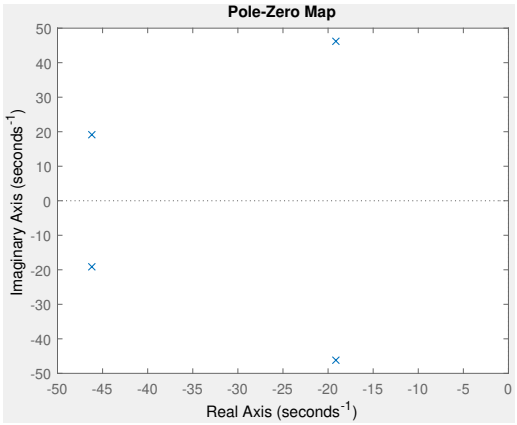
7.4 Summary

- We can analytically represent analogue systems in computing platforms, either formally using **symbolic math** or indirectly with vectors that represent system components.

- MATLAB has toolboxes with functions that enable us to implement most of the tasks that we have completed mathematically and "by hand" throughout these notes.



(a) Frequency response.



(b) Pole-zero plot.

Figure 7.3: Output plots of script in Example 7.2.

7.5 Further Reading

- The MATLAB documentation is an excellent resource, whether online or within the MATLAB help browser.
- Chapter 17 of "Essential MATLAB for engineers and scientists," B. Hahn and D. Valentine, Academic Press, 7th Edition, 2019.

7.6 End-of-Lesson Problems

1. Find the output of a filter with transfer function

$$H(s) = \frac{R_2}{s^2 LCR_1 + s(L + CR_1R_2) + R_1 + R_2},$$

where $R_1 = R_2 = 2\,\Omega$, $C = 1\,\mathrm{F}$, and $2\,\mathrm{H}$. The input is the sinusoid $x(t) = \sin(10t)$. Do *not* try to solve this by hand!

2. The `butter` function requires the target order N and cut-off frequency ω_c. Write a function that will calculate these from the more general low pass filter specifications (i.e., from the pass band and stop band gains and frequencies).

Part II

Digital Signals and Systems

Lesson 8

Signal Conversion Between Analogue and Digital

The second part of this module is the study of **Digital systems and signals**. Although most practical physical phenomena can be represented as continuous-time analogue signals, most modern signal processing is done in the digital domain. Not surprisingly, we need to be able to convert between the analogue and digital domains. This lesson provides an overview of the main steps involved in signal conversion in both directions, i.e., analogue-to-digital and digital-to-analogue. We describe conversion within the context of the end-to-end workflow of digital signal processing.

8.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the basic signal processing workflow.
2. **Understand** and **Apply** sampling theory and the presence of aliasing.
3. **Understand** the quantisation and hold operations for signal conversion.

8.2 Basic Signal Processing Workflow

Many practical digital signal processing systems need an interface with the real world, e.g., to perform sensing and control. We show the basic end-to-end workflow in Fig. 8.1, which has 5 key components:

1. A low pass filter is applied to the time-domain input signal $x(t)$ to limit its frequencies.

Figure 8.1: Digital signal processing workflow

2. An **analogue-to-digital converter** (ADC) samples and quantises the continuous-time analogue signal to convert it into a discrete-time digital signal $x[n]$.

3. The digital signal processing chip (DSP) performs the operations required and generates the output signal $y[n]$.

4. A **digital-to-analogue converter** (DAC) uses hold operations to reconstruct an analogue signal from $y[n]$.

5. An output low pass filter removes the high frequency components introduced by the DAC operation to give the final output $y(t)$.

It is important to note that a practical signal processing system might not always have all of these stages, in particular if the input does not need to come from an analogue system (i.e., we omit the ADC and input filter) or if the output does not need to be translated back into analogue (i.e., we omit the DAC and the output filter). Some examples:

- A weather monitoring station takes in analogue sensor measurements to store data digitally but does not need to convert back to analogue.

- Radar systems detect real objects using advanced digital signal processing techniques.

Nevertheless, there are many practical systems that will use the entire digital signal processing chain:

- Smart temperature control systems measure the temperature to turn the heating system on and off and require a digital interface for remote scheduling and history tracking. Generally, any kind of control system will need analogue-to-digital-to-analogue conversion.

- Digital cameras take images of the real world that we can touch up before printing.

- Wireless repeaters need to receive and re-send analogue signals, but the intermediate processing is digital.

- Video games convert some form of analogue input (e.g., touchscreen, gamepad) and provide a visual response.

- Digital audio amplifiers take vibration information from an instrument, apply different digital effects, and send the output to a speaker.

While there are also isolated systems that are almost entirely digital, such as computer simulations and virtual currencies, there are usually analogue interfaces for humans to monitor these systems.

We covered analogue filtering in the first part of this module. Later lessons in this part will cover signal processing within the digital domain. The rest of this lesson on the ADC and DAC processes.

8.3 Sampling Analogue Signals

The first step in analogue-to-digital conversion is to convert a signal from continuous-time to discrete-time. This is done by **sampling**, where we record the amplitude of the analogue signal at specified times. Usually, the time between the samples is fixed, such that we have a sampling period T_s and a sampling frequency f_s, as shown in Fig. 8.2.

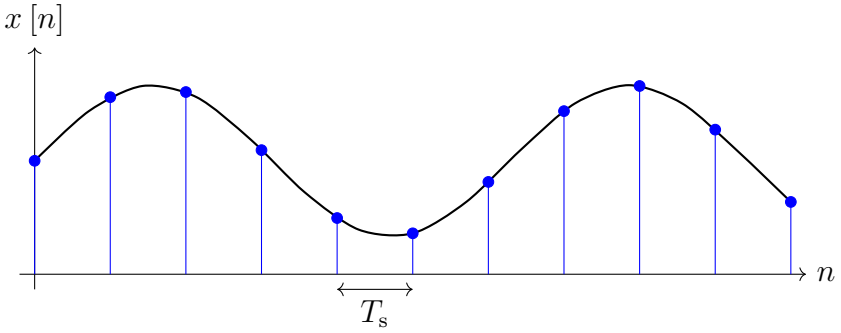


Figure 8.2: Sampling an analogue signal with sampling period $T_\mathrm{s} = 1/f_\mathrm{s}$.

The challenge with sampling is to determine the best sampling rate, but this is straightforward if we know something about the frequencies present in the signal that we are sampling. If we **over-sample**, i.e., sample too often, then we are using more complexity than we need and we are wasting energy. However, if we **under-sample**, then we may get **aliasing** of our signal, which happens when multiple signals of different frequencies yield the same data when sampled. We show an example of this visually in the time-domain in Fig. 8.3. If we sample the black sinusoid at the times indicated with the blue marker, it could be mistaken for the red dashed sinusoid. This happens when under-sampling, and the lower frequency signal is called the **alias**. The alias makes it *impossible* to recover the original data.

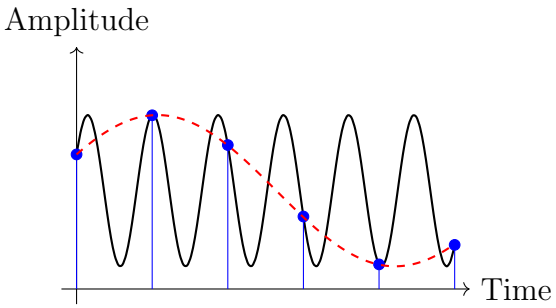
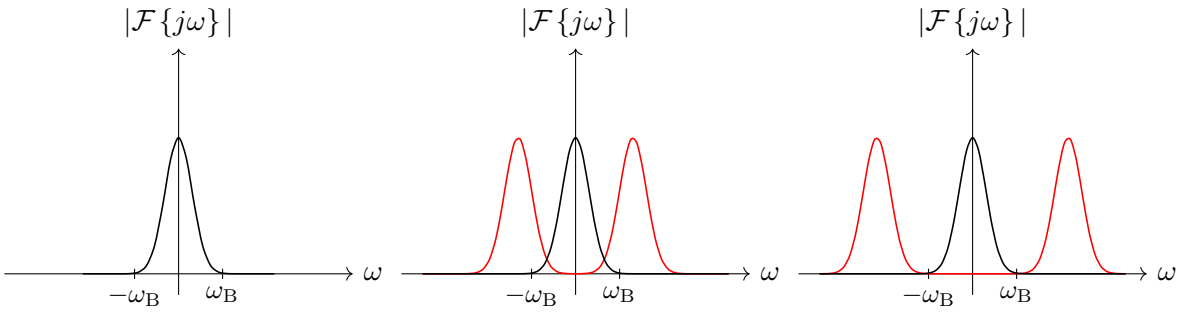


Figure 8.3: Aliasing of a signal in the time-domain.

To better understand how we determine the necessary sampling rate, we go to the frequency domain. Consider a signal $f(t)$ with the magnitude response shown in Fig. 8.4(a). We show its **double-sided frequency response**, which includes negative frequencies that mirror the positive frequency behaviour. It is constrained so that it has no components with a frequency greater than ω_B (which we can guarantee because of the input low pass filter!). We call ω_B the **bandwidth** of the signal.



(a) Original signal. (b) Sampling $\omega_s < 2\omega_B$. (c) Sampling at $\omega_s = 2\omega_B$.

Figure 8.4: Impact of sampling frequency on magnitude response.

We will see that discrete-time signals have *repeating* spectra. This means that when we sample $f(t)$ with sampling frequency $\omega_s = 2\pi f_s$, we get infinite copies of the spectra, each separated by the sampling frequency ω_s. When ω_s is too small, as shown in Fig. 8.4(b), the spectra overlap and we get aliasing. However, if we have $\omega_s \geq 2\omega_B$, then the spectra remain distinguishable. Thus, the minimum anti-aliasing sampling frequency is $\omega_s = 2\omega_B$ or equivalently $f_s = 2f_B$. This minimum frequency is also known as the **Nyquist rate**.

The **sampling theorem** is the formalisation of the minimum sampling idea. It states that for a continuous-time analogue signal with frequency components less than f_B, the signal can be adequately represented by a sampled version if the

sampling rate exceeds $f_\mathrm{s} \geq 2f_\mathrm{B}$. Otherwise, aliasing will occur. Thus, we need to sample at a frequency that is *at least* twice the highest frequency of the signal. In practice we may try to sample higher since low pass filters are not ideal, however the opposing constraint is the cost and complexity of faster sampling (i.e., we need to collect samples faster and store more of them in memory).

Example 8.1: Nyquist sampling

If we have music that ranges from 0 to 100 kHz, but the highest audible frequency is about 20 kHz, what is the ideal sampling frequency to enable re-construction of the audible frequencies?

By ideal sampling frequency, we mean the minimum sampling frequency. This would be twice the audible bandwidth, so we sample at $f_\mathrm{s} = 2f_\mathrm{B} = \boxed{40\,\mathrm{kHz}}$.

Mathematically, we can represent sampling as multiplying the time-domain signal by an infinite train of impulses $\delta(t)$. Thus, ignoring quantisation, we can write the **sampled discrete-time signal** $x\,[n]$ as

$$x\,[n] = \sum_{n=0}^{\infty} \delta(t - nT_\mathrm{s}) x\,(t)\,, \tag{8.1}$$

where we use index n instead of time t. We can refer to $x\,[n]$ as a **sequence** because of the discrete sampling times.

8.4 Quantisation

Our discussion here is important from a practical perspective but will not have a significant impact on the derivations in the rest of this part of the module, as mathematical manipulation is generally easier to apply to signals with continuous amplitude. Thus, we will be brief.

Digital systems are based on a binary representation of data, where discrete bits of memory are used to store values. **Quantisation** is the mapping of continuous amplitude levels to a binary representation. If we are coding with W bits, then there are 2^W quantisation levels available and we say that the ADC word length is W. Continuous amplitude levels must be approximated to their nearest level (e.g., by rounding), and the resulting error between the nearest level and the actual level is known as **quantisation noise**.

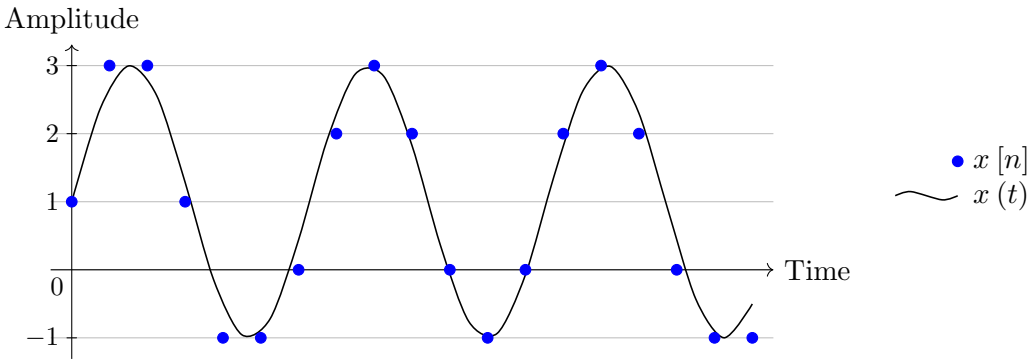


Figure 8.5: Analogue signal $x\,(t)$ that is sampled and quantised to the nearest integer to give digital signal $x\,[n]$.

EXTRA 8.1: More on Quantisation

There are many variations of quantisation, including where to start the mapping and how to separate the levels. Signals that vary by orders of magnitude, such as human speech, can benefit from **non-uniform quantisation**. By giving smaller separation between the lower amplitude levels, we can capture more detail than if we had equal separation between all levels, i.e., **uniform quantisation**. Double precision floating point numbers use non-uniform quantisation.

Example 8.2: Quantisation Levels

Let's say that we have a uniform ADC with $W = 3$ bits and the range of possible values is 0 to 2. What is the separation between quantisation levels?

There are $2^W = 2^3 = 8$ levels. The range of values has a length of 2, and there will be $8 - 1 = 7$ intervals. Thus, the separation between levels is $\boxed{2/7}$.

8.5 Analogue Reconstruction

To convert a digital signal back to the analogue domain, we need to **reconstruct** a continuous signal from a discrete-time series of points. This is achieved via **data interpolation**. The simplest interpolation in a DAC is a **hold circuit**, where the

amplitude of the continuous-time signal matches that of the previous discrete-time signal, i.e., we *hold* the amplitude until the next discrete-time value. The result is that the DAC produces a "staircase"-like output, such as that shown in Fig. 8.6.

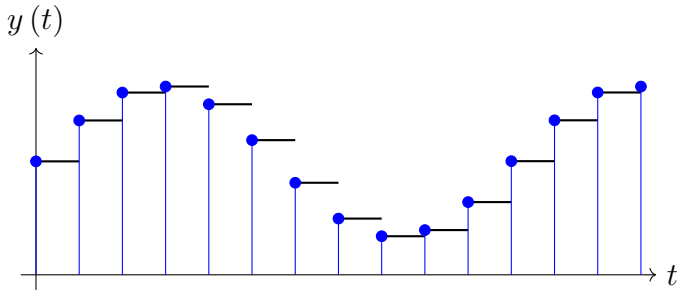


Figure 8.6: DAC re-construction with an ideal hold circuit.

EXTRA 8.2: Zero-Order Hold

The simplest hold circuit is also known as a **zero-order hold**. This is because if we write the signal $f(t)$ between times $t = nT_\mathrm{s}$ and $t = (n+1)T_\mathrm{s}$ as a power series, then the term $f(t) = f(nT_\mathrm{s})$, for $nT_\mathrm{s} \leq t \leq (n+1)T_\mathrm{s}$, is the first term in the series. The transfer function for the zero-order hold circuit can be found to be

$$F(s) = \frac{1}{s} - \frac{1}{s}e^{-sT_\mathrm{s}} \approx \frac{T_\mathrm{s}}{1+sT_\mathrm{s}}.$$

The approximation of this response matches that of a simple low pass filter, as we saw in Lesson 4. Thus, we can see that the zero-order hold helps to repress higher order harmonics.

Two commonly defined DAC parameters are the **resolution** and **dynamic range**. The resolution refers to the space between the levels, and it is often represented as a percentage. For a W-bit DAC with uniform levels, the DAC resolution is then $\frac{1}{2^W} \times 100\%$. The dynamic range describes the range of signal amplitudes that the DAC can resolve between its smallest and largest (undistorted) values. It is calculated as $20\log_{10} 2^W \approx 6W$ dB.

Example 8.3: DAC Resolution

What is the resolution of an 8-bit DAC?
Resolution is $\frac{1}{2^W} \times 100\% = \frac{1}{2^8} \times 100\% = 0.39\,\%$.

8.6 Summary

- The basic end-to-end signal processing workflow includes an input low pass filter, an **ADC**, the DSP chip, a **DAC**, and an output low pass filter.
- To be resolvable, a signal must be sampled at a frequency that is at least twice the signal bandwidth, otherwise **aliasing** can occur.
- **Quantisation** maps the amplitude of an analogue signal to discrete values, and hold operations convert a discrete-time signal back to continuous-time.

8.7 Further Reading

- Chapter 3 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

8.8 End-of-Lesson Problems

1. Consider sampling the signal $f(t) = \cos(20\pi t) + \sin(30\pi t)$. What is the minimum sampling frequency (in Hz) needed to avoid aliasing?
2. There are two analogue signals $f_1(t) = \cos(\omega_1 t)$ and $f_2(t) = \cos((\omega_1 + \omega_s)t)$, where $\omega_s = 2\pi f_s$ is the sampling frequency. The sampled versions of these signals are $f_1[n] = \cos(n\omega_1 T_s)$ and $f_2[n] = \cos(n(\omega_1 + \omega_s)T_s)$. Show that these must be aliased signals.
3. Consider that we are sampling the signal $f_1(t) = \cos(20\pi t)$ at a sampling frequency of $f_s = 35\,\text{Hz}$. Can $f_1(t)$ be an alias for a sinusoid with a frequency of 25 Hz? If yes, what is the sinusoid $f_2(t)$? If not, what is the lowest frequency signal for which $f_1(t)$ is an alias?
4. Suppose that we have a uniform W-bit quantiser that rounds a sample to the nearest level, and where the levels are distributed over the range $[0, 1]$ (inclusive). What is the size of the largest quantisation error that can occur?
5. What are the resolution and dynamic range of a 12-bit digital-to-analogue converter?

Lesson 9

Z-Transforms and LSI Systems

For analogue systems we used the Laplace transform to transform LTI signals and systems to the Laplace domain. For digital systems, we have a different class of signal and we need a different kind of transform. In this lesson, we present linear shift invariant (LSI) systems and the Z-transform. This forms the basis for all of the analysis of digital signal processing that we will perform in this part of the module. Many of the ideas will seem familiar from our analogue analysis, so in some sense we are translating everything from the analogue domain, but beware that there are distinct differences between continuous-time and discrete-time analysis, so it's important to not confuse the two.

9.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Apply** the Z-transform and the inverse Z-transform.
2. **Apply** properties of the Z-transform and inverse Z-transform.
3. **Understand** what a Linear Shift Invariant (LSI) system is and its common components.
4. **Analyse** LSI systems to find their transfer function and time-domain output.

9.2 Linear Shift Invariant Systems

Linear shift invariant (LSI) systems are precisely the discrete-time equivalent of LTI systems. We will update our notation and we will see that it corresponds very closely with that of LTI systems. Given an LSI system defined by the functional

$\mathcal{F}\{\cdot\}$ acting on discrete-time input signals (or *sequences*) $x_1[n]$ and $x_2[n]$, where n is the discrete-time sampling index, the following properties hold:

1. The system is **linear**, meaning that:

   (a) The system is **additive**, i.e.,

   $$\mathcal{F}\{x_1[n] + x_2[n]\} = \mathcal{F}\{x_1[n]\} + \mathcal{F}\{x_2[n]\} \tag{9.1}$$

   (b) The system is **scalable** (or **homogeneous**), i.e.,

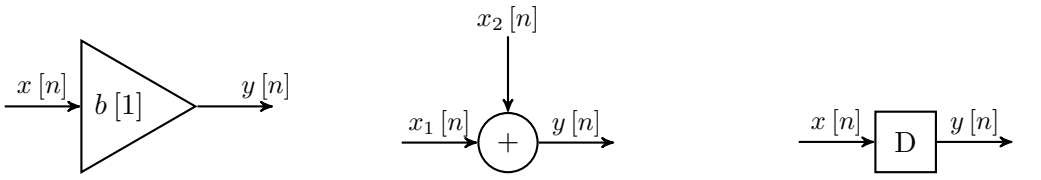
   $$\mathcal{F}\{ax_1[n]\} = a\mathcal{F}\{x_1[n]\} \tag{9.2}$$

   for any real or complex constant a.

2. The system is **shift-invariant**, i.e., if output $y[n] = \mathcal{F}\{x_1[n]\}$, then

   $$y[n-k] = \mathcal{F}\{x_1[n-k]\}. \tag{9.3}$$

   In other words, shifting (i.e., delaying) the input by some constant number of time steps k will delay the output and make no other changes.



(a) Multiplier. (b) Adder. (c) Unit delay.

Figure 9.1: Common components of LSI systems.

In continuous-time, we saw some examples of implementing LTI systems using circuit components. However, we did not take a deliberate approach. For digital signal processing, we are taking a closer look at *how* digital systems are implemented in the time domain. Fortunately, we only need to learn 3 types of LSI circuit components (shown in Fig. 9.1). All digital systems that we will consider in this part of the module can be composed of combinations of these components:

1. A **multiplier** scales the current input by a constant, i.e., $y[n] = b[1]\,x[n]$.
2. An **adder** outputs the sum of two or more inputs, e.g., $y[n] = x_1[n] + x_2[n]$.
3. A **unit delay** imposes a delay of one sample on the input, i.e, $y[n] = x[n-1]$.

Example 9.1: Writing Output of an LSI System

Consider the LSI circuit shown in Fig. 9.2. Write the output $y[n]$ as a function of the input $x[n]$.

We see the $x[n]$ is added to a delayed version of itself, and then the sum is scaled by $1/2$. Therefore, the output is

$$y[n] = \frac{x[n] + x[n-1]}{2}.$$

This is a simple *moving average* filter.

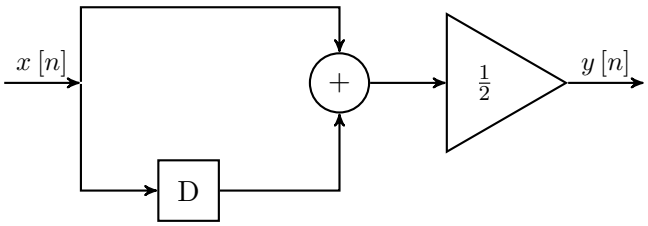


Figure 9.2: LSI system in Example 9.1.

The notion of an impulse response is very similar in the digital domain, as it is the system output when the input is an impulse, but now we have a discrete-time impulse function $\delta[n]$. The impulse response sequence $h[n]$ is then

$$h[n] = \mathcal{F}\{\delta[n]\}. \tag{9.4}$$

Given an LSI system's impulse response $h[n]$, the output is the **discrete convolution** of the input signal with the impulse response, i.e.,

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] = x[n] * h[n] = h[n] * x[n]. \tag{9.5}$$

Because Eq. 9.5 has a summation instead of an integral, discrete convolution is actually much easier to evaluate by hand than continuous-time convolution. In fact, as we saw in Example 9.1, it is also easier to directly write the output $y[n]$ by inspection of an LSI circuit. Nevertheless, due to the insight that we get by considering digital systems in other domains, we will still find it very helpful to identify suitable transforms. We will start with the **Z-transform**, which will enable comparable system analysis to that from the Laplace transform.

9.3 The Z-Transform

The **Z-transform** $\mathcal{Z}\{\cdot\}$ converts a discrete-time domain function $f[n]$, which we will assume is causal, i.e., $f[n] = 0, \forall n < 0$, into a complex domain function $F(z)$ that we say is in the z-domain. It is defined as follows:

$$\mathcal{Z}\{f[n]\} = F(z) = \sum_{k=0}^{\infty} f[k]\, z^{-k}, \tag{9.6}$$

which makes it the discrete-time equivalent of the Laplace transform. However, unlike the Laplace transform, the Z-transform can be written by *direct inspection* of the data sequence of the function being transformed. This is because we have a summation instead of an integral. The **inverse Z-transform** $\mathcal{Z}^{-1}\{\cdot\}$ is equally simple in such cases and can be written by observation. We can show conversion between the time domain and z-domain as

$$f[n] \Longleftrightarrow F(z)\,. \tag{9.7}$$

EXTRA 9.1: More on the Z-Transform

Just as we did for the Laplace transform, we are actually using the definition of the *unilateral* Z-transform which is specifically for causal signals. Once again, we will not be concerned with using the more general *bilateral* Z-transform. Furthermore, we are not going to write out the formal definition of the inverse Z-transform, because it involves a contour integral. You can read more about the bilateral Z-transform in Section 3.7 of Lathi and Green, and more about the inverse Z-transform in Section 3.9.

Example 9.2: Finding the Z-Transform by Inspection

What is the Z-transform of the following function?

$$f[n] = \{2, 1, 4, 0, 4\}.$$

By Eq. 9.6, the Z-transform is

$$\begin{aligned} F(z) &= 2z^0 + z^{-1} + 4z^{-2} + 0z^{-3} + 4z^{-4} \\ &= 2 + z^{-1} + 4z^{-2} + 4z^{-4} \end{aligned}$$

Example 9.3: Finding the Inverse Z-Transform by Inspection

What is the inverse Z-transform of the following function?

$$F(z) = 1 - z^{-4}$$

By inspection, the inverse Z-transform is

$$f[n] = \{1, 0, 0, 0, -1\}.$$

We cannot always find the inverse Z-transform by immediate inspection, in particular if the Z-transform is written as a ratio of polynomials of z. However, sometimes we can use the **Binomial theorem** to convert such cases into a single (but sometimes infinite-length) polynomial of z. The Binomial theorem is

$$\sum_{n=0}^{\infty} a^n = \frac{1}{1-a} \tag{9.8}$$

Example 9.4: Using Binomial Theorem to Find Inverse Z-Transform

What is the inverse Z-transform of the following function?

$$F(z) = \frac{z^{-2} - z^{-7}}{1 - z^{-1}}$$

We can apply the Binomial theorem to write

$$\begin{aligned} \frac{1}{1-z^{-1}} &= 1 + z^{-1} + z^{-2} + z^{-3} + z^{-4} + \ldots \\ \implies F(z) &= (z^{-2} - z^{-7})(1 + z^{-1} + z^{-2} + z^{-3} + z^{-4} + \ldots) \\ &= z^{-2} + z^{-3} + z^{-4} + z^{-5} + z^{-6}. \end{aligned}$$

We can then apply the inverse Z-transform by inspection and write

$$f[n] = \{0, 0, 1, 1, 1, 1, 1, 0, \ldots\}.$$

Just as with the Laplace transform, there are transform pairs between the time and z-domains, and furthermore many of them are similar to Laplace transform pairs. Common function pairs are published in tables and there is a Z-transform table in the Engineering Data Book; a brief list of common transform pairs is presented here in Example 9.5.

Example 9.5: Sample Z-Transform Pairs

$$\begin{array}{rcc} f[n], n > 0 & & F(z) \\ 1 \text{ for } n = 0 \text{ only} & \Longleftrightarrow & 1 \\ 1 \text{ for } n = k \text{ only} & \Longleftrightarrow & z^{-k} \\ 1 \text{ for all } n & \Longleftrightarrow & \dfrac{1}{1 - z^{-1}} \end{array}$$

9.3.1 Z-Transform Properties

The Z-transform has several properties that we will find relevant in this module. They are as follows:

1. **Linearity** - the Z-transform of a sum of scaled functions is equal to the sum of scaled Z-transforms of the individual functions, i.e.,

$$af_1[n] + bf_2[n] \Longleftrightarrow aF_1(z) + bF_2(z). \tag{9.9}$$

2. **Time Shifting** - a delay in the time-domain corresponds to multiplication by a power of z in the z-domain, i.e.,

$$f_1[n] = f_2[n-k] \Longleftrightarrow F_1(z) = z^{-k}F_2(z). \tag{9.10}$$

3. **Convolution** - convolution in the time-domain is equal to multiplication in the z-domain.

$$f_1[n] * f_2[n] \Longleftrightarrow F_1(z)\,F_2(z). \tag{9.11}$$

Example 9.6: Z-Transform of a Step Function

Let's prove the Z-transform of the discrete step function $u[n] = 1, \forall n \geq 0$, knowing that the Z-transform of the discrete delta function is 1. We let $f[n] = u[n]$, and then write the delta function in terms of step functions, i.e.,

$$\delta[n] = u[n] - u[n-1] = f[n] - f[n-1].$$

We take the Z-transform of the left and right sides and get

$$\begin{aligned} 1 &= F(z) - z^{-1}F(z) \\ \implies F(z) &= \frac{1}{1 - z^{-1}} \end{aligned}$$

9.4 Z-Domain Transfer Functions

We saw in the Z-transform properties that **convolution in the (discrete) time domain is equal to multiplication in the z-domain**. We can therefore apply this to LSI systems by taking the Z-transform of the input signal $x[n]$ and the impulse response $h[n]$ to write the Z-transform of the output signal, i.e.,

$$Y(z) = \mathcal{Z}\{y[n]\} = \mathcal{Z}\{x[n] * h[n]\} = \mathcal{Z}\{x[n]\}\,\mathcal{Z}\{h[n]\} = X(z)\,H(z) \quad (9.12)$$

$$\implies Y(z) = X(z)\,H(z) \quad (9.13)$$

where $H(z)$ is referred to as the **pulse transfer function**, as it is also the system output when the time-domain input is a unit impulse. However, by our convention we will just refer to $H(z)$ as the **transfer function**.

So, we can find the time-domain output $y[n]$ of an LSI system by

1. Transforming $x[n]$ and $h[n]$ into the z-domain.
2. Finding the product $Y(z) = X(z)\,H(z)$.
3. Taking the inverse Z-transform of $Y(z)$.

We have already seen that time-domain analysis is simpler in the discrete-time case than in the continuous-time case. This is because we avoid having to solve integrations when modelling LSI components, so we can write the time-domain output $y[n]$ directly as a function of the time-domain input $x[n]$, as well as *previous*

time-domain outputs (i.e., there can be **feedback**). We call this formulation a **difference equation**. We will consider difference equations in greater detail in later lessons, but for now we consider a simple example of finding $H(z)$.

Example 9.7: Determining $H(z)$ From a Difference Equation

Find the transfer function $H(z)$ of the LSI system described by the following difference equation:

$$y[n] = -\frac{1}{2}y[n-2] + x[n] + x[n-1]$$

We take the Z-transform of both sides and re-arrange as follows:

$$Y(z) = -\frac{1}{2}z^{-2}Y(z) + X(z) + z^{-1}X(z)$$

$$\implies H(z) = \frac{Y(z)}{X(z)} = \frac{1+z^{-1}}{1+\frac{1}{2}z^{-2}} = \frac{z^2+z}{z^2+\frac{1}{2}}.$$

We note that the convention is to write $H(z)$ in terms of *positive* powers of z, as this facilitates analysis (as we will see in future lessons).

9.5 Summary

- The output of a **linear shift invariant** (LSI) system can be found by convolving the time-domain input with the system impulse response.
- We can use the Z-transform to convert an LSI system into the z-domain, where the impulse response is known as the transfer function. Convolution in the discrete time-domain is equivalent to multiplication in the z-domain.
- LSI circuits can be implemented using **adders**, **multipliers**, and **delays**. We can often write the time-domain system output directly from inspection of the implementation.

9.6 Further Reading

- Section 4.5 and Sections 7.1 to 7.5 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

9.7 End-of-Lesson Problems

1. Determine the Z-transforms of the following sequences:
   (a) $f_1[n] = \{1, 1, 0, 0, 0, \ldots\}$
   (b) $f_2[n] = \{1, 1, 1, 1, 1, \ldots\}$
   (c) $f_3[n] = \{1, 0.5, 0.25, 0.125, \ldots\}$
   (d) $f_4[n] = f_1[n] + f_2[n]$

2. Determine the time-domain sequences that correspond to the following Z-transforms:
   (a) $F_1(z) = \frac{1-z^{-5}}{1-z^{-1}}$
   (b) $F_2(z) = \frac{1+z^{-4}}{1-z^{-4}}$

3. Given $f_1[n] = \{5, 4, 3, 2, 1\}$ and $f_2[n] = \{0, 5, 4, 3, 2, 1\}$, show that $F_2(z) = z^{-1}F_1(z)$.

4. Consider the input $x[n] = \{1, 1, 2, 1, 2, 2, 1, 1\}$ to a filter with impulse response $h[n] = \{1, 2, -1, 1\}$. What is the discrete-time output $y(n)$?

5. Find the Z-transform of the convolution of $x[n] = 3\delta[n] + 2\delta[n-1]$ and $h[n] = 2\delta[n] - \delta[n-1]$.

6. Find the transfer function $H(z)$ corresponding to the moving average system in Example 9.1.

7. Find the difference equation and transfer function corresponding to the discrete integrator LSI circuit shown in Fig. 9.3.

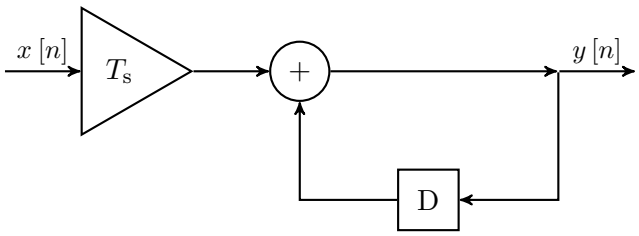


Figure 9.3: Discrete integrator circuit.

Lesson 10

Stability of Digital Systems

Now that we have established the Z-transform as the discrete-time equivalent of the Laplace transform, we can proceed to study digital systems in a similar way to analogue systems. We saw that stability was an important consideration in the design of analogue systems, and that is also the case for digital systems. In this lesson, we consider the stability of digital systems. z-domain analysis makes this possible by looking at the roots of the system's transfer function.

10.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Use** a digital system's transfer function to identify its poles and zeroes.
2. **Apply** the criterion for digital system stability.

10.2 Digital Poles and Zeros

Just as in the Laplace domain, we can write a transfer function $H(z)$ as a fraction of two polynomials, i.e.,

$$H(z) = \frac{b[M]\,z^{-M} + b[M-1]\,z^{1-M} + \ldots + b[1]\,z^{-1} + b[0]}{a[N]\,z^{-N} + a[N-1]\,z^{1-N} + \ldots + a[1]\,z^{-1} + 1}, \tag{10.1}$$

where for notational purposes we find it useful to write the transfer function with negative powers of z. We still have that the numerator is a Mth order polynomial with coefficients $b[n]$, and the denominator is a Nth order polynomial with coefficients $a[n]$, but generally we have $a[0] = 1$ (which we will also see why in later lessons). We will find that coefficient indexing in the form of $b[n]$ and $a[n]$ is more

convenient for digital systems than the form b_n and a_n. Unlike analogue systems, we have no constraint on the values of M and N for a system to be real, but in practice we often assume that $M = N$.

If we factorize Eq. 10.1 into its roots, then we can re-write $H(z)$ as a ratio of products, i.e.,

$$H(z) = K\frac{(z - z_1)(z - z_2)\dots(z - z_M)}{(z - p_1)(z - p_2)\dots(z - p_N)} \tag{10.2}$$

$$= K\frac{\prod_{i=1}^{M} z - z_i}{\prod_{i=1}^{N} z - p_i} \tag{10.3}$$

where we emphasise that the coefficient on each z in this form is 1. We now see **poles** p_i and **zeros** z_i. They carry the same meaning as for analogue system. When z is equal to any z_i, the transfer function $H(z) = 0$. When z is equal to any p_i, the transfer function $H(z)$ will be infinite. Unfortunately, the symbols for the variable z and the zeros z_i are very similar, so we will need to be very careful to be sure that the meaning is clear for a given context.

We will find it insightful to plot poles and zeros on the z-domain, i.e., to make pole-zero plots.

Example 10.1: First Digital Pole-Zero Plot

Plot the pole-zero plot for the transfer function

$$H(z) = \frac{(z - 0.9)(z^2 - z + 0.5)}{(z - 0.2)(z^2 + z + 0.5)}$$

First we need to figure out the poles and zeros, i.e., the roots of the transfer function. We can see that the numerator is zero when $z = 0.9$ or $z = 0.5 \pm 0.5j$, and the denominator is zero when $z = 0.2$ or $z = -0.5 \pm 0.5j$.

We draw a 2D axes, where the real components of z lie along the horizontal axis and the imaginary components of z lie along the vertical axis. The convention is also to draw a circle of radius 1 that is centred around the origin, i.e., where $|z| = 1$ (the motivation for doing so will soon be clear). The plot is shown in Fig. 10.1.

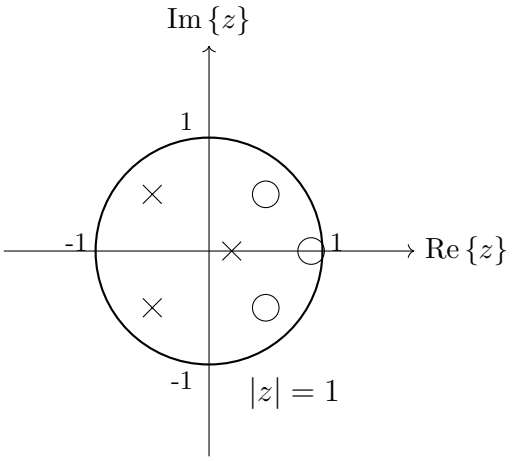


Figure 10.1: Pole-zero plot for Example 10.1.

10.3 Poles and Stability

In Lesson 3, we tried to infer information about the time-domain output $y(t)$ based on the locations of the poles and zeros. Here, this will not be as necessary, as we can already write the output $y[n]$ as a difference equation. However, the notion of stability is still important, and the poles once again play a role.

We stated in Lesson 3 that a system is considered to be **stable** if its impulse response tends to zero in the time domain. However, for digital systems, it is more convenient for us to describe stability from the context of the input signal $x[n]$ and output signal $y[n]$. Therefore, we will use the notion of **bounded input and bounded output** (BIBO) stability. A system is **BIBO stable** if a bounded input sequence yields a bounded output sequence. An input sequence $x[n]$ is bounded if each element in the sequence is smaller than some value A. An output sequence $y[n]$ corresponding to $x[n]$ is bounded if each element in the sequence is smaller than some value B.

Our practical criterion for BIBO stability is as follows. A *system is BIBO stable if all of the poles lie inside the* $|z| = 1$ *unit circle* (hence why we draw the circle on a z-domain pole-zero plot!). If there is at least 1 pole directly on the unit circle, then the system is said to be **conditionally stable** (as it will depend on the input used). We show the digital system stability criterion visually in Fig. 10.2. The motivation for using the unit circle for stability in the z-domain will become clearer once we discuss the frequency response of digital systems in Lesson 11.

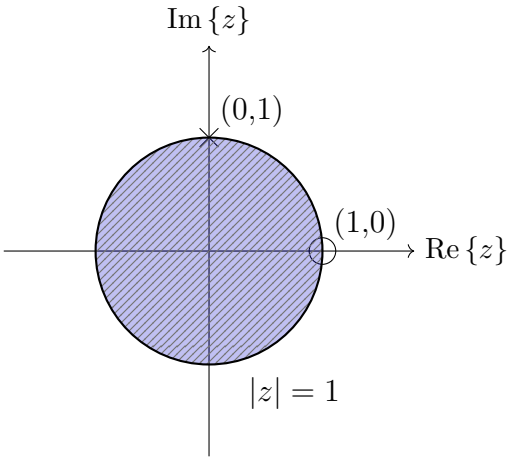


Figure 10.2: Pole-zero plot highlighting where poles must be located for system to be stable. There is pole on the $|z| = 1$ circle in this example, so this system is conditionally stable.

10.4 Summary

- Digital filters also have the concept of **poles** and **zeros**.
- We use the **BIBO stability criterion** for digital systems, such that a system is stable if any bounded input leads to a bounded output.
- We can infer stability of a digital system from whether all of its poles lie inside the $|z| = 1$ unit circle.

10.5 Further Reading

- Section 5.7 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

10.6 End-of-Lesson Problems

1. Find the transfer function $H(z)$ of the LSI system described by the following difference equation:

$$y[n] = x[n] + 0.6y[n-1] - 0.5y[n-2].$$

Determine the poles and zeros of the system, draw a pole-zero plot, and state whether it is stable.

2. Draw a pole-zero plot and comment on the stability for the transfer function of the moving average system:

$$H(z) = \frac{1+z}{2z}.$$

3. Draw a pole-zero plot and comment on the stability for the transfer function of the discrete integrator:

$$H(z) = \frac{T_s z}{z-1}.$$

4. Determine the z-domain transfer function $H(z)$ of the LSI system described by the pole-zero plot in Fig. 10.3. State whether the system is stable.

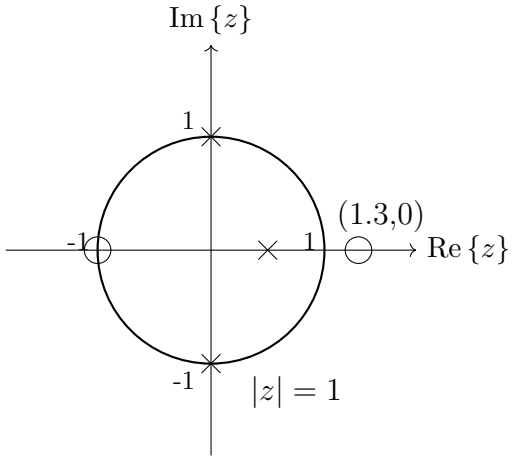


Figure 10.3: Pole-zero plot for Question 4.

Lesson 11

Digital Frequency Response

In Lesson 10, we saw how a LSI system's pole-zero plot enabled us to describe the stability of a system. Unlike analogue systems, the stability of digital systems relies on the magnitude of the poles. In this lesson, we will focus on the behaviour of LSI systems in response to periodic inputs of a particular frequency. This will enable our discussions of digital filters in Lesson 12, and will also help to justify the digital stability criterion. Similar to how we converted between the Laplace and Fourier transforms for analogue systems, we will find it convenient to introduce the discrete-time Fourier transform from the Z-transform. Once again, this translation is straightforward, and we can then use the z-domain pole-zero plot to determine the frequency response of a system.

11.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the Discrete-Time Fourier transform and its relation to the Z-transform.

2. **Analyse** the frequency response of digital systems.

3. **Determine** and **Analyse** magnitude and phase plots of digital systems.

11.2 Digital Frequency Response

Similarly to what we saw for analogue systems, the **frequency response** of an LSI system is its output in response to a sinusoid input of unit magnitude and some specified frequency. We still show the frequency response in two plots (1 for the

magnitude and 1 for the **phase**) as a function of the input frequency. System design typically targets a specific frequency or range of frequencies.

To find the *digital* frequency response, we recall the definition of the Z-transform:

$$F(z) = \sum_{k=0}^{\infty} f[k]\, z^{-k}, \tag{11.1}$$

for complex z. Let us specify z is *polar* coordinates, such that $z = re^{j\Omega}$, i.e., we fix the magnitude to r and the angle Ω. We can then re-write the Z-transform as

$$F\left(re^{j\Omega}\right) = \sum_{k=0}^{\infty} f[k]\, r^{-k} e^{-jk\Omega}. \tag{11.2}$$

When we set $r = 1$, then any point $z = e^{j\Omega}$ lies on the unit circle, and this special case of the Z-transform is known as the **Discrete-Time Fourier Transform** (DTFT), i.e.,

$$F\left(e^{j\Omega}\right) = \sum_{k=0}^{\infty} f[k]\, e^{-jk\Omega}. \tag{11.3}$$

The angle Ω is the angle of the unit vector measure from the positive real z-axis. It denotes *digital* radial frequency and is *measured in radians per sample* ($\frac{\text{rad}}{\text{sample}}$). We refer to $F\left(e^{j\Omega}\right)$ as the spectrum of $f[n]$ or as its frequency response. Our convention for writing the DTFT will include $F\left(e^{j\Omega}\right)$ or simply $F(\Omega)$. We can also write the **inverse DTFT** as

$$f[k] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F\left(e^{j\Omega}\right) e^{jk\Omega} d\Omega. \tag{11.4}$$

We recall from Lesson 10 that we can write an LSI system's transfer function $H(z)$ as a ratio of products that included its poles and zeros. We can now re-write the transfer function using $z = e^{j\Omega}$ as

$$H\left(e^{j\Omega}\right) = K \frac{\prod_{i=1}^{M} e^{j\Omega} - z_i}{\prod_{i=1}^{N} e^{j\Omega} - p_i}. \tag{11.5}$$

Just as we used $H(j\omega)$ to determine the frequency response of an analogue system, we can use $H\left(e^{j\Omega}\right)$ to determine the frequency response of a digital LSI system. $H\left(e^{j\Omega}\right)$ is a function of vectors from the system's poles and zeros *to the unit circle* at angle Ω. Thus, from a pole-zero plot, we can geometrically determine the magnitude and phase of the frequency response.

Even though the z-domain vectors are pointing somewhere different than in the s-domain case (i.e., to the $|z| = 1$ unit circle instead of the imaginary s-axis),

the *approach for calculating the frequency response is the same*. Thus, we find the magnitude of the frequency response as

$$|H\left(e^{j\Omega}\right)| = |K|\frac{\prod_{i=1}^{M}|e^{j\Omega} - z_i|}{\prod_{i=1}^{N}|e^{j\Omega} - p_i|}. \tag{11.6}$$

In words, the **magnitude of the frequency response** (MFR) $|H\left(e^{j\Omega}\right)|$ is equal to *the gain multiplied by the magnitudes of the vectors corresponding to the zeros, divided by the magnitudes of the vectors corresponding to the poles*.

If we assume that the gain $K > 0$, then the phase of the frequency response is

$$\angle H\left(e^{j\Omega}\right) = \sum_{i=1}^{M}\angle(e^{j\Omega} - z_i) - \sum_{i=1}^{N}\angle(e^{j\Omega} - p_i). \tag{11.7}$$

In words, the **phase angle of the frequency response** (PAFR) $\angle H\left(e^{j\Omega}\right)$ is equal to *the sum of the phases of the vectors corresponding to the zeros, minus the sum of the phases of the vectors corresponding to the poles*.

Notably, both the magnitude and phase of the frequency response repeat every 2π. This is clear to see using Euler's formula:

$$e^{j(\Omega+2\pi)} = e^{j\Omega}e^{j2\pi} \tag{11.8}$$

$$= e^{j\Omega}(\cos(2\pi) + j\sin(2\pi)) = e^{j\Omega}(1+0) = e^{j\Omega}. \tag{11.9}$$

Furthermore, due to usual symmetry of poles and zeros about the real z-axis, the frequency response is also symmetric about $\Omega = \pi$. Thus, *we only need to find the magnitude and phase over one interval of* π. It is sometimes common (e.g., in MATLAB) to normalise the frequency response by π, such that the normalised frequency range is 0 to 1.

We recall that sampling resulted in "infinite" copies of our spectrum in the frequency domain. One perspective to understand this is that, as we increase Ω, we cycle around and around the unit circle. If we sample an analogue signal fast enough to avoid aliasing (i.e., at double the signal bandwidth), then we guarantee that we encounter all of the unique signal components within one *half* circle.

11.3 Digital Frequency Response Example

In this section, we consider an example of determining digital frequency responses from a system's transfer function via the z-domain pole-zero plot.

Example 11.1: Simple Digital High Pass Filter

Consider this transfer function of a simple high pass filter:

$$H(z) = \frac{z-1}{z+0.5},$$

which has a pole at $z = -0.5$ and a zero at $z = 1$. We sketch the pole-zero plot and draw a vector from the pole to an arbitrary (usually positive) Ω along the unit circle in Fig. 11.1.

Rather than try to write a general equation for the magnitude and phase of the frequency response for any frequency Ω, we will rely on evaluating at several test frequencies and making a rough sketch from those. Due to the behaviour of the frequency response around the unit circle, we only need to consider frequencies between 0 and π. A good first choice of test frequencies is to include these two and at least one more. We will consider $\pi/2$. If there were many poles or zeros, particularly near the unit circle, then additional points may be needed.

At $\Omega = 0$, we have $|H\left(e^{j\Omega}\right)| = 0$ and $\angle H\left(e^{j\Omega}\right) = \pi/2 \frac{\text{rad}}{\text{sample}}$. For the angle, note the angle of the zero-vector as $\Omega \to 0$.

At $\Omega = \pi/2$, we have $|H\left(e^{j\Omega}\right)| = 1.265$ and $\angle H\left(e^{j\Omega}\right) = 1.257 \frac{\text{rad}}{\text{sample}}$.

At $\Omega = \pi$, we have $|H\left(e^{j\Omega}\right)| = 4$ and $\angle H\left(e^{j\Omega}\right) = 0$.

From these points we sketch the frequency response in Fig. 11.2. We have chosen to plot linear magnitudes and radial phase angles over linear radial frequency. However, it is also possible to plots over ordinary frequency or over logarithmic scales (we will see that MATLAB does both). Recall that we can convert linear filter gain G_{linear} to dB gain G_{dB} via

$$G_{\text{dB}} = 20 \log_{10} G_{\text{linear}}.$$

11.4 Summary

- The **frequency response** of a system is its response to an input with unit magnitude and a fixed frequency. The response is given by the magnitude and phase of the system output as a function of the input frequency.

- The **discrete-time Fourier transform** (DTFT) is the Z-transform evaluated on the $|z| = 1$ unit circle at some frequency Ω.

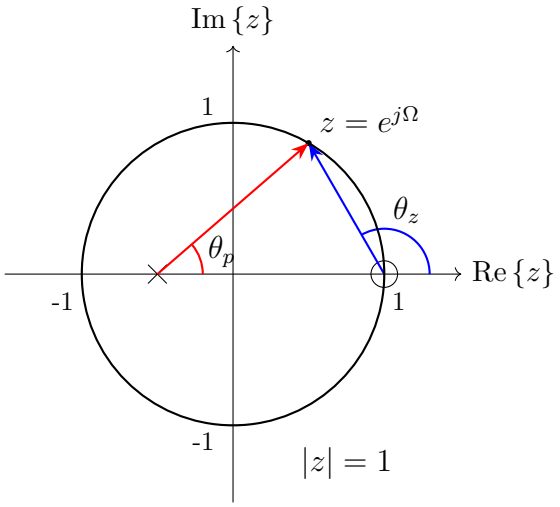
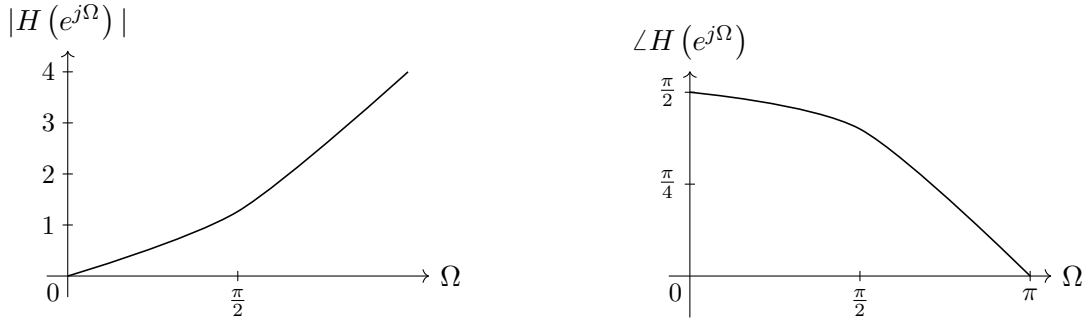


Figure 11.1: Pole-zero plot for Example 11.1.



(a) Magnitude plot.

(b) Phase plot.

Figure 11.2: Sketch of frequency response for Example 11.1

- The frequency response can be found geometrically by assessing the vectors made by the poles and zeros of the system's transfer function to the $|z| = 1$ unit circle.

11.5 Further Reading

- Sections 6.1, 6.2, and 7.6 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

11.6 End-of-Lesson Problems

1. Consider a digital filter with 2 poles at $z = -0.9$ and 2 zeros at $z = 0$. Write the transfer function for this filter. What is the phase when $\Omega = \pi$?

2. Consider a digital filter with poles at $z = 0.5e^{\pm j\frac{\pi}{2}}$, and zeros at $z = -1$ and $z = 1$. From the pole-zero plot and *without* numerical calculations, determine the minimum magnitude of the frequency response and the angular frequency at which this occurs. Also determine the angular frequency where the magnitude of the frequency response is maximum.

3. Sketch the pole-zero plot, magnitude of the frequency response (at frequencies $\Omega = \{0, \pi/2, \pi\}$), and phase of the frequency response of the system with the transfer function:

$$H(z) = \frac{z + 0.5}{z - 0.5}.$$

Lesson 12

Filter Difference Equations and Impulse Responses

In Lesson 11 we presented the digital frequency response and how it depended on the locations of poles and zeros in the z-domain. We considered one example of a simple filter, but here we begin an in-depth discussion of digital filter design. We are going into more detail than what we saw for analogue filters in Lesson 5, because here we discuss how to implement a filter design in the time-domain and discuss the notion of finite versus infinite impulse responses. This lesson then goes on to define ideal digital filters and simple filter implementations. The fundamentals in this lesson will inform our discussion of practical digital filter design in the next lesson.

12.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the basic types and properties of digital filters.
2. **Understand** and **Compare** finite impulse response (FIR) and infinite impulse response (IIR) filters.
3. **Understand** the constraints on filter realisability.
4. **Analyse** digital filter stability.
5. **Design** and **Evaluate** digital filter implementations, including their frequency responses.

12.2 Filter Difference Equations

In Lesson 9 we introduced how LSI systems could be implemented in the time domain using LSI circuit components (i.e., multipliers, adders, and delays). An LSI implementation could be inferred from the system's **difference equation**, which writes the system output as a function of current input as well as previous inputs and outputs. We now re-visit this concept more formally. We may recall from Lesson 10 that we wrote the transfer function $H(z)$ in the form

$$H(z) = \frac{b[M]z^{-M} + b[M-1]z^{1-M} + \ldots + b[1]z^{-1} + b[0]}{a[N]z^{-N} + a[N-1]z^{1-N} + \ldots + a[1]z^{-1} + 1}. \tag{12.1}$$

It turns out that the general difference equation can be written as

$$y[n] = \sum_{k=0}^{M} b[k]\,x[n-k] - \sum_{k=1}^{N} a[k]\,y[n-k], \tag{12.2}$$

and the *real* coefficients $b[\cdot]$s and $a[\cdot]$s are the *same* as those that appear in Eq. 12.1 (and we note that $a[0] = 1$ so that there is no coefficient corresponding to $y[n]$). So, rather than use the time-domain impulse response $h[n]$, it is usually easier to convert directly between the transfer function $H(z)$ (written with *negative* powers of z) and the difference equation for the output $y[n]$. This is particularly suitable when we seek an implementation of the system.

Example 12.1: Proof that $y[n]$ can be Obtained Directly from $H(z)$

Let's prove using the Z-transform that Eq. 12.1 can lead us directly to Eq. 12.2.

Recall that $H(z) = Y(z)/X(z)$. We can substitute this into Eq. 12.1 and cross-multiply to get

$$b[M]z^{-M}X(z) + b[M-1]z^{1-M}X(z) + \ldots + b[1]z^{-1}X(z) + b[0]X(z) = Y(z) + a[N]z^{-N}Y(z) + a[N-1]z^{1-N}Y(z) + \ldots + a[1]z^{-1}Y(z).$$

We can now immediately take the inverse Z-transform and write

$$b[M]x[n-M] + b[M-1]x[n-M+1] + \ldots + b[1]x[n-1] + b[0]x[n] = y[n] + a[N]y[n-N] + a[N-1]y[n-N+1] + \ldots + a[1]y[n-1].$$

We re-arrange and combine terms into summations to arrive at Eq. 12.2.

In practice, we do not need to memorize this relationship, especially since we may not always have $H(z)$ in a form that directly matches Eq. 12.1. Thus,

we tend to work through the steps in this proof for a specific $H(z)$ to find the corresponding $y[n]$.

Given Eq. 12.2, and what we know about LSI implementations, the general implementation of a difference equation is presented in Fig. 12.1 where we have assumed that $M = N$. For a particular system, if there is a coefficient $b[k] = 0$ or $a[k] = 0$, then we simply omit the corresponding multiplier and its wiring from the implementation. We immediately see that a higher N or M increases the *complexity* (and hence the *cost*) of the implementation by requiring more components. We refer to the **order** of a filter as the greater of N and M. The number of **taps** in a filter is the minimum number of unit delay blocks required, which is also equal to the order of the filter.

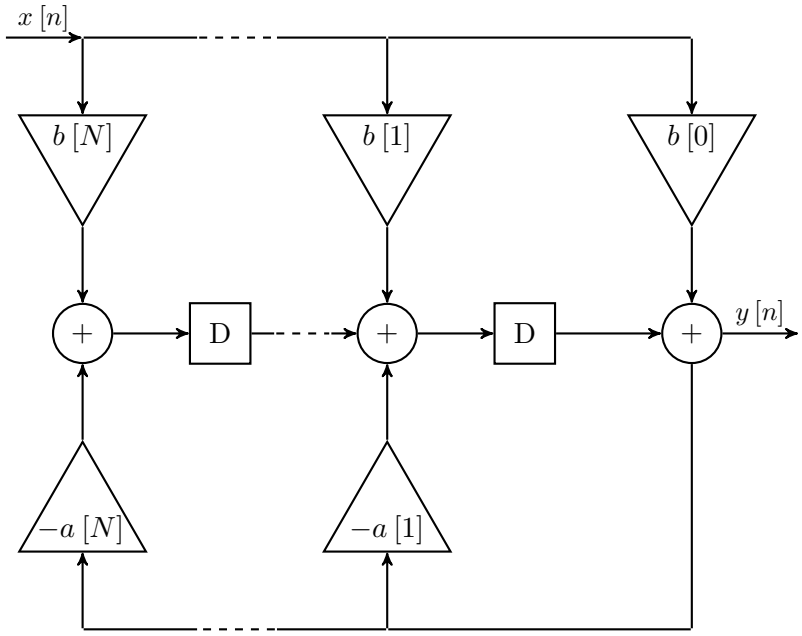


Figure 12.1: General difference equation implementation where $M = N$.

Example 12.2: Filter Order and Taps

Let's say that we need to implement a 2nd order filter. How many delay blocks will we need?

A 2nd order filter is a 2-tap filter and will need 2 unit delays. This is a rather simple example, but we emphasise the reuse of delay blocks for both the inputs and delayed outputs, as shown with adders in Fig. 12.1. While

it would also be possible to instead use 2 unit delays for the inputs and 2 separate unit delays for the outputs, this would use more components and hence be *more expensive* (more space, more energy, and more money).

EXTRA 12.1: State Variables

We see from the use of delay blocks in Fig. 12.1 that digital filter implementations have **memory** stored in them. We can refer to the states of internal memory using **state variables**, with one corresponding to the output at each delay block. It is possible to write equations for states as functions of other states and the system input and not explicitly include the function output. While relevant for digital computing, this concept is not necessary for our understanding of filters in this module.

12.2.1 Tabular Method for Difference Equations

Since the output to a digital system is generally written as a difference equation, it can be challenging to visualise the system output for a specific input. Given a system difference equation and its input $x[n]$, we can write the specific output $y[n]$ using the **tabular method**. We apply the tabular method as follows:

1. Starting with input $x[n]$, make a column for every input and output that appears in the system's difference equation.
2. Assume that every output and delayed input is initially zero, i.e., the filter is causal and there is initially no memory in the system (we also call such a system **quiescent**).
3. Fill in the column for $x[n]$ with the given system input for all rows needed, then fill in the delayed inputs (as relevant) with delayed versions of $x[n]$.
4. Evaluate $y[0]$ from the initial input, then propagate the value of $y[0]$ to delayed outputs (as relevant).
5. Evaluate $y[1]$ from $x[0]$, $x[1]$, and $y[0]$.
6. Continue evaluating the output and propagating delayed outputs as necessary.

The tabular method can also be used as an alternative approach for finding the time-domain impulse response $h[n]$.

Example 12.3: Tabular Method to Find Step Response

Use the tabular method to find the step response of the filter implementation in Fig. 12.2.

By inspection, we write the difference equation of this filter as

$$y[n] = x[n] + 0.6y[n-1] - 0.5y[n-2].$$

We complete a table for the output in Table 12.1, and sketch the output in Fig. 12.3.

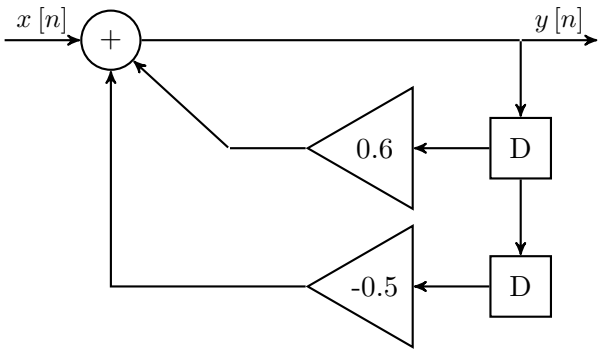


Figure 12.2: LSI Circuit for Example 12.3.

Table 12.1: Tabular method for Example 12.3

$x[n]$	$y[n-2]$	$y[n-1]$	$y[n]$
1	0	0	1
1	0	1	1.6
1	1	1.6	1.46
1	1.6	1.46	1.08
1	1.46	1.08	0.92
1	1.08	0.92	1.01
1	0.92	1.01	1.15

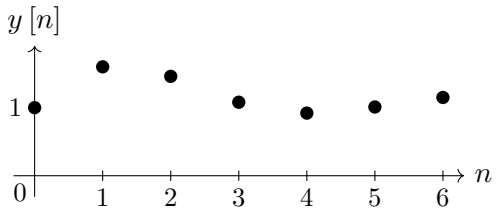


Figure 12.3: Ouput $y[n]$ for Example 12.3.

12.3 Infinite Impulse Response Filters

Now that we have more intuition about translating between a digital system's transfer function and its implementation, we can begin discussion about the two major classes of digital filters: those with **finite impulse responses** (FIR filters) and those with **infinite impulse responses** (IIR filters). We start with IIR filters because they have a more general form.

As implied by their name, IIR filters have impulse responses that are *infinite* in length because they are recursive, i.e., there are feedback terms associated with non-zero poles in the transfer function. They can be represented by the general form of the transfer function in Eq. 12.1, difference equation in Eq. 12.2, and implementation in Fig. 12.1. For an IIR filter, it is not possible to have a linear phase response (so there are different delays associated with different frequencies), and they are not always stable (depending on the exact locations of the poles). However, IIR filters are more efficient than FIR designs at controlling the gain of the response. And although the impulse response is technically infinite, in practice it often decays towards zero or can be **truncated** to zero (i.e., we assume that the response is $h[n] = 0$ beyond some finite value of n).

Example 12.4: IIR Filter Example

Given the IIR transfer function

$$H(z) = \frac{1 - 5z^{-1}}{1 - 0.995z^{-1}},$$

find the difference equation and draw an LSI implementation.

Using $H(z) = Y(z)/X(z)$, we can write

$$Y(z) - 0.995z^{-1}Y(z) = X(z) - 5z^{-1}X(z)$$
$$\implies y[n] = \boxed{x[n] - 5x[n-1] + 0.995y[n-1]}$$

Using the difference equation, we can draw the implementation shown in Fig. 12.4.

Finally, note what the coefficient 0.995 implies about the stability of this system. There is a pole very close to the unit circle, but the system is still stable.

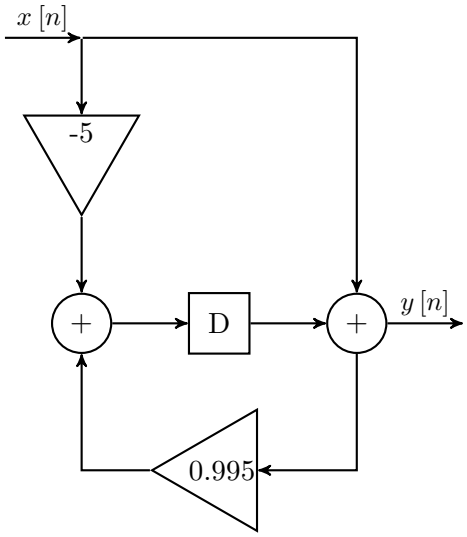


Figure 12.4: Implementation of the IIR filter in Example 12.4.

12.4 Finite Impulse Response Filters

FIR filters are non-recursive, i.e., they have *no feedback components*. From the general form of the transfer function in Eq. 12.1, we say that $a[k] = 0$ for $k \neq 0$. This means that the difference equation is simply

$$y[n] = \sum_{k=0}^{M} b[k]\, x[n-k], \tag{12.3}$$

and the implementation is just the "upper half" of Fig. 12.1, as shown in Fig. 12.5.

Since there is no feedback, we can readily write the impulse response $h[n]$ as

$$h[n] = b[n], \tag{12.4}$$

and this is what we show in Fig. 12.5 to emphasise that the impulse response comes *directly* from the knowledge of the numerator coefficients.

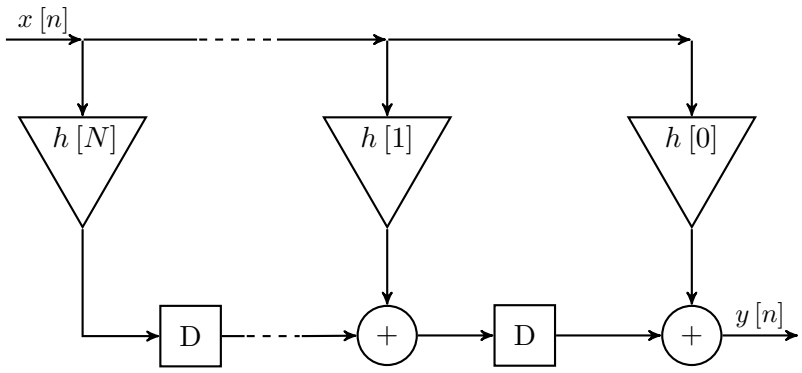


Figure 12.5: Generic LSI FIR Filter.

FIR filters are "finite" because their impulse response is finite in length and strictly zero beyond that, i.e., $h[n] = 0$ for $n > M$. Thus, *the number of filter taps dictates the length of an FIR impulse response.*

The pole-zero plot of an FIR filter is easy to identify and leads us to a universal property about FIR filter stability. To see this, let's simplify the transfer function in Eq. 12.1 for an FIR filter:

$$H(z) = b[M] z^{-M} + b[M-1] z^{1-M} + \ldots + b[1] z^{-1} + b[0], \tag{12.5}$$

since the denominator is simply 1. To write $H(z)$ as a function of its roots, we will find it more convenient to work with positive powers of z, so we multiply the numerator and denominator by z^M and then factor:

$$H(z) = \frac{b[M] + b[M-1] z + \ldots + b[1] z^{M-1} + b[0] z^M}{z^M} = K \frac{\prod_{k=0}^{M} (z - z_k)}{z^M} \tag{12.6}$$

We see that while there is no particular constraints on the locations of the zeros, all M poles are at the origin of the complex z-plane, which also places them inside of the $|z| = 1$ unit circle. Thus, **FIR filters are always stable**.

Finally, we note that FIR filters often have a **linear phase response**. This means that the phase shift at the output corresponds to a time delay.

Example 12.5: FIR Filter Example

Find the impulse response and transfer function of the FIR filter given by the implementation in Fig. 12.6. Write the transfer function with positive powers of z.

By inspection, we can write the difference equation as

$$y[n] = x[n] + 2x[n-1] + x[n-2],$$

and the impulse response is $\boxed{h[n] = \{1, 2, 1, 0, 0, \ldots\}}$. We can apply the Z-transform to write

$$Y(z) = X(z) + 2z^{-1}X(z) + z^{-2}X(z)$$
$$\implies H(z) = 1 + 2z^{-1} + z^{-2} = \boxed{\frac{z^2 + 2z + 1}{z^2}},$$

from which we could then also go on to find the pole-zero plot and frequency response.

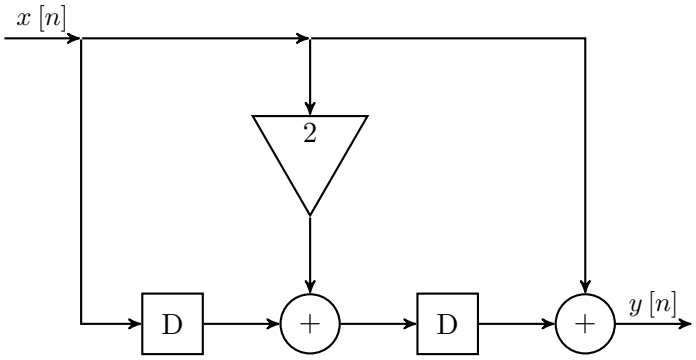


Figure 12.6: FIR filter for Example 12.5.

EXTRA 12.2: Filter Realisations

Our method for realising LSI implementations is taken directly from the difference equation. It is the most direct approach but not the only one. Section 7.5 of Lathi and Green discusses several methods of implementation, and there is further discussion specific to IIR and FIR filter realisations in Sections 8.1.5 and 8.2.2, respectively. For example, it is common to *partially* factor the transfer function and then implement a filter by cascading two or more filters together.

12.5 Ideal Digital Filters

We now focus our discussion to the types of digital filters by considering the ideal filter frequency responses, as we show in Fig. 12.7. As with analogue filters, this enables us to clearly define the different types of filters and helps us to establish the targets that we will try to realise with practical filters. Each ideal filter has unambiguous **pass bands**, which are ranges of frequencies that pass through the system without distortion, and **stop bands**, which are ranges of frequencies that are rejected and do not pass through the system at all. The **transition band** between stop and pass bands in ideal filters has a size of 0; transitions occur at single frequencies. The biggest difference with analogue frequency responses is that digital responses repeat every 2π.

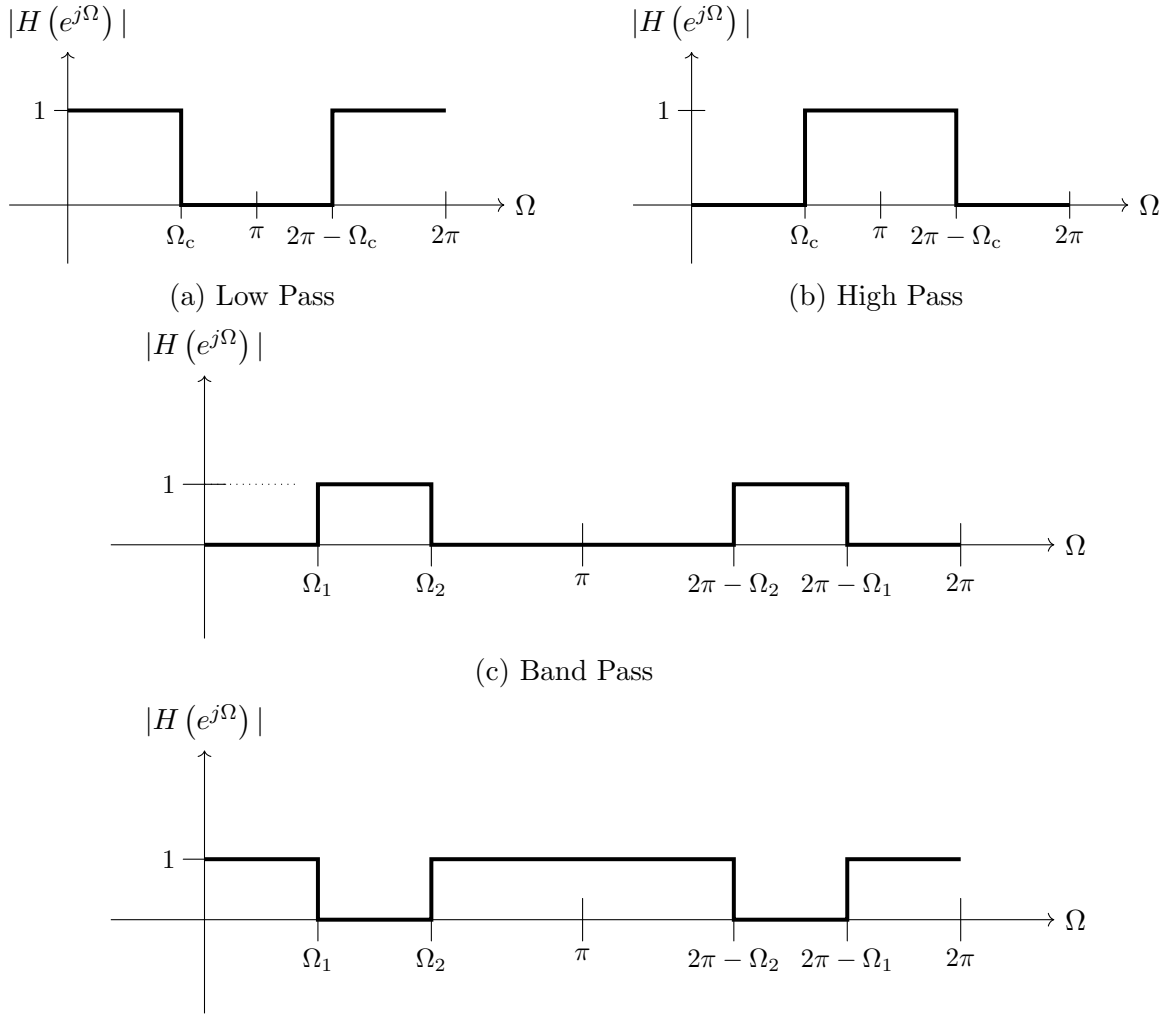
We re-iterate that the four main types of filter magnitude responses are as follows (where behaviour is defined over $0 \leq \Omega \leq \pi$, mirrored over $\pi \leq \Omega < 2\pi$, and repeated every 2π):

- **Low pass** filters pass frequencies less than cutoff frequency Ω_c and reject frequencies greater than Ω_c.
- **High pass** filters reject frequencies less than cutoff frequency Ω_c and pass frequencies greater than Ω_c.
- **Band pass** filters pass frequencies that are within a specified range, i.e., between Ω_1 and Ω_2, and reject frequencies that are either below or above the band within $[0, \pi]$.
- **Band stop** filters reject frequencies that are within a specified range, i.e., between Ω_1 and Ω_2, and pass all other frequencies within $[0, \pi]$.

These ideal responses appear to be fundamentally different from the ideal analogue responses that we considered in Lesson 5. However, we only really care about the behaviour over the *fundamental band* $[-\pi, \pi)$, over which the behaviour of ideal digital filters is *identical* to that of ideal analogue filters.

12.6 Simple Digital Filters

In Lesson 5, we went into rather great detail about *why* ideal analogue filters are ideal and not realisable. This is somewhat more cumbersome for ideal digital filters, but we will return to that discussion in the following lesson. For now, we focus on the design of *simple* (low order) digital filters, where we consider the intuitive placement of poles and zeros to obtain a target filter type that is also *stable*. Intuition



(a) Low Pass

(b) High Pass

(c) Band Pass

(d) Band Stop

Figure 12.7: Frequency response magnitudes of common ideal digital filters.

is somewhat easier for digital filters than for analogue filters because we only need to consider the response over the fixed and finite frequency band $[-\pi, \pi)$.

It's helpful to keep a few concepts in mind:

- To be physically realisable, complex poles and zeros need to be in conjugate pairs.
- We can place zeros anywhere, so we will often place them directly on the unit circle when we need a frequency or range of frequencies to be attenuated.
- Poles on the unit circle should generally be avoided. If we can keep all of

the poles at the origin, then our design is FIR; otherwise it is IIR and there is feedback. Poles are used to amplify the response in the neighbourhood of some frequency.

From these concepts, and what we know about the ideal filter responses, we can have the following intuition for each kind of filter:

- **Low pass** – place zeros at or near $\Omega = \pi$. Poles near $\Omega = 0$ can amplify the maximum gain or be used at a higher frequency to increase the size of the pass band.
- **High pass** – literally the inverse of low pass. Place zeroes at or near $\Omega = 0$. Poles near $\Omega = \pi$ can amplify the maximum gain or be used at a lower frequency to increase the size of the pass band.
- **Band pass** – place zeros at or near both $\Omega = 0$ and $\Omega = \pi$; such a filter must be at least second order. Place poles if needed to amplify the signal in the neighbourhood of the pass band.
- **Band stop** – place zeros at or near the stop band. These zeros must be complex so such a filter must be at least second order. Place poles at or near both $\Omega = 0$ and $\Omega = \pi$ if needed.

We will now highlight these ideas in an example.

Example 12.6: Simple High Pass Filter Design

Design a high pass FIR filter with 1 tap, a gain of 1 at high frequency, and a gain of 0 at low frequency. Sketch the pole-zero plot and LSI implementation.

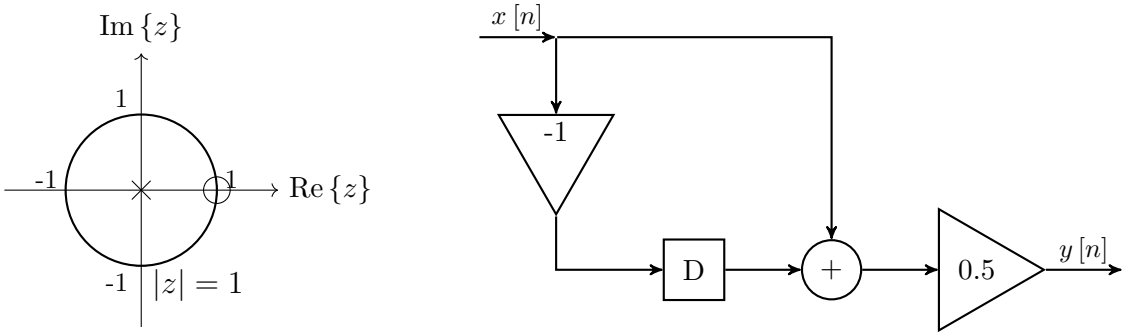
Since there is only one tap, we can have no more than one pole and one zero. It makes sense to place a zero on the unit circle at frequency $\Omega = 0$. Since the filter is FIR, the pole *must* be at the origin. With this orientation of pole and zero, we know that the magnitude of the frequency response at $\Omega = \pi$ is 2. So, we need to include a multiplier in the implementation that imposes a gain of $K = 0.5$. The transfer function is then

$$H(z) = 0.5\frac{z-1}{z} = 0.5(1 - z^{-1}),$$

so we know that the numerator coefficients are $b[0] = 1$ and $b[1] = -1$. We sketch the pole-zero plot and LSI implementation in Fig. 12.8. We emphasise that we don't need a multiplier for $b[0] = 1$ because it has a value of 1.

Consider what we could do if the design were IIR. We would then have

flexibility to place the pole somewhere besides the origin. This would be helpful if we have a specific multiplier available.



(a) Pole-zero plot.

(b) LSI implementation.

Figure 12.8: Filter design for Example 12.6.

12.7 Summary

- The difference equation of a digital filter can be obtained directly from the z-domain transfer function.
- The two major classes of digital filters are **infinite impulse response** (IIR) filters and **finite impulse response** (FIR) filters. IIR filters have feedback. FIR filters have no feedback and are always stable.
- We can design simple digital filters of any of the major filter types by starting with a pole-zero plot.

12.8 Further Reading

- Sections 6.3.2, 7.5.1, and 8.2.2 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

12.9 End-of-Lesson Problems

You might notice that many questions can be asked about any one digital filter. In particular, we have an LSI implementation, difference equation, transfer function, pole-zero plot, frequency response, filter type, filter class (FIR/IIR), and stability.

Even when not explicitly asked in one of the following problems or previous examples, you can consider all of the representations and analysis of a given system. This makes for excellent practice.

1. Consider the filter LSI implementation in Fig. 12.9. Determine the transfer function of the filter. Is it FIR or IIR? Sketch a pole-zero plot and state the type of filter (low pass, high pass, band pass, or band stop). Compare the implementation with that in Fig. 12.6. Comment on the difference (if any) in filter type.

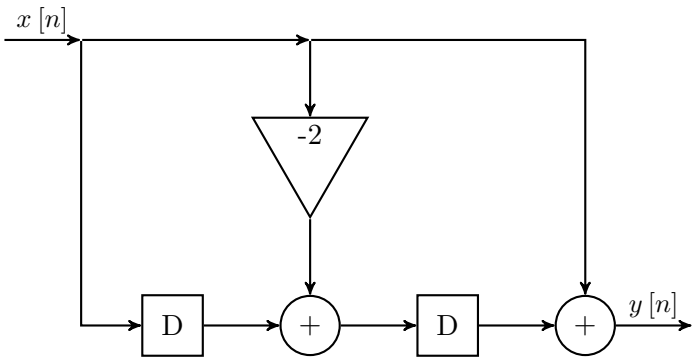


Figure 12.9: Filter for Question 1.

2. Consider the pole-zero plot in Fig. 12.10. Is the filter FIR or IIR? Determine the magnitude of the filter's frequency response. What type of filter is it (low pass, high pass, band pass, or band stop), and is it an effective filter of that type? Why or why not?

3. Consider the pole-zero plot in Fig. 12.11. Is the filter FIR or IIR? Determine the magnitude of the filter's frequency response. What type of filter is it (low pass, high pass, band pass, or band stop)? Comment on the quality of the filter.

4. Consider an LSI system in the form of Fig. 12.1 where $b[0] = 1$, $b[1] = 1$, $b[2] = 0.5$, $a[1] = -1$, and $a[2] = 0.5$. Sketch the pole-zero plot, state the type and order of filter, and comment on its stability.

5. Consider a digital filter with zeros at $z = 1 \pm j$ and poles at $z = -0.25 \pm j0.25$. Sketch the pole-zero plot and use it to find the magnitude of the filter's frequency response (at frequencies $\Omega = \{0, \pi/2, \pi\}$) and hence the type of filter. Write the system's transfer function and provide an LSI implementation.

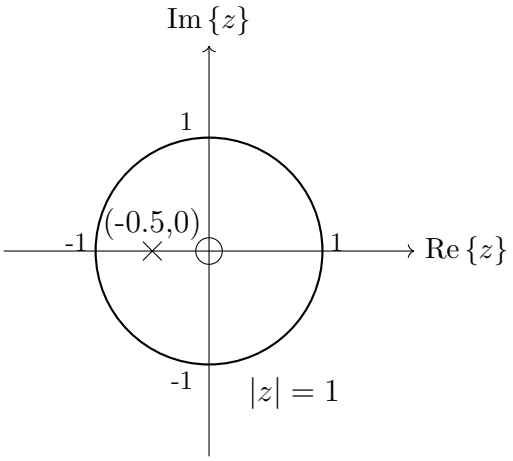


Figure 12.10: Pole-zero plot for Question 2.

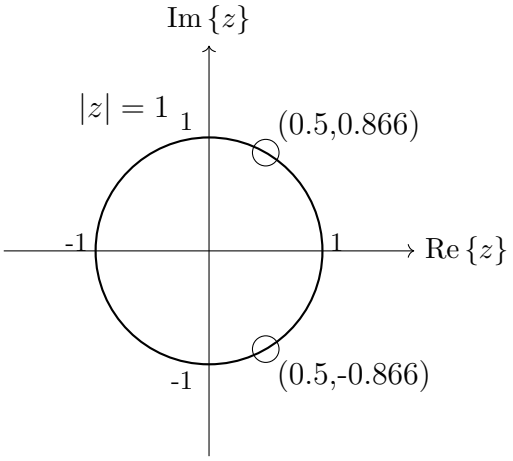


Figure 12.11: Pole-zero plot for Question 3.

6. Consider the following transfer function for a second-order system:

$$H(z) = \frac{z^2}{z^2 - z + 0.5}.$$

Determine the difference equation for this system and use the tabular method to find the impulse response. Is this system stable?

7. You are asked to design a simple digital filter (i.e., you can work out by hand where to place all of the poles and zeros). The filter must be stable, first-order, and high pass. The filter will process signals that are sampled at $f_s = 1\,\text{kHz}$ and have a transfer function gain of $K = 0.8$. The filter must have

ideal performance for constant magnitude input signals, and a gain of 0 dB for input signals at a frequency of 250 Hz. Where are the possible locations to place poles that will meet all of the design constraints? Sketch the pole-zero plot of each valid design.

8. You are asked to design a simple digital filter (i.e., you can work out by hand where to place all of the poles and zeros). The filter must be stable, first-order, FIR, and low pass. The filter will process signals that are sampled at $f_s = 1\,\text{kHz}$. The filter must have ideal stopband performance for input signals at a frequency of 500 Hz, and ideal passband performance for constant magnitude input signals. Where should poles and zeros be placed? What should be the value of the transfer function gain K? Sketch the pole-zero plot of the design.

Lesson 13

FIR Digital Filter Design

In this lesson, we build upon the digital filter fundamentals that we introduced in Lesson 12. We discuss practical digital filter design, including the roles of truncation and windowing for effective and efficient FIR filters. As with analogue filters, we focus our discussion on the implementation of low pass filters.

13.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the constraints on filter realisability.
2. **Design** and **Evaluate** practical digital filter implementations, including their frequency responses.
3. **Apply** Fourier series for digital filter design.

13.2 Realising the Ideal Filter Response

Our aim in practical digital filter design is to get *as close as we can* to ideal behaviour. We will recall the inverse DTFT from Eq. 11.4:

$$f[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} F\left(e^{j\Omega}\right) e^{jn\Omega} d\Omega. \tag{13.1}$$

We have the ideal magnitude of the impulse response in Fig. 12.7, i.e., $|H\left(e^{j\Omega}\right)| = 1$ for $|\Omega| < \Omega_c$. We will ignore the phase and assume in the following example that $\angle H\left(e^{j\Omega}\right) = 0$.

Example 13.1: Realisability of an Ideal Digital Filter

Let us consider the ideal low pass filter response that we presented in Lesson 12 and attempt to transform this response into the time domain to assess its realisability. Using the inverse DTFT, the *ideal* impulse response is as follows:

$$\begin{aligned} h_i[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H\left(e^{j\Omega}\right)e^{jn\Omega}d\Omega &= \frac{1}{2\pi}\int_{-\Omega_c}^{\Omega_c} 1\times e^{jn\Omega}d\Omega \\ &= \frac{1}{2\pi}\left[\frac{e^{jn\Omega}}{jn}\right]_{\Omega=-\Omega_c}^{\Omega=\Omega_c} \\ &= \frac{1}{j2\pi n}\left[e^{jn\Omega_c}-e^{-jn\Omega_c}\right] = \frac{1}{\pi n}\sin(n\Omega_c)\frac{n\Omega_c}{n\Omega_c} \\ &= \frac{\Omega_c}{\pi}\operatorname{sinc}(n\Omega_c) \end{aligned}$$

This is analogous to the inverse Fourier transform of the ideal analogue low pass frequency response that we performed in Example 5.1. We have a sampled scaled sinc function, which has non-zero values for $n < 0$. This implies that the ideal system needs to respond to an input before that input is applied, so this ideal response (and all ideal responses) is $\boxed{\textit{unrealisable}}$.

We can plot $h_i[n]$ over discrete-time samples without knowing the sampling frequency ω_s, however ω_s would be needed if we wanted to indicate the actual time between samples of $h[n]$. We plot $h_i[n]$ for $\Omega_c = \pi/3$ in Fig. 13.1.

EXTRA 13.1: Filter Realisation Methods

In the rest of this Lesson, we focus on FIR filter design using the *windowing* method, which tries to match the ideal filter response in the time domain. It is also possible to take a frequency domain approach, which is more intuitive for complicated frequency responses, but the steps are more involved and are thus outside the scope of this module. You can read more about the frequency domain approach in Section 8.2.6 of Lathi and Green.

There are also IIR filter implementations. These relate nicely to analogue filter design, because it is common for digital IIR filters to start with a realisable analogue filter (e.g., Butterworth) and then convert it into an equivalent

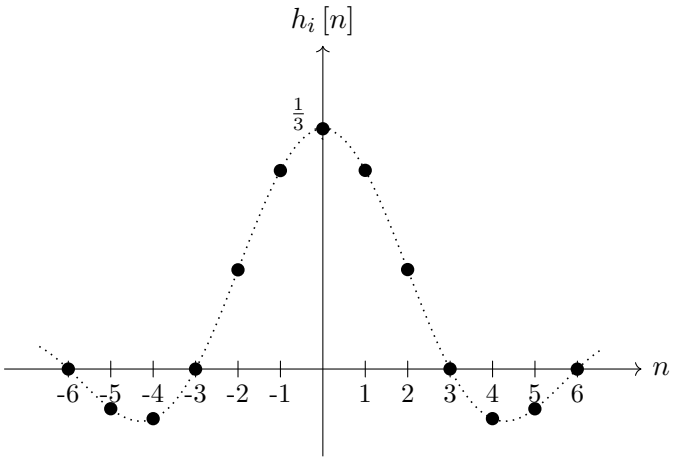


Figure 13.1: Impulse response $h_i[n]$ of ideal low pass digital filter in Example 13.1 for $\Omega_c = \pi/3$.

digital filter. This conversion uses the **bilinear transform**, which is outside the scope of this module but will be discussed in Lesson 16. You can read more about realising IIR filters in Section 8.1 of Lathi and Green.

13.3 Practical Digital Filters

Unlike the simple digital filters that we considered in Lesson 12, we pose that a *good* digital low pass filter will try to realise the (unrealisable) ideal response in Fig. 13.1 (or for whatever Ω_c as specified). We will try to do this with FIR filters, which are *always* stable and tend to have greater flexibility to implement different ideal frequency responses. Following similar arguments to what we used for realisable analogue filters in Lesson 5, we note the following:

- We need to introduce a delay to capture most of the ideal signal energy in causal time, i.e., use $h_i[n-k]\,u[n]$.
- We **truncate** the response according to a delay tolerance k_d, such that $h[n] = 0$ for $n > k_d$. Truncation also limits the complexity of the filter as a shorter filter corresponds to a smaller order.
- We **window** the response, which scales each sample in the impulse response according to a particular window $w[n]$. For example, a rectangular window is simply truncation alone, but other windows try to mitigate the negative effects of truncation.

We refer to the above approach as the **window method**, as our design process is starting with an ideal $h_i[n]$ and windowing this infinite time-domain response to obtain a realisable $h[n]$ that we can then immediately implement following the guidance of Lesson 12.

If we use a simple rectangular window (i.e., just truncate), then we are multiplying the ideal response by a rectangle function in the time domain. While we will not go into the mathematical details, the discrete-time rectangle function corresponds to a sinc function in the spectral domain (see Fig. 13.2), and multiplication in time corresponds to convolution in frequency, so we end up with non-ideal stop band behaviour. Nevertheless, the rectangle window is simple and is also known as the **Fourier series method**, because the windowed filter coefficients that remain are the first Fourier series coefficients that describe the *ideal* frequency-domain behaviour.

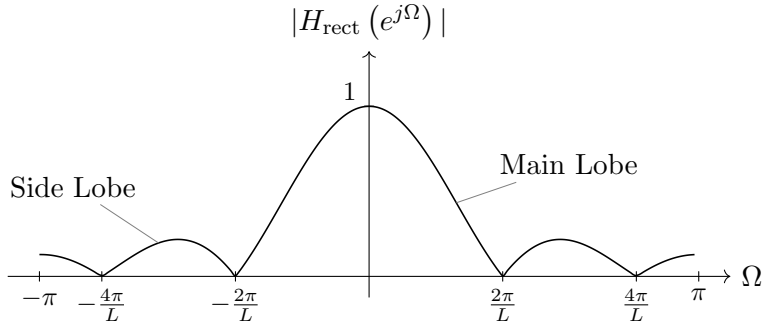


Figure 13.2: Magnitude of the frequency response of discrete-time rectangular window of length $L = 5$.

To discuss and compare windows, we need to understand **lobes**. The main lobe is the largest portion of the frequency response magnitude, and for a low pass filter is centred around the origin $\Omega = 0$. The side lobes are all of the other portions of the magnitude response. Picture the response in Fig. 13.2 where we have labelled the main lobe and one of the side lobes, and consider how it compares to the ideal response for a low pass filter. We observe a rather gradual roll-off in the pass band as the magnitude of the main lobe goes to 0, and all of the side lobes are contributing non-zero responses in the stop bands. Thus, the criteria that we are interested in are:

1. **Main lobe width** – the width in frequency of the main lobe ($4\pi/L$ for the rectangular window).

2. **Roll-off rate** – how sharply the main lobe decreases, measured in dB/dec (i.e., dB per decade).

3. **Peak side lobe level** – the peak magnitude of the largest side lobe relative to the main lobe, measured in dB.

Given these criteria, we are able to compare different window designs. We list several common windows and their (approximate) performance criteria in Table 13.1. We observe that the rectangular window, while simple, has a much poorer peak side lobe level than the other windows. The other windows have larger main lobe widths, which means that longer filters are needed to maintain a comparable pass band. The Hann and Hamming windows have very similar forms but different trade-offs between the roll-off rate and the peak side lobe level. The Blackman window has the best combination of roll-off rate and peak side lobe level but the largest main lobe width (thus requiring the longest implementation for given cut-off and sampling frequencies).

Table 13.1: Common FIR Window Functions. Windows are $w[n] = 0$ outside of $0 \leq n \leq L-1$.

Name	Window $w[n]$ for $0 \leq n \leq L-1$	Main Lobe Width	Roll-off Rate [dB/dec]	Peak Side Lobe Level [dB]
Rectangular	1	$\frac{4\pi}{L}$	-20	-13.3
Hann	$\frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{L-1}\right)\right]$	$\frac{8\pi}{L}$	-60	-31.5
Hamming	$0.54 - 0.46\cos\left(\frac{2\pi n}{L-1}\right)$	$\frac{8\pi}{L}$	-20	-42.7
Blackman	$0.42 - 0.5\cos\left(\frac{2\pi n}{L-1}\right) + 0.08\cos\left(\frac{4\pi n}{L-1}\right)$	$\frac{12\pi}{L}$	-60	-58.1

With the use of Table 13.1, combined with our knowledge of the ideal filter realisation and the discrete-time Fourier transform (DTFT), we can compare the time-domain implementations of practical filters with their spectra to visualise both their implementations and their performance.

Example 13.2: Practical FIR Filter Design

Design a low pass FIR filter for a cut-off frequency of $\Omega_c = \pi/2$ using $L = 7$ coefficients with a rectangular window. Compare the time-domain and magnitude spectra with that of a Hamming window.

From $\Omega_c = \pi/2$, we know that the ideal time domain response is $h_i[n] = \frac{1}{2}\operatorname{sinc}\left(\frac{n\pi}{2}\right)$. We need $L = 7$ coefficients from this function. They should be centred about $n = 0$, so we consider $n = \{0, \pm1, \pm2, \pm3\}$. By symmetry, we

only need to evaluate for non-negative values of n. We find that

$$\begin{aligned} h_i[0] &= 0.5 \\ h_i[1] &= 0.3183 = h_i[-1] \\ h_i[2] &= 0 = h_i[-2] \\ h_i[3] &= -0.1061 = h_i[-2] \end{aligned}$$

For the rectangular window, we just need to shift these coefficients so that the filter is causal. We get

$$h_{\text{rect}}[n] = \{-0.1061, 0, 0.3183, 0.5, 0.3183, 0, -0.1061\}, 0 \leq n \leq 6.$$

To apply a Hamming window, we just need to multiply each of these coefficients by the corresponding Hamming weights. For $L = 7$, we evaluate the Hamming weights as $w_{\text{Hamming}}[n] = \{0.08, 0.31, 0.77, 1, 0.77, 0.31, 0.08\}$, such that the filter is

$$h_{\text{Hamming}}[n] = \{-0.0085, 0, 0.245, 0.5, 0.245, 0, -0.0085\}, 0 \leq n \leq 6.$$

We plot a time-domain comparison of the two filters in Fig. 13.3(a), where we also include the envelope of the corresponding shifted sinc wave. We see that the Hamming weights are much smaller at the edges of the window.

To compare the spectra, we need to find the Z-transform of each filter and set $z = e^{j\Omega}$, or equivalently find the DTFT. We will find it easiest to find the spectra using the *unshifted* filters, which will have the same magnitude responses as the actual filters (since a shift in time corresponds to multiplication by a complex exponential in frequency). The frequency response of the rectangular window filter is then

$$\begin{aligned} H_{\text{rect}}\left(e^{j\Omega}\right) &= \sum_{n=-\infty}^{\infty} h_{\text{rect}}[n]\, e^{-jn\Omega} \\ &= -0.1061e^{j3\Omega} + 0.3183e^{j\Omega} + 0.5 + 0.3183e^{-j\Omega} - 0.1061e^{-j3\Omega} \\ &= 0.5 + 0.3183(e^{j\Omega} + e^{-j\Omega}) - 0.1061(e^{j3\Omega} + e^{-j3\Omega}) \\ &= 0.5 + 0.6366\cos(\Omega) - 0.2122\cos(3\Omega), \end{aligned}$$

where we have applied Euler's formula to write the complex exponentials as cosines. Similarly, we can show that the frequency response of the Hamming

window filter is

$$H_{\text{Hamming}}\left(e^{j\Omega}\right) = 0.5 + 0.49\cos(\Omega) - 0.017\cos(3\Omega).$$

We plot the two spectra and compare with the ideal response in Fig. 13.3(b) over the range $[-\pi, \pi]$.

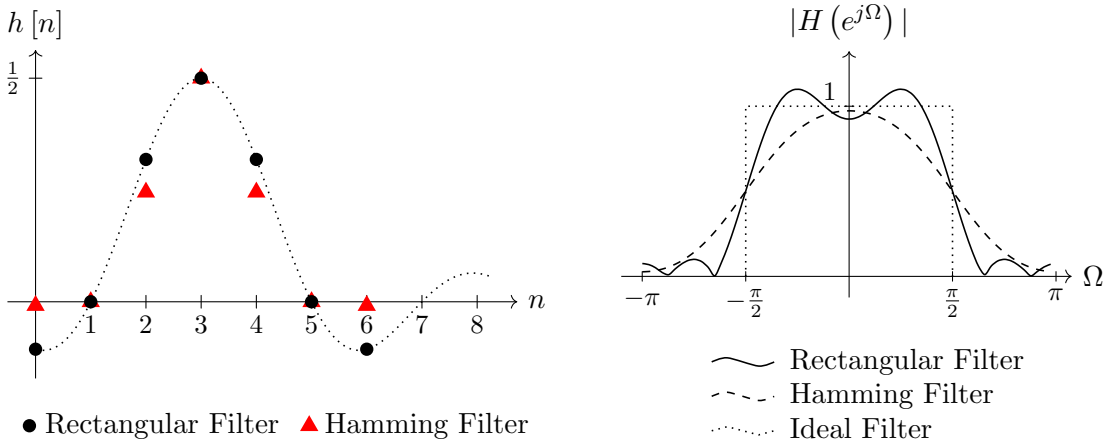


Figure 13.3: Responses of FIR filters designed in Example 13.2.

13.4 Designing Filters for Target Specifications

Before we discuss filter design for target specifications, we need to clarify the relationship between Hz and radial frequencies in digital signals. We note that the discrete-time radial frequency Ω is related to continuous-time radial frequency ω via $\Omega = \omega/f_s$, as long as $\Omega \leq \pi$ (otherwise there will be an alias at a lower frequency). From the definition of ω in terms of f, we can then write

$$\Omega = \frac{2\pi f}{f_s}, \tag{13.2}$$

from which we also know that we need $2f \leq f_s$ to avoid aliasing.

The target specifications for digital filters are somewhat different than those for analogue filters. Rather than define pass band and stop band gains, we instead define pass band and stop band **ripples**:

- The gain over the pass band, defined by the pass band frequency Ω_p, can vary about unity between $1 - \delta_p/2$ and $1 + \delta_p/2$. δ_p is the **pass band ripple**.

- The gain over the stop band, defined by the stop band frequency Ω_s, must be less than the **stop band ripple** δ_s.

We define the stop band ripple δ_s in dB analogously to the stop band gain for analogue filters, such that $G_\mathrm{dB} = 20\log_{10}\delta_\mathrm{s}$. The pass band ripple in dB is defined by the **pass band ripple parameter** r_p:

$$r_\mathrm{p} = 20\log_{10}\left(\frac{1+\frac{\delta_\mathrm{p}}{2}}{1-\frac{\delta_\mathrm{p}}{2}}\right). \tag{13.3}$$

Interestingly, when using the window method for filter design, we cannot independently specify pass band and stop band ripples. Generally, a filter's stop band ripple will equal half of its pass band ripple $\delta_\mathrm{s} = \frac{\delta_\mathrm{p}}{2}$. Furthermore, the ripple characteristics of the windowing filters are approximately *fixed* for the type of window and change very little with window length (i.e., filter order). The **transition band** $\Omega_\Delta = \Omega_\mathrm{s} - \Omega_\mathrm{p}$, i.e., the difference between the pass band and stop band frequencies, does change with filter length. All of these details are typically found in a table such as Table 13.2.

Table 13.2: Band Characteristics of Common FIR Window Functions.

Name	Main Lobe Width	Peak Side Lobe Level [dB]	Ω_Δ	$\delta_\mathrm{s} = \frac{\delta_\mathrm{p}}{2}$	G_s [dB]	r_p [dB]
Rectangular	$\frac{4\pi}{L}$	-13.3	$\frac{1.84\pi}{L-1}$	0.090	-20.9	1.57
Hann	$\frac{8\pi}{L}$	-31.5	$\frac{6.22\pi}{L-1}$	0.0064	-43.9	0.11
Hamming	$\frac{8\pi}{L}$	-42.7	$\frac{6.64\pi}{L-1}$	0.0019	-54.5	0.033
Blackman	$\frac{12\pi}{L}$	-58.1	$\frac{11.13\pi}{L-1}$	0.00017	-75.3	0.0030

In practice, for a given target specification, we need to trial the determined parameters to confirm that the specification can be met. Otherwise, we need to adjust the window length or change the window type until the specification can be confirmed.

Example 13.3: Specification for FIR Filters

Design an FIR low pass filter with cut-off frequency at 20 kHz, pass band frequency no less than 19 kHz, pass band ripple $r_\mathrm{p} \leq 0.1$ dB, stop band frequency no greater than 21 kHz, and stop band gain $G_\mathrm{s} \leq -50$ dB. Set the sampling frequency to be 4 times the cut-off frequency, i.e., 80 kHz.

From Table 13.2 we see that a Hamming window could meet this specification because $r_\mathrm{p} = 0.033\,\mathrm{dB}$ and $G_\mathrm{s} = -54.5\,\mathrm{dB}$. We need to translate each of the frequencies based on the sampling frequency:

$$\Omega_\mathrm{c} = \frac{2\pi(20000)}{80000} = \frac{\pi}{2},$$
$$\Omega_\mathrm{p} = \frac{2\pi(19000)}{80000} = \frac{19\pi}{40},$$
$$\Omega_\mathrm{s} = \frac{2\pi(21000)}{80000} = \frac{21\pi}{40},$$

so we find that $\Omega_\Delta = \Omega_\mathrm{s} - \Omega_\mathrm{p} = 0.05\pi$. We then apply the Hamming window formula for Ω_Δ and calculate that we need a filter of length $L = 134$. You would not be expected to find these terms by hand (!), though a computer program could be helpful.

13.5 Summary

- Realisable filters are constrained by causality and delay.
- The **windowing method** is an effective means to design filters based on truncations of the ideal filter response in the time domain.
- We can use the DTFT to determine the spectra of an FIR filter.

13.6 Further Reading

- Sections 8.2.3 to 8.2.5 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.

13.7 End-of-Lesson Problems

1. Design an FIR low pass filter with cut-off frequency $\Omega_\mathrm{c} = \pi/4$ using $L = 5$ coefficients with a rectangular window. Compare the time-domain and magnitude spectra with that of a Hamming window. *Note: You would not be expected to sketch the spectra in an exam, since it is a sum of sinusoids.*

2. Consider an FIR low pass digital filter implementation of the subwoofer design in Question 5.4. This filter was to have a pass band frequency of at least 100 Hz

and a stop band frequency of no more than 250 Hz. The stop band gain was to be $G_s \leq -30\,\text{dB}$. Assume that the signal is being sampled at 5 kHz.

(a) What are the radial pass band and stop band frequencies in $\frac{\text{rad}}{\text{sample}}$?

(b) What length of a Hann window filter is needed to achieve this performance?

(c) Can a rectangular window filter meet the specification?

Lesson 14

Discrete Fourier Transform and the FFT

We have seen in previous lessons that discrete-time systems produce continuous spectra. While we can readily plot continuous spectra, it would be more convenient for analysis and design to have discrete spectra, since computers are fully digital platforms. In this lesson we discuss the discrete Fourier transform (DFT) to provide discrete spectra. Importantly, the DFT can be efficiently computed with the FFT, which is a very widely used and practical algorithm. In the words of Lathi and Green, "*it is nearly impossible to exaggerate the importance*" of the DFT using the FFT algorithm.

14.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Apply** the Discrete Fourier Transform (DFT) to produce discrete spectra of fixed-length data.
2. **Understand** that the Fast Fourier Transform (FFT) efficiently evaluates the DFT.

14.2 Discrete Fourier Transform

Let us re-visit the discrete-time Fourier transform (DTFT), which takes a discrete-time signal and provides a continuous spectrum that repeats every 2π:

$$X\left(e^{j\Omega}\right) = \sum_{n=0}^{\infty} x\left[n\right] e^{-jn\Omega}. \tag{14.1}$$

The DTFT is defined for the *infinite*-length sequence $x\,[n]$ and gives a continuous spectrum with values at all frequencies. We saw in the previous lesson that we often have *finite*-length sequences, and for digital computation it would be convenient to have a finite number of frequencies. Furthermore, the IDTFT requires solving an integration that in general must be approximated. We can address all of these concerns at once. Let's suppose that our sequence $x\,[n]$ is of length N. The DTFT is then

$$X\left(e^{j\Omega}\right) = \sum_{n=0}^{N-1} x\,[n]\, e^{-jn\Omega}. \tag{14.2}$$

Now suppose that we sample the spectrum $X\left[e^{j\Omega}\right]$. We know that $X\left[e^{j\Omega}\right]$ repeats every 2π, so it should be sufficient to sample over the window $[0, 2\pi)$. We will find it convenient to take that same number of samples in the frequency domain as the length of the time-domain signal, such that we take N evenly-spaced samples of $X\left(e^{j\Omega}\right)$. These samples occur at multiples of the *fundamental frequency* $\Omega_0 = 2\pi/N$, i.e., at the frequencies

$$\Omega = \left\{0, \frac{2\pi}{N}, \frac{4\pi}{N}, \ldots, \frac{2\pi(N-1)}{N}\right\}, \tag{14.3}$$

which we can substitute into the DTFT and write the sampled form of the spectrum as

$$X\,[k] = \sum_{n=0}^{N-1} x\,[n]\, e^{-jnk\frac{2\pi}{N}}, \quad k = \{0, 1, 2, \ldots, N-1\}. \tag{14.4}$$

This sampled spectrum is known as the **forward discrete Fourier transform** (DFT). We emphasize that this is *not* the discrete-time Fourier transform, but *samples* of it over the interval $[0, 2\pi)$. The kth sample frequency in Hz is

$$f_k = k\frac{f_\mathrm{s}}{N}, \quad 0 \leq k \leq N-1 \tag{14.5}$$

where f_s is the sampling frequency and we emphasise the periodicity in the frequency domain. It is common to refer to the sampled DFT frequencies as **bins**. We can also reverse the transform using the inverse discrete Fourier transform (IDFT):

$$x\,[n] = \frac{1}{N}\sum_{k=0}^{N-1} X\,[k]\, e^{jnk\frac{2\pi}{N}}, \quad n = \{0, 1, 2, \ldots, N-1\}. \tag{14.6}$$

Importantly for implementation, the DFT and IDFT look very similar; very similar algorithms can be used for both transforms. Computers can evaluate both of them *exactly* and *without any approximation*. The length N can be as large as we wish.

Example 14.1: DFT of a Sinusoid

Calculate the DFT of a sine wave that is evenly sampled $N = 6$ times over one period, as shown in Fig. 14.1(a). Plot the magnitude and phase spectra. The time-domain signal is

$$x[n] = \left\{0, \frac{\sqrt{3}}{2}, \frac{\sqrt{3}}{2}, 0, -\frac{\sqrt{3}}{2}, -\frac{\sqrt{3}}{2}\right\}.$$

The DFT is then

$$\begin{aligned} X[k] &= \sum_{n=0}^{5} x[n]\, e^{-jnk\frac{2\pi}{N}} \\ &= 0 + \frac{\sqrt{3}}{2}e^{-j\frac{k\pi}{3}} + \frac{\sqrt{3}}{2}e^{-j\frac{2k\pi}{3}} + 0 - \frac{\sqrt{3}}{2}e^{-j\frac{4k\pi}{3}} - \frac{\sqrt{3}}{2}e^{-j\frac{5k\pi}{3}}, \end{aligned}$$

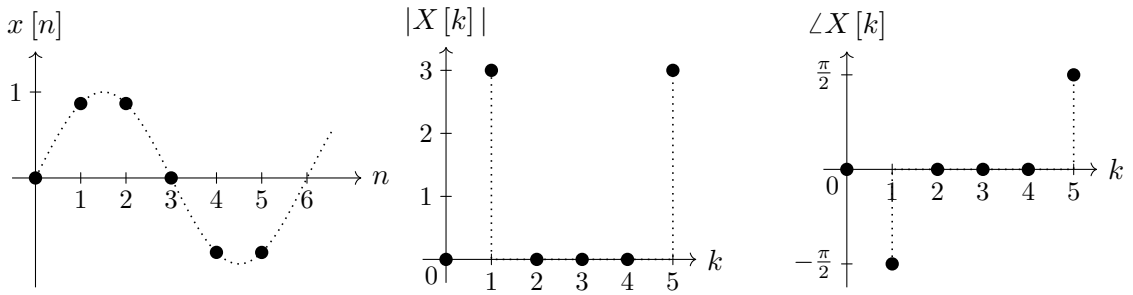
which we solve for each value of k. We find that

$$\begin{aligned} X[0] &= 0 \\ X[1] &= \frac{\sqrt{3}}{2}\left[e^{-j\frac{\pi}{3}} + e^{-j\frac{2\pi}{3}} - e^{-j\frac{4\pi}{3}} - e^{-j\frac{5\pi}{3}}\right] \\ &= \frac{\sqrt{3}}{2}\left[\cos\left(\frac{\pi}{3}\right) - j\sin\left(\frac{\pi}{3}\right) + \cos\left(\frac{2\pi}{3}\right) - j\sin\left(\frac{2\pi}{3}\right)\right. \\ &\quad \left. - \cos\left(\frac{4\pi}{3}\right) + j\sin\left(\frac{4\pi}{3}\right) - \cos\left(\frac{5\pi}{3}\right) + j\sin\left(\frac{5\pi}{3}\right)\right] \\ &= \frac{\sqrt{3}}{2}\left[\frac{1}{2} - j\frac{\sqrt{3}}{2} - \frac{1}{2} - j\frac{\sqrt{3}}{2} + \frac{1}{2} - j\frac{\sqrt{3}}{2} - \frac{1}{2} - j\frac{\sqrt{3}}{2}\right] \\ &= -j3 \\ X[2] &= 0, \end{aligned}$$

and so on, and we find that

$$\begin{aligned} X[k] &= \{0, -j3, 0, 0, 0, j3\} \\ |X[k]| &= \{0, 3, 0, 0, 0, 3\} \\ \angle X[k] &= \left\{0, -\frac{\pi}{2}, 0, 0, 0, \frac{\pi}{2}\right\}. \end{aligned}$$

We plot the magnitude and phase spectra in Figs. 14.1(b) and (c). We note these both have periodic extensions of length N; the 3 at $k = 5$ is the same as being at $k = -1$. Also, for sampling frequency f_s, the separation between samples in frequency is $f_s/N = f_s/6$.



(a) Time-domain signal. (b) Magnitude spectra. (c) Phase spectra.

Figure 14.1: Plots for Example 14.1.

As we have noted, the DFT provides a sampled view of the DTFT. It is referred to as a picket fence, since we only see the DTFT at N frequencies. If we want to see more detail in the DTFT, then we can artificially increase the length of the time-domain signal $x[n]$ by adding zeros to the end. This is known as **zero padding**. We will see the effect of zero padding in an example.

Example 14.2: Effect of Zero Padding

Find the DFT $X[k]$ of the signal $x[n] = \{3,2,3\}$ as shown in Fig. 14.2(a). Compare the DFT *magnitude* with 1) the corresponding DTFT $X\left(e^{j\Omega}\right)$; and 2) the DFT when we pad $x[n]$ to be length $N = 6$.

Since $N = 3$ know that the frequencies are separated by $\Omega_0 = 2\pi/3$. The DFT is then

$$X[k] = \sum_{n=0}^{2} x[n]\, e^{-jnk\frac{2\pi}{3}} = 3 + 2e^{-jk\frac{2\pi}{3}} + 3e^{-jk\frac{4\pi}{3}},$$

which we need to solve for $k = \{0, 1, 2\}$. We find that

$$\begin{aligned} X[0] &= 3 + 2 + 3 = 8 \\ X[1] &= 3 + (-1 - j\sqrt{3}) + \left(-\frac{3}{2} + j\frac{3\sqrt{3}}{2}\right) = 0.5 + j0.866 \\ X[2] &= 0.5 - j0.866 \end{aligned}$$

We plot the magnitude of $X[k]$ in Fig. 14.2(b) using a radial frequency scale (where each DFT point is at frequency $k\Omega_0$).

Since $x[n]$ has a finite duration, there is a corresponding DTFT:

$$\begin{aligned} X\left(e^{j\Omega}\right) &= \sum_{n=0}^{2} x[n]\, e^{-jn\Omega} \\ &= 3 + 2e^{-j\Omega} + 3e^{-2j\Omega} \\ &= e^{-j\Omega}\left(2 + 3e^{j\Omega} + 3e^{-j\Omega}\right) = e^{-j\Omega}\left(2 + 6\cos(\Omega)\right). \end{aligned}$$

The DTFT is defined for all Ω but we plot its magnitude in Fig. 14.2(b) over $[0, 2\pi)$. We see that the DFT is sampling the DTFT, as expected, though it doesn't capture the shape of the DTFT particularly well.

Now consider zero-padding $x[n]$ (as in Fig. 14.2(c)) so that $N = 6$ and re-calculating the DFT. The frequency separation is now $\Omega_0 = \pi/3$ and the DFT is then

$$X_{\text{pad}}[k] = \sum_{n=0}^{5} x[n]\, e^{-jnk\frac{2\pi}{6}} = 3 + 2e^{-jk\frac{2\pi}{6}} + 3e^{-jk\frac{4\pi}{6}},$$

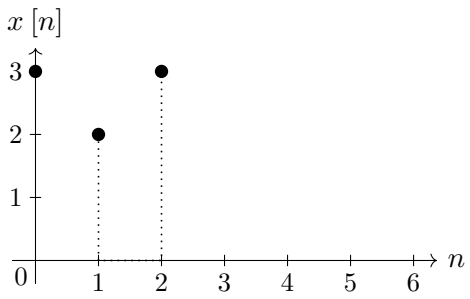
and we can evaluate the DFT as

$$X_{\text{pad}}[k] = \left\{8, 5e^{-j\frac{\pi}{3}}, e^{j\frac{\pi}{3}}, 4, e^{-j\frac{\pi}{3}}, 5e^{j\frac{\pi}{3}}\right\},$$

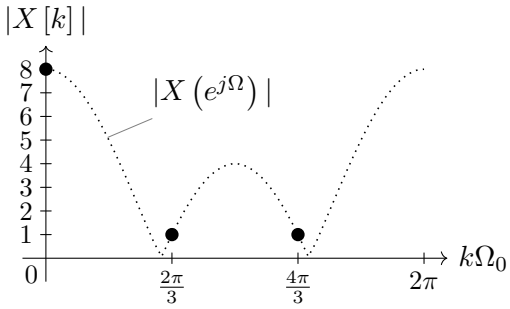
and we plot the magnitude of $X_{\text{pad}}[k]$ in Fig. 14.2(d) with the same corresponding DTFT. By increasing N the DFT has a better representation of the DTFT.

EXTRA 14.1: DFT Properties

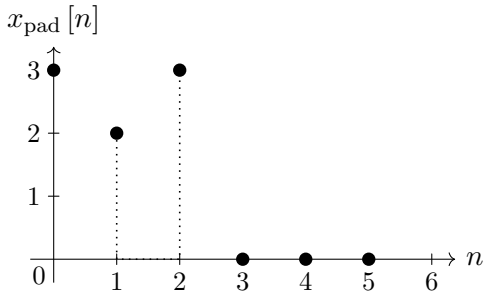
We will not go over properties of the DFT in detail, but we note the DFT



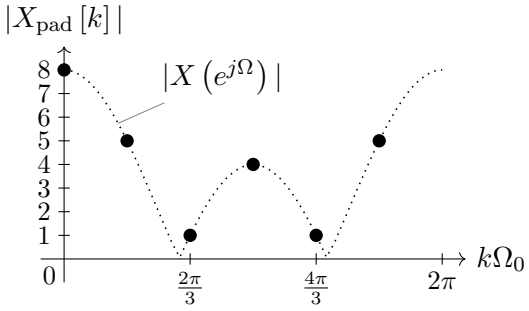
(a) Time-domain signal.



(b) Magnitude spectra.



(c) Zero-padded time-domain signal.



(d) Zero-padded magnitude spectra.

Figure 14.2: Plots for Example 14.2.

has some properties that are similar to others that we have seen throughout the module, in addition to properties that derive from its periodicity such as circular convolution and circular correlation. More details are in Section 9.3 of Lathi and Green.

14.3 Fast Fourier Transform

As N increases, it quickly becomes cumbersome to evaluate the DFT and inverse DFT manually by hand. The computational complexity is $\mathcal{O}(N^2)$, so even in a computer implementation this is inefficient with increasing N. The key detail is that there are algorithms that simplify the complexity of the DFT. The **fast Fourier transform** (FFT) is a family of algorithms that do this and they only have a complexity of $\mathcal{O}(N \log_2 N)$. The complexity gain is about an order of magnitude for sequences with just 16 samples and two orders of magnitude with sequences of 1000 samples. This gain is achieved with *no approximations*, which has made the FFT extremely popular in practice.

The precise details of the FFT are beyond the scope of this module, as they

are still cumbersome to evaluate by hand. However, importantly, the FFT and its inverse are readily available in MATLAB (and other computing platforms). The syntax of the `fft` and `ifft` functions in MATLAB is quite simple:

```
% Given discrete-time sequence x (already zero-padded if desired)
X = fft(x); % FFT

x2 = ifft(X); % Inverse FFT; should have x2 == x
```

While we give a more extensive presentation of computing with digital signals in Lesson 15, here we repeat Example 14.1 with a larger N where we sample the sinusoid more frequently (i.e., not zero padding).

Example 14.3: FFT of a Sinusoid

Calculate the DFT of a sine wave that is evenly sampled $N = 64$ times over one period, as shown in Fig. 14.3. Plot the magnitude spectra.

The sequence length is too long to calculate by hand, so will use the FFT in MATLAB. The following code can do this:

```
N=64; % Sequence length
nVec = 0:(N-1); % Vector of index values
x = sin(2*pi*nVec/N); % Sample N times over one period of 2*pi

X = fft(x); % FFT

XMag = abs(X); % Magnitude
figure; stem(nVec, XMag);
xlabel('Spectral Index k')
ylabel('|X[k]|')
```

We plot the magnitude of the frequency response in Fig. 14.4. As in Example 14.1, the separation between the samples in frequency is $f_s/64$, so the frequency of the observed sinusoid is $f_s/64$ (which is consistent with the time-domain signal, as we observe one period over 64 samples).

EXTRA 14.2: Double-Sided vs Single-Sided FFT

We should note that many of the examples of the FFT in the MATLAB

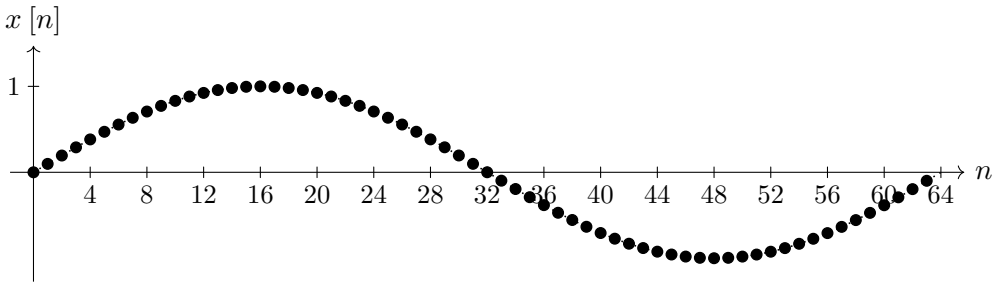


Figure 14.3: Time-domain signal for Example 14.3.

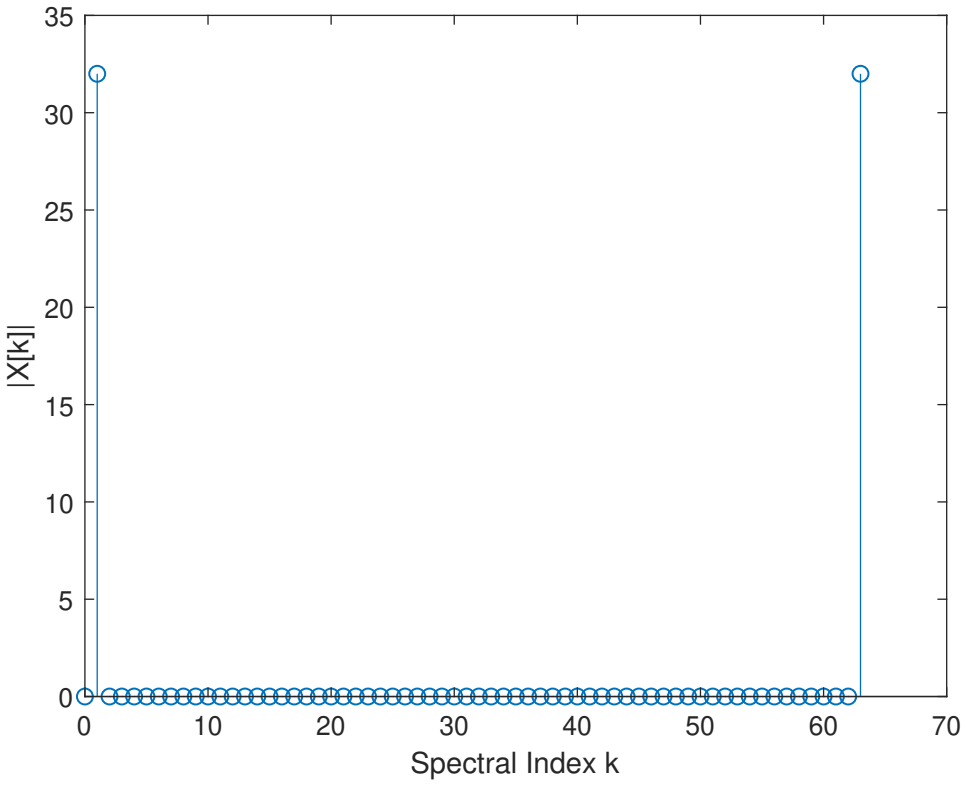


Figure 14.4: Magnitude spectrum for Example 14.3.

help documentation refer to and use the "single-sided" FFT, which can be obtained from the "double-sided" FFT. The FFT as we have introduced it here, and as calculated by the `fft` function, is the "double-sided" FFT. For simplicity, we will always mean the "double-sided" FFT in this module.

14.4 Summary

- Discrete spectra are more convenient than continuous spectra for computations.
- The **discrete Fourier transform** (DFT) provides samples of the DTFT to show discrete spectra of a fixed-length signal.
- The **fast Fourier transform** (FFT) is a highly efficient implementation of the DFT that is widely used in practice.

14.5 Further Reading

- Chapter 9 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014.
- Section 15.2 of "Essential MATLAB for engineers and scientists," B. Hahn and D. Valentine, Academic Press, 7th Edition, 2019.

14.6 End-of-Lesson Problems

1. Find the DFT $X[k]$ of the signal $x[n] = \{2,0,2\}$. Compare the DFT *magnitude* with 1) the corresponding DTFT $X\left(e^{j\Omega}\right)$; and 2) the DFT when we pad $x[n]$ to be length $N = 8$. Try verifying your manual calculations using the `fft` function.
2. Demonstrate the linearity property of the FFT by plotting the spectra of a signal composed of two sinusoids of different frequencies. Let $N = 64$ and let the frequencies of the two sinusoids be $f_s/16$ and $f_s/4$.

Lesson 15

Computing with Digital Signals

This lesson is the digital counterpart to Lesson 7, where we saw how MATLAB can be used as a computing tool for analogue systems. It is even more "natural" to represent digital signals and systems in a computer (in fact this is by design). While we have already covered some cases of digital signal processing on a computer, e.g., when computing the FFT in Lesson 14, this lesson provides a more general overview for working with digital signals and systems. In particular, we will cover tasks that we've already done "by hand" in the previous lessons of this part of the module, so this should facilitate revision of previous learning concepts.

We once again re-iterate that MATLAB is not the only tool available for these purposes; practically any programming language has (or should have) libraries available for common digital signal processing tasks.

15.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** how digital signals and systems are represented in computing systems.
2. **Analyse** digital signals and systems with a computing package.
3. **Design** and **Implement** digital filters with a computing package.

15.2 Digital Signals in MATLAB

Computers are inherently digital, and are designed for storing information in discrete containers such as arrays and matrices. This makes them well-suited for discrete-time signals and systems. It is intuitive to think of a one-dimensional discrete-time

signal (e.g., voltage level) being stored in a one-dimensional array, where each element in the array corresponds to one discrete-time index of the signal. This can also be extended to multi-dimensional signals being stored in multi-dimensional arrays, e.g., as we will see with images in the Image Processing part of the module. MATLAB stands for "MATrix LABoratory," in case the relevance wasn't sufficiently apparent, though really any modern program language or general computing platform should have functionality to work with and manipulate arrays of data.

Throughout this module we have referred to "digital" signals even though we have treated them as continuous in amplitude. This has facilitated our mathematical analysis and has minimal impact on implementation, especially given the precision of floating point numbers (see more on this in Lesson 7).

We now discuss some common ways to represent digital systems and signals in MATLAB, in particular those that are most relevant to the material covered throughout this part of the module.

15.2.1 Digital Signals in Arrays

This should be obvious if you have previously done any programming (in any language), but arrays can be direct implementations of digital signals. However, for completeness, we provide the MATLAB syntax here and list some useful functions for creating arrays with certain properties. For example, we can create an array directly from its elements:

```
x1 = [1, 0, 1, 1, 0, 1, 0, 0]; % Array of length 8
x2 = 1:10; % Array of integers from 1 to 10
x3 = 4:2:20; % Array of even numbers from 4 to 20
x4 = 0:-0.5:-5; % array of values from 0 to -5, decreasing by -0.5
```

2D arrays can also be readily assembled, using a semi-colon between rows:

```
x5 = [2:6; 9:13]; % 2 rows, 5 columns
```

There are also function for creating arrays with pre-determined values. These can be in any number of dimensions by supplying additional arguments.

```
x6 = ones(1,10); % Create array of ones with 1 row and 10 columns
x7 = ones(2,5,3); % Create a 2x5x3 array of ones
x8 = zeros(3); % Create a 3x3 matrix of zeros (or use zeros(3,3))
```

We have noted that convolution is easier in discrete-time than in continuous time. We can evaluate it directly in MATLAB using the `conv` function:

```
x = ones (1,100); % Input
```

```
h = [1,2,-1,1,2,0]; % System impulse response
y = conv(x,h); % System output
```

We note that the length of the output from `conv` is one less than the sum of the length of the inputs (i.e., the length of the full convolution vector).

Signals can also be readily converted to the spectral domain using the FFT implementation of the DFT. We have already seen examples of this in Lesson 14.

15.2.2 Digital Systems in the Symbolic Math Toolbox

EXTRA 15.1: Digital Systems in the Symbolic Math Toolbox

We discussed in Lesson 7 how analogue systems can be represented *symbolically*, where can define arbitrary variables and use them in functions. While this was particularly helpful for analogue systems since we cannot precisely represent continuous-time on a computer, symbolic math can also be used for digital systems, *though it is not recommended - hence why this is EXTRA content.* We can begin by creating symbolic variables for n and z:

```
syms n z
```

If we create functions using these variables, we can convert between the time and z domains using the `ztrans` and `iztrans` functions. We can use the same pre-built functions that apply to continuous-time systems (e.g., rectangularPulse, dirac, heaviside). Here we repeat Example 7.1 for an equivalent digital system.

Example 15.1: Finding Expression for System Output

Find the output of a simple low pass filter with transfer function

$$H(z) = \frac{z+1}{z},$$

when the input is a cosine $x(t) = \cos(\Omega n)$. Using our knowledge of LSI systems, we can find the answer in a short function using symbolic math.

```
function y = lowpassOutputDigital(omega)
% Find output to my low pass filter
% Create symbolic variables
syms n z
% Create symbolic functions for input and system
x = cos(omega*n);
H = (z+1)/z;

X = ztrans(x); Y = H*X; y = iztrans(Y);
```

If you call this function with frequency $\Omega = 5$, then you will find that the output is different from that which we saw in Example 7.1.

Some additional useful functions when using the Symbolic Math Toolbox:

- `subs` – substitute a numeric value in place of a variable in a symbolic function.
- `double` – convert a symbolic value into a double-precision number that can then be processed by non-symbolic numeric functions.

15.2.3 Digital Systems in the Signal Processing Toolbox

We described the `freqs` function for plotting the frequency response of analogue systems. Similarly, the `freqz` function is used for plotting the frequency response of digital systems. The syntax is quite similar; the first two input arguments for this function are two arrays, one describing the coefficients of the numerator polynomial and one describing the coefficients of the denominator polynomial, both with negative powers of z. The standard naming convention for these arrays is consistent with the form that we used in Lesson 10; the `b` vector lists the numerator coefficients

in decreasing order and the `a` vector lists the denominator coefficients in decreasing order (and normally with 1 as the first term in a). We do emphasise that MATLAB numbers indexing from index "1" and not index "0." For the system in Example 15.1 and converting the transfer function to have negative powers of z, we have `b=[1,1]` and `a=[1]`. We can then plot the frequency response of this filter:

```
b = [1,1]; a = 1;
figure; % New figure
w = 0.1:0.1:pi;
freqz(b,a,w)
```

The figure is shown in Fig. 15.1. The argument `w` is a vector of *normalised* (radial) frequencies that can range from 0 to π, but if you make it scalar instead then it defines the number of frequency points are determined automatically. As we noted for `freqs`, `freqz` plots the magnitude and phase of the frequency response on a logarithmic scale (for both the frequencies and for the magnitude). Also, if you assign `freqz` to an output argument, then it will store the complex frequency response values for each of the frequencies. If you append a semi-colon `;` after the function then it will omit the plot.

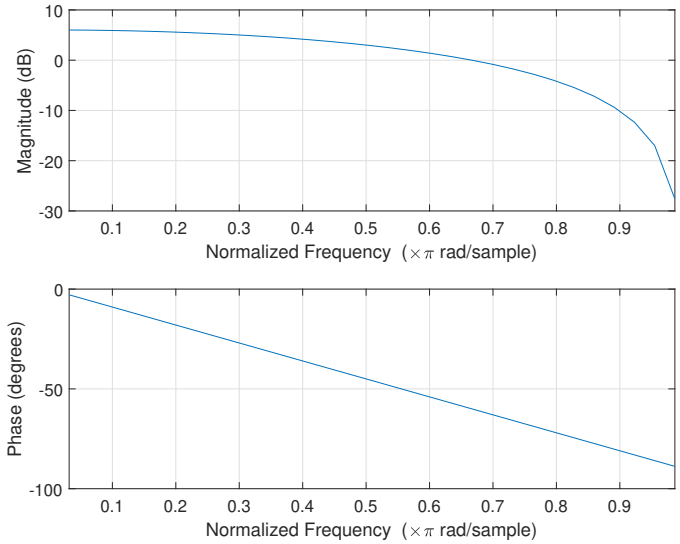


Figure 15.1: Frequency response of the system in Example 15.1.

Correspondingly, we can use the `impz` function to return the digital system impulse response using the transfer function coefficients in the same format:

```
b = [1,1]; a = 1;
h = impz(b,a);
```

15.2.4 Digital Systems in the Control Systems Toolbox

We have seen that the Control Systems Toolbox has the function `pzplot` to generate a pole zero plot from an LTI transfer function. This also works for LSI systems since the way of identifying roots is the same.

```
b = [1,1]; a = [1];
figure; % New figure
H = tf(b,a);
pzplot(H)
```

15.3 Digital Filter Design

In Lesson 13 we focused on the design of digital FIR filters. MATLAB's Signal Processing Toolbox provides tools for both FIR and IIR filter design, and we will discuss both of them here. To get an overview of all of the approaches available, as well as functions for analysing filters, go to the Signal Processing Toolbox in the MATLAB help documentation and visit the section on "Digital and Analog Filters" and then "Digital Filter Design."

One general and powerful function provided for either FIR or IIR filter design is `designfilt`. There are many options to define the filter specification and it will return a structure that includes the corresponding filter coefficients. A corresponding GUI is available by calling `filterDesigner`; it is **highly recommended** that you check this out and explore the options available. However, it contains many more features that what we need for this module, so we will not discuss it here in detail.

15.3.1 IIR Filters

The IIR filter families correspond to those that we have already discussed for analogue filters and they have the following functions for determining transfer functions:

- Butterworth: `butter`
- Chebyshev: `cheby1` (i.e., type 1)
- Inverse Chebyshev: `cheby2` (i.e., type 2)
- Elliptic: `ellip`
- Bessel (i.e., Bessel-Thomson): `besself`

Since the syntax and use of these functions is consistent, we focus discussion on the `butter` function. The default input arguments are the order of the filter and the cut-off frequency. By default the filter will be low pass, but a third optional argument can indicate whether the filter is to be a different type, i.e., one of `'low'`, `'high'`, `'bandpass'`, or `'stop'`. For the output, we will consider one of 2 formats. If you define two input arguments then they will be of the form `[b,a]`, i.e., the coefficients of the transfer function's numerator and denominator polynomials, respectively. If you define three input arguments then they will be of the form `[z,p,k]`, i.e., you will get the poles, zeros, and the transfer function gain K. From our knowledge of transfer functions, either of these two forms are sufficient to fully define the transfer function. You can also convert between the two forms using the `zp2tf` or `tf2zp` functions.

15.3.2 FIR Filters

In Lesson 13 we focused on *window-based* design of FIR filters. This is one of many approaches available in the Signal Processing Toolbox and can be achieved directly using the `fir1` function. The first argument is the order of the filter, the second argument is the vector (or scalar) of cut-off frequency constraints (normalised from 0 to 1), the third is the type of filter (default is `'low'`), the fourth is the type of window (default is `hamming`; length must be one more than the order), and the fifth argument is a switch to normalise the coefficients for gain 1 (0dB) at the centre of the pass band (default is to normalise). The output is the vector of filter coefficients; it is only one vector because the filter is FIR (i.e., all poles are at the origin). We see use of this filter in the following example.

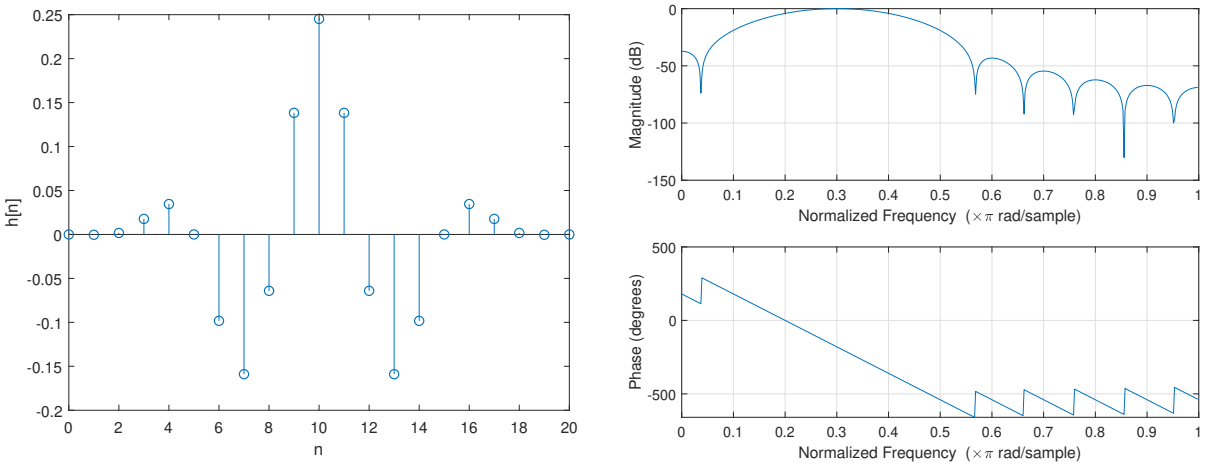
Example 15.2: FIR Filter Design in MATLAB

Design a 20th order band pass FIR filter with a Hann window that processes signals sampled at 1 kHz and passes signals between 100 Hz and 200 Hz.

If $f_s = 1$ kHz then the filter is designed for signals with frequencies between 0 and 500 Hz. Thus, the normalised cut-off frequencies are 0.2 and 0.4.

```
b = fir1(20, [0.2,0.4], 'band', hann(21)); % Design FIR filter
freqz(b,1)     % Plot filter's frequency response
stem(0:20,b) % Plot FIR filter coefficients
xlabel('n') % Label stem plot
ylabel('h[n]') % Label stem plot
```

We plot the impulse response and frequency response in Fig. 15.2.



(a) Impulse response.

(b) Frequency response.

Figure 15.2: Filter in Example 15.2.

15.3.3 Filter Outputs

Once we have designed a filter, we want to be able to put it to use, i.e., pass inputs to and observe the output. We have already discussed the use of the convolution function `conv`, which we can use to find an FIR filter output given its impulse response and an input signal. We briefly discuss a few alternatives and their uses:

- `filter` – similar to `conv` but takes transfer function coefficients (or filter design returned by `designfilt`) instead of the impulse response, which means that it can be used for FIR or IIR filters. Also, the output length is the same as that of the input.

```
y = filter(b,a,x); % Default call with coefficient vectors a and
    b. If FIR, a = 1
y = filter(d,x); % Call with filter "d" returned by call to
    designfilt
```

- `filtfilt` – similar to `filter` but it actually filters *twice* (once forward and once backward) so that there is *zero phase distortion*. As a result, the function squares the magnitude (and doubles the order) of the original filter defined by the transfer function.

```
y = filtfilt(b,a,x); % Default call with coefficient vectors a
    and b. If FIR, a = 1
```

```
y = filtfilt(d,x); % Call with filter "d" returned by call to
    designfilt
```

15.4 Summary

- Digital signals and systems are easy to represent directly in digital computers.
- Approaches for analytical representations, such as symbolic math and vectors for transfer function coefficients, apply to digital systems as well as analogue systems.
- MATLAB has very powerful tools for FIR and IIR filter design and analysis. We can use these tools to implement many of the tasks that we have completed "by hand" throughout this part of the notes, or to design systems that are more complex than what we can readily do on paper.

15.5 Further Reading

- The MATLAB documentation is an excellent resource, whether online or within the MATLAB help browser.
- Chapter 17 of "Essential MATLAB for engineers and scientists," B. Hahn and D. Valentine, Academic Press, 7th Edition, 2019.

15.6 End-of-Lesson Problems

1. Consider the response to a moving-average filter `h = [.2 .2 .2 .2 .2]` when the input is the discrete-time sinusoid `x = cos(pi*n/4)` for $0 \leq n \leq 39$. Find the output using

   (a) `conv`

   (b) `ztrans` and `iztrans` (`subs` and `double` will also help)

   (c) `filter`

   Plot the outputs (using `stem` or some comparable plotting function) and comment on differences.

2. Verify the results of Question 12.1, where you were asked to design an FIR low pass filter with cut-off frequency $\Omega_c = \pi/4$ using $L = 5$ coefficients with

a rectangular window, and compare the time-domain and magnitude spectra with that of a Hamming window. Find the impulse responses and spectra using MATLAB.

Lesson 16

Digital Vs Analogue Recap

We now end the Digital part of the module in the way we started by re-visiting analogue systems. However, we are not discussing signal conversion as we did in Lesson 8. Instead, in this lesson we recap and highlight key similarities and differences between analogue and digital systems, signals, and their analysis. One of our goals is to avoid confusion over which tools and transforms apply to which class of systems. We also draw attention to the conversion of designs developed for one class to be translated to the other, i.e., using the bilinear transform.

16.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** which transforms apply to which types of signals.

2. **Understand** the bi-linear transform to map between the analogue and digital domains.

16.2 Comparison of Transforms

We re-iterate that our primary distinction (for mathematical purposes) between analogue and digital systems is that analogue systems are continuous-time and digital systems are discrete-time. Thus, we can highlight differences between the transforms that correspond to continuous-time and discrete-time systems.

We summarize the transforms in the following (and as shown in Fig. 16.1). We refer to $f(t)$ and $f[n]$ as signals in the generic sense (i.e., they are functions that could be a system input, output, or impulse response):

- For an aperiodic (or *simple* periodic) continuous-time signal $f(t)$, we can convert to the Laplace domain (or s-domain) via the Laplace transform, which for $s = j\omega$ is the (continuous) Fourier transform. The Fourier transform of the signal is its frequency response $F(j\omega)$, and is generally defined for all ω. The Laplace and Fourier transforms also have corresponding inverse transforms to convert $F(s)$ or $F(j\omega)$ back to $f(t)$.

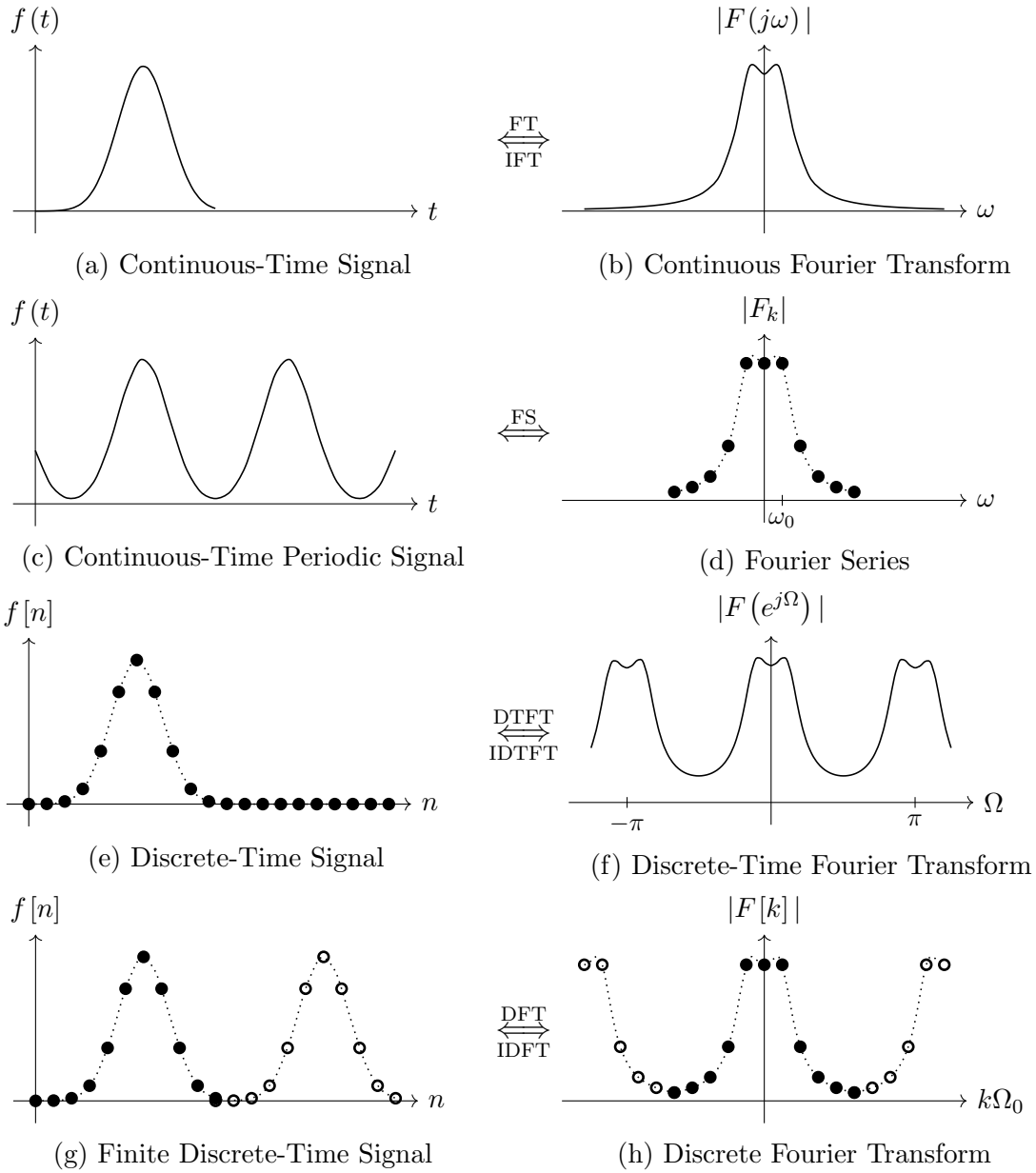
- For a more complex periodic continuous-time signal $f(t)$, the Fourier series representation decomposes the signal into its frequency components F_k at multiples of the fundamental frequency ω_0. This can be interpreted as samples of the frequency response $F(j\omega)$, which then corresponds to periodicity of $f(t)$ over time. The coefficients F_k are found over one period of $f(t)$.

- For discrete-time signal $f[n]$, we can convert to the z-domain via the Z-transform, which for $z = e^{j\Omega}$ is the discrete-time Fourier transform. The discrete-time Fourier transform of the signal is its frequency response $F\left(e^{j\Omega}\right)$ and repeats every 2π (i.e., sampling in time corresponds to periodicity in frequency). There are corresponding inverse transforms to convert $F(z)$ or $F\left(e^{j\Omega}\right)$ back to $f[n]$.

- For discrete-time signal $f[n]$ with finite length (or truncated to) N, we can convert to the frequency domain using the discrete Fourier transform, which is also discrete in frequency. The discrete Fourier transform also has N points distributed over 2π and is otherwise periodic. Here, we see sampling in both frequency and time, corresponding to periodicity in the other domain (that we usually ignore in analysis and design because we define both the time domain signal $f[n]$ and frequency domain signal $F[k]$ over one period of length N).

With familiarity with the above, one could be given an arbitrary $f(t)$, $f[n]$, or some transform F, and in principle we should know which transforms would apply. In practice, you are **not** expected to memorise these differences. The emphasis here is to present the transforms together so that we can see how they are qualitatively related and distinguish between the circumstances where they can be applied.

One detail that *is* important to memorise is the nature of frequency when used for continuous-time versus discrete-time systems. As noted above, *any* frequency ω can be observed in a continuous-time function. The frequency range Ω in a discrete-time function is only 2π, i.e., one trip around the unit circle, and the range of frequencies covered by this circle varies according to the sampling frequency (if we have frequencies beyond the range $-\pi < \Omega \leq \pi$, then they are *aliased*). This difference is evident when we consider system stability, as shown by the comparison of stability regions in Fig. 16.2. For a system to be stable, its poles in the s-domain



(a) Continuous-Time Signal

(b) Continuous Fourier Transform

(c) Continuous-Time Periodic Signal

(d) Fourier Series

(e) Discrete-Time Signal

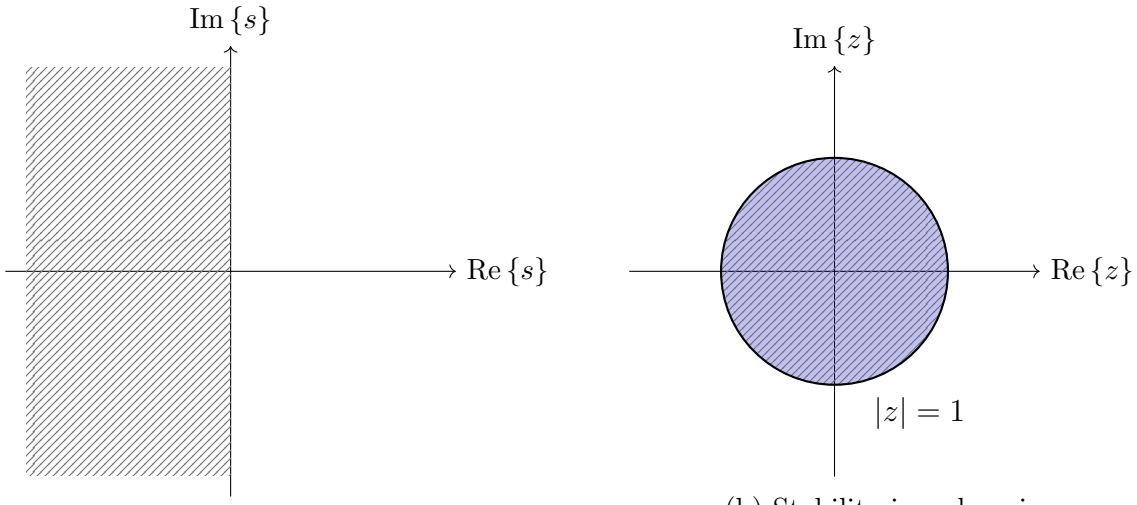
(f) Discrete-Time Fourier Transform

(g) Finite Discrete-Time Signal

(h) Discrete Fourier Transform

Figure 16.1: All frequency-based transforms that we have seen in this module.

must have negative real components, but poles in the z-domain must lie inside the unit circle.



(a) Stability in s-domain

(b) Stability in z-domain

Figure 16.2: Stability comparison of analogue and digital systems.

16.3 Bilinear Transform

Except for signal conversion in Lesson 8, we have kept our discussion of analogue and digital systems relatively isolated, although the narratives progressed similarly (i.e., linear transforms, stability, frequency response, filtering, periodicity). However, analogue and digital systems are not always clearly isolated from each other in design. In fact, it is *very* common to take a design for one domain and translate it to the other. Our final topic is a (beyond scope) discussion of the bilinear transform, which is used for this purpose.

EXTRA 16.1: Bilinear Transform

Let's take a closer look to compare the Laplace and Z-transforms. We see that there is a similar role being played by e^{-st} and z^{-k}. If we were to say that we have a sampling period of T_s, then we could sample the complex exponential at times $t = kT_\text{s}$. By setting $e^{-st} = z^{-k}$, we then have $z = e^{sT_\text{s}}$. We can use this to do an approximate mapping between the s-domain and z-domain. If we apply a Taylor series approximation ($e^a \approx 1 + a$), then we

can write the following:

$$z = e^{sT_s} = \frac{e^{sT_s/2}}{e^{-sT_s/2}} \approx \frac{1 + sT_s/2}{1 - sT_s/2},$$

which we can also re-arrange to write

$$s \approx \frac{2}{T_s}\frac{z-1}{z+1}.$$

This conversion is known as the **bilinear transform**, and it enables us to take a filter that was designed in the s-domain and map it to the z-domain. Doing so preserves stability and maps low frequency features very well. We will not apply the transform directly in this module, but you may see it elsewhere during your course.

16.4 Summary

- The transforms that we have applied throughout this module are generally related to each other, but primarily differ in whether they apply to functions that are continuous or discrete in frequency or time.
- Sampling in frequency or time corresponds to periodicity in the other domain.
- The **bilinear transform** enables conversion of analogue systems into digital systems (or vice versa).

16.5 Further Reading

Most material in this lesson is a recollection of material from previous lessons.

- Sections 8.1.2-8.1.4 of "Essentials of Digital Signal Processing," B.P. Lathi and R.A. Green, Cambridge University Press, 2014. These sections cover use of the bilinear transform to translate analogue filter family designs to IIR digital filters.

16.6 End-of-Lesson Problems

As this is a summary lesson, there are no specific end-of-lesson problems. If you have a particular interest in using the bilinear transform for filter design (which is beyond the scope of this module), you will find examples of doing so in Sections 8.1.2-8.1.4 of Lathi and Green.

Part III

Random Signal Processing

Lesson 17

Probabilities and Random Signals

The third part of this module is a *brief* study of **Random Signal Processing**. Most practical signals contain an unknown component that could be undesirable (e.g., thermal noise impairs all sensors) or intentional (e.g., communication signals have embedded information that we need to learn). In this lesson, we provide a brief overview of random probability distributions and some of the roles that random variables can play in practical signal processing applications. Subsequent lessons consider the estimation of signals in the presence of noise and measuring additional characteristics of random signals.

17.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** how random variable distributions are defined.
2. **Understand** common random variable distributions.
3. **Evaluate** common properties of random variables.
4. **Understand** the role of random variables in practical signal processing applications.

17.2 Random Variables

A **random variable** is a quantity that takes *non-deterministic* values, i.e., we do not know what the value will be in advance. Instead, there is a **probability distribution** (which may also be unknown) that defines the probability that the random variable will take some value. The value could be voltage, time, temperature,

or practically any measurable quantity. Random variables can also be discrete (e.g., result of a coin toss) or continuous (e.g., impairments in a wireless communication signal).

Mathematically, continuous random variables are described by their **probability density function** (PDF), whereas discrete random variables are described by their **probability mass function** (PMF). The notation for both types of distribution is very similar. For random variable X that can take values between $x_{\min}$ and $x_{\max}$ (which could be $\pm\infty$), $p(x)$ is the probability that $X = x$. If X is discrete, then the summation of these probabilities over all x is equal to 1, i.e.,

$$\sum_{x=x_{\min}}^{x_{\max}} p(x) = 1, \tag{17.1}$$

and if X is continuous, then the integration of these probabilities over all x is equal to 1, i.e.,

$$\int_{x=x_{\min}}^{x_{\max}} p(x)\, dx = 1. \tag{17.2}$$

We can take the PMF summation or PDF integral over a *subset* of the domain of X to calculate the probability of X being within that subset.

An important feature of a random variable is its **moments**. Roughly speaking, they summarise the values that a random variable can take. The general formula for the nth order moment of X $E[X^n]$ is

$$E[X^n] = \sum_{x=x_{\min}}^{x_{\max}} x^n p(x) \qquad E[X^n] = \int_{x=x_{\min}}^{x_{\max}} x^n p(x)\, dx, \tag{17.3}$$

for discrete and continuous X, respectively.

When $n = 1$, the moment is called the **mean** value μ_X, and it is the expected (average) value of the random variable. When $n = 2$, the moment is called the **mean-squared** value, and it describes the spread of the random variable. For second-order moments, we more commonly refer to the **variance** σ_X^2, which is the mean-squared value with a correction for the mean, i.e.,

$$\sigma_X^2 = E\left[(X - \mu_X)^2\right] = E\left[X^2\right] - (E[X])^2, \tag{17.4}$$

so it can be directly calculated from the first and second order moments. The **standard deviation** is the square root of the variance.

Many distributions are defined in terms of their moments, as we will see in the following.

17.2.1 Random Variable Distributions

There are a number of well-known PDFs and PMFs that are commonly found in engineering. We will discuss several of them here. Generally, the name of a distribution is also the name of a random variable that is drawn from that distribution (e.g., a uniform random variable is drawn from a uniform distribution).

The **uniform distribution** has an equal probability for a random variable to take any value in its domain, i.e., over $x_{\min} \leq x \leq x_{\max}$. The PDF of the continuous version of a uniform random variable is shown in Fig. 17.1. There are also many practical examples of discrete uniform distributions (e.g., the result of a dice roll, coin toss, or roulette wheel). Intuitively, the mean is simply the average of the limit values, i.e., $\mu_X = \frac{x_{\min}+x_{\max}}{2}$.

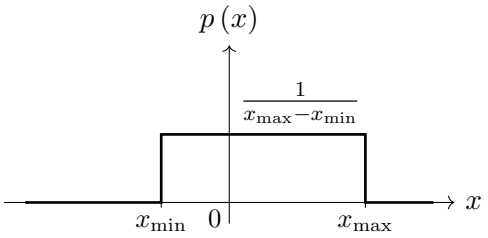


Figure 17.1: Uniform distribution. There is no constraint that $x_{\min} \leq 0 \leq x_{\max}$; there can be any $x_{\min} < x_{\max}$.

A **Bernoulli** random variable has a discrete probability distribution with only two possible values (nominally 1 and 0, but could be yes and no, blue and red, wet and dry, etc.). These two values can have different probabilities, and in general $p(1) = 1 - p(0)$. The mean is simply $\mu_X = p(1)$, and the variance is $\sigma_X^2 = p(1)\,p(0)$.

A **Gaussian** (or **normal**) random variable has a continuous probability distribution over $(-\infty, \infty)$ where values closer to the mean are more likely. Specifically, the PDF is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_X^2}} \exp\left(-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right), \tag{17.5}$$

which we see is an explicit function of the mean μ_X and variance σ_X^2, and we plot the PDF in Fig. 17.2. The Gaussian distribution is arguably the most important continuous distribution because it appears very frequently in practical systems, even for phenomena that are *not* inherently Gaussian distributed. This is thanks to the **Central Limit Theorem** (CLT), which states that a sum of *independent* random variables can be approximated with a Gaussian distribution, and the approximation improves as more random variables are included in the sum. This is true for *any* probability distribution. **Independent random variables** are random variables

that have no dependency on each other (i.e., if knowing the value of one random variable gives you no information to be able to better guess another random variable, then those random variables are independent of each other). While we won't worry about the mathematical details of the CLT, we emphasise the relevance of the Gaussian distribution and we will see it repeatedly in Lesson 18 when modelling estimation problems.

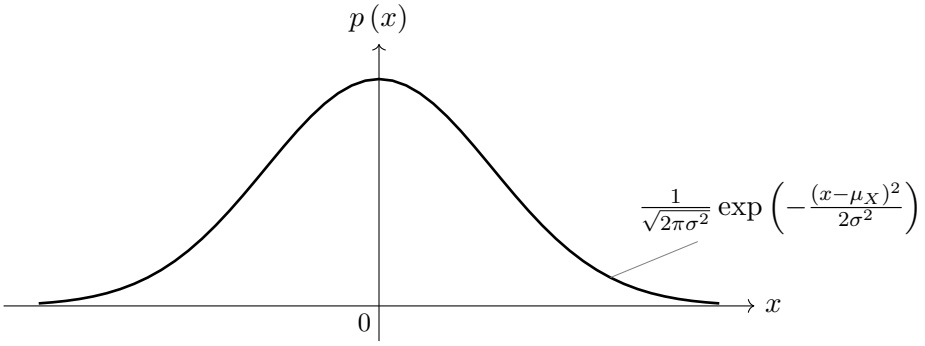


Figure 17.2: Gaussian distribution with mean $\mu_X = 0$.

17.2.2 Empirical Distributions

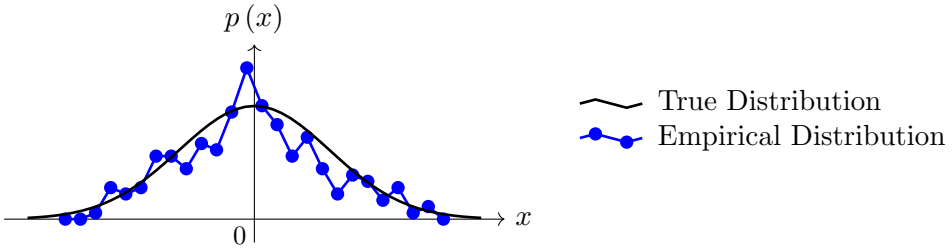


Figure 17.3: Example empirical distribution for a Gaussian random variable.

We have seen PDFs and PMFs that are analytical. Even though they describe non-deterministic quantities, the distributions themselves can be written with an equation. We emphasise that this does *not* mean that a set of samples that are drawn from a distribution will exactly re-create the distribution. To observe behaviour that would match a PDF or PMF, we require an *infinite* number of samples (random variables drawn from the distribution). In practice, for a set of random variables drawn from a distribution, we can make a **histogram** that divides our domain into bins and we count the number of random variables that belong to each bin. If we scale a histogram by the total number of samples, then we have an **empirical**

distribution. We show a sample empirical distribution for a Gaussian random variable in Fig. 17.3 in comparison with the actual distribution.

Example 17.1: Binary Empirical Distribution

Consider flipping a coin 20 times. Even though the probability of heads or tails is 0.5, we may not get exactly 10 heads and 10 tails. Suppose that we our results are 11 heads and 9 tails. We find that the heads and tails events occurred 55 % and 45 % of the time, respectively, and plot the empirical distribution in Fig. 17.4.

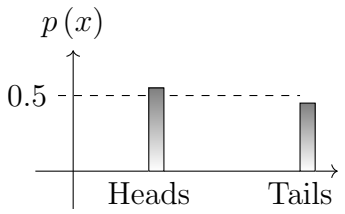


Figure 17.4: Empirical distribution from flipping a coin for Example 17.1.

EXTRA 17.1: Bell Curve Grading

The Gaussian distribution has a somewhat notorious reputation among students everywhere. This is not because of the complexity of the PDF itself in Eq. 17.5, but how it presents itself. Many phenomena behave according to the Gaussian distribution, including student grades. If you plot a histogram of student marks from an assessment, then you will likely see a "bell shape", i.e., as in Fig. 17.2. The key word is *likely*; a PDF is a measure of probabilities. Controversy arises if it is suspected that measured marks are adjusted so that the data set more closely resembles a bell curve.

17.3 Random Signals

Random variables can appear in signals in fundamentally different ways. Here are some examples:

1. *All* electronics exhibit **thermal noise** due to the agitation of electrons. This is often modelled by adding a Gaussian random variable to the signal of interest.

2. Signal processing techniques themselves introduce noise terms. We have already seen the imperfections in aliasing, quantisation, and non-ideal filters. For example, if we quantise a continuous signal, we have uncertainty about the signals true value that can be modelled as a uniform random variable.

3. Random variables can be used to store information, e.g., data can be encoded into bits and delivered across a communication channel. A **receiver** does not know the information in advance and can treat each bit as a Bernoulli random variable that it needs to estimate.

4. Signals can be drastically transformed by the world around them before they reach a transducer. Wireless signals can be obstructed by buildings and trees, chemicals can react, musical instruments can have defects, etc. Effectively, these analogue signals are passing through an *unknown system* $h(t)$ and this system can also vary with time. The variations in the system can be represented using random variables that describe the phenomenon under consideration, e.g., there are particular distributions for the obstruction of wireless signals.

17.4 Summary

- Continuous random variables are described by a **probability density function** (PDF) and discrete random variables are described by a **probability mass function** (PMF).

- Some common random variable distributions are the **uniform**, **Bernoulli**, and **Gaussian** (normal) distributions.

- Randomness has many roles to play in signal processing, including uncertainty in a signal, uncertainty in the system that it passes through, and noise introduced to convert an analogue signal to digital.

17.5 Further Reading

- Chapters 3-5 of "Probability and Statistics for Engineering and the Sciences," J. L. Devore, Brooks/Cole, 9th edition, 2016.

17.6 End-of-Lesson Problems

1. Can a random variable distribution have $x_{\min} = x_{\max}$? Why or why not?
2. Prove using the definition of the mean for a continuous random variable that the mean of a uniformly-distributed random variable is $\mu_X = \frac{x_{\min}+x_{\max}}{2}$.
3. A random variable X is non-negative and has the PDF $p(x) = kxe^{-x}$. Find the following:
   (a) The only possible value of k.
   (b) The probability that $0 \leq X \leq 1$.
   (c) $E[X]$.
   (d) $E[X^2]$ and σ_X^2.
   (e) The most likely value of X.
   (f) The probability that $0 \leq X \leq E[X]$.

Lesson 18

Signal Estimation

In this lesson we provide an introduction to signal estimation problems. These problems encompass a diverse range of signal processing applications, and are also closely related to signal detection. We are covering several specific estimation approaches to gain some broad exposure to the topic. While we will be restricting ourselves to relatively simple techniques and problems, we are building upon material you may have seen in previous modules and we intend to give you some motivation to pursue more complex problems and approaches beyond this module.

18.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** the scope of signal estimation problems and their applications.
2. **Apply** simple signal estimation methods.
3. **Apply** signal estimation methods in a computing platform.

18.2 The Estimation Problem

Signal estimation refers to estimating the values of a parameter (or parameters) that are embedded in a signal. We can interpret this as trying to infer information about a signal. Estimation is a key problem in many signal processing applications. Some examples in different fields are as follows:

- Radar, sonar – estimating the position and trajectory of distant objects (e.g., aircraft, submarines).
- Speech – voice recognition estimating the words that are spoken.

- Image processing – estimating location of an object in an image.
- Biomedicine – estimating fetal heart rate.
- Communications – estimating the transmission channel quality based on the attenuation of a pilot signal.

The above examples all involve estimating parameters from continuous-time signals, however in practice *we perform estimation in discrete-time* to facilitate computation. The general estimation problem is as follows. We have a discrete-time data signal (or sequence) of length N, $y[n]$, and this signal depends on some **unknown parameter** ϕ. To estimate ϕ, we apply some estimating function (or **estimator**) $g(\cdot)$ to calculate $\hat{\phi}$, i.e.,

$$\hat{\phi} = g(y[0], y[1], \ldots, y[N-1]), \tag{18.1}$$

where $\hat{\phi}$ is the estimate of ϕ. We can also generalise Eq. 18.1 to a vector of unknown parameters. The study of signal estimation involves selecting and applying a suitable estimator $g(\cdot)$ while understanding its ability to estimate ϕ in the given situation.

Why are we "estimating" and not just "calculating"? The issue is that in practice the signal $y[n]$ is impaired by noise, so we have to deal with imperfect information (this is also why we are discussing estimation in the context of *random* signal processing). In general, if there were no noise in a signal, then estimation would be trivial and we can just observe (calculate) ϕ directly.

It is important to distinguish estimation from the complementary problem of signal **detection**. Detection is when we do not know whether the signal of interest is currently present from among noise or possibly other candidate signals. Detection is the fundamental problem in applications such as communication, where a receiver ultimately needs to decide which message was transmitted from among possible candidate messages (i.e., transmission symbols).

For the rest of this lesson, we provide a brief survey of some popular and useful estimation methods. We focus on their *implementation and use* rather than a rigorous study of their derivation and performance, as these details can get quite mathematically dense and are well beyond the scope of an undergraduate module. We do note that a significant portion of engineering practice and research is in the development and application of signal estimation (and detection) methods.

EXTRA 18.1: Our Context of Signal Estimation

To be precise about what are we presenting in this lesson, we make several additional distinctions. We study estimation where there is some function $f(\cdot)$ that is a function of the unknown parameter ϕ, i.e., $y[n] = f(\phi, n)$

We assume that know the form of the function $f(\cdot)$ but not the value of ϕ. However, we do assume that ϕ is constant. This is referred to as the *classical* approach to estimation.

The **Bayesian approach** to estimation still assumes the form of the function $f(\cdot)$ but it also assumes that ϕ is a *random variable* and we need to estimate the current realisation of that random variable. In particular, it is assumed that the probability distribution (whether a PDF or PMF) of ϕ is known. This approach, which is based on the Bayes' theorem, is useful because it enables us to take advantage of any prior information that we have about ϕ in the design of our estimator. Some popular approaches based on the Bayesian approach include minimum mean square error estimation, maximum a posteriori estimation, and Kalman filtering. You can read more about these in Chapters 10-13 of Kay.

A common feature about both classical and Bayesian approaches is that we assume knowledge about the function $f(\cdot)$. There are also more automated methods that make no assumptions about the underlying function $f(\cdot)$. Instead, we try to learn about ϕ from simply the signal $y[n]$ alone. **Artificial neural networks** follow this approach.

18.3 Linear Model

One class of problems where it is "easy" to determine a suitable estimator are those that follow a *linear model*. You may have seen this model in ES197 and ES2C7. For example, consider a straight-line model for $y[n]$, such that we have

$$y[n] = A + Bn + w[n], \tag{18.2}$$

where A and B are unknown constants and $w[n]$ refers to noise that is corrupting the signal. We will assume that $w[n]$ is a sequence of Gaussian random variables with mean $\mu_w = 0$ and variance σ_w^2 (we are also making assumptions about the spectral nature of the noise, such that we call the noise *white*, but we will not elaborate on this detail here). If we take Eq. 18.2 for each value of n, and there are N data points in total, then we have a system of N equations to solve the 2 unknowns (A and B). The most convenient and compact way to describe these equations is in matrix form. We can write the data sequence, parameters, and noise in column vector form, i.e.,

$$\vec{y} = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[N-1] \end{bmatrix}, \qquad \vec{\phi} = \begin{bmatrix} A \\ B \end{bmatrix}, \qquad \vec{w} = \begin{bmatrix} w[0] \\ w[1] \\ w[2] \\ \vdots \\ w[N-1] \end{bmatrix}, \tag{18.3}$$

and to include how A and B affect $y[n]$ we build the **observation matrix $\mathbf{\Theta}$**. Since here we have two parameters, the observation matrix is an $N \times 2$ matrix structured as follows:

$$\mathbf{\Theta} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & N-1 \end{bmatrix}, \tag{18.4}$$

where the first column reflects the fact that $y[n]$ is a constant function of A and the second column reflects the fact that $y[n]$ is a linear function of B where the scaling factor in the nth row is n.

This linear model problem can now be written as

$$\vec{y} = \mathbf{\Theta}\vec{\phi} + \vec{w}, \tag{18.5}$$

and this form can be achieved for *any* linear model. Assuming that $\vec{w}$ is a vector of white Gaussian noise, then the *optimal* estimate $\hat{\vec{\phi}}$ (which *minimises the variance of the error*) for the parameters is

$$\hat{\vec{\phi}} = (\mathbf{\Theta}^{\mathrm{T}}\mathbf{\Theta})^{-1}\mathbf{\Theta}^{\mathrm{T}}\vec{y}, \tag{18.6}$$

where $\mathbf{\Theta}^{\mathrm{T}}$ is the **transpose** of the matrix $\mathbf{\Theta}$ (i.e., every element swaps its row and element indices). In this particular example, $\mathbf{\Theta}^{\mathrm{T}}$ is

$$\mathbf{\Theta}^{\mathrm{T}} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 0 & 1 & 2 & \cdots & N-1 \end{bmatrix}. \tag{18.7}$$

From the estimate $\hat{\vec{\phi}}$, we can calculate the corresponding predictions for $y[n]$ for each value of n (even $n \geq N$ if we wanted to extrapolate). We write these predictions as

$$\hat{\vec{y}} = \mathbf{\Theta}\hat{\vec{\phi}}, \tag{18.8}$$

where here we omit the noise vector.

It can be shown that the estimate $\hat{\vec{\phi}}$ minimises the **mean square error** (MSE), which we calculate for the signal $y[n]$ as follows:

$$\mathrm{MSE}(\hat{\vec{y}}) = \frac{1}{N}\sum_{n=0}^{N-1}(\hat{y}[n] - y[n])^2, \tag{18.9}$$

where $\hat{y}[n]$ is the nth element of $\hat{\vec{y}}$.

EXTRA 18.2: Optimality in a Minimum Variance Sense

It is useful for us to clarify what we mean by "optimal" in a minimum variance sense, without getting into too much mathematical detail. We recall the definition in Lesson 17 for the variance of random variable X:

$$\sigma_X^2 = E\left[(X - \mu_X)^2\right], \tag{18.10}$$

where the mean of X is μ_X and we were able to separately derive μ_X. Variance in estimation has a similar meaning, except that here μ_X becomes our predicted model $\hat{y}[n]$ and the random variable X becomes our observed signal $y[n]$. The variance σ_X^2 becomes a measure that we want to *minimise*, as obviously the best that we can do is if $\hat{\phi} = \phi$. We cannot take a true expectation from a finite number of samples, so the mean square error in Eq. 18.9 acts as a tractable equivalent.

We emphasise that Eq. 18.6 is optimal if the noise $\vec{w}$ is white Gaussian noise. If the noise were not white, then we could include information about the noise in a modified version of Eq. 18.6 to maintain optimality. For more detail, see Section 4.5 of Kay.

Importantly, our signal $y[n]$ can be more generally defined than in Eq. 18.2 and still follow a linear model whose optimal estimate is given by Eq. 18.6. In particular:

- $y[n]$ can depend on more variables than just the index n. For example, $y[n]$ could depend on the underlying continuous time t, or some other variable *that we must know or be able to measure* for each sample in the sequence. Importantly, $y[n]$ does not need to depend on time at all, though we are usually interested in time-varying signals in the context of this module.

- $y[n]$ can include polynomial terms of the function variables. The *linearity* of the linear model means that $y[n]$ must be a linear function of the *unknown* parameters.

For completeness, let us now write a more general form for the components in Eq. 18.3. We assume that there are P unknown parameters and Q known variables that can vary for each sample n. The structures of $\vec{y}$ and $\vec{w}$ are the same as in

Eq. 18.3. The parameter vector becomes

$$\vec{\phi} = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_P \end{bmatrix}. \tag{18.11}$$

It is inconvenient to write a general form for the observation matrix $\mathbf{\Theta}$, which has size $N \times P$, thanks to the possibility of multiple variables. It is easier for us to see formulations in examples.

Example 18.1: Curve Fitting

Suppose that we have data that we want to match to a polynomial curve that is a function of the underlying continuous time t, such that the nth sample is at $t = nT_\mathrm{s}$. We write the time-domain function as

$$y(t) = \phi_1 + \phi_2 t + \phi_3 t^2 + \ldots \phi_P t^{P-1} + w(t),$$

which we sample at times $\{0, T_\mathrm{s}, 2T_\mathrm{s}, \ldots, (N-1)T_\mathrm{s}\}$. For this problem, the observation matrix is

$$\mathbf{\Theta} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & T_\mathrm{s} & T_\mathrm{s}^2 & \ldots & T_\mathrm{s}^{P-1} \\ 1 & 2T_\mathrm{s} & 4T_\mathrm{s}^2 & \ldots & (2T_\mathrm{s})^{P-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (N-1)T_\mathrm{s} & (N-1)^2 T_\mathrm{s}^2 & \ldots & ((N-1)T_\mathrm{s})^{P-1} \end{bmatrix}.$$

To consider a more specific example, suppose that we have data that we want to fit to the function

$$y(t) = A + Bt + Ct^2 + w(t),$$

which we sample every $T_\mathrm{s} = 0.1\,\mathrm{s}$ for $10\,\mathrm{s}$, then the observation matrix is

$$\boldsymbol{\Theta} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0.1 & 0.01 \\ 1 & 0.2 & 0.04 \\ \vdots & \vdots & \vdots \\ 1 & 10 & 100 \end{bmatrix}.$$

Example 18.2: FIR Filter Estimation

We recall from Lesson 12 that the difference equation for an Mth order FIR filter is

$$y\left[n\right] = \sum_{k=0}^{M} b\left[k\right] x\left[n-k\right],$$

where the $b\left[\cdot\right]$ terms are the transfer function numerator coefficients (recall that FIR filters have no feedback and hence there are no denominator coefficients). The $b\left[\cdot\right]$ terms are also the tap multiplier coefficients in the implementation of the filter. In practice, there are many natural systems that we can describe as a (noisy) FIR filter, where we need to *estimate* the filter taps to learn about the system. We assume that we have a noise term such that our actual observed signal is

$$y\left[n\right] = \sum_{k=0}^{M} b\left[k\right] x\left[n-k\right] + w\left[n\right],$$

and we seek to estimate the taps from N samples (where $N \geq M+1$) given a *known* input sequence $x\left[n\right]$. While it can actually be shown to be suboptimal, let us assume that the input sequence is a step function, i.e., $x\left[n\right] = 1, \forall n \geq 0$. The observation matrix is of size $N \times (M+1)$ and is

$$\boldsymbol{\Theta} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}.$$

More generally, for any input function $x[n]$, the observation matrix is

$$\boldsymbol{\Theta} = \begin{bmatrix} x[0] & 0 & 0 & \dots & 0 \\ x[1] & x[0] & 0 & \dots & 0 \\ x[2] & x[1] & x[0] & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x[N-1] & x[N-2] & x[N-3] & \dots & x[N-(M+1)] \end{bmatrix}.$$

EXTRA 18.3: The DFT Solves a Linear Model

We recall from Lesson 14 that the discrete Fourier transform (DFT) converts a sequence of N data points into N spectral points. It can be shown that the DFT is *equivalently* a linear estimation of the amplitudes of the sinusoid components in the signal, such that Eq. 18.3 is equivalent to finding the DFT coefficients. For more detail, refer to Example 4.2 of Kay. We clarify that Kay uses the sinusoidal representation of the DFT, which is equivalent to the complex exponential one that we studied in Lesson 14.

We make note of another generalisation of the linear model. Suppose that $x[n]$ includes a known signal component, such that is of the form

$$\vec{y} = \boldsymbol{\Theta}\vec{\phi} + \vec{s} + \vec{w}, \tag{18.12}$$

where $\vec{s}$ is an $N \times 1$ vector of known samples. It is straightforward to account for this in the estimator by subtracting both sides by $\vec{s}$, such that the optimal solution becomes

$$\hat{\vec{\phi}} = (\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{\Theta})^{-1}\boldsymbol{\Theta}^{\mathrm{T}}(\vec{y} - \vec{s}). \tag{18.13}$$

The form of Eq. 18.13 is convenient if we know that our signal is contaminated by some large interference with known characteristics.

18.3.1 Solving Linear Models

The wary student will observe that we have not yet discussed how to solve Eq. 18.3. Because of its inclusion of matrix multiplications and inversions, it is *very* cumbersome to evaluate by hand, even for very short signals. For this module, we are more interested in setting up the observation matrix $\boldsymbol{\Theta}$ and then evaluating on a

computer. Given data signal `y` (as a *column* vector) and linear observation matrix `Obs` defined in MATLAB, the model estimate is simply

```
theta = Obs\y;
```

This method is also known as **linear regression** or **ordinary least squares** (OLS). We note that the form of the observation matrix $\boldsymbol{\Theta}$ had to be assumed and may be unknown. In practice, we may need to test multiple candidate models, each with a different corresponding $\boldsymbol{\Theta}$, and choose the one that is most accurate. We can compare accuracies by calculating the MSE associated with each model as found using Eq. 18.9 where we use the estimate to find the corresponding predicted signal vector $\hat{\vec{y}}$. If we find multiple models that produce a very similar MSE, then intuition would lead us to select the *simplest* model (and we would assume that more complex models have too many parameters).

18.4 Least Squares

The solution in Eq. 18.3 to the linear model is more generally also known as the **least squares estimate**, whether or not it is the optimal estimate (as this would depend on the statistics of the noise $\vec{w}$). Here we discuss a variation of least squares estimation and how it modifies the ordinary least squares (OLS) estimate.

Weighted least squares includes a symmetric weight matrix $\boldsymbol{W}$. The simplest weighting matrix, where each sample is associated with a positive weight $W[n] > 0$, is diagonal where the nth diagonal element is $W[n]$. The benefit of including a weight matrix is to place more emphasis on more reliable samples (without simply discarding any that are deemed less reliable). For example, if the variance of the noise is known to vary with n, such that the variance of the noise in the nth sample is σ_n^2, then a reasonable choice of weight is $W[n] = 1/\sigma_n^2$. Given $\boldsymbol{W}$, the least squares estimate becomes

$$\hat{\vec{\phi}} = (\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{\Theta})^{-1}\boldsymbol{\Theta}^{\mathrm{T}}\boldsymbol{W}\vec{y}, \tag{18.14}$$

and we must rely on a more general approach for solving Eq. 18.14. Fortunately, MATLAB has such a general purpose function for least squares problems called `lscov`. First, we show that `lscov` can be used to find OLS estimates which is equivalent to using the backslash operator:

```
theta = lscov(Obs,y);
```

Once again, we note that the signal `y` must be a column vector. To perform weighted least squares, where we have a diagonal weight matrix $\boldsymbol{W}$ where the nth

diagonal element is $W[n]$, we simply add the *column vector* of weights $W[n]$ as a third argument:

```
theta = lscov(Obs,y,W);
```

EXTRA 18.4: Other Variations of Least Squares

There are other interesting variations of least squares estimation available, but the corresponding mathematics is too cumbersome for us to even present final results. **Sequential least squares** (or **recursive least squares**) enables us to update a least squares estimate as we add new data points, *without* needing to recalculate the estimate from the entire data set. It can be performed in MATLAB using the `rlsFilt` function in the DSP System Toolbox. **Constrained least squares** is a generalisation where the unknown parameters must have constraints applied. For example we might know that some parameters must be within a certain range, e.g., positive or negative, or that some of the unknown parameters depend on other unknown parameters in some way. When the both the model and the constraints are linear, constrained least squares can be performed using the `lsqlin` function in the Optimization Toolbox.

18.5 Maximum Likelihood Estimation

While we will not elaborate on the mathematical details, we will find it useful to introduce the concept of **maximum likelihood estimation** (MLE). MLE is quite popular as it can be readily computed for complicated estimation problems. It is also *approximately optimal* in a minimum variance sense if we have a large number of signal samples available.

We recall the concept of probability density functions (PDFs) from Lesson 17. The MLE is found by determining $\hat{\vec{\phi}}$ that maximises the PDF of the signal $y[n]$, which depends on the statistics of the noise $w[n]$. The PDF is maximised by taking the derivative of the signal PDF with respect to the estimation parameters. Once again, we will not cover the mathematical details, but we only need to assume the *shape* of the PDF. Given some type of probability distribution, the MLE can be found.

MATLAB has the `mle` function in the Statistics and Machine Learning Toolbox. The only arguments needed are the data set and the assumed probability distribu-

tion (the default is a Gaussian distribution). A wide selection of distributions are available (see the documentation for `mle`), or you can define a custom distribution.

Example 18.3: MLE Estimation

Write a MATLAB script that generates 1000 realisations of a Gaussian random variable with mean 1 and variance 4. Try to estimate the mean and variance using `mle`.

```
x = 1 + 2*randn(1,1000); % Create vector of random elements
    with mean 1 and variance 4 (so standard deviation is 2)
phat = mle(x); % Maximum likelihood estimation. Output
    arguments are estimated 1) mean, 2) standard deviation
meanX = phat(1);
varX = phat(2)^2;
```

18.6 Comments on Bias and Accuracy

EXTRA 18.5: Comments on Bias and Accuracy

There are important concepts regarding the formal performance of estimators that we have omitted for clarity, but we briefly comment on here for your interest.

We have omitted the notion of **bias** in an estimator. If an estimator is **unbiased**, then on average it will estimate the true value of a parameter. In the case of the linear model, the optimal minimum variance estimator is also unbiased. The maximum likelihood estimator is in general biased, but it becomes unbiased as we take an asymptotically large number of samples. Bias isn't necessarily bad; we may be able to find an estimator with lower variance if we are willing to accept some bias. You can read more about bias and unbiased estimation in Chapter 2 of Kay.

For a given estimator, we can always wonder if we can do better. It is helpful to be aware of bounds on estimator variance, and in particular on the variance of an unbiased estimator. A lower bound on estimator variance enables us to evaluate any proposed estimator with what is possible, and informs whether any improvement is physically possible. One such bound that is (relatively) easy to determine is the **Cramer-Rao lower bound** (CRLB).

The CRLB is quite useful because, upon finding it, we can also determine whether an estimator exists that can attain the bound. However, the mathematics of the CRLB are quite involved and include taking derivatives of the logarithm of the probability distribution of interest (as is also needed in maximum likelihood estimation). You can read more about the CRLB in Chapter 3 of Kay.

18.7 Summary

- **Signal estimation** problems involve finding the *values of unknown parameters* that are embedded in a signal of interest. These are found across all areas of signal processing.
- If the signal of interest follows a **linear model** of the unknown parameters, then the parameters can be estimated using **linear regression**, which is a **least squares solution**.
- Variations of least squares include **weighted least squares**, **sequential least squares**, and **constrained least squares**.
- **Maximum likelihood estimation** can find the parameters of a probability distribution as long as we know the shape of the distribution.

18.8 Further Reading

- Chapters 1, 4, 7-8 of "Fundamentals of Statistical Signal Processing, Volume 1: Estimation Theory," S. M. Kay, Prentice Hall, 1993.

18.9 End-of-Lesson Problems

1. Suppose that we want to fit the signal $y(t)$ to the function

$$y(t) = A + Bt + Ct^2 + Dt^3 + w(t),$$

where we sample every $T_s = 0.5\,\text{s}$ for $3\,\text{s}$. Write out the complete observation matrix $\mathbf{\Theta}$.

2. Suppose that we are try to estimate the filter coefficients $b[\cdot]$ of the system defined by the difference equation

$$y[n] = b[0]\,x[n] + b[1]\,x[n-1] + b[2]\,x[n-2] + w[n].$$

Determine the observation matrix $\mathbf{\Theta}$ where we perturb this system with the input sequence $x[n] = \{1, 0, 1, 1, 0\}$.

3. Suppose that we want to estimate the parameters in a linear model where we know that the first 5 samples were contaminated with noise of variance $\sigma^2 = 2$ and the last 5 samples were contaminated with noise of variance $\sigma^2 = 5$. What would be a suitable weight vector to perform weighted least squares?

4. Poisson random variables are commonly used to describe the occurrence of random events. One of their pioneering applications was to model the number of Prussian soldiers killed by horse kicks, but they can be used for counting any kind of independent event, e.g., phone calls to a call centre, chemical reactions in some volume, photons emitted by a light source, etc. A convenient feature of Poisson random variables is that they are characterised by a single parameter: the mean λ. Use the `poissrnd` function in MATLAB to generate 100 Poisson realisations with mean $\lambda = 1$, then see how well the `mle` function can estimate the mean.

Lesson 19

Correlation and Power Spectral Density

We finish our brief study of random signal processing with additional mathematical tools for comparing signals in both the time and frequency domains. These enable us to quantify similarities between signals or how a signal varies with time. While we will not cover the use of these tools in detail, they give us additional methods for performing estimation of random signals.

19.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Apply** cross-correlation to compare two signals in the time domain.
2. **Apply** autocorrelation to compare one signal with its future values.
3. **Understand** properties of power spectral density (PSD).
4. **Understand** that the PSD characterises the frequency spectra of signals.

19.2 Correlation

Correlation gives us a measure of time-domain similarity between two signals. Consider pairs of data in Fig. 19.1. The data sets paired to make the blue circles are highly correlated, whereas the data sets paired to make the red triangles are less

correlated. Given discrete-time sequences $x_1[n]$ and $x_2[n]$ of length N, the **cross-correlation** between them can be approximated as

$$R_{X_1X_2}[k] \approx \frac{1}{N-k}(x_1[0]\,x_2[k] + x_1[1]\,x_2[k+1] + x_1[2]\,x_2[k+2] + \ldots + x_1[N]\,x_2[k+N]) \tag{19.1}$$

$$= \frac{1}{N-k}\sum_{n=0}^{N-1} x_1[n]\,x_2[k+n]\,, \tag{19.2}$$

where k is the time shift of the $x_2[n]$ sequence relative to the $x_1[n]$ sequence. It is an approximation because the signal lengths are finite and the signals could be random, but in practice this is the form that is used to calculate the cross-correlation.

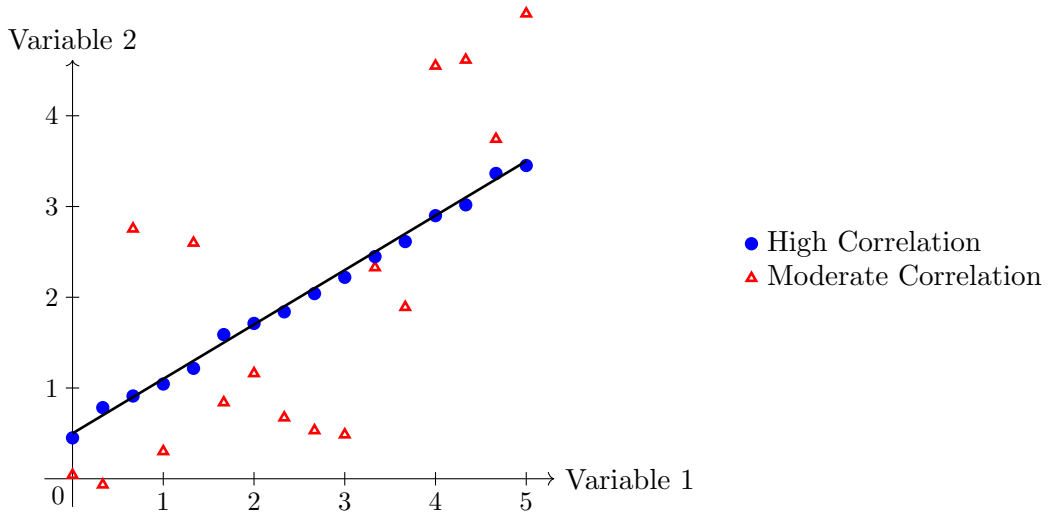


Figure 19.1: Signal correlation of two pairs of sets of data.

EXTRA 19.1: Cross-Correlation of Analogue Signals

For this module we are more concerned with calculating the cross-correlation for specified sequences and not deriving it for random processes or continuous-time signals. However, for completeness, the cross-correlation between deter-

ministic analogue signals $x_1(t)$ and $x_2(t)$ is

$$R_{X_1X_2}(\tau) = \lim_{T\to\infty} \frac{1}{2T} \int_{t=-T}^{T} x_1(t)\, x_2(\tau + t)\, dt, \tag{19.3}$$

where τ is the continuous-time shift between the two signals.

Example 19.1: Discrete Cross-Correlation

Calculate the cross-correlation between

$$x_1[n] = \{0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5\}$$
$$x_2[n] = \{6, 8, 10, 0, 2, 4, 6, 8, 10, 0, 2, 4\}$$

We have $N = 12$ and from Eq. 19.2 we can write

$$R_{X_1X_2}[0] = \frac{1}{12}(0 + 8 + 20 + 0 + 8 + 20 + 0 + 8 + 20 + 0 + 8 + 20) = 9.33$$
$$R_{X_1X_2}[1] = \frac{1}{11}(0 + 10 + 0 + 6 + 16 + 30 + 0 + 10 + 0 + 6 + 16) = 8.55$$
$$R_{X_1X_2}[2] = \frac{1}{10}(0 + 0 + 4 + 12 + 24 + 40 + 0 + 0 + 4 + 12) = 9.60$$
$$R_{X_1X_2}[3] = \frac{1}{9}(0 + 2 + 8 + 18 + 32 + 50 + 0 + 2 + 8) = 13.33,$$

and so on. We find that

$$R_{X_1X_2}[k] = \{9.33, 8.55, 9.60, 13.33, 10.50, 8.86, 9.33, 6.40, 4.00, 3.33, 2.00, 0\}.$$

We observe that the peak value of 13.33 occurs when $k = 3$. This makes sense because we see that $x_2[n]$ is twice $x_1[n]$ and delayed by 3.

A special case of a correlation function is when we correlate a signal with *itself*, i.e., $x_2[n] = x_1[n]$ or $x_2(t) = x_1(t)$. We call this the **autocorrelation**. It gives us a measure of whether knowing the current value of the signal says anything about a future value, which is particularly useful for random signals. The autocorrelation of a signal has several important properties:

1. The autocorrelation for zero delay is the same as the signal's mean square value. The autocorrelation is never bigger for any non-zero delay.

2. Autocorrelation is an even function of k or τ, i.e., $R_{X_1X_1}[k] = R_{X_1X_1}[-k]$.

3. The autocorrelation of the sum of two *uncorrelated* signals is the sums of the autocorrelations of the two individual signals. In other words, for $x_1[n]$ and $x_2[n]$ uncorrelated, we have

$$y[n] = x_1[n] + x_2[n] \implies R_{YY}[k] = R_{X_1X_1}[k] + R_{X_2X_2}[k]. \quad (19.4)$$

Example 19.2: Discrete Autocorrelation

Calculate the autocorrelation of

$$x[n] = \{0, 1, 2, 3, 2, 1, 0\}$$

We have $N = 7$ and from Eq. 19.2 we can write

$$R_{XX}[0] = \frac{1}{7}(0 + 1 + 4 + 9 + 4 + 1 + 0) = 2.71$$
$$R_{XX}[1] = \frac{1}{6}(0 + 2 + 6 + 6 + 2 + 0) = 2.67$$
$$R_{XX}[2] = \frac{1}{5}(0 + 3 + 4 + 3 + 0) = 2.00$$

and so on. We find that

$$R_{XX}[k] = \{2.71, 2.67, 2.00, 1.00, 0.33, 0, 0\}.$$

As expected for autocorrelation, the peak value $R_{XX}[0] = 2.71$ occurs when $k = 0$, and this is also the mean square value of the signal.

Example 19.3: Correlation in MATLAB

We can calculate the cross-correlation and autocorrelation of discrete signals in MATLAB using the `xcorr` function. To be consistent with our definitions, we use the syntax

```
R = xcorr(X2, X1, 'unbiased')
```

and this will normalise the result properly. The input signals $X1$ and $X2$ must be of the same length (N); also note their ordering. The output R will

actually be of length $2N-1$ because it includes negative values of k. Since our definition does not account for negative values of k, simply index the last N elements in R, e.g., `R((end-N+1):end)`.

EXTRA 19.2: Estimation Using Correlation

One common application of cross-correlation and autocorrelation is for estimation. Here are a few examples, which we will not cover in detail:

1. **Ranging** – we can estimate the distance to an object by measuring the delay in a signal reflected by that object. The reflected signal might be strongly attenuated and buried in noise, but we can cross-correlate with the original signal to help recover the delay.

2. **Detect a Periodic Signal** – we may want to detect the presence of a periodic signal under heavy noise. We could use autocorrelation and look for the signal at high delay k or τ (since noise would have no correlation at high delay), or we could cross-correlate against a target periodic signal. In the latter case we could also measure a phase shift but we would need to accurately know the frequency in order to do so.

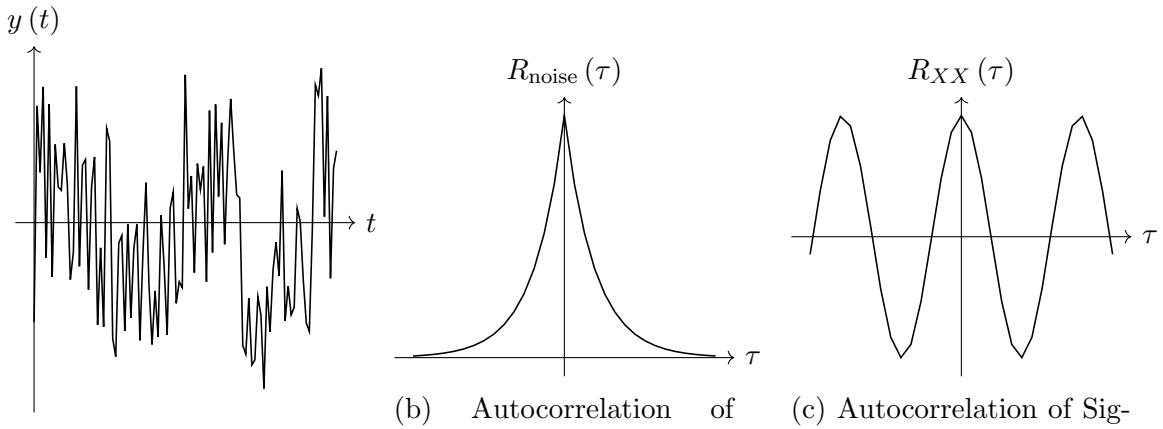
Example 19.4: Detecting Periodic Signal in Noise

Consider the noisy signal in Fig. 19.2(a). Let's say that we are trying to detect whether the signal contains a sinusoid of a particular frequency ω, i.e., whether

$$y(t) = A\sin(\omega t + \theta) + w(t) = x(t) + w(t),$$

where $w(t)$ is a random noise signal. We can assume that $x(t)$ and $w(t)$ are uncorrelated. From the properties of autocorrelation, we know that the autocorrelation of $y(t)$ is the sum of the autocorrelations of the two components (shown in Fig. 19.2(b) and (c)). Since random noise should have no correlation at high τ, only the autocorrelation of $x(t)$ should remain after a certain time, so we should be able to recover ω.

Alternatively, we could cross-correlate $y(t)$ with another sinusoid of our desired frequency ω. From this method the phase θ is also recoverable, but we need to accurately know ω for this method.



(a) Original Noisy Signal.

(b) Autocorrelation of Noise

(c) Autocorrelation of Signal

Figure 19.2: Plots for Example 19.4.

19.3 Power Spectral Density

EXTRA 19.3: Power Spectral Density

Suppose that we have a "very long" random signal that it is effectively infinite in duration. Such a signal is also infinite in energy, so we cannot find its transform directly. However, we are not prevented from applying spectral analysis because we can transform a function of the delay in the signal (i.e., the autocorrelation). The Fourier transform of an autocorrelation function is known as the **Power Spectral Density** (PSD) $S_{XX}(\omega)$ of the signal, and in continuous-time it is defined as

$$S_{XX}(\omega) = \int_{-\infty}^{\infty} R_{XX}(\tau)\, e^{-j\omega\tau} d\tau. \tag{19.5}$$

The PSD gives the power per frequency interval in a signal. We can also recover the autocorrelation from the PSD via an inverse Fourier transform:

$$R_{XX}(\tau) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S_{XX}(\omega)\, e^{j\omega\tau} d\omega. \tag{19.6}$$

We are not focusing here on the math of the power spectral density, but we make note of several properties:

- PSD is a measurement of *power* per unit bandwidth of a signal
- PSD has non-negative real values as both power and frequency are real
- Phase information is lost in the PSD
- PSD is an even function of frequency, i.e., $S_{XX}(\omega) = S_{XX}(-\omega)$
- A PSD will contain impulses at frequencies that correspond to sinusoidal components in the time domain signal. If the time domain signal has a sinusoid component with frequency ω, then the PSD will contain pulses at $\pm\omega$.

Following up on our earlier discussion of detection with correlation, we can also supplement with the PSD. If we find the autocorrelation of a noisy signal, then we can then take the PSD to more easily detect the frequency of the underlying time-domain signal.

19.4 Summary

- **Correlation** enables us to compare a signal with itself or with another signal.
- **Cross-correlation** measures the time-domain correlation between one signal and the delay in another signal.
- **Autocorrelation** measures the time-domain correlation between a signal and its future values.
- The **Power Spectral Density** is the Fourier transform of the autocorrelation and measures power per unit bandwidth.

19.5 Further Reading

- Chapters 1-4 and Chapter 9 of "Signal processing: Principles and Applications," D. Brook and R. J. Wynne, Edward Arnold, 1988.

19.6 End-of-Lesson Problems

1. Calculate the cross-correlation between $x_1[n] = \{3, 2, 1, 3, 2, 1\}$ and $x_2[n] = \{-4, -2, -6, -4, -2, -6\}$. Verify your result using MATLAB. When are the signals most correlated and why?
2. Calculate the autocorrelation of $x[n] = \{1, -2, 3, -1, 0, -2, 1, 3, 2\}$. Verify your result using MATLAB. What is the mean square value of the signal?

Part IV

Introduction to Image Processing

Lesson 20

Image Processing

The fourth and final part of this module is a *very brief* study of **Image Processing**. We introduce image processing as an application of filtering as we have studied it in this module. Image processing is ideal for this purpose because we can "see" the results, but we also need the concept of multi-dimensional signals. We are going to take a simplified approach and focus on very small black-and-white or greyscale images that can be readily processed by hand. We will also demonstrate several MATLAB functions that can efficiently perform these tasks on much larger images and in colour.

20.1 Learning Outcomes

By the end of this lesson, you should be able to ...

1. **Understand** images as two-dimensional signals with colour data.
2. **Apply** filters to images.
3. **Implement** image filters in a computing package.

20.2 Images as Signals

We discussed in Lesson 1 that physical signals are usually defined over time, frequency, or space. Images are inherently spatial signals (and when they are also defined over time then we have a video). Image processing generally relies on a *digital* representation of signals, such that signals are discretised *both* spatially and in amplitude. Thus, we note that we consider images as *truly* digital signals. In addition, there are a few other practical differences compared with our earlier studies of discrete-time systems:

1. Images are typically defined over *two*-dimensional space, so we need an additional indexing component to accommodate the additional dimension. A discrete 2-D image point is referred to as a **pixel**.

2. Image signal *amplitudes can also have multiple components* corresponding to the colour information. Depending on how the colour information is encoded, each pixel will include one or more colour values.

As a result, images are *generally* represented as three-dimensional arrays, where the first two dimensions describe the location and the third dimension gives the colour information for each pixel. There are many different ways to represent colour information. Here are the main four methods used by MATLAB (and shown in Fig. 20.1):

1. **Binary** – each pixel has value 0 or 1 to represent black and white, respectively.

2. **Indexed** – each pixel has one value within a range corresponding to a single colour map. A colour map is a pre-determined list of colours and the index refers to a row of colour data within the list.

3. **Grayscale** – each pixel has one value within a grayscale range between 0 (black) and 1 (white).

4. **Truecolor** – each pixel has *three* associated values corresponding to red, blue, and green channels. These are referred to as RGB values, but it is also possible to translate from other colour definition formats (e.g., HSV).

For convenience in applying image signal processing "by hand," we will focus on binary and grayscale images. These only need a single value to define a pixel, and the meaning of each value is fairly intuitive (i.e., black, white, or something in between). Thus, we only need a *two*-dimensional array to define an image signal, which we will generally refer to using the notation $f[i][j]$. We will follow the same indexing conventions as MATLAB, where i refers to the vertical coordinate (i.e., row), j refers to the horizontal coordinate (i.e., column), and the numbering is such that $(i, j) = (1, 1)$ refers to the top left pixel. We show an example of pixel values and coordinates in Fig. 20.2.

While grayscale colour values are supposed to be defined within the range $[0, 1]$, we will often write in whole numbers for convenience. We just need to normalise the signal (e.g., by the highest value) before we try to view it as an image.

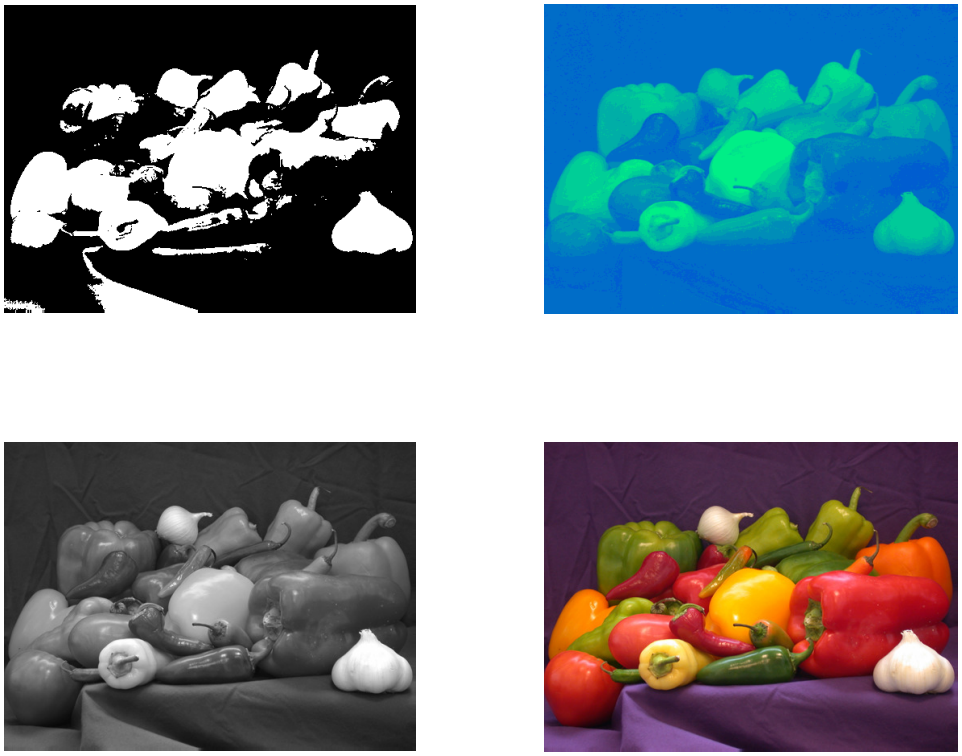


Figure 20.1: Colour representations in MATLAB using official MATLAB example image file `'peppers.png'`. Clockwise from top left: binary, indexed (with colour map `winter`), truecolor, and grayscale.

8	1	6
3	5	7
4	9	2

(a) Image pixel values $f[i][j]$.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

(b) Image pixel coordinates of form (i, j).

Figure 20.2: Our representation of images.

20.3 Re-Visiting Digital Convolution

Back in Lesson 9, we defined discrete convolution for a one-dimensional system (i.e., the signal was a single quantity that varied over time). Recall that we wrote the

system output $y[n]$ as

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]\, h[n-k] = x[n] * h[n] = h[n] * x[n]. \tag{20.1}$$

where $x[n]$ is the input and $h[n]$ describes the system's impulse response. While we noted that discrete-time convolution was easier to evaluate than continuous-time convolution, we focused on the use of difference equations for our time-domain analysis. Here, we re-introduce ourselves to discrete convolution with an example.

Example 20.1: 1D Discrete Convolution

Find the output of a system where

$$h[n] = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, \ldots\right\}$$
$$x[n] = \{1, 2, 3, 4, 5, 4, 3, 2, 1, 0, 0, \ldots\},$$

and we assume for consistency with MATLAB that the indexing starts at $n = 1$ (such that $h[n] = 0$ and $x[n] = 0$ for $n < 1$). Sketch and compare the input and output.

Using Eq. 20.1, we can find that

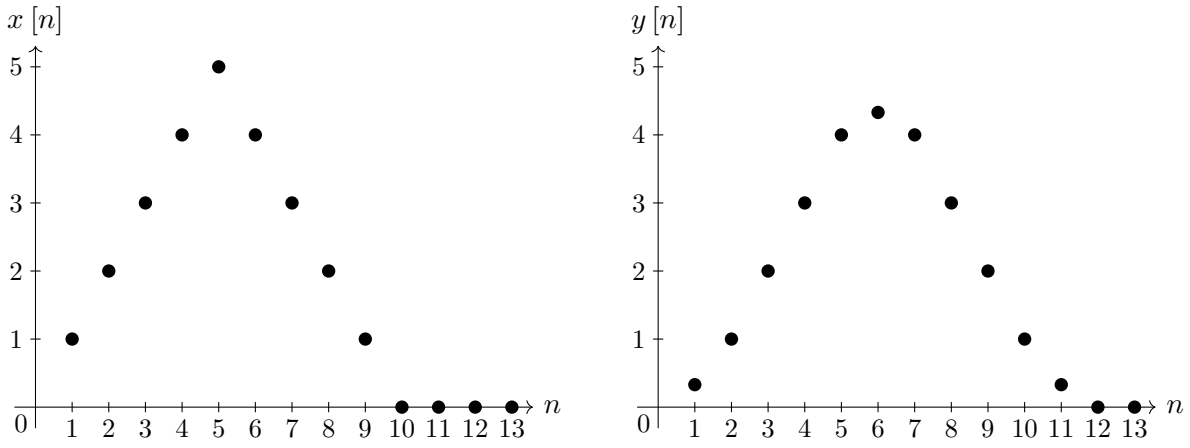
$$y[1] = x[1]\, h[1] = 1 \times \frac{1}{3} = \frac{1}{3}$$
$$y[2] = x[1]\, h[2] + x[2]\, h[1] = 1 \times \frac{1}{3} + 2 \times \frac{1}{3} = 1$$
$$y[3] = x[1]\, h[3] + x[2]\, h[2] + x[3]\, h[1] = 1 \times \frac{1}{3} + 2 \times \frac{1}{3} + 3 \times \frac{1}{3} = 2,$$

and so on. We will find that the output is

$$y[n] = \boxed{\{0.33, 1.00, 2.00, 3.00, 4.00, 4.33, 4.00, 3.00, 2.00, 1.00, 0.33\}},$$

and 0 for higher values of n. We plot the input and output to the same scale in Fig. 20.3. We see that the output "follows" the input somewhat. In fact, by inspection of $h[n]$, this system is a moving-average filter that averages over 3 samples.

The system in Example 20.1 can be readily translated as applying to a one-dimensional image. We can interpret each system component (input, output, and



(a) System Input. (b) System Output.

Figure 20.3: Input and output for Example 20.1.

impulse response) as a 1-D vector. We do this by repeating the problem visually in the following example.

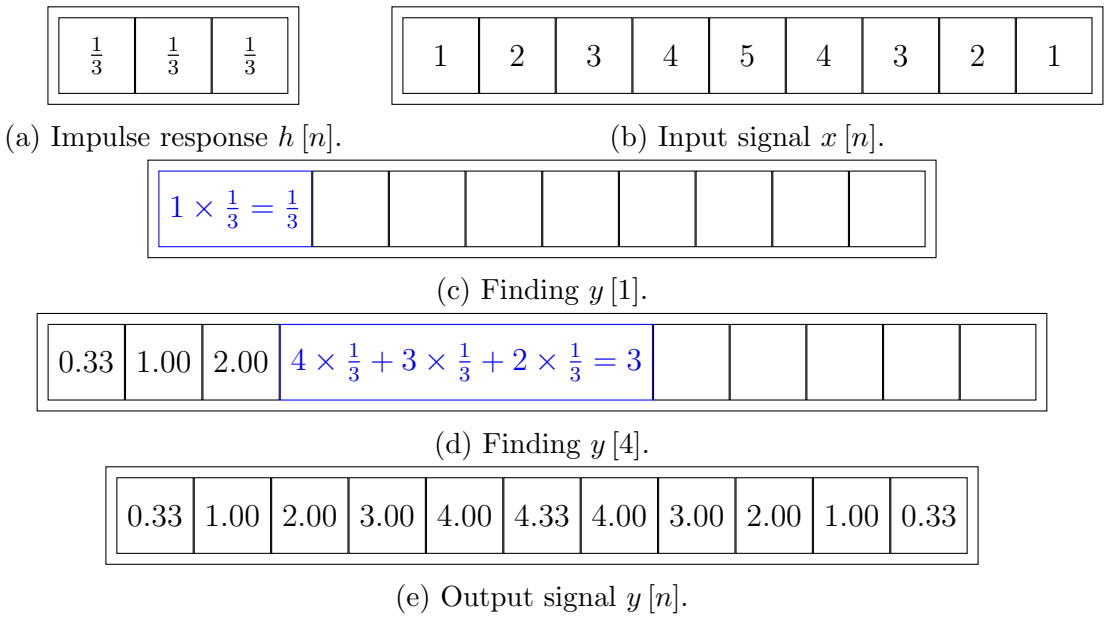
Example 20.2: Visual 1D Discrete Convolution

Let's repeat Example 20.1 visually, as shown in Fig. 20.4. We start by drawing out the arrays $h[n]$ and $x[n]$. To convolve, we need to "flip" $h[n]$ (which in this case brings no change) and picture moving the flipped $h[n]$ along the array $x[n]$. Every time we advance $h[n]$ by one element, we find the products of the corresponding $h[n]$ and $x[n]$, add them together, and enter the sum in the element of $y[n]$ corresponding to the lead element of $h[n]$. By progressing in this manner we find all of the elements of $y[n]$. We note that if $h[n]$ has length N and $x[n]$ has length M, then $y[n]$ has length $N + M - 1$.

20.4 Image Filtering

Now that we have re-visited 1D convolution and understood it from the perspective of 1D image signals, we are ready to consider the filtering of 2D images. We refer to the impulse response $h[i][j]$ as the **filter** or the **kernel**. Besides the additional dimension, the filtering process to determine the output $y[i][j]$ from the input $x[i][j]$ is effectively the same:

1. "Flip" the impulse response $h[i][j]$ to get $h^{\star}[i][j]$. Flipping is achieved by



(a) Impulse response $h\,[n]$.

(b) Input signal $x\,[n]$.

(c) Finding $y\,[1]$.

(d) Finding $y\,[4]$.

(e) Output signal $y\,[n]$.

Figure 20.4: Visually solving Example 20.2.

mirroring all elements in $h\,[i]\,[j]$ around the centre element. By symmetry, $h^{\star}\,[i]\,[j]$ is sometimes the same as $h\,[i]\,[j]$.

2. Move the "flipped" impulse response $h^{\star}\,[i]\,[j]$ along the input image $x\,[i]\,[j]$.

3. Each time the kernel is moved, multiply all of the elements of $h^{\star}\,[i]\,[j]$ by the corresponding covered pixels in $x\,[i]\,[j]$. Add together the products and store the sum in the output pixel $y\,[i]\,[j]$ that corresponds to the "middle" pixel covered by the kernel (we generally impose that $h\,[i]\,[j]$ is a square matrix with an odd number of rows and columns, so there is an unambiguous "middle".). We only need to consider overlaps between $h^{\star}\,[i]\,[j]$ and $x\,[i]\,[j]$ where the middle element of the kernel covers a pixel in the input image.

EXTRA 20.1: Which Pixel to Draw

Strictly speaking, if we are doing proper 2D convolution, then we should write the sum of products (of multiplying the image pixels by the kernel) into the output pixel corresponding to the "lower right" pixel covered by kernel, and not the "middle" pixel. However, this is not convenient for images, as repeated filtering will make the image details start to drift. Using the middle pixel for the output keeps the image "in place," so to speak.

The `conv2` function for 2D convolution in MATLAB will use the lower right element for output by default, but the middle element is used if passing the option `'same'`, i.e., keep the size of the output the same as the input. The middle element is the default for the `imfilter` function.

We must also decide what to do when we are filtering at the edges of the image. There are different possible conventions, including:

- Treat all "off-image" pixels as having value 0, i.e., $x[i][j] = 0$ beyond the defined image. This is known as **zero-padding**. This is the simplest but could lead to unusual artefacts at the edges of the output image. This is the only option for the `conv2` function for 2D convolution in MATLAB, and the default for the `imfilter` function which is specifically for images.

- Assume that "off-image" elements have the same values as the nearest element along the edge of the image. For example, we could assume $x[0][1] = x[0][0] = x[1][0] = x[1][1]$, i.e., the pixels at the outside of a corner take the value of the corner pixel. This is known as **replicating** the border, and is one of several options for the `imfilter` function.

Now it is appropriate to discuss some common designs for the filter kernel $h[i][j]$. We show several 3×3 examples in Fig. 20.5 (larger kernels have increased sensitivity but are more expensive to compute). You will notice that the elements in a kernel often add up to 0 or 1, such that the filter either removes signal strength to accentuate certain details or maintains signal strength by re-distributing it, respectively. The sample kernels are as follows:

- **Low pass filter** – shown in Fig. 20.5(a). It is equivalent to taking a weighted average around the neighbourhood of a pixel.

- **Blurring filter** – shown in Fig. 20.5(b). It is very similar to the low pass filter, but the sum of the elements adds up to more than one which "washes out" an image more.

- **High pass filter** – shown in Fig. 20.5(c). This kernel accentuates transitions between colours and can be used as a simple **edge detector**. Edge detection is an important task in image processing since it is the first step towards detecting objects in an image.

- **Sobel operator** – shown in Figs. 20.5(d) and (e). This kernel is more effective for detecting edges than a high pass filter, but we need to apply separate kernels for different directions. The X-gradient operator shown is for detecting vertical edges of objects, and the Y-gradient operator shown is for detecting the horizontal edges of objects.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

(a) Low pass filter.

1	1	1
1	1	1
1	1	1

(b) Blurring filter.

0	1	0
1	-4	1
0	1	0

(c) High pass filter.

-1	0	1
-2	0	2
-1	0	1

(d) X gradient Sobel operator.

1	2	1
0	0	0
-1	-2	-1

(e) Y gradient Sobel operator.

Figure 20.5: Examples of filter kernel $h[i][j]$. $h^\star[i][j]$ is the same for (a), (b), and (c), but we need to swap the signs of the elements in (d) and (e).

We now consider an example that filters an image "by hand."

Example 20.3: Image Filtering

Apply the high pass filter shown in Fig. 20.5(c) to the image matrix in Fig. 20.6(a). Assume that the image is zero-padded.

We note that the kernel is the same as its mirror, i.e., $h^\star[i][j] = h[i][j]$ We start by placing the middle element of $h^\star[i][j]$ (-4) over the top left pixel of $x[i][j]$ and add up the products of overlapping elements to determine the output value at this pixel. Going element-by-element, we show the result in Fig. 20.6(b). We see that there are accentuated transitions along what were the edges of the boundary in $x[i][j]$. As expected, this filter kernel can be used as an **edge detector**.

EXTRA 20.2: More on Filtering

Although what we have covered is the foundation of image filtering and could be used for practical applications, there are other ways to filter an image

0	0	0	0	0
0	0	0	0	0
1	1	1	0	0
1	1	1	0	0
1	1	1	0	0

(a) Input image $x\,[i]\,[j]$.

0	0	0	0	0
1	1	1	0	0
-2	-1	-2	1	0
-1	0	-1	1	0
-2	-1	-2	1	0

(b) Output image $y\,[i]\,[j]$.

Figure 20.6: Input and output for Example 20.3.

that we do not cover here. For example, we could define a function-based filter instead of having a fixed kernel $h\,[i]\,[j]$, where we perform mathematical operations besides multiplication. A median-based noise filter will take the median value of the pixels in a kernel.

20.5 Image Filtering in MATLAB

Many programming languages are suitable for processing images, and MATLAB is no different. In fact, MATLAB has a dedicated toolbox (the **Image Processing Toolbox**) for this purpose, although if you have understanding of the mathematical operations (particularly 2D convolution) then you can implement many of the tools yourself or with default MATLAB functions. We have mentioned some MATLAB functions throughout this lesson, and we have already summarised how colours are interpreted, but here we provide a (very) brief guide on filtering images in MATLAB. The MATLAB documentation is an excellent resource for more information on this topic.

As with other data types, MATLAB stores images as arrays. Depending on how the colour information is stored, these could be in two or three dimensions. However, filtering with the Image Processing Toolbox is straightforward no matter what colouring scheme is used.

MATLAB has many pre-loaded images available to study. We have used (and will continue to use) `'peppers.png'`, which you can load into your current MATLAB directory via

```
P = imread('peppers.png');
```

which your workspace will show is a $384 \times 512 \times 3$ array of `unit8`s (i.e., unsigned 8-bit integers for each of red, green, and blue in every pixel). Other interesting examples include `ngc6543a.jpg` and `coins.png`. There doesn't appear to be a centralised list (unless you do a web search), but most example images are used throughout the MATLAB documentation for the Image Processing Toolbox. Of course, you can also import your own images using `imread` and referring to the relative directory where your images are stored. Most image formats are compatible (e.g., tif, png, jpg, bmp). Finally, you could create your own images directly in code, which could be helpful to verify filtering that you perform by hand.

MATLAB provides *many* default filter kernels available (see the MATLAB documentation page on "Image Filtering and Enhancement" for an overview), but we will focus on using the simple kernels that we have already defined in this lesson. These kernels can be implemented by creating a 3×3 matrix with the appropriate elements. To apply a kernel to an image, we can use the default 2D convolution function `conv2` for grayscale images, or the Image Processing Toolbox function `imfilter` more generally.

Example 20.4: Filtering Images in MATLAB

Apply all of the filters described in Fig. 20.5 to the image `'peppers.png'`.

Code to apply the low pass filter is as follows:

```
h = ones(3)/9; % Equivalently h =
    [1/9,1/9,1/9;1/9,1/9,1/9;1/9,1/9,1/9]
P = imread('peppers.png');
output = imfilter(P,h,'conv');
figure; imshow(output)
```

We show the result for this and all other filters in Fig. 20.7. We note that the Sobel operators accentuate edges much more strongly than the high pass filter, but the high pass filter is able to accentuate edges in all directions at once.

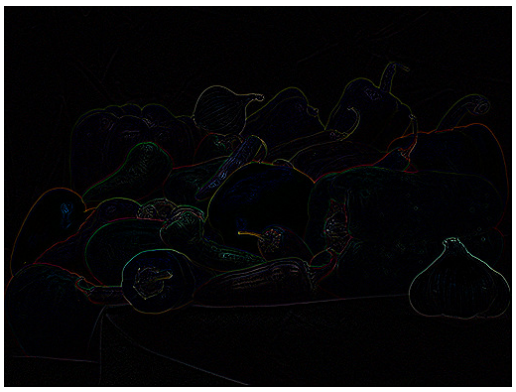
EXTRA 20.3: Using the `imfilter` function

We note that `imfilter` uses correlation and not convolution for filtering by default. We added the `'conv'` argument to force convolution for consistency with our discussion here. In practice the two operations are quite similar;

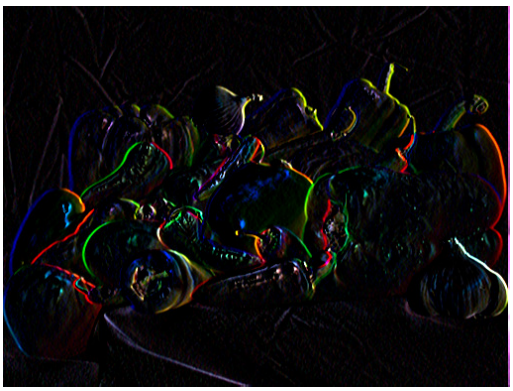
(a) Original image.

(b) Low pass filter.

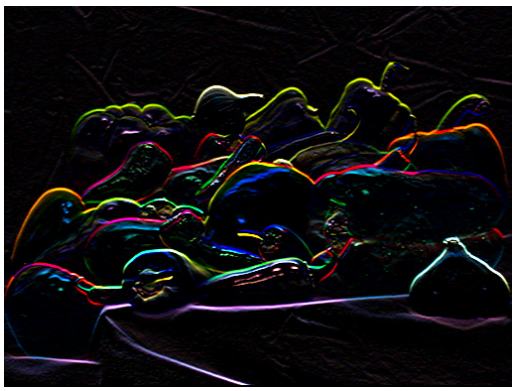
(c) Blurring filter.



(d) High pass filter.



(e) X gradient Sobel.



(f) Y gradient Sobel.

Figure 20.7: Simple filter operations from Example 20.4.

correlation is essentially the same but without the "flipping" of the impulse response. For symmetric kernels like the low pass filter, blurring filter, and

high pass filter there is no difference, but for the Sobel operators it would be necessary to mirror the kernel in order to achieve the same result with correlation.

We also note that, when applied to standard image arrays (typically composed of unsigned 8-bit integers), the `imfilter` function will discard results with negative values and set the values in those pixels to 0.

20.6 Summary

- Images are signals defined over two-dimensional space. We can extend signal processing ideas covered earlier in this module once we account for the extra dimension(s).
- We can apply filters to images and visualise the differences.
- MATLAB has extensive options for image filtering.

20.7 Further Reading

- The MATLAB documentation is an excellent resource, whether online or within the MATLAB help browser.
- Section 9.6 of "Essential MATLAB for engineers and scientists," B. Hahn and D. Valentine, Academic Press, 7th Edition, 2019.

20.8 End-of-Lesson Problems

1. Apply the low pass filter kernel in Fig. 20.5 to the image in Fig. 20.8. What does the filter do? Verify your result in MATLAB.
2. Consider the image in Fig. 20.9. We want to detect the object in the image. What kernel options do we have to detect the edges and what are their strengths? Filter with the possible kernels and comment on the results. Verify your results in MATLAB.

0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0

Figure 20.8: Input image $x\,[i]\,[j]$ for Question 1.

0	0	0	0	0
0	0	1	1	0
0	0	1	1	0
0	0	1	1	0
0	0	0	0	0

Figure 20.9: Input image $x\,[i]\,[j]$ for Question 2.

Part V

Appendices

Appendix A

Mathematical Background

ES3C5: Signal Processing is quite mathematically-focussed and it is assumed that you have mathematical background from previous modules in your degree program. In this appendix we provide a quick overview of several topics. Generally, we will recall concepts as needed when we go through sample problems.

A.1 Partial Fractions

When working with transfer functions, whether in continuous-time or discrete-time, we will often find it helpful to use partial fraction expansion when transforming back to the time domain. We will start with a ratio of polynomials, which will be of the form

$$H(s) = \frac{b_0 s^M + b_1 s^{M-1} + \ldots + b_{M-1} s + b_M}{a_0 s^N + a_1 s^{N-1} + \ldots + a_{N-1} s + a_N}, \tag{A.1}$$

where s is the transform domain variable (in this case the Laplace domain), the numerator is a Mth order polynomial and the denominator is a Nth order polynomial. The bs and as are the constant polynomial coefficients. You should be able to factor polynomials up to second order by hand (as well as factor out additional s terms), and re-write the ratio into the form

$$H(s) = K\frac{b_0 s^M + b_1 s^{M-1} + \ldots + b_{M-1} s + b_M}{(s-p_1)(s-p_2)\ldots(s-p_N)}, \tag{A.2}$$

where K and the ps are constants. Partial fraction expansion re-writes this ratio as a sum of ratios. The key detail is that we want each denominator to be a linear function of s. Thus, we need our transfer function in the form

$$H(s) = \frac{A}{(s-p_1)} + \frac{B}{(s-p_2)} + \frac{C}{(s-p_3)} + \ldots \tag{A.3}$$

and we need to solve for the numerators. To do so, we need to cross-multiply the ratios in Eq. A.3 so that we get a polynomial that we equate with the numerator of Eq. A.1. We can then write a linear equation for each power of s. In other words, we equate

$$A(s-p_2)(s-p_3)\cdots + B(s-p_1)(s-p_3)\cdots + \ldots = \tag{A.4}$$

$$K(b_0 s^M + b_1 s^{M-1} + \ldots + b_{M-1}s + b_M), \tag{A.5}$$

multiply through the terms on the left, and compare powers of s on both sides to solve for A, B, etc. The following is a simple example.

Example A.1: Partial Fraction Expansion

$$\frac{5s+10}{s^2+3s-4} = \frac{5s+10}{(s+4)(s-1)} = \frac{A}{s+4} + \frac{B}{s-1}.$$

We cross multiply to get

$$5s+10 = A(s-1) + B(s+4) = s(A+B) + 4B - A.$$

So we compare powers of s and see that we need to solve

$$A + B = 5$$
$$4B - A = 10$$

We can use substitution to see that $5B = 15 \implies B = 3$ and then $A = 2$, so we can then write

$$\frac{5s+10}{s^2+3s-4} = \frac{2}{s+4} + \frac{3}{s-1}.$$

Next, we consider a more challenging example where there is a repeated root. In such a case, we will find it more useful to leave the corresponding denominator term(s) as higher order (usually second order) functions of s. This approach is also useful when there are complex conjugate roots. The key thing to remember in these cases is that the corresponding numerator term(s) need to be 1 order lower than the denominator terms. The following is an example of this with a repeated root.

Example A.2: Partial Fraction Expansion with Repeated Root

$$\frac{2s-7}{s^3+2s^2+s} = \frac{2s-7}{s(s^2+2s+1)} = \frac{A}{s} + \frac{Bs+C}{s^2+2s+1}.$$

We cross multiply to get

$$2s-7 = A(s^2+2s+1) + s(Bs+C) = s^2(A+B) + s(2A+C) + A.$$

So we compare powers of s and see that we need to solve

$$\begin{aligned} A+B &= 0 \\ 2A+C &= 2 \\ A &= -7 \end{aligned}$$

We immediately have $A-7$, then we can use substitution to find that $C=16$ and $B=7$. So, the final expanded result is

$$\frac{2s-7}{s^3+2s^2+s} = \frac{7s+16}{s^2+2s+1} - \frac{7}{s}.$$

A.2 Integration by Parts

Integration by parts is used when we need to solve the integral with an integrand that is the product of something that is easy to integrate (which we label dv) and something that is easy to differentiate (which we label u). We can then use the property $\int udv = uv - \int vdu$ to solve the integral. The following is an example.

Example A.3: Integration by Parts

Find

$$y = \int_0^\infty xe^{-x}dx,$$

We let $u = x$ and $dv = e^{-x}dx$, so then $du = dx$, $v = -e^{-x}$, and we have

$$\begin{aligned} y &= -xe^{-x}\Big|_{x=0}^{x=\infty} - \int_0^\infty -e^{-x}dx \\ &= -(0-0) - e^{-x}\Big|_{x=0}^{x=\infty} = 0 - (0-1) = 1. \end{aligned}$$

A.3 Euler's Formula

We will occasionally need Euler's formula, which translates between sinusoids and complex exponentials:

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

where $j = \sqrt{-1}$. This formula can be manipulated to create many trigonometric identities, for example to write

$$\cos(x) = \frac{e^{jx} + e^{-jx}}{2} \tag{A.7}$$

$$\sin(x) = \frac{e^{jx} - e^{-jx}}{2j}. \tag{A.8}$$

Example A.4: Using Euler's formula

Show the validity of Eq. A.7. From Eq. A.6 we can write

$$e^{-jx} = \cos(-x) + j\sin(-x) = \cos(x) - j\sin(x), \tag{A.9}$$

because $\cos(-x) = \cos(x)$ and $\sin(-x) = -\sin(x)$. We can then combine Eq. A.6 and Eq. A.9 to write

$$e^{jx} + e^{-jx} = \cos(x) + j\sin(x) + \cos(x) - j\sin(x) = 2\cos(x), \tag{A.10}$$

which can then be re-arranged to write Eq. A.7.

A.4 The Sinc Function

It is quite likely that you are not familiar with the sinc function, but it plays such an important role in signal processing that we define it here. The sinc function is

simply a sine function that is scaled by the argument, i.e.,

$$\text{sinc}(x) = \frac{\sin(x)}{x}, \tag{A.11}$$

and when $x = 0$ we have $\text{sinc}(0) = 1$ (this can be proven from l'Hopital's rule). Thus, the sinc function is a sine that decays away from $x = 0$. A plot of the sinc function evaluated in radians is shown below.

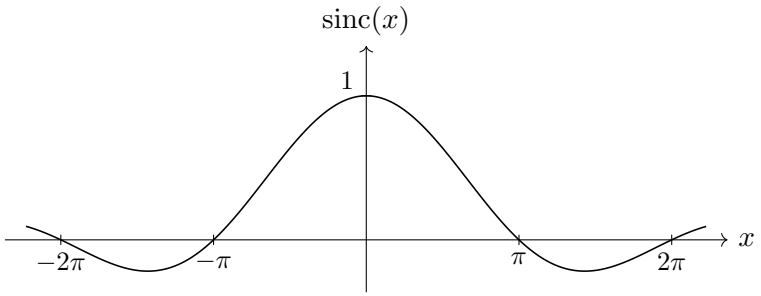


Figure A.1: Sinc function $\text{sinc}(x)$.

A.5 Interval Limits

There will be several occasions where we need to be *very* precise about the definitions of the limits for an interval, so we specify these definitions here:

- A square bracket means that the corresponding value **is included** in the interval.
- A round bracket means that the corresponding value **is not included** in the interval.

Thus the interval $[a, b)$ includes a but not b, the interval (a, b) excludes both a and b, and so on. These conventions are independent of the type of number in the interval (e.g., integers, real numbers, negative numbers) and are consistent with the MATLAB convention for defining arrays with square brackets.

Appendix B

Answers/Hints to End-of-Lesson Problems

B.1 Lesson 2

1. The Laplace transforms are

   (a) $F(s) = \frac{8}{s^3}$

   (b) $F(s) = \frac{4s}{(s^2+4)^2}$

   (c) $F(s) = \frac{2}{(s-2)^2} + \frac{2}{s^3}$

   (d) $F(s) = \frac{s-1}{(s-3)^2+4}$

   (e) $F(s) = \frac{4(s+3)}{(s^2+6s+13)^2}$

2. The inverse Laplace transforms are

   (a) $f(t) = \frac{e^{2t}\sin(3t)}{3}$

   (b) $f(t) = 2e^{-4t} + 3e^{t}$

3. $f(t) = e^{2t} - e^{t}$.

4. This question is a proof.

5. This question is a proof. You can get started by writing $\mathcal{L}\left\{\frac{df'(t)}{dt}\right\} = s\mathcal{L}\{f'(t)\} - f'(0)$ and substitute in the definition of $\mathcal{L}\{f'(t)\}$.

6. i) $y(t) = 1 - e^{-t/\tau}$; ii) $y(t) = t - \tau(1 - e^{-t/\tau})$.

7. Write equations for voltage drops over each loop.

$$H(s) = \frac{R_2}{s^2 LCR_1 + s(L + CR_1R_2) + R_1 + R_2}$$

8. Write 3 equations for currents (current coming out of node to right of R_1 and C_1, current through R_2, and current through C_2). These are all the same current. Combine to solve.

$$H(s) = \frac{(1 + sR_1C_1)(1 + sR_2C_2)}{s^2 R_1C_1R_2C_2 + s(R_1C_1 + R_2C_2 + R_1C_2) + 1}$$

9. A perfect op-amp has no current coming in or out of any of its terminals, and there is no voltage drop across the input terminals. This enables you to write 2 current equations that are equal (current through the capacitor and R_1, and current through R_2).

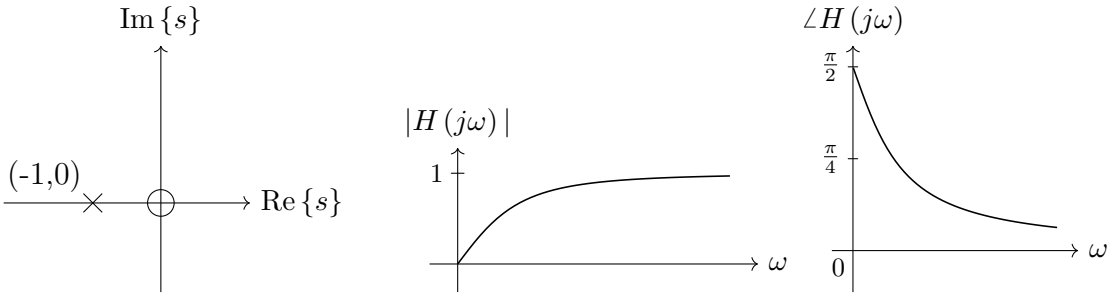
$$H(s) = \frac{-sR_2C}{1 + sR_1C}$$

B.2 Lesson 3

1. There are zeros at $s = \{-25, -\frac{1}{11}\}$; poles are at $s = \{-0.08, -29.56\}$
2. The poles are at $s = -23.33 \pm j11.06$ and the system is stable.
3. The transfer function is $H(s) = \frac{s^2 - s}{s^2 + 2s + 2}$ and the system is stable.
4. (a) $s = 1$, unstable; (b) $s = \pm j\omega$, marginally stable; (c) $s = -1 \pm 3j$, stable.

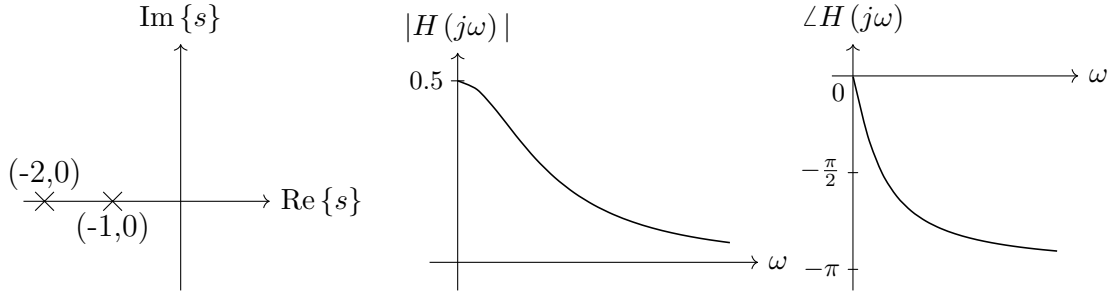
B.3 Lesson 4

1. The plots are as shown in Fig. B.1.
2. The plots are as shown in Fig. B.2.
3. This question is a proof.
4. The plots are as shown in Fig. B.3.
5. For the system to remain stable and have the required gain, the pole must be placed at $s = -2$.



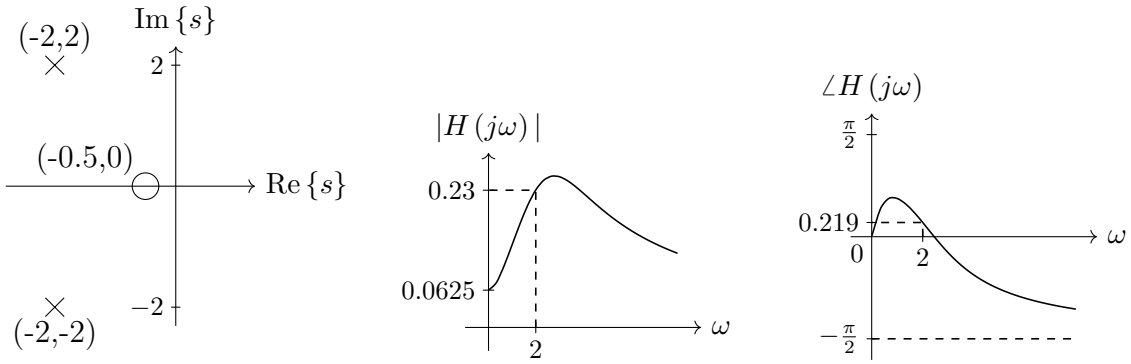
(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

Figure B.1: Plots for Question 4.1.



(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

Figure B.2: Plots for Question 4.2.



(a) Pole-zero plot. (b) Magnitude plot. (c) Phase plot.

Figure B.3: Plots for Question 4.4.

B.4 Lesson 5

1. The ideal low pass filter frequency response is unrealisable because a rectangle function in the frequency domain is a sinc function in the time domain. The sinc function is defined for all time, including $t < 0$, so the ideal filter must respond to input before the input is applied, which is impossible.

2. $G = -42.3\,\text{dB}$.

3. Order $N = 3$. Cut-off frequency $10.94\,\frac{\text{rad}}{\text{s}} \leq \omega_c \leq 13.95\,\frac{\text{rad}}{\text{s}}$.

4. Order $N = 5$. Choose cut-off frequency between $114.47\,\text{Hz} \leq f \leq 125.31\,\text{Hz}$.

B.5 Lesson 6

1. The signal components at the output of the filter will have frequencies 0 Hz, 3.33 Hz, and 6.67 Hz.

2. There will be 2 sinusoidal terms at the output with frequencies $1.6\pi/\tau$ and $2.4\pi/\tau$.

3. Amplitude will be $0.209A$.

B.6 Lesson 7

1. The following function will find the system output for any input frequency `omega`:

```
function y = analogue_output_example(omega)
% Find output to my low pass filter
R = 2; C = 1; L = 2;
% Create symbolic variables
syms t s
% Create symbolic functions for input and system
x = sin(omega*t);
H = R/(s^2*L*C*R + s*(L+C*R^2)+2*R);
% Convert input to Laplace, solve, and convert output to time
X = laplace(x); Y = H*X; y = ilaplace(Y);
```

   Calling this function with frequency `omega=10` returns

```
y = (5*exp(-(3*t)/4)*(cos((7^(1/2)*t)/4) +
    (267*7^(1/2)*sin((7^(1/2)*t)/4))/7))/6684 -
    (11*sin(10*t))/2228 - (5*cos(10*t))/6684
```

Note that symbolic math will leave real numbers as fractions whenever possible.

2. The following function should work and return both the upper bound and lower bound cutoff frequencies. All of the equations are implementations of those in Lesson 5.

```
function [cutoff, order] = butterworth_design(gainPass,
    gainStop, radPass, radStop)
% gainPass - target pass band gain in dB
% gainStop - target stop band gain in dB

order =
    ceil(log10((10^(-gainStop/10)-1)/(10^(-gainPass/10)-1))...
    /2/log10(radStop/radPass));

cutoff(1) = radPass/(10^(-gainPass/10)-1)^(1/2/order);
cutoff(2) = radStop/(10^(-gainStop/10)-1)^(1/2/order);
```

B.7 Lesson 8

1. Minimum sampling frequency is $f_s = 30\,\text{Hz}$.
2. This question is a proof. You need to show that $f_2[n] = \cos(n\omega_1 T_s) = f_1[n]$.
3. Yes, the sinusoid $\cos(-50\pi t) = \cos(50\pi t)$ is a 25 Hz sinusoid that would have $f_1(t)$ as an alias.
4. Largest error is $\frac{1}{2(2^W-1)}$.
5. Resolution is 0.024 %; dynamic range is 72.25 dB.

B.8 Lesson 9

1. The Z-transforms are
   (a) $F_1(z) = 1 + z^{-1}$

(b) $F_2(z) = \frac{1}{1-z^{-1}}$

(c) $F_3(z) = \frac{1}{1-0.5z^{-1}}$

(d) $F_4(z) = \frac{2-z^{-2}}{1-z^{-1}}$

2. The sequences are

   (a) $f_1[n] = \{1, 1, 1, 1, 1\}$

   (b) $f_2[n] = \{1, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, \ldots\}$

3. This question is a proof. Take the Z-transform of each time-domain sequence.

4. This question can be solved two ways, but it is faster to solve in the z-domain. $y[n] = \{1, 3, 3, 5, 3, 7, 4, 3, 3, 0, 1\}$.

5. $Y(z) = 6 + z^{-1} - 2z^{-2}$.

6. $H(z) = \frac{1+z}{2z}$.

7. $y[n] = y[n-1] + T_s x[n]$, $H(z) = \frac{T_s z}{z-1}$.

B.9 Lesson 10

1. $H(z) = \frac{z^2}{z^2-0.6z+0.5}$. There are poles at $z = 0.3 \pm 0.64j$ and two zeros at $z = 0$. The system is stable.

2. There is one pole at $z = 0$, so the system is stable.

3. There is one pole at $z = 1$, so the system is conditionally stable.

4. The transfer function is $H(z) = \frac{z^2-0.3z-1.3}{z^3-0.5z^2+z-0.5}$. There are poles on the unit circle so the system is conditionally stable.

B.10 Lesson 11

1. $H(z) = \frac{z^2}{(z+0.9)^2}$. When $\Omega = \pi$, the phase is 0.

2. The minimum frequency response magnitude is 0 at $\Omega = 0$ and $\Omega = \pi$. The maximum frequency response magnitude is at $\Omega = \frac{\pi}{2}$.

3. Pole-zero plot is as shown in Fig. B.4. Frequency response is as shown in Fig. B.5.

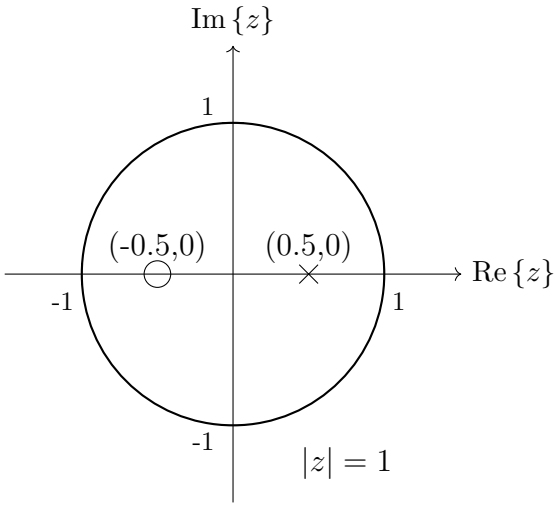
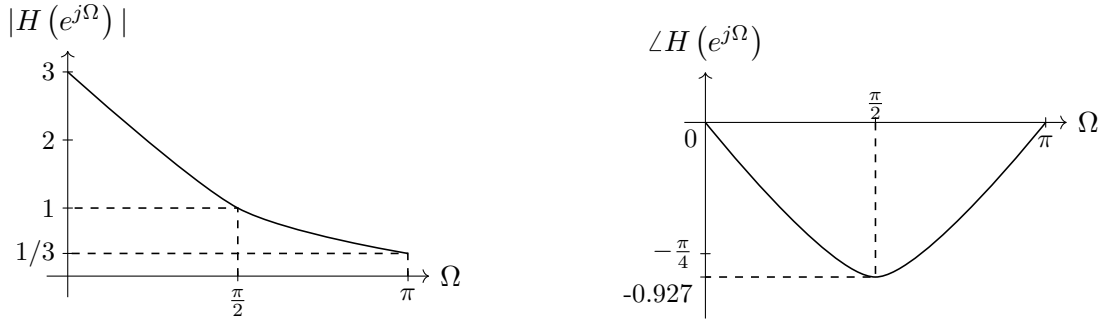


Figure B.4: Pole-zero plot for Question 11.3.



(a) Magnitude plot.

(b) Phase plot.

Figure B.5: Sketch of frequency response for Question 11.3.

B.11 Lesson 12

1. $y[n] = x[n] - 2x[n-1] + x[n-2]$; $H(z) = (z^2 - 2z + 1)/z^2$; this is an FIR high pass filter. The filter in Fig. 12.6 has a multiplier of 2 instead of -2 and it is a low pass filter.

2. This is an IIR filter. The frequency response has a magnitude of 0.67 at $\Omega = 0$, 0.89 at $\Omega = \pi/2$, and 2 at $\Omega = \pi$. It might be best described as high pass as the highest gain is at high frequencies, but it is a poor high pass filter because low frequencies are not attenuated very much.

3. This is an FIR filter (no poles are shown so they are assumed to be at the origin). The frequency response has a magnitude of 1 at $\Omega = 0$, 0 at $\Omega = \pi/3$,

and 3 at $\Omega = \pi$. This is a band stop filter. Although the pass bands have different gains, the stop band is implemented effectively.

4. This filter has 2 zeros at $z = -0.5 \pm j0.5$ and poles at $z = 0.5 \pm j0.5$. It is a stable 2nd order low pass filter.

5. The magnitude of the frequency response is 0.6154, 2.2188, and 8 at the frequencies $\Omega = \{0, \pi/2, \pi\}$, respectively. Hence, this is a high pass filter. The transfer function is

$$H(z) = \frac{1 - 2z^{-1} + 2z^{-2}}{1 + 0.5z^{-1} + 0.125z^{-2}},$$

and the LSI implementation is given in Fig. B.6.

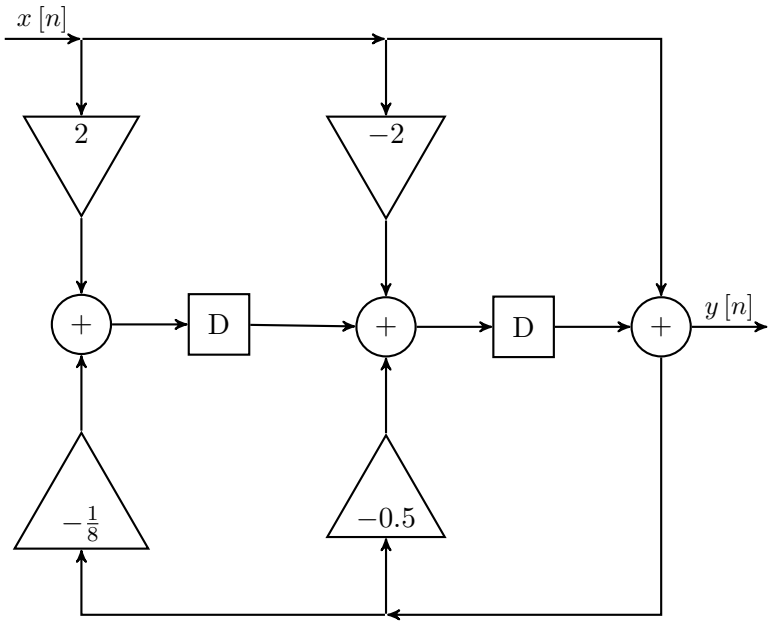
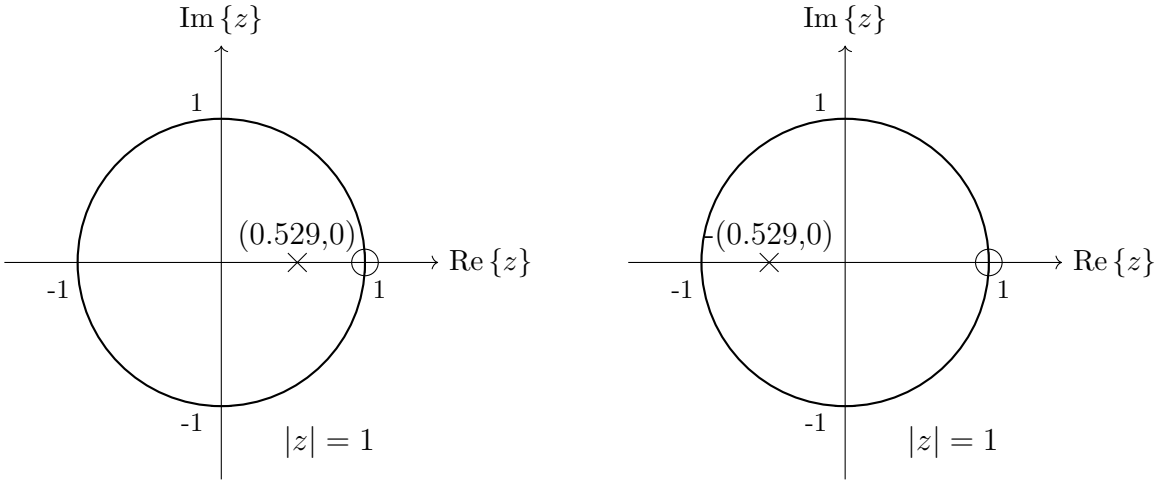


Figure B.6: LSI implementation for Question 12.5.

6. The difference equation is $y[n] = x[n] + y[n-1] - 0.5y[n-2]$. Using the tabular method we can find that the impulse response is $h[n] = \{1, 1, 0.5, 0.-0.25, -0.25, -0.125, 0, \ldots\}$. The response is oscillating but decaying, so we infer that it is stable. This can be confirmed by finding that the poles are at $z = 0.5 \pm j0.5$, which are inside the unit circle.

7. Since the filter is stable and first-order, then there can only be one pole and it must be placed within the unit circle. A single pole must be on the real axis

for realisability. The filter must have a zero placed at $(1, 0)$ in order to filter out constant magnitude input signals. Here, 250 Hz signals correspond to a digital radial frequency of $\Omega = \pi/2$. In order to have a gain of 0 dB (i.e., 1) at $\Omega = \pi/2$, then the pole must be placed 1.1314 from $(0, 1)$. There are 2 valid locations, i.e., $(-0.529, 0)$ and $(0.529, 0)$. The corresponding pole-zero plots are included in Fig. B.7.



(a) Solution 1 (b) Solution 2

Figure B.7: Valid pole-zero plots for Question 12.7.

8. Since the filter is first-order and FIR, then there can be at most one pole and one zero and the pole must be at the origin (which also satisfies stability). Ideal stopband gain (0) must be at $\Omega = 2\pi \times 500/1000 = \pi$, so we place the zero at $z = (-1, 0)$. Ideal passband band (0 dB, i.e., 1) must be at $\Omega = 0$, where the magnitude of the zero vector is 2. Thus we need $K = 0.5$. The corresponding pole-zero plot is in Fig. B.8.

B.12 Lesson 13

1. The filters are:

$$h_{\text{rect}}[n] = \{0.159, 0.225, 0.25, 0.225, 0.159\}, 0 \leq n \leq 4$$
$$h_{\text{Hamming}}[n] = \{0.013, 0.122, 0.25, 0.122, 0.013\}, 0 \leq n \leq 4.$$

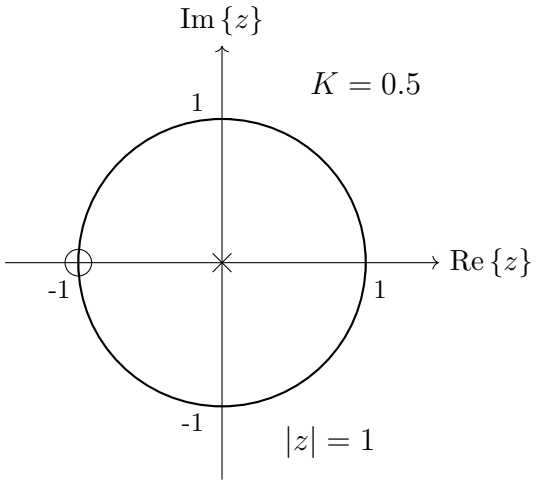


Figure B.8: Valid pole-zero plot for Question 12.8.

The *unshifted* spectra are:

$$H_{\text{rect}}\left(e^{j\Omega}\right) = 0.25 + 0.45\cos(\Omega) + 0.318\cos(2\Omega)$$
$$H_{\text{Hamming}}\left(e^{j\Omega}\right) = 0.25 + 0.244\cos(\Omega) + 0.026\cos(2\Omega).$$

We plot the FIR impulse responses and magnitude spectra in Fig. B.9.

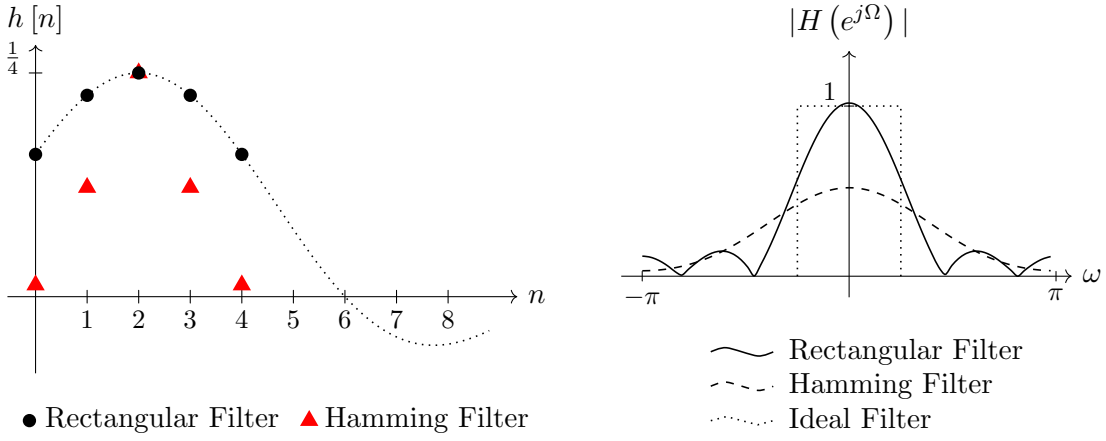


Figure B.9: Responses of FIR filters designed in Question 13.1.

2. (a) $\Omega_{\text{p}} = \frac{\pi}{25}\frac{\text{rad}}{\text{sample}}$, $\Omega_{\text{s}} = \frac{\pi}{10}\frac{\text{rad}}{\text{sample}}$.

   (b) Need $L = 105$.

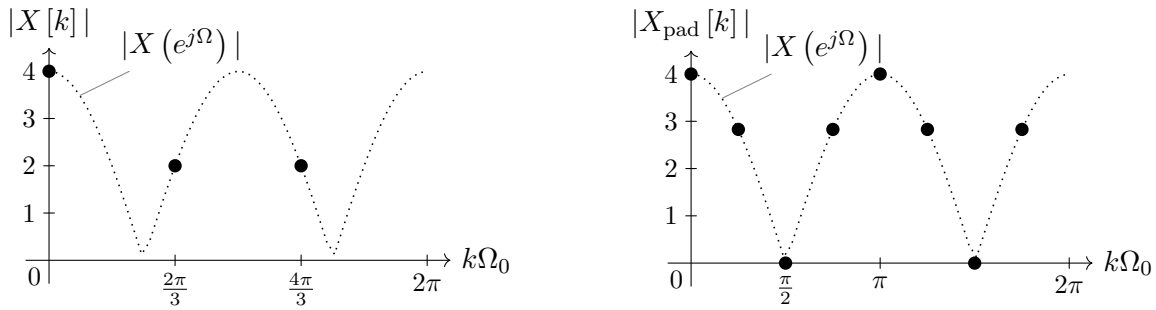
(c) The stop band gain of the rectangular filter is only $G_s = -20.9\,\text{dB}$, which is less than the target stop band gain, so such a filter cannot meet the design specification.

B.13 Lesson 14

1. From calculations we find that

$$\begin{aligned} X[k] &= \{4, 1+j\sqrt{3}, 1-j\sqrt{3}\}, 0 \le n \le 2 \\ X\left(e^{j\Omega}\right) &= 4e^{-j\Omega}\cos(\Omega) \\ X_{\text{pad}}[k] &= \{4, 2-j2, 0, 2+j2, 4, 2-j2, 0, 2+j2\}. \end{aligned}$$

We plot the magnitudes in Fig. B.10. We see that the zero-padded DFT captures much more of the detail in the DTFT.



(a) Magnitude spectra.

(b) Zero-padded magnitude spectra.

Figure B.10: Plots for Question 14.1.

2. To sample the sinusoids with frequencies $f_s/16$ and $f_s/4$, multiply the sinusoid argument in Example 14.3 by 4 and 16, respectively. The magnitude spectra of their sum are shown in Fig. B.11.

B.14 Lesson 15

1. Code to find and plot the outputs is as follows:

```
syms n z

h = 0.2*ones(1,5); % Impulse response
i = 0:39;          % Input signal sample index
```

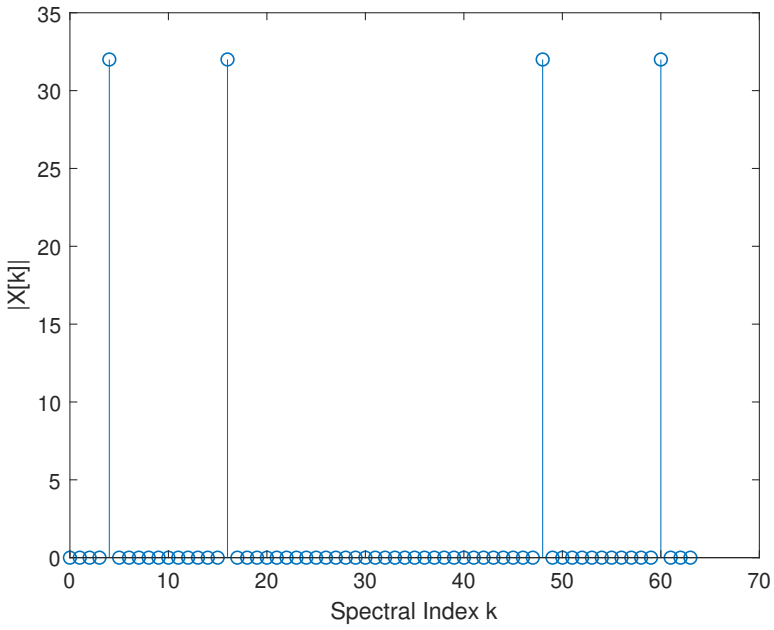


Figure B.11: Magnitude spectrum for Question 14.2

```
x = cos(pi*i/4);  % Input signal
L = length(i) + length(h) -1; % Length of conv output

xSym = cos(pi*n/4); % Input signal in symbolic math
HSym = 0.2*(1 + z^-1 + z^-2 + z^-3 + z^-4); % Transfer function
    in symbolic math

y1 = conv(x,h); % Output using conv function
X = ztrans(xSym);
Y = X*HSym;
y2Sym = iztrans(Y); % Output in symbolic math
y2 = double(subs(y2Sym,0:(L-1))); % Symbolic math output
    converted to numeric values
y3 = filter(h,1,x); % Output using filter function

figure; hold on;
subplot(3,1,1)
stem(0:(L-1),y1,'k')
legend('conv')
subplot(3,1,2)
stem(0:(L-1),y2,'b')
legend('symbolic')
```

```
subplot(3,1,3)
stem(0:(length(i)-1),y3,'r')
legend('filter')
```

We plot the resulting figure in Fig. B.12. We see that the results look nearly identical, except that `filter` does not evaluate the final 4 time steps (i.e., those beyond the existence of the input) and the symbolic approach behaves as if the input continues indefinitely. In each case the filter output is a sinusoid with about half the amplitude of the input signal.

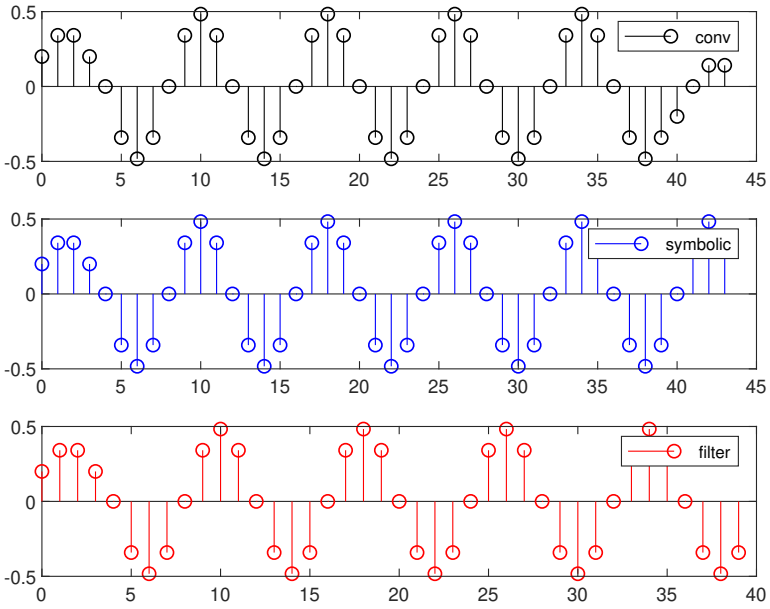


Figure B.12: Output for Question 15.1

2. Code to design the filters and plot their responses is as follows (we set the filter design to not scale to make sure we can directly compare with our design by hand):

```
bRect = fir1(4, 0.25, 'low', rectwin(5), 'noscale'); %
    Rectangular filter
bHamm = fir1(4, 0.25, 'low', hamming(5), 'noscale'); % Hamming
    filter

figure; hold on;
subplot(2,1,1)
stem(0:4, bRect);
xlabel('n')
```

```
ylabel('h[n]')
title('Rectangular Window Impulse Response')
subplot(2,1,2)
stem(0:4, bHamm);
xlabel('n')
ylabel('h[n]')
title('Hamming Window Impulse Response')

figure
freqz(bRect,1)
title('Rectangular Window Filter')

figure;
freqz(bHamm,1)
title('Hamming Window Filter')
```

We plot the resulting figures in Figs. B.13-B.15. We include several data points in the magnitude responses (that we can convert to linear form) to help verify the design in Question 12.1.

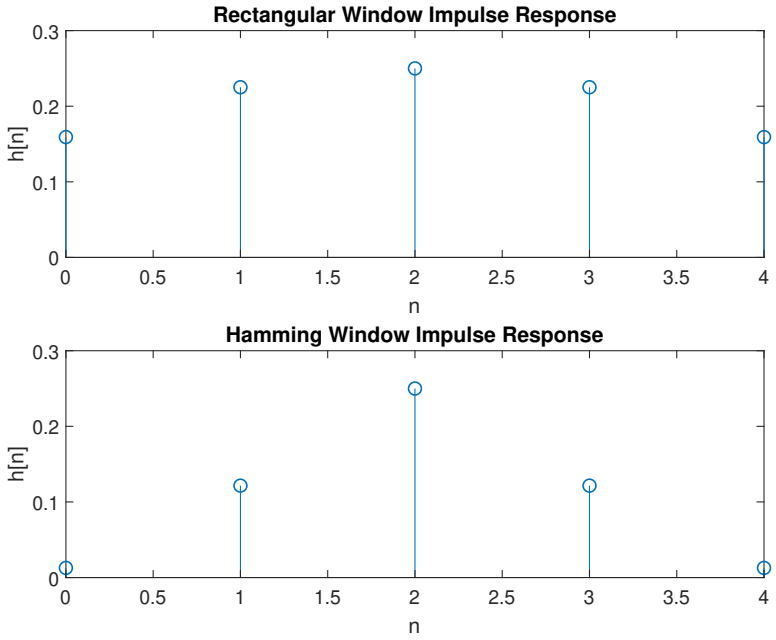


Figure B.13: Impulse responses for Question 15.2

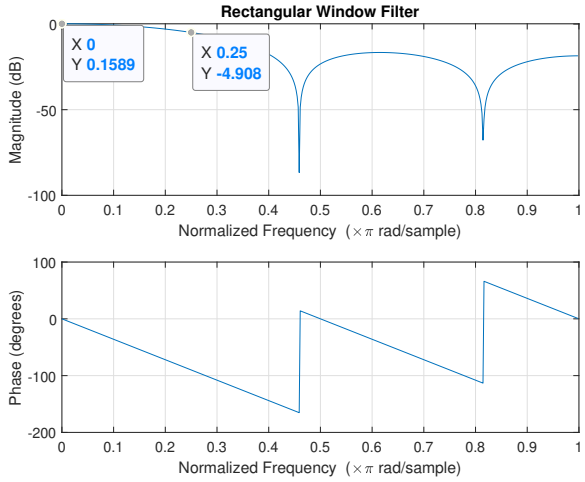


Figure B.14: Rectangular frequency response for Question 15.2

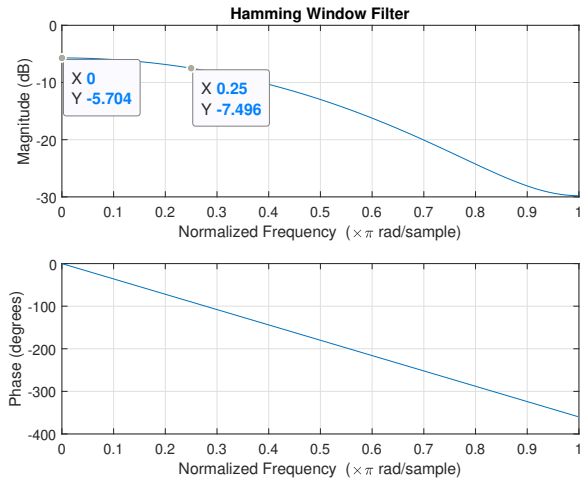


Figure B.15: Hamming frequency response for Question 15.2

B.15 Lesson 17

1. If $x_{\min} = x_{\max}$, then the random variable is deterministic and not actually a random variable.

2. This question is a proof.

3. (a) Use integration by parts to show that $k = 1$.

   (b) 0.264.

   (c) $E[X] = 2$.

(d) $E[X^2] = 6$ and $\sigma_X^2 = 2$.

(e) The most probable value occurs when $\frac{dp(x)}{dx} = 0$. It occurs when $X = 1$.

(f) 0.594.

B.16 Lesson 18

1. The observation matrix is

$$\mathbf{\Theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0.5 & 0.25 & 0.125 \\ 1 & 1.0 & 1.0 & 1.0 \\ 1 & 1.5 & 2.25 & 3.375 \\ 1 & 2.0 & 4 & 8.0 \\ 1 & 2.5 & 6.25 & 15.625 \\ 1 & 3.0 & 9 & 27 \end{bmatrix}.$$

2. The observation matrix is

$$\mathbf{\Theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

3. The weight vector should be

$$\vec{W} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{bmatrix}.$$

4. Code to generate the Poisson realisations and estimate the mean using a maximum likelihood estimator is as follows:

```
lambda = 5; % Poisson mean
y = poissrnd(lambda,1,100); % Generate random numbers from
    Poisson distribution
lambdaEst = mle(y, 'distribution', 'poiss'); % Max. Likelihood
    estimate of lambda
```

B.17 Lesson 19

1. $R_{X_1X_2}[k] = \{-7.33, -8.00, -11.50, -7.33, -9.00, -18\}$. The highest correlation occurs when $k = 5$ because these two signals differ by a factor of -2 except that the second signal is shifted by 5 (which in this case is also equal to $5 - N = -1$).

2. $R_{XX}[k] = \{3.67, -0.50, 0.43, -2.00, 0.80, 0.75, 0.33, -0.50, 2.00\}$. The mean square value is $R_{XX}[0] = 3.67$.

B.18 Lesson 20

1. The filter output is shown in Fig. B.16. This low pass filter kernel has smoothed the abrupt transitions between 0s and 1s.

0.22	0.33	0.33	0.33	0.22
0.33	0.44	0.56	0.44	0.33
0.33	0.56	0.44	0.56	0.33
0.33	0.44	0.56	0.44	0.33
0.22	0.33	0.33	0.33	0.22

Figure B.16: Output image $y[i][j]$ for Question 20.1.

2. To detect the edges we could use a high pass filter or Sobel operators. The high pass filter only needs to be applied once, but Sobel operators will accentuate the edges more clearly. We show the output to the high pass filter and the X gradient Sobel operator in Fig. B.17. From the high pass filter we

see transitions from negative to positive values along all 4 edges. The Sobel filter output only has negative to positive transitions along the right edge and positive to negative transitions along the left edge, but they are much stronger transitions.

0	0	1	1	0
0	1	-2	-2	1
0	1	-1	-1	1
0	1	-2	-2	1
0	0	1	1	0

(a) High pass filter output.

0	-1	-1	1	1
0	-3	-3	3	3
0	-4	-4	4	4
0	-3	-3	3	3
0	-1	-1	1	1

(b) X gradient Sobel filter output.

Figure B.17: Output images $y[i][j]$ for Question 20.2.