

DISCRETE-TIME SIGNAL PROCESSING



Carl Greco
Arkansas Tech University

Discrete-Time Signal Processing

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (<https://LibreTexts.org>) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden born from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by [NICE CXone Expert](#) and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptations contact info@LibreTexts.org or visit our main website at <https://LibreTexts.org>.

The LibreTexts name, logo, and book covers are trademarked by LibreTexts, Inc. (a 501(c)(3) not-for-profit organization) and protected against unauthorized use.

This text was compiled on 08/26/2026

TABLE OF CONTENTS

Licensing

1: Introduction to Digital Signal Processing

2: The Sampling Theorem

3: Amplitude Quantization

4: Discrete -Time Signals and Systems

5: Z-Transform and Discrete Time System Design

5.1: Z-Transform

5.2: Common Z-Transforms

5.3: Properties of the Z-Transform

5.4: Inverse Z-Transform

5.5: Poles and Zeros in the Z-Plane

5.6: Region of Convergence for the Z-Transform

5.7: Rational Functions and the Z-Transform

5.8: Difference Equations

5.9: Discrete Time Filter Design

6: Discrete Time Fourier Series (DTFS)

6.1: Discrete Time Periodic Signals

6.2: Discrete Time Fourier Series (DTFS)

6.3: Common Discrete Fourier Series

6.4: Properties of the DTFS

6.5: Discrete Time Circular Convolution and the DTFS

7: Discrete -Time Fourier Transform (DTFT)

8: Discrete Fourier Transforms (DFT)

9: DFT - Computational Complexity

10: Fast Fourier Transform (FFT)

11: Spectrograms

12: Discrete-Time Systems

13: Discrete-Time Systems in the Time-Domain

14: Discrete -Time Systems in the Frequency Domain

15: Filtering in the Frequency Domain

16: Efficiency of Frequency - Domain Filtering

17: Discrete -Time Filtering of Analog Signals

18: Digital Signal Processing Problems

Index

Index

Glossary

Detailed Licensing

Detailed Licensing

Licensing

A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.

This page titled [Licensing](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

1: Introduction to Digital Signal Processing

Learning Objectives

- Segue for DSP chapter.

Not only do we have analog signals --- signals that are real- or complex-valued functions of a continuous variable such as time or space --- we can define **discrete/digital** ones as well. Digital signals are **sequences**, functions defined only for the integers. We thus use the notation $s(n)$ to denote a discrete-time one-dimensional signal such as a digital music recording and $s(m,n)$ for a discrete-"time" two-dimensional signal like a photo taken with a digital camera. Sequences are fundamentally different than continuous-time signals. For example, continuity has no meaning for sequences.

Despite such fundamental differences, the theory underlying digital signal processing mirrors that of analog signals: Fourier transforms, linear filtering and linear systems parallel what previous chapters described. These similarities make it easy to understand the definitions and why we need them, but the similarities should not be construed as "analog wannabes." We will discover that digital signal processing is **not** an approximation to analog processing. We must explicitly worry about the fidelity of converting analog signals into digital ones. The music stored on CDs, the speech sent over digital cellular telephones, and the video carried by digital television all evidence that analog signals can be accurately converted to digital ones and back again.

The key reason why digital signal processing systems have a technological advantage today is the **computer**: computations, like the Fourier transform, can be performed quickly enough to be calculated as the signal is produced, and programmability means that the signal processing system can be easily changed. This flexibility has obvious appeal and has been widely accepted in the marketplace. Programmability means that we can perform signal processing operations impossible with analog systems (circuits). We will also discover that digital systems enjoy an **algorithmic** advantage that contributes to rapid processing speeds: Computations can be restructured in non-obvious ways to speed the processing. This flexibility comes at a price, a consequence of how computers work. How do computers perform signal processing?

Note

Taking a systems viewpoint for the moment, a system that produces its output as rapidly as the input arises is said to be a **real-time** system. All analog systems operate in real-time; digital ones that depend on a computer to perform system computations may or may not work in real-time. Clearly, we need real-time signal processing systems. Only recently have computers become fast enough to meet real-time requirements while performing non-trivial signal processing.

Digital signal processing has been applied to many different areas including medicine, telecommunication, military, and scientific to mention a few. For example, the number of sun spots have been observed since the mid 1700's and analyzed to determine patterns of the frequency of their occurrence.

Example

The following is an example of digital signal processing applied to scientific data analysis. The sunspot data was obtained from the [Solar Influences Data Analysis Center \(SILSO\)](#) (opens in new window) CC BY-NC 4.0. Also, see the [National Solar Observatory](#) (NSO) for more information solar activity.

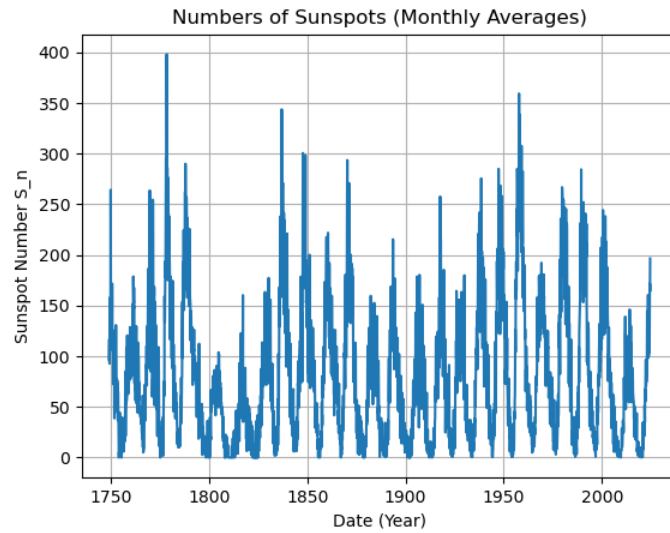


Figure 1.1: Sunspot Monthly Average Data. Late 1700's to early 1800's is know as the Dalton Minimum.

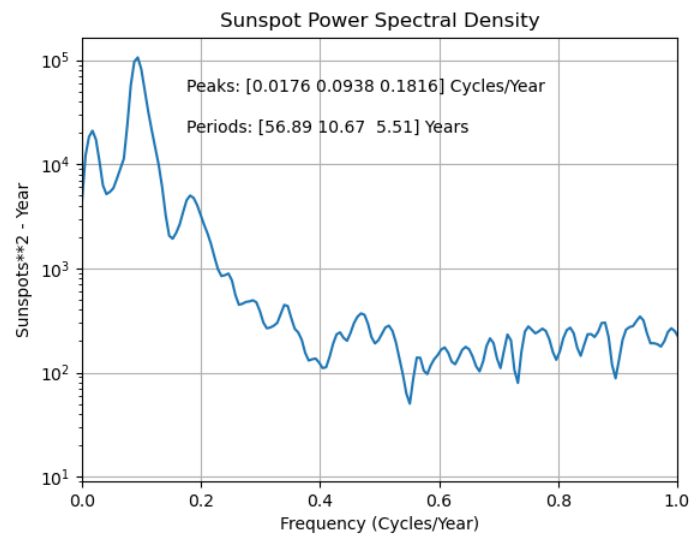


Figure 1.2: Power Spectral Density of Sunspot Data.

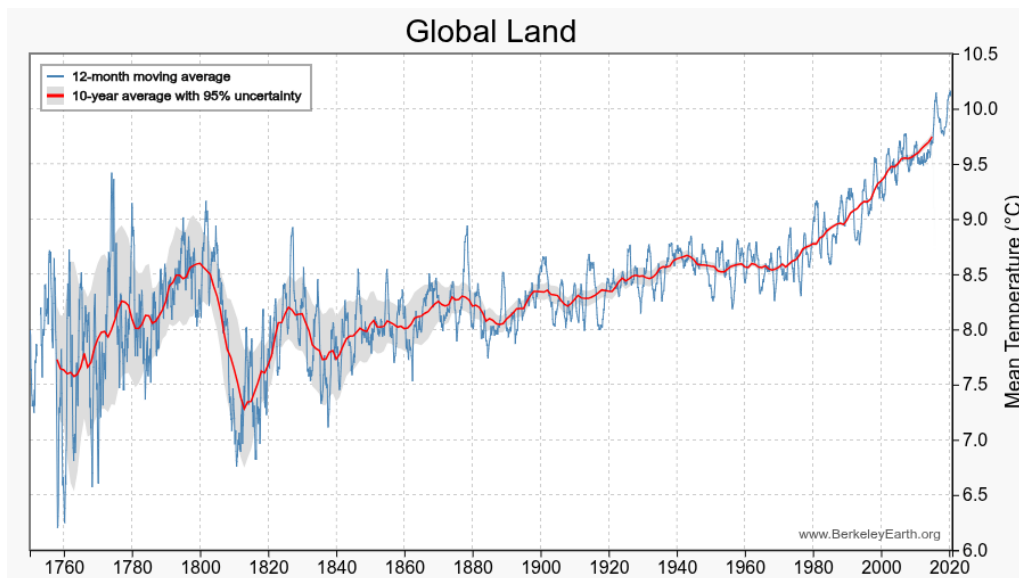


Figure 1.3: Global average land temperature. (Berkeley Earth [Global temperature history](https://berkeleyearth.org) [berkeleyearth.org])

The average number of sunspots that appeared on a monthly basis is displayed in the above graph, and the frequency analysis of these sunspots was obtained from power spectral density (PSD) analysis of this sunspot data as shown in the second graph. Not surprisingly, there was a peak in the PSD analysis at 0.0938 cycles/year for a period of 10.67 years. This corresponds to the observed cyclic variation of sunspots in the above graph. The PSD analysis also reveals that there are two additional peaks: one at 0.0176 cycles/year with a period of 56.89 years and another at 0.1816 cycles/year for a 5.51 year period. The ~57 year variation may be observed as the waxing and waning in the peak heights that occur every ~10.7 years corresponding to the completion of the sun's magnetic pole reversal. The higher frequency peak that occurs at ~5.5 year cycle, is not noticeable in the sunspot data in the above figure which is consistent with its diminished peak magnitude in the PSD analysis relative to the other two peaks.

Python Code

```
'''
Sunspot Number - Power Spectral Density
Download Sunspot Data and Estimate Power Spectral Density
Solar Influences Data Analysis Center (WDC-SILSO) [https://www.sidc.be/SILSO/home]
Royal Observatory of Belgium, Brussels
% Input data Monthly mean total sunspot number (1/1749 - 8/2024)
Columns separated by one or more spaces
    Column 1-2: Gregorian calendar date
        - Year
        - Month
    Column 3: Date in fraction of year.
    Column 4: Monthly mean total sunspot number.
    Column 5: Monthly mean standard deviation of the input sunspot numbers.
    Column 6: Number of observations used to compute the monthly mean total
              sunspot number.
    Column 7: Definitive/provisional marker. A "1" indicates that the value is
              definitive. A '0' indicates that the value is still provisional.
'''

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
import sys

def load_csv_with_fallback(primary_path, fallback_path, **kwargs):
    """Attempts to read a CSV file from a primary location.
```

Falls back to a secondary location if the primary fails.

Parameters:

- primary_path (str): The main file path or URL to try first.
- fallback_path (str): The backup file path or URL.
- kwargs: Additional arguments passed directly to pd.read_csv().

Returns:

- pd.DataFrame: The loaded DataFrame if successful.

Raises:

- FileNotFoundError: If both locations fail to load.
- ```
"""
```

```
1. Try the primary location
```

```
try:
```

```
 print(f"Attempting to read primary file: {primary_path}")
 df = pd.read_csv(primary_path, **kwargs)
 print("Successfully loaded from primary location.")
 return df
```

```
except (FileNotFoundError, Exception) as e:
```

```
 print(f"Primary location failed. Reason: {e}")
 print("-" * 40)
```

```
2. Try the fallback location
```

```
try:
```

```
 print(f"Attempting to read fallback file: {fallback_path}")
 df = pd.read_csv(fallback_path, **kwargs)
 print("Successfully loaded from fallback location.")
 return df
```

```
except (FileNotFoundError, Exception) as fallback_e:
```

```
 print(f"Fallback location also failed. Reason: {fallback_e}")
 # Raise an error if both attempts fail
 raise FileNotFoundError(
 f"Critical: Could not load CSV from either \n"
 f"Primary: '{primary_path}' or \n"
 f"Fallback: '{fallback_path}'"
) from fallback_e
```

```
Attempt to read the local file first
```

```
'''
```

```
 Read Sunspot Data from local file or online source
```

- File format: csv or txt
- Columns separated by one or more spaces or semicolon
- Use sep=';' for csv file or sep=r'\s+' for txt file with columns separated by one or
- If the file is not found, it will try to read from the online source
- If both fail, it will print an error message and set df to None

```
'''
```

```
filename_csv = "https://commons.libretexts.org/permalink/IKTzPvY930/f9abe71d-b478-42d9-a22
```

```
filename_url = "https://www.sidc.be/SILSO/DATA/SN_m_tot_V2.0.csv" # Online source
```

```
try:
```

```
 # You can pass standard read_csv parameters like sep, encoding, etc.
 df = load_csv_with_fallback(filename_csv, filename_url, sep=',|;', index_col=False)
 # Do something with the data
 print("\nData Preview:")
 print(df)
```

```
except Exception as final_error:
```

```
 print(f"\nScript execution failed: {final_error}")
```

```
Design the FIR lowpass filter & filter data
Order of the filter
Sampling frequency (cycles/year)
fs = 12 # Once per month (12 times per year)
Cutoff frequency (cycles/year)
fc = 0.4 # Cutoff frequency cycles/year
order = 12
M1 = order + 1
Lowpass Filter the Data
Design the FIR lowpass filter using the window method
h1 = signal.firwin(M1, fc, window='hann', fs = fs)
y_filtered = signal.filtfilt(h1,[1],df["Sunspots"])

Display Graph of Sunspots
plt.figure(figsize=(12,6))
plt.plot(df["Date"],df["Sunspots"], label='Monthly Averages')
plt.plot(df["Date"], y_filtered, 'r-', linewidth=1.5, label=f'Lowpass Filtered, fc = {fc}')
plt.xlim(df["Date"].min(), df["Date"].max())
plt.grid()
plt.xlabel("Date (Year)")
plt.ylabel("Sunspot Number S_n")
plt.title("Numbers of Sunspots 1749 to Present (Monthly Averages), Lowpass Filtered with R")
plt.legend()

Power Spectral Density Estimation
NFFT = 2048
nyears = 100
f, Pxx = signal.welch(df["Sunspots"].to_numpy(),fs=fs,window='hann',nperseg=12*nyears, \
 noverlap=12*nyears//2,nfft=NFFT,detrend='linear',return_onesided=True,scaling='density')

plt.figure(figsize=(10,6))
plt.semilogy(f, Pxx)
plt.xlim(0,1)
plt.grid()
plt.xlabel('Frequency (Cycles/Year)')
plt.ylabel('Sunspots**2 - Year')
plt.title('Sunspot Power Spectral Density')
EndRange = int(0.25*NFFT/12)
peaks, _ = signal.find_peaks(Pxx[0:EndRange])
plt.text(0.2,5e4,rf'Peaks: {np.around(peaks*fs/NFFT, decimals=4)} Cycles/Year',fontsize=16)
plt.text(0.2,2e4,rf'Periods: {np.around(np.reciprocal(peaks*fs/NFFT), decimals=2)} Years',
plt.show()
```

---

This page titled [1: Introduction to Digital Signal Processing](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [5.1: Introduction to Digital Signal Processing](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 2: The Sampling Theorem

### Learning Objectives

- Converting between a signal and numbers.

### Analog-to-Digital Conversion

Because of the way computers are organized, signal must be represented by a finite number of bytes. This restriction means that **both** the time axis and the amplitude axis must be **quantized**: They must each be a multiple of the integers.<sup>1</sup> Quite surprisingly, the Sampling Theorem allows us to quantize the time axis **without error** for some signals. The signals that can be sampled without introducing error are interesting, and as described in the next section, we can make a signal "samplable" by filtering. In contrast, no one has found a way of performing the amplitude quantization step without introducing an unrecoverable error. Thus, a signal's value can no longer be any real number. Signals processed by digital computers must be **discrete-valued**: their values must be proportional to the integers. Consequently, **analog-to-digital conversion introduces error**.

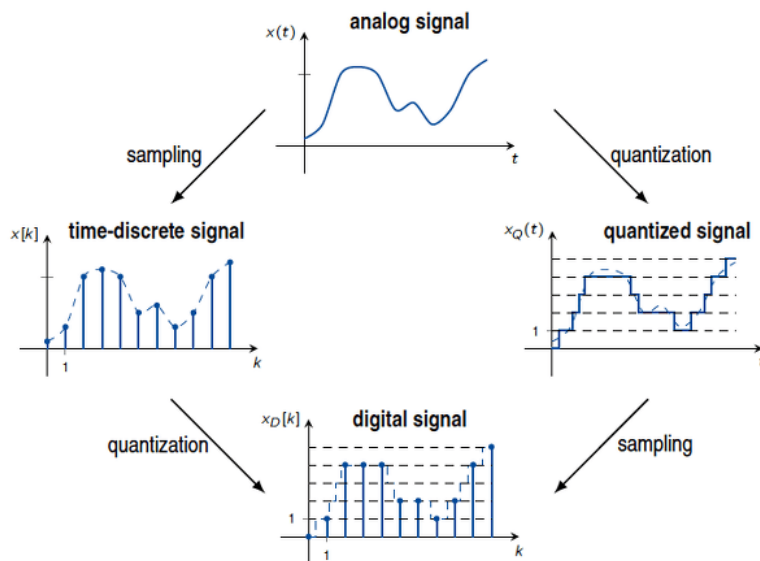


Figure 2.1: Analog to digital signal stages of signal representation. Sasche Spors, Digital Signal Processing - Theory and Computational Examples. Introduction [<https://nbviewer.org/github/spatiala...duction.ipynb>]

### The Sampling Theorem

Digital transmission of information and digital signal processing all require signals to first be "acquired" by a computer. One of the most amazing and useful results in electrical engineering is that signals can be converted from a function of time into a sequence of numbers **without error**: We can convert the numbers back into the signal with (theoretically) **no** error. Harold Nyquist, a Bell Laboratories engineer, first derived this result, known as the Sampling Theorem, in the 1920s. It found no real application back then. Claude Shannon, also at Bell Laboratories, revived the result once computers were made public after World War II.

The sampled version of the analog signal  $s(t)$  is  $s(nT_s)$ , with  $T_s$  known as the **sampling interval**. Clearly, the value of the original signal at the sampling times is preserved; the issue is how the signal values **between** the samples can be reconstructed since they are lost in the sampling process. To characterize sampling, we approximate it as the product:

$$x(t) = s(t)P_{T_s}(t)$$

with  $P_{T_s}$  being the periodic pulse signal. The resulting signal, as shown in Figure 5.3.1 has nonzero values only during the time intervals :

$$\left(nT_s - \frac{\Delta}{2}, nT_s + \frac{\Delta}{2}\right), n \in \{\dots, -1, 0, 1, \dots\}$$

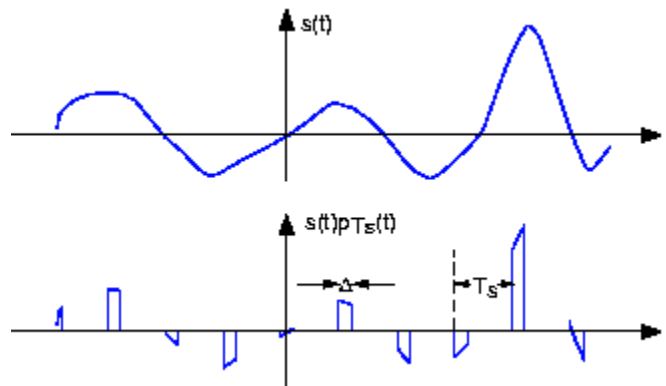


Figure 2.2: The waveform of an example signal is shown in the top plot with its sampled version in the bottom.

For our purposes here, we center the periodic pulse signal about the origin so that its Fourier series coefficients are real (the signal is even).

$$P_{T_s}(t) = \sum_{k=-\infty}^{\infty} c_k e^{\frac{j2\pi kt}{T_s}}$$

$$c_k = \frac{\sin\left(\frac{\pi k \Delta}{T_s}\right)}{\pi k}$$

If the properties of  $s(t)$  and the periodic pulse signal are chosen properly, we can recover  $s(t)$  from  $x(t)$  by filtering.

To understand how signal values between the samples can be "filled" in, we need to calculate the sampled signal's spectrum. Using the Fourier series representation of the periodic sampling signal,

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{\frac{j2\pi kt}{T_s}} s(t)$$

Considering each term in the sum separately, we need to know the spectrum of the product of the complex exponential and the signal. Evaluating this transform directly is quite easy.

$$\int_{-\infty}^{\infty} s(t) e^{\frac{j2\pi kt}{T_s}} e^{j2\pi ft} dt = \int_{-\infty}^{\infty} s(t) e^{-j2\pi \left(f - \frac{k}{T_s}\right)t} dt = S\left(f - \frac{k}{T_s}\right)$$

Thus, the spectrum of the sampled signal consists of weighted (by the coefficients  $c_k$ ) and delayed versions of the signal's spectrum Figure 5.3.2 below.

$$X(f) = \sum_{k=-\infty}^{\infty} c_k S\left(f - \frac{k}{T_s}\right)$$

In general, the terms in this sum overlap each other in the frequency domain, rendering recovery of the original signal impossible. This unpleasant phenomenon is known as **aliasing**.

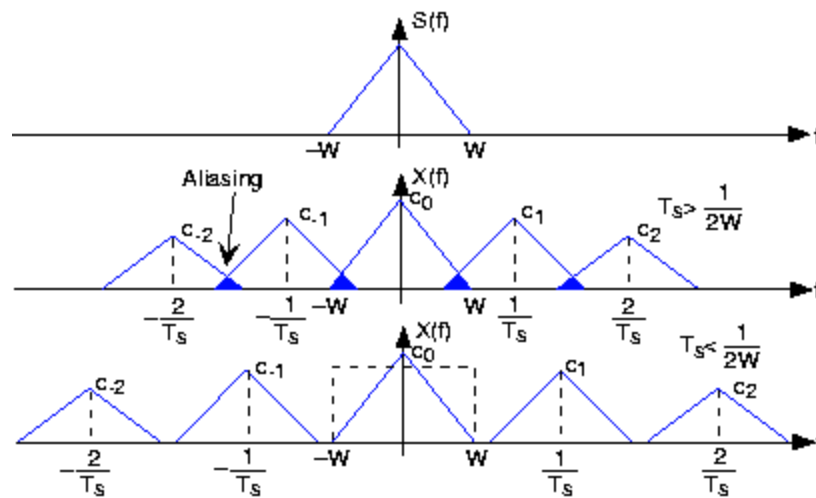


Figure 2.3: The spectrum of some bandlimited (to  $W$  Hz) signal is shown in the top plot. If the sampling interval  $T_s$  is chosen too large relative to the bandwidth  $W$ , aliasing will occur. In the bottom plot, the sampling interval is chosen sufficiently small to avoid aliasing. Note that if the signal were not bandlimited, the component spectra would **always** overlap.

If, however, we satisfy two conditions:

- The signal  $s(t)$  is **bandlimited**—has power in a restricted frequency range—to  $W$  Hz
- the sampling interval  $T_s$  is small enough so that the individual components in the sum do not overlap -  $T_s < 1/2 W$

Aliasing will not occur. In this delightful case, we can recover the original signal by lowpass filtering  $x(t)$  with a filter having a cutoff frequency equal to  $W$  Hz. These two conditions ensure the ability to recover a bandlimited signal from its sampled version: We thus have the **Sampling Theorem**.

### Example 2.1

The example below demonstrates the effect of under sampling a signal.

#### Solution

```
'''
 Aliasing Demonstration
'''
import numpy as np
import matplotlib.pyplot as plt

Define the original signal
f_original = 5 # Frequency of the original signal (Hz)
fs = 200 # Sampling frequency (Hz)
t = np.arange(0, 1+1/fs, 1/fs) # Time vector

Generate the original signal
original_signal = np.sin(2 * np.pi * f_original * t)

Undersample the signal
fs_undersampled = 4 # Undersampled sampling frequency (Hz)
t_undersampled = np.arange(0, 1+1/fs, 1/fs_undersampled)
undersampled_signal = np.sin(2 * np.pi * f_original * t_undersampled)

Plot the signals
plt.figure(figsize=(10, 6))
plt.plot(t, original_signal, label=f'Original Signal ({f_original} Hz)')
plt.plot(t_undersampled, undersampled_signal, 'ro-',
 label=f'Undersampled Signal (fs = {fs_undersampled} Hz)')
```

```
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.title('Aliasing Demonstration')
plt.legend(loc='upper right')
plt.grid()
plt.show()
```

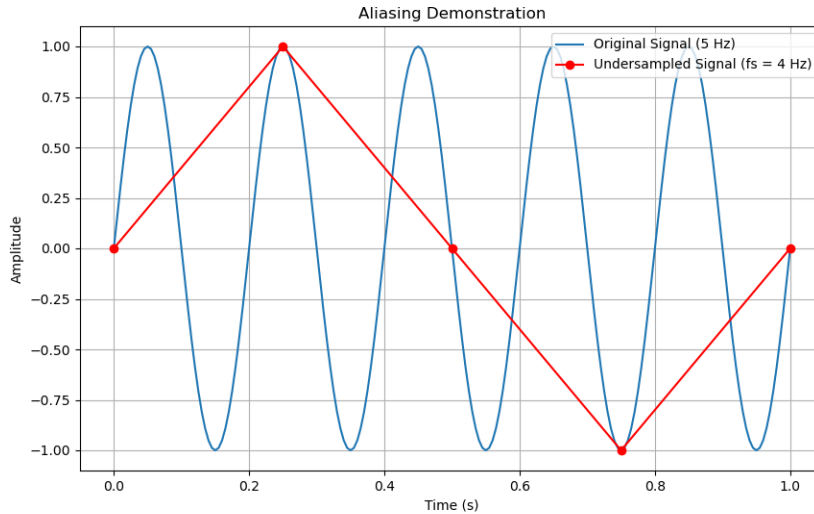


Figure 2.4: Aliasing example.

### Exercise 2.1

The Sampling Theorem (as stated) does not mention the pulse width  $\Delta$ . What is the effect of this parameter on our ability to recover a signal from its samples (assuming the Sampling Theorem's two conditions are met)?

#### Solution

The only effect of pulse duration is to unequally weight the spectral repetitions. Because we are only concerned with the repetition centered about the origin, the pulse duration has no significant effect on recovering a signal from its samples.

The frequency  $1/2T_s$ , known today as the **Nyquist frequency** and the **Shannon sampling frequency**, corresponds to the highest frequency at which a signal can contain energy and remain compatible with the Sampling Theorem. High-quality sampling systems ensure that no aliasing occurs by unceremoniously lowpass filtering the signal (cutoff frequency being slightly lower than the Nyquist frequency) before sampling. Such systems therefore vary the **anti-aliasing** filter's cutoff frequency as the sampling rate varies. Because such quality features cost money, many sound cards do **not** have anti-aliasing filters or, for that matter, post-sampling filters. They sample at high frequencies, 44.1 kHz for example, and hope the signal contains no frequencies above the Nyquist frequency (22.05 kHz in our example). If, however, the signal contains frequencies beyond the sound card's Nyquist frequency, the resulting aliasing can be impossible to remove.

If we satisfy the Sampling Theorem's conditions, the signal will change only slightly during each pulse. As we narrow the pulse, making

## Exercise 2.2

What is the simplest bandlimited signal? Using this signal, convince yourself that less than or equal two samples/period will not suffice to specify it. If the sampling rate  $1/T_s$  is not high enough, what signal would your resulting undersampled signal become? For example, let  $f_{s\_undersampled} = 10$  (the Nyquist frequency) in Exercise 2.1.

### Solution

The simplest bandlimited signal is the sine wave. At the Nyquist frequency, exactly two samples/period would occur. Reducing the sampling rate would result in fewer samples/period, and these samples would appear to have arisen from a lower frequency sinusoid.

## Footnotes

1. We assume that we do not use floating-point A/D converters.

## Continuous-Time Periodic Signals

A continuous-time period signal must meet the following requirement:

$$x(t) = x(t + T), \quad -\infty < t < \infty \quad (2.1)$$

for example

$$x(t) = A \cos(2\pi F t + \theta), \text{ where } T = 1/F \quad (2.2)$$

where F may be an irrational floating point number.

## Discrete-Time Periodic Signals

A discrete-time period signal (or sequence) must meet the following requirement:

$$x(n) = x(n + N), \quad -\infty < n < \infty \quad (2.3)$$

where n and N are integers.

Since n and N must be integers, there an additional requirement for trigonometric functions.

$$x(n) = A \cos(\omega n) \quad (2.4)$$

is a discrete-time periodic signal if

$$\omega N = 2\pi k \quad (2.5)$$

where the period, N, and k are both integer, and the frequency,  $f = \frac{k}{N}$ , is a rational fraction.

## Example 2.2

Is the following function periodic and if so what is the period?

$$x(n) = \cos(0.6\pi n) \quad (2.6)$$

### Solution

$$N = \frac{2\pi k}{\omega} = \frac{2\pi k}{0.6\pi} = \frac{10k}{3} \Rightarrow \frac{k}{N} = \frac{3}{10} \Rightarrow N = 10 \quad (2.7)$$

Note the  $\frac{k}{N}$  fraction was reduced to the lowest common denominator.

```
'''
 Example of a Periodic function
'''
import numpy as np
from numpy.fft import fft, fftshift, fftfreq
import matplotlib.pyplot as plt

k, N = 3, 10
F1 = k / N
n = np.arange(0, 2*N)
Create cosine sequence two periods long
x = np.cos(2*np.pi*F1*n)
X = fftshift(fft(x)/len(x))
F = fftshift(fftfreq(len(X)))
plt.figure(figsize=(14,6))
plt.subplot(121)
plt.stem(n,x)
plt.xlim(0,20)
plt.grid()
plt.xlabel('n')
plt.ylabel('x(n)')
plt.title('Index Domain')
plt.xticks(np.arange(0,2*N,2))

plt.subplot(122)
plt.stem(F,abs(X))
plt.grid()
plt.xlabel('$F = k/N$')
plt.ylabel('$|X_1(F)|$')
plt.title(rf'Frequency Domain, F1 = {F1:.3f}')
plt.show()
```

### Exercise 2.3

Add a second cos sequence with twice the magnitude,  $k = 1$ , and  $N = 5$ . Where will the components of this second signal appear in the frequency domain plot?

*Answer available at the end of this section.*

### Exercise 2.4

What if the period of the cos wave is changed to 11 but sampled for a period of 10?

**Answer**

*Answer available at the end of this section.*

### Exercise 2.5

What if the period of the cos wave is changed to 11 but sampled for a period of 10?

**Answer**

The power spectral density includes leakage components.

## Answers

### Exercise 2.3

They will appear at  $F = \pm 1/5 = \pm 0.2$  with twice the magnitude of the first signals frequency components.

### Exercise 2.4

The power spectral density includes leakage components.

---

This page titled [2: The Sampling Theorem](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.3: The Sampling Theorem](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 3: Amplitude Quantization

### Learning Objectives

- Analog-to-digital conversion.

The Sampling Theorem says that if we sample a bandlimited signal  $s(t)$  fast enough, it can be recovered without error from its samples

$$s(nT_s), n \in \{\dots, -1, 0, 1, \dots\}$$

Sampling is only the first phase of acquiring data into a computer: Computational processing further requires that the samples be **quantized**: analog values are converted into digital form. In short, we will have performed **analog-to-digital (A/D) conversion**.

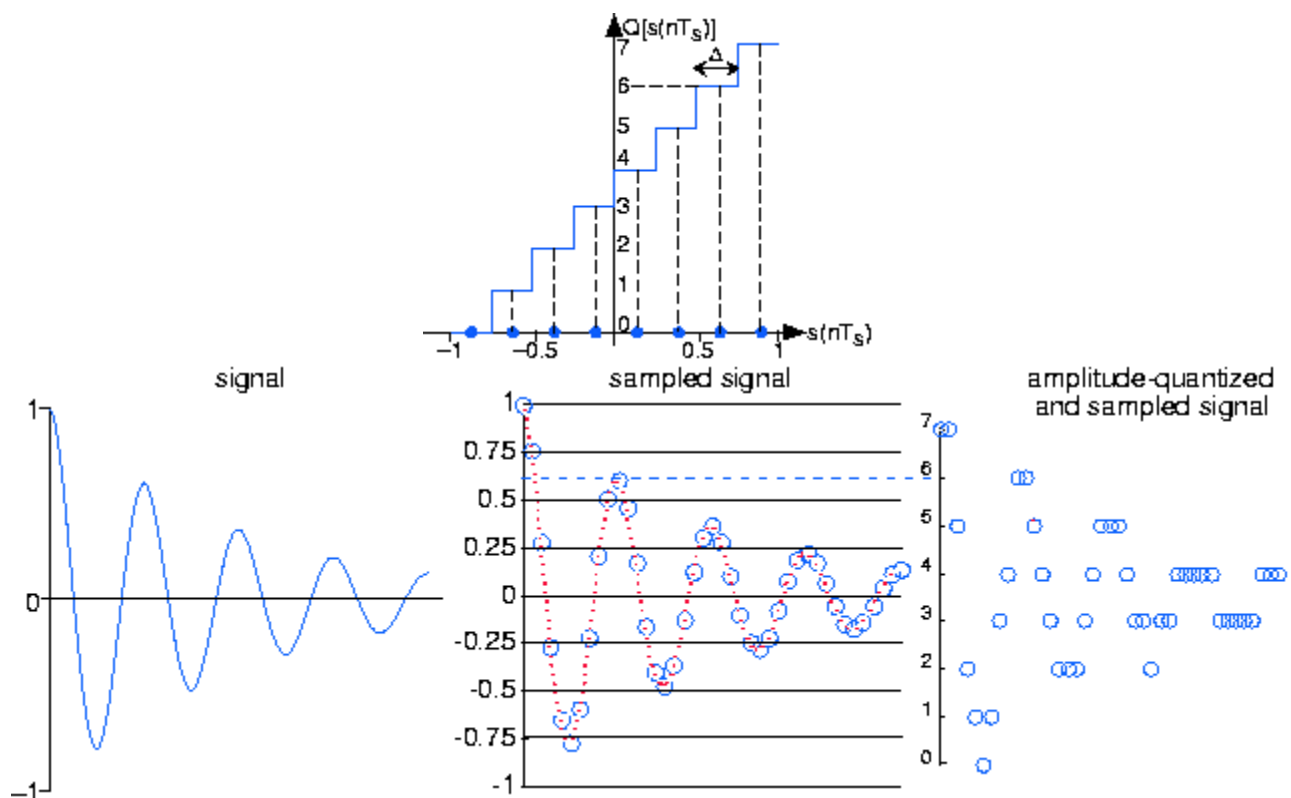


Figure 3.1: A three-bit A/D converter assigns voltage in the range  $[-1,1]$  to one of eight integers between 0 and 7. For example, all inputs having values lying between 0.5 and 0.75 are assigned the integer value six, and, upon conversion back to an analog value, they all become 0.625. The width of a single quantization interval  $\Delta$  equals  $2/2^B$ . The bottom panel shows a signal going through the analog-to-digital converter, where  $B$  is the number of bits used in the A/D conversion process (3 in the case depicted here). First, it is sampled, then amplitude-quantized to three bits. Note how the sampled signal waveform becomes distorted after amplitude quantization. For example, the two signal values between 0.5 and 0.75 become 0.625. This distortion is irreversible; it can be reduced (but not eliminated) by using more bits in the A/D converter.

A phenomenon reminiscent of the errors incurred in representing numbers on a computer prevents signal amplitudes from being converted with no error into a binary number representation. In analog-to-digital conversion, the signal is assumed to lie within a predefined range. Assuming we can scale the signal without affecting the information it expresses, we'll define this range to be  $[-1,1]$ . Furthermore, the A/D converter assigns amplitude values in this range to a set of integers. A  $B$ -bit converter produces one of the integers

$$\{0, 1, \dots, 2^B - 1\}$$

for each sampled input. Figure 5.4.1 shows how a three-bit A/D converter assigns input values to the integers. We define a **quantization interval** to be the range of values assigned to the same integer. Thus, for our example three-bit A/D converter, the quantization interval  $\Delta$  is 0.25; in general, it is  $2/2^B$ .

### Exercise 3.1

Recalling the plot of average daily highs in [this frequency domain problem](#), why is this plot so jagged? Interpret this effect in terms of analog-to-digital conversion.

#### Solution

The plotted temperatures were quantized to the nearest degree. Thus, the high temperature's amplitude was quantized as a form of A/D conversion.

Because values lying anywhere within a quantization interval are assigned the same value for computer processing, **the original amplitude value cannot be recovered without error**. Typically, the D/A converter, the device that converts integers to amplitudes, assigns an amplitude equal to the value lying halfway in the quantization interval. The integer 6 would be assigned to the amplitude 0.625 in this scheme. The error introduced by converting a signal from analog to digital form by sampling and amplitude quantization then back again would be half the quantization interval for each amplitude value. Thus, the so-called **A/D error** equals half the width of a quantization interval:  $1/2^B$ . As we have fixed the input-amplitude range, the more bits available in the A/D converter, the smaller the quantization error.

To analyze the amplitude quantization error more deeply, we need to compute the **signal-to-noise** ratio, which equals the ratio of the signal power and the quantization error power. Assuming the signal is a sinusoid, the signal power is the square of the rms amplitude:

$$power(s) = \left( \frac{1}{\sqrt{2}} \right)^2 = \frac{1}{2}$$

The Figure 5.4.2 details a single quantization interval.

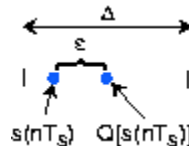


Figure 3.2: A single quantization interval is shown, along with a typical signal's value before amplitude quantization  $s(nT_s)$  and after  $Q[s(nT_s)]$ .  $\epsilon$  denotes the error thus incurred.

Its width is  $\Delta$  and the quantization error is denoted by  $\epsilon$ . To find the power in the quantization error, we note that no matter into which quantization interval the signal's value falls, the error will have the same characteristics. To calculate the rms value, we must square the error and average it over the interval.

$$\begin{aligned} rms(\epsilon) &= \sqrt{\frac{1}{\Delta} \int_{-\Delta}^{\Delta} \epsilon^2 d\epsilon} \\ &= \left( \frac{\Delta^2}{12} \right)^{\frac{1}{2}} \end{aligned}$$

Since the quantization interval width for a B-bit converter equals

$$\frac{2}{2^B} = 2^{-(B-1)}$$

we find that the signal-to-noise ratio for the analog-to-digital conversion process equals

$$SNR = \frac{\frac{1}{2}}{\frac{2^{-(2B-1)}}{12}} = \frac{3}{2} 2^{2B} = 6B + \log 1.5 dB$$

Thus, every bit increase in the A/D converter yields a 6 dB increase in the signal-to-noise ratio. The constant term **10 log1.5 equals 1.76**.

### Exercise 3.2

This derivation assumed the signal's amplitude lay in the range **[-1,1]**. What would the amplitude quantization signal-to-noise ratio be if it lay in the range **[-A,A]**?

#### Solution

The signal-to-noise ratio does not depend on the signal amplitude. With an A/D range of **[-A,A]** the quantization interval

$$\Delta = \frac{2A}{2^B}$$

and the signal's rms value (again assuming it is a sinusoid) is

$$\frac{A}{\sqrt{2}}$$

### Exercise 3.3

How many bits would be required in the A/D converter to ensure that the maximum amplitude quantization error was less than 60 db smaller than the signal's peak value?

#### Solution

Solving

$$2^{-B} = 0.001$$

results in

$$B = 10 \text{ bits}$$

### Exercise 3.4

Music on a CD is stored to 16-bit accuracy. To what signal-to-noise ratio does this correspond?

#### Solution

A 16-bit A/D converter yields a SNR of

$$6 \times 16 + 10 \log 1.5 = 97.8 dB$$

Once we have acquired signals with an A/D converter, we can process them using digital hardware or software. It can be shown that if the computer processing is linear, the result of sampling, computer processing, and unsampling is equivalent to some analog linear system. Why go to all the bother if the same function can be accomplished using analog techniques? Knowing when digital processing excels and when it does not is an important issue.

---

This page titled [3: Amplitude Quantization](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.4: Amplitude Quantization](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 4: Discrete -Time Signals and Systems

### Learning Objectives

- Study of Discrete-Time signals.

Mathematically, analog signals are functions having as their independent variables continuous quantities, such as space and time. Discrete-time signals are functions defined on the integers; they are sequences. As with analog signals, we seek ways of decomposing discrete-time signals into simpler components. Because this approach leads to a better understanding of signal structure, we can exploit that structure to represent information (create ways of representing information with signals) and to extract information (retrieve the information thus represented). For symbolic-valued signals, the approach is different: We develop a common representation of all symbolic-valued signals so that we can embody the information they contain in a unified way. From an information representation perspective, the most important issue becomes, for both real-valued and symbolic-valued signals, efficiency: what is the most parsimonious and compact way to represent information so that it can be extracted later.

### Real- and Complex-valued Signals

A discrete-time signal is represented symbolically as  $s(n)$  where  $n=\{..., -1, 0, 1, ...\}$

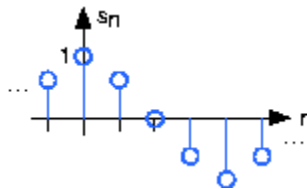


Figure 4.1: The discrete-time cosine signal is plotted as a stem plot. Can you find the formula for this signal?

We usually draw discrete-time signals as stem plots to emphasize the fact they are functions defined only on the integers. We can delay a discrete-time signal by an integer just as with analog ones. A signal delayed by  $m$  samples has the expression  $s(n-m)$ .

### Complex Exponentials

The most important signal is, of course, the **complex exponential sequence**.

$$s(n) = e^{i2\pi fn}$$

Note that the frequency variable  $f$  is dimensionless and that adding an integer to the frequency of the discrete-time complex exponential has no effect on the signal's value.

$$e^{i2\pi(f+m)n} = e^{i2\pi fn} e^{i2\pi mn} = e^{i2\pi fn}$$

This derivation follows because the complex exponential evaluated at an integer multiple of

### Sinusoids

Discrete-time sinusoids have the obvious form

$$s(n) = A \cos(2\pi fn + \varphi)$$

As opposed to analog complex exponentials and sinusoids that can have their frequencies be any real value, frequencies of their discrete-time counterparts yield unique waveforms **only** when  $f$  lies in the interval:

$$\left(-\frac{1}{2}, \frac{1}{2}\right]$$

This choice of frequency interval is arbitrary; we can also choose the frequency to lie in the interval **[0,1]**. How to choose a unit-length interval for a sinusoid's frequency will become evident later.

## Unit Sample

The second-most important discrete-time signal is the **unit sample**, which is defined to be

$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if otherwise} \end{cases}$$

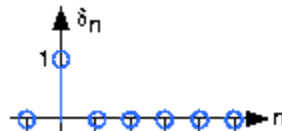


Figure 4.2: The unit sample.

Examination of a discrete-time signal's plot, like that of the cosine signal shown in Figure 5.5.2, reveals that all signals consist of a sequence of delayed and scaled unit samples. Because the value of a sequence at each integer **m** is denoted by **s(m)** and the unit sample delayed to occur at **m** is written **δ(n-m)**, we can decompose **any** signal as a sum of unit samples delayed to the appropriate location and scaled by the signal value.

$$s(n) = \sum_{m=-\infty}^{\infty} s(m)\delta(n-m)$$

This kind of decomposition is unique to discrete-time signals, and will prove useful subsequently.

## Unit Step

The **unit step** in discrete-time is well-defined at the origin, as opposed to the situation with analog signals.

$$u(n) = \begin{cases} 1 & \text{if } n \geq 0 \\ 0 & \text{if } n < 0 \end{cases}$$

## Symbolic Signals

An interesting aspect of discrete-time signals is that their values do not need to be real numbers. We do have real-valued discrete-time signals like the sinusoid, but we also have signals that denote the sequence of characters typed on the keyboard. Such characters certainly aren't real numbers, and as a collection of possible signal values, they have little mathematical structure other than that they are members of a set. More formally, each element of the **symbolic-valued** signal **s(n)** takes on one of the values **{a<sub>1</sub>, ..., a<sub>k</sub>}** which comprise the **alphabet A**. This technical terminology does not mean we restrict symbols to being members of the English or Greek alphabet. They could represent keyboard characters, bytes (8-bit quantities), integers that convey daily temperature. Whether controlled by software or not, discrete-time systems are ultimately constructed from digital circuits, which consist **entirely** of analog circuit elements. Furthermore, the transmission and reception of discrete-time signals, like e-mail, is accomplished with analog signals and systems. Understanding how discrete-time and analog signals and systems intertwine is perhaps the main goal of this course.

## Discrete-Time Systems

Discrete-time systems can act on discrete-time signals in ways similar to those found in analog signals and systems. Because of the role of software in discrete-time systems, many more different systems can be envisioned and "constructed" with programs than can be with analog signals. In fact, a special class of analog signals can be converted into discrete-time signals, processed with software, and converted back into an analog signal, all without the incursion of error. For such signals, systems can be easily produced in software, with equivalent analog realizations difficult, if not impossible, to design.

## Discrete-Time System Classifications

### Linearity

A discrete-time system is linear if it satisfies the properties of homogeneity (scaling) and additivity

- Scaling:  $\mathcal{T}[ax(n)] = a\mathcal{T}[x(n)]$
- Additivity:  $\mathcal{T}[x_1(n) + x_2(n)] = \mathcal{T}[x_1(n)] + \mathcal{T}[x_2(n)]$

### Time or Shift Invariant

A discrete-time systems is *shift invariant* if and only if

$$y(n) = \mathcal{T}[x(n)] \Rightarrow y(n - k) = \mathcal{T}[x(n - k)]$$

If the input to a system is delayed by  $k$ , the output should be the same output delayed by  $k$ .

### Causality

A discrete-time system is causal if the output at any index  $n$  depends only the current and previous indices

$$y(n) = \mathcal{T}[x(n), x(n - 1), x(n - 2), \dots]$$

A system that is a function of future values of the input in addition to the current and previous inputs is noncausal. A systems that is a function of only future inputs is anti-causal.

### Stability

A discrete-time system is bounded input-bounded output (BIBO) stable if every bounded input produces a bounded output.

If  $|x(n)| \leq K < \infty \Rightarrow |y(n)| \leq M < \infty \quad \forall n$  then the system is BIBO stable.

### Static vs Dynamic

A discrete-time system that is only dependent on the input at current index is *static* or a *memoryless* system. Systems that depend on previous or future values of the input are *dynamic* systems.

---

This page titled [4: Discrete -Time Signals and Systems](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.5: Discrete -Time Signals and Systems](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 5: Z-TRANSFORM AND DISCRETE TIME SYSTEM DESIGN

- 5.1: Z-Transform
- 5.2: Common Z-Transforms
- 5.3: Properties of the Z-Transform
- 5.4: Inverse Z-Transform
- 5.5: Poles and Zeros in the Z-Plane
- 5.6: Region of Convergence for the Z-Transform
- 5.7: Rational Functions and the Z-Transform
- 5.8: Difference Equations
- 5.9: Discrete Time Filter Design

---

This page titled [5: Z-Transform and Discrete Time System Design](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- 12: Z-Transform and Discrete Time System Design by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.1: Z-Transform

### Introduction

The Z transform is a generalization of the Discrete-Time Fourier Transform (Section 9.2). It is used because the DTFT does not converge/exist for many important signals, and yet does for the z-transform. It is also used because it is notationally cleaner than the DTFT. In contrast to the DTFT, instead of using complex exponentials (Section 7.2) of the form  $(e^{j\omega n})$ , with purely imaginary parameters, the Z transform uses the more general,  $z^n$ , where  $z$  is complex. The Z-transform thus allows one to bring in the power of complex variable theory into Digital Signal Processing.

### The Z-Transform

#### Bilateral Z-transform Pair

Although Z transforms are rarely solved in practice using integration (tables and computers (e.g. Matlab) are much more common), we will provide the **bilateral Z transform pair** here for purposes of discussion and derivation. These define the forward and inverse Z transformations. Notice the similarities between the forward and inverse transforms. This will give rise to many of the same symmetries found in Fourier analysis (Section 5.1).

#### Z Transform

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

#### Inverse Z Transform

$$x[n] = \frac{1}{2\pi j} \oint_r X(z)z^{n-1}dz$$

#### Note

We have defined the bilateral z-transform. There is also a **unilateral z-transform**,

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n}$$

which is useful for solving the difference equations with nonzero initial conditions. This is similar to the **unilateral Laplace Transform** in continuous time.

### Relation between Z-transform and DTFT

Taking a look at the equations describing the Z-Transform and the Discrete-Time Fourier Transform:

#### Discrete-Time Fourier Transform

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

#### Z-Transform

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

We can see many similarities; first, that:

$$X(e^{j\omega}) = X(z)$$

for all  $z = e^{j\omega}$

## Visualizing the Z-transform

With the DTFT, we have a complex-valued function of a real-valued variable  $\omega$  (and  $2\pi$  periodic). The Z-transform is a complex-valued function of a complex valued variable  $z$ .

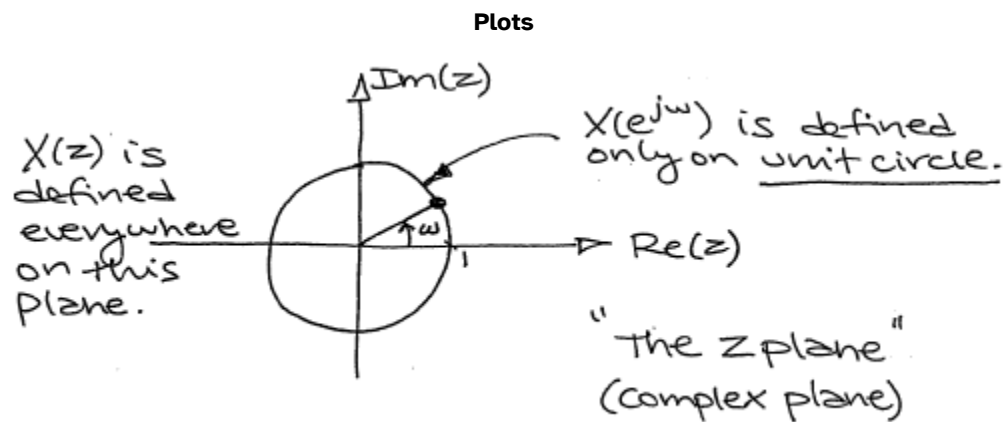


Figure  
5.1.1

With the Fourier transform, we had a **complex-valued function** of a **purely imaginary variable**,  $F(j\omega)$ . This was something we could envision with two 2-dimensional plots (real and imaginary parts or magnitude and phase). However, with Z, we have a **complex-valued function** of a **complex variable**. In order to examine the magnitude and phase or real and imaginary parts of this function, we must examine 3-dimensional surface plots of each component.

Consider the z-transform given by  $H(z) = z$ , as illustrated below.

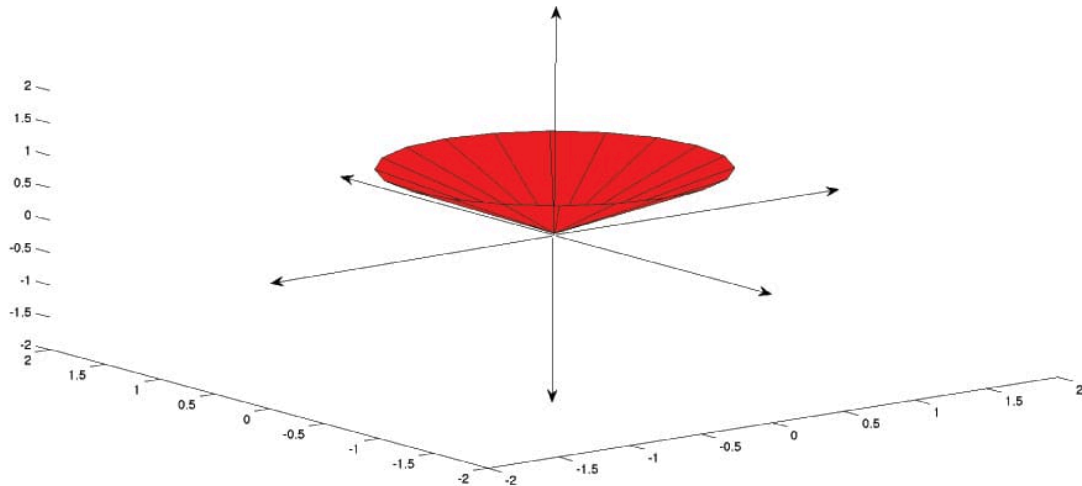
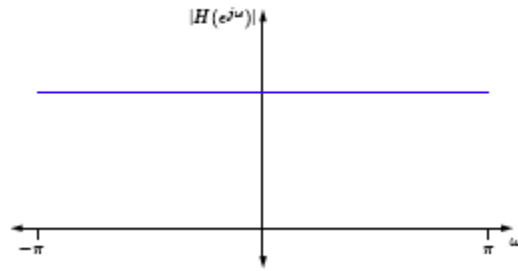


Figure  
5.1.2

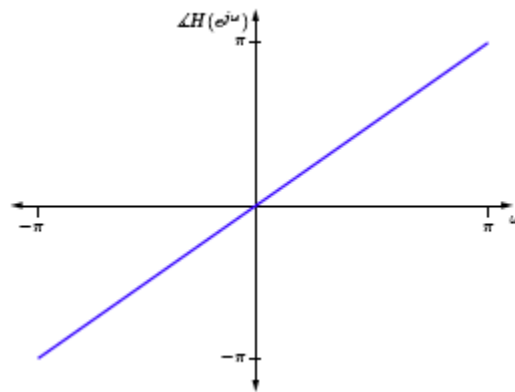
The corresponding DTFT has magnitude and phase given below.

#### Note

While these are legitimate ways of looking at a signal in the Z domain, it is quite difficult to draw and/or analyze. For this reason, a simpler method has been developed. Although it will not be discussed in detail here, the method of [Poles and Zeros](#) is much easier to understand and is the way both the Z transform and its continuous-time counterpart the **Laplace-transform** are represented graphically.



(a)



(b)

Figure 5.1.3: Magnitude and Phase of  $H(z)$ .

What could the system  $H$  be doing? It is a perfect all-pass, linear-phase system. But what does this mean?

Suppose  $h[n] = \delta[n - n_0]$ . Then

$$\begin{aligned} H(z) &= \sum_{n=-\infty}^{\infty} h[n] z^{-n} \\ &= \sum_{n=-\infty}^{\infty} \delta[n - n_0] z^{-n} \\ &= z^{-n_0} \end{aligned}$$

Thus,  $H(z) = z^{-n_0}$  is the  $z$ -transform of a system that simply delays the input by  $n_0$ .  $H(z)$  is the  $z$ -transform of a unit-delay.

Now consider  $x[n] = \alpha^n u[n]$

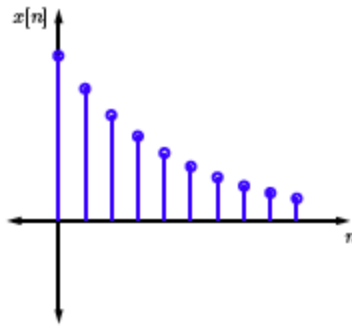


Figure  
5.1.4

$$\begin{aligned}
 X(z) &= \sum_{n=-\infty}^{\infty} x[n]z^{-n} = \sum_{n=0}^{\infty} \alpha^n z^{-n} \\
 &= \sum_{n=0}^{\infty} \left(\frac{\alpha}{z}\right)^n \\
 &= \frac{1}{1 - \frac{\alpha}{z}} \left( \text{if } \left|\frac{\alpha}{z}\right| < 1 \right) \text{ ( Geometric series )} \\
 &= \frac{z}{z - \alpha}
 \end{aligned} \tag{5.1.1}$$

What if  $\left|\frac{\alpha}{z}\right| \geq 1$ ? Then  $\sum_{n=0}^{\infty} \left(\frac{\alpha}{z}\right)^n$  does not converge! Therefore, whenever we compute a  $z$ -transform, we must also specify the set of  $z$ 's for which the  $z$ -transform exists. This is called the *region of convergence* (ROC).

#### Note: Using a computer to find the Z-transform

Matlab has two functions,

`ztrans`

and

`iztrans`

that are both part of the symbolic toolbox, and will find the Z and inverse Z transforms respectively. This method is generally preferred for more complicated functions. Simpler and more contrived functions are usually found easily enough by using [tables](#).

## Application to Discrete Time Filters

The  $z$ -transform might seem slightly ugly. We have to worry about the region of convergence, and stability issues, and so forth. However, in the end it is worthwhile because it proves extremely useful in analyzing digital filters with feedback. For example, consider the system illustrated below

## Plots

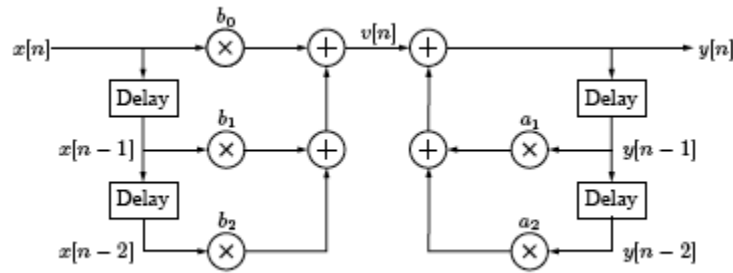


Figure  
5.1.5

We can analyze this system via the equations

$$v[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2]$$

and

$$y[n] = v[n] + a_1y[n-1] + a_2y[n-2]$$

More generally,

$$v[n] = \sum_{k=0}^N b_kx[n-k]$$

and

$$y[n] = \sum_{k=1}^M a_ky[n-k] + v[n]$$

or equivalently,

$$\sum_{k=0}^N b_kx[n-k] = y[n] - \sum_{k=1}^M a_ky[n-k].$$

What does the  $z$ -transform of this relationship look like?

$$\begin{aligned} Z \sum_{k=0}^M a_ky[n-k] &= Z \sum_{k=0}^M b_kx[n-k] \\ \sum_{k=0}^M a_kZ\{y[n-k]\} &= \sum_{k=0}^M b_kZ\{x[n-k]\} \end{aligned}$$

Note that

$$\begin{aligned} Z\{y[n-k]\} &= \sum_{n=-\infty}^{\infty} y[n-k]z^{-n} \\ &= \sum_{m=-\infty}^{\infty} y[m]z^{-m}z^{-k} \\ &= Y(z)z^{-k} \end{aligned}$$

Thus the relationship reduces to

$$\begin{aligned}\sum_{k=0}^M a_k Y(z) z^{-k} &= \sum_{k=0}^N b_k X(z) z^{-k} \\ Y(z) \sum_{k=0}^M a_k z^{-k} &= X(z) \sum_{k=0}^N b_k z^{-k} \\ \frac{Y(z)}{X(z)} &= \frac{\sum_{k=0}^N b_k z^{-k}}{\sum_{k=0}^M a_k z^{-k}}\end{aligned}\tag{5.1.2}$$

Hence, given a system the one above, we can easily determine the system's transfer function, and end up with a ratio of two polynomials in  $z^{-1}$ : a rational function. Similarly, given a rational function, it is easy to realize this function in a simple hardware architecture.

## Interactive Z-Transform Demonstration

### Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 5.1.6: Interact (when online) with a Mathematica CDF demonstrating the Z Transform. To Download, right-click and save target as .cdf.

## Conclusion

The z-transform proves a useful, more general form of the Discrete Time Fourier Transform. It applies equally well to describing systems as well as signals using the eigenfunction method, and proves extremely useful in digital filter design.

This page titled [5.1: Z-Transform](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.1: Z-Transform](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.2: Common Z-Transforms

Table 5.2.1: z-Transform Pairs

| Signal, $x[n]$              | Z-transform, $X(z)$                                                  | Region of Convergence (ROC) |
|-----------------------------|----------------------------------------------------------------------|-----------------------------|
| $\delta[n]$                 | 1                                                                    | all $z$                     |
| $\delta[n - m]$             | $z^{-m}$                                                             | all $z \neq 0$              |
| $u[n]$                      | $\frac{1}{1-z^{-1}}$                                                 | $ z  > 1$                   |
| $a^n u[n]$                  | $\frac{1}{1-az^{-1}}$                                                | $ z  >  a $                 |
| $na^n u[n]$                 | $\frac{az^{-1}}{(1-az^{-1})^2}$                                      | $ z  >  a $                 |
| $-a^n u[-n - 1]$            | $\frac{1}{1-az^{-1}}$                                                | $ z  <  a $                 |
| $-na^n u[-n - 1]$           | $\frac{az^{-1}}{(1-az^{-1})^2}$                                      | $ z  <  a $                 |
| $\cos(\omega_0 n) u[n]$     | $\frac{1-z^{-1}\cos(\omega_0)}{1-2z^{-1}\cos(\omega_0)+z^{-2}}$      | $ z  > 1$                   |
| $\sin(\omega_0 n) u[n]$     | $\frac{z^{-1}\sin(\omega_0)}{1-2z^{-1}\cos(\omega_0)+z^{-2}}$        | $ z  > 1$                   |
| $a^n \cos(\omega_0 n) u[n]$ | $\frac{1-az^{-1}\cos(\omega_0)}{1-2az^{-1}\cos(\omega_0)+a^2z^{-2}}$ | $ z  >  a $                 |
| $a^n \sin(\omega_0 n) u[n]$ | $\frac{az^{-1}\sin(\omega_0)}{1-2az^{-1}\cos(\omega_0)+a^2z^{-2}}$   | $ z  >  a $                 |

This page titled 5.2: Common Z-Transforms is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by Carl Greco.

- 12.2: Common Z-Transforms by Richard Baraniuk et al. is licensed CC BY 3.0. Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.3: Properties of the Z-Transform

### Introduction

This module will look at some of the basic properties of the Z-Transform (Section 9.2) (DTFT).

#### Note

We will be discussing these properties for aperiodic, discrete-time signals but understand that very similar properties hold for continuous-time signals and periodic signals as well.

### Discussion of Z-Transform Properties

#### Linearity

The combined addition and scalar multiplication properties in the table above demonstrate the basic property of linearity. What you should see is that if one takes the Z-transform of a linear combination of signals then it will be the same as the linear combination of the Z-transforms of each of the individual signals. This is crucial when using a table (Section 8.3) of transforms to find the transform of a more complicated signal.

#### Example 5.3.1

We will begin with the following signal:

$$x[n] = af_1[n] + bf_2[n]$$

Now, after we take the Fourier transform, shown in the equation below, notice that the linear combination of the terms is unaffected by the transform.

$$X(z) = aF_1(z) + bF_2(z)$$

#### Symmetry

Symmetry is a property that can make life quite easy when solving problems involving Z-transforms. Basically what this property says is that since a rectangular function in time is a sinc function in frequency, then a sinc function in time will be a rectangular function in frequency. This is a direct result of the symmetry between the forward Z and the inverse Z transform. The only difference is the scaling by  $2\pi$  and a frequency reversal.

#### Time Scaling

This property deals with the effect on the frequency-domain representation of a signal if the time variable is altered. The most important concept to understand for the time scaling property is that signals that are narrow in time will be broad in frequency and vice versa. The simplest example of this is a delta function, a unit pulse with a **very** small duration, in time that becomes an infinite-length constant function in frequency.

The table above shows this idea for the general transformation from the time-domain to the frequency-domain of a signal. You should be able to easily notice that these equations show the relationship mentioned previously: if the time variable is increased then the frequency range will be decreased.

## Time Shifting

Time shifting shows that a shift in time is equivalent to a linear phase shift in frequency. Since the frequency content depends only on the shape of a signal, which is unchanged in a time shift, then only the phase spectrum will be altered. This property is proven below:

### Example 5.3.2

We will begin by letting  $x[n] = f[n - \eta]$ . Now let's take the z-transform with the previous expression substituted in for  $x[n]$ .

$$X(z) = \sum_{n=-\infty}^{\infty} f[n - \eta] z^{-n}$$

Now let's make a simple change of variables, where  $\sigma = n - \eta$ . Through the calculations below, you can see that only the variable in the exponential are altered thus only changing the phase in the frequency domain.

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} f[\sigma] z^{-(\sigma+\eta)} \\ &= z^{-\eta} \sum_{\sigma=-\infty}^{\infty} f[\sigma] z^{-\sigma} \\ &= z^{-\eta} F(z) \end{aligned} \tag{5.3.1}$$

## Convolution

Convolution is one of the big reasons for converting signals to the frequency domain, since convolution in time becomes multiplication in frequency. This property is also another excellent example of symmetry between time and frequency. It also shows that there may be little to gain by changing to the frequency domain when multiplication in time is involved.

We will introduce the convolution integral here, but if you have not seen this before or need to refresh your memory, then look at the discrete-time convolution (Section 4.3) module for a more in depth explanation and derivation.

$$\begin{aligned} y[n] &= (f_1[n], f_2[n]) \\ &= \sum_{\eta=-\infty}^{\infty} f_1[\eta] f_2[n - \eta] \end{aligned} \tag{5.3.2}$$

## Time Differentiation

Since discrete LTI (Section 2.1) systems can be represented in terms of difference equations, it is apparent with this property that converting to the frequency domain may allow us to convert these complicated difference equations to simpler equations involving multiplication and addition.

## Parseval's Relation

$$\sum_{n=-\infty}^{\infty} x[n] x^*[n] = \int_{-\pi}^{\pi} F(z) F^*(z) dz$$

Parseval's relation tells us that the energy of a signal is equal to the energy of its Fourier transform.

## Modulation (Frequency Shift)

Modulation is absolutely imperative to communications applications. Being able to shift a signal to a different frequency, allows us to take advantage of different parts of the electromagnetic spectrum is what allows us to transmit television, radio and other applications through the same space without significant interference.

The proof of the frequency shift property is very similar to that of the time shift; however, here we would use the inverse Fourier transform in place of the Fourier transform. Since we went through the steps in the previous, time-shift proof, below we will just show the initial and final step to this proof:

$$z(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega - \phi) e^{j\omega t} d\omega$$

Now we would simply reduce this equation through another change of variables and simplify the terms. Then we will prove the property expressed in the table above:

$$z(t) = f(t) e^{j\phi t}$$

## Properties Demonstration

An interactive example demonstration of the properties is included below:

### Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 5.3.1: Interactive Signal Processing Laboratory Virtual Instrument created using NI's Labview.

## Summary Table

Table 5.3.1: Properties of the Z-Transform

| Property                    | Signal                                                                     | Z-Transform                                             | Region of Convergence                     |
|-----------------------------|----------------------------------------------------------------------------|---------------------------------------------------------|-------------------------------------------|
| Linearity                   | $\alpha x_1(n) + \beta x_2(n)$                                             | $\alpha X_1(z) + \beta X_2(z)$                          | At least $\text{ROC}_1 \cap \text{ROC}_2$ |
| Time shifting               | $x(n - k)$                                                                 | $z^{-k} X(z)$                                           | ROC                                       |
| Time scaling                | $x(n/k) = \begin{cases} x[r] & n = rk \\ 0 & \text{otherwise} \end{cases}$ | $X(z^k)$                                                | $\text{ROC}^{1/k}$                        |
| Z-domain scaling            | $a^n x(n)$                                                                 | $X(z/a)$                                                | $ a $ ROC                                 |
| Conjugation                 | $\overline{x(n)}$                                                          | $\overline{X(\bar{z})}$                                 | ROC                                       |
| Convolution                 | $x_1(n) * x_2(n)$                                                          | $X_1(z) X_2(z)$                                         | At least $\text{ROC}_1 \cap \text{ROC}_2$ |
| Differentiation in z-Domain | $[nx[n]]$                                                                  | $-\frac{d}{dz} X(z)$                                    | ROC = all $\mathbb{R}$                    |
| Initial value theorem       | $x[0]$                                                                     | $\lim_{z \rightarrow \infty} X(z)$                      |                                           |
| Final value theorem         | $\lim_{n \rightarrow \infty} x[n]$                                         | $\lim_{z \rightarrow 1} (z - 1) X(z)$                   | if $x[\infty]$ exists                     |
| Parseval's Theorem          | $\sum_{n=-\infty}^{\infty} x_1[n] x_2^*[n]$                                | $\frac{1}{2\pi j} \oint_C X_1(\nu) X_2^*(1/\nu^*) d\nu$ |                                           |

---

This page titled [5.3: Properties of the Z-Transform](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.3: Properties of the Z-Transform](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.4: Inverse Z-Transform

### Introduction

When using the [z-transform](#)

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

it is often useful to be able to find  $x[n]$  given  $X(z)$ . There are at least 4 different methods to do this:

1. Inspection
2. Partial-Fraction Expansion
3. Power Series Expansion
4. Contour Integration

### Inspection Method

This "method" is to basically become familiar with the [z-transform pair tables](#) and then "reverse engineer".

#### Example 5.4.1

When given

$$X(z) = \frac{z}{z - \alpha}$$

with an ROC (Section 12.6) of

$$|z| > \alpha$$

we could determine "by inspection" that

$$x[n] = \alpha^n u[n]$$

### Partial-Fraction Expansion Method

When dealing with **linear time-invariant systems** the z-transform is often of the form

$$\begin{aligned} X(z) &= \frac{B(z)}{A(z)} \\ &= \frac{\sum_{k=0}^M b_k z^{-k}}{\sum_{k=0}^N a_k z^{-k}} \end{aligned} \tag{5.4.1}$$

This can also expressed as

$$X(z) = \frac{a_0}{b_0} \frac{\prod_{k=1}^M (1 - c_k z^{-1})}{\prod_{k=1}^N (1 - d_k z^{-1})}$$

where  $c_k$  represents the nonzero zeros of  $X(z)$  and  $d_k$  represents the nonzero poles.

If  $N \leq M$

$$X(z) = \sum_{k=1}^N \frac{A_k}{1 - d_k z^{-1}}$$

This form allows for easy inversions of each term of the sum using the inspection method and the [transform table](#). If the numerator is a polynomial, however, then it becomes necessary to use [partial-fraction expansion](#) to put  $X(z)$  in the above form. If  $M \geq N$  then  $X(z)$  can be expressed as

$$X(z) = \sum_{r=0}^{M-N} B_r z^{-r} + \frac{\sum_{k=0}^{N-1} b'_k z^{-k}}{\sum_{k=0}^N a_k z^{-k}}$$

### Example 5.4.2

Find the inverse z-transform of

$$X(z) = \frac{1 + 2z^{-1} + z^{-2}}{1 - 3z^{-1} + 2z^{-2}}$$

where the ROC is  $|z| > 2$ . In this case  $M = N = 2$ , so we have to use long division to get

$$X(z) = \frac{1}{2} + \frac{\frac{1}{2} + \frac{7}{2}z^{-1}}{1 - 3z^{-1} + 2z^{-2}}$$

Next factor the denominator.

$$X(z) = 2 + \frac{-1 + 5z^{-1}}{(1 - 2z^{-1})(1 - z^{-1})}$$

Now do partial-fraction expansion.

$$X(z) = \frac{1}{2} + \frac{A_1}{1 - 2z^{-1}} + \frac{A_2}{1 - z^{-1}} = \frac{1}{2} + \frac{\frac{9}{2}}{1 - 2z^{-1}} + \frac{-4}{1 - z^{-1}}$$

Now each term can be inverted using the inspection method and the z-transform table. Thus, since the ROC is  $|z| > 2$ ,

$$x[n] = \frac{1}{2}\delta[n] + \frac{9}{2}2^n u[n] - 4u[n]$$

## Demonstration of Partial Fraction Expansion

A demonstration involving Partial Fraction Expansion

Figure 5.4.1: Interactive experiment illustrating how the Partial Fraction Expansion method is used to solve a variety of numerator and denominator problems. (To view and interact with the simulation, download the free Mathematica player at [www.wolfram.com/products/player/download.cgi](http://www.wolfram.com/products/player/download.cgi))

### YouTube Video

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this YouTube Video online](#)



Figure 5.4.2: video from [Khan Academy](#)

## Power Series Expansion Method

When the z-transform is defined as a power series in the form

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

then each term of the sequence  $x[n]$  can be determined by looking at the coefficients of the respective power of  $z^{-n}$ .

### Example 5.4.3

Now look at the z-transform of a **finite-length sequence**.

$$\begin{aligned} X(z) &= z^2 (1 + 2z^{-1}) \left(1 - \frac{1}{2}z^{-1}\right) (1 + z^{-1}) \\ &= z^2 + \frac{5}{2}z + \frac{1}{2} + -z^{-1} \end{aligned}$$

In this case, since there were no poles, we multiplied the factors of  $X(z)$ . Now, by inspection, it is clear that

$$x[n] = \delta[n + 2] + \frac{5}{2}\delta[n + 1] + \frac{1}{2}\delta[n] + -\delta[n - 1]$$

One of the advantages of the power series expansion method is that many functions encountered in engineering problems have their power series' tabulated. Thus functions such as log, sin, exponent, sinh, etc, can be easily inverted.

### Example 5.4.4

Suppose

$$X(z) = \log_n (1 + \alpha z^{-1})$$

Noting that

$$\log_n (1 + x) = \sum_{n=1}^{\infty} \frac{-1^{n+1} x^n}{n}$$

Then

$$X(z) = \sum_{n=1}^{\infty} \frac{-1^{n+1} \alpha^n z^{-n}}{n}$$

Therefore

$$X(z) = \begin{cases} \frac{-1^{n+1} \alpha^n}{n} & \text{if } n \geq 1 \\ 0 & \text{if } n \leq 0 \end{cases}$$

## Contour Integration Method

Without going in to much detail

$$x[n] = \frac{1}{2\pi j} \oint_r X(z) z^{n-1} dz$$

where  $r$  is a counter-clockwise contour in the ROC of  $X(z)$  encircling the origin of the z-plane. To further expand on this method of finding the inverse requires the knowledge of complex variable theory and thus will not be addressed in this module.

## Demonstration of Contour Integration

A demonstration involving Contour Integration

Figure 5.4.3: Interactive experiment illustrating how the contour integral is applied on a simple example. For a more in-depth discussion of this method, some background in complex analysis is required. (To view and interact with the simulation, download the free Mathematica player at [www.wolfram.com/products/player/download.cgi](http://www.wolfram.com/products/player/download.cgi))

## Conclusion

The Inverse Z-transform is very useful to know for the purposes of designing a filter, and there are many ways in which to calculate it, drawing from many disparate areas of mathematics. All nevertheless assist the user in reaching the desired time-domain signal that can then be synthesized in hardware(or software) for implementation in a real-world filter.

---

This page titled [5.4: Inverse Z-Transform](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.4: Inverse Z-Transform](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.5: Poles and Zeros in the Z-Plane

### Introduction to Poles and Zeros of the Z-Transform

It is quite difficult to qualitatively analyze the Laplace transform (Section 11.1) and [Z-transform](#), since mappings of their magnitude and phase or real part and imaginary part result in multiple mappings of 2-dimensional surfaces in 3-dimensional space. For this reason, it is very common to examine a plot of a transfer function's poles and zeros to try to gain a qualitative idea of what a system does.

Once the Z-transform of a system has been determined, one can use the information contained in function's polynomials to graphically represent the function and easily observe many defining characteristics. The Z-transform will have the below structure, based on Rational Functions (Section 12.7):

$$X(z) = \frac{P(z)}{Q(z)}$$

The two polynomials,  $P(z)$  and  $Q(z)$ , allow us to find the [poles and zeros](#) of the Z-Transform.

#### Definition: zeros

1. The value(s) for  $z$  where  $P(z) = 0$ .
2. The complex frequencies that make the overall gain of the filter transfer function zero.

#### Definition: poles

1. The value(s) for  $z$  where  $Q(z) = 0$ .
2. The complex frequencies that make the overall gain of the filter transfer function infinite.

#### Example 5.5.1

Below is a simple transfer function with the poles and zeros shown below it.

$$H(z) = \frac{z + 1}{\left(z - \frac{1}{2}\right)\left(z + \frac{3}{4}\right)}$$

The zeros are:  $-1$

The poles are:  $\left\{\frac{1}{2}, -\frac{3}{4}\right\}$

### The Z-Plane

Once the poles and zeros have been found for a given Z-Transform, they can be plotted onto the Z-Plane. The Z-plane is a complex plane with an imaginary and real axis referring to the complex-valued variable  $z$ . The position on the complex plane is given by  $re^{j\theta}$  and the angle from the positive, real axis around the plane is denoted by  $\theta$ . When mapping poles and zeros onto the plane, poles are denoted by an "x" and zeros by an "o". The below figure shows the Z-Plane, and examples of plotting zeros and poles onto the plane can be found in the following section.

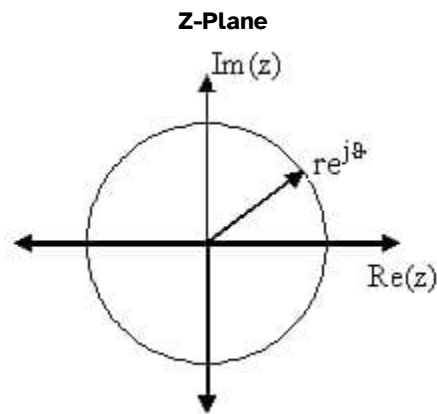


Figure 5.5.1

## Examples of Pole/Zero Plots

This section lists several examples of finding the poles and zeros of a transfer function and then plotting them onto the Z-Plane.

### Example 5.5.2: Simple Pole/Zero Plot

$$H(z) = \frac{z}{(z - \frac{1}{2})(z + \frac{3}{4})}$$

The zeros are:  $\{0\}$

The poles are:  $\{\frac{1}{2}, -\frac{3}{4}\}$

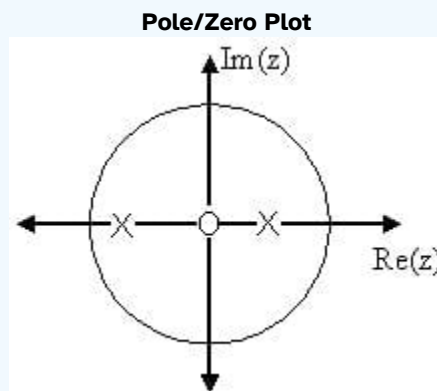


Figure 5.5.2: Using the zeros and poles found from the transfer function, the one zero is mapped to zero and the two poles are placed at  $\frac{1}{2}$  and  $-\frac{3}{4}$

### Example 5.5.3: Complex Pole/Zero Plot

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

The zeros are:  $\{j, -j\}$

The poles are:  $\{-1, \frac{1}{2} + \frac{1}{2}j, \frac{1}{2} - \frac{1}{2}j\}$

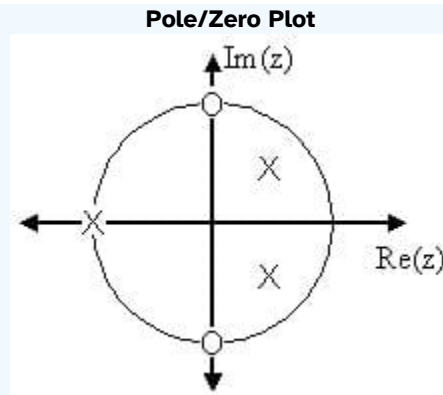


Figure 5.5.3: Using the zeros and poles found from the transfer function, the zeros are mapped to  $\pm(j)$ , and the poles are placed at  $-1$ ,  $\frac{1}{2} + \frac{1}{2}j$  and  $\frac{1}{2} - \frac{1}{2}j$

#### Example 5.5.4: Pole-Zero Cancellation

An easy mistake to make with regards to poles and zeros is to think that a function like  $\frac{(s+3)(s-1)}{s-1}$  is the same as  $s + 3$ . In theory they are equivalent, as the pole and zero at  $s = 1$  cancel each other out in what is known as **pole-zero cancellation**. However, think about what may happen if this were a transfer function of a system that was created with physical circuits. In this case, it is very unlikely that the pole and zero would remain in exactly the same place. A minor temperature change, for instance, could cause one of them to move just slightly. If this were to occur a tremendous amount of volatility is created in that area, since there is a change from infinity at the pole to zero at the zero in a very small range of signals. This is generally a very bad way to try to eliminate a pole. A much better way is to use **control theory** to move the pole to a better place.

#### Note: Repeated Poles and Zeros

It is possible to have more than one pole or zero at any given point. For instance, the discrete-time transfer function  $H(z) = z^2$  will have two zeros at the origin and the continuous-time function  $H(s) = \frac{1}{s^{25}}$  will have 25 poles at the origin.

**MATLAB** - If access to MATLAB is readily available, then you can use its functions to easily create pole/zero plots. Below is a short program that plots the poles and zeros from the above example onto the Z-Plane.

```
% Set up vector for zeros
z = [j ; -j];

% Set up vector for poles
p = [-1 ; .5+.5j ; .5-.5j];

figure(1);
zplane(z,p);
title('Pole/Zero Plot for Complex Pole/Zero Plot Example');
```

## Interactive Demonstration of Poles and Zeros

### Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 5.5.4: Interact (when online) with a Mathematica CDF demonstrating Pole/Zero Plots. To Download, right-click and save target as .cdf.

## Applications for pole-zero plots

### Stability and Control theory

Now that we have found and plotted the poles and zeros, we must ask what it is that this plot gives us. Basically what we can gather from this is that the magnitude of the transfer function will be larger when it is closer to the poles and smaller when it is closer to the zeros. This provides us with a qualitative understanding of what the system does at various frequencies and is crucial to the discussion of stability (Section 3.6).

### Pole/Zero Plots and the Region of Convergence

The region of convergence (ROC) for  $X(z)$  in the complex Z-plane can be determined from the pole/zero plot. Although several regions of convergence may be possible, where each one corresponds to a different impulse response, there are some choices that are more practical. A ROC can be chosen to make the transfer function causal and/or stable depending on the pole/zero plot.

Filter Properties from ROC

- If the ROC extends outward from the outermost pole, then the system is **causal**.
- If the ROC includes the unit circle, then the system is **stable**.

Below is a pole/zero plot with a possible ROC of the Z-transform in the Simple Pole/Zero Plot (Example 5.5.2) discussed earlier. The shaded region indicates the ROC chosen for the filter. From this figure, we can see that the filter will be both causal and stable since the above listed conditions are both met.

#### Example 5.5.5

$$H(z) = \frac{z}{\left(z - \frac{1}{2}\right)\left(z + \frac{3}{4}\right)}$$

### Region of Convergence for the Pole/Zero Plot

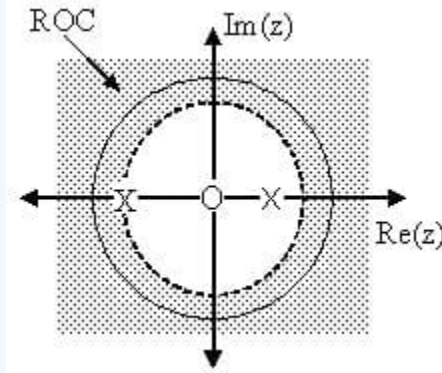


Figure 5.5.5: The shaded area represents the chosen ROC for the transfer function.

## Frequency Response and Pole/Zero Plots

The reason it is helpful to understand and create these pole/zero plots is due to their ability to help us easily design a filter. Based on the location of the poles and zeros, the magnitude response of the filter can be quickly understood. Also, by starting with the pole/zero plot, one can design a filter and obtain its transfer function very easily.

This page titled [5.5: Poles and Zeros in the Z-Plane](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.5: Poles and Zeros in the Z-Plane](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.6: Region of Convergence for the Z-Transform

### Introduction

With the [z-transform](#), the s-plane represents a set of signals (complex exponentials (Section 1.8)). For any given LTI (Section 2.1) system, some of these signals may cause the output of the system to converge, while others cause the output to diverge ("blow up"). The set of signals that cause the system's output to converge lie in the **region of convergence (ROC)**. This module will discuss how to find this region of convergence for any discrete-time, LTI system.

### The Region of Convergence

The region of convergence, known as the **ROC**, is important to understand because it defines the region where the [z-transform](#) exists. The **z-transform** of a sequence is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

The ROC for a given  $x[n]$ , is defined as the range of  $z$  for which the z-transform converges. Since the z-transform is a **power series**, it converges when  $x[n]z^{-n}$  is absolutely summable. Stated differently,

$$\sum_{n=-\infty}^{\infty} |x[n]z^{-n}| < \infty$$

must be satisfied for convergence.

### Properties of the Region of Convergence

The Region of Convergence has a number of properties that are dependent on the characteristics of the signal,  $x[n]$ .

- **The ROC cannot contain any poles.** By definition a pole is a where  $X(z)$  is infinite. Since  $X(z)$  must be finite for all  $z$  for convergence, there cannot be a pole in the ROC.
- **If  $x[n]$  is a finite-duration sequence, then the ROC is the entire z-plane, except possibly  $z = 0$  or  $|z| = \infty$ .** A **finite-duration sequence** is a sequence that is nonzero in a finite interval  $n_1 \leq n \leq n_2$ . As long as each value of  $x[n]$  is finite then the sequence will be absolutely summable. When  $n_2 > 0$  there will be a  $z^{-1}$  term and thus the ROC will not include  $z = 0$ . When  $n_1 < 0$  then the sum will be infinite and thus the ROC will not include  $|z| = \infty$ . On the other hand, when  $n_2 \leq 0$  then the ROC will include  $z = 0$ , and when  $n_1 \geq 0$  the ROC will include  $|z| = \infty$ . With these constraints, the only signal, then, whose ROC is the entire z-plane is  $x[n] = c\delta[n]$ .

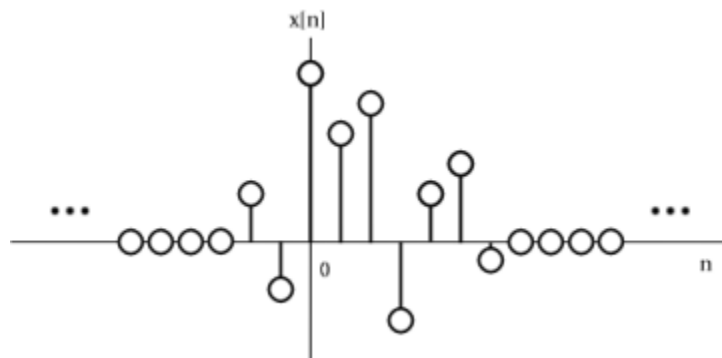


Figure 5.6.1: An example of a finite duration sequence.

The next properties apply to infinite duration sequences. As noted above, the z-transform converges when  $|X(z)| < \infty$ . So we can write

$$|X(z)| = \left| \sum_{n=-\infty}^{\infty} x[n]z^{-n} \right| \leq \sum_{n=-\infty}^{\infty} |x[n]z^{-n}| = \sum_{n=-\infty}^{\infty} |x[n]|(|z|)^{-n}$$

We can then split the infinite sum into positive-time and negative-time portions. So

$$|X(z)| \leq N(z) + P(z)$$

where

$$N(z) = \sum_{n=-\infty}^{-1} |x[n]|(|z|)^{-n}$$

and

$$P(z) = \sum_{n=0}^{\infty} |x[n]|(|z|)^{-n}$$

In order for  $|X(z)|$  to be finite,  $|x[n]|$  must be bounded. Let us then set

$$|x(n)| \leq C_1 r_1^n$$

for

$$n < 0$$

and

$$|x(n)| \leq C_2 r_2^n$$

for

$$n \geq 0$$

From this some further properties can be derived:

- If  $x[n]$  is a **right-sided sequence**, then the **ROC extends outward from the outermost pole in  $X(z)$** . A **right-sided sequence** is a sequence where  $x[n] = 0$  for  $\forall n$

$$P(z) \leq C_2 \sum_{n=0}^{\infty} r_2^n (|z|)^{-n} = C_2 \sum_{n=0}^{\infty} \left( \frac{r_2}{|z|} \right)^n$$

Thus in order for this sum to converge,  $|z| > r_2$ , and therefore the ROC of a right-sided sequence is of the form  $|z| > r_2$ .

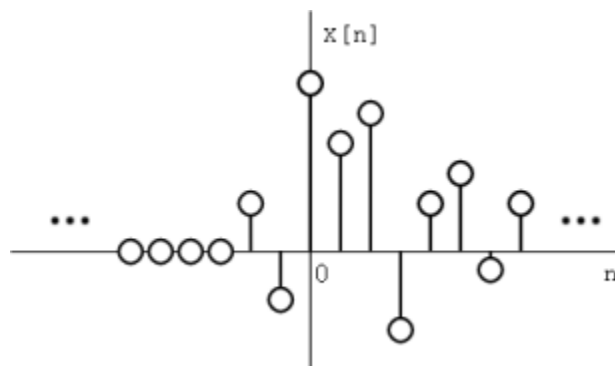


Figure 5.6.2: A right-sided sequence.

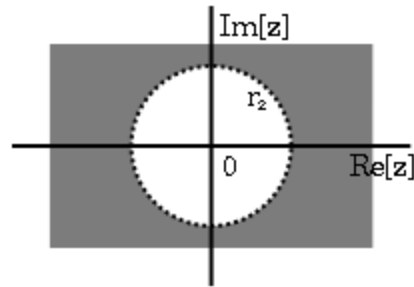


Figure 5.6.3: The ROC of a right-sided sequence.

- If  $x[n]$  is a **left-sided sequence**, then the ROC extends inward from the innermost pole in  $X(z)$ . A **left-sided sequence** is a sequence where  $x[n] = 0$  for  $n > n_2 > -\infty$ . Looking at the negative-time portion from the above derivation, it follows that

$$N(z) \leq C_1 \sum_{n=-\infty}^{-1} r_1^n (|z|)^{-n} = C_1 \sum_{n=-\infty}^{-1} \left( \frac{r_1}{|z|} \right)^n = C_1 \sum_{k=1}^{\infty} \left( \frac{|z|}{r_1} \right)^k$$

Thus in order for this sum to converge,  $|z| < r_1$

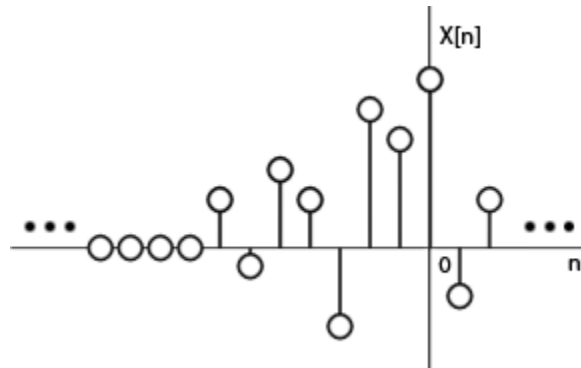


Figure 5.6.4: A left-sided sequence.

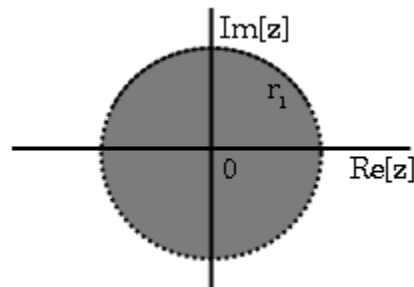


Figure 5.6.5: The ROC of a left-sided sequence.

- If  $x[n]$  is a **two-sided sequence**, the ROC will be a ring in the z-plane that is bounded on the interior and exterior by a pole. A **two-sided sequence** is an sequence with infinite duration in the positive and negative directions. From the derivation of the above two properties, it follows that if  $-r_2 < |z| < r_1$

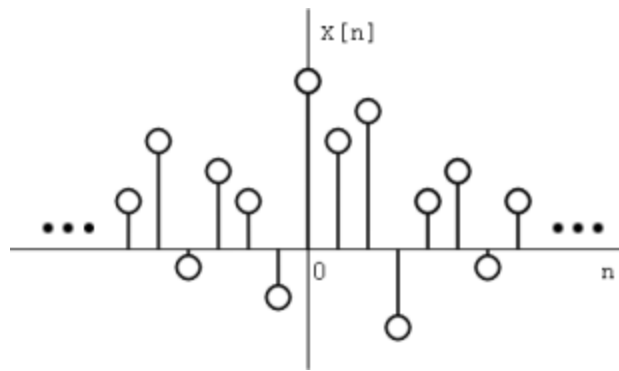


Figure 5.6.6: A two-sided sequence.

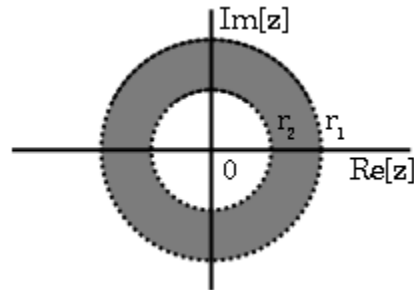


Figure 5.6.7: The ROC of a two-sided sequence.

## Examples

### Example 5.6.1

Let's take

$$x_1[n] = \left(\frac{1}{2}\right)^n u[n] + \left(\frac{1}{4}\right)^n u[n]$$

The z-transform of  $\left(\frac{1}{2}\right)^n u[n]$  is  $\frac{z}{z-\frac{1}{2}}$  with an ROC at  $|z| > \frac{1}{2}$ .

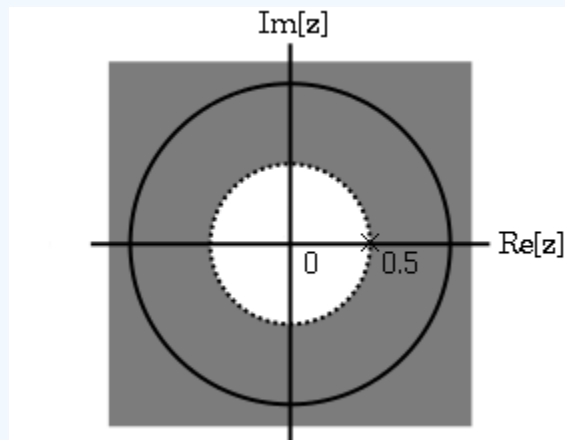


Figure 5.6.8: The ROC of  $\left(\frac{1}{2}\right)^n u[n]$

The z-transform of  $\left(\frac{-1}{4}\right)^n u[n]$  is  $\frac{z}{z+\frac{1}{4}}$  with an ROC at  $|z| > \frac{-1}{4}$ .

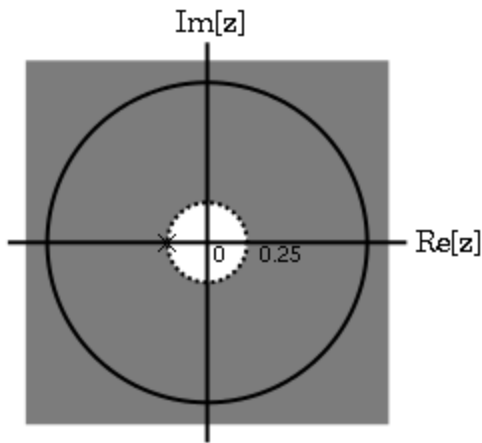


Figure 5.6.9: The ROC of  $(\frac{-1}{4})^n u[n]$

Due to linearity,

$$\begin{aligned} X_1[z] &= \frac{z}{z - \frac{1}{2}} + \frac{z}{z + \frac{1}{4}} \\ &= \frac{2z(z - \frac{1}{8})}{(z - \frac{1}{2})(z + \frac{1}{4})} \end{aligned} \quad (5.6.1)$$

By observation it is clear that there are two zeros, at 0 and  $\frac{1}{8}$ , and two poles, at  $\frac{1}{2}$ , and  $\frac{-1}{4}$ . Following the above properties, the ROC is  $|z| > \frac{1}{2}$ .

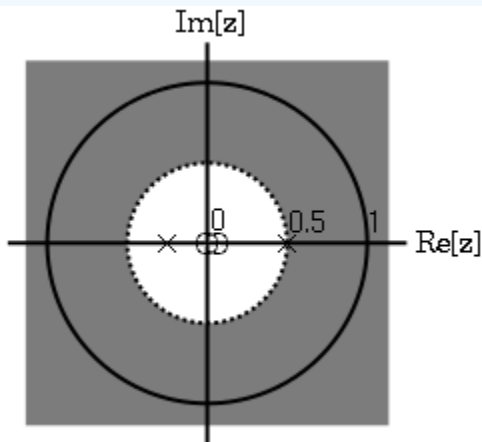


Figure 5.6.10: The ROC of  $x_1[n] = (\frac{1}{2})^n u[n] + (\frac{-1}{4})^n u[n]$

### Example 5.6.2

Now take

$$x_2[n] = \left(\frac{-1}{4}\right)^n u[n] - \left(\frac{1}{2}\right)^n u[(-n) - 1]$$

The z-transform and ROC of  $(\frac{-1}{4})^n u[n]$  was shown in the example above. The z-transform of  $(-\frac{1}{2})^n u[(-n) - 1]$  is  $\frac{z}{z - \frac{1}{2}}$  with an ROC at  $|z| > \frac{1}{2}$ .

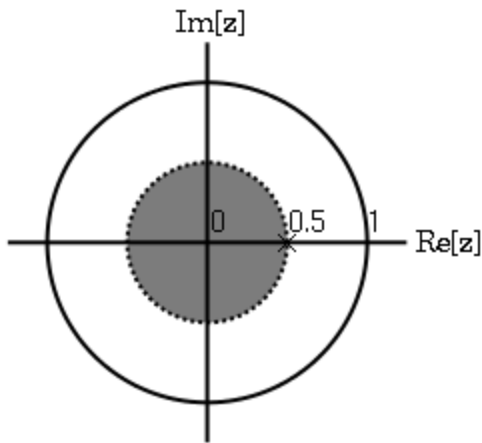


Figure 5.6.11: The ROC of  $(-\frac{1}{2})^n u[(-n) - 1]$

Once again, by linearity,

$$\begin{aligned} X_2[z] &= \frac{z}{z + \frac{1}{4}} + \frac{z}{z - \frac{1}{2}} \\ &= \frac{z(2z - \frac{1}{8})}{(z + \frac{1}{4})(z - \frac{1}{2})} \end{aligned} \quad (5.6.2)$$

By observation it is again clear that there are two zeros, at 0 and  $\frac{1}{16}$ , and two poles, at  $\frac{1}{2}$ , and  $-\frac{1}{4}$ . In this case though, the ROC is  $|z| < \frac{1}{2}$ .

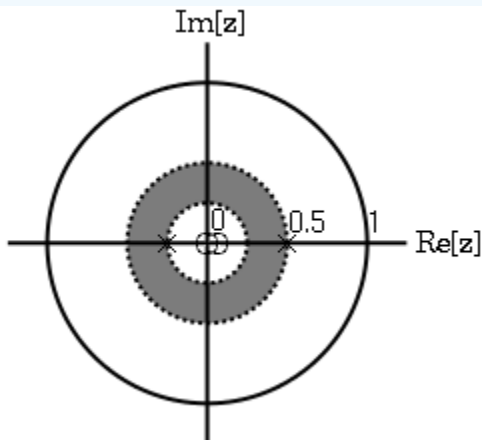


Figure 5.6.12: The ROC of  $x_2[n] = (\frac{-1}{4})^n u[n] - (\frac{1}{2})^n u[(-n) - 1]$ .

## Graphical Understanding of ROC

Using the demonstration, learn about the region of convergence for the Laplace Transform.

## Conclusion

Clearly, in order to craft a system that is actually useful by virtue of being causal and BIBO stable, we must ensure that it is within the Region of Convergence, which can be ascertained by looking at the pole zero plot. The Region of Convergence is the area in the pole/zero plot of the transfer function in which the function exists. For purposes of useful filter design, we prefer to work with rational functions, which can be described by two polynomials, one each for determining the poles and the zeros, respectively.

---

This page titled [5.6: Region of Convergence for the Z-Transform](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.6: Region of Convergence for the Z-Transform](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.7: Rational Functions and the Z-Transform

### Introduction

When dealing with operations on polynomials, the term **rational function** is a simple way to describe a particular relationship between two polynomials.

#### Definition: Rational Function

For any two polynomials, A and B, their quotient is called a rational function.

#### Example 5.7.1

Below is a simple example of a basic rational function,  $f(x)$ . Note that the numerator and denominator can be polynomials of any order, but the rational function is undefined when the denominator equals zero.

$$f(x) = \frac{x^2 - 4}{2x^2 + x - 3} \quad (5.7.1)$$

If you have begun to study the [Z-transform](#), you should have noticed by now they are all rational functions. Below we will look at some of the properties of rational functions and how they can be used to reveal important characteristics about a z-transform, and thus a signal or LTI system.

### Properties of Rational Functions

In order to see what makes rational functions special, let us look at some of their basic properties and characteristics. If you are familiar with rational functions and basic algebraic properties, skip to the next subsection to see how rational functions are useful when dealing with the z-transform.

### Roots

To understand many of the following characteristics of a rational function, one must begin by finding the roots of the rational function. In order to do this, let us factor both of the polynomials so that the roots can be easily determined. Like all polynomials, the roots will provide us with information on many key properties. The function below shows the results of factoring the above rational function, Equation 5.7.1.

$$f(x) = \frac{(x + 2)(x - 2)}{(2x + 3)(x - 1)} \quad (5.7.2)$$

Thus, the roots of the rational function are as follows:

Roots of the numerator are:  $\{-2, 2\}$

Roots of the denominator are:  $\{-3, 1\}$

#### Note

In order to understand rational functions, it is essential to know and understand the roots that make up the rational function.

## Discontinuities

Because we are dealing with division of two polynomials, we must be aware of the values of the variable that will cause the denominator of our fraction to be zero. When this happens, the rational function becomes undefined, i.e. we have a discontinuity in the function. Because we have already solved for our roots, it is very easy to see when this occurs. When the variable in the denominator equals any of the roots of the denominator, the function becomes undefined.

### Example 5.7.2

Continuing to look at our rational function above, Equation 5.7.1, we can see that the function will have discontinuities at the following points:  $x = \{-3, 1\}$

In respect to the Cartesian plane, we say that the discontinuities are the values along the x-axis where the function is undefined. These discontinuities often appear as **vertical asymptotes** on the graph to represent the values where the function is undefined.

## Domain

Using the roots that we found above, the **domain** of the rational function can be easily defined.

### Definition: Domain

The group, or set, of values that are defined by a given function.

### Example 5.7.3

Using the rational function above, Equation 5.7.1, the domain can be defined as any real number  $x$  where  $x$  does not equal 1 or negative 3. Written out mathematical, we get the following:

$$\{x \in \mathbb{R} \mid (x \neq -3) \text{ and } (x \neq 1)\} \quad (5.7.3)$$

## Intercepts

The **x-intercept** is defined as the point(s) where  $f(x)$ , i.e. the output of the rational functions, equals zero. Because we have already found the roots of the equation this process is very simple. From algebra, we know that the output will be zero whenever the numerator of the rational function is equal to zero. Therefore, the function will have an x-intercept wherever  $x$  equals one of the roots of the numerator.

The **y-intercept** occurs whenever  $x$  equals zero. This can be found by setting all the values of  $x$  equal to zero and solving the rational function.

## Rational Functions and the Z-Transform

As we have stated above, all z-transforms can be written as rational functions, which have become the most common way of representing the z-transform. Because of this, we can use the properties above, especially those of the roots, in order to reveal certain characteristics about the signal or LTI system described by the z-transform.

Below is the general form of the z-transform written as a rational function:

$$X(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \dots + a_N z^{-N}} \quad (5.7.4)$$

If you have already looked at the module about Understanding Pole/Zero Plots and the Z-transform (Section 12.5), you should see

how the roots of the rational function play an important role in understanding the z-transform. The equation above, Equation 5.7.4, can be expressed in factored form just as was done for the simple rational function above, see Equation 5.7.2. Thus, we can easily find the roots of the numerator and denominator of the z-transform. The following two relationships become apparent:

#### Relationship of Roots to Poles and Zeros

- The roots of the numerator in the rational function will be the **zeros** of the z-transform
- The roots of the denominator in the rational function will be the **poles** of the z-transform

## Conclusion

Once we have used our knowledge of rational functions to find its roots, we can manipulate a z-transform in a number of useful ways. We can apply this knowledge to representing an LTI system graphically through a Pole/Zero Plot (Section 12.5), or to analyze and design a digital filter through Filter Design from the Z-Transform (Section 12.9).

---

This page titled [5.7: Rational Functions and the Z-Transform](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.

- [12.7: Rational Functions and the Z-Transform](#) by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.8: Difference Equations

### Introduction

One of the most important concepts of DSP is to be able to properly represent the input/output relationship to a given LTI system. A linear constant-coefficient **difference equation** (LCCDE) serves as a way to express just this relationship in a discrete-time system. Writing the sequence of inputs and outputs, which represent the characteristics of the LTI system, as a difference equation help in understanding and manipulating a system.

#### Definition: Difference Equation

An equation that shows the relationship between consecutive values of a sequence and the differences among them. They are often rearranged as a recursive formula so that a systems output can be computed from the input signal and past outputs.

#### Example 5.8.1

$$y[n] + 7y[n-1] + 2y[n-2] = x[n] - 4x[n-1]$$

### General Formulas for the Difference Equation

As stated briefly in the definition above, a difference equation is a very useful tool in describing and calculating the output of the system described by the formula for a given sample  $n$ . The key property of the difference equation is its ability to help easily find the transform,  $H(z)$ , of a system. In the following two subsections, we will look at the general form of the difference equation and the general conversion to a z-transform directly from the difference equation.

### Difference Equation

The general form of a linear, constant-coefficient difference equation (LCCDE), is shown below:

$$\sum_{k=0}^N a_k y[n-k] = \sum_{k=0}^M b_k x[n-k] \quad (5.8.1)$$

We can also write the general form to easily express a recursive output, which looks like this:

$$y[n] = -\sum_{k=1}^N a_k y[n-k] + \sum_{k=0}^M b_k x[n-k] \quad (5.8.2)$$

From this equation, note that  $y[n-k]$  represents the outputs and  $x[n-k]$  represents the inputs. The value of  $N$  represents the **order** of the difference equation and corresponds to the memory of the system being represented. Because this equation relies on past values of the output, in order to compute a numerical solution, certain past outputs, referred to as the **initial conditions**, must be known.

### Conversion to Z-Transform

Using the above formula, Equation 5.8.2, we can easily generalize the **transfer function**,  $H(z)$ , for any difference equation. Below are the steps taken to convert any difference equation into its transfer function, i.e. z-transform. The first step involves taking the Fourier Transform of all the terms in Equation 5.8.2. Then we use the linearity property to pull the transform inside the summation and the time-shifting property of the z-transform to change the time-shifting terms to exponentials. Once this is done, we arrive at the following equation:  $a_0 = 1$ .

$$Y(z) = - \sum_{k=1}^N a_k Y(z) z^{-k} + \sum_{k=0}^M b_k X(z) z^{-k}$$

$$H(z) = \frac{Y(z)}{X(z)}$$

$$= \frac{\sum_{k=0}^M b_k z^{-k}}{1 + \sum_{k=1}^N a_k z^{-k}} \quad (5.8.3)$$

## Conversion to Frequency Response

Once the z-transform has been calculated from the difference equation, we can go one step further to define the frequency response of the system, or filter, that is being represented by the difference equation.

### Note

Remember that the reason we are dealing with these formulas is to be able to aid us in filter design. A LCCDE is one of the easiest ways to represent FIR filters. By being able to find the frequency response, we will be able to look at the basic properties of any filter represented by a simple LCCDE.

Below is the general formula for the frequency response of a z-transform. The conversion is simple a matter of taking the z-transform formula,  $H(z)$ , and replacing every instance of  $z$  with  $e^{jw}$ .

$$H(w) = H(z)|_{z=e^{jw}} \quad (5.8.4)$$

$$= \frac{\sum_{k=0}^M b_k e^{-(jwk)}}{\sum_{k=0}^N a_k e^{-(jwk)}} \quad (5.8.5)$$

Once you understand the derivation of this formula, look at the module concerning Filter Design from the Z-Transform (Section 12.9) for a look into how all of these ideas of the [Z-transform](#), Difference Equation, and Pole/Zero Plots (Section 12.5) play a role in filter design.

## Example

### Example 5.8.2: Finding Difference Equation

Below is a basic example showing the opposite of the steps above: given a transfer function one can easily calculate the systems difference equation.

$$H(z) = \frac{(z+1)^2}{(z-\frac{1}{2})(z+\frac{3}{4})}$$

Given this transfer function of a time-domain filter, we want to find the difference equation. To begin with, expand both polynomials and divide them by the highest order  $z$ .

$$H(z) = \frac{(z+1)(z+1)}{(z-\frac{1}{2})(z+\frac{3}{4})}$$

$$= \frac{z^2 + 2z + 1}{z^2 + 2z + 1 - \frac{3}{8}}$$

$$= \frac{1 + 2z^{-1} + z^{-2}}{1 + \frac{1}{4}z^{-1} - \frac{3}{8}z^{-2}} \quad (5.8.6)$$

From this transfer function, the coefficients of the two polynomials will be our  $a_k$  and  $b_k$  values found in the general difference equation formula, Equation 5.8.2. Using these coefficients and the above form of the transfer function, we can easily write the difference equation:

$$x[n] + 2x[n-1] + x[n-2] = y[n] + \frac{1}{4}y[n-1] - \frac{3}{8}y[n-2]$$

In our final step, we can rewrite the difference equation in its more common form showing the recursive nature of the system.

$$y[n] = x[n] + 2x[n-1] + x[n-2] + \frac{-1}{4}y[n-1] + \frac{3}{8}y[n-2]$$

## Solving a LCCDE

In order for a linear constant-coefficient difference equation to be useful in analyzing a LTI system, we must be able to find the systems output based upon a known input,  $x(n)$ , and a set of initial conditions. Two common methods exist for solving a LCCDE: the **direct method** and the **indirect method**, the later being based on the z-transform. Below we will briefly discuss the formulas for solving a LCCDE using each of these methods.

### Direct Method

The final solution to the output based on the direct method is the sum of two parts, expressed in the following equation:

$$y(n) = y_h(n) + y_p(n)$$

The first part,  $y_h(n)$ , is referred to as the **homogeneous solution** and the second part,  $y_p(n)$ , is referred to as **particular solution**. The following method is very similar to that used to solve many differential equations, so if you have taken a differential calculus course or used differential equations before then this should seem very familiar.

### Homogeneous Solution

We begin by assuming that the input is zero,  $x(n) = 0$ . Now we simply need to solve the homogeneous difference equation:

$$\sum_{k=0}^N a_k y[n-k] = 0$$

In order to solve this, we will make the assumption that the solution is in the form of an exponential. We will use lambda,  $\lambda$ , to represent our exponential terms. We now have to solve the following equation:

$$\sum_{k=0}^N a_k \lambda^{n-k} = 0$$

We can expand this equation out and factor out all of the lambda terms. This will give us a large polynomial in parenthesis, which is referred to as the **characteristic polynomial**. The roots of this polynomial will be the key to solving the homogeneous equation. If there are all distinct roots, then the general solution to the equation will be as follows:

$$y_h(n) = C_1 (\lambda_1)^n + C_2 (\lambda_2)^n + \dots + C_N (\lambda_N)^n$$

However, if the characteristic equation contains multiple roots then the above general solution will be slightly different. Below we have the modified version for an equation where  $\lambda_1$  has  $K$  multiple roots:

$$y_h(n) = C_1 (\lambda_1)^n + C_1 n (\lambda_1)^n + C_1 n^2 (\lambda_1)^n + \dots + C_1 n^{K-1} (\lambda_1)^n + C_2 (\lambda_2)^n + \dots + C_N (\lambda_N)^n$$

### Particular Solution

The particular solution,  $y_p(n)$ , will be any solution that will solve the general difference equation:

$$\sum_{k=0}^N a_k y_p(n-k) = \sum_{k=0}^M b_k x(n-k)$$

In order to solve, our guess for the solution to  $y_p(n)$  will take on the form of the input,  $x(n)$ . After guessing at a solution to the above equation involving the particular solution, one only needs to plug the solution into the difference equation and solve it out.

## Indirect Method

The indirect method utilizes the relationship between the difference equation and z-transform, discussed earlier, to find a solution. The basic idea is to convert the difference equation into a z-transform, as described above, to get the resulting output,  $Y(z)$ . Then by inverse transforming this and using partial-fraction expansion, we can arrive at the solution.

$$Z\{y(n+1) - y(n)\} = zY(z) - y(0)$$

This can be iteratively extended to an arbitrary order derivative as in Equation 5.8.7.

$$Z\left\{-\sum_{m=0}^{N-1} y(n-m)\right\} = z^n Y(z) - \sum_{m=0}^{N-1} z^{n-m-1} y^{(m)}(0) \quad (5.8.7)$$

Now, the Laplace transform of each side of the differential equation can be taken

$$Z\left\{\sum_{k=0}^N a_k \left[y(n-m+1) - \sum_{m=0}^{N-1} y(n-m)y(n)\right]\right\} = Z\{x(n)\}$$

which by linearity results in

$$\sum_{k=0}^N a_k Z\left\{y(n-m+1) - \sum_{m=0}^{N-1} y(n-m)y(n)\right\} = Z\{x(n)\}$$

and by differentiation properties in

$$\sum_{k=0}^N a_k \left(z^k Z\{y(n)\} - \sum_{m=0}^{N-1} z^{k-m-1} y^{(m)}(0)\right) = Z\{x(n)\}.$$

Rearranging terms to isolate the Laplace transform of the output,

$$Z\{y(n)\} = \frac{Z\{x(n)\} + \sum_{k=0}^N \sum_{m=0}^{k-1} a_k z^{k-m-1} y^{(m)}(0)}{\sum_{k=0}^N a_k z^k}.$$

Thus, it is found that

$$Y(z) = \frac{X(z) + \sum_{k=0}^N \sum_{m=0}^{k-1} a_k z^{k-m-1} y^{(m)}(0)}{\sum_{k=0}^N a_k z^k}. \quad (5.8.8)$$

In order to find the output, it only remains to find the Laplace transform  $X(z)$  of the input, substitute the initial conditions, and compute the inverse Z-transform of the result. Partial fraction expansions are often required for this last step. This may sound daunting while looking at Equation 5.8.8, but it is often easy in practice, especially for low order difference equations. Equation 5.8.8 can also be used to determine the transfer function and frequency response.

As an example, consider the difference equation

$$y[n-2] + 4y[n-1] + 3y[n] = \cos(n)$$

with the initial conditions  $y'(0) = 1$  and  $y(0) = 0$  Using the method described above, the Z transform of the solution  $y[n]$  is given by

$$Y[z] = \frac{z}{[z^2 + 1][z + 1][z + 3]} + \frac{1}{[z + 1][z + 3]}.$$

Performing a partial fraction decomposition, this also equals

$$Y[z] = .25 \frac{1}{z+1} - .35 \frac{1}{z+3} + .1 \frac{z}{z^2+1} + .2 \frac{1}{z^2+1}.$$

Computing the inverse Laplace transform,

$$y(n) = (.25z^{-n} - .35z^{-3n} + .1 \cos(n) + .2 \sin(n)) u(n).$$

One can check that this satisfies that this satisfies both the differential equation and the initial conditions.

---

This page titled [5.8: Difference Equations](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.8: Difference Equations](#) by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 5.9: Discrete Time Filter Design

### Estimating Frequency Response from Z-Plane

One of the primary motivating factors for utilizing the z-transform and analyzing the pole/zero plots is due to their relationship to the frequency response of a discrete-time system. Based on the position of the poles and zeros, one can quickly determine the frequency response. This is a result of the correspondence between the frequency response and the transfer function evaluated on the unit circle in the pole/zero plots. The frequency response, or DTFT, of the system is defined as:

$$H(w) = H(z)|_{z=e^{jw}} = \frac{\sum_{k=0}^M b_k e^{-(jwk)}}{\sum_{k=0}^N a_k e^{-(jwk)}} \quad (5.9.1)$$

Next, by factoring the transfer function into poles and zeros and multiplying the numerator and denominator by  $e^{jw}$  we arrive at the following equations:

$$H(w) = \left| \frac{b_0}{a_0} \right| \frac{\prod_{k=1}^M |e^{jw} - c_k|}{\prod_{k=1}^N |e^{jw} - d_k|} \quad (5.9.2)$$

From Equation 5.9.2 we have the frequency response in a form that can be used to interpret physical characteristics about the filter's frequency response. The numerator and denominator contain a product of terms of the form  $|e^{jw} - h|$ , where  $h$  is either a zero, denoted by  $c_k$  or a pole, denoted by  $d_k$ . Vectors are commonly used to represent the term and its parts on the complex plane. The pole or zero,  $h$ , is a vector from the origin to its location anywhere on the complex plane and  $e^{jw}$  is a vector from the origin to its location on the unit circle. The vector connecting these two points,  $|e^{jw} - h|$ , connects the pole or zero location to a place on the unit circle dependent on the value of  $w$ . From this, we can begin to understand how the magnitude of the frequency response is a ratio of the distances to the poles and zero present in the z-plane as  $w$  goes from zero to  $\pi$ . These characteristics allow us to interpret  $|H(w)|$  as follows:

$$|H(w)| = \left| \frac{b_0}{a_0} \right| \frac{\prod \text{"distances from zeros"}}{\prod \text{"distances from poles"}}$$

### Drawing Frequency Response from Pole/Zero Plot

Let us now look at several examples of determining the magnitude of the frequency response from the pole/zero plot of a z-transform. If you have forgotten or are unfamiliar with pole/zero plots, please refer back to the Pole/Zero Plots (Section 12.5) module.

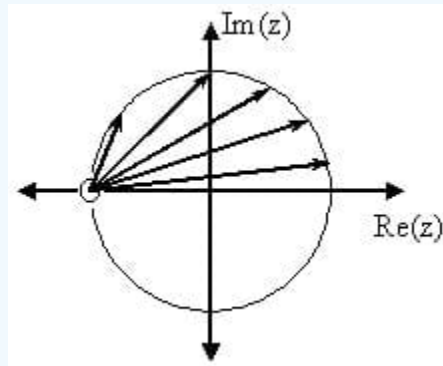
#### Example 5.9.1

In this first example we will take a look at the very simple z-transform shown below:

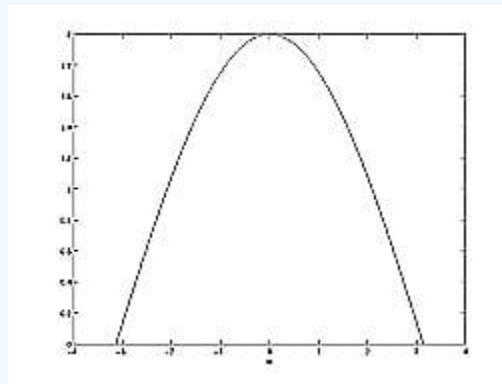
$$H(z) = z + 1 = 1 + z^{-1}$$

$$H(w) = 1 + e^{-(jw)}$$

For this example, some of the vectors represented by  $|e^{jw} - h|$ , for random values of  $w$ , are explicitly drawn onto the complex plane shown in Figure 5.9.1 below. These vectors show how the amplitude of the frequency response changes as  $w$  goes from 0 to  $2\pi$ , and also show the physical meaning of the terms in Equation 5.9.2 above. One can see that when  $w = 0$ , the vector is the longest and thus the frequency response will have its largest amplitude here. As  $w$  approaches  $\pi$ , the length of the vectors decrease as does the amplitude of  $|H(w)|$ . Since there are no poles in the transform, there is only this one vector term rather than a ratio as seen in Equation 5.9.2.



(a) Pole/Zero Plot



(b) Frequency Response:  
 $|H(w)|$

Figure 5.9.1: The first figure represents the pole/zero plot with a few representative vectors graphed while the second shows the frequency response with a peak at  $+2$  and graphed between plus and minus  $\pi$ .

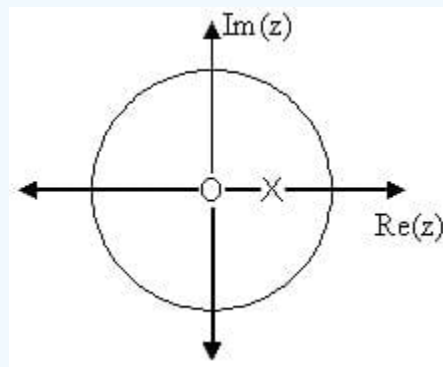
### Example 5.9.2

For this example, a more complex transfer function is analyzed in order to represent the system's frequency response.

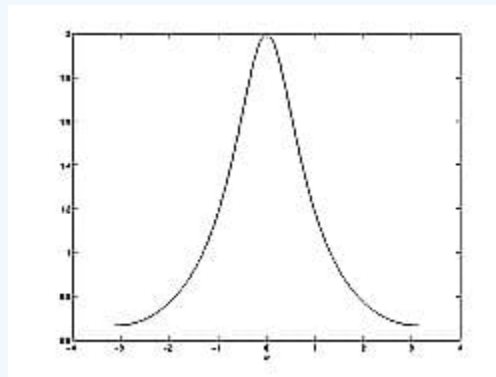
$$H(z) = \frac{z}{z - \frac{1}{2}} = \frac{1}{1 - \frac{1}{2}z^{-1}}$$

$$H(w) = \frac{1}{1 - \frac{1}{2}e^{-jw}}$$

Below we can see the two figures described by the above equations. The Figure 5.9.2(a) represents the basic pole/zero plot of the z-transform,  $H(w)$ . Figure 5.9.2(b) shows the magnitude of the frequency response. From the formulas and statements in the previous section, we can see that when  $w = 0$  the frequency will peak since it is at this value of  $w$  that the pole is closest to the unit circle. The ratio from Equation 5.9.2 helps us see the mathematics behind this conclusion and the relationship between the distances from the unit circle and the poles and zeros. As  $w$  moves from 0 to  $\pi$ , we see how the zero begins to mask the effects of the pole and thus force the frequency response closer to 0.



(a) Pole/Zero Plot



(b) Frequency Response:  
 $|H(w)|$

Figure 5.9.2: The first figure represents the pole/zero plot while the second shows the frequency response with a peak at  $+2$  and graphed between plus and minus  $\pi$ .

## Interactive Filter Design Illustration

### Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 5.9.3: Digital filter design LabVIEW virtual instrument by NI from <http://cnx.org/content/m13115/latest/>.

## Conclusion

In conclusion, using the distances from the unit circle to the poles and zeros, we can plot the frequency response of the system. As  $w$  goes from  $0$  to  $2\pi$ , the following two properties, taken from the above equations, specify how one should draw  $|H(w)|$ .

While moving around the unit circle...

1. if close to a zero, then the magnitude is small. If a zero is on the unit circle, then the frequency response is zero at that point.
2. if close to a pole, then the magnitude is large. If a pole is on the unit circle, then the frequency response goes to infinity at that point.

---

This page titled [5.9: Discrete Time Filter Design](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [12.9: Discrete Time Filter Design](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6: DISCRETE TIME FOURIER SERIES (DTFS)

6.1: Discrete Time Periodic Signals

6.2: Discrete Time Fourier Series (DTFS)

6.3: Common Discrete Fourier Series

6.4: Properties of the DTFS

6.5: Discrete Time Circular Convolution and the DTFS

---

This page titled [6: Discrete Time Fourier Series \(DTFS\)](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.

- [7: Discrete Time Fourier Series \(DTFS\)](#) by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6.1: Discrete Time Periodic Signals

### Introduction

This module describes the type of signals acted on by the Discrete Time Fourier Series.

### Relevant Spaces

The Discrete Time Fourier Series maps finite-length (or  $N$ -periodic), discrete time signals in  $L^2$  to finite-length, discrete-frequency signals in  $l^2$ .

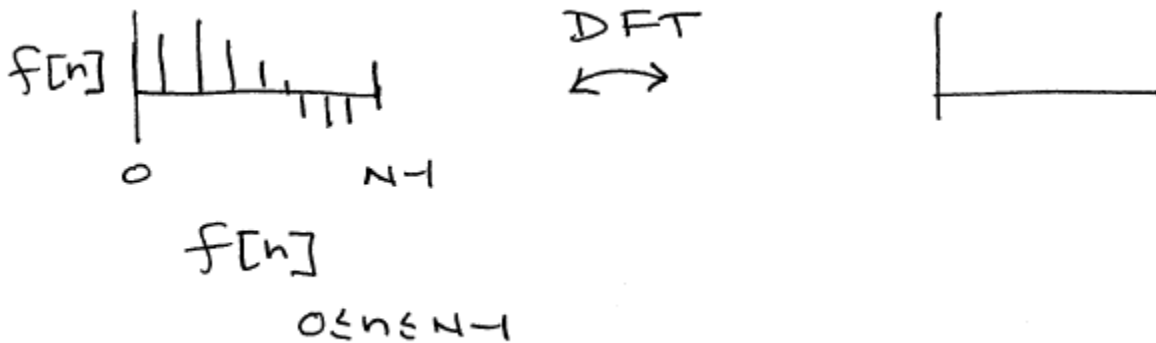


Figure 6.1.1

Periodic signals in discrete time repeats themselves in each cycle. However, only integers are allowed as time variable in discrete time. We denote signals in such case as  $x[n]$ ,  $n = \dots, -2, -1, 0, 1, 2, \dots$

### Periodic Signals

When a function repeats itself exactly after some given period, or cycle, we say it's **periodic**. A **periodic function** can be mathematically defined as:

$$f[n] = f[n + mN] \forall m : (m \in \mathbb{Z}) \quad (6.1.1)$$

where  $N > 0$  represents the **fundamental period** of the signal, which is the smallest positive value of  $N$  for the signal to repeat. Because of this, you may also see a signal referred to as an  $N$ -periodic signal. Any function that satisfies this equation is said to be **periodic** with period  $N$ . Here's an example of a **discrete-time periodic signal** with period  $N$ :

## discrete-time periodic signal

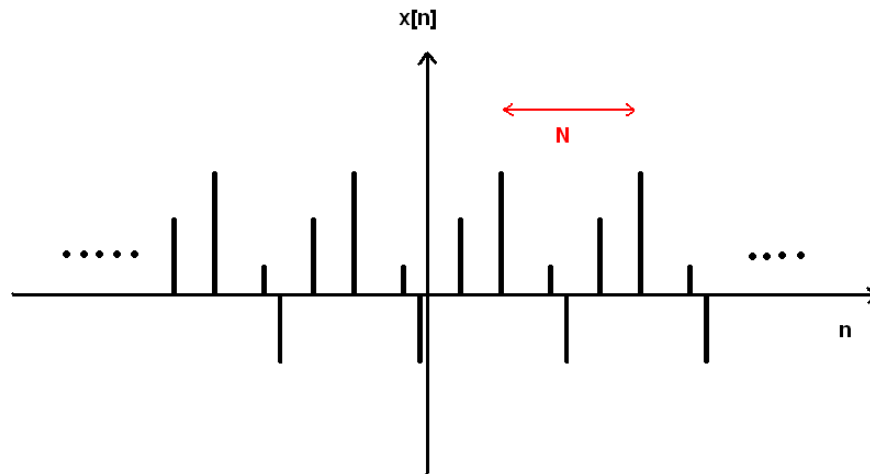


Figure 6.1.2: Notice the function is the same after a time shift of  $N$

We can think of **periodic functions** (with period  $N$ ) two different ways:

1. as functions on **all** of  $\mathbb{R}$

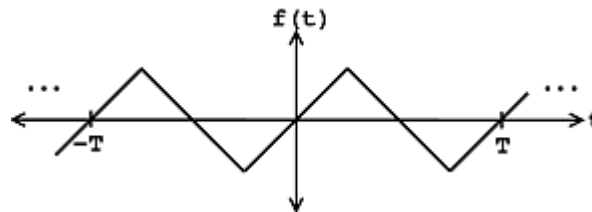


Figure 6.1.3: discrete time periodic function over all of  $\mathbb{R}$  where  $f[n_0] = f[n_0 + N]$

2. or, we can cut out all of the redundancy, and think of them as functions on an interval  $[0, N]$  (or, more generally,  $[a, a + N]$ ). If we know the signal is  $N$ -periodic then all the information of the signal is captured by the above interval.

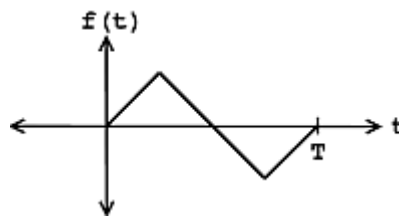


Figure 6.1.4: Remove the redundancy of the period function so that  $f[n]$  is undefined outside  $[0, N]$ .

An **aperiodic** DT function  $f[n]$  does not repeat for **any**  $N \in \mathbb{R}$ ; i.e. there exists no  $N$  such that Equation 6.1.1 holds.

## SinDrillDiscrete Demonstration

Here's an example demonstrating a **periodic** sinusoidal signal with various frequencies, amplitudes and phase delays:

## Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 6.1.5: Interact (when online) with a Mathematica CDF demonstrating a discrete periodic sinusoidal signal with various frequencies, amplitudes and phase delays.

## Conclusion

A discrete periodic signal is completely defined by its values in one period, such as the interval  $[0, N]$ .

This page titled [6.1: Discrete Time Periodic Signals](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [7.1: Discrete Time Periodic Signals](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6.2: Discrete Time Fourier Series (DTFS)

### Introduction

In this module, we will derive an expansion for discrete-time, periodic functions, and in doing so, derive the **Discrete Time Fourier Series** (DTFS), or the Discrete Fourier Transform (DFT).

### DTFS

#### Eigenfunction analysis

Since complex exponentials (Section 1.8) are eigenfunctions of linear time-invariant (LTI) systems (Section 14.5), calculating the output of an LTI system  $\mathcal{H}$  given  $e^{j\omega n}$  as an input amounts to simple multiplication, where  $\omega_0 = \frac{2\pi k}{N}$ , and where  $H[k] \in \mathbb{C}$  is the eigenvalue corresponding to  $k$ . As shown in the figure, a simple exponential input would yield the output

$$y[n] = H[k]e^{j\omega n}$$

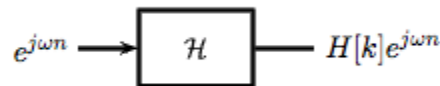


Figure 6.2.1: Simple LTI system.

Using this and the fact that  $\mathcal{H}$  is linear, calculating  $y[n]$  for combinations of **complex exponentials** is also straightforward.

$$\begin{aligned} c_1 e^{j\omega_1 n} + c_2 e^{j\omega_2 n} &\rightarrow c_1 H[k_1] e^{j\omega_1 n} + c_2 H[k_2] e^{j\omega_1 n} \\ \sum_l c_l e^{j\omega_l n} &\rightarrow \sum_l c_l H[k_l] e^{j\omega_l n} \end{aligned}$$

The action of  $H$  on an input such as those in the two equations above is easy to explain.  $\mathcal{H}$  **independently scales** each exponential component  $e^{j\omega_l n}$  by a different complex number  $H[k_l] \in \mathbb{C}$ . As such, if we can write a function  $y[n]$  as a combination of complex exponentials it allows us to easily calculate the output of a system.

#### DTFS synthesis

It can be demonstrated that an arbitrary Discrete Time-periodic function  $f[n]$  can be written as a linear combination of harmonic complex sinusoids

$$f[n] = \sum_{k=0}^{N-1} c_k e^{j\omega_0 k n} \quad (6.2.1)$$

where  $\omega_0 = \frac{2\pi}{N}$  is the fundamental frequency. For almost all  $f[n]$  of practical interest, there exists  $c_n$  to make Equation 6.2.1 true. If  $f[n]$  is finite energy ( $f[n] \in L^2[0, N]$ ), then the equality in Equation 6.2.1 holds in the sense of energy convergence; with discrete-time signals, there are no concerns for divergence as there are with continuous-time signals.

The  $c_n$  - called the Fourier coefficients - tell us "how much" of the sinusoid  $e^{j\omega_0 k n}$  is in  $f[n]$ . The formula shows  $f[n]$  as a sum of complex exponentials, each of which is easily processed by an LTI system (since it is an eigenfunction of every LTI system). Mathematically, it tells us that the set of complex exponentials  $\{e^{j\omega_0 k n}, k \in \mathbb{Z}\}$  form a basis for the space of N-periodic discrete time functions.

## DFT Synthesis Demonstration

### Embedded Object

This interactive element is not available in this format. Visit the link or scan the QR code to access it online.

[View this Embedded Object online](#)



Figure 6.2.2: [Download](#) or Interact (when online) with a Mathematica CDF demonstrating Discrete Harmonic Sinusoids. To download, right click and save as .cdf.

## DTFS Analysis

Say we have the following set of numbers that describe a periodic, discrete-time signal, where  $N = 4$ :

$$\{\dots, 3, 2, -2, 1, 3, \dots\}$$

Such a periodic, discrete-time signal (with period  $N$ ) can be thought of as a **finite** set of numbers. For example, we can represent this signal as either a periodic signal or as just a single interval as follows:

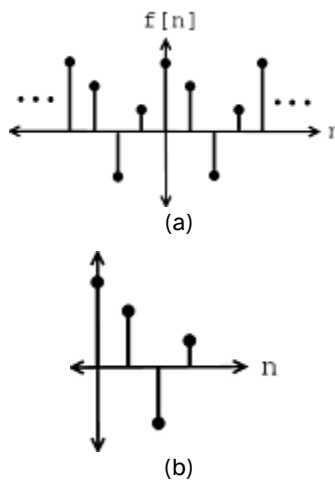


Figure 6.2.3: Here we can look at just one period of the signal that has a vector length of four and is contained in  $\mathbb{C}^4$ . (a) Periodic Function (b) Function on the interval  $[0, T]$

### Note

The cardinality of the set of discrete time signals with period  $N$  equals  $\mathbb{C}^N$ .

Here, we are going to form a basis using **harmonic sinusoids**. Before we look into this, it will be worth our time to look at the discrete-time, complex sinusoids in a little more detail.

## Complex Sinusoids

If you are familiar with the basic sinusoid signal and with complex exponentials (Section 1.8) then you should not have any problem understanding this section. In most texts, you will see the discrete-time, complex sinusoid noted as:

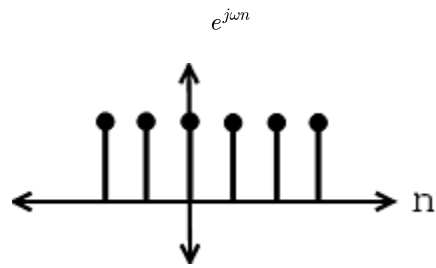


Figure 6.2.4: Complex sinusoid with frequency  $\omega = 0$

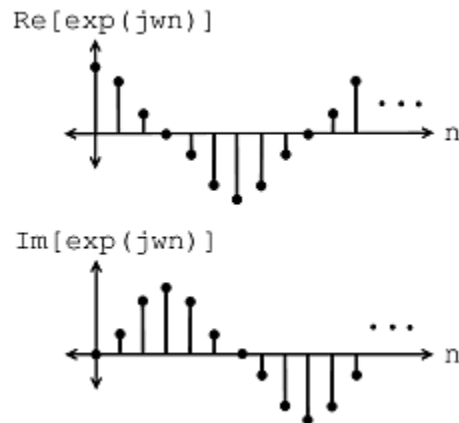


Figure 6.2.5: Complex sinusoid with frequency  $\omega = \frac{\pi}{4}$

## In the Complex Plane

The complex sinusoid can be directly mapped onto our [complex plane](#), which allows us to easily visualize changes to the complex sinusoid and extract certain properties. The absolute value of our complex sinusoid has the following characteristic:

$$|e^{j\omega n}| = 1, n \in \mathbb{R}$$

which tells that our complex sinusoid only takes values on the unit circle. As for the angle, the following statement holds true:

$$\angle(e^{j\omega n}) = \omega n$$

For more information, see the section on the **Discrete Time Complex Exponential** to learn about **Aliasing**, **Negative Frequencies**, and the formal definition of the **Complex Conjugate**.

Now that we have looked over the concepts of complex sinusoids, let us turn our attention back to finding a basis for discrete-time, periodic signals. After looking at all the complex sinusoids, we must answer the question of which discrete-time sinusoids do we need to represent periodic sequences with a period  $N$ .

### Equivalent Question

Find a set of vectors  $\forall n, n = \{0, \dots, N-1\} : (b_k = e^{j\omega_k n})$  such that  $b_k$  are a **basis** for  $\mathbb{C}^N$

In answer to the above question, let us try the "harmonic" sinusoids with a fundamental frequency  $\omega_0 = \frac{2\pi}{N}$ :

### Harmonic Sinusoid

$$e^{j\frac{2\pi}{N}kn}$$

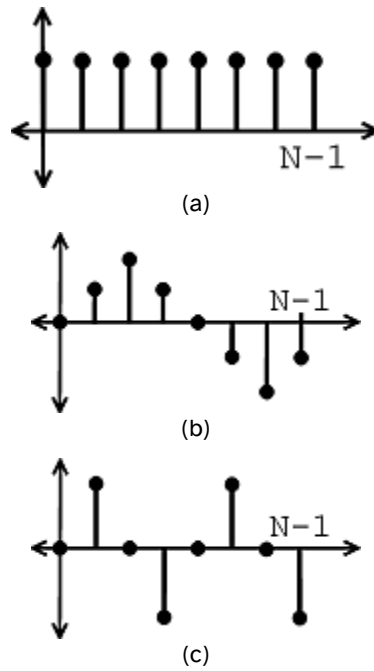


Figure 6.2.6: Examples of our Harmonic Sinusoids (a) Harmonic sinusoid with  $k = 0$  (b) Imaginary part of sinusoid,  $\text{Im}\left(e^{j\frac{2\pi}{N}1n}\right)$ , with  $k = 1$  (c) Imaginary part of sinusoid,  $\text{Im}\left(e^{j\frac{2\pi}{N}2n}\right)$ , with  $k = 2$

$e^{j\frac{2\pi}{N}kn}$  is periodic with period  $N$  and has  $k$  "cycles" between  $n = 0$  and  $n = N - 1$ .

### Theorem 6.2.1

If we let

$$b_k[n] = \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}kn}, \quad n = \{0, \dots, N-1\}$$

where the exponential term is a vector in  $\mathbb{C}^N$ , then  $\{b_k\}_{k=\{0,\dots,N-1\}}$  is an orthonormal basis (Section 15.8) for  $\mathbb{C}^N$ .

#### Proof

First of all, we must show  $b_k$  is orthonormal, i.e.  $\langle b_k, b_l \rangle = \delta_{kl}$

$$\langle b_k, b_l \rangle = \sum_{n=0}^{N-1} b_k[n] b_l[n]^* = \frac{1}{N} \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}kn} e^{-j\frac{2\pi}{N}ln}$$

$$\langle b_k, b_l \rangle = \frac{1}{N} \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(l-k)n}$$

If  $l = k$ , then

$$\begin{aligned} \langle b_k, b_l \rangle &= \frac{1}{N} \sum_{n=0}^{N-1} 1 \\ &= 1 \end{aligned} \tag{6.2.2}$$

If  $l \neq k$  then we must use the "partial summation formula" shown below:

$$\sum_{n=0}^{N-1} \alpha^n = \sum_{n=0}^{\infty} \alpha^n - \sum_{n=N}^{\infty} \alpha^n = \frac{1}{1-\alpha} - \frac{\alpha^N}{1-\alpha} = \frac{1-\alpha^N}{1-\alpha}$$

$$\langle b_k, b_l \rangle = \frac{1}{N} \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(l-k)n}$$

where in the above equation we can say that  $\alpha = e^{j\frac{2\pi}{N}(l-k)}$ , and thus we can see how this is in the form needed to utilize our partial summation formula.

$$\langle b_k, b_l \rangle = \frac{1}{N} \frac{1 - e^{j\frac{2\pi}{N}(l-k)N}}{1 - e^{j\frac{2\pi}{N}(l-k)}} = \frac{1}{N} \frac{1 - 1}{1 - e^{j\frac{2\pi}{N}(l-k)}} = 0$$

So,

$$\langle b_k, b_l \rangle = \begin{cases} 1 & \text{if } k = l \\ 0 & \text{if } k \neq l \end{cases}$$

Therefore:  $b_k$  is an orthonormal set.  $b_k$  is also a basis (Section 14.1), since there are  $N$  vectors which are linearly independent (Section 14.1 - Linear Independence) (orthogonality implies linear independence).

And finally, we have shown that the harmonic sinusoids  $\left\{ \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}kn} \right\}$  form an orthonormal basis for  $\mathbb{C}^n$

## Periodic Extension to DTFS

Now that we have an understanding of the discrete-time Fourier series (DTFS), we can consider the **periodic extension** of  $c[k]$  (the Discrete-time Fourier coefficients). Figure 6.2.7 shows a simple illustration of how we can represent a sequence as a periodic signal mapped over an infinite number of intervals.

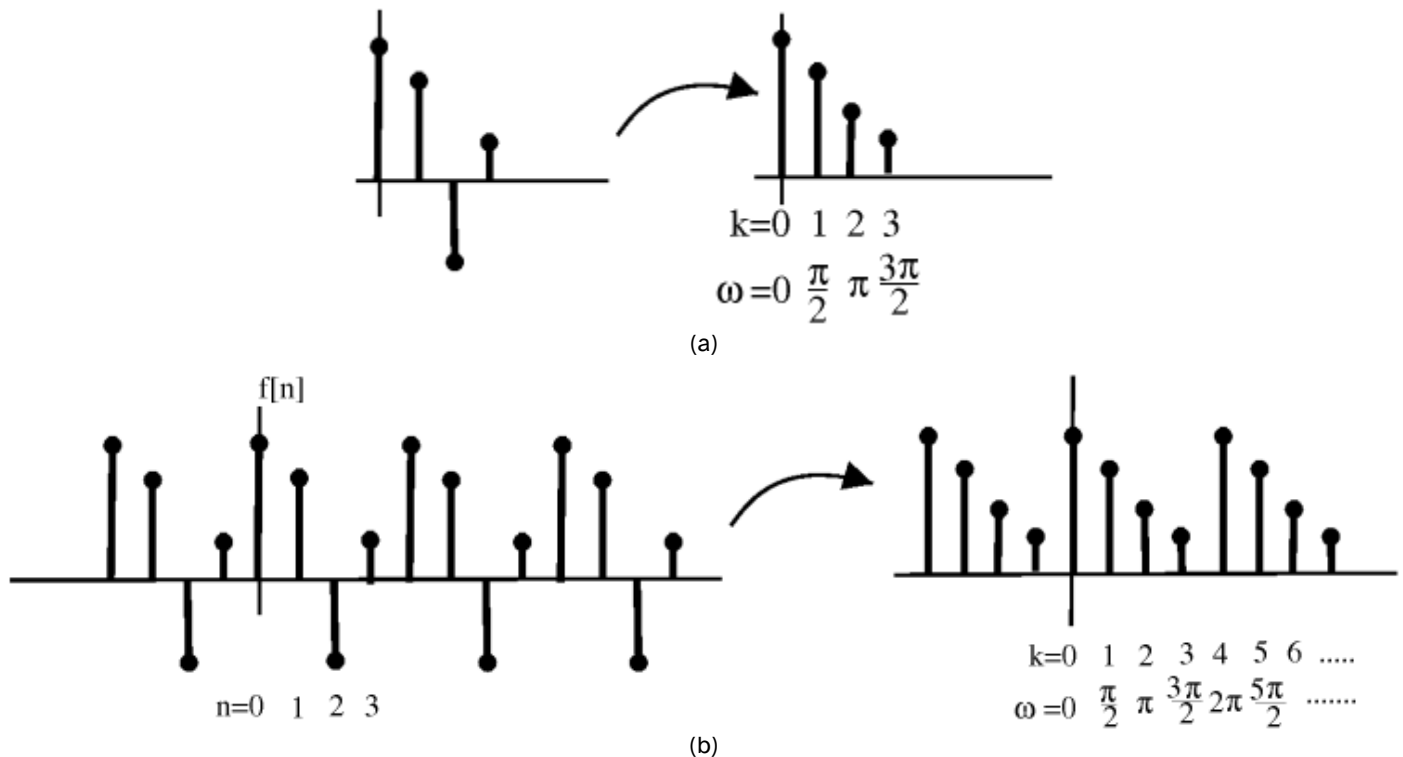


Figure 6.2.7: (a) vectors (b) periodic sequences

### Exercise 6.2.1

Why does a periodic (Section 6.1) extension to the DTFS coefficients  $c[k]$  make sense?

## Examples

### Example 6.2.1: Discrete Time Square Wave

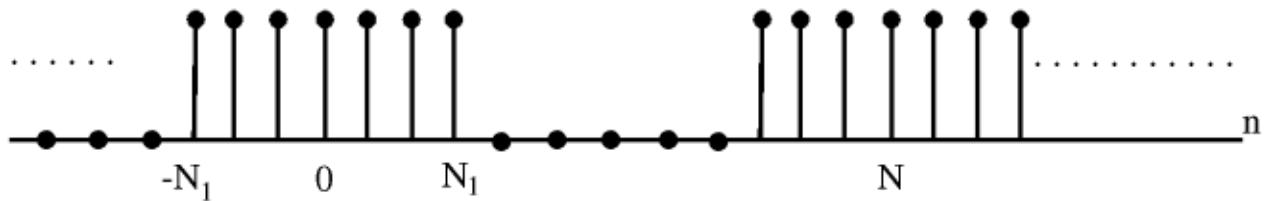


Figure  
6.2.8

the DTFS  $c[k]$  using:

$$c[k] = \frac{1}{N} \sum_{n=0}^{N-1} f[n] e^{-j\frac{2\pi}{N}kn}$$

Just like continuous time Fourier series, we can take the summation over any interval, so we have

$$c_k = \frac{1}{N} \sum_{n=-N_1}^{N_1} e^{-j\frac{2\pi}{N}kn}$$

Let  $m = n + N_1$  (so we can get a geometric series starting at 0)

$$\begin{aligned} \sum_{n=0}^M a^n &= \frac{1 - a^{M+1}}{1 - a} \\ c_k &= \frac{1}{N} e^{j\frac{2\pi}{N}N_1k} \sum_{m=0}^{2N_1} \left( e^{-j\frac{2\pi}{N}k} \right)^m \\ &= \frac{1}{N} e^{j\frac{2\pi}{N}N_1k} \frac{1 - e^{-j\frac{2\pi}{N}(2N_1+1)k}}{1 - e^{-j\frac{2\pi}{N}k}} \end{aligned} \quad (6.2.3)$$

Now, using the "partial summation formula"

$$\sum_{n=0}^M a^n = \frac{1 - a^{M+1}}{1 - a}$$

$$\begin{aligned}
 c_k &= \frac{1}{N} e^{j\frac{2\pi}{N} N_1 k} \sum_{m=0}^{2N_1} \left( e^{-j\frac{2\pi}{N} k} \right)^m \\
 &= \frac{1}{N} e^{j\frac{2\pi}{N} N_1 k} \frac{1 - e^{-j\frac{2\pi}{N} (2N_1+1)k}}{1 - e^{-j\frac{2\pi}{N} k}}
 \end{aligned} \tag{6.2.4}$$

Manipulate to make this look like a sinc function (distribute):

$$\begin{aligned}
 c_k &= \frac{1}{N} \frac{e^{-j\frac{2\pi}{N} k} \left( e^{jk\frac{2\pi}{N} (N_1+\frac{1}{2})} - e^{-jk\frac{2\pi}{N} (N_1+\frac{1}{2})} \right)}{e^{-j\frac{2\pi}{N} k} \left( e^{jk\frac{2\pi}{N} \frac{1}{2}} - e^{-jk\frac{2\pi}{N} \frac{1}{2}} \right)} \\
 &= \frac{1}{N} \frac{\sin\left(\frac{2\pi k(N_1+\frac{1}{2})}{N}\right)}{\sin\left(\frac{\pi k}{N}\right)} \\
 &= \text{digital sinc}
 \end{aligned} \tag{6.2.5}$$

#### Note

It's periodic! Figure 6.2.9, Figure 6.2.10, and Figure 6.2.11 show our above function and coefficients for various values of  $N_1$ .

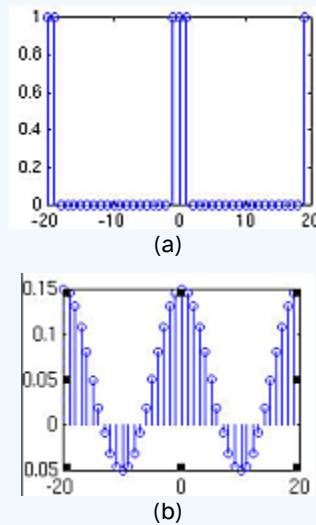


Figure 6.2.9:  $N_1 = 1$  (a) Plot of  $f[n]$ . (b) Plot of  $c[k]$

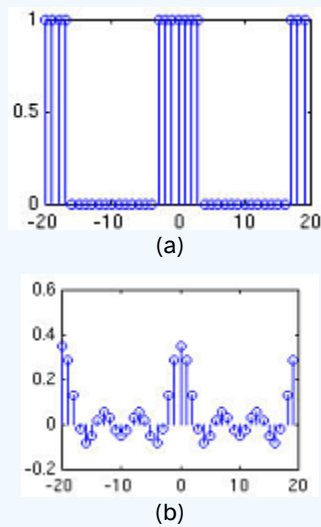


Figure 6.2.10:  $N_1 = 3$  (a) Plot of  $f[n]$ . (b) Plot of  $c[k]$ .

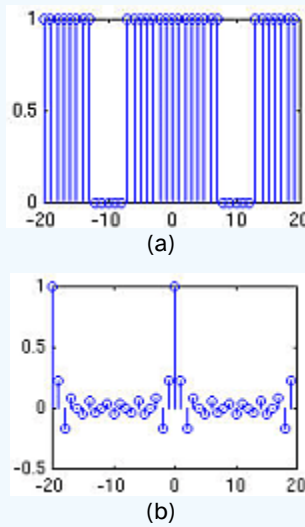


Figure 6.2.11:  $N_1 = 7$  (a) Plot of  $f[n]$ . (b) Plot of  $c[k]$ .

## DTFS conclusion

Using the steps shown above in the derivation and our previous understanding of Hilbert Spaces (Section 15.4) and Orthogonal Expansions (Section 15.9), the rest of the derivation is automatic. Given a discrete-time, periodic signal (vector in  $\mathbb{C}^n$ )  $f[n]$ , we can write:

$$f[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} c_k e^{j\frac{2\pi}{N} kn}$$

$$c_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f[n] e^{-j\frac{2\pi}{N} kn}$$

Note: Most people collect both the  $\frac{1}{\sqrt{N}}$  terms into the expression for  $c_k$ .

### Discrete Time Fourier Series

Here is the common form of the DTFS with the above note taken into account:

$$f[n] = \sum_{k=0}^{N-1} c_k e^{j\frac{2\pi}{N}kn}$$

$$c_k = \frac{1}{N} \sum_{n=0}^{N-1} f[n] e^{-j\frac{2\pi}{N}kn}$$

This is what the `fft` command in MATLAB does.

## Answers

### Exercise 6.2.1

Aliasing:  $b_k = e^{j\frac{2\pi}{N}kn}$

$$\begin{aligned} b_{k+N} &= e^{j\frac{2\pi}{N}(k+N)n} \\ &= e^{j\frac{2\pi}{N}kn} e^{j2\pi n} \\ &= e^{j\frac{2\pi}{N}kn} \\ &= b_k \end{aligned} \tag{6.2.6}$$

This page titled [6.2: Discrete Time Fourier Series \(DTFS\)](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.

- [7.2: Discrete Time Fourier Series \(DTFS\)](#) by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6.3: Common Discrete Fourier Series

### Introduction

Once one has obtained a solid understanding of the fundamentals of **Fourier series analysis** and the **General Derivation of the Fourier Coefficients**, it is useful to have an understanding of the common signals used in Fourier Series Signal Approximation.

### Deriving the Coefficients

Consider a square wave  $f(x)$  of length 1. Over the range  $[0,1)$ , this can be written as

$$x(t) = \begin{cases} 1 & t \leq \frac{1}{2} \\ -1 & t > \frac{1}{2} \end{cases}$$

Fourier series approximation of a square wave

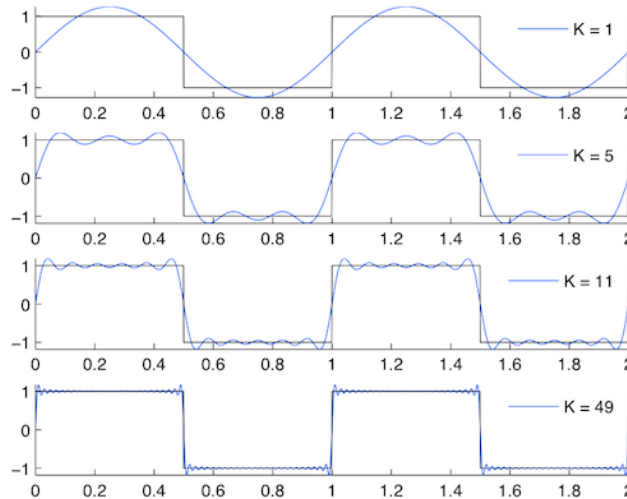


Figure 6.3.1: Fourier series approximation to  $sq(t)$ . The number of terms in the Fourier sum is indicated in each plot, and the square wave is shown as a dashed line over two periods.

**Real Even Signals** Given that the square wave is a real and even signal,

- $f(t) = f(-t)$  EVEN
- $f(t) = f^*(t)$  REAL

therefore,

- $c_n = c_{-n}$  EVEN
- $c_n = c_n^*$  REAL

### Deriving the Coefficients for other signals

The Square wave is the standard example, but other important signals are also useful to analyze, and these are included here.

## Constant Waveform

This signal is relatively self-explanatory: the time-varying portion of the Fourier Coefficient is taken out, and we are left simply with a constant function over all time.

$$x(t) = 1$$

**Fourier series approximation of a constant wave**

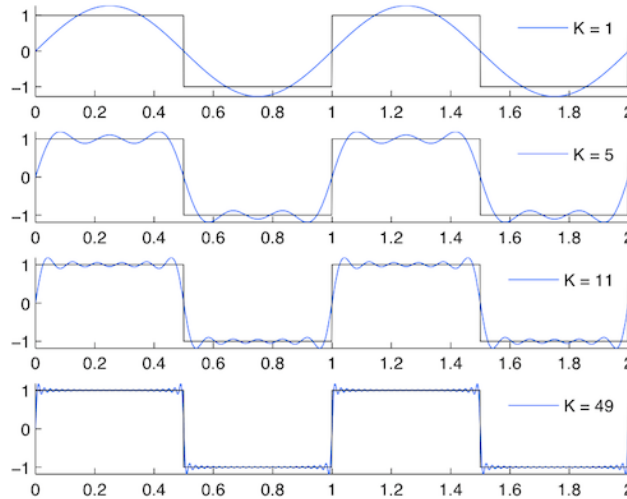


Figure  
6.3.2

## Sinusoid Waveform

With this signal, only a specific frequency of time-varying Coefficient is chosen (given that the Fourier Series equation includes a sine wave, this is intuitive), and all others are filtered out, and this single time-varying coefficient will exactly match the desired signal.

$$x(t) = \cos(2\pi t)$$

## Fourier series approximation of a sinusoid wave

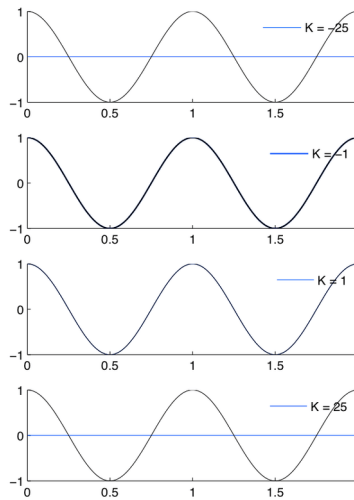


Figure  
6.3.3

## Triangle Waveform

$$x(t) = \begin{cases} t & t \leq 1/2 \\ 1 - t & t > 1/2 \end{cases}$$

This is a more complex form of signal approximation to the square wave. Because of the **Symmetry Properties** of the Fourier Series, the triangle wave is a real and odd signal, as opposed to the real and even square wave signal. This means that

- $f(t) = -f(-t)$  ODD
- $f(t) = f^*(t)$  REAL

therefore,

- $c_n = -c_{-n}$
- $c_n = -c_n^*$  IMAGINARY

### Fourier series approximation of a triangle wave

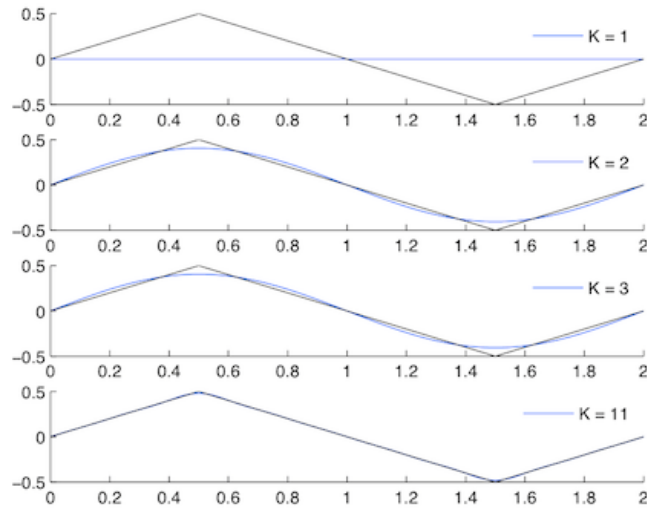


Figure  
6.3.4

## Sawtooth Waveform

$$x(t) = t/2$$

Because of the **Symmetry Properties** of the Fourier Series, the sawtooth wave can be defined as a real and odd signal, as opposed to the real and even square wave signal. This has important implications for the Fourier Coefficients.

### Fourier series approximation of a sawtooth wave

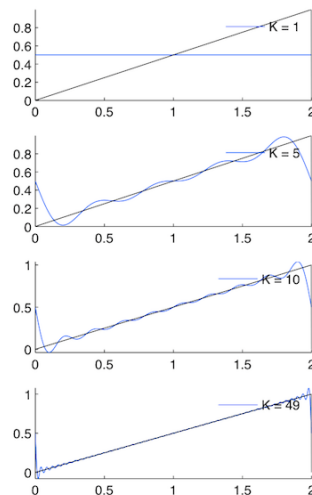


Figure  
6.3.5

## DFT Signal Approximation

### Embedded Object

This interactive element is not available in this format. Visit the link or scan the QR code to access it online.

[View this Embedded Object online](#)



Figure 6.3.6: Interact (when online) with a Mathematica CDF demonstrating the common Discrete Fourier Series. To download, right-click and save as .cdf.

## Conclusion

To summarize, a great deal of variety exists among the common Fourier Transforms. A summary table is provided here with the essential information.

Table 6.3.1: Common Discrete Fourier Transforms

| Description                    | Time Domain Signal for $n \in \mathbb{Z}[0, N - 1]$              | Frequency Domain Signal $k \in \mathbb{Z}[0, N - 1]$             |
|--------------------------------|------------------------------------------------------------------|------------------------------------------------------------------|
| Constant Function              | 1                                                                | $\delta(k)$                                                      |
| Unit Impulse                   | $\delta(n)$                                                      | $\frac{1}{N}$                                                    |
| Complex Exponential            | $e^{j2\pi mn/N}$                                                 | $\delta((k - m)_N)$                                              |
| Sinusoid Waveform              | $\cos(j2\pi mn/N)$                                               | $\frac{1}{2} (\delta((k - m)_N) + \delta((k + m)_N))$            |
| Box Waveform ( $M < N/2$ )     | $\delta(n) + \sum_{m=1}^M \delta((n - m)_N) + \delta((n + m)_N)$ | $\frac{\sin((2M+1)k\pi/N)}{N \sin(k\pi/N)}$                      |
| Dsinc Waveform $\backslash((M$ | $\frac{\sin((2M+1)n\pi/N)}{\sin(n\pi/N)}$                        | $\delta(k) + \sum_{m=1}^M \delta((k - m)_N) + \delta((k + m)_N)$ |

This page titled 6.3: Common Discrete Fourier Series is shared under a CC BY-SA 4.0 license and was authored, remixed, and/or curated by Carl Greco.

- 7.3: Common Discrete Fourier Series by Richard Baraniuk et al. is licensed CC BY 3.0. Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6.4: Properties of the DTFS

### Introduction

In this module we will discuss the basic properties of the Discrete-Time Fourier Series. We will begin by refreshing your memory of our basic [Fourier series](#) equations:

$$f[n] = \sum_{k=0}^{N-1} c_k e^{j\omega_0 kn}$$

$$c_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} f[n] e^{-\left(j\frac{2\pi}{N} kn\right)}$$

Let  $\mathcal{F}(\cdot)$  denote the transformation from  $f[n]$  to the Fourier coefficients

$$\mathcal{F}(f[n]) = c_k \quad , \quad k \in \mathbb{Z}$$

$\mathcal{F}(\cdot)$  maps complex valued functions to sequences of complex numbers.

### Linearity

$\mathcal{F}(\cdot)$  is a **linear transformation**.

#### Theorem 6.4.1

If  $\mathcal{F}(f[n]) = c_k$  and  $\mathcal{F}(g[n]) = d_k$ . Then

$$\mathcal{F}(\alpha f[n]) = \alpha c_k \quad , \quad \alpha \in \mathbb{C}$$

and

$$\mathcal{F}(f[n] + g[n]) = c_k + d_k$$

#### Proof

Easy. Just linearity of integral.

$$\begin{aligned} \mathcal{F}(f[n] + g[n]) &= \sum_{n=0}^N (f[n] + g[n]) e^{-\left(j\omega_0 kn\right)}, k \in \mathbb{Z} \\ &= \frac{1}{N} \sum_{n=0}^N f[n] e^{-\left(j\omega_0 kn\right)} + \frac{1}{N} \sum_{n=0}^N g[n] e^{-\left(j\omega_0 kn\right)}, k \in \mathbb{Z} \\ &= c_k + d_k, k \in \mathbb{Z} \\ &= c_k + d_k \end{aligned} \tag{6.4.1}$$

### Shifting

Shifting in time equals a phase shift of [Fourier coefficients](#).

### Theorem 6.4.2

$\mathcal{F}(f[n - n_0]) = e^{-(j\omega_0 k n_0)} c_k$  if  $c_k = |c_k| e^{j\angle(c_k)}$ , then

$$|e^{-(j\omega_0 k n_0)} c_k| = |e^{-(j\omega_0 k n_0)}| |c_k| = |c_k|$$

$$\angle(e^{-(j\omega_0 k n_0)} c_k) = \angle(c_k) - \omega_0 n_0 k$$

#### Proof

$$\begin{aligned} \mathcal{F}(f[n - n_0]) &= \frac{1}{N} \sum_{n=0}^N f[n - n_0] e^{-j\omega_0 k n}, \quad k \in \mathbb{Z} \\ &= \frac{1}{N} \sum_{n=-n_0}^{N-n_0} f[n - n_0] e^{-(j\omega_0 k(n-n_0))} e^{-(j\omega_0 k n_0)}, \quad k \in \mathbb{Z} \\ &= \frac{1}{N} \sum_{n=-n_0}^{N-n_0} f[\tilde{n}] e^{-(j\omega_0 k \tilde{n})} e^{-(j\omega_0 k n_0)}, \quad k \in \mathbb{Z} \\ &= e^{-(j\omega_0 k n_0)} c_k, \quad k \in \mathbb{Z} \end{aligned} \quad (6.4.2)$$

## Parseval's Relation

$$\sum_{n=0}^N (|f[n]|)^2 = N \sum_{k=0}^{N-1} (|c_k|)^2$$

Parseval's relation tells us that the energy of a signal is equal to the energy of its Fourier transform.

#### Note

Parseval tells us that the Fourier series **maps**  $L^2[0, N]$  to  $l^2$ .

$$\begin{aligned} f[n] &\xleftrightarrow{\text{DTFS}} \{c_k\} \\ f[n] \in L^2[0, N] &\iff \{c_k\} \in l^2\{\mathbb{Z}\} \end{aligned}$$

Figure  
6.4.1

### Exercise 6.4.1

For  $f[n]$  to have "finite energy," what do the  $c_k$  do as  $k \rightarrow \infty$ ?

Answer available at the end of this section.

### Exercise 6.4.2

If  $\forall k, |k| > 0 : (c_k = \frac{1}{k})$ , is  $f \in L^2[[0, N]]$ ?

Answer available at the end of this section.

### Exercise 6.4.3

Now, if  $\forall k, |k| > 0 : \left(c_k = \frac{1}{\sqrt{k}}\right)$ , is  $f \in L^2[[0, N]]$ ?

Answer available at the end of this section.

The rate of decay of the Fourier series determines if  $f[n]$  has **finite energy**.

## Parsevals Theorem Demonstration

### Embedded Object

This interactive element is not available in this format. Visit the link or scan the QR code to access it online.

[View this Embedded Object online](#)



Figure 6.4.2: Interact (when online) with a Mathematica CDF demonstrating Parsevals Theorem. To download, right-click and save as .cdf.

## Symmetry Properties

### Rule 6.4.1: Even Signals

$$f[n] = f[-n]$$

$$\|c_k\| = \|c_{-k}\|$$

#### Proof

$$\begin{aligned} c_k &= \frac{1}{N} \sum_0^N [n] \exp[-j\omega_0 kn] \\ &= \frac{1}{N} \sum_0^{\frac{N}{2}} f[n] \exp[-j\omega_0 kn] + \frac{1}{N} \sum_{\frac{N}{2}}^N f[n] \exp[-j\omega_0 kn] \\ &= \frac{1}{N} \sum_0^{\frac{N}{2}} f[-n] \exp[-j\omega_0 kn] + \frac{1}{N} \sum_{\frac{N}{2}}^N f[-n] \exp[-j\omega_0 kn] \\ &= \frac{1}{N} \sum_0^N [n] [\exp[j\omega_0 kn] + \exp[-j\omega_0 kn]] \\ &= \frac{1}{N} \sum_0^N [n] 2 \cos[\omega_0 kn] \end{aligned} \tag{6.4.3}$$

### Rule 6.4.2: Odd Signals

$$f[n] = -f[-n]$$

$$c_k = c_{-k}^*$$

#### Proof

$$\begin{aligned}
 c_k &= \frac{1}{N} \sum_0^N f[n] \exp[-j\omega_0 kn] \\
 &= \frac{1}{N} \sum_0^{\frac{N}{2}} f[n] \exp[-j\omega_0 kn] + \frac{1}{N} \sum_{\frac{N}{2}}^N f[n] \exp[-j\omega_0 kn] \\
 &= \frac{1}{N} \sum_0^{\frac{N}{2}} f[n] \exp[-j\omega_0 kn] - \frac{1}{N} \sum_{\frac{N}{2}}^N f[-n] \exp[j\omega_0 kn] \\
 &= -\frac{1}{N} \sum_0^N f[n] [\exp[j\omega_0 kn] - \exp[-j\omega_0 kn]] \\
 &= -\frac{1}{N} \sum_0^N f[n] 2j \sin[\omega_0 kn]
 \end{aligned} \tag{6.4.4}$$

### Rule 6.4.3: Real Signals

$$f[n] = f^*[n]$$

$$c_k = c_{-k}^*$$

#### Proof

Erroneous nesting of equation structures

## Differentiation in Fourier Domain

$$(\mathcal{F}(f[n]) = c_k) \Rightarrow \left( \mathcal{F}\left(\frac{df[n]}{dn}\right) = jk\omega_0 c_k \right)$$

Since

$$f[n] = \sum_{k=0}^N c_k e^{j\omega_0 kn}$$

then

$$\begin{aligned}
 \frac{d}{dn} f[n] &= \sum_{k=0}^N c_k \frac{de^{j\omega_0 kn}}{dn} \\
 &= \sum_{k=0}^N c_k j\omega_0 k e^{j\omega_0 kn}
 \end{aligned} \tag{6.4.5}$$

A differentiator **attenuates** the low frequencies in  $f[n]$  and **accentuates** the high frequencies. It removes general trends and accentuates areas of sharp variation.

### Note

A common way to mathematically measure the smoothness of a function  $f[n]$  is to see how many derivatives are finite energy.

This is done by looking at the Fourier coefficients of the signal, specifically how fast they **decay** as  $k \rightarrow \infty$ . If  $\mathcal{F}(f[n]) = c_k$  and  $|c_k|$  has the form  $\frac{1}{k^l}$ , then  $\mathcal{F}\left(\frac{d^m f[n]}{dn^m}\right) = (jk\omega_0)^m c_k$  and has the form  $\frac{k^m}{k^l}$ . So for the  $m^{\text{th}}$  derivative to have finite energy, we need

$$\sum_k \left( \left| \frac{k^m}{k^l} \right| \right)^2 < \infty$$

thus  $\frac{k^m}{k^l}$  decays **faster** than  $\frac{1}{k}$  which implies that

$$2l - 2m > 1$$

or

$$l > \frac{2m + 1}{2}$$

Thus the decay rate of the Fourier series dictates smoothness.

## Fourier Differentiation Demo

### Embedded Object

*This interactive element is not available in this format. Visit the link or scan the QR code to access it online.*

[View this Embedded Object online](#)



Figure 6.4.3: Interact (when online) with a Mathematica CDF demonstrating Differentiation in a Fourier Domain. To download, right-click and save as .cdf.

## Integration in the Fourier Domain

If

$$\mathcal{F}(f[n]) = c_k$$

then

$$\mathcal{F}\left(\sum_{\eta=0}^n f[\eta]\right) = \frac{1}{j\omega_0 k} c_k$$

### Note

If  $c_0 \neq 0$ , this expression doesn't make sense.

Integration accentuates low frequencies and attenuates high frequencies. Integrators bring out the **general trends** in signals and suppress short term variation (which is noise in many cases). Integrators are **much** nicer than differentiators.

## Fourier Integration Demo

### Embedded Object

This interactive element is not available in this format. Visit the link or scan the QR code to access it online.

[View this Embedded Object online](#)



Figure 6.4.4: Interact (when online) with a Mathematica CDF demonstrating the effect of Integration in a Fourier Domain. To download, right-click and save as .cdf.

## Signal Multiplication

Given a signal  $f[n]$  with Fourier coefficients  $c_k$  and a signal  $g[n]$  with Fourier coefficients  $d_k$ , we can define a new signal,  $y[n]$ , where  $y[n] = f[n]g[n]$ . We find that the Fourier Series representation of  $y[n]$ ,  $e_k$ , is such that  $e_k = \sum_{l=0}^N c_l d_{k-l}$ . This is to say that signal multiplication in the time domain is equivalent to discrete-time circular convolution (Section 4.3) in the frequency domain. The proof of this is as follows

$$\begin{aligned}
 e_k &= \frac{1}{N} \sum_{n=0}^N f[n]g[n]e^{-j\omega_0 kn} \\
 &= \frac{1}{N} \sum_{n=0}^N \sum_{l=0}^N c_l e^{j\omega_0 ln} g[n]e^{-j\omega_0 kn} \\
 &= \sum_{l=0}^N c_l \left( \frac{1}{N} \sum_{n=0}^N g[n]e^{-j\omega_0 (k-l)n} \right) \\
 &= \sum_{l=0}^N c_l d_{k-l}
 \end{aligned} \tag{6.4.6}$$

## Conclusion

Like other Fourier transforms, the DTFS has many useful properties, including linearity, equal energy in the time and frequency domains, and analogs for shifting, differentiation, and integration.

Table 6.4.1: Properties of the Discrete Fourier Transform

| Property             | Signal               | DTFS                  |
|----------------------|----------------------|-----------------------|
| Linearity            | $ax(n) + by(n)$      | $aX(k) + bY(k)$       |
| Time Shifting        | $x(n - m)$           | $X(k)e^{-j2\pi mk/N}$ |
| Time Modulation      | $x(n)e^{j2\pi mn/N}$ | $X(k - m)$            |
| Multiplication       | $x(n)y(n)$           | $X(k) * Y(k)$         |
| Circular Convolution | $x(n) * y(n)$        | $X(k)Y(k)$            |

## Answers

### Exercise 6.4.1

$(|c_k|)^2 < \infty$  for  $f[n]$  to have finite energy.

### Exercise 6.4.2

Yes, because  $(|c_k|)^2 = \frac{1}{k^2}$ , which is summable.

**Exercise 6.4.3**

No, because  $(|c_k|)^2 = \frac{1}{k}$ , which is not summable.

---

This page titled [6.4: Properties of the DTFS](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- [7.4: Properties of the DTFS](#) by [Richard Baraniuk et al.](#) is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 6.5: Discrete Time Circular Convolution and the DTFS

### Introduction

This module relates circular convolution of periodic signals in one domain to multiplication in the other domain.

You should be familiar with Discrete-Time Convolution (Section 4.3), which tells us that given two discrete-time signals  $x[n]$ , the system's input, and  $h[n]$ , the system's response, we define the output of the system as

$$\begin{aligned} y[n] &= x[n] * h[n] \\ &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \end{aligned} \quad (6.5.1)$$

When we are given two DFTs (finite-length sequences usually of length  $N$ ), we cannot just multiply them together as we do in the above convolution formula, often referred to as **linear convolution**. Because the DFTs are periodic, they have nonzero values for  $n \geq N$  and thus the multiplication of these two DFTs will be nonzero for  $n \geq N$ . We need to define a new type of convolution operation that will result in our convolved signal being zero outside of the range  $n = 0, 1, \dots, N-1$ . This idea led to the development of **circular convolution**, also called cyclic or periodic convolution.

### Signal Circular Convolution

Given a signal  $f[n]$  with Fourier coefficients  $c_k$  and a signal  $g[n]$  with Fourier coefficients  $d_k$ , we can define a new signal,  $v[n]$ , where  $v[n] = f[n] \otimes g[n]$ . We find that the [Fourier Series](#) representation of  $v[n]$ ,  $a_k$ , is such that  $a_k = c_k d_k$ .  $f[n] \otimes g[n]$  is the circular convolution (Section 7.5) of two periodic signals and is equivalent to the convolution over one interval, i.e.

$$f[n] \otimes g[n] = \sum_{n=0}^N \sum_{\eta=0}^N f[\eta]g[n-\eta].$$

#### Note

Circular convolution in the time domain is equivalent to multiplication of the Fourier coefficients.

This is proved as follows

$$\begin{aligned} a_k &= \frac{1}{N} \sum_{n=0}^N v[n]e^{-j\omega_0 kn} \\ &= \frac{1}{N^2} \sum_{n=0}^N \sum_{\eta=0}^{\eta} f[\eta]g[n-\eta]e^{-j\omega_0 kn} \\ &= \frac{1}{N} \sum_{\eta=0}^N f[\eta] \left( \frac{1}{N} \sum_{n=0}^N g[n-\eta]e^{-j\omega_0 kn} \right) \\ &= \left( \frac{1}{N} \sum_{\eta=0}^N f[\eta] \left( \frac{1}{N} \sum_{\nu=-\eta}^{N-\eta} g[\nu]e^{-j\omega_0(\nu+\eta)} \right) \right) \quad , \quad \nu = n - \eta \\ &= \frac{1}{N} \sum_{\eta=0}^N f[\eta] \left( \frac{1}{N} \sum_{\nu=-\eta}^{N-\eta} g[\nu]e^{-j\omega_0 k\nu} \right) e^{-j\omega_0 k\eta} \\ &= \frac{1}{N} \sum_{\eta=0}^N f[\eta]d_k e^{-j\omega_0 k\eta} \\ &= d_k \left( \frac{1}{N} \sum_{\eta=0}^N f[\eta]e^{-j\omega_0 k\eta} \right) \\ &= c_k d_k \end{aligned} \quad (6.5.2)$$

## Circular Convolution Formula

What happens when we multiply two DFT's together, where  $Y[k]$  is the DFT of  $y[n]$ ?

$$Y[k] = F[k]H[k]$$

when  $0 \leq k \leq N - 1$

Using the DFT synthesis formula for  $y[n]$

$$y[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k]H[k]e^{j\frac{2\pi}{N}kn}$$

And then applying the analysis formula  $F[k] = \sum_{m=0}^{N-1} f[m]e^{(-j)\frac{2\pi}{N}km}$

$$\begin{aligned} y[n] &= \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} f[m]e^{(-j)\frac{2\pi}{N}km} H[k]e^{j\frac{2\pi}{N}kn} \\ &= \sum_{m=0}^{N-1} f[m] \left( \frac{1}{N} \sum_{k=0}^{N-1} H[k]e^{j\frac{2\pi}{N}k(n-m)} \right) \end{aligned} \quad (6.5.3)$$

where we can reduce the second summation found in the above equation into  $h[(n-m)_N] = \frac{1}{N} \sum_{k=0}^{N-1} H[k]e^{j\frac{2\pi}{N}k(n-m)}y[n]$   
 $= \sum_{m=0}^{N-1} f[m]h[(n-m)_N]$  which equals circular convolution! When we have  $0 \leq n \leq N - 1$  in the above, then we get:

$$y[n] \equiv f[n] \circledast h[n]$$

### Note

The notation  $\circledast$  represents cyclic convolution "mod N".

## Alternative Convolution Formula

### Alternative Circular Convolution Algorithm

- Step 1: Calculate the DFT of  $f[n]$  which yields  $F[k]$  and calculate the DFT of  $h[n]$  which yields  $H[k]$ .
- Step 2: Pointwise multiply  $Y[k] = F[k]H[k]$
- Step 3: Inverse DFT  $Y[k]$  which yields  $y[n]$

Seems like a roundabout way of doing things, **but** it turns out that there are **extremely** fast ways to calculate the DFT of a sequence.

To circularly convolve 2  $N$ -point sequences:  $y[n] = \sum_{m=0}^{N-1} f[m]h[(n-m)_N]$ . For each  $n : N$  multiples,  $N - 1$  additions.

$N$  points implies  $N^2$  multiplications,  $N(N - 1)$  additions implies  $O(N^2)$  complexity.

## Steps for Circular Convolution

We can picture periodic (Section 6.1) sequences as having discrete points on a circle as the domain

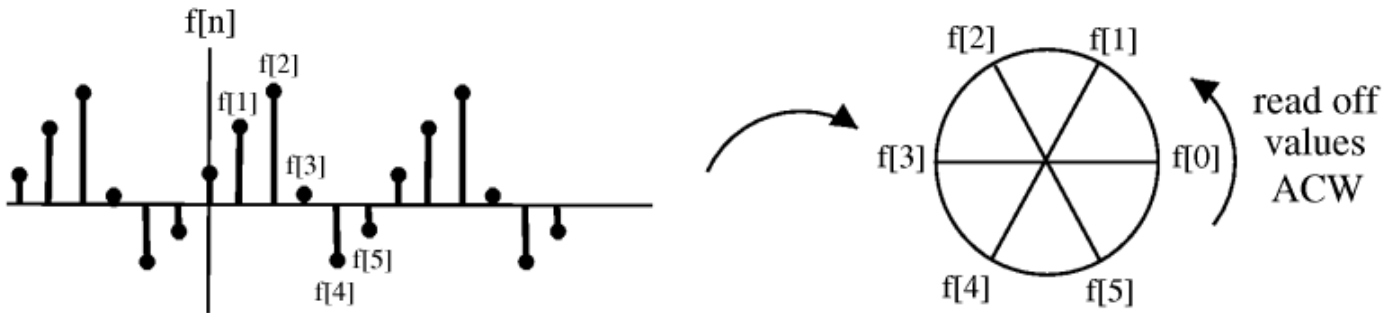


Figure 6.5.1

Shifting by  $m$ ,  $f(n + m)$ , corresponds to rotating the cylinder  $m$  notches ACW (counter clockwise). For  $m = -2$ , we get a shift equal to that in the following illustration:

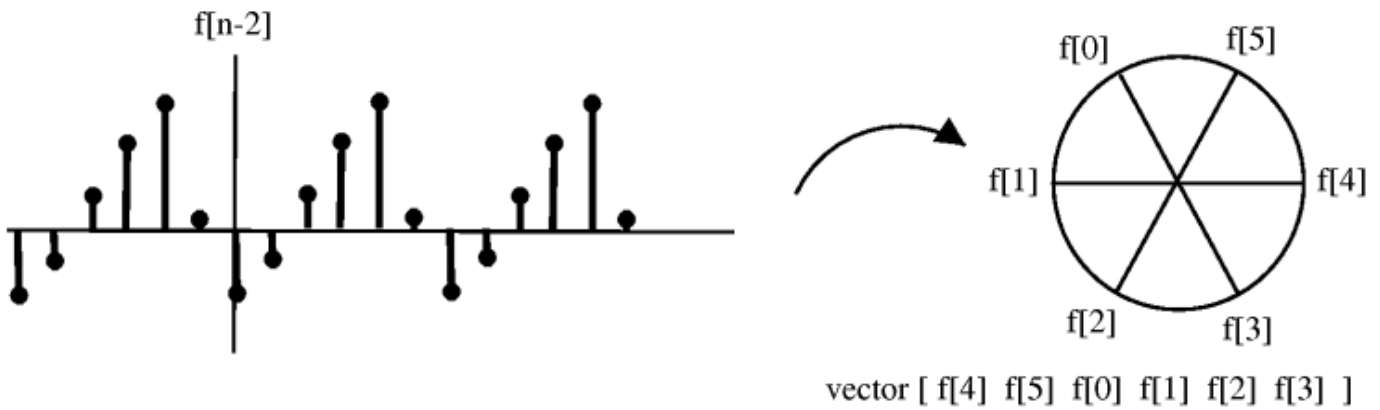


Figure 6.5.2: for  $m = -2$

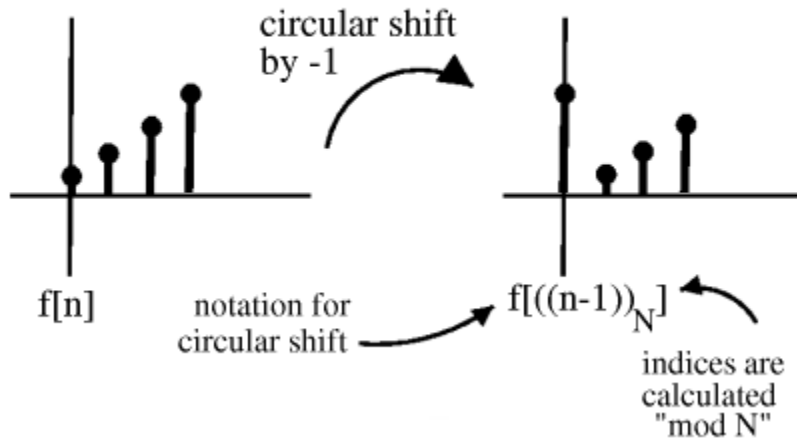


Figure 6.5.3

To cyclic shift we follow these steps:

1) Write  $f(n)$  on a cylinder, ACW

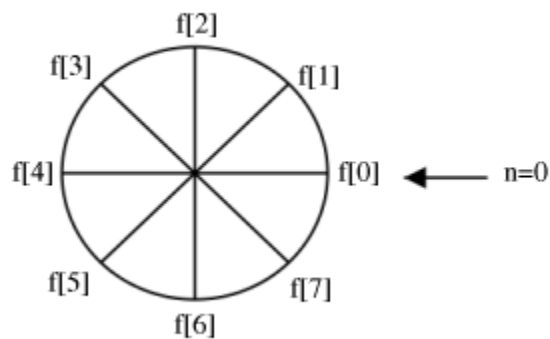


Figure 6.5.4:  $N = 8$

2) To cyclic shift by  $m$ , spin cylinder  $m$  spots ACW

$$f[n] \rightarrow f[(n + m)_N]$$

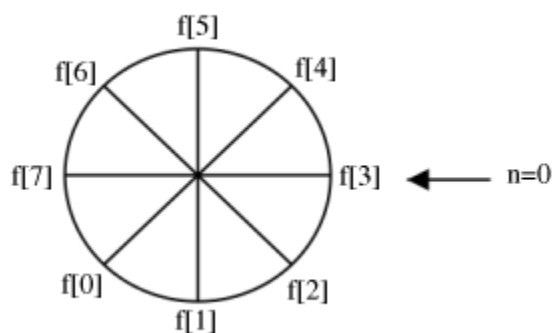


Figure 6.5.5:  $m = -3$

## Notes on circular shifting

$f[(n + N)_N] = f[n]$  Spinning  $N$  spots is the same as spinning all the way around, or not spinning at all.

$f[(n + N)_N] = f[(n - (N - m))_N]$  Shifting ACW  $m$  is equivalent to shifting CW  $N - m$

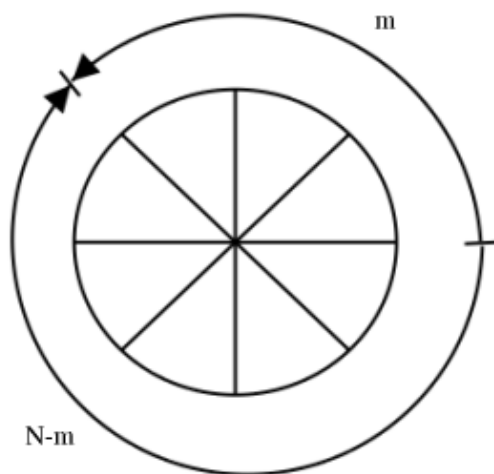
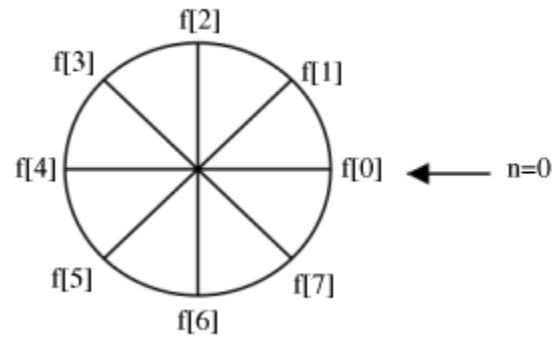
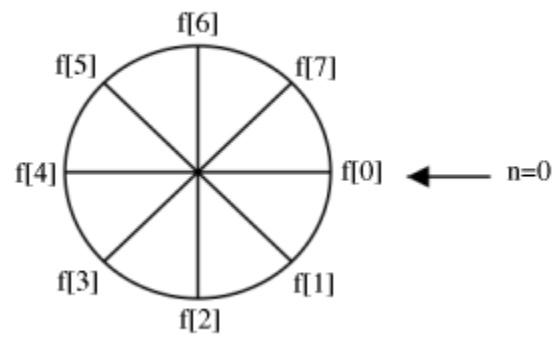


Figure  
6.5.6

$f[(-n)_N]$  The above expression, simply writes the values of  $f[n]$  clockwise.



(a)  
 $f[n]$



(b)  
 $f[(n+8)s]$

Figure 6.5.7

### Example 6.5.1

Convolve ( $n = 4$ )

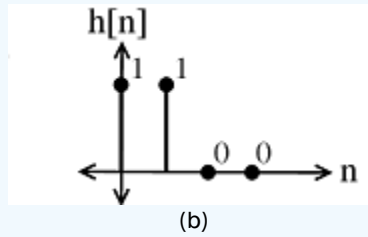
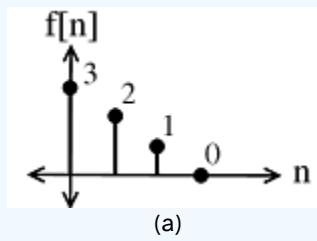


Figure 6.5.8: Two discrete-time signals to be convolved.

- $h[((-m))_N]$

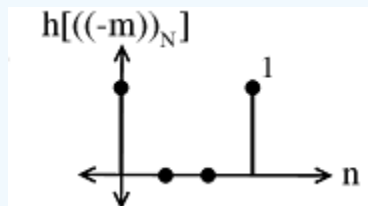


Figure 6.5.9

Multiply  $f[m]$  and sum to yield:  $y[0] = 3$

- $h[((1-m))_N]$

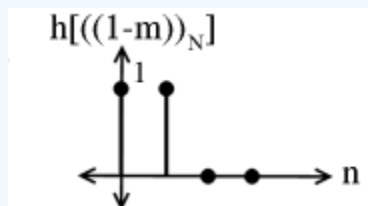


Figure 6.5.10

Multiply  $f[m]$  and sum to yield:  $y[1] = 5$

- $h[((2-m))_N]$

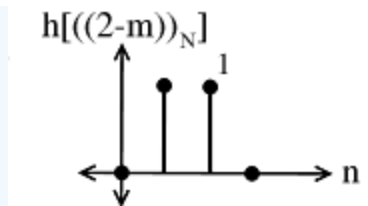


Figure  
6.5.11

Multiply  $f[m]$  and sum to yield:  $y[2] = 3$

- $h[((3-m))_N]$

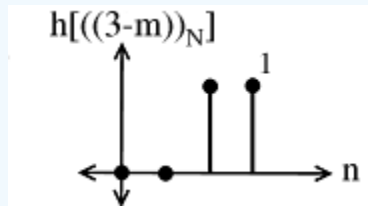


Figure  
6.5.12

Multiply  $f[m]$  and sum to yield:  $y[3] = 1$

## Exercise

Take a look at a square pulse with a period of  $T$ .

For this signal  $c_k = \begin{cases} \frac{1}{N} & \text{if } k = 0 \\ \frac{1}{2} \frac{\sin(\frac{\pi}{2}k)}{\frac{\pi}{2}k} & \text{otherwise} \end{cases}$

Take a look at a triangle pulse train with a period of  $T$ .

This signal is created by circularly convolving the square pulse with itself. The Fourier coefficients for this signal are  $a_k = c_k^2$   
 $= \frac{1}{4} \frac{\sin^2(\frac{\pi}{2}k)}{(\frac{\pi}{2}k)}$

### Exercise 6.5.1

Find the Fourier coefficients of the signal that is created when the square pulse and the triangle pulse are convolved.

*Answer available at the end of this section.*

## Circular Shifts and the DFT

### Theorem 6.5.1: Circular Shifts and DFT

If  $f[n] \xleftrightarrow{\text{DFT}} F[k]$  then  $f[(n-m)_N] \xleftrightarrow{\text{DFT}} e^{-j(\frac{2\pi}{N}km)} F[k]$  (i.e. circular shift in time domain = phase shift in DFT)

### Proof

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{j \frac{2\pi}{N} kn}$$

so phase shifting the DFT

$$\begin{aligned} f[n] &= \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{-j \frac{2\pi}{N} kn} e^{j \frac{2\pi}{N} kn} \\ &= \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{j \frac{2\pi}{N} k(n-m)} \\ &= f[(n-m)_N] \end{aligned} \quad (6.5.4)$$

## Circular Convolution Demonstration

### Embedded Object

This interactive element is not available in this format. Visit the link or scan the QR code to access it online.

[View this Embedded Object online](#)



Figure 6.5.13: Interact (when online) with a Mathematica CDF demonstrating Circular Shifts.

## Conclusion

Circular convolution in the time domain is equivalent to multiplication of the Fourier coefficients in the frequency domain.

## Answers

### Exercise 6.5.1

$$a_k = \begin{cases} \text{undefined} & k = 0 \\ \frac{1}{8} \frac{\sin^3[\frac{\pi}{4}k]}{[\frac{\pi}{4}k]^3} & \text{otherwise} \end{cases}$$

This page titled [6.5: Discrete Time Circular Convolution and the DTFS](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

- 7.5: Discrete Time Circular Convolution and the DTFS by Richard Baraniuk et al. is licensed [CC BY 3.0](#). Original source: [https://cnx.org/contents/d2CEAGW5@15.4:l-e\\_wSqM@24/Signal-Classifications-and-Properties](https://cnx.org/contents/d2CEAGW5@15.4:l-e_wSqM@24/Signal-Classifications-and-Properties).

## 7: Discrete -Time Fourier Transform (DTFT)

### Learning Objectives

- Discussion of Discrete-time Fourier Transforms.
- Topics include comparison with analog transforms and discussion of Parseval's theorem.

The Fourier transform of the discrete-time signal  $\mathbf{s(n)}$  is defined to be

$$S(e^{i2\pi f}) = \sum_{n=-\infty}^{\infty} s(n)e^{-i2\pi fn}$$

Frequency here has no units. As should be expected, this definition is linear, with the transform of a sum of signals equaling the sum of their transforms. Real-valued signals have conjugate-symmetric spectra:

$$S(e^{-i2\pi f}) = \overline{S(e^{i2\pi f})}$$

### Exercise 7.1

A special property of the discrete-time Fourier transform is that it is periodic with period one:

$$S(e^{i2\pi(f+1)}) = S(e^{i2\pi f})$$

Derive this property from the definition of the DTFT.

#### Solution

$$\begin{aligned} S(e^{i2\pi(f+1)}) &= \sum_{n=-\infty}^{\infty} s(n)e^{-i2\pi(f+1)n} \\ S(e^{i2\pi(f+1)}) &= \sum_{n=-\infty}^{\infty} e^{-i2\pi n} s(n)e^{-i2\pi fn} \\ S(e^{i2\pi(f+1)}) &= \sum_{n=-\infty}^{\infty} s(n)e^{-i2\pi fn} \\ S(e^{i2\pi(f+1)}) &= S(e^{i2\pi f}) \end{aligned}$$

Because of this periodicity, we need only plot the spectrum over one period to understand completely the spectrum's structure; typically, we plot the spectrum over the frequency range

$$\left[-\frac{1}{2}, \frac{1}{2}\right]$$

When the signal is real-valued, we can further simplify our plotting chores by showing the spectrum only over

$$\left[0, \frac{1}{2}\right]$$

the spectrum at negative frequencies can be derived from positive-frequency spectral values.

When we obtain the discrete-time signal via sampling an analog signal, the **Nyquist frequency** corresponds to the discrete-time frequency **1/2**. To show this, note that a sinusoid having a frequency equal to the Nyquist frequency **1/2T<sub>s</sub>** has a sampled waveform that equals

$$\cos\left(2\pi \times \frac{1}{2T_s} nT_s\right) = \cos(\pi n) = (-1)^n$$

The exponential in the DTFT at frequency 1/2 equals

$$e^{-\frac{j2\pi n}{2}} = e^{-j\pi n} = (-1)^n$$

meaning that discrete-time frequency equals analog frequency multiplied by the sampling interval

$$f_D = f_A T_s$$

**f<sub>D</sub>** and **f<sub>A</sub>** represent discrete-time and analog frequency variables, respectively. The [aliasing figure](#) provides another way of deriving this result. As the duration of each pulse in the periodic sampling signal **p<sub>Ts</sub>(t)** narrows, the amplitudes of the signal's spectral repetitions, which are governed by the [Fourier series coefficients](#) of **p<sub>Ts</sub>(t)**, become increasingly equal. Examination of the [periodic pulse signal](#) reveals that as **Δ** decreases, the value of **c<sub>0</sub>**, the largest Fourier coefficient, decreases to zero:

$$|c_0| = \frac{A\Delta}{T_s}$$

Thus, to maintain a mathematically viable Sampling Theorem, the amplitude **A** must increase as **1/Δ**, becoming infinitely large as the pulse duration decreases. Practical systems use a small value of **Δ**, say **0.1T<sub>s</sub>** and use amplifiers to rescale the signal. Thus, the sampled signal's spectrum becomes periodic with period **1/T<sub>s</sub>**. Thus, the Nyquist frequency **1/2T<sub>s</sub>** corresponds to the frequency **½**.

### Example 7.2:

Let's compute the discrete-time Fourier transform of the exponentially decaying sequence

$$s(n) = a^n u(n)$$

where **u(n)** is the unit-step sequence. Simply plugging the signal's expression into the Fourier transform formula:

$$S(e^{j2\pi f}) = \sum_{n=-\infty}^{\infty} a^n u(n) e^{-j2\pi f n}$$

$$S(e^{j2\pi f}) = \sum_{n=0}^{\infty} (ae^{-j2\pi f n})^n$$

This sum is a special case of the **geometric series**.

$$\sum_{n=0}^{\infty} = \forall \alpha, |\alpha| < 1 : \left( \frac{1}{1 - \alpha} \right)$$

Thus, as long as **|a| < 1**, we have our Fourier transform.

$$S(e^{j2\pi f}) = \frac{1}{1 - ae^{-j2\pi f}}$$

Using Euler's relation, we can express the magnitude and phase of this spectrum.

$$|S(e^{j2\pi f})| = \frac{1}{\sqrt{(1 - a \cos(2\pi f))^2 + a^2 \sin^2(2\pi f)}}$$

$$\angle(S(e^{j2\pi f})) = -\tan^{-1} \left( \frac{a \sin 2\pi f}{1 - a \cos(2\pi f)} \right)$$

No matter what value of  $a$  we choose, the above formulae clearly demonstrate the periodic nature of the spectra of discrete-time signals. Figure 5.6.1 below shows indeed that the spectrum is a periodic function. We need only consider the spectrum between  $-\frac{1}{2}$  and  $\frac{1}{2}$  to unambiguously define it. When  $a > 0$ , we have a lowpass spectrum—the spectrum diminishes as frequency increases from  $0$  to  $\frac{1}{2}$ —with increasing  $a$  leading to a greater low frequency content; for  $a < 0$ , we have a highpass spectrum as shown in Figure 5.6.2 below.

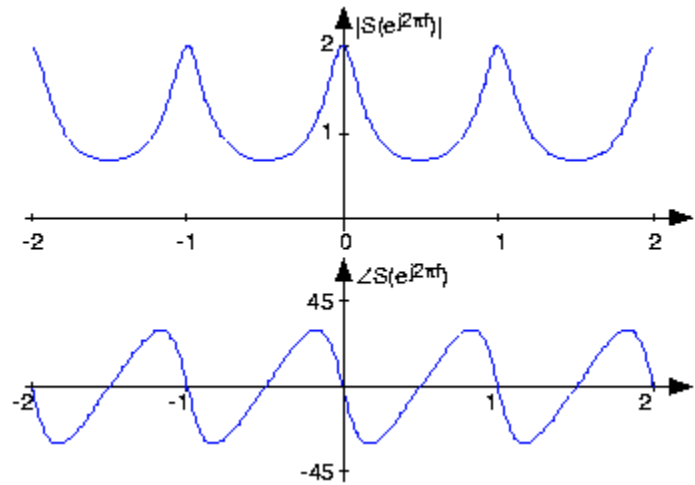


Figure 7.1: The spectrum of the exponential signal ( $a = 0.5$ ) is shown over the frequency range  $[-2, 2]$ , clearly demonstrating the periodicity of all discrete-time spectra. The angle has units of degrees.

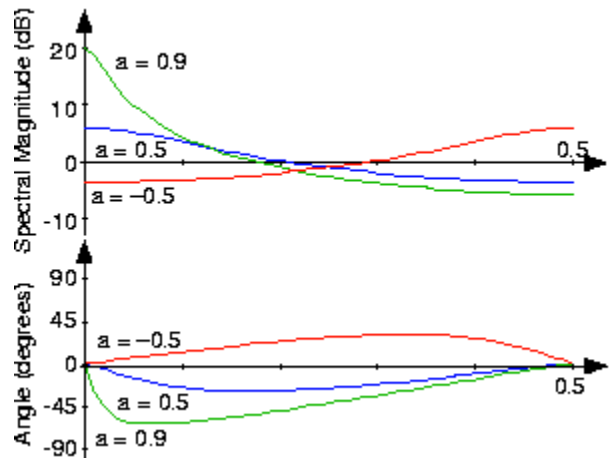


Figure 7.2: The spectra of several exponential signals are shown. What is the apparent relationship between the spectra for  $a = 0.5$  and  $a = -0.5$ ?

### Example 7.3:

Analogous to the analog pulse signal, let's find the spectrum of the length-  $N$  pulse sequence.

$$s(n) = \begin{cases} 1 & \text{if } 0 \leq n \leq N-1 \\ 0 & \text{if otherwise} \end{cases}$$

The Fourier transform of this sequence has the form of a truncated geometric series.

$$S(e^{j2\pi f}) = \sum_{n=0}^{N-1} e^{-j2\pi f n}$$

For the so-called finite geometric series, we know that

$$\sum_{n=n_0}^{N+n_0-1} \alpha^n = \alpha^{n_0} \frac{1 - \alpha^N}{1 - \alpha}$$

for **all** values of  $\alpha$ .

### Exercise 7.4

Derive this formula for the finite geometric series sum. The "trick" is to consider the difference between the series' sum and the sum of the series multiplied by  $\alpha$ .

#### Solution

$$\alpha \sum_{n=n_0}^{N+n_0-1} \alpha^n - \sum_{n=n_0}^{N+n_0-1} \alpha^n = \alpha^{N+n_0} - \alpha^{n_0}$$

which, after manipulation, yields the geometric sum formula.

Applying this result yields to Figure 5.6.3,

$$S(e^{i2\pi f}) = \frac{1 - e^{-(i2\pi fN)}}{1 - e^{-(i2\pi f)}}$$

$$S(e^{i2\pi f}) = e^{-(i\pi f(N-1))} \frac{\sin(\pi fN)}{\sin(\pi f)}$$

The ratio of sine functions has the generic form of

$$\frac{\sin(Nx)}{\sin(x)}$$

which is known as the **discrete-time sinc function dsinc (x)**. Thus, our transform can be concisely expressed as

$$S(e^{i2\pi f}) = e^{-(i\pi f(N-1))} dsinc(\pi f)$$

The discrete-time pulse's spectrum contains many ripples, the number of which increase with **N**, the pulse's duration.

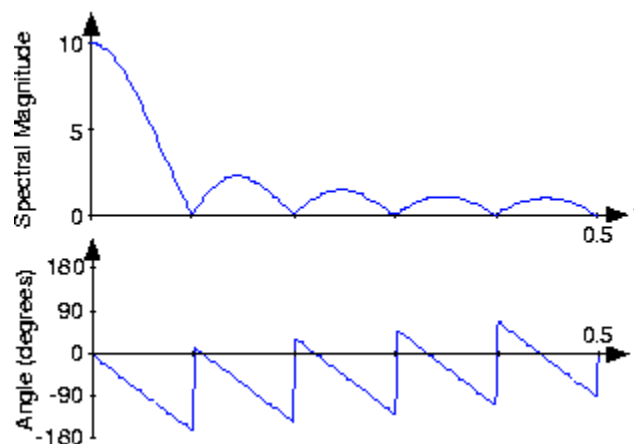


Figure 7.3: The spectrum of a length-ten pulse is shown. Can you explain the rather complicated appearance of the phase?

The inverse discrete-time Fourier transform is easily derived from the following relationship:

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-(i2\pi fm)} e^{i2\pi fn} df = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{if } m \neq n \end{cases}$$

$$\int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-(i2\pi fm)} e^{i2\pi fn} df = \delta(m - n)$$

Therefore, we find that

$$\begin{aligned} \int_{-\frac{1}{2}}^{\frac{1}{2}} S(e^{i2\pi f}) e^{i2\pi fn} df &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_m s(m) e^{-(i2\pi fm)} e^{i2\pi fn} df \\ \int_{-\frac{1}{2}}^{\frac{1}{2}} S(e^{i2\pi f}) e^{i2\pi fn} df &= \sum_m s(m) \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{-(i2\pi f)(m-n)} df \\ \int_{-\frac{1}{2}}^{\frac{1}{2}} S(e^{i2\pi f}) e^{i2\pi fn} df &= s(n) \end{aligned}$$

The Fourier transform pairs in discrete-time are

$$\begin{aligned} S(e^{i2\pi f}) &= \sum_{n=-\infty}^{\infty} s(n) e^{-(i2\pi fn)} \\ s(n) &= \int_{-\frac{1}{2}}^{\frac{1}{2}} S(e^{i2\pi f}) e^{i2\pi fn} df \end{aligned}$$

The properties of the discrete-time Fourier transform mirror those of the analog Fourier transform. The DTFT properties table below shows similarities and differences. One important common property is Parseval's Theorem.

### Summary Table of DTFT Properties

|                      | Sequence Domain                      | Frequency Domain                                          |
|----------------------|--------------------------------------|-----------------------------------------------------------|
| Linearity            | $a_1 s_1(n) + a_2 s_2(n)$            | $a_1 S_1(e^{i2\pi f}) + a_2 S_2(e^{i2\pi f})$             |
| Conjugate Symmetry   | $s(n)$ real                          | $S(e^{i2\pi f}) = \overline{S(e^{-i2\pi f})}$             |
| Even Symmetry        | $s(n) = s(-n)$                       | $S(e^{i2\pi f}) = S(e^{-i2\pi f})$                        |
| Odd Symmetry         | $s(n) = -s(-n)$                      | $S(e^{i2\pi f}) = -S(e^{-i2\pi f})$                       |
| Time Delay           | $s(n - n_0)$                         | $e^{-i2\pi f n_0} S(e^{i2\pi f})$                         |
| Multiplication by n  | $ns(n)$                              | $\frac{1}{-i2\pi} \frac{dS(e^{i2\pi f})}{df}$             |
| Sum                  | $\sum_{n=-\infty}^{\infty} s(n)$     | $S(e^{i2\pi 0})$                                          |
| Value at Origin      | $s(0)$                               | $\int_{-\frac{1}{2}}^{\frac{1}{2}} S(e^{i2\pi f}) df$     |
| Parseval's Theorem   | $\sum_{n=-\infty}^{\infty}  s(n) ^2$ | $\int_{-\frac{1}{2}}^{\frac{1}{2}}  S(e^{i2\pi f}) ^2 df$ |
| Complex Modulation   | $e^{i2\pi f_0 n} s(n)$               | $S(e^{i2\pi(f-f_0)})$                                     |
| Amplitude Modulation | $s(n) \cos(2\pi f_0 n)$              | $\frac{S(e^{i2\pi(f-f_0)}) + S(e^{i2\pi(f+f_0)})}{2}$     |
|                      | $s(n) \sin(2\pi f_0 n)$              | $\frac{S(e^{i2\pi(f-f_0)}) - S(e^{i2\pi(f+f_0)})}{2i}$    |

Discrete-Time Fourier Transform Properties Discrete-time Fourier transform properties and relations.

Figure 7.4: DTFT Properties

$$\sum_{n=-\infty}^{\infty} (|s(n)|)^2 = \int_{-\frac{1}{2}}^{\frac{1}{2}} |S(e^{i2\pi f})|^2 df$$

To show this important property, we simply substitute the Fourier transform expression into the frequency-domain expression for power.

$$\begin{aligned} \int_{-\frac{1}{2}}^{\frac{1}{2}} |S(e^{i2\pi f})|^2 df &= \int_{-\frac{1}{2}}^{\frac{1}{2}} \sum_n s(n) e^{-(i2\pi fn)} \sum_m \overline{s(m)} e^{i2\pi fm} df \\ \int_{-\frac{1}{2}}^{\frac{1}{2}} |S(e^{i2\pi f})|^2 df &= \sum_{n,m,n,m} s(n) \overline{s(m)} \int_{-\frac{1}{2}}^{\frac{1}{2}} e^{i2\pi f(m-n)} df \end{aligned}$$

Using the orthogonality relation, the integral equals  $\delta(\mathbf{m}-\mathbf{n})$ , where  $\delta(\mathbf{n})$  is the [unit sample](#). Thus, the double sum collapses into a single sum because nonzero values occur only when  $\mathbf{n}=\mathbf{m}$ , giving Parseval's Theorem as a result. We term

$$\sum_n s^2(n)$$

the energy in the discrete-time signal  $\mathbf{s}(\mathbf{n})$  in spite of the fact that discrete-time signals don't consume (or produce for that matter) energy. This terminology is a carry-over from the analog world.

### Exercise 7.5

Suppose we obtained our discrete-time signal from values of the product  $\mathbf{s}(\mathbf{t})\mathbf{pT_s}(\mathbf{t})$ , where the duration of the component pulses in  $\mathbf{pT_s}(\mathbf{t})$  is  $\Delta$ . How is the discrete-time signal energy related to the total energy contained in  $\mathbf{s}(\mathbf{t})$ ? Assume the signal is bandlimited and that the sampling rate was chosen appropriate to the Sampling Theorem's conditions.

#### Solution

If the sampling frequency exceeds the Nyquist frequency, the spectrum of the samples equals the analog spectrum, but over the normalized analog frequency  $\mathbf{fT}$ . Thus, the energy in the sampled signal equals the original signal's energy multiplied by  $\mathbf{T}$ .

## Contributor

- ContribEEOpenStax

---

This page titled [7: Discrete -Time Fourier Transform \(DTFT\)](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.6: Discrete -Time Fourier Transform \(DTFT\)](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 8: Discrete Fourier Transforms (DFT)

### Learning Objectives

- The Fourier transform can be computed in discrete-time despite the complications caused by a finite signal and continuous frequency.

The discrete-time Fourier transform (and the continuous-time transform as well) can be evaluated when we have an analytic expression for the signal. Suppose we just have a signal, such as the speech signal used in the previous chapter, for which there is no formula. How then would you compute the spectrum? For example, how did we compute a spectrogram such as the one shown in the [speech signal example](#)? The Discrete Fourier Transform (DFT) allows the computation of spectra from discrete-time data. While in discrete-time we can **exactly** calculate spectra, for analog signals no similar exact spectrum computation exists. For analog-signal spectra, we must build special devices, which turn out in most cases to consist of A/D converters and discrete-time computations. Certainly discrete-time spectral analysis is more flexible than continuous-time spectral analysis.

The formula for the **DTFT** is a sum, which conceptually can be easily computed save for two issues.

- Signal duration.** The sum extends over the signal's duration, which must be finite to compute the signal's spectrum. It is exceedingly difficult to store an infinite-length signal in any case, so we'll assume that the signal extends over **[0,N-1]**
- Continuous frequency.** Subtler than the signal duration issue is the fact that the frequency variable is continuous: It may only need to span one period, like  $[-\frac{1}{2}, \frac{1}{2}]$  or  $[0,1]$ , but the DTFT formula as it stands requires evaluating the spectra at **all** frequencies within a period. Let's compute the spectrum at a few frequencies; the most obvious ones are the equally spaced ones

$$f = \frac{k}{K}, k \in \{0, \dots, K-1\}$$

We thus define the **discrete Fourier transform** (DFT) to be

$$\forall k, k \in \{0, \dots, K-1\} : \left( S(k) = \sum_{n=0}^{N-1} s(n) e^{-\frac{i2\pi nk}{K}} \right)$$

Here

$$S(k) = S(e^{i2\pi \frac{k}{K}})$$

We can compute the spectrum at as many equally spaced frequencies as we like. Note that you can think about this computationally motivated choice as **sampling** the spectrum; more about this interpretation later. The issue now is how many frequencies are enough to capture how the spectrum changes with frequency. One way of answering this question is determining an inverse discrete Fourier transform formula: given **S(k), k={0,...,K-1}** how do we find **s(n), n={0,...,N-1}**? Presumably, the formula will be of the form

$$s(n) = \sum_{k=0}^{K-1} S(k) e^{\frac{i2\pi nk}{K}}$$

Substituting the DFT formula in this prototype inverse transform yields

$$s(n) = \sum_{k=0}^{K-1} \sum_{m=0}^{N-1} s(m) e^{-\left(\frac{i2\pi mk}{K}\right)} e^{\frac{i2\pi nk}{K}}$$

Note that the orthogonality relation we use so often has a different character now.

$$\sum_{k=0}^{K-1} e^{-\left(\frac{i2\pi mk}{K}\right)} e^{\frac{i2\pi nk}{K}} = \begin{cases} K & \text{if } m = \{n, n \pm K, n \pm 2K, \dots\} \\ 0 & \text{if otherwise} \end{cases}$$

We obtain nonzero value whenever the two indices differ by multiples of **K**. We can express this result as

$$K \sum_l \delta(m - n - lK)$$

Thus, our formula becomes

$$s(n) = \sum_{m=0}^{N-1} s(m) K \sum_{l=-\infty}^{\infty} \delta(m - n - lK)$$

The integers **n** and **m** both range over **{0,...,N-1}**. To have an inverse transform, we need the sum to be a **single** unit sample for **m,n** in this range. If it did not, then **s(n)** would equal a sum of values, and we would not have a valid transform: Once going into the frequency domain, we could not get back unambiguously! Clearly, the term **l=0** always provides a unit sample (we'll take care of the factor of **K** soon). If we evaluate the spectrum at fewer frequencies than the signal's duration, the term corresponding to **m=n+K** will also appear for some values of **m,n={0,...,N-1}**. This situation means that our prototype transform equals **s(n)+s(n+K)** for some values of **n**. The only way to eliminate this problem is to require

$$K \geq N$$

We **must** have at least as many frequency samples as the signal's duration. In this way, we can return from the frequency domain we entered via the DFT.

### Exercise 8.1

When we have fewer frequency samples than the signal's duration, some discrete-time signal values equal the sum of the original signal values. Given the sampling interpretation of the spectrum, characterize this effect a different way.

#### Solution

This situation amounts to aliasing in the time-domain.

Another way to understand this requirement is to use the theory of linear equations. If we write out the expression for the DFT as a set of linear equations,

$$\begin{aligned} s(0) + s(1) + \dots + s(N-1) &= S(0) \\ s(0) + s(1)e^{(-i)\frac{2\pi}{K}} + \dots + s(N-1)e^{(-i)\frac{2\pi(N-1)}{K}} &= S(1) \\ &\vdots \\ s(0) + s(1)e^{(-i)\frac{2\pi(K-1)}{K}} + \dots + s(N-1)e^{(-i)\frac{2\pi(N-1)(K-1)}{K}} &= S(K-1) \end{aligned}$$

we have **K** equations in **N** unknowns if we want to find the signal from its sampled spectrum. This requirement is impossible to fulfill if **K**; we must have

$$K \geq N$$

**Our orthogonality relation essentially says that if we have a sufficient number of equations (frequency samples), the resulting set of equations can indeed be solved.**

**By convention, the number of DFT frequency values K is chosen to equal the signal's duration N. The discrete Fourier transform pair consists of Discrete Fourier Transform Pair**

$$\begin{aligned} S(k) &= \sum_{n=0}^{N-1} s(n) e^{-i\frac{2\pi nk}{N}} \\ s(n) &= \frac{1}{N} \sum_{k=0}^{N-1} S(k) e^{i\frac{2\pi nk}{N}} \end{aligned}$$

---

This page titled [8: Discrete Fourier Transforms \(DFT\)](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.7: Discrete Fourier Transforms \(DFT\)](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 9: DFT - Computational Complexity

### Learning Objectives

- A brief explanation of calculation complexity and how the complexity of the discrete Fourier transform is order  $N$  squared.

We now have a way of computing the spectrum for an arbitrary signal: The Discrete Fourier Transform (**DFT**) computes the spectrum at  $N$  equally spaced frequencies from a length-  $N$  sequence. An issue that never arises in analog "computation," like that performed by a circuit, is how much work it takes to perform the signal processing operation such as filtering. In computation, this consideration translates to the number of basic computational steps required to perform the needed processing. The number of steps, known as the **complexity**, becomes equivalent to how long the computation takes (how long must we wait for an answer). Complexity is not so much tied to specific computers or programming languages but to how many steps are required on any computer. Thus, a procedure's stated complexity says that the time taken will be **proportional** to some function of the amount of data used in the computation and the amount demanded.

For example, consider the formula for the discrete Fourier transform. For each frequency we choose, we must multiply each signal value by a complex number and add together the results. For a real-valued signal, each real-times-complex multiplication requires two real multiplications, meaning we have  $2N$  multiplications to perform. To add the results together, we must keep the real and imaginary parts separate. Adding  $N$  numbers requires  $N-1$  additions. Consequently, each frequency requires

$$2N + 2(N - 1) = 4N - 2$$

basic computational steps. As we have  $N$  frequencies, the total number of computations is

$$N(4N - 2)$$

In complexity calculations, we only worry about what happens as the data lengths increase, and take the dominant term—here the  $4N^2$  term—as reflecting how much work is involved in making the computation. As multiplicative constants don't matter since we are making a "proportional to" evaluation, we find the DFT is an  **$O(N^2)$**  computational procedure. This notation is read "order  $N$ -squared". Thus, if we double the length of the data, we would expect that the computation time to approximately quadruple.

### Exercise 9.1

In making the complexity evaluation for the DFT, we assumed the data to be real. Three questions emerge. First of all, the spectra of such signals have conjugate symmetry, meaning that negative frequency components

$$k = \left\{ \frac{N}{2} + 1, \dots, N + 1 \right\} \quad (9.1)$$

in the **DFT**) can be computed from the corresponding positive frequency components. Does this symmetry change the DFT's complexity? Secondly, suppose the data are complex-valued; what is the DFT's complexity now? Finally, a less important but interesting question is suppose we want  $K$  frequency values instead of  $N$ ; now what is the complexity?

#### Solution

When the signal is real-valued, we may only need half the spectral values, but the complexity remains unchanged. If the data are complex-valued, which demands retaining all frequency values, the complexity is again the same. When only  $K$  frequencies are needed, the complexity is  **$O(KN)$** .

---

This page titled [9: DFT - Computational Complexity](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.8: DFT - Computational Complexity](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 10: Fast Fourier Transform (FFT)

## Learning Objectives

- The DFT can be reduced from exponential time with the Fast Fourier Transform algorithm.

One wonders if the DFT can be computed faster: Does another computational procedure -- an **algorithm** -- exist that can compute the same quantity, but more efficiently. We could seek methods that reduce the constant of proportionality, but do not change the DFT's complexity  $O(N^2)$ . Here, we have something more dramatic in mind: Can the computations be restructured so that a **smaller** complexity results?

In 1965, IBM researcher Jim Cooley and Princeton faculty member John Tukey developed what is now known as the Fast Fourier Transform (FFT). It is an algorithm for computing that DFT that has order  **$O(N \log N)$  for certain length inputs**. Now when the length of data doubles, the spectral computational time will not quadruple as with the DFT algorithm; instead, it approximately doubles. Later research showed that no algorithm for computing the DFT could have a smaller complexity than the FFT. Surprisingly, historical work has shown that **Gauss** in the early nineteenth century developed the same algorithm, but did not publish it! After the FFT's rediscovery, not only was the computation of a signal's spectrum greatly speeded, but also the added feature of **algorithm** meant that computations had flexibility not available to analog implementations.

## Exercise 10.1

Before developing the FFT, let's try to appreciate the algorithm's impact. Suppose a short-length transform takes 1 ms. We want to calculate a transform of a signal that is **10** times longer. Compare how much longer a straightforward implementation of the DFT would take in comparison to an FFT, both of which compute exactly the same quantity.

### Solution

If a DFT required 1ms to compute, and signal having ten times the duration would require 100ms to compute. Using the FFT, a 1ms computing time would increase by a factor of about

$$10 \log_2 10 = 33$$

A factor of 3 less than the DFT would have needed.

To derive the FFT, we assume that the signal's duration is a power of two:

$$N = 2^L$$

Consider what happens to the even-numbered and odd-numbered elements of the sequence in the DFT calculation.

$$S(k) = s(0) + s(2)e^{(-i)\frac{2\pi 2k}{N}} + \dots + s(N-2)e^{(-i)\frac{2\pi(N-2)k}{N}} + s(1)e^{(-i)\frac{2\pi k}{N}} + s(3)e^{(-i)\frac{2\pi \times (2+1)k}{N}} + \dots + s(N-1)e^{(-i)\frac{2\pi(N-1)k}{N}}$$

$$S(k) = \left[ s(0) + s(2)e^{(-i)\frac{2\pi 2k}{N}} + \dots + s(N-2)e^{(-i)\frac{2\pi(\frac{N}{2}-1)k}{N}} \right] + \left[ s(1)e^{(-i)\frac{2\pi k}{N}} + s(3)e^{(-i)\frac{2\pi k}{N}} + \dots + s(N-1)e^{(-i)\frac{2\pi(\frac{N}{2}-1)k}{N}} \right] e^{\frac{-i(2\pi k)}{N}}$$

Each term in square brackets has the **form** of a  **$N/2$**  length DFT. The first one is a DFT of the even-numbered elements, and the second of the odd-numbered elements. The first DFT is combined with the second multiplied by the complex exponential

$$e^{\frac{-i(2\pi k)}{N}}$$

The half-length transforms are each evaluated at frequency indices  **$k=0, \dots, N-1$** . Normally, the number of frequency indices in a DFT calculation range between zero and the transform length minus one. The **computational advantage** of the FFT comes from recognizing the periodic nature of the discrete Fourier transform. The FFT simply reuses the computations made in the half-length transforms and combines them through additions and the multiplication by

$$e^{\frac{-j(2\pi k)}{N}}$$

which is not periodic over  $N/2$ .

Figure 5.9.1 below illustrates this decomposition. As it stands, we now compute two length  $N/2$  transforms of complexity

$$2O\left(\frac{N^2}{4}\right)$$

multiply one of them by the complex exponential of complexity  $O(N)$  and add the results (complexity  $O(N)$ ). At this point, the total complexity is still dominated by the half-length DFT calculations, but the proportionality coefficient has been reduced.

Now for the fun. Because

$$N = 2^L$$

each of the half-length transforms can be reduced to two quarter-length transforms, each of these to two eighth-length ones, etc. This decomposition continues until we are left with length-2 transforms. This transform is quite simple, involving only additions. Thus, the first stage of the FFT has  $N/2$  length-2 transforms (see the bottom part of Figure 5.9.1). Pairs of these transforms are combined by adding one to the other multiplied by a complex exponential. Each pair requires 4 additions and 2 multiplications, giving a total number of computations equaling

$$6 \cdot \frac{N}{4} = \frac{3N}{2}$$

This number of computations does not change from stage to stage. Because the number of stages, the number of times the length can be divided by two, equals

$$\log_2 N$$

the number of arithmetic operations equals

$$\frac{3N}{2} \log_2 N$$

which makes the complexity of the FFT

$$O(N \log_2 N)$$

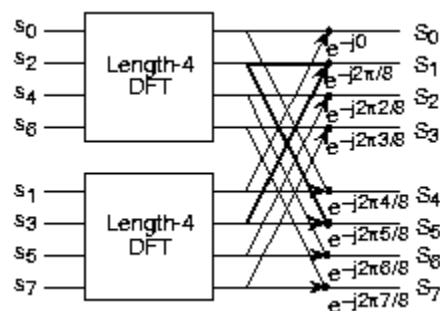


Figure 10.1a: Length-8 DFT decomposition

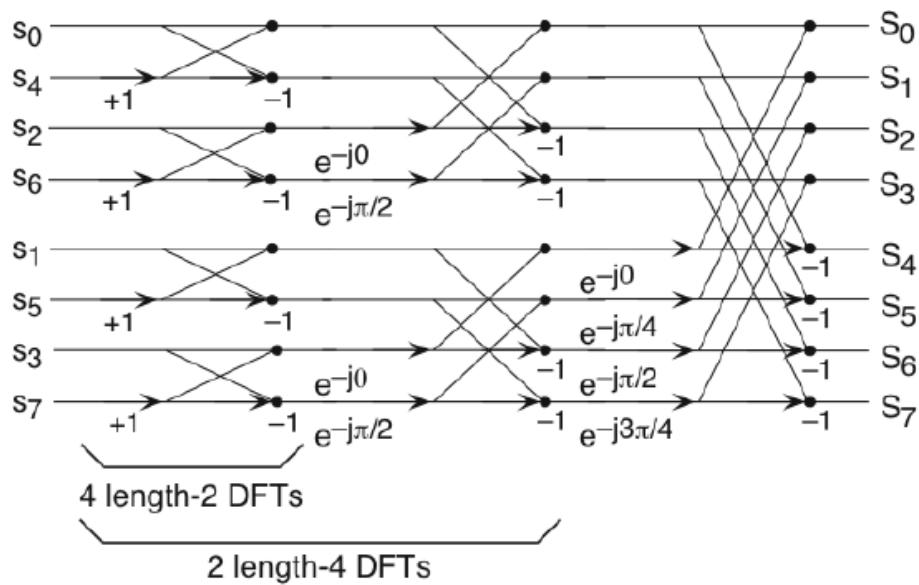


Figure 10.1b: Length-8 DFT computation

Figure 5.9.1 the initial decomposition of a length-8 DFT into the terms using even- and odd-indexed inputs marks the first phase of developing the FFT algorithm. When these half-length transforms are successively decomposed, we are left with the diagram shown in the bottom panel that depicts the length-8 FFT computation.

Doing an example will make computational savings more obvious. Let's look at the details of a length-8 DFT. As shown on Figure 5.9.1b, Doing an example will make computational savings more obvious. Let's look at the details of a length-8 DFT. Considering Figure 5.9.2 below, as the frequency index goes from 0 through 7, we recycle values from the length-4 DFTs into the final calculation because of the periodicity of the DFT output. Examining how pairs of outputs are collected together, we create the basic computational element known as a **butterfly** as shown in Figure 5.9.2.

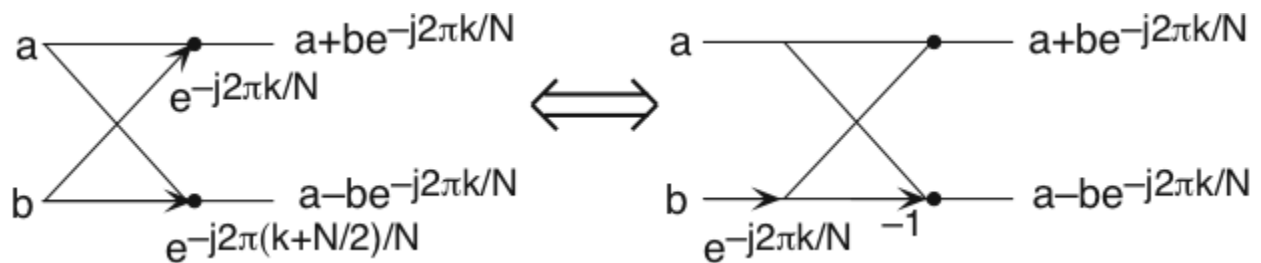


Figure 10.2: The basic computational element of the fast Fourier transform is the butterfly. It takes two complex numbers, represented by **a** and **b**, and forms the quantities shown. Each butterfly requires one complex multiplication and two complex additions.

By considering together the computations involving common output frequencies from the two half-length DFTs, we see that the two complex multiplies are related to each other, and we can reduce our computational work even further. By further decomposing the length-4 DFTs into two length-2 DFTs and combining their outputs, we arrive at the diagram summarizing the length-8 fast Fourier transform (Figure 5.9.1b). Although most of the complex multiplies are quite simple such as multiplying by

$$e^{-j(\frac{\pi}{4})}$$

means swapping real and imaginary parts and changing their signs. Let's count those for purposes of evaluating the complexity as full complex multiplies.

We have  $N/2 = 4$  complex multiplies and  $N = 8$  complex additions for each stage and

$$\log_2 N = 3$$

stages, making the number of basic computations

$$\frac{3N}{2} \log_2 N$$

as predicted.

### Exercise 10.2

Note that the ordering of the input sequence in the two parts of Figure 5.9.1 aren't quite the same. Why not? How is the ordering determined?

#### Solution

The upper panel has not used the FFT algorithm to compute the length-4 DFTs while the lower one has. The ordering is determined by the algorithm.

Other "fast" algorithms were discovered, all of which make use of how many common factors the transform length

### Exercise 10.3

Suppose the length of the signal were **500**? How would you compute the spectrum of this signal using the Cooley-Tukey algorithm? What would the length **N** of the transform be?

#### Solution

The transform can have **any** greater than or equal to the actual duration of the signal. We simply "pad" the signal with zero-valued samples until a computationally advantageous signal length results. Recall that the FFT is an **algorithm** to compute the **DFT**. Extending the length of the signal this way merely means we are sampling the frequency axis more finely than required. To use the Cooley-Tukey algorithm, the length of the resulting zero-padded signal can be 512, 1024, etc. samples long.

---

This page titled [10: Fast Fourier Transform \(FFT\)](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.9: Fast Fourier Transform \(FFT\)](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 11: Spectrograms

## Learning Objectives

- Spectrograms visually represent the speech signal, and the calculation of the Spectrogram is briefly explained.

We know how to acquire analog signals for digital processing ([pre-filtering](#), [sampling](#), and [A/D conversion](#)) and to compute spectra of discrete-time signals (using the [FFT algorithm](#)), let's put these various components together to learn how the spectrogram shown Figure 11.1 below, which is used to [analyze speech](#), is calculated. The speech was sampled at a rate of 11.025 kHz and passed through a 16-bit A/D converter.

## Example 11.1:

Music compact discs (CDs) encode their signals at a sampling rate of 44.1 kHz. We'll learn the rationale for this number later. The 11.025 kHz sampling rate for the speech is 1/4 of the CD sampling rate, and was the lowest available sampling rate commensurate with speech signal bandwidths available on my computer.

## Exercise 11.2

Looking at Figure 11.1, the signal lasted a little over 1.2 seconds. How long was the sampled signal (in terms of samples)? What was the datarate during the sampling process in bps (bits per second)? Assuming the computer storage is organized in terms of bytes (8-bit quantities), how many bytes of computer memory does the speech consume?

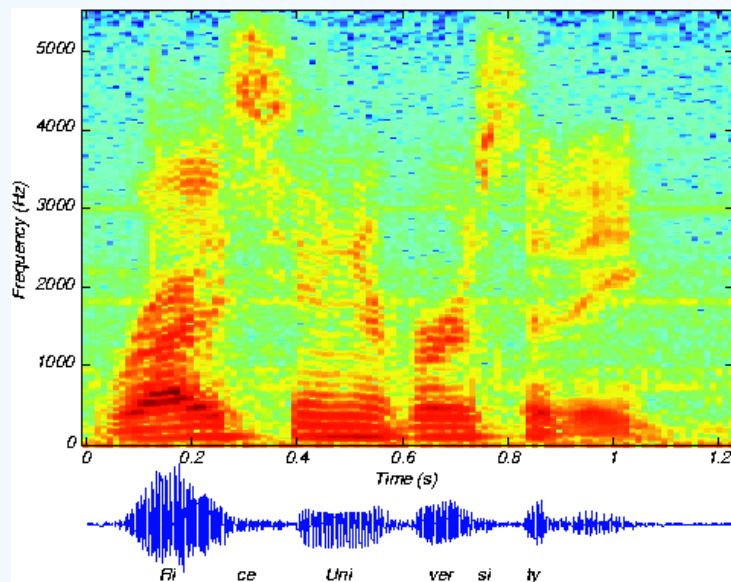


Figure 11.1: Spectrogram to analyze Speech

## Solution

Number of samples equals  $1.2 \times 11025 = 13230$

The datarate is  $11025 \times 16 = 176.4$  kbps

The storage required would be **26460** bytes

The resulting discrete-time signal, shown in the bottom of Figure 11.1 clearly changes its character with time. To display these spectral changes, the long signal was sectioned into **frames**: comparatively short, contiguous groups of samples. Conceptually, a Fourier transform of each frame is calculated using the FFT. Each frame is not so long that significant signal variations are retained within a frame, but not so short that we lose the signal's spectral character. Roughly speaking, the speech signal's spectrum is evaluated over successive time segments and stacked side by side so that the **x**-axis corresponds to time and the **y**-axis frequency, with color indicating the spectral amplitude.

An important detail emerges when we examine each framed signal from the above figure.

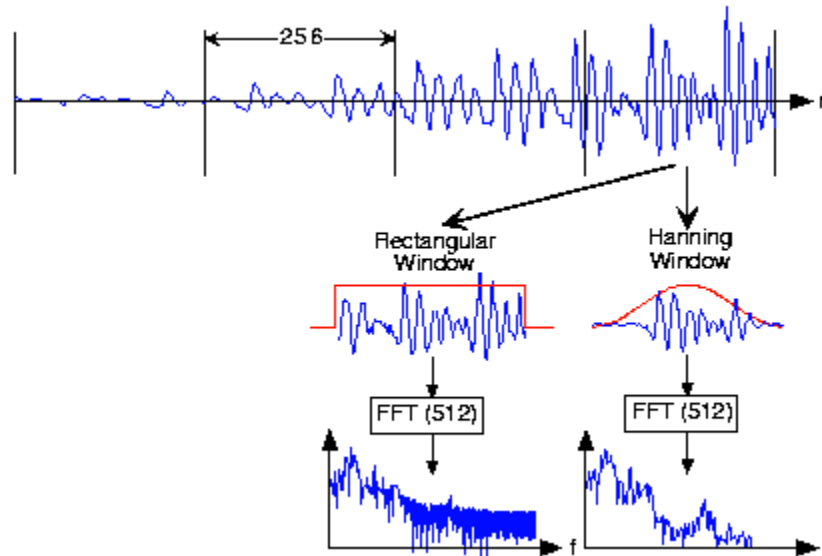


Figure 11.2: involved creating frames, here demarked by the vertical lines, that were 256 samples long and finding the spectrum of each. If a rectangular window is applied (corresponding to extracting a frame from the signal), oscillations appear in the spectrum (middle of bottom row). Applying a Hanning window gracefully tapers the signal toward frame edges, thereby yielding a more accurate computation of the signal's spectrum at that moment of time.

At the frame's edges, the signal may change very abruptly, a feature not present in the original signal. A transform of such a segment reveals a curious oscillation in the spectrum, an artifact directly related to this sharp amplitude change. A better way to frame signals for spectrograms is to apply a **window**: Shape the signal values within a frame so that the signal decays gracefully as it nears the edges. This shaping is accomplished by multiplying the framed signal by the sequence **w(n)**. In sectioning the signal, we essentially applied a rectangular window:

$$w(n) = 1, 0 \leq n \leq N - 1$$

A much more graceful window is the **Hanning window**; it has the cosine shape

$$w(n) = \frac{1}{2} \left( 1 - \cos \left( \frac{2\pi n}{N} \right) \right)$$

As shown in Figure 11.2, this shaping greatly reduces spurious oscillations in each frame's spectrum. Considering the spectrum of the Hanning windowed frame, we find that the oscillations resulting from applying the rectangular window obscured a formant (the one located at a little more than half the Nyquist frequency).

### Exercise 11.3

What might be the source of these oscillations? To gain some insight, what is the length-**2N** discrete Fourier transform of a length-**N** pulse? The pulse emulates the rectangular window, and certainly has edges. Compare your answer with the length-**2N** transform of a length-**N** Hanning window.

#### Solution

The oscillations are due to the boxcar window's Fourier transform, which equals the sinc function.

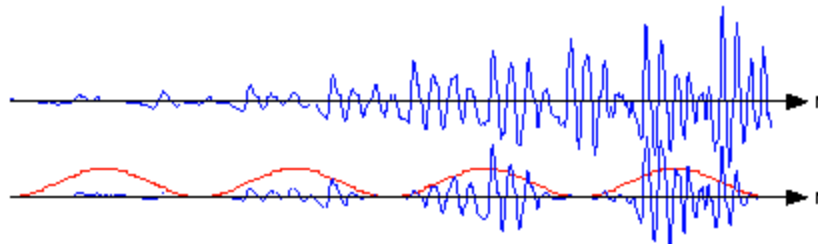


Figure 11.3: In comparison with the original speech segment shown in the upper plot, the non-overlapped Hanning windowed version shown below it is very ragged. Clearly, spectral information extracted from the bottom plot could well miss important features present in the original.

If you examine the windowed signal sections in sequence to examine windowing's effect on signal amplitude, we see that we have managed to amplitude-modulate the signal with the periodically repeated window (Figure 11.3). To alleviate this problem, frames are overlapped (typically by half a frame duration). This solution requires more Fourier transform calculations than needed by rectangular windowing, but the spectra are much better behaved and spectral changes are much better captured.

The speech signal, such as shown in the Figure 11.1 above, is sectioned into overlapping, equal-length frames, with a Hanning window applied to each frame. The spectra of each of these is calculated, and displayed in spectrograms with frequency extending vertically, window time location running horizontally, and spectral magnitude color-coded. Figure 5.10.4 illustrates these computations.

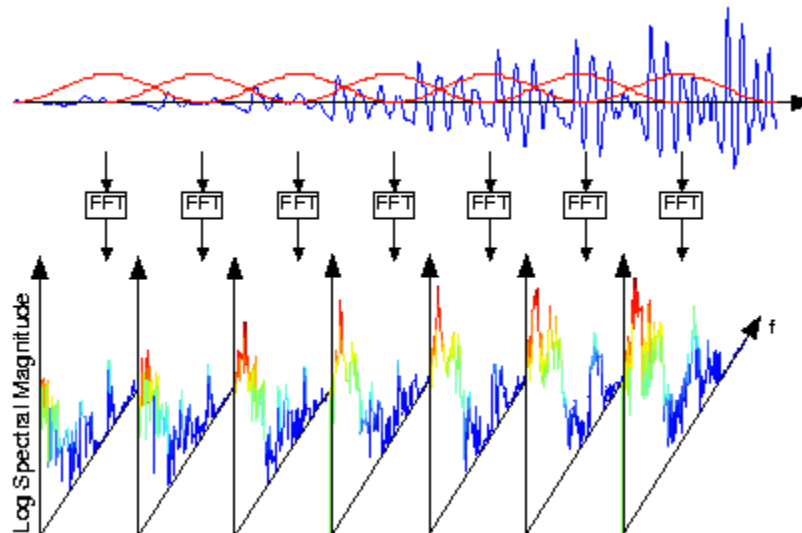


Figure 11.4: The original speech segment and the sequence of overlapping Hanning windows applied to it are shown in the upper portion. Frames were 256 samples long and a Hanning window was applied with a half-frame overlap. A length-512 FFT of each frame was computed, with the magnitude of the first 257 FFT values displayed vertically, with spectral amplitude values color-coded.

#### Exercise 11.4

Why the specific values of 256 for **N** and 512 for **K**? Another issue is how was the length-512 transform of each length-256 windowed frame computed?

#### Solution

These numbers are powers-of-two, and the FFT algorithm can be exploited with these lengths. To compute a longer transform than the input signal's duration, we simply zero-pad the signal.

## 12: Discrete-Time Systems

### Learning Objectives

- Discrete-time calculations are generally easier and more flexible than analog ones, allowing for practical calculation of Fourier transform.

When we developed analog systems, interconnecting the circuit elements provided a natural starting place for constructing useful devices. In discrete-time signal processing, we are not limited by hardware considerations but by what can be constructed in software.

### Exercise 12.1

One of the first analog systems we described was the [amplifier](#). We found that implementing an amplifier was difficult in analog systems, requiring an op-amp at least. What is the discrete-time implementation of an amplifier? Is this especially hard or easy?

#### Solution

In discrete-time signal processing, an amplifier amounts to a multiplication, a very easy operation to perform.

In fact, we will discover that frequency-domain implementation of systems, wherein we multiply the input signal's Fourier transform by a frequency response, is not only a viable alternative, but also a computationally efficient one. We begin with discussing the underlying mathematical structure of linear, shift-invariant systems, and devise how software filters can be constructed.

This page titled [12: Discrete-Time Systems](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.11: Discrete-Time Systems](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 13: Discrete-Time Systems in the Time-Domain

## Learning Objectives

- Discrete-time systems allow for mathematically specified processes like the difference equation.

A discrete-time signal  $s(n)$  is **delayed** by  $n_0$  samples when we write  $s(n - n_0)$  with  $n_0 > 0$ . Choosing  $n_0$  to be negative advances the signal along the integers. As opposed to **analog delays**, discrete-time delays can **only** be integer valued. In the frequency domain, delaying a signal corresponds to a linear phase shift of the signal's discrete-time Fourier transform:

$$s(n - n_0) \leftrightarrow e^{-j2\pi f n_0} S(e^{j2\pi f})$$

**Linear** discrete-time systems have the superposition property.

$$S(a_1 x_1(n) + a_2 x_2(n)) = a_1 S(x_1(n)) + a_2 S(x_2(n))$$

A discrete-time system is called **shift-invariant** (analogous to **time-invariant analog systems**) if delaying the input delays the corresponding output. If

$$S(x(n)) = y(n)$$

Then a shift-invariant system has the property

$$S(x(n - n_0)) = y(n - n_0)$$

We use the term shift-invariant to emphasize that delays can only have integer values in discrete-time, while in analog signals, delays can be arbitrarily valued.

We want to concentrate on systems that are both linear and shift-invariant. It will be these that allow us the full power of frequency-domain analysis and implementations. Because we have no physical constraints in "constructing" such systems, we need only a mathematical specification. In analog systems, the differential equation specifies the input-output relationship in the time-domain. The corresponding discrete-time specification is the **difference equation**.

$$y(n) = a_1 y(n-1) + \dots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \dots + b_q x(n-q)$$

Here, the output signal  $y(n)$  is related to its **past** values

$$y(n-l), l = \{1, \dots, p\}$$

and to the current and past values of the input signal  $x(n)$ . The system's characteristics are determined by the choices for the number of coefficients  $p$  and  $q$  and the coefficients' values

$$\{a_1, \dots, a_p\} \text{ and } \{b_0, b_1, \dots, b_q\}$$

## Note

There is an asymmetry in the coefficients: where is  $a_0$ ? This coefficient would multiply the  $y(n)$  term in the above equation. We have essentially divided the equation by it, which does not change the input-output relationship. We have thus created the convention that  $a_0$  is always one.

As opposed to differential equations, which only provide an **implicit** description of a system (we must somehow solve the differential equation), difference equations provide an **explicit** way of computing the output for any input. We simply express the difference equation by a program that calculates each output from the previous output values, and the current and previous inputs.

Difference equations are usually expressed in software with **for** loops. A MATLAB program that would compute the first 1000 values of the output has the form `for n=1:1000 y(n) = sum(a.*y(n-1:-1:n-p)) + sum(b.*x(n:-1:n-q)); end`. An important detail emerges when we consider making this program work; in fact, as written it has (at least) two bugs. What input and output values enter into the computation of **y(1)**? We need values for **y(0)**, **y(-1)**,..., values we have not yet computed. To compute them, we would need more previous values of the output, which we have not yet computed. To compute these values, we would need even earlier values, ad infinitum. The way out of this predicament is to specify the system's **initial conditions**: we must provide the **p** output values that occurred before the input started. These values can be arbitrary, but the choice does impact how the system responds to a given input. **One** choice gives rise to a linear system: Make the initial conditions zero. The reason lies in the definition of a **linear system**: The only way that the output to a sum of signals can be the sum of the individual outputs occurs when the initial conditions in each case are zero.

### Exercise 13.1

The initial condition issue resolves making sense of the difference equation for inputs that start at some index. However, the program will not work because of a programming, not conceptual, error. What is it? How can it be "fixed?"

#### Solution

The indices can be negative, and this condition is not allowed in MATLAB. To fix it, we must start the signals later in the array.

### Example 13.1

Let's consider the simple system having  $p = 1$  and  $q = 0$ .

$$y(n) = ay(n-1) + bx(n)$$

To compute the output at some index, this difference equation says we need to know what the previous output **y(n-1)** and what the input signal is at that moment of time. In more detail, let's compute this system's output to a unit-sample input:

$$x(n) = \delta(n)$$

Because the input is zero for negative indices, we start by trying to compute the output at **n = 0**.

$$y(0) = ay(-1) + b$$

What is the value of **y(-1)**? Because we have used an input that is zero for all negative indices, it is reasonable to assume that the output is also zero. Certainly, the difference equation would not describe a **linear system** if the input that is zero for **all** time did not produce a zero output. With this assumption, **y(-1) = 0**, leaving **y(0) = b**. For **n > 0**, the input unit-sample is zero, which leaves us with the difference equation

$$\forall n, n > 0 : (y(n) = ay(n-1))$$

We can envision how the filter responds to this input by making a table.

$$y(n) = ay(n-1) + b\delta(n)$$

| n  | x(n) | y(n)            |
|----|------|-----------------|
| -1 | 0    | 0               |
| 0  | 1    | b               |
| 1  | 0    | ba              |
| 2  | 0    | ba <sup>2</sup> |
| :  | 0    | :               |
| n  | 0    | ba <sup>n</sup> |

Coefficient values determine how the output behaves. The parameter **b** can be any value, and serves as a gain. The effect of the parameter **a** is more complicated (see Table above). If it equals zero, the output simply equals the input times the gain **b**. For all non-zero values of **a**, the output lasts forever; such systems are said to be **IIR** (**I**nfinite **I**mpulse **R**esponse). The reason for this terminology is that the unit sample also known as the impulse (especially in analog situations), and the system's response to the "impulse" lasts forever. If **a** is positive and less than one, the output is a decaying exponential. When **a = 1**, the output is a unit step. If **a** is negative and greater than **-1**, the output oscillates while decaying exponentially. When **a = -1**, the output changes sign forever, alternating between **b** and **-b**. More dramatic effects when **|a| > 1**; whether positive or negative, the output signal becomes larger and larger, **growing** exponentially.

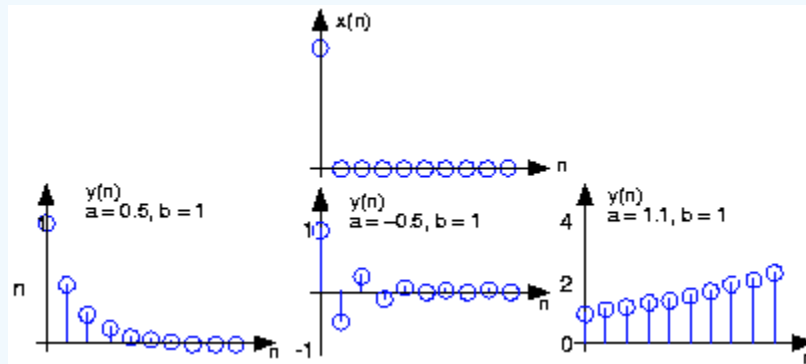


Figure 13.1: The input to the simple example system, a unit sample, is shown at the top, with the outputs for several system parameter values shown below.

Positive values of

### Exercise 13.2

Note that the difference equation

$$y(n) = a_1 y(n-1) + \dots + a_p y(n-p) + b_0 x(n) + b_1 x(n-1) + \dots + b_q x(n-q)$$

does not involve terms like  $y(n+1)$  or  $x(n+1)$  on the equation's right side. Can such terms also be included? Why or why not?

#### Solution

Such terms would require the system to know what future input or output values would be before the current value was computed. Thus, such terms can cause difficulties.

### Example 13.2:

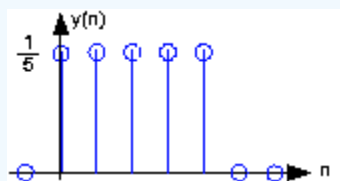


Figure 13.2: The plot shows the unit-sample response of a length-5 boxcar filter.

A somewhat different system has no "**a**" coefficients. Consider the difference equation

$$y(n) = \frac{1}{q} (x(n) + \dots + x(n-q+1))$$

Because this system's output depends only on current and previous input values, we need not be concerned with initial conditions. When the input is a unit-sample, the output equals

$$\frac{1}{q} \text{ for } n = \{0, \dots, q - 1\}$$

then equals zero thereafter. Such systems are said to be **FIR (Finite Impulse Response)** because their unit sample responses have finite duration. Plotting this response (Figure 13.2:) shows that the unit-sample response is a pulse of width **q** and height **1/q**. This waveform is also known as a boxcar, hence the name **boxcar filter** given to this system. We'll derive its frequency response and develop its filtering interpretation in the next section. For now, note that the difference equation says that each output value equals the **average** of the input's current and previous values. Thus, the output equals the running average of input's previous **q** values. Such a system could be used to produce the average weekly temperature (**q = 7**) that could be updated daily.

## Contributor

- ContribEEOpenStax

---

This page titled [13: Discrete-Time Systems in the Time-Domain](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.12: Discrete-Time Systems in the Time-Domain](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 14: Discrete -Time Systems in the Frequency Domain

## Learning Objectives

- As with analog linear systems, we need to find the frequency response of discrete-time systems.

As with analog linear systems, we need to find the frequency response of discrete-time systems. We used impedances to derive directly from the circuit's structure the frequency response. The only structure we have so far for a discrete-time system is the difference equation. We proceed as when we used impedances: let the input be a complex exponential signal. When we have a linear, shift-invariant system, the output should also be a complex exponential of the same frequency, changed in amplitude and phase. These amplitude and phase changes comprise the frequency response we seek. The complex exponential input signal is

$$x(n) = X e^{i2\pi f n}$$

Note that this input occurs for **all** values of **n**. No need to worry about initial conditions here. Assume the output has a similar form:

$$y(n) = Y e^{i2\pi f n}$$

Plugging these signals into the [fundamental difference equation](#), we have

$$Y e^{i2\pi f n} = a_1 Y e^{i2\pi f (n-1)} + \dots + a_p Y e^{i2\pi f (n-p)} + b_0 X e^{i2\pi f n} + b_1 X e^{i2\pi f (n-1)} + \dots + b_q X e^{i2\pi f (n-q)}$$

The assumed output does indeed satisfy the difference equation if the output complex amplitude is related to the input amplitude by

$$Y = \frac{b_0 + b_1 e^{-(i2\pi f)} + \dots + b_q e^{-(i2\pi q f)}}{1 - a_1 e^{-(i2\pi f)} - \dots - a_p e^{-(i2\pi p f)}} X$$

This relationship corresponds to the system's frequency response or, by another name, its transfer function. We find that any discrete-time system defined by a difference equation has a transfer function given by

$$H(e^{i2\pi f}) = \frac{b_0 + b_1 e^{-(i2\pi f)} + \dots + b_q e^{-(i2\pi q f)}}{1 - a_1 e^{-(i2\pi f)} - \dots - a_p e^{-(i2\pi p f)}}$$

Furthermore, because **any** discrete-time signal can be expressed as a superposition of complex exponential signals and because linear discrete-time systems obey the Superposition Principle, the transfer function relates the discrete-time Fourier transform of the system's output to the input's Fourier transform.

$$Y(e^{i2\pi f}) = X(e^{i2\pi f}) H(e^{i2\pi f})$$

## Example 14.1:

The frequency response of the simple IIR system

$$H(e^{i2\pi f}) = \frac{b}{1 - a e^{-(i2\pi f)}}$$

This Fourier transform occurred in a previous example; the [exponential signal spectrum](#) portrays the magnitude and phase of this transfer function. When the filter coefficient **a** is positive, we have a lowpass filter; negative **a** results in a highpass filter. The larger the coefficient in magnitude, the more pronounced the lowpass or highpass filtering.

### Example 14.2:

The length- $q$  boxcar filter has the frequency response

$$H(e^{i2\pi f}) = \frac{1}{q} \sum_{m=0}^{q-1} e^{-i2\pi f m}$$

This expression amounts to the Fourier transform of the **boxcar signal**.

There we found that this frequency response has a magnitude equal to the absolute value of **dsinc( $\pi f$ )**. See the **length-10 filter's frequency response**. We see that boxcar filters--length- $q$  signal averagers--have a lowpass behavior, having a cutoff frequency of  **$1/q$** .

### Exercise 14.1

Suppose we multiply the boxcar filter's coefficients by a sinusoid:

$$b_m = \frac{1}{q} \cos(2\pi f_0 m)$$

Use Fourier transform properties to determine the transfer function. How would you characterize this system: Does it act like a filter? If so, what kind of filter and how do you control its characteristics with the filter's coefficients?

#### Solution

It now acts like a bandpass filter with a center frequency of  **$f_0$**  and a bandwidth equal to **twice** of the original lowpass filter.

These examples illustrate the point that systems described (and implemented) by difference equations serve as filters for discrete-time signals. The filter's **order** is given by the number  **$p$**  of denominator coefficients in the transfer function (if the system is IIR) or by the number  **$q$**  of numerator coefficients if the filter is FIR. When a system's transfer function has both terms, the system is usually IIR, and its order equals  **$p$**  regardless of  **$q$** . By selecting the coefficients and filter type, filters having virtually any frequency response desired can be designed. This design flexibility can't be found in analog systems. In the next section, we detail how analog signals can be filtered by computers, offering a much greater range of filtering possibilities than is possible with circuits.

This page titled **14: Discrete -Time Systems in the Frequency Domain** is shared under a **CC BY-SA 1.0** license and was authored, remixed, and/or curated by **Carl Greco** via **source content** that was edited to the style and standards of the LibreTexts platform.

- **5.13: Discrete -Time Systems in the Frequency Domain** by **Don H. Johnson** is licensed **CC BY 1.0**. Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 15: Filtering in the Frequency Domain

## Learning Objectives

- Investigation of different aspects of filtering in the frequency domain, particularly the use of discrete Fourier transforms.

Because we are interested in actual computations rather than analytic calculations, we must consider the details of the discrete Fourier transform. To compute the length- $N$  DFT, we assume that the signal has a duration less than or equal to  $N$ . Because frequency responses have an explicit [frequency-domain specification](#) in terms of filter coefficients, we don't have a direct handle on which signal has a Fourier transform equaling a given frequency response. Finding this signal is quite easy. First of all, note that the discrete-time Fourier transform of a unit sample equals one for all frequencies. Since the input and output of linear, shift-invariant systems are related to each other by

$$Y(e^{i2\pi f}) = H(e^{i2\pi f})X(e^{i2\pi f})$$

**A unit-sample input, which has**

$$X(e^{i2\pi f}) = 1$$

**results in the output's Fourier transform equaling the system's transfer function.**

## Exercise 15.1

The above statement is a very important result. Derive it yourself.

### Solution

The DTFT of the unit sample equals a constant (equaling 1). Thus, the Fourier transform of the output equals the transfer function.

In the time-domain, the output for a unit-sample input is known as the system's **unit-sample response**, and is denoted by  $h(n)$ . Combining the frequency-domain and time-domain interpretations of a linear, shift-invariant system's unit-sample response, we have that  $h(n)$  and the transfer function are Fourier transform pairs **in terms of the discrete-time Fourier transform**.

$$h(n) \leftrightarrow H(e^{i2\pi f})$$

Returning to the issue of how to use the DFT to perform filtering, we can analytically specify the frequency response, and derive the corresponding length- $N$  DFT by sampling the frequency response.

$$\forall k, k = \{0, \dots, N-1\} : (H(k) = H(e^{i2\pi k/N}))$$

Computing the inverse DFT yields a length- $N$  signal **no matter what the actual duration of the unit-sample response might be**. If the unit-sample response has a duration less than or equal to  $N$  (it's a FIR filter), computing the inverse DFT of the sampled frequency response indeed yields the unit-sample response. If, however, the duration exceeds  $N$ , errors are encountered. The nature of these errors is easily explained by appealing to the Sampling Theorem. By sampling in the frequency domain, we have the potential for aliasing in the time domain (sampling in one domain, be it time or frequency, can result in aliasing in the other) unless we sample fast enough. Here, the duration of the unit-sample response determines the minimal sampling rate that prevents aliasing. For FIR systems — they by definition have finite-duration unit sample responses — the number of required DFT samples equals the unit-sample response's duration:

$$N \geq q$$

## Exercise 15.1

Derive the minimal DFT length for a length- $q$  unit-sample response using the Sampling Theorem. Because sampling in the frequency domain causes repetitions of the unit-sample response in the time domain, sketch the time-domain result for various choices of the DFT length  $N$ .

### Solution

In sampling a discrete-time signal's Fourier transform  $L$  times equally over  $[0, 2\pi)$  to form the DFT, the corresponding signal equals the periodic repetition of the original signal.

$$S(k) \leftrightarrow \sum_{i=-\infty}^{\infty} s(n - iL)$$

To avoid aliasing (in the time domain), the transform length must equal or exceed the signal's duration.

## Exercise 15.1

Express the unit-sample response of a FIR filter in terms of difference equation coefficients. Note that the corresponding question for IIR filters is far more difficult to answer: Consider the [example](#).

### Solution

The difference equation for an FIR filter has the form

$$y(n) = \sum_{m=0}^q b_m x(n - m)$$

The unit-sample response equals

$$h(n) = \sum_{m=0}^q b_m \delta(n - m)$$

which corresponds to the representation described in [a problem](#) of a length- $q$  boxcar filter.

For IIR systems, we cannot use the DFT to find the system's unit-sample response: aliasing of the unit-sample response will **always** occur. Consequently, we can only implement an IIR filter accurately in the time domain with the system's difference equation. **Frequency-domain implementations are restricted to FIR filters.**

Another issue arises in frequency-domain filtering that is related to time-domain aliasing, this time when we consider the output. Assume we have an input signal having duration  $N_x$  that we pass through a FIR filter having a length- $q+1$  unit-sample response. What is the duration of the output signal? The difference equation for this filter is

$$y(n) = b_0 x(n) + \dots + b_q x(n - q)$$

This equation says that the output depends on current and past input values, with the input value  $q$  samples previous defining the extent of the filter's **memory** of past input values. For example, the output at index  $N_x$  depends on  $x(N_x)$  (which equals zero),

$$x(N_x - 1) \text{ through } x(N_x - q)$$

Thus, the output returns to zero only after the last input value passes through the filter's memory. As the input signal's last value occurs at index  $N_x - 1$ , the last nonzero output value occurs when

$$n - q = N_x - 1 \text{ or } n = q + N_x - 1$$

Thus, the output signal's duration equals  $q + N_x$ .

## Exercise 15.1

In words, we express this result as "The output's duration equals the input's duration plus the filter's duration minus one.". Demonstrate the accuracy of this statement.

### Solution

The unit-sample response's duration is  $q + 1$  and the signal's  $N_x$ . Thus the statement is correct.

The main theme of this result is that a filter's output extends longer than either its input or its unit-sample response. Thus, to avoid aliasing when we use DFTs, the dominant factor is not the duration of input or of the unit-sample response, but of the output. Thus, the number of values at which we must evaluate the frequency response's DFT must be at least  $q + N_x$  and we must compute the same length DFT of the input. To accommodate a shorter signal than DFT length, we simply **zero-pad** the input: Ensure that for indices extending beyond the signal's duration that the signal is zero. Frequency-domain filtering, as shown in Figure 5.14.1 below, is accomplished by storing the filter's frequency response as the DFT  $H(k)$ , computing the input's DFT  $X(k)$ , multiplying them to create the output's DFT

$$Y(k) = H(k)X(k)$$

and computing the inverse DFT of the result to yield  $y(n)$ .

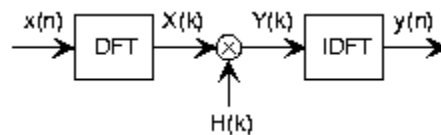


Figure 5.14.1 To filter a signal in the frequency domain, first compute the DFT of the input, multiply the result by the sampled frequency response, and finally compute the inverse DFT of the product. The DFT's length **must** be at least the sum of the input's and unit-sample response's duration minus one. We calculate these discrete Fourier transforms using the fast Fourier transform algorithm, of course.

Before detailing this procedure, let's clarify why so many new issues arose in trying to develop a frequency-domain implementation of linear filtering. The frequency-domain relationship between a filter's input and output is **always** true:

$$Y(e^{i2\pi f}) = H(e^{i2\pi f})X(e^{i2\pi f})$$

The Fourier transforms in this result are discrete-time Fourier transforms; for example,

$$X(e^{i2\pi f}) = \sum_n x(n)e^{-i2\pi fn}$$

Unfortunately, using this relationship to perform filtering is restricted to the situation when we have analytic formulas for the frequency response and the input signal. The reason why we had to "invent" the discrete Fourier transform (DFT) has the same origin: The spectrum resulting from the discrete-time Fourier transform depends on the **continuous** frequency variable  $f$ . That's fine for analytic calculation, but computationally we would have to make an uncountably infinite number of computations.

### Note

Did you know that two kinds of infinities can be meaningfully defined? A **countably infinite** quantity means that it can be associated with a limiting process associated with integers. An **uncountably infinite** quantity cannot be so associated. The number of rational numbers is countably infinite (the numerator and denominator correspond to locating the rational by row and column; the total number so-located can be counted, voila!); the number of irrational numbers is uncountably infinite. Guess which is "bigger?"

The DFT computes the Fourier transform at a finite set of frequencies — samples the true spectrum — which can lead to aliasing in the time-domain unless we sample sufficiently fast. The sampling interval here is  $1/K$  for a length- $K$  DFT: faster sampling to avoid aliasing thus requires a longer transform calculation. Since the longest signal among the input, unit-sample response and output is the output, it is that signal's duration that determines the transform length. We simply extend the other two signals with

zeros (zero-pad) to compute their DFTs.

### Example 15.1:

Suppose we want to average daily stock prices taken over last year to yield a running weekly average (average over five trading sessions). The filter we want is a length-5 averager (as shown in the [unit-sample response](#)), and the input's duration is 253 (365 calendar days minus weekend days and holidays). The output duration will be:

$$253 + 5 - 1 = 257$$

This determines the transform length we need to use. Because we want to use the FFT, we are restricted to power-of-two transform lengths. We need to choose any FFT length that exceeds the required DFT length. As it turns out, 256 is a power of two

$$2^8 = 256$$

This length just undershoots our required length. To use frequency domain techniques, we must use length-512 fast Fourier transforms.

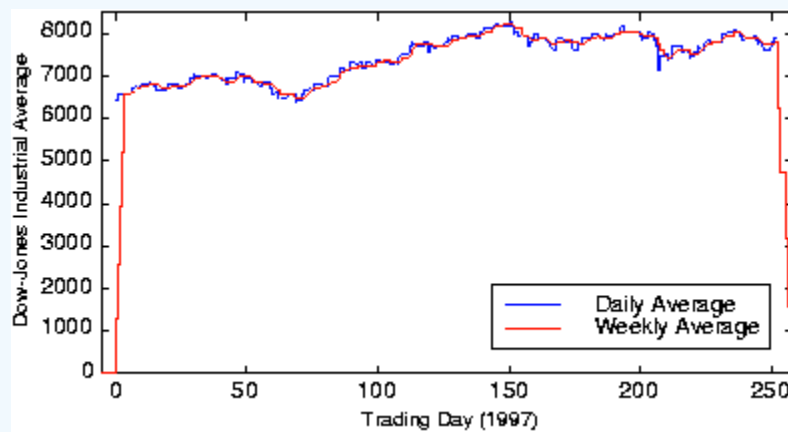


Figure 5.14.2 The blue line shows the Dow Jones Industrial Average from 1997, and the red one the length-5 boxcar-filtered result that provides a running weekly of this market index. Note the "edge" effects in the filtered output.

Figure 5.14.2 above, shows the input and the filtered output. The MATLAB programs that compute the filtered output in the time and frequency domains are:

```
Time Domain
h = [1 1 1 1 1]/5;
y = filter(h,1,[djia zeros(1,4)]);

Frequency Domain
h = [1 1 1 1 1]/5;
DJIA = fft(djia, 512);
H = fft(h, 512);
Y = H.*X;
y = ifft(Y);
```

### Note

The **filter** program has the feature that the length of its output equals the length of its input. To force it to produce a signal having the proper length, the program zero-pads the input appropriately.

MATLAB's **fft** function automatically zero-pads its input if the specified transform length (its second argument) exceeds the signal's length. The frequency domain result will have a small imaginary component — largest value is  $2.2 \times 10^{-11}$  — because of the inherent finite precision nature of computer arithmetic. Because of the unfortunate misfit between signal lengths and favored FFT lengths, the number of arithmetic operations in the time-domain implementation is far less than those required by the frequency domain version: 514 versus 62,271. If the input signal had been one sample shorter, the frequency-domain computations would have been more than a factor of two less (28,696), but far more than in the time-domain implementation.

An interesting signal processing aspect of this example is demonstrated at the beginning and end of the output. The ramping up and down that occurs can be traced to assuming the input is zero before it begins and after it ends. The filter "sees" these initial and final values as the difference equation passes over the input. These artifacts can be handled in two ways: we can just ignore the edge effects or the data from previous and succeeding years' last and first week, respectively, can be placed at the ends.

---

This page titled [15: Filtering in the Frequency Domain](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by Carl Greco via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.14: Filtering in the Frequency Domain](#) by Don H. Johnson is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

## 16: Efficiency of Frequency - Domain Filtering

### Learning Objectives

- Compares the efficiency of frequency domain and time domain filtering.

To determine for what signal and filter durations a time- or frequency-domain implementation would be the most efficient, we need only count the computations required by each. For the time-domain, difference-equation approach, we need

$$(N_x + q)(2(q) + 1)$$

The frequency-domain approach requires three Fourier transforms, each requiring

$$\frac{5K}{2}(\log_2 K)$$

computations for a length-**K** FFT, and the multiplication of two spectra (**6K** computations). The output-signal-duration-determined length must be at least

$$N_x + q$$

Thus, we must compare

$$(N_x + q)(2q + 1) \leftrightarrow 6(N_x + q) + 5(N_x + q) \log_2(N_x + q)$$

Exact analytic evaluation of this comparison is quite difficult (we have a transcendental equation to solve). Insight into this comparison is best obtained by dividing by  **$N_x + q$** .

$$2q + 1 \leftrightarrow 6 + 5 \log_2(N_x + q)$$

With this manipulation, we are evaluating the number of computations per sample. For any given value of the filter's order  **$q$** , the right side, the number of frequency-domain computations, will exceed the left if the signal's duration is long enough. However, for filter durations greater than about **10**, as long as the input is at least **10** samples, the frequency-domain approach is faster **so long as the FFT's power-of-two constraint is advantageous**.

The frequency-domain approach is not yet viable; what will we do when the input signal is infinitely long? The difference equation scenario fits perfectly with the [envisioned digital filtering structure](#), but so far we have required the input to have limited duration (so that we could calculate its Fourier transform). The solution to this problem is quite simple: Section the input into frames, filter each, and add the results together. To section a signal means expressing it as a linear combination of length- **$N_x$**  non-overlapping "chunks." Because the filter is linear, filtering a sum of terms is equivalent to summing the results of filtering each term.

$$\left( x(n) = \sum_{m=-\infty}^{\infty} x(n - mN_x) \right) \Rightarrow \left( y(n) = \sum_{m=-\infty}^{\infty} y(n - mN_x) \right)$$

As illustrated in Figure 16.1 below, note that each filtered section has a duration longer than the input. Consequently, we must literally add the filtered sections together, not just butt them together.

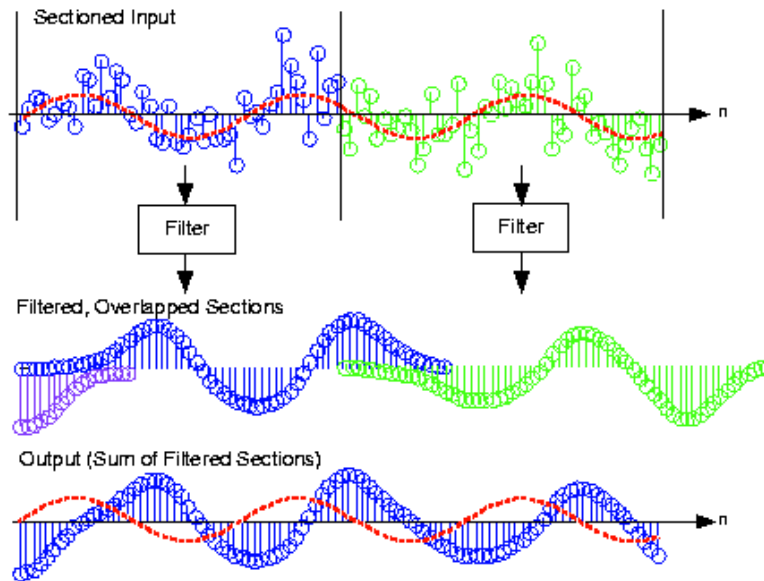


Figure 16.1: The noisy input signal is sectioned into length-48 frames, each of which is filtered using frequency-domain techniques. Each filtered section is added to other outputs that overlap to create the signal equivalent to having filtered the entire input. The sinusoidal component of the signal is shown as the red dashed line.

Computational considerations reveal a substantial advantage for a frequency-domain implementation over a time-domain one. The number of computations for a time-domain implementation essentially remains constant whether we section the input or not. Thus, the number of computations for each output is  $2(q+1)$ . In the frequency-domain approach, computation counting changes because we need only compute the filter's frequency response  $H(k)$  once, which amounts to a fixed overhead. We need only compute two DFTs and multiply them to filter a section. Letting  $N_x$  denote a section's length, the number of computations for a section amounts to:

$$(N_x + q) \log_2(N_x + q) + 6(N_x + q)$$

In addition, we must add the filtered outputs together; the number of terms to add corresponds to the excess duration of the output compared with the input ( $q$ ). The frequency-domain approach thus requires:

$$\log_2(N_x + q) + 6 + \frac{q}{N_x + q}$$

computations per output value. For even modest filter orders, the frequency-domain approach is much faster.

### Exercise 16.1

Show that as the section length increases, the frequency domain approach becomes increasingly more efficient.

#### Solution

Let  $N$  denote the input's total duration. The time-domain implementation requires a total of  $N(2q+1)$  computations, or  $2(q+1)$  computations per input value. In the frequency domain, we split the input into  $N/N_x$  sections, each of which requires

$$\log_2(N_x + q) + 6 + \frac{q}{N_x + q}$$

per input in the section. Because we divide **again** by  $N_x$  to find the number of computations per input value in the entire input, this quantity **decreases** as  $N_x$  increases. For the time-domain implementation, it stays constant.

Note that the choice of section duration is arbitrary. Once the filter is chosen, we should section so that the required FFT length is precisely a power of two: Choose  $N_x$  so that

$$N_x + q = 2^L$$

Implementing the digital filter shown in the [A/D block diagram](#) with a frequency-domain implementation requires some additional signal management not required by time-domain implementations. Conceptually, a real-time, time-domain filter could accept each sample as it becomes available, calculate the difference equation, and produce the output value, all in less than the sampling interval  $T_s$ . Frequency-domain approaches don't operate on a sample-by-sample basis; instead, they operate on sections. They filter in real time by producing  $N_x$  outputs for the same number of inputs faster than  $N_x T_s$ . Because they generally take longer to produce an output section than the sampling interval duration, we must filter one section while accepting into memory the **next** section to be filtered. In programming, the operation of building up sections while computing on previous ones is known as **buffering**. Buffering can also be used in time-domain filters as well but isn't required.

### Example 16.1:

We want to lowpass filter a signal that contains a sinusoid and a significant amount of noise. The example shown in [Figure](#) shows a portion of the noisy signal's waveform. If it weren't for the overlaid sinusoid, discerning the sine wave in the signal is virtually impossible. One of the primary applications of linear filters is **noise removal**: preserve the signal by matching filter's passband with the signal's spectrum and greatly reduce all other frequency components that may be present in the noisy signal.

A smart Rice engineer has selected a FIR filter having a unit-sample response corresponding a period-17 sinusoid:

$$h(n) = \frac{1}{17} \left( 1 - \cos \left( \frac{2\pi n}{17} \right) \right), n = \{0, \dots, 16\} \text{ which makes } q = 16$$

Its frequency response (determined by computing the discrete Fourier transform) is shown in Figure 16.2 below. To apply, we can select the length of each section so that the frequency-domain filtering approach is maximally efficient: Choose the section length  $N_x$  so that  $N_x + q$  is a power of two. To use a length-64 FFT, each section must be 48 samples long. Filtering with the difference equation would require 33 computations per output while the frequency domain requires a little over 16; this frequency-domain implementation is over twice as fast! Figure 16.1 shows how frequency-domain filtering works.

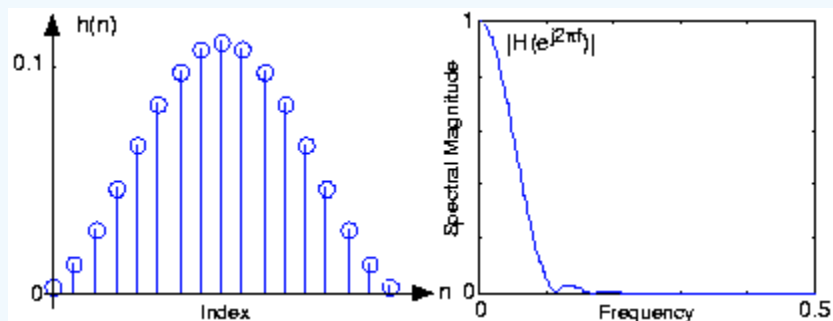


Figure 16.2: The figure shows the unit-sample response of a length-17 Hanning filter on the left and the frequency response on the right. This filter functions as a lowpass filter having a cutoff frequency of about 0.1.

We note that the noise has been dramatically reduced, with a sinusoid now clearly visible in the filtered output. Some residual noise remains because noise components within the filter's passband appear in the output as well as the signal.

### Exercise 16.2

Note that when compared to the input signal's sinusoidal component, the output's sinusoidal component seems to be delayed. What is the source of this delay? Can it be removed?

#### Solution

The delay is **not** computational delay here--the plot shows the first output value is aligned with the filter's first input--although in real systems this is an important consideration. Rather, the delay is due to the filter's phase shift: A phase-shifted sinusoid is equivalent to a time-delayed one:

$$\cos(2\pi fn - \varphi) = \cos\left(2\pi f\left(n - \frac{\varphi}{2\pi f}\right)\right)$$

All filters have phase shifts. This delay could be removed if the filter introduced no phase shift. Such filters do not exist in analog form, but digital ones can be programmed, but not in real time. Doing so would require the output to emerge before the input arrives!

---

This page titled [16: Efficiency of Frequency - Domain Filtering](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.15: Efficiency of Frequency - Domain Filtering](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 17: Discrete -Time Filtering of Analog Signals

## Learning Objectives

- A brief introduction on how to filter digital signals.

Because of the [Sampling Theorem](#), we can process, in particular filter, analog signals "with a computer" by constructing the system shown in Figure 17.1. To use this system, we are assuming that the input signal has a lowpass spectrum and can be bandlimited without affecting important signal aspects. Bandpass signals can also be filtered digitally, but require a more complicated system. Highpass signals cannot be filtered digitally. Note that the input and output filters must be analog filters; trying to operate without them can lead to potentially very inaccurate digitization.

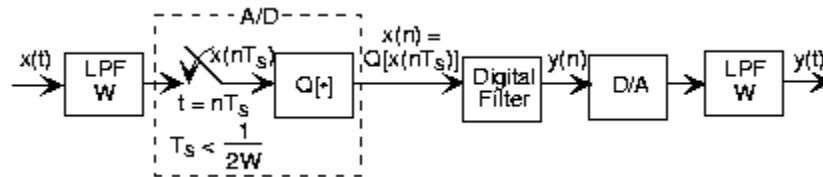


Figure 17.1: To process an analog signal digitally, the signal  $x(t)$  must be filtered with an anti-aliasing filter (to ensure a bandlimited signal) before A/D conversion. This lowpass filter (LPF) has a cutoff frequency of  $W$  Hz, which determines allowable sampling intervals  $T_s$ . The greater the number of bits in the amplitude quantization portion  $Q[\cdot]$  of the A/D converter, the greater the accuracy of the entire system. The resulting digital signal  $x(n)$  can now be filtered in the time-domain with a difference equation or in the frequency domain with Fourier transforms. The resulting output  $y(n)$  then drives a D/A converter and a second anti-aliasing filter (having the same bandwidth as the first one).

Another implicit assumption is that the digital filter can operate in **real time**: The computer and the filtering algorithm must be sufficiently fast so that outputs are computed faster than input values arrive. The sampling interval, which is determined by the analog signal's bandwidth, thus determines how long our program has to compute **each** output  $y(n)$ . The computational complexity for calculating each output with a [difference equation](#) is  $O(p+q)$ . Frequency domain implementation of the filter is also possible. The idea begins by computing the Fourier transform of a length- $N$  portion of the input  $x(n)$ , multiplying it by the filter's transfer function, and computing the inverse transform of the result. This approach seems overly complex and potentially inefficient. Detailing the complexity, however, we have  $O(N \log N)$  for the two transforms (computed using the FFT algorithm) and  $O(N)$  for the multiplication by the transfer function, which makes the total complexity  $O(N \log N)$  for  $N$  input values. A frequency domain implementation thus requires  $O(\log N)$  computational complexity for each output value. The complexities of time-domain and frequency-domain implementations depend on different aspects of the filtering: The time-domain implementation depends on the combined orders of the filter while the frequency-domain implementation depends on the logarithm of the Fourier transform's length.

It could well be that in some problems the time-domain version is more efficient (more easily satisfies the real time requirement), while in others the frequency domain approach is faster. In the latter situations, it is the FFT algorithm for computing the Fourier transforms that enables the superiority of frequency-domain implementations. Because complexity considerations only express how algorithm running-time increases with system parameter choices, we need to detail both implementations to determine which will be more suitable for any given filtering problem. Filtering with a difference equation is straightforward, and the number of computations that must be made for each output value is  $2(p+q)$ .

## Exercise 17.1

Derive this value for the number of computations for the general [difference equation](#).

### Solution

We have  $p+q+1$  multiplications and  $p+q-1$  additions. Thus, the total number of arithmetic operations equals  $2(p+q)$ .

## Contributor

- ContribEEOpenStax

---

This page titled [17: Discrete -Time Filtering of Analog Signals](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.16: Discrete -Time Filtering of Analog Signals](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# 18: Digital Signal Processing Problems

## Learning Objectives

- Practice problems.

## Sampling and Filtering=?

The signal  $s(t)$  is bandlimited to 4 kHz. We want to sample it, but it has been subjected to various signal processing manipulations.

1. What sampling frequency (if any works) can be used to sample the result of passing  $s(t)$  through an RC highpass filter with  $R = 10k\Omega$  and  $C = 8nF$ ?
2. What sampling frequency (if any works) can be used to sample the **derivative** of  $s(t)$ ?
3. The signal  $s(t)$  has been modulated by an 8 kHz sinusoid having an unknown phase: the resulting signal is

$$s(t) \sin(2\pi f_0 t + \varphi)$$

with  $f_0 = 8\text{kHz}$  and  $\varphi=?$  Can the modulated signal be sampled so that the **original** signal can be recovered from the modulated signal regardless of the phase value  $\varphi$ ? If so, show how and find the smallest sampling rate that can be used; if not, show why not.

## Non-Standard Sampling

Using the properties of the Fourier series can ease finding a signal's spectrum.

1. Suppose a signal  $s(t)$  is periodic with period  $T$ . If  $c_k$  represents the signal's Fourier series coefficients, what are the Fourier series coefficients of

$$s\left(t - \frac{T}{2}\right)$$

2. Find the Fourier series of the signal  $p(t)$  shown in Figure 18.1.
3. Suppose this signal is used to sample a signal bandlimited to  $1/T$  Hz. Find an expression for and sketch the spectrum of the sampled signal.
4. Does aliasing occur? If so, can a change in sampling rate prevent aliasing; if not, show how the signal can be recovered from these samples.

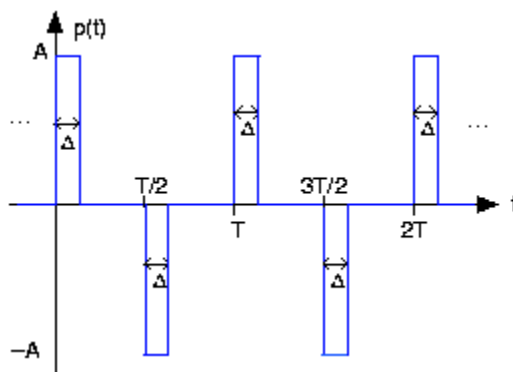


Figure 18.1

## A Different Sampling Scheme

A signal processing engineer from Texas A&M claims to have developed an improved sampling scheme. He multiplies the bandlimited signal by the depicted periodic pulse signal to perform sampling.

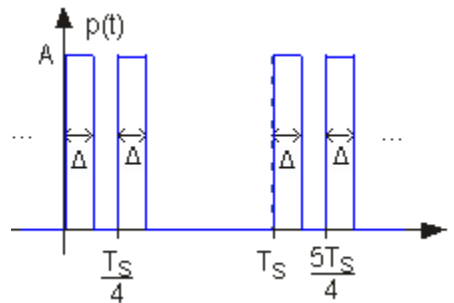


Figure 18.2

1. Find the Fourier spectrum of this signal.
2. Will this scheme work? If so, how should  $T_S$  be related to the signal's bandwidth? If not, why not?

## Bandpass Sampling

The signal  $s(t)$  has the indicated spectrum.

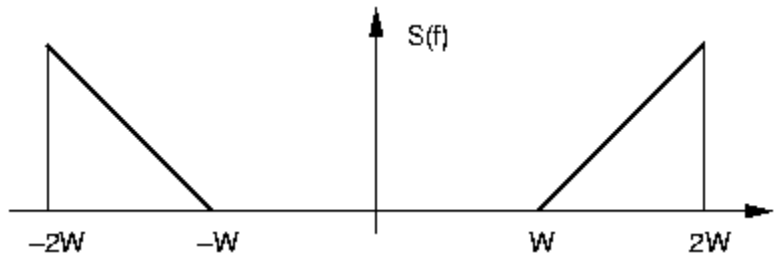


Figure 18.3

1. What is the minimum sampling rate for this signal suggested by the Sampling Theorem?
2. Because of the particular structure of this spectrum, one wonders whether a lower sampling rate could be used. Show that this is indeed the case, and find the system that reconstructs  $s(t)$  from its samples.

## Sampling Signals

If a signal is bandlimited to  $W$  Hz, we can sample it at any rate

$$\frac{1}{T_S} > 2W$$

and recover the waveform exactly. This statement of the Sampling Theorem can be taken to mean that all information about the original signal can be extracted from the samples. While true in principle, you do have to be careful how you do so. In addition to the rms value of a signal, an important aspect of a signal is its peak value, which equals

$$\max \{|s(t)|\}$$

1. Let  $s(t)$  be a sinusoid having frequency  $W$  Hz. If we sample it at precisely the Nyquist rate, how accurately do the samples convey the sinusoid's amplitude? In other words, find the worst case example.
2. How fast would you need to sample for the amplitude estimate to be within 5% of the true value?
3. Another issue in sampling is the inherent amplitude quantization produced by A/D converters. Assume the maximum voltage allowed by the converter is  $V_{\max}$  volts and that it quantizes amplitudes to  $b$  bits. We can express the quantized sample

$$Q(s(nT_S)) \text{ as } s(nT_S) + \varepsilon(t)$$

where  $\varepsilon(t)$  represents the quantization error at the  $n^{\text{th}}$  sample. Assuming the converter rounds, how large is maximum quantization error?

4. We can describe the quantization error as noise, with a power proportional to the square of the maximum error. What is the signal-to-noise ratio of the quantization error for a full-range sinusoid? Express your result in decibels.

## Hardware Error

An A/D converter has a curious hardware problem: Every other sampling pulse is half its normal amplitude.

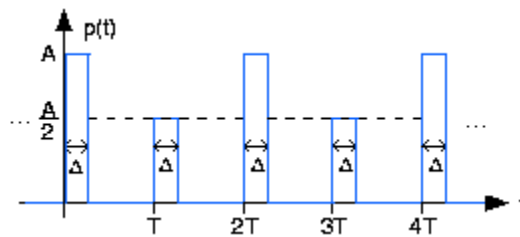


Figure 18.4

1. Find the Fourier series for this signal.
2. Can this signal be used to sample a bandlimited signal having highest frequency

$$W = \frac{1}{2T}$$

## Simple D/A Converter

Commercial digital-to-analog converters don't work this way, but a simple circuit illustrates how they work. Let's assume we have a  $B$ -bit converter. Thus, we want to convert numbers having a  $B$ -bit representation into a voltage proportional to that number. The first step taken by our simple converter is to represent the number by a sequence of  $B$  pulses occurring at multiples of a time interval  $T$ . The presence of a pulse indicates a "1" in the corresponding bit position, and pulse absence means a "0" occurred. For a 4-bit converter, the number 13 has the binary representation 1101

$$13_{10} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

and would be represented by the depicted pulse sequence. Note that the pulse sequence is "backwards" from the binary representation. We'll see why that is.

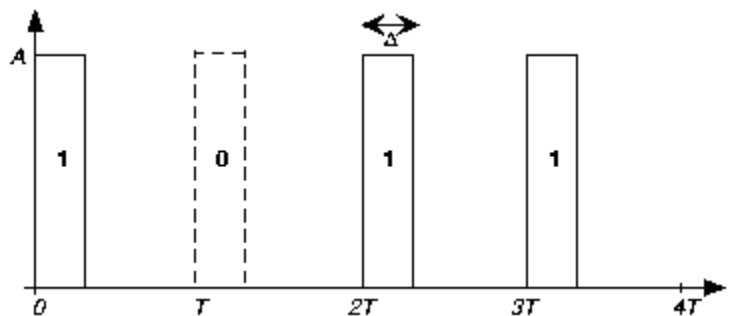


Figure 18.5

This signal serves as the input to a first-order RC lowpass filter. We want to design the filter and the parameters  $\Delta$  and  $T$  so that the output voltage at time  $4T$  (for a 4-bit converter) is proportional to the number. This combination of pulse creation and filtering constitutes our simple D/A converter. The requirements are

- The voltage at time  $t = 4T$  should diminish by a factor of 2 the further the pulse occurs from this time. In other words, the voltage due to a pulse at  $3T$  should be twice that of a pulse produced at  $2T$ , which in turn is twice that of a pulse at  $T$ , etc.
- The 4-bit D/A converter must support a 10 kHz sampling rate.

1. Show the circuit that works. How do the converter's parameters change with sampling rate and number of bits in the converter?

## Discrete-Time Fourier Transforms

Find the Fourier transforms of the following sequences, where  $\mathbf{s(n)}$  is some sequence having Fourier transform

$$S(e^{j2\pi f})$$

1.  $(-1)^n s(n)$
2.  $s(n) \cos(2\pi f_0 n)$
3.  $x(n) = \begin{cases} s(\frac{n}{2}) & \text{if } n \text{ even} \\ 0 & \text{if } n \text{ odd} \end{cases}$
4.  $ns(n)$

## Spectra of Finite-Duration Signals

Find the indicated spectra for the following signals.

1. The discrete-time Fourier transform of

$$s(n) = \begin{cases} \cos^2(\frac{\pi}{4}n) & \text{if } n = \{-1, 0, 1\} \\ 0 & \text{if otherwise} \end{cases}$$

2. The discrete-time Fourier transform of

$$s(n) = \begin{cases} n & \text{if } n = \{-2, -1, 0, 1, 2\} \\ 0 & \text{if otherwise} \end{cases}$$

3. The discrete-time Fourier transform of

$$s(n) = \begin{cases} \sin(\frac{\pi}{4}n) & \text{if } n = \{0, \dots, 7\} \\ 0 & \text{if otherwise} \end{cases}$$

4. The length-8 DFT of the previous signal.

## Just Whistlin'

Sammy loves to whistle and decides to record and analyze his whistling in lab. He is a very good whistler; his whistle is a pure sinusoid that can be described by

$$s_a(t) = \sin(4000t)$$

To analyze the spectrum, he samples his recorded whistle with a sampling interval of

$$T_s = 2.5 \times 10^{-4}$$

to obtain

$$s(n) = s_a(nT_s)$$

Sammy (wisely) decides to analyze a few samples at a time, so he grabs 30 consecutive, but arbitrarily chosen, samples. He calls this sequence  $\mathbf{x(n)}$  and realizes he can write it as

$$x(n) = \sin(4000nT_s + \theta), \quad n = \{0, \dots, 29\}$$

1. Did Sammy under- or over-sample his whistle?
2. What is the discrete-time Fourier transform of  $\mathbf{x(n)}$  and how does it depend on  $\theta$ ?
3. How does the 32-point DFT of  $\mathbf{x(n)}$  depend on  $\theta$ ?

## Discrete-Time Filtering

We can find the input-output relation for a discrete-time filter much more easily than for analog filters. The key idea is that a sequence can be written as a weighted linear combination of unit samples.

1. Show that

$$x(n) = \sum_i x(i)\delta(n-i)$$

where  $\delta(n)$  is the unit-sample.

$$\delta(n) = \begin{cases} 1 & \text{if } n = 0 \\ 0 & \text{if otherwise} \end{cases}$$

2. If  $h(n)$  denotes the **unit-sample response**—the output of a discrete-time linear, shift-invariant filter to a unit-sample input—find an expression for the output.
3. In particular, assume our filter is FIR, with the unit-sample response having duration  $q+1$ . If the input has duration  $N$ , what is the duration of the filter's output to this signal?
4. Let the filter be a boxcar averager:

$$h(n) = \frac{1}{q+1} \text{ for } n = \{0, \dots, q\}$$

and zero otherwise. Let the input be a pulse of unit height and duration  $N$ . Find the filter's output when

$$N = \frac{q+1}{2}$$

$q$  an odd integer.

## A Digital Filter

A digital filter has the depicted unit-sample response.

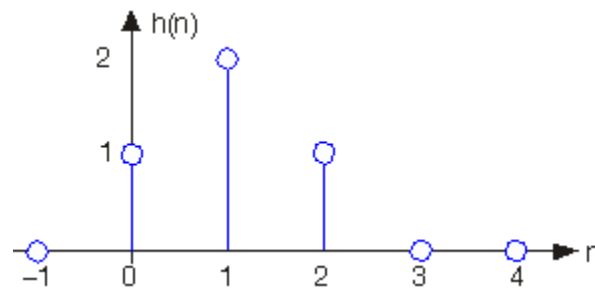


Figure 18.6

1. What is the difference equation that defines this filter's input-output relationship?
2. What is this filter's transfer function?
3. What is the filter's output when the input is

$$\sin\left(\frac{\pi n}{4}\right)$$

## A Special Discrete-Time Filter

Consider a FIR filter governed by the difference equation

$$y(n) = \frac{1}{3}x(n+2) + \frac{2}{3}x(n+1) + x(n) + \frac{2}{3}x(n-1) + \frac{1}{3}x(n-2)$$

1. Find this filter's unit-sample response.
2. Find this filter's transfer function. Characterize this transfer function (i.e., what classic filter category does it fall into).
3. Suppose we take a sequence and stretch it out by a factor of three.

$$x(n) = \begin{cases} s(\frac{n}{3}) & \text{if } \forall m, m = \{\dots, -1, 0, 1, \dots\} : (n - 3m) \\ 0 & \text{if otherwise} \end{cases}$$

- Sketch the sequence **x(n)** for some example **s(n)**.
- What is the filter's output to this input?
- In particular, what is the output at the indices where the input **x(n)** is intentionally zero?
- Now how would you characterize this system?

## Simulating the Real World

Much of physics is governed by differential equations, and we want to use signal processing methods to simulate physical problems. The idea is to replace the derivative with a discrete-time approximation and solve the resulting differential equation. For example, suppose we have the differential equation

$$\frac{dy(t)}{dt} + ay(t) = x(t)$$

and we approximate the derivative by

$$\frac{dy(t)}{dt} \Big|_{t=nT} \simeq \frac{y(nT) - y((n-1)T)}{T}$$

where **T** essentially amounts to a sampling interval.

- What is the difference equation that must be solved to approximate the differential equation?
- When **x(t) = u(t)**, the unit step, what will be the simulated output?
- Assuming **x(t)** is a sinusoid, how should the sampling interval **T** be chosen so that the approximation works well?

## Derivatives

The derivative of a sequence makes little sense, but still, we can approximate it. The digital filter described by the difference equation

$$y(n) = x(n) - x(n-1]$$

resembles the derivative formula. We want to explore how well it works.

- What is this filter's transfer function?
- What is the filter's output to the depicted triangle output?

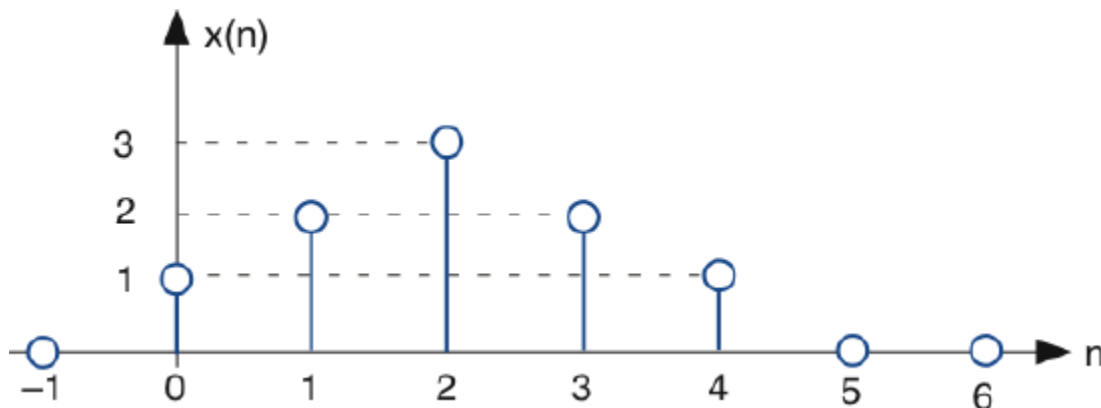


Figure 18.7

- Suppose the signal **x(n)** is a sampled analog signal:

$$x(n) = x(nT_S)$$

Under what conditions will the filter act like a differentiator? In other words, when will **y(n)** be proportional to

$$\left. \frac{dx(t)}{dt} \right|_{t=nT_s}$$

## The DFT

Let's explore the DFT and its properties.

1. What is the length-**K** DFT of length-**N** boxcar sequence, where **N < K**?
2. Consider the special case where **K = 4**. Find the inverse DFT of the product of the DFTs of two length-3 boxcars.
3. If we could use DFTs to perform linear filtering, it should be true that the product of the input's DFT and the unit-sample response's DFT equals the output's DFT. So that you can use what you just calculated, let the input be a boxcar signal and the unit-sample response also be a boxcar. The result of part (b) would then be the filter's output **if** we could implement the filter with length-4 DFTs. Does the actual output of the boxcar-filter equal the result found in the previous part?
4. What would you need to change so that the product of the DFTs of the input and unit-sample response in this case equaled the DFT of the filtered output?

## DSP Tricks

Sammy is faced with computing **lots** of discrete Fourier transforms. He will, of course, use the FFT algorithm, but he is behind schedule and needs to get his results as quickly as possible. He gets the idea of computing **two** transforms at one time by computing the transform of

$$s(n) = s_1(n) + is_2(n)$$

where **s<sub>1</sub>(n)** and **s<sub>2</sub>(n)** are two real-valued signals of which he needs to compute the spectra. The issue is whether he can retrieve the individual DFTs from the result or not.

1. What will be the DFT **S(k)** of this complex-valued signal in terms of **S<sub>1</sub>(k)** and **S<sub>2</sub>(k)**, the DFTs of the original signals?
2. Sammy's friend, an Aggie who knows some signal processing, says that retrieving the wanted DFTs is easy: "Just find the real and imaginary parts of **S(k)** ." Show that this approach is too simplistic.
3. While his friend's idea is not correct, it does give him an idea. What approach will work? **Hint:** Use the symmetry properties of the DFT.
4. How does the number of computations change with this approach? Will Sammy's idea ultimately lead to a faster computation of the required DFTs?

## Discrete Cosine Transform (DCT)

The discrete cosine transform of a length-**N** sequence is defined to be

$$S_c(k) = \sum_{n=0}^{N-1} s(n) \cos\left(\frac{2\pi nk}{2N}\right)$$

Note that the number of frequency terms is

$$2N - 1 : k = \{0, \dots, 2N - 1\}$$

1. Find the inverse DCT.
2. Does a Parseval's Theorem hold for the DCT?
3. You choose to transmit information about the signal **s(n)** according to the DCT coefficients. You could only send one, which one would you send?

## A Digital Filter

A digital filter is described by the following difference equation:

$$y(n) = ay(n-1) + ax(n) - x(n-1), a = \frac{1}{\sqrt{2}}$$

1. What is this filter's unit sample response?
2. What is this filter's transfer function?
3. What is this filter's output when the input is

$$\sin\left(\frac{\pi n}{4}\right)$$

## Another Digital Filter

A digital filter is determined by the following difference equation.

$$y(n) = y(n-1) + x(n) - x(n-4)$$

1. Find this filter's unit sample response.
2. What is the filter's transfer function? How would you characterize this filter (lowpass, highpass, special purpose, ...)?
3. Find the filter's output when the input is the sinusoid

$$\sin\left(\frac{\pi n}{2}\right)$$

4. In another case, the input sequence is zero for  $n < 0$ , then becomes nonzero. Sammy measures the output to be

$$y(n) = \delta(n) + \delta(n-1)$$

Can his measurement be correct? In other words, is there an input that can yield this output? If so, find the input  $x(n)$  that gives rise to this output. If not, why not?

## Yet Another Digital Filter

A filter has an input-output relationship given by the difference equation

$$y(n] = \frac{1}{4}x(n) + \frac{1}{2}x(n-1) + \frac{1}{4}x(n-2)$$

1. What is the filter's transfer function? How would you characterize it?
2. What is the filter's output when the input equals

$$\cos\left(\frac{\pi n}{2}\right)$$

3. What is the filter's output when the input is the depicted discrete-time square wave below?

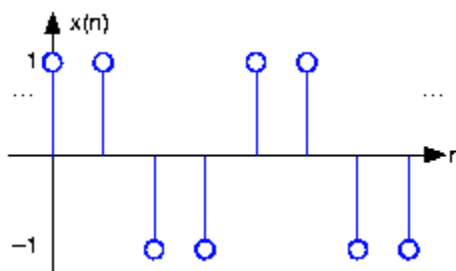


Figure 18.8

## A Digital Filter in the Frequency Domain

We have a filter with the transfer function

$$H(e^{i2\pi f}) = e^{-(i2\pi f)} \cos(2\pi f)$$

operating on the input signal

$$x(n) = \delta(n) - \delta(n-2)$$

that yields the output  $y(n)$ .

1. What is the filter's unit-sample response?
2. What is the discrete-Fourier transform of the output?
3. What is the time-domain expression for the output?

## Digital Filters

A discrete-time system is governed by the difference equation

$$y(n) = y(n-1) + \frac{x(n) + x(n-1)}{2}$$

1. Find the transfer function for this system.
2. What is this system's output when the input is

$$\sin\left(\frac{\pi n}{2}\right)$$

3. If the output is observed to be

$$y(n) = \delta(n) + \delta(n-1)$$

then what is the input?

## Digital Filtering

A digital filter has an input-output relationship expressed by the difference equation

$$y(n) = \frac{x(n) + x(n-1) + x(n-2) + x(n-3)}{4}$$

1. Plot the magnitude and phase of this filter's transfer function.
2. What is this filter's output when

$$x(n) = \cos\left(\frac{\pi n}{2}\right) + 2\sin\left(\frac{2\pi n}{3}\right)$$

## Detective Work

1. The signal

$$x(n) = \delta(n) - \delta(n-1)$$

- i. Find the length-8 DFT (discrete Fourier transform) of this signal.
- ii. You are told that when  $\mathbf{x(n)}$  served as the input to a linear FIR (finite impulse response) filter, the output was

$$y(n) = \delta(n) - \delta(n-1) + 2\delta(n-2)$$

Is this statement true? If so, indicate why and find the system's unit sample response; if not, show why not.

2. A discrete-time, shift invariant, linear system produces an output

$$y(n) = \{1, -1, 0, 0, \dots\}$$

when its input  $\mathbf{x(n)}$  equals a unit sample.

- i. Find the difference equation governing the system.
- ii. Find the output when

$$x(n) = \cos(2\pi f_0 n)$$

- iii. How would you describe this system's function?

## Time Reversal has Uses

A discrete-time system has transfer function

$$H(e^{i2\pi f})$$

A signal  $\mathbf{x}(n)$  is passed through this system to yield the signal  $\mathbf{w}(n)$ . The time-reversed signal  $\mathbf{w}(-n)$  is then passed through the system to yield the time-reversed output  $\mathbf{y}(-n)$ . What is the transfer function between  $\mathbf{x}(n)$  and  $\mathbf{y}(n)$ ?

## Removing “Hum”

The slang word “hum” represents power line waveforms that creep into signals because of poor circuit construction. Usually, the 60 Hz signal (and its harmonics) are added to the desired signal. What we seek are filters that can remove hum. In this problem, the signal and the accompanying hum have been sampled; we want to design a **digital** filter for hum removal.

1. Find filter coefficients for the length-3 FIR filter that can remove a sinusoid having **digital** frequency  $\mathbf{f_0}$  from its input.
2. Assuming the sampling rate is  $\mathbf{f_s}$  to what analog frequency does  $\mathbf{f_0}$  correspond?
3. A more general approach is to design a filter having a frequency response **magnitude** proportional to the absolute value of a cosine:

$$|H(e^{i2\pi f})| \propto |\cos(\pi fN)|$$

In this way, not only can the fundamental but also its first few harmonics be removed. Select the parameter  $\mathbf{N}$  and the sampling rate so that the frequencies at which the cosine equals zero correspond to 60 Hz and its odd harmonics through the fifth.

4. Find the difference equation that defines this filter.

## Digital AM Receiver

Thinking that digital implementations are **always** better, our clever engineer wants to design a digital AM receiver. The receiver would bandpass the received signal, pass the result through an A/D converter, perform all the demodulation with digital signal processing systems, and end with a D/A converter to produce the analog message signal. Assume in this problem that the carrier frequency is always a large **even** multiple of the message signal's bandwidth  $\mathbf{W}$ .

1. What is the smallest sampling rate that would be needed?
2. Show the block diagram of the least complex digital AM receiver.
3. Assuming the channel adds white noise and that a  $\mathbf{b}$ -bit A/D converter is used, what is the output's signal-to-noise ratio?

## DFTs

A problem on Samantha's homework asks for the **8-point** DFT of the discrete-time signal

$$\delta(n-1) + \delta(n-7)$$

1. What answer should Samantha obtain?
2. As a check, her group partner Sammy says that he computed the inverse DFT of her answer and got

$$\delta(n+1) + \delta(n-1)$$

Does Sammy's result mean that Samantha's answer is wrong?

3. The homework problem says to lowpass-filter the sequence by multiplying its DFT by

$$H(k) = \begin{cases} 1 & \text{if } k = \{0, 1, 7\} \\ 0 & \text{if otherwise} \end{cases}$$

and then computing the inverse DFT. Will this filtering algorithm work? If so, find the filtered output; if not, why not?

## Stock Market Data Processing

Because a trading week lasts five days, stock markets frequently compute running averages each day over the previous five trading days to smooth price fluctuations. The technical stock analyst at the Buy-Lo--Sell-Hi brokerage firm has heard that FFT filtering techniques work better than any others (in terms of producing more accurate averages).

1. What is the difference equation governing the five-day averager for daily stock prices?

2. Design an efficient FFT-based filtering algorithm for the broker. How much data should be processed at once to produce an efficient algorithm? What length transform should be used?
3. Is the analyst's information correct that FFT techniques produce more accurate averages than any others? Why or why not?

## Echoes

Echoes not only occur in canyons, but also in auditoriums and telephone circuits. In one situation where the echoed signal has been sampled, the input signal  $x(n)$  emerges as

$$x(n) + a_1x(n - n_1) + a_2x(n - n_2)$$

1. Find the difference equation of the system that models the production of echoes.
2. To simulate this echo system, ELEC 241 students are asked to write the most efficient (quickest) program that has the same input-output relationship. Suppose the duration of  $x(n)$  is 1,000 and that

$$a_1 = \frac{1}{2}, n_1 = 10, a_2 = \frac{1}{5}, n_2 = 25$$

Half the class votes to just program the difference equation while the other half votes to program a frequency domain approach that exploits the speed of the FFT. Because of the undecided vote, you must break the tie. Which approach is more efficient and why?

3. Find the transfer function and difference equation of the system that suppresses the echoes. In other words, with the echoed signal as the input, what system's output is the signal  $x(n)$ ?

## Digital Filtering of Analog Signals

RU Electronics wants to develop a filter that would be used in analog applications, but that is implemented digitally. The filter is to operate on signals that have a 10 kHz bandwidth, and will serve as a lowpass filter.

1. What is the block diagram for your filter implementation? Explicitly denote which components are analog, which are digital (a computer performs the task), and which interface between analog and digital worlds.
2. What sampling rate must be used and how many bits must be used in the A/D converter for the acquired signal's signal-to-noise ratio to be at least 60 dB? For this calculation, assume the signal is a sinusoid.
3. If the filter is a length-128 FIR filter (the duration of the filter's unit-sample response equals 128), should it be implemented in the time or frequency domain?
4. Assuming

$$H(e^{i2\pi f})$$

is the transfer function of the digital filter, what is the transfer function of your system?

## Signal Compression

Because of the slowness of the Internet, lossy signal compression becomes important if you want signals to be received quickly. An enterprising 241 student has proposed a scheme based on frequency-domain processing. First of all, he would section the signal into length- $N$  blocks, and compute its  $N$ -point DFT. He then would discard (zero the spectrum) at **half** of the frequencies, quantize them to **b**-bits, and send these over the network. The receiver would assemble the transmitted spectrum and compute the inverse DFT, thus reconstituting an  $N$ -point block.

1. At what frequencies should the spectrum be zeroed to minimize the error in this lossy compression scheme?
2. The nominal way to represent a signal digitally is to use simple **b**-bit quantization of the time-domain waveform. How long should a section be in the proposed scheme so that the required number of bits/sample is smaller than that nominally required?
3. Assuming that effective compression can be achieved, would the proposed scheme yield satisfactory results?

---

This page titled [18: Digital Signal Processing Problems](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

- [5.17: Digital Signal Processing Problems](#) by [Don H. Johnson](#) is licensed [CC BY 1.0](#). Original source: <https://cnx.org/contents/d442r0wh@9.72:g9deOnx5@19>.

# INDEX

## A

analog signals

*1: Introduction to Digital Signal Processing*

## B

basis

*6.2: Discrete Time Fourier Series (DTFS)*

bilateral z transform pair

*5.1: Z-Transform*

## C

causal

*5.5: Poles and Zeros in the Z-Plane*

characteristic polynomial

*5.8: Difference Equations*

circular convolution

*6.5: Discrete Time Circular Convolution and the DTFS*

complex conjugate

*6.2: Discrete Time Fourier Series (DTFS)*

complex exponential

*6.2: Discrete Time Fourier Series (DTFS)*

computer technology

*1: Introduction to Digital Signal Processing*

control theory

*5.5: Poles and Zeros in the Z-Plane*

## D

data analysis

*1: Introduction to Digital Signal Processing*

difference equation

*5.8: Difference Equations*

digital signal processing

*1: Introduction to Digital Signal Processing*

direct method

*5.8: Difference Equations*

discrete signals

*1: Introduction to Digital Signal Processing*

discrete time fourier series

*6.2: Discrete Time Fourier Series (DTFS)*

## F

Fast Fourier Transform (FFT)

*10: Fast Fourier Transform (FFT)*

Fourier series analysis

*6.3: Common Discrete Fourier Series*

Fourier Transforms

*1: Introduction to Digital Signal Processing*

## H

Hanning window

*11: Spectrograms*

harmonic sinusoid

*6.2: Discrete Time Fourier Series (DTFS)*

homogeneous solution

*5.8: Difference Equations*

## I

indirect method

*5.8: Difference Equations*

## L

linear convolution

*6.5: Discrete Time Circular Convolution and the DTFS*

linear filtering

*1: Introduction to Digital Signal Processing*

## P

particular solution

*5.8: Difference Equations*

poles

*5.5: Poles and Zeros in the Z-Plane*

power series

*5.6: Region of Convergence for the Z-Transform*

Power spectral density

*1: Introduction to Digital Signal Processing*

## Q

quantization

*3: Amplitude Quantization*

quantization interval

*3: Amplitude Quantization*

## R

rational function

*5.7: Rational Functions and the Z-Transform*

region of convergence

*5.6: Region of Convergence for the Z-Transform*

ROC

*5.6: Region of Convergence for the Z-Transform*

## S

signal processing

*1: Introduction to Digital Signal Processing*

spectrograms

*11: Spectrograms*

stable

*5.5: Poles and Zeros in the Z-Plane*

## T

telecommunications

*1: Introduction to Digital Signal Processing*

transfer function

*5.8: Difference Equations*

## U

unilateral laplace transform

*5.1: Z-Transform*

## Z

zeros

*5.5: Poles and Zeros in the Z-Plane*

---

This page titled [Index](#) is shared under a [CC BY-SA 1.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#) via [source content](#) that was edited to the style and standards of the LibreTexts platform.

# INDEX

## A

analog signals

*1: Introduction to Digital Signal Processing*

## B

basis

*6.2: Discrete Time Fourier Series (DTFS)*

bilateral z transform pair

*5.1: Z-Transform*

## C

causal

*5.5: Poles and Zeros in the Z-Plane*

characteristic polynomial

*5.8: Difference Equations*

circular convolution

*6.5: Discrete Time Circular Convolution and the DTFS*

complex conjugate

*6.2: Discrete Time Fourier Series (DTFS)*

complex exponential

*6.2: Discrete Time Fourier Series (DTFS)*

computer technology

*1: Introduction to Digital Signal Processing*

control theory

*5.5: Poles and Zeros in the Z-Plane*

## D

data analysis

*1: Introduction to Digital Signal Processing*

difference equation

*5.8: Difference Equations*

digital signal processing

*1: Introduction to Digital Signal Processing*

direct method

*5.8: Difference Equations*

discrete signals

*1: Introduction to Digital Signal Processing*

discrete time fourier series

*6.2: Discrete Time Fourier Series (DTFS)*

## F

Fast Fourier Transform (FFT)

*10: Fast Fourier Transform (FFT)*

Fourier series analysis

*6.3: Common Discrete Fourier Series*

Fourier Transforms

*1: Introduction to Digital Signal Processing*

## H

Hanning window

*11: Spectrograms*

harmonic sinusoid

*6.2: Discrete Time Fourier Series (DTFS)*

homogeneous solution

*5.8: Difference Equations*

## I

indirect method

*5.8: Difference Equations*

## L

linear convolution

*6.5: Discrete Time Circular Convolution and the DTFS*

linear filtering

*1: Introduction to Digital Signal Processing*

## P

particular solution

*5.8: Difference Equations*

poles

*5.5: Poles and Zeros in the Z-Plane*

power series

*5.6: Region of Convergence for the Z-Transform*

Power spectral density

*1: Introduction to Digital Signal Processing*

## Q

quantization

*3: Amplitude Quantization*

quantization interval

*3: Amplitude Quantization*

## R

rational function

*5.7: Rational Functions and the Z-Transform*

region of convergence

*5.6: Region of Convergence for the Z-Transform*

ROC

*5.6: Region of Convergence for the Z-Transform*

## S

signal processing

*1: Introduction to Digital Signal Processing*

spectrograms

*11: Spectrograms*

stable

*5.5: Poles and Zeros in the Z-Plane*

## T

telecommunications

*1: Introduction to Digital Signal Processing*

transfer function

*5.8: Difference Equations*

## U

unilateral laplace transform

*5.1: Z-Transform*

## Z

zeros

*5.5: Poles and Zeros in the Z-Plane*



## GLOSSARY

### S

---

**Sample Word 1** | Sample Definition 1

---

This page titled [Glossary](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by [Carl Greco](#).

## DETAILED LICENSING

### Overview

**Title:** Discrete-Time Signal Processing

**Webpages:** 44

**All licenses found:**

- [CC BY-SA 4.0](#): 59.1% (26 pages)
- [CC BY-SA 1.0](#): 38.6% (17 pages)
- [Undeclared](#): 2.3% (1 page)

### By Page

- Discrete-Time Signal Processing - [CC BY-SA 4.0](#)
  - Front Matter - [CC BY-SA 4.0](#)
    - TitlePage - [CC BY-SA 4.0](#)
    - InfoPage - [CC BY-SA 4.0](#)
    - Table of Contents - [Undeclared](#)
    - Licensing - [CC BY-SA 4.0](#)
  - 1: Introduction to Digital Signal Processing - [CC BY-SA 1.0](#)
  - 2: The Sampling Theorem - [CC BY-SA 1.0](#)
  - 3: Amplitude Quantization - [CC BY-SA 1.0](#)
  - 4: Discrete -Time Signals and Systems - [CC BY-SA 1.0](#)
  - 5: Z-Transform and Discrete Time System Design - [CC BY-SA 4.0](#)
    - 5.1: Z-Transform - [CC BY-SA 4.0](#)
    - 5.2: Common Z-Transforms - [CC BY-SA 4.0](#)
    - 5.3: Properties of the Z-Transform - [CC BY-SA 4.0](#)
    - 5.4: Inverse Z-Transform - [CC BY-SA 4.0](#)
    - 5.5: Poles and Zeros in the Z-Plane - [CC BY-SA 4.0](#)
    - 5.6: Region of Convergence for the Z-Transform - [CC BY-SA 4.0](#)
    - 5.7: Rational Functions and the Z-Transform - [CC BY-SA 4.0](#)
    - 5.8: Difference Equations - [CC BY-SA 4.0](#)
    - 5.9: Discrete Time Filter Design - [CC BY-SA 4.0](#)
  - 6: Discrete Time Fourier Series (DTFS) - [CC BY-SA 4.0](#)
    - 6.1: Discrete Time Periodic Signals - [CC BY-SA 4.0](#)
    - 6.2: Discrete Time Fourier Series (DTFS) - [CC BY-SA 4.0](#)
    - 6.3: Common Discrete Fourier Series - [CC BY-SA 4.0](#)
    - 6.4: Properties of the DTFS - [CC BY-SA 4.0](#)
    - 6.5: Discrete Time Circular Convolution and the DTFS - [CC BY-SA 4.0](#)
  - 7: Discrete -Time Fourier Transform (DTFT) - [CC BY-SA 1.0](#)
  - 8: Discrete Fourier Transforms (DFT) - [CC BY-SA 1.0](#)
  - 9: DFT - Computational Complexity - [CC BY-SA 1.0](#)
  - 10: Fast Fourier Transform (FFT) - [CC BY-SA 1.0](#)
  - 11: Spectrograms - [CC BY-SA 1.0](#)
  - 12: Discrete-Time Systems - [CC BY-SA 1.0](#)
  - 13: Discrete-Time Systems in the Time-Domain - [CC BY-SA 1.0](#)
  - 14: Discrete -Time Systems in the Frequency Domain - [CC BY-SA 1.0](#)
  - 15: Filtering in the Frequency Domain - [CC BY-SA 1.0](#)
  - 16: Efficiency of Frequency - Domain Filtering - [CC BY-SA 1.0](#)
  - 17: Discrete -Time Filtering of Analog Signals - [CC BY-SA 1.0](#)
  - 18: Digital Signal Processing Problems - [CC BY-SA 1.0](#)
  - Back Matter - [CC BY-SA 4.0](#)
    - Index - [CC BY-SA 1.0](#)
    - Index - [CC BY-SA 4.0](#)
    - Glossary - [CC BY-SA 4.0](#)
    - Detailed Licensing - [CC BY-SA 4.0](#)
    - Detailed Licensing - [CC BY-SA 4.0](#)

This page titled [Discrete-Time Signal Processing](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.

## DETAILED LICENSING

### Overview

**Title:** Discrete-Time Signal Processing

**Webpages:** 44

**All licenses found:**

- [CC BY-SA 4.0](#): 59.1% (26 pages)
- [CC BY-SA 1.0](#): 38.6% (17 pages)
- [Undeclared](#): 2.3% (1 page)

### By Page

- Discrete-Time Signal Processing - [CC BY-SA 4.0](#)
  - Front Matter - [CC BY-SA 4.0](#)
    - TitlePage - [CC BY-SA 4.0](#)
    - InfoPage - [CC BY-SA 4.0](#)
    - Table of Contents - [Undeclared](#)
    - Licensing - [CC BY-SA 4.0](#)
  - 1: Introduction to Digital Signal Processing - [CC BY-SA 1.0](#)
  - 2: The Sampling Theorem - [CC BY-SA 1.0](#)
  - 3: Amplitude Quantization - [CC BY-SA 1.0](#)
  - 4: Discrete -Time Signals and Systems - [CC BY-SA 1.0](#)
  - 5: Z-Transform and Discrete Time System Design - [CC BY-SA 4.0](#)
    - 5.1: Z-Transform - [CC BY-SA 4.0](#)
    - 5.2: Common Z-Transforms - [CC BY-SA 4.0](#)
    - 5.3: Properties of the Z-Transform - [CC BY-SA 4.0](#)
    - 5.4: Inverse Z-Transform - [CC BY-SA 4.0](#)
    - 5.5: Poles and Zeros in the Z-Plane - [CC BY-SA 4.0](#)
    - 5.6: Region of Convergence for the Z-Transform - [CC BY-SA 4.0](#)
    - 5.7: Rational Functions and the Z-Transform - [CC BY-SA 4.0](#)
    - 5.8: Difference Equations - [CC BY-SA 4.0](#)
    - 5.9: Discrete Time Filter Design - [CC BY-SA 4.0](#)
  - 6: Discrete Time Fourier Series (DTFS) - [CC BY-SA 4.0](#)
    - 6.1: Discrete Time Periodic Signals - [CC BY-SA 4.0](#)
    - 6.2: Discrete Time Fourier Series (DTFS) - [CC BY-SA 4.0](#)
    - 6.3: Common Discrete Fourier Series - [CC BY-SA 4.0](#)
    - 6.4: Properties of the DTFS - [CC BY-SA 4.0](#)
    - 6.5: Discrete Time Circular Convolution and the DTFS - [CC BY-SA 4.0](#)
  - 7: Discrete -Time Fourier Transform (DTFT) - [CC BY-SA 1.0](#)
  - 8: Discrete Fourier Transforms (DFT) - [CC BY-SA 1.0](#)
  - 9: DFT - Computational Complexity - [CC BY-SA 1.0](#)
  - 10: Fast Fourier Transform (FFT) - [CC BY-SA 1.0](#)
  - 11: Spectrograms - [CC BY-SA 1.0](#)
  - 12: Discrete-Time Systems - [CC BY-SA 1.0](#)
  - 13: Discrete-Time Systems in the Time-Domain - [CC BY-SA 1.0](#)
  - 14: Discrete -Time Systems in the Frequency Domain - [CC BY-SA 1.0](#)
  - 15: Filtering in the Frequency Domain - [CC BY-SA 1.0](#)
  - 16: Efficiency of Frequency - Domain Filtering - [CC BY-SA 1.0](#)
  - 17: Discrete -Time Filtering of Analog Signals - [CC BY-SA 1.0](#)
  - 18: Digital Signal Processing Problems - [CC BY-SA 1.0](#)
  - Back Matter - [CC BY-SA 4.0](#)
    - Index - [CC BY-SA 1.0](#)
    - Index - [CC BY-SA 4.0](#)
    - Glossary - [CC BY-SA 4.0](#)
    - Detailed Licensing - [CC BY-SA 4.0](#)
    - Detailed Licensing - [CC BY-SA 4.0](#)

This page titled [Discrete-Time Signal Processing](#) is shared under a [CC BY-SA 4.0](#) license and was authored, remixed, and/or curated by Carl Greco.