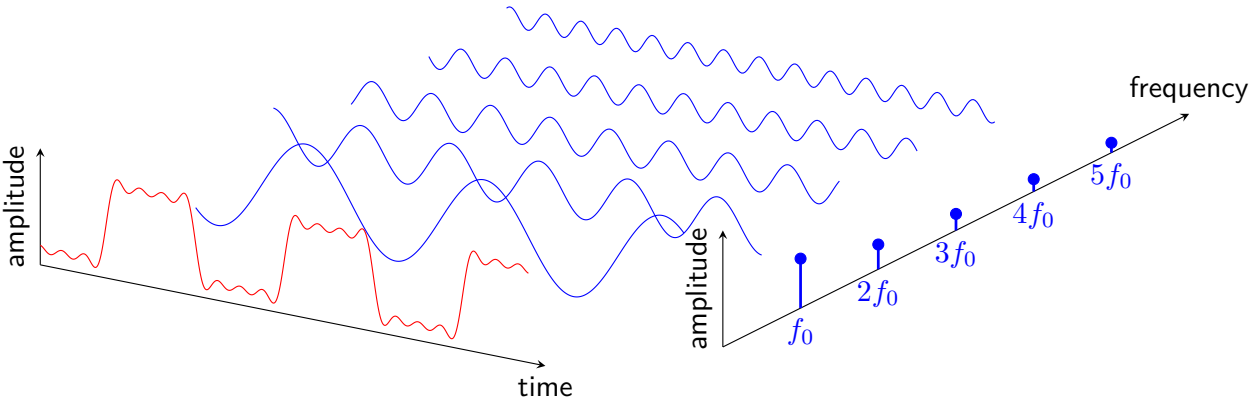
Manchester Metropolitan University

6G5Z3015 Computational Mathematics

Sound Processing Notes

Dr Jon Shiach

2022



Contents

Chapter 1

Introduction to Digital Signals

Communication has moved from analogue to digital and is now ubiquitous in everyday life. Examples of this can be found in audio and video streaming, CDs, DVDs, Blu-ray discs, mp3s and mobile phones. This part of the unit aims to introduce some of the fundamental concepts underpinning the this digital revolution and to apply them to sounds. It is possible to explain some of these concepts using only the English language but it would then be impossible to apply them. For example, you might be able to write an essay on digital speech after a descriptive lecture on it but you wouldn't have a clue how to compress it or remove noise or do anything useful with it at all. Fortunately there is another language which is precise enough to describe fundamental concepts and which contains the structures and formalism necessary to apply these concepts to useful areas: mathematics.

1.1 Signal representation

An information bearing signal is the variation of some quantity with respect to some other quantity. A signal may be represented by a mathematical function relating an input variable (e.g. time) to a unique output variable (e.g. pressure).

A sound signal is the continuous variation of pressure with time, so we can write,

$$s = f(t),$$

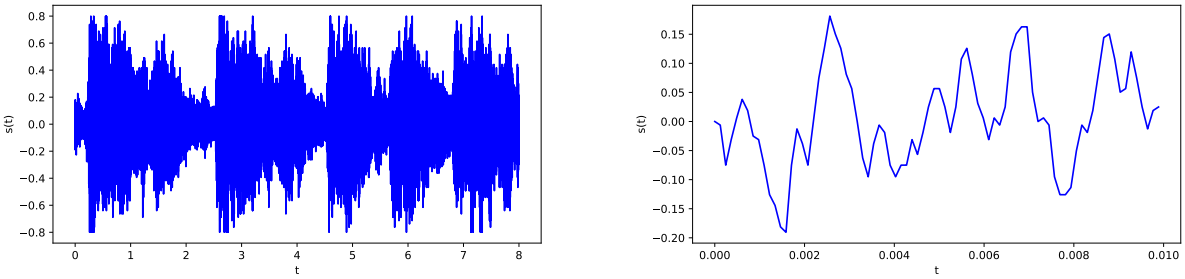
to describe such a signal. Here t is time and s is pressure (or voltage) and f denotes the fact that s is some function of time. s is the dependent variable (because it depends on t) and t is the independent variable. It is common to write $s = s(t)$.

1.1.1 Analogue signals

A sound signal can be represented by the function $s(t)$ which can be viewed as a graph of $s(t)$ against t. For example, a graph of the first 8 seconds of the Handel's 'Hallelujah Chorus' from *Messiah* can be seen in figure 1.1(a) with a zoomed in portion of the first 0.01 seconds can be seen in figure 1.1(b).

The important thing to note is that the signal is defined at any value of t over the range $[0, 8]$. Furthermore, at any particular value of t the corresponding signal value, $s(t)$, comes from a continuum of possible values. This defines an analogue signal.

An analogue signal requires an infinite number of numerical values to describe it.



(a) first 8 seconds

(b) zoom in on the first 0.01 seconds

Figure 1.1: Graph of a segment of the signal for Handel's Hallelujah Chorus.

1.1.2 Digital signals

Modern digital computers can only work with finite amounts of data so if we want to store and process signals in a digital computer they must be approximated by a finite set of numbers, i.e. they must be represented digitally.

In practice, because we will be concerned with digital signals, an analogue sound signal $s = s(t)$ will be replaced by a one-dimensional array (also called a **sequence**) whose elements are signal values in time order, e.g.,

$$s(t) = (3, 0, -2, 1, 0, -3, 2, 3, -2, 4, 5, 2, 1, 0, 2).$$

The main advantage of a digital signal over its corresponding analogue version is that it can be stored and manipulated in a digital computer. This means that a digital signal can be processed in a useful way by implementing a computer algorithm. Examples include algorithms to compress data (e.g. mp3 for audio and mp4 for video), automatically detect and correct errors and automatic feature extraction e.g., speech recognition. A digital signal is inherently robust since allowable signal values are separated which means that transmitted signal quality can high. Processing hardware and software is cheap and flexible. The main disadvantage with a digital signal is that it requires more bandwidth than an equivalent analogue signal. The advantages of digital over analogue far outweigh the disadvantages and communication is going ever more digital. Even today people who collect vinyl musical recordings (i.e. analogue) are regarded as eccentric (but then again music died in the 90's and most of the popular music released today sounds like someone falling down stairs whilst having a fight with a cat).

1.2 Digital storage

Digital signals are stored and accessed by computers, mobile phones, audio equipment etc. by making use of the **binary** number system. We are all used to using the decimal number system where strings consisting of 10 symbols, '0', '1', '2', '3', '4', '5', '6', '7', '8' and '9', along with the position of the symbols in the string are used to represent a number. For example,

$$\begin{aligned} 1234 &= 1 \text{ thousand} + 2 \text{ hundreds} + 3 \text{ tens} + 4 \text{ units} \\ &= 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0. \end{aligned}$$

The value of each digit is a multiple of 10 raised to some power. The digit on the far right multiplied by 10^0 and the powers of 10 increase by 1 as we step through each digit to the left. The decimal number system expresses numbers to the **base 10**.

Binary number are numbers expressed to the **base 2** where the value of each digit is a multiple of 2 raised to some power. Since we only need 2 symbols for each digit in a binary number we use '0' and '1'. For

example, consider the binary number 10110.

$$\begin{aligned} 10110_2 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 \\ &= 22_{10}. \end{aligned}$$

So the number 10110 expressed in binary has the same value as 22 in decimal. When dealing with numbers expressed to different bases we represent the base as a subscript following the number.

1.2.1 Converting a decimal number to a binary number

To convert a decimal number to a binary number we use the following steps

1. Divide the number by 2;
2. Get the integer part for the next iteration;
3. Get the remainder for the binary digit;
4. Repeat steps 1 – 3 until the integer part is 0. The binary string is a concatenation of the remainders starting at the last remainder and finishing at the first remainder.

For example, to convert the decimal number 22 to binary

Division by 2	Integer part	Remainder
$22 \div 2 = 11$	11	0
$11 \div 2 = 5.5$	5	1
$5 \div 2 = 2.5$	2	1
$2 \div 2 = 1$	1	0
$1 \div 2 = 0.5$	0	1

Therefore $22_{10} = 10110_2$ which was verified earlier.

1.2.2 Word length

The **word length** is the number of digits in a binary number. For example, the word length of 10110_2 is 5. Usually we remove any leading zeros from a binary number since they carry no value, e.g., $0011 = 11$, however, in computing it is common to express binary strings using a fixed word length so that multiple numbers can be expressed as one long **bit string**.

For example, consider the following bit string.

$$1011001001011101,$$

If we are using a word length of 4 then since there are 16 bits in the bit string we have a list of $16 \div 4 = 4$ numbers.

$$\underbrace{1011}_{11}\ \underbrace{0010}_{2}\ \underbrace{0101}_{5}\ \underbrace{1101}_{13} = (11, 2, 5, 13).$$

1.2.3 Units of memory

The memory required to store a bit string is measured in terms of the number of digits in the bit string. Each digit of a bit string is known as a **bit** and is the base unit of measurement of computer memory. In early computers, a word length 8 was used so that a single text character could be represented using 8 bits and since $2^8 = 256$ this allows for all upper and lower case characters in the Latin alphabet as well as punctuation marks and other special characters using ASCII (American Standard Code for Information Interchange) encoding. This lead to the definition of a **byte** which is 8 bits.

The values and symbols for common measures of computer memory are given in table 1.1.

Table 1.1: Units of measurement for computer memory.

Metric	Symbol	Value
bit	b	1 bit
byte	B	8 bits
kilobyte	kB	10^3 bytes
megabyte	MB	10^6 bytes
gigabyte	GB	10^9 bytes

1.2.4 Analogue-to-digital conversion

To be able to process sound signals using a computer we need to digitise analogue signals by passing them through an **Analogue-to-Digital Converter** (ADC). The digital signal is then processed, stored or transmitted and then converted back to an analogue signal by passing it through a **Digital-to-Analogue Converter** (DAC) (figure 1.2).

1.3 Sampling

To sample an analogue signal the time domain is divided into a finite number N of discrete time values t_i. The value of the signal s at each value of t_i is written as $s(t_i)$. We therefore have a finite sequence $[s(t_1), s(t_2), \ldots, s(t_N)]$ of discrete signal values.

Let T be the sample spacing in seconds called the **sampling interval**. Starting at time $t = 0$ the N sampled times are

$$t_0 = 0, \qquad t_1 = T, \qquad t_2 = 2T, \qquad \ldots \qquad t_{N-2} = (N-2)T, \qquad t_{N-1} = (N-1)T,$$

so we have the sequence of sampling times

$$t = (t_0, t_1, t_2, \ldots, t_{N-2}, t_{N-1}).$$

Note that the i^{th} time sample is $t_i = iT$ and $t_{i+1} = t_i + T$.

For technical reasons the signal value at a sample time is assumed constant until the next sample time (this is called 'sample and hold'). The signal has therefore been sampled over the time interval $t \in [0, NT]$. This gives us an expression for the **length** (or **duration**) of the signal

$$L = NT = \frac{N}{f_s}, \tag{1}$$

so the number of sample values for a signal of length L is

$$N = Lf_s. \tag{2}$$

It is more usual to talk about sampling a signal at a given rate which is called the **sampling frequency**. The sampling frequency is denoted by f_s and is the number of samples taken per second. The sampling frequency is measured in hertz (Hz) or more usually for sound in kilohertz (kHz). A sampling frequency of 1 kHz means that 1,000 samples are taken every second. Thus the sampling interval T for a sampling frequency of 1 kHz is 1/1,000th of a second. In general,

$$f_s = \frac{1}{T}, \tag{3}$$

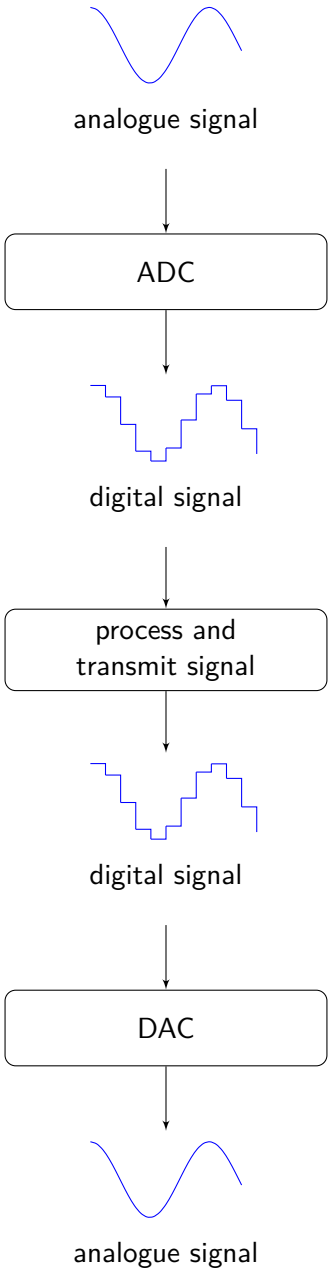


Figure 1.2: Flow chart showing the steps used in a typical sound processing implementation.

which means that

$$T = \frac{1}{f_s}. \tag{4}$$

Once the sample times are known we simply evaluate the signal at these times to get the N discrete time signal values

$$s(t) = (s(t_0), s(t_1), s(t_2), \ldots, s(t_{N-2}), s(t_{N-1})),$$

which are usually written as

$$s(t) = (s_0, s_1, s_2, \ldots, s_{N-2}, s_{N-1}).$$

The diagram in figure 1.3 illustrates the sampling process for an analogue signal.

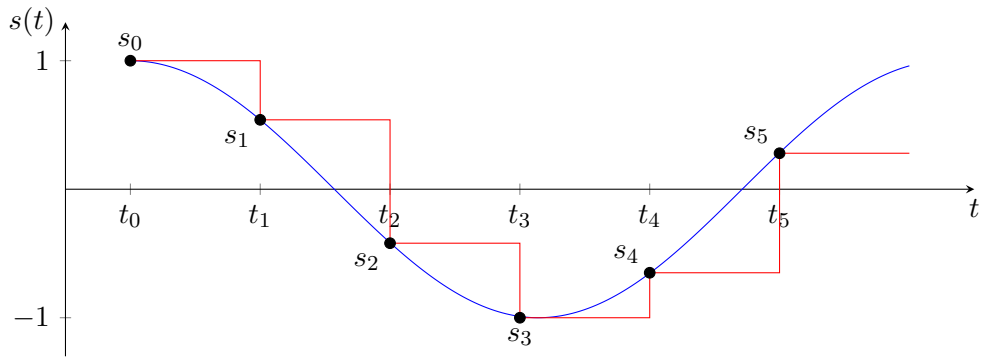


Figure 1.3: Uniform sampling of an analogue signal.

To save space only sample signal values are stored in a file and either the file also contains the sampling frequency (as in wav format) or the sampling frequency is given separately as an assumed value, e.g., $f_s = 44100$Hz is common for audio signals.

Example 1.1. An analogue signal of length 2 seconds is modelled by $s(t) = \cos(4t)$ and sampled using a sampling frequency of $f_s = 4$. Determine the digital signal.

Since $f_s = 4$ then by equation (4)

$$T = \frac{1}{f_s} = \frac{1}{4} = 0.25,$$

and the number of sampled points can be calculated using equation (2)

$$N = 2 \times 4 = 8.$$

So the sampling times are

$$t = (0, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75).$$

Note that the last time sample is $L - 1/f_s = 2 - 1/4 = 1.75$ and not 2.

The digital signal values are

$$s_i = \cos(4t_i),$$

which gives the following sampled signal

$$\begin{aligned} s(t) &= (\cos(0), \cos(1), \cos(2), \cos(3), \cos(4), \cos(5), \cos(6), \cos(7)) \\ &= (1, 0.5430, -0.4161, -0.9900, -0.6536, 0.2837, 0.9602, 0.7539). \end{aligned}$$

The sampled signal has been plotted in figure 1.4. Note that the sampled signal is constant over the sampling interval until the next sample time.

□

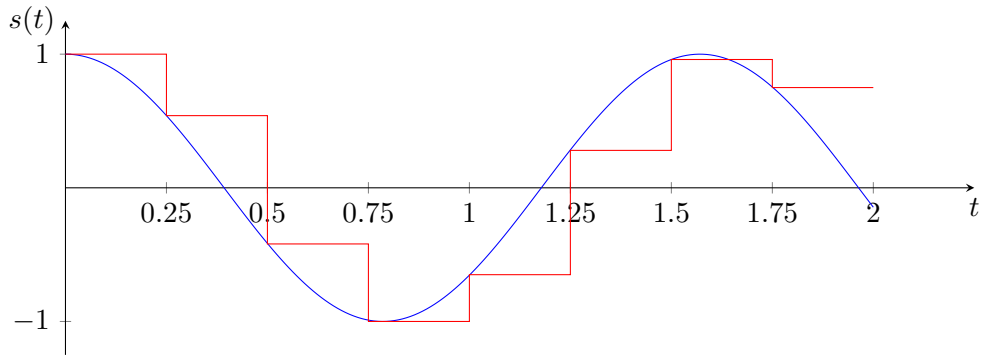


Figure 1.4: Digital sampling of an analogue signal.

1.4 Quantisation

After a signal has been digitised we now have the signal represented by a finite sequence

$$s(t) = (s_0, s_1, s_2, \ldots, s_{N-2}, s_{N-1}).$$

Each signal value, s_i, can be one of an infinite number of values so we need to restrict these values to a finite set. This process is called **quantisation**. The quantisation process uses two stages: the **classification stage** and the **reconstruction stage**.

1.4.1 Classification stage

The sampled signal values, s_i, are each mapped to a value from a finite sequence of **quantised values** that span the range of the signal. The number of quantised values used will effect the quality of the digitisation of the analogue signal. The more levels used, the better the digital approximation of the analogue signal, however this will mean more memory is required to store the digital signal.

The quantised signal values are then rounded to the corresponding **quantisation level** which is an integer in the range $[0, 2^M - 1]$ where M is the number of bits used to store a single sample value which is known as the **bit depth**. The spacing between each quantisation level, Δq is calculated using

$$\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1}. \tag{5}$$

where $s_{\min}$ and $s_{\max}$ are chosen so that $s(t) \in [s_{\min}, s_{\max}]$, for example, wav files use $s(t) \in [-1, 1]$. Equation (5) shows that the more bits are used to store a sampled value the more quantisation levels we have and therefore a better quality signal.

The quantised signal values are rounded to the nearest quantisation level using

$$q_i = \left\lfloor \frac{s_i - s_{\min}}{\Delta q} + \frac{1}{2} \right\rfloor, \tag{6}$$

where $\lfloor x \rfloor$ is the floor operator that rounds x to the integer below. For example, if we had a signal where $s \in [-1, 1]$ which was quantised using $M = 3$ we have

$$\Delta q = \frac{1 - (-1)}{2^3 - 1} = \frac{2}{7} = 0.2857,$$

and the quantisation levels correspond to the following signal values

$$(-1, -0.7143, -0.4286, -0.1429, 0.1429, 0.4286, 0.7143, 1).$$

The classification stage of a signal is illustrated in figure 1.5. The quantisation levels are represented using binary numbers of bit depth M. For this signal the binary encoding of the quantisation levels are shown in table 1.2.

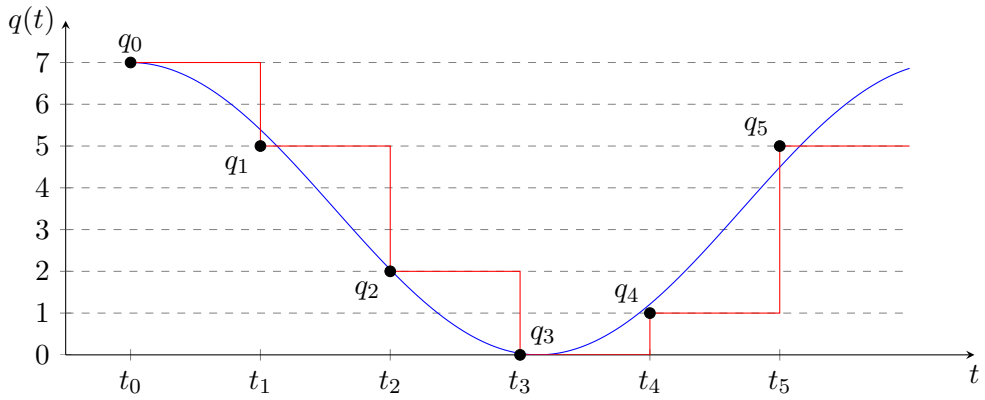


Figure 1.5: Digitised sampling rounded to the nearest quantisation level.

Table 1.2: Binary representation of a digitised signal.

Sample	Quantised value	Quantisation level	Natural binary encoding
s_0	1.0000	7	111
s_1	0.4286	5	101
s_2	-0.4286	2	010
s_3	-1.0000	0	000
s_4	-0.7143	1	001
s_5	0.4286	5	101

The binary representation of the quantisation level are concatenated to form one bit string

$$\underbrace{111}_{s_0}\ \underbrace{101}_{s_1}\ \underbrace{010}_{s_2}\ \underbrace{000}_{s_3}\ \underbrace{001}_{s_4}\ \underbrace{101}_{s_5}.$$

Example 1.2. Quantise the signal sampled in example 1.1 using a bit depth of $M = 2$ and determine its bit string.

The sampled signal is

$$s(t) = (1, 0.5493, -0.4161, -0.9900, -0.6536, 0.2837, 0.9602, 0.7539).$$

Since $s_{\min} = -1$, $s_{\max} = 1$ and $M = 2$ then using equation (5) we have

$$\Delta q = \frac{1-(-1)}{2^2-1} = \frac{2}{3},$$

and calculating the quantisation levels using equation (6)

$$q_0 = \left\lfloor \frac{1-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 3.5 \rfloor = 3, \qquad q_1 = \left\lfloor \frac{0.5493-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 2.8105 \rfloor = 2,$$
$$q_2 = \left\lfloor \frac{-0.4161-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 1.3758 \rfloor = 1, \qquad q_3 = \left\lfloor \frac{-0.9900-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 0.5150 \rfloor = 0,$$
$$q_4 = \left\lfloor \frac{-0.6536-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 1.0195 \rfloor = 1, \qquad q_5 = \left\lfloor \frac{0.2837-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 2.4255 = 2,$$
$$q_6 = \left\lfloor \frac{0.9602-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 3.4403 \rfloor = 3, \qquad q_7 = \left\lfloor \frac{0.7539-(-1)}{2/3} + \frac{1}{2} \right\rfloor = \lfloor 3.1309 \rfloor = 3.$$

These are converted to binary and written as the bit string

$$11\,10\,01\,00\,01\,10\,11\,11.$$

The quantised signal has been plotted in figure 1.6.

□

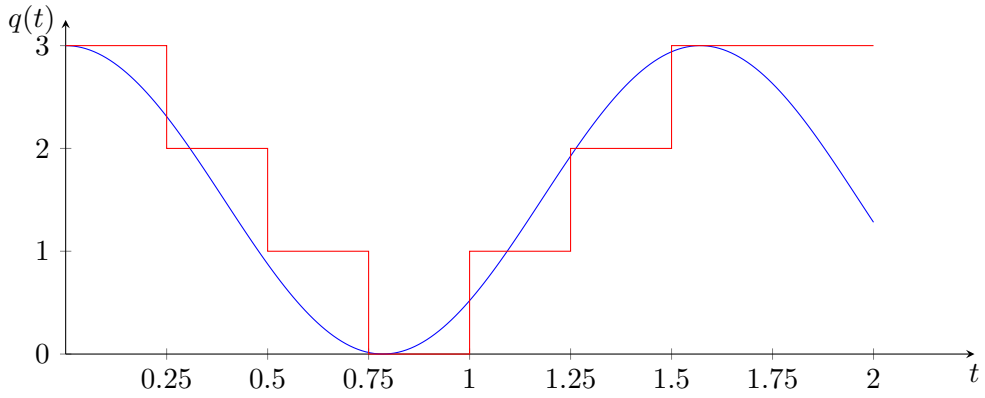


Figure 1.6: Quantisation of a digital signal.

1.4.2 Reconstructing stage

The **reconstruction stage** of the quantisation process is an attempt to reconstruct the original sampled signal. In addition to the quantised signal, the bit string that defines an audio signal will also contain information needed to reconstruct the signal, e.g., sampling frequency f_s, $s_{\min}$ and $s_{\max}$. The bit string that contains the signal is converted into decimal form to give the quantisation levels q. The reconstructed signal is calculated using

$$s_i = q_i \Delta q + s_{\min}. \tag{7}$$

We saw in table 1.2 that the signal from figure 1.3 is represented by the following bit string

$$111101010000001101.$$

We know that this was encoded using a bit depth of $M = 3$ and $s_{\min} = -1$ and $s_{\max} = 1$ so each quantised value is represented using 3 bits. Converting from binary to decimal gives

$$q = (7, 5, 2, 0, 1, 5).$$

If we are given $M = 3$, $s_{\min} = -1$ and $s_{\max} = 1$ then by equation (5)

$$\Delta q = \frac{1 - (-1)}{2^3 - 1} = \frac{2}{7},$$

and using equation (7) to reconstruct the digital signal

$$s_0 = 7\left(\frac{2}{7}\right) - 1 = 1, \qquad s_1 = 5\left(\frac{2}{7}\right) - 1 = 0.4286,$$
$$s_2 = 2\left(\frac{2}{7}\right) - 1 = -0.4286, \qquad s_3 = 0\left(\frac{2}{7}\right) - 1 = -1,$$
$$s_4 = 1\left(\frac{2}{7}\right) - 1 = -0.7143, \qquad s_5 = 5\left(\frac{2}{7}\right) - 1 = 0.4286.$$

which are the same as that in table 1.2 and illustrated in figure 1.7.

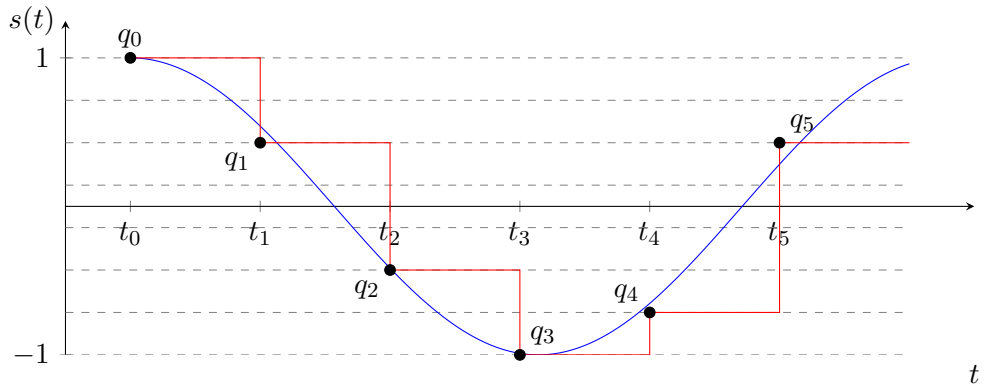


Figure 1.7: Reconstruction of a quantised signal.

If we know the sampling frequency f_s we can also determine the sampling times used. The length of the signal is calculated using

$$L = \frac{f_s \times \text{length of bit string}}{M},$$

and the sampling times are $t_i = iT$ where $T = 1/f_s$.

Note that it is not possible to reverse the quantisation process to reproduce the sampled signal exactly. This is because we have rounded the s_i values to the closest quantisation level.

Example 1.3. A signal was quantised in example 1.2 using a bit depth of $M = 2$, $s_{\min} = -1$ and $s_{\max} = 1$ resulting in the bit string

$$1110010001101111.$$

Reconstruct the sampled signal values.

Since $M = 2$ then we know that each sample point is represented by 2 bits, converting to decimal

$$q = (3, 2, 1, 0, 1, 2, 3, 3).$$

Using equation (5)

$$\Delta q = \frac{1 - (-1)}{2^2 - 1} = \frac{2}{3},$$

and using equation (7) we have

$$s_0 = 3\left(\frac{2}{3}\right) - 1 = 1, \qquad s_1 = 2\left(\frac{2}{3}\right) - 1 = 0.3333,$$
$$s_2 = 1\left(\frac{2}{3}\right) - 1 = -0.3333, \qquad s_3 = 0\left(\frac{2}{3}\right) - 1 = -1,$$
$$s_4 = 1\left(\frac{2}{3}\right) - 1 = -0.3333, \qquad s_5 = 2\left(\frac{2}{3}\right) - 1 = 0.3333,$$
$$s_6 = 3\left(\frac{2}{3}\right) - 1 = 1, \qquad s_7 = 3\left(\frac{2}{3}\right) - 1 = 1.$$

so

$$s = (1, 0.3333, -0.3333, -1, -0.3333, 0.3333, 1, 1).$$

The pre-quantised signal values were

$$s = (1, 0.5493, -0.4161, -0.9900, -0.6536, 0.2837, 0.9602, 0.7539),$$

which demonstrates the the reconstructed signal is only an approximation of the sampled signal.

The reconstructed signal has been plotted along with the analogue signal in figure 1.8. Note that the reconstructed signal does not lie exactly on the analogue signal at the sampling times, this is due to the rounding of the digital signal in the quantisation stage.

□

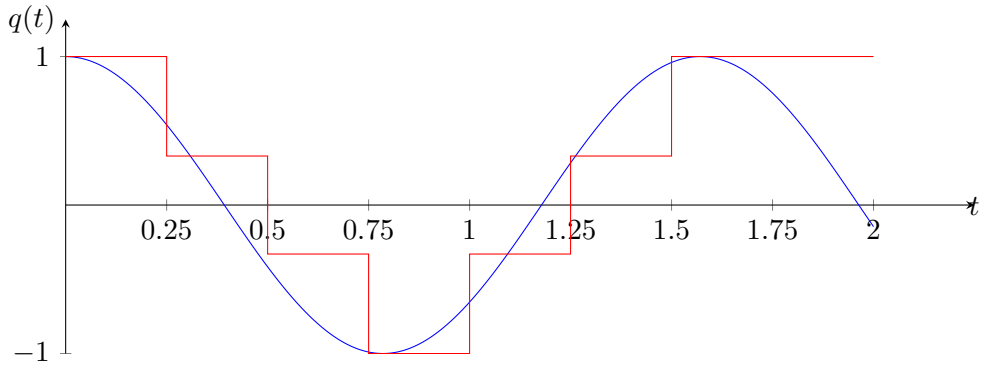


Figure 1.8: Digital signal reconstructed from the quantised signal.

1.4.3 Quantisation error

There is obviously some loss of information when a signal is converted from analogue to a digital signal since an infinite set of data (the analogue signal) is replaced by a finite set of data (the digital signal). Consider figure 1.9 where an analogue signal has been sampled using a sampling rate of $f_s = 100$Hz and quantised using a bit depth of $M = 3$. The plot of the quantisation error which is defined as the difference between the analogue and quantised signal for the corresponding time value shows that we have introduced noise when quantising the signal.

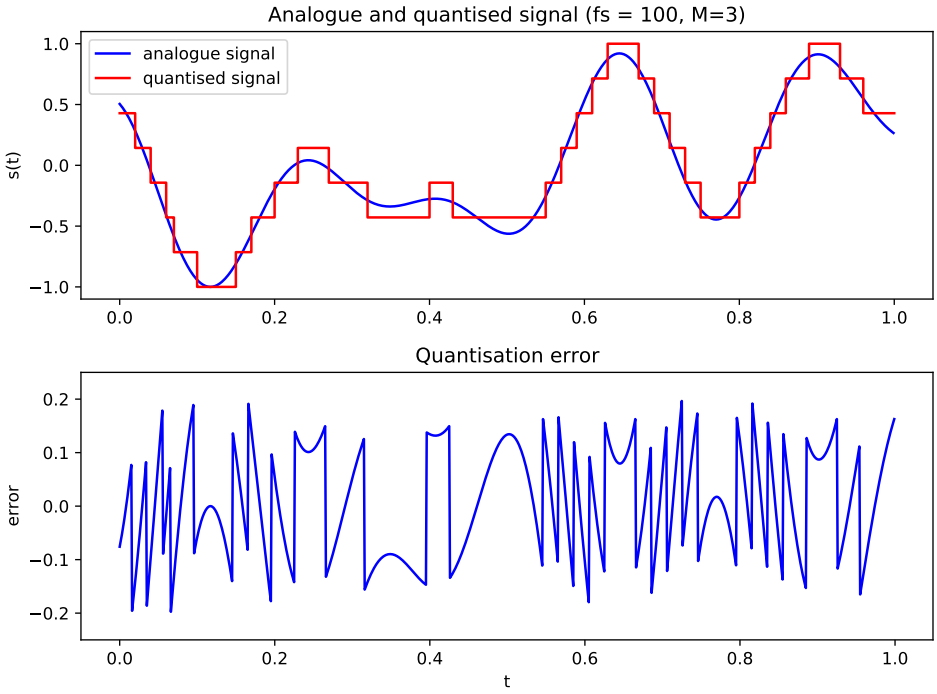


Figure 1.9: Plots of the original and quantised signal and the quantisation errors for $f_s = 100$Hz and $M = 3$.

The quantisation error is dependent on the bit depth used in quantisation. The larger the bit depth the

more quantisation levels we can use and therefore the closer the quantised signal is to the sampled signal. Figure 1.10 shows the same analogue signal from figure 1.9 quantised using a bit depth of $M = 8$.

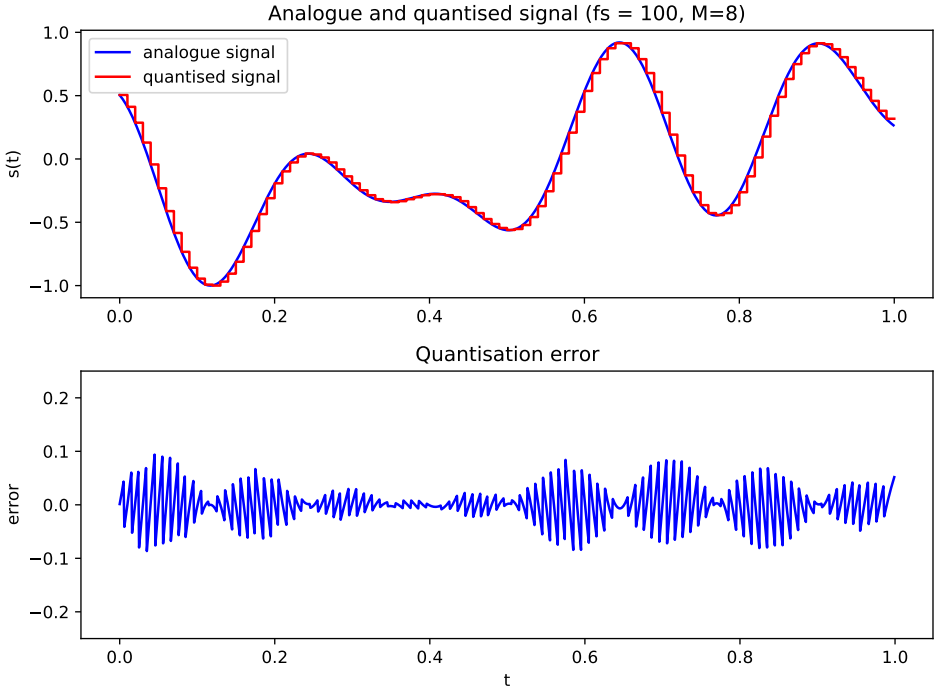


Figure 1.10: Plots of the original and quantised signal and the quantisation errors for $f_s = 100$Hz and $M = 8$.

We can calculate the a single value that measures the error between the analogue signal and quantised signal using the **Mean Square Error (MSE)** which is defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(s_q - s_a)^2. \tag{8}$$

where s_q is the quantised signal and s_a is the analogue signal. Performing this calculation for the two examples gives MSE $= 0.009$ for $M = 3$ and MSE $= 0.0008$ for $M = 8$ which confirms our visual analysis. The problem with using the MSE to analyse the quantisation error is that in practical cases we will not have the analogue signal which to calculate equation (8).

1.4.4 Bit rate

We have seen that increasing the sampling frequency and bit depth leads to a better reconstruction of the sampled signal and therefore a higher quality audio signal when passed through a DAC. Of course, increasing these also increases the length of the bit string (and therefore memory) used to store the signal.

The standard way to express the storage requirements for a digital signal is the **bit rate** which is defined as the number of bits of information that is conveyed of a unit of time. For audio signals, it is standard to express the bit rate in in **bits-per-second (bps)** and is calculated using

$$\text{bit rate} = \text{sampling frequency} \times \text{bit depth} \times \text{number of channels}. \tag{9}$$

For example, a 2-channel analogue signal sampled using $f_s = 1,000$Hz and quantised using 8 bits has a bit rate of $1000 \times 8 \times 2 = 16000$ bps or 16 kbps.

The total number of bits required to store a signal is calculated simply by multiplying the bit rate by the duration (in seconds) of the signal. So if our 2-channel signal was one minute in duration it would require $16 \times 60 = 960$ kb or 120 kB.

In practice the sampling frequency used is dependent upon the source of the analogue signal and its intended use. Speech that is transmitted between mobile phones needs to use minimal bandwidth and the audio does not have to be very clear in order for the listener to understand the message being communicated so a lower sampling frequency is used. Music played from a CD needs does not need to be transmitted via radio waves and should be indecipherable from the original analogue recording so requires a higher sampling frequency. The bitrates for some common applications are given in table 1.3.

Table 1.3: Sampling frequencies for common applications.

Application	No. channels	Sampling frequency	Bit depth	Bit rate
Speech transmission	1	8,000Hz	8	64 kbps
Audio CD	2	44,100Hz	16	1411.2 kbps
Blu-ray audio	7	192,000Hz	24	32.256 Mbps

1.5 Exercises

1. An analogue signal of length 5 seconds is digitised by sampling it 10 times starting at $t = 0$.
   (a) Calculate the sampling interval T;
   (b) Calculate the sampling frequency f_s;
   (c) List all the sampling times $(t_1, t_2, t_3 \ldots)$.
2. A speech signal is uniformly sampled 5000 times starting at $t = 0$ using a sampling frequency $f_s = 8$Hz.
   (a) calculate the sampling interval T;
   (b) calculate the length L of the signal;
   (c) write down the last sample time t_N.
3. An analogue signal is uniformly sampled at 1 kHz starting from $t = 0$ and the sample values stored in the array $s = (-2, 0, -2, 0, 3, 4, 2, -1)$.
   (a) Write down the sampling times $(t_1, t_2, t_3 \ldots)$;
   (b) Write down s_6;
   (c) At what time does the sampled signal have a value of 3?;
   (d) Sketch a plot of $s(t)$ against t.
4. A signal has the values $(-4.2, 0.1, -1.8, 2.9, 4.9, -0.5)$ is uniformly quantised using a bit depth of $M = 3$ with the first quantisation level corresponding to a signal value of -5 and the last level corresponding to 5.
   (a) Write down the signal values that correspond to each quantisation level;
   (b) Determine the quantisation levels for the signal;
   (c) Write down the bit string for the signal.
5. The following bit string represents signal that has values in the range $s(t) \in [-2, 2]$ has been sampling using a sampling frequency of $f_s = 5$Hz and quantised using a bit depth of $M = 2$

$$1011010010.$$

   (a) Write down the sampling times used;
   (b) Write down the quantised levels for the signal $(q_1, q_2, q_3, \ldots)$;
   (c) Reconstruct the signal;
   (d) Plot the reconstructed signal.
6. A single channel digital signal has been digitised using a bit rate of 12 kbs, a sampling frequency $f_s = 6$ kHz and required 360000 bytes of memory to store.
   (a) What is the sampling interval T?;
   (b) What is the value of t_{300}?;
   (c) What is the signal length L?
   (d) How many samples N are there?;
   (e) How many quantisation levels are used?
7. A CD can hold 750 MB of data. What is the maximum number of minutes of of a 2-channel audio signal that can be stored on a CD when digitised using:

(a) a sampling frequency of $f_s = 8,000$Hz and a bit depth of $M = 8$;

(b) a sampling frequency of $f_s = 44,100$Hz and a bit depth of $M = 16$.

8. The graph of an analogue signal is given in figure 1.11. Digitise this signal using a sampling frequency of $f_s = 1$ kHz and 3 bit quantisation with $s_{\min} = -2$ and $s_{\max} = 4$. Express your answer as a bit string.

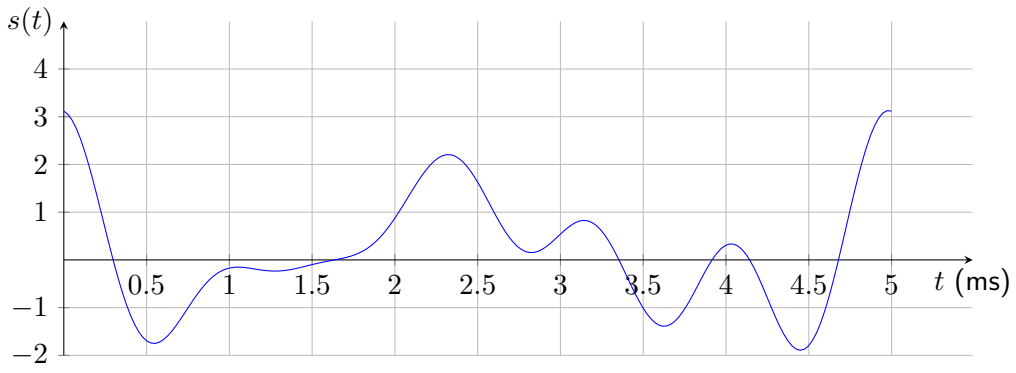


Figure 1.11: Analogue signal.

The solutions to these exercises are given on page 83.

Chapter 2

Spectral Analysis

2.1 Signal properties

Signals are described by two main properties: frequency and amplitude. Remarkably, all realistic signals, no matter how complicated, can be represented as a sum of simple periodic signals of known frequencies and amplitudes. This property allows us to be able to analyse signals based on these individual periodic signals. This technique is called **spectral analysis**

2.1.1 Signal frequency

The most significant property of a signal is the frequency which is the number of times that an event occurs over a period of 1 second and is measured in Hertz (Hz). Consider the plots of the two signals in figure 2.1, the signal on top varies rapidly up and down over the time interval $[0, 2]$ whereas the signal on the bottom varies much more slowly over the same interval. If we count the number of times the signals pass from negative to positive values (known as **up-crossing**) over a 1 second period the signal on the top crosses 6 times and the signal on the bottom crosses once. Therefore, the signal on the top has a frequency of 6Hz and doing similar for the signal on the bottom we see that it has a frequency of 1Hz. Note that we could have also counted the number of down-crossings, peaks or troughs over the same period and would have obtained the same frequencies.

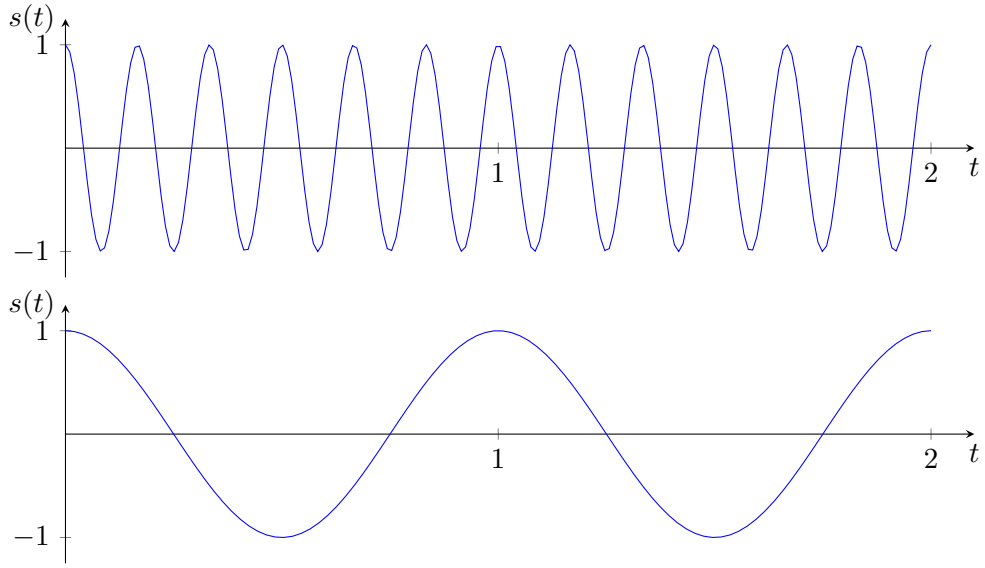


Figure 2.1: Plots of rapidly (top) and a slowly (bottom) varying signals.

You have all met frequency before in everyday life - radio channels are distinguished by frequency and that higher pitch sounds have a higher frequency. Also you may know that humans can only hear a limited

range of frequencies. Humans cannot hear high frequency dog whistles. Also the range of frequencies that we can hear is not the same for everyone, for example, younger people can hear higher frequencies that older people cannot. In general, the human ear can perceive sound in the frequency band 20Hz to 20 kHz. The ability to split a signal into frequency components is very useful and can be used to remove unwanted 'noise', compress the signal or automatically recognise signal elements like words, or parts of words, in human speech.

2.1.2 Sampling frequency

It is important to distinguish the signal frequency which is an inherent property of the analogue signal from the sampling frequency. The **sampling frequency** is the number of times per second that the analogue signal has been sampled (see section 1.3). The sampling frequency is chosen to be able to capture the properties of the analogue signal. For example, the higher frequency signal on the top in figure 2.1 will require a higher sampling frequency than the low frequency signal on the bottom. The question of what sampling frequency we should use is answered in section 2.2.1.

2.1.3 Periodic signals

Any signal that repeats itself indefinitely is called a **periodic signal**. The smallest value of the independent variable after which the signal repeats itself is called the **period**, P. A periodic signal over one period is sometimes called a **cycle** . The **fundamental frequency**, f_0 ("f nought"), of a periodic signal measures how many repetitions of the signal there are in a one second interval. f_0 is inversely proportional to P, i.e.,

$$P = \frac{1}{f_0}, \tag{10}$$

$$f_0 = \frac{1}{P}. \tag{11}$$

For example, a signal which repeats itself 5 times a second has a fundamental frequency of $f_0 = 5$Hz. Since there are 5 cycles in a second one cycle must last $1/5^{\text{th}}$ of a second, i.e. $P = 0.2$ s. Consider figure 2.2 which shows a periodic signal. Its period, P, is found by starting at any point and then measuring the time taken to reach the corresponding point on the next cycle.

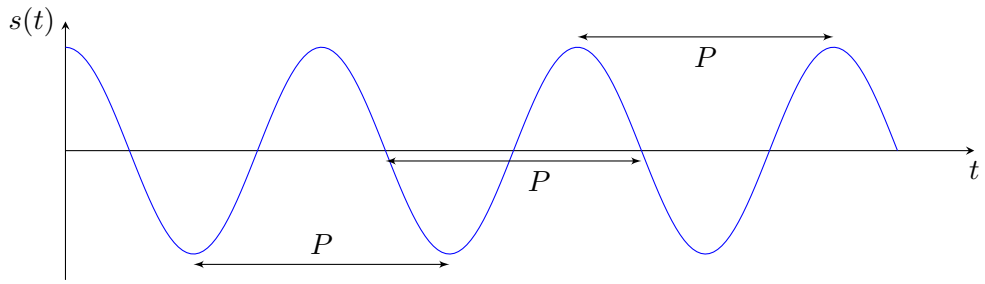


Figure 2.2: A periodic signal with period P.

2.1.4 Amplitude

The **amplitude** of a signal is the deviation in the signal from zero (figure 2.3). In audio signal, amplitude is directly related to volume and in radio signals the amplitude is related to the signal strength. For periodic signals, the amplitude is the peak deviation of the signal from zero.

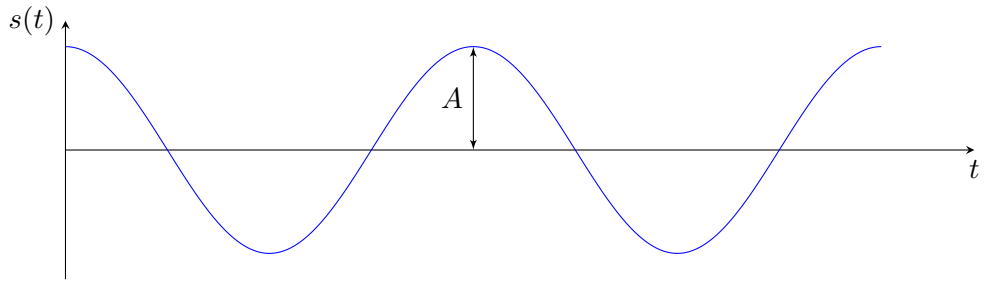


Figure 2.3: The amplitude of a periodic signal is the change in the signal over a single period.

2.2 Sinusoids

A **sinusoid** is a smooth periodic oscillating wave that is given by the time function

$$s(t) = A\cos(2\pi f_0 t + \phi), \tag{12}$$

where

- A is the **amplitude**;
- π (the Greek letter 'pi') is the constant $3.14159\ldots$;
- f_0 is the fundamental frequency of the wave;
- ϕ (the Greek letter 'phi') is the phase angle (in radians) where in the wave cycle the oscillation is at $t = 0$;
- t is time in seconds;
- $s(t)$ is the signal strength in volts (electric) or Pascals (pressure) depending on the medium.

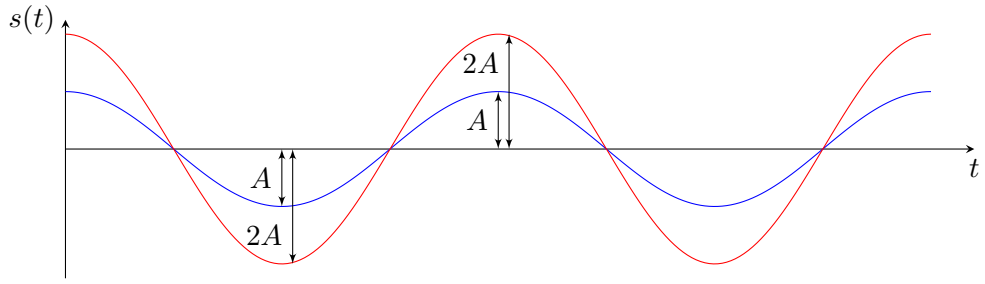
For sound signals is helps to think of A and f_0 as controlling the volume and pitch and ϕ is moving the signal along the time axis. Consider figure 2.4 which show how the sinusoid changes when the values of the amplitude, phase angle and frequency changes. In figure 2.4(a) when the amplitude is doubled this has the affect of increasing the deviation from zero. In figure 2.4(b) we can see that a negative value of ϕ shifts the sinusoid to the right along the time axis (a positive value of ϕ would shift the sinusoid to the left). In figure 2.4(c) we see that doubling the frequency halves the period of the sinusoid. A sinusoid with amplitude A is a minimum value of $-A$ and a maximum value of A so its **range** is $A - (-A) = 2A$.

Example 2.1. A sinusoid is given by $s(t) = 5\cos(8\pi t)$. What is the amplitude, fundamental frequency, phase angle and period of the sinusoid?

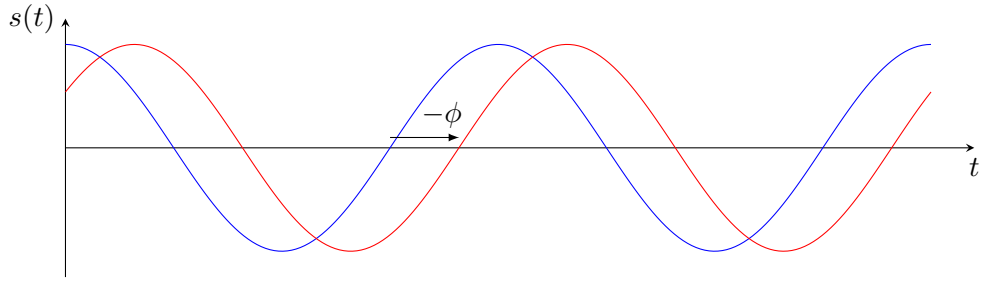
Comparing with equation (12) gives

- $A = 5$;
- $2\pi f_0 = 8\pi$ so $f_0 = 4$Hz which means $s(t)$ repeats 4 times a second;
- $\phi = 0$;
- $P = \dfrac{1}{f_0} = \dfrac{1}{4} = 0.25$ which means that one cycle of $s(t)$ lasts for 0.25 s.

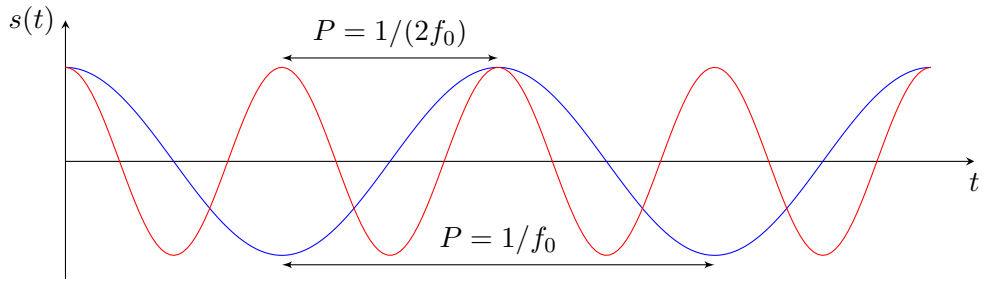
□



(a) changing the amplitude A



(b) changing the phase angle ϕ



(c) changing the frequency f_0

Figure 2.4: The affects of changing the amplitude, frequency and phase angle of a sinusoid.

2.2.1 The Nyquist frequency

We saw in chapter 1 that we need to digitise analogue signals by sampling them using the sampling frequency f_s which is measured inHz. If f_s is too small then the signals with high frequencies will be sampled at a rate which is not high enough the correctly capture the analogue signal, this results in an affect called **aliasing**. An example of aliasing is can be seen when filming a spoked wheel rotating where the frame rate of the camera (i.e., the sampling frequency) is less than that of the period the wheel is rotating which results in the wheel appearing to move too slowly or in reverse.

Consider figure 2.5 where two sinusoids have been fitted to the same sample points. If attempting to sample the higher frequency sinusoid using these sample points it would result in the lower frequency sinusoid.

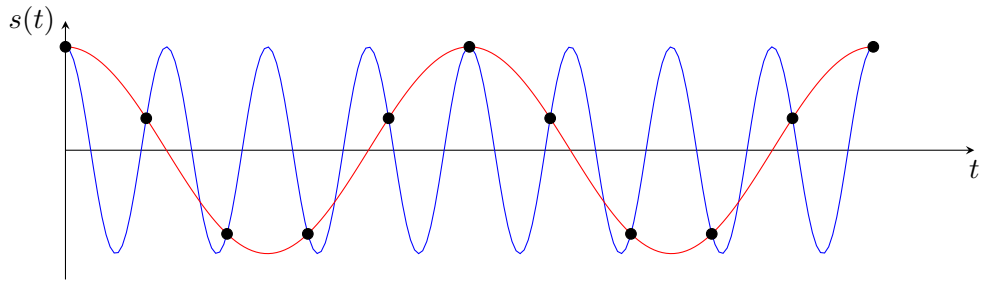


Figure 2.5: Two sinusoids fitted to the same sample points.

If f_s is too large then we are using more sampling points than we need to and thus using more memory. So the optimum sampling frequency for a signal is the smallest value of f_s which correctly samples the signal which is known as the **Nyquist frequency** (named after Harry Nyquist) which is denoted by f_{Nyq}.

Figure 2.6: Harry Nyquist (1889 – 1976)

We can make use of the Shannon-Whittaker sampling theorem to determine f_{Nyq}.

Theorem 2.2.1 (Shannon-Whittaker sampling theorem)**.** *If a function $s(t)$ contains no frequencies higher than $f_{\max}$Hz, it is completely determined by giving its ordinates at a series of points spaced $1/(2f_{\max})$ seconds apart.*

In other words, if f_s is the sampling frequency of an analogue signal then theorem 2.2.1 states that

$$f_s > 2f_{\max}, \tag{13}$$

so $f_{\text{Nyq}} = 2f_{\max}$, i.e., an analogue signal can be exactly represented by sampling at a rate greater than twice the highest frequency in the signal.

Notes:

- Theorem 2.2.1 shows that signals containing higher frequencies need smaller sampling intervals to discretise them.
- Sometimes the sampling frequency is expressed using the radians per second and denoted by ω_s where

$$\omega_s = 2\pi f_s$$

- In practice, analogue signals which are not band-limited are made so by removing the high frequencies before the ADC process.

Example 2.2. A signal is modelled by two sinusoids

$$s(t) = 10\cos(4000\pi t + 1.2) + 4\cos(8000\pi t).$$

What is the Nyquist frequency for this signal?

The maximum frequency is in the second sinusoid so comparing the second sinusoid to the general definition of a sinusoid $A\cos(2\pi f_0 t + \phi)$ gives

$$\begin{aligned} 2\pi f_0 &= 8000\pi \\ \therefore f_0 &= 4000\text{Hz}. \end{aligned}$$

so the maximum frequency for this signal is $f_{\max} = 4000$Hz and the Nyquist frequency is

$$f_{\text{Nyq}} = 2f_{\max} = 8000\text{Hz}.$$

□

2.3 The Fourier series

The brilliant French mathematician Joseph Fourier (1768 – 1830) showed that it is possible to represent a periodic function using a weighted sum of sinusoids. This weighted sum is known as the **Fourier series** and the sinusoids in the weighted sum are known as the **harmonics**. Fourier was working on the solution to the heat equation and had no idea that his series could be used to analyse signals. Without Fourier, mobile phones and digital media would not be possible so remember to think about him next time you are listening to Spotify or watching Netflix.

Figure 2.7: Joseph Fourier (1768 – 1830)

Consider a function $s(t)$ which is periodic over an interval of length P then this can be written as the Fourier series (known as the **sine-cosine form** of the Fourier series)

$$s(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t), \tag{14}$$

where

- $f_0 = 1/P$ is the fundamental frequency, i.e., the frequency of the 1st harmonic;
- $a_0/2$ is the amplitude of a harmonic with frequency 0Hz which is often called DC (from AC/DC electrical current);
- a_n is the amplitude of the cosine term of the n^{th} harmonic;
- b_n is the amplitude of the sine term of the n^{th} harmonic.

The values of a_n and b_n are calculated using

$$a_0 = \frac{2}{P}\int_P s(t)\,\mathrm{d}t, \tag{15}$$
$$a_n = \frac{2}{P}\int_P s(t)\cos(nf_0 t)\,\mathrm{d}t, \tag{16}$$
$$b_n = \frac{2}{P}\int_P s(t)\sin(nf_0 t)\,\mathrm{d}t. \tag{17}$$

The Fourier series means that any periodic signal with the fundamental frequency f_0m ay be written as a sum of terms which are periodic and have frequencies that are integer multiples of f_0. Since the sine function is a phase-shifted form of the cosine function so we can write the sine-cosine n^{th} harmonic using as a single cosine function, i.e.,

$$a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t) = A_n \cos(2\pi n f_0 t + \phi_n),$$

where

$$A_n = \sqrt{a_n^2 + b_n^2}, \tag{18}$$
$$\phi_n = -\operatorname{atan2}(b_n, a_n). \tag{19}$$

The $\operatorname{atan2}$ function is defined in appendix A.1.3 on page 80. Equation (14) can be written using only cosines by

$$s(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} A_n \cos(2\pi n f_0 t + \phi_n). \tag{20}$$

Equation (20) is known as the **amplitude-phase form** of the Fourier series.

Example 2.3. Find the amplitude-phase form of the Fourier series for a square wave with period of length $P = 2\pi$ defined as

$$s(t) = \begin{cases} 0, & t \in (-\pi, 0], \\ 1, & t \in (0, \pi]. \end{cases}$$

The fundamental frequency is $f_0 = 1/P = 1/2\pi$ and we calculate a_0, a_n and b_n using equations (15)

to (17). $s(t)$ is zero in $t \in [-\pi, 0]$ we only need to integrate over $(0, \pi]$

$$
\begin{aligned}
a_0 &= \frac{1}{\pi}\int_0^{\pi} 1 \,\mathrm{d}t = \frac{1}{\pi}[t]_0^{\pi} = 1, \\
a_n &= \frac{1}{\pi}\int_0^{\pi} 1 \cdot \cos(nt) \,\mathrm{d}t = \frac{1}{\pi}\left[\frac{1}{n}\sin(nt)\right]_0^{\pi} = \frac{1}{n\pi}(\sin(nt) - \sin(0)) = 0, \\
b_n &= \frac{1}{\pi}\int_0^{\pi} 1 \cdot \sin(nt) \,\mathrm{d}t = \frac{1}{\pi}\left[-\frac{1}{n}\cos(nt)\right]_0^{\pi} = \frac{1}{n\pi}(-\cos(n\pi) + \cos(0)) = \frac{1 - \cos(n\pi)}{n\pi}.
\end{aligned}
$$

Since $\cos(n\pi) = (-1)^n$ then

$$
b_n = \frac{1 - (-1)^n}{n\pi}.
$$

The values of A_n and ϕ_n for the amplitude-phase form of this Fourier series are calculated using equations (18) and (19)

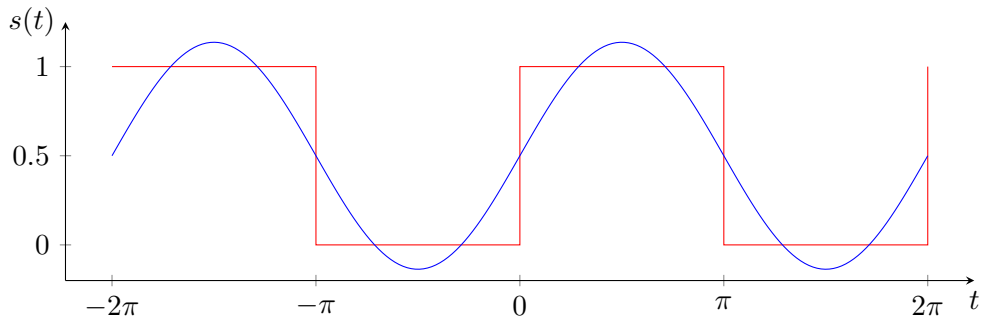
$$
\begin{aligned}
A_n &= \sqrt{0^2 + \left(\frac{1 - (-1)^n}{n\pi}\right)^2} = \frac{1 - (-1)^n}{n\pi}, \qquad n \geq 1, \\
\phi_n &= -\operatorname{atan2}\left(\frac{1 - (-1)^n}{n\pi}, 0\right) = -\frac{\pi}{2},
\end{aligned}
$$

so the amplitude-phase form of the Fourier series for the square wave is

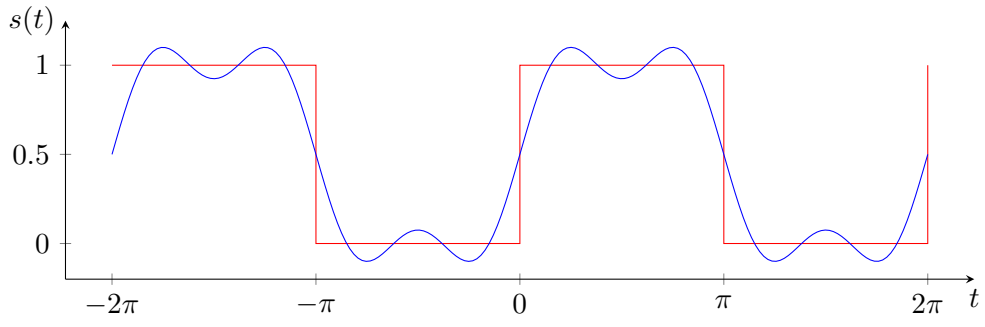
$$
\begin{aligned}
s(t) &= \frac{1}{2} + \sum_{n=1}^{\infty} \frac{1 - (-1)^n}{n\pi} \cos\left(nt - \frac{\pi}{2}\right) \\
&= \frac{1}{2} + \frac{2}{\pi}\cos\left(t - \frac{\pi}{2}\right) + \frac{2}{3\pi}\cos\left(3t - \frac{\pi}{2}\right) + \frac{2}{5\pi}\cos\left(5t - \frac{\pi}{2}\right) + \ldots
\end{aligned}
$$

Plots of the Fourier series of the square wave using 1, 4 and 8 harmonics are shown in figure 2.8. We can clearly see that the more harmonics that are included in the summation the closer we get to the square wave.

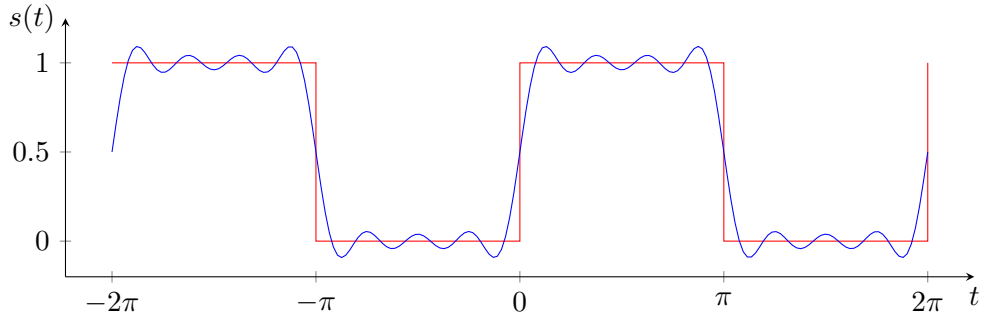
□



(a) first harmonic only



(b) first 4 harmonics



(c) first 8 harmonics

Figure 2.8: Fourier series approximation of the square wave from example 2.3.

2.3.1 Frequency domain

We have seen that we can plot a signal by plotting the signal, $s(t)$, against time, t. This is known as a plot of the **time domain**. Since using the Fourier series a signal can be written as a sum of harmonics we can also represent the signal as a plot of the amplitude of each harmonic against its frequency. This is known as a plot of the **frequency domain**.

Consider the diagram in figure 2.9 which shows the first five harmonics of a signal. The plot on the time-amplitude axes is the summation of the harmonics to give the signal in the time domain. We can also represent the Fourier series on the frequency-amplitude axes as a frequency spectrum which shows the same information but allows us to analyse the effects that each harmonic has on the signal.

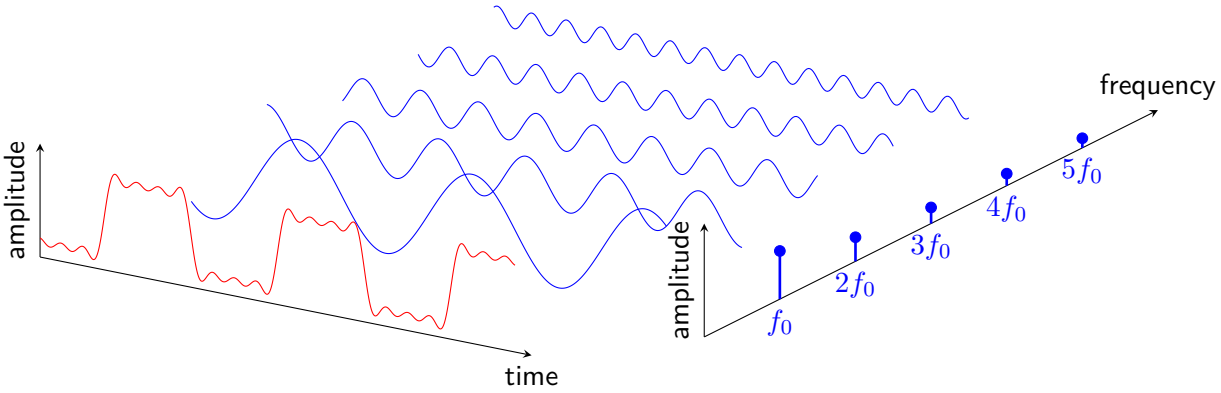


Figure 2.9: The time domain and frequency domain representations of harmonics.

Figure 2.10 shows the plot of the amplitudes for the harmonics A_n plotted against the frequencies nf_0. This plot is known as a **frequency spectrum** and is very useful in analysing a signal. Here each harmonic is represented by a vertical line known as a **spectral lines**. In practice where we may be analysing thousands of frequencies we tend to use a normal line plot for the frequency spectrum.

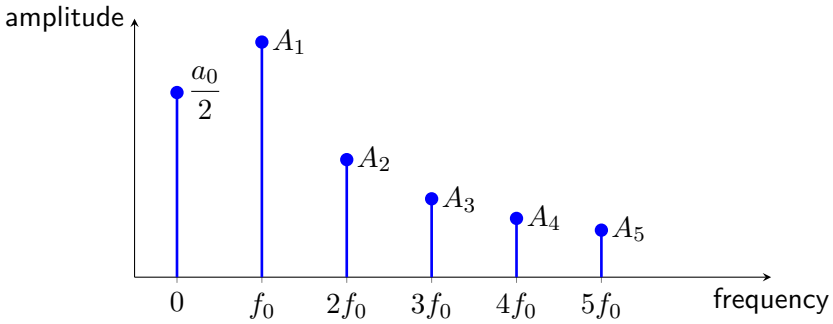


Figure 2.10: The frequency spectrum in the frequency domain.

Example 2.4. Plot the frequency spectrum for the Fourier series from example 2.3.

The fundamental frequency was $f_0 = 1/(2\pi)$ so the frequencies for the first 7 harmonics are

$$f_0 = \frac{1}{2\pi}, \quad 2f_0 = \frac{2}{2\pi}, \quad 3f_0 = \frac{3}{2\pi}, \quad 4f_0 = \frac{4}{2\pi}, \quad 5f_0 = \frac{5}{2\pi}, \quad 6f_0 = \frac{6}{2\pi}, \quad 7f_0 = \frac{7}{2\pi}, \quad \ldots$$

and the amplitudes are

$$\frac{a_0}{2} = \frac{1}{2}, \quad A_1 = \frac{2}{\pi}, \quad A_2 = 0, \quad A_3 = \frac{2}{3\pi}, \quad A_4 = 0, \quad A_5 = \frac{2}{5\pi}, \quad a_6 = 0, \quad A_7 = \frac{2}{7\pi}, \quad \ldots$$

The frequency spectrum is plotted in figure 2.11.

□

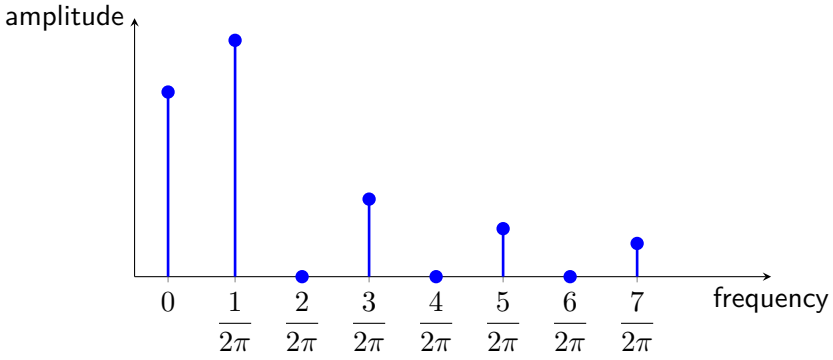


Figure 2.11: Plot of the frequency spectrum.

2.3.2 Exponential form of the Fourier series

The Fourier series in equation (14) can be expressed using exponential functions which is a more convenient form when analysing signals using Fourier transforms (see sections 2.4 and 2.6). The **exponential form of the Fourier series** is

$$s(t) = \sum_{n=-\infty}^{\infty} c_n e^{i2\pi n f_0 t} \tag{21}$$

where $i^2 = -1$ is the imaginary number and $c_n \in \mathbb{C}$ are **complex coefficients** which relate to the DC and amplitudes a_n and b_n in the sine-cosine form of the Fourier series. For the DC (zero frequency) term we have

$$\frac{a_0}{2} = c_0 e^0 = c_0.$$

For the harmonics in the Fourier series we need to ensure that equations (14) and (21) are equivalent. Each harmonic in equation (21) consists of a positive and 'negative' frequencies, i.e.,

$$a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t) = c_n e^{i2\pi n f_0 t} + c_{-n} e^{-i2\pi n f_0 t}.$$

Note that since the left-hand side is real then the right-hand side must also be real. Since $e^{i2\pi n f_0 t}$ is the complex conjugate of $e^{-i2\pi n f_0 t}$ (i.e., the sign of the imaginary part is negated) then c_{-n} must also be the complex conjugate of c_n (i.e, $c_{-n} = \bar{c}_n$), therefore the exponential form of the Fourier series can be written using a summation over the positive frequencies

$$s(t) = |c_0| + \sum_{n=1}^{\infty} c_n e^{i2\pi n f_0 t} + c_n^* e^{-i2\pi n f_0 t}. \tag{22}$$

To find the relationship between the complex coefficients c_n used in the exponential form of the Fourier series and the real coefficients a_n and b_n used in the sine-cosine form of the Fourier series we simply equate the n^{th} harmonic. Using Euler's formula, equation (45) on page 82, and let $c_n = x + iy$ and $c_{-n} = x - iy$ then for the n^{th} harmonic we have

$$\begin{aligned}
a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t) &= c_n e^{i2\pi n f_0 t} + c_{-n} e^{-i2\pi n f_0 t} \\
&= (x + iy)(\cos(2\pi n f_0 t) + i \sin(2\pi n f_0 t)) \\
&\quad + (x - yi)(\cos(2\pi n f_0 t) - i \sin(2\pi n f_0 t)) \\
&= x \cos(2\pi n f_0 t) + ix \sin(2\pi n f_0 t) + iy \cos(2\pi n f_0 t) \\
&\quad - y \sin(2\pi n f_0 t) + x \cos(2\pi n f_0 t) - ix \sin(2\pi n f_0 t) \\
&\quad - iy \cos(2\pi n f_0 t) - y \sin(2\pi n f_0 t) \\
&= 2x \cos(2\pi n f_0 t) - 2y \sin(2\pi n f_0 t),
\end{aligned}$$

so

$$a_n = 2\,\mathrm{Re}(c),$$
$$b_n = -2\,\mathrm{Im}(c),$$

therefore the c_n values in equation (22) can be calculated from a_0, a_n and b_n using

$$c_n = \begin{cases} \frac{a_0}{2}, & n = 0, \\ \frac{1}{2}(a_n - ib_n), & n > 0, \\ \frac{1}{2}(a_n + ib_n), & n < 0. \end{cases} \tag{23}$$

The amplitudes of the positive and negative frequencies are

$$|c_n| = |c_{-n}| = \frac{1}{2}\sqrt{a_n^2 + b_n^2}, \tag{24}$$

The frequency spectrum for the exponential Fourier series is symmetric about the vertical axis as shown in figure 2.12.

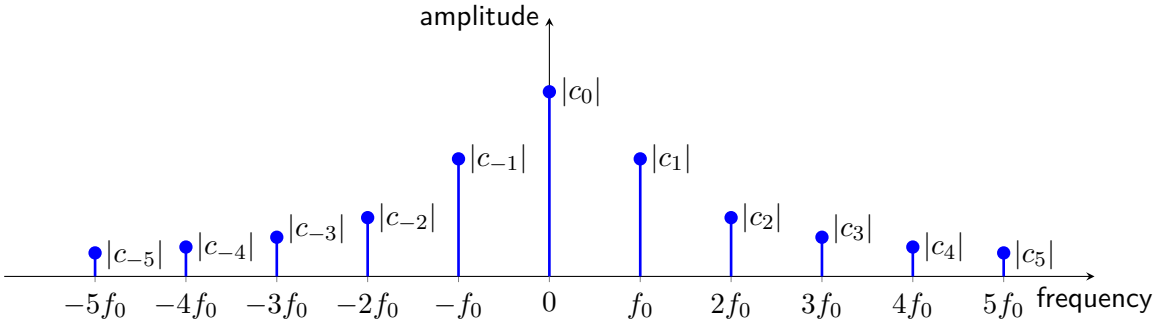


Figure 2.12: The two-sided frequency spectrum.

It is more common to represent the frequency spectrum as single sided frequency spectrum which is equivalent to the frequency spectrum using the sine-cosine form of the Fourier series. This is done by adding the amplitudes of the negative frequencies to the corresponding amplitudes of the positive frequencies, i.e.,

$$\frac{a_0}{2} = |c_0|, \tag{25}$$
$$A_n = 2|c_n|, \qquad n = 1, 2, \ldots, \tag{26}$$

which is shown in figure 2.13.

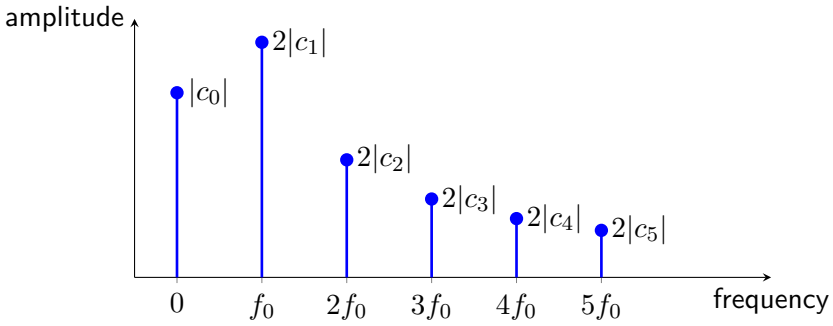


Figure 2.13: The single-sided frequency spectrum.

Example 2.5. Find the exponential form of the Fourier series for a square wave with period of length $P = 2\pi$ defined as

$$s(t) = \begin{cases} 0, & t \in [-\pi, 0], \\ 1, & t \in (0, \pi]. \end{cases}$$

We saw in example 2.3 that the sine-cosine form of the Fourier series for this wave was represented by

$$\begin{aligned} a_0 &= 1, \\ a_n &= 0, \\ b_n &= \frac{1 - \cos(n\pi)}{n\pi}. \end{aligned}$$

Using equation (23) to calculate the complex coefficients

$$\begin{aligned} c_0 &= \frac{a_0}{2} = \frac{1}{2}, \\ c_n &= \frac{1}{2}(a_n - ib_n) = \frac{1}{2}\left(0 - i\left(\frac{1 - \cos(n\pi)}{n\pi}\right)\right) = -\frac{i}{2n\pi}(1 - (-1)^n), \\ c_{-n} &= \frac{1}{2}(a_n + ib_n) = \frac{1}{2}\left(0 + i\left(\frac{1 - \cos(n\pi)}{n\pi}\right)\right) = \frac{i}{2n\pi}(1 - (-1)^n). \end{aligned}$$

Note that $1 - (-1)^n$ is zero when n is even and 2 when n is odd. Since $P = 2\pi$ then $f_0 = 1/(2\pi)$ and the exponential form of the Fourier series is

$$\begin{aligned} s(t) &= \sum_{n=-\infty}^{\infty} -\frac{i}{2|n|\pi}(1 - (-1)^2)e^{int} \\ &= \frac{1}{2} - \underbrace{\left(\frac{i}{\pi}e^{it} - \frac{i}{\pi}e^{-it}\right)}_{\text{1st harmonic}} - \underbrace{\left(\frac{i}{3\pi}e^{3it} - \frac{i}{3\pi}e^{-3it}\right)}_{\text{3rd harmonic}} - \underbrace{\left(\frac{i}{5\pi}e^{5it} - \frac{i}{5\pi}e^{-5it}\right)}_{\text{5th harmonic}} - \dots \end{aligned}$$

The frequencies and amplitudes of the harmonics are the same as those calculated in example 2.3 and frequency spectrum for $s(t)$ is equivalent to that shown in figure 2.11.

□

2.4 The Discrete Fourier Transform (DFT)

The Fourier transform is a linear transformation $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$ defined by

$$\mathcal{F}(s(t))_\nu = \int_{-\infty}^{\infty} s(t)e^{-i2\pi\nu t}\,\mathrm{d}t, \tag{27}$$

where $s(t)$ is a periodic function. In signals processing the analogue signal $s(t)$ is sampled at discrete points in t so we have $(s_0, s_1, \dots, s_{N-1})$ where $s_i = iT$, N is the number of sample points and $T = 1/f_s$ is the sampling interval. The integrand in equation (27) exists at these sample points so if $F_n = \mathcal{F}[\{s_0, s_1, s_2, \dots, s_{N-1}\}](n)$ then

$$F_n = \sum_{k=0}^{N-1} s_k e^{-(i2\pi kn)/N}. \tag{28}$$

Equation (28) is the **Discrete Fourier Transform (DFT)** and is incredibly useful in signal processing because it relates a discrete signal in the time domain defined using N points to a discrete periodic signal

in the frequency domain with period $P = N$. We can use equation (28) to approximate the the amplitudes in the exponential form of the Fourier series using

$$c_n \approx \frac{F_n}{N}. \tag{29}$$

But what about the approximations of c_{-n}? Since the signal is periodic we have $F_{-n} = F_{N-n}$ and because $c_{-n} = \overline{c}_n$ the

$$c_{-n} \approx \frac{F_{N-n}}{N} = \frac{\overline{F}_n}{N}.$$

The application of the DFT produces an N element array F. We can divide F by N to give the complex coefficients where the first element F_0 is the DC, the elements $F_1, F_2, \ldots F_n$ are the complex coefficients $c_1, c_2, \ldots, c_n$ and the elements $F_{N-n}, F_{N-n+1}, \ldots, F_N$ are the complex coefficients $c_{-n}, c_{-n+1}, \ldots, c_{-1}$ (see figure 2.14 for how these relate to each other). Since $c_{-n} = \overline{c}_n$ then given a signal of N sample points we can approximate up to the n^{th} harmonic where

$$n = \left\lfloor \frac{N-1}{2} \right\rfloor, \tag{30}$$

and the fundamental frequency is

$$f_0 = \frac{f_s}{N}, \tag{31}$$

where f_s is the sampling frequency.

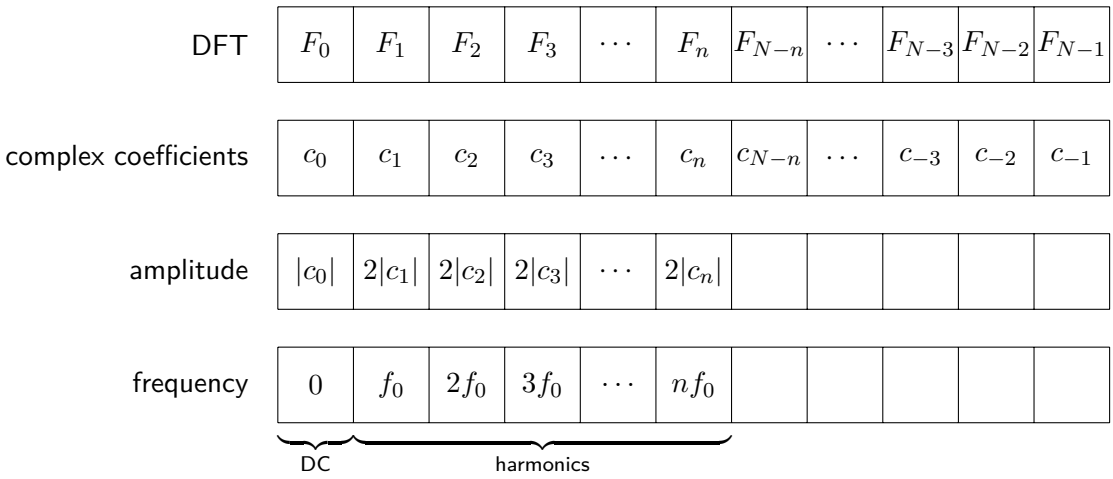


Figure 2.14: The structure of the F array which is produced by the DFT.

2.4.1 The DFT matrix

For calculating a DFT it is convenient to represent the summation in equation (28) as the matrix equation

$$F = W \cdot s,$$

where $F = (F_0, F_1, F_2, \ldots, F_{N-1})^T$, $s = (s_0, s_1, s_2, \ldots, s_{N-1})^T$ and W is the **DFT matrix** given by

$$W = \begin{pmatrix} e^0 & e^0 & e^0 & \cdots & e^0 \\ e^0 & e^{-i2\pi/N} & (e^{-i2\pi/N})^2 & \cdots & (e^{-i2\pi/N})^{N-1} \\ e^0 & (e^{-i2\pi/N})^2 & (e^{-i2\pi/N})^4 & \cdots & (e^{-i2\pi/N})^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^0 & (e^{-i2\pi/N})^{N-1} & (e^{-i2\pi/N})^{2(N-1)} & \cdots & (e^{-i2\pi/N})^{(N-1)(N-1)} \end{pmatrix} \tag{32}$$

Note the following:

- W is a symmetric matrix, i.e., $W = W^T$;
- Since $F_n = \overline{F}_{-n}$ we only need to calculate the first $\lfloor (N+1)/2 \rfloor$ rows of W;
- $[W]_{nk} = (e^{-i2\pi/N})^{(n-1)(k-1)}$, i.e., the powers of the elements in W follow

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 2 & \cdots & N-1 \\ 0 & 2 & 4 & \cdots & 2(N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & N-1 & 2(N-1) & \cdots & (N-1)(N-1) \end{pmatrix};$$

- The sequence $\{e^{-i2\pi/N}, (e^{-i2\pi/N})^2, (e^{-i2\pi/N})^3, \ldots\}$ is periodic with period N (i.e., $(e^{-i2\pi/N})^N = 1$) so

$$(e^{-i2\pi/N})^n = (e^{-i2\pi/N})^{n+N} = (e^{-i2\pi/N})^{n+2N} = \ldots$$

Example 2.6. A periodic analogue signal has been sampled using a sampling frequency of $f_s = 10$Hz and the sampled signal is

$$s = (2, 0, -1, 1, 0).$$

Calculate the DFT for the sampled signal, plot the Fourier series approximation of this signal and the frequency spectrum.

The sample interval is $T = 1/f_s = 1/10$ so the sample times are

$$t = (0, 0.1, 0.2, 0.4, 0.5).$$

We need to calculate the elements of the DFT matrix, since $N = 5$ then we only need to calculate the first $\lfloor (5+1)/2 \rfloor = 3$ rows of W. The powers of $e^{-i2\pi/N}$ in W are

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 2 & 4 & 6 & 8 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

so

$$\begin{aligned} e^{-i2\pi/5} &= \cos\left(\frac{2\pi}{5}\right) - i\sin\left(\frac{2\pi}{5}\right) = 0.31 - 0.95i, \\ (e^{-i2\pi/5})^2 &= (-0.81 - 0.95i)^2 = -0.81 + 0.59i, \\ (e^{-i2\pi/5})^3 &= e^{-i2\pi/5}(e^{-i2\pi/5})^2 = (0.31 + 0.95i)(-0.81 + 0.59i) = -0.81 - 0.59i, \\ (e^{-i2\pi/5})^4 &= (e^{-i2\pi/5})^2(e^{-i2\pi/5})^2 = (-0.81 + 0.59i)^2 = 0.31 + 0.95i, \\ (e^{-i2\pi/5})^6 &= e^{-i2\pi/5} = 0.31 - 0.95i, \\ (e^{-i2\pi/5})^8 &= (e^{-i2\pi/5})^3 = -0.81 + 0.59i, \end{aligned}$$

and the DFT of the sampled signal is

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0.31 - 0.95i & -0.81 - 0.59i & -0.81 + 0.59i & 0.31 + 0.95i \\ 1 & -0.81 - 0.59i & 0.31 + 0.95i & 0.31 - 0.95i & -0.81 + 0.59i \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} 2 \\ 0 \\ -1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 + 1.18i \\ 2 - 1.90i \\ \vdots \end{pmatrix}.$$

Since $N = 5$ we can approximate up to harmonic $n = \lfloor (5-1)/2 \rfloor = 2$ so the complex coefficients are

$$\begin{aligned} c_0 &= \frac{2}{5} = 0.4, \\ c_1 &= \frac{2 + 1.18i}{5} = 0.4 + 0.24i, \\ c_2 &= \frac{2 - 1.9i}{5} = 0.4 - 0.38i, \end{aligned}$$

and $c_{-1} = \overline{c_1}$ and $c_{-2} = \overline{c_2}$. The sampling frequency is $f_s = 10$Hz so the fundamental frequency is $f_0 = f_s/N = 10/5 = 2$Hz and the Fourier series approximation of this signal is

$$s(t) = 0.4 + (0.4 + 0.25i)e^{i4\pi t} + (0.4 - 0.25i)e^{-i4\pi t} + (0.4 - 0.38i)e^{i8\pi t} + (0.4 + 0.38i)e^{-i8\pi t}.$$

The amplitude of the DC and first and second harmonics are

$$\begin{aligned} \frac{a_0}{2} &\approx \left| \frac{2}{5} \right| = 0.4, \\ A_1 &\approx 2\,|c_1| = 2\sqrt{0.4^2 + 0.24^2} = 0.93, \\ A_2 &\approx 2\,|c_2| = 2\sqrt{0.4^2 + (-0.38)^2} = 1.1. \end{aligned}$$

The Fourier series for this signal has been plotted against the sampled signal values in figure 2.15. Note that the Fourier series matches the signal values at the sampling times. This does not mean that the signal values were sampled from the Fourier series, just that this Fourier series was the best approximation. If the sampling frequency f_s was too low then we would see aliasing (see section 2.2.1). The frequency spectrum is shown in figure 2.16.

□

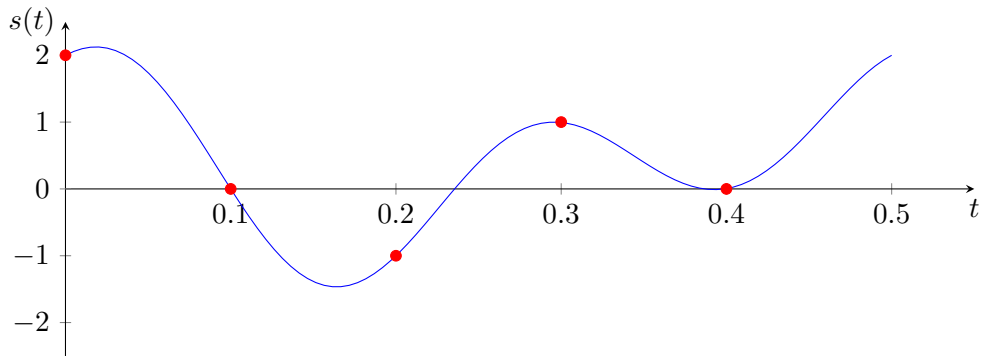


Figure 2.15: Plot of the Fourier series approximation of the sampled signal from example 2.6.

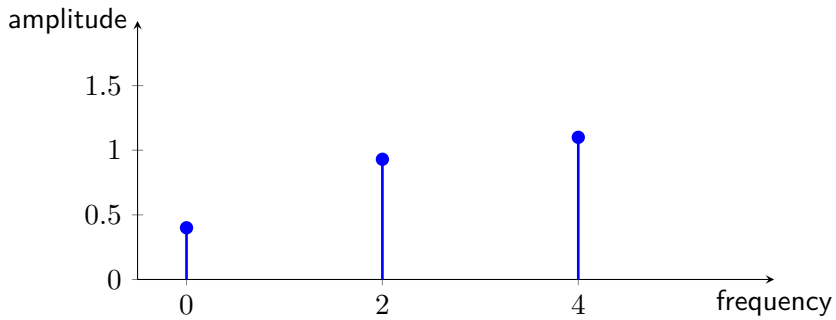


Figure 2.16: The frequency spectrum for the sampled signal from example 2.6.

2.4.2 Inverse discrete Fourier transform

Since the DFT is a linear transformation then it has an inverse transformation $\mathcal{F}^{-1} : \mathbb{C}^N \to \mathbb{C}^N$ which is defined by

$$s_n = \frac{1}{N} \sum_{k=0}^{N-1} F_n e^{(i2\pi kn)/N}. \tag{33}$$

Equation (33) is known as the **Inverse Discrete Fourier Transform (IDFT)** and is used to calculate the signal in the time domain given the DFT values, F_n. As with the DFT, for calculating the IDFT it is convenient to write the equation (33) as the matrix equation

$$s = W^{-1}F,$$

where W^{-1} is the **IDFT matrix** is

$$W^{-1} = \frac{1}{N} \begin{pmatrix} e^0 & e^0 & e^0 & \cdots & e^0 \\ e^0 & e^{i2\pi/N} & (e^{i2\pi/N})^2 & \cdots & (e^{i2\pi/N})^{N-1} \\ e^0 & (e^{i2\pi/N})^2 & (e^{i2\pi/N})^4 & \cdots & (e^{i2\pi/N})^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e^0 & (e^{i2\pi/N})^{N-1} & (e^{i2\pi/N})^{2(N-1)} & \cdots & (e^{i2\pi/N})^{(N-1)(N-1)} \end{pmatrix}. \tag{34}$$

Note that the elements in the IDFT matrix are the complex conjugates of the elements from the DFT matrix in equation (32).

Example 2.7. The complex coefficients of a signal sampled at $f_s = 10$Hz in the frequency domain are

$$F = (2, 2 + 1.18i, 2 - 1.9i, 2 + 1.9i, 2 - 1.18i).$$

Use the IDFT to calculate the signal in the time domain.

Here $N = 5$ so the elements of the IDFT matrix are the complex conjugates of the elements in W calculated in example 2.6, therefore

$$\begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0.31 + 0.95i & -0.81 + 0.59i & -0.81 - 0.59i & 0.31 - 0.95i \\ 1 & -0.81 + 0.59i & 0.31 - 0.95i & 0.31 + 0.95i & -0.81 - 0.59i \\ 1 & -0.81 - 0.59i & 0.31 + 0.95i & 0.31 - 0.95i & -0.81 + 0.59i \\ 1 & 0.31 - 0.95i & -0.81 - 0.59i & -0.81 + 0.59i & 0.31 + 0.95i \end{pmatrix} \begin{pmatrix} 2 \\ 2 + 1.18i \\ 2 - 1.9i \\ 2 + 1.9i \\ 2 - 1.18i \end{pmatrix}$$

$$= \begin{pmatrix} 2 + 0i \\ 0 + 0i \\ -1 + 0i \\ 1 + 0i \\ 0 + 0i \end{pmatrix},$$

therefore $s = (2, 0, -1, 1, 0)$ which is the sampled signal from example 2.6. Since $T = 1/f_s = 1/10$s we know that these sample points correspond to times $t = (0, 0.1, 0.2, 0.3, 0.4)$ and we can reproduce the signal.

□

2.5 Windowing

We have seen that the Fourier series allows us to represent any periodic signal using a sum of harmonics that can be found using the DFT. Unfortunately most signals in real life are not periodic. For example,

recall the plot of the signal of Handel's 'Hallelujah Chorus' from *Messiah* which is reproduced in figure 2.17. Here there is no obvious periodic behaviour because the signal is not periodic.

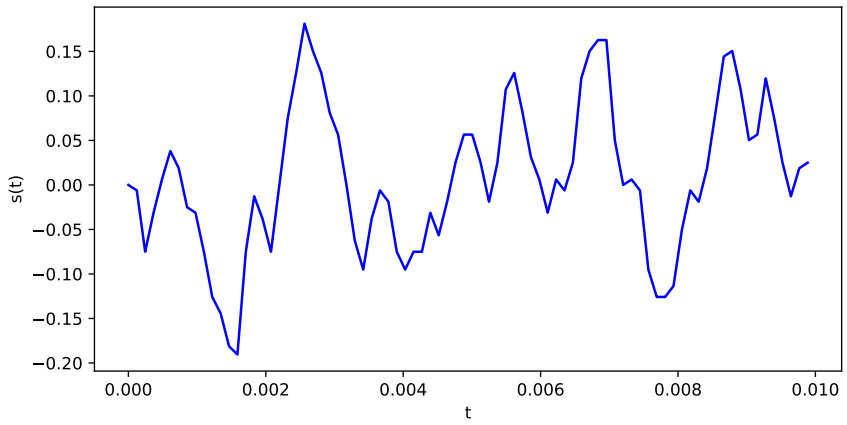


Figure 2.17: Plot of the signal for Handel's 'Hallelujah Chorus' from *Messiah*.

So how can we use a DFT that assumes a signal is periodic to calculate the amplitudes of the harmonics? The answer is we divide the non-periodic signal into intervals of length P seconds and assume that this interval contains a single cycle of a periodic signal with period P. This is of course a simplifying assumption but if P is sufficiently small any errors should be undetectable. This process is called **windowing**.

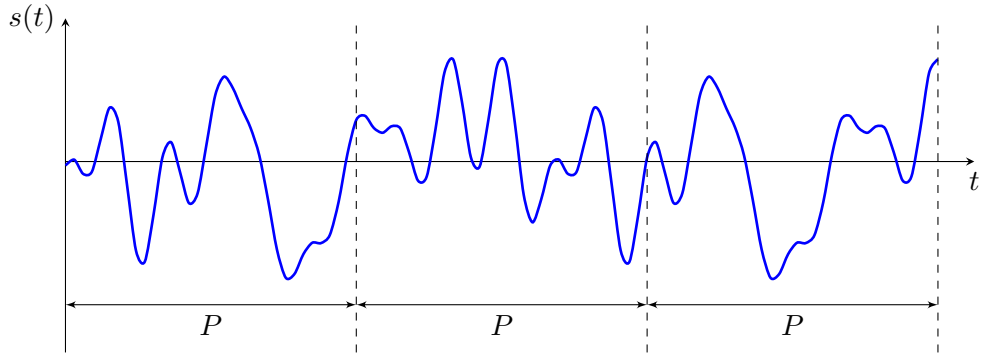


Figure 2.18: Windowing divides a signal into intervals of length P which are assumed to contain a periodic signal.

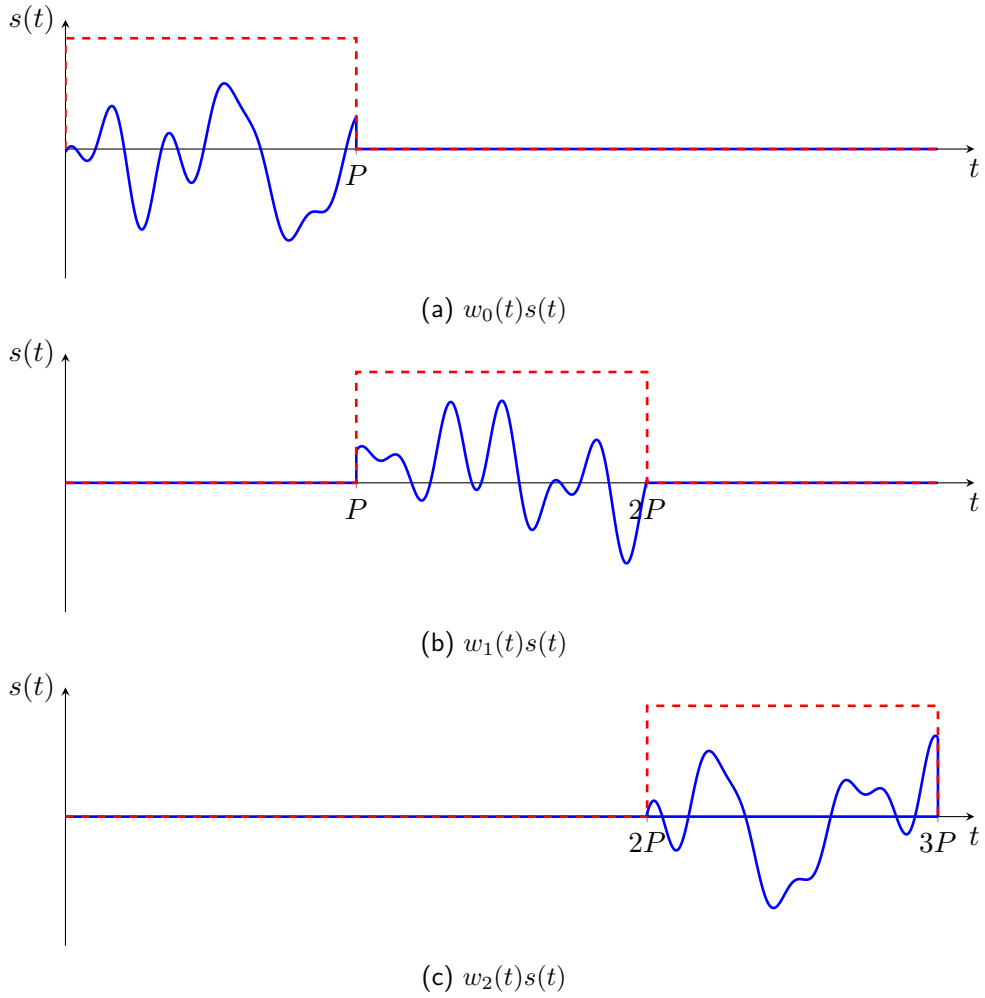
Consider the signal shown in figure 2.18. This is not a periodic signal and would contain too much data for us to calculate the DFT. So we only consider the windows of length P in which we calculate the DFT for the signal in each window. To isolate the signal for this window we multiply $s(t)$ by a **window function** $w(t)$

$$s(t) = w_i(t)s(t).$$

The window function $w(t)$ needs be such that it leaves the signal unchanged in the time interval $t \in [iP, (i+1)P]$ where $i = 0, 1, 2, \ldots$ and reduce to zero elsewhere. The simplest way to do this is to use the rectangular window function

$$w_i(t) = \begin{cases} 1, & t \in [iP, (i+1)P], \\ 0, & \text{elsewhere.} \end{cases} \tag{35}$$

The application of the rectangular window function is shown in figure 2.19. Here we can see that the rectangular window function does isolate the signal over the interval $t \in [0, P]$, however, there is an abrupt drop-off at the boundary of the window which will cause errors when calculating the frequencies.



(a) $w_0(t)s(t)$

(b) $w_1(t)s(t)$

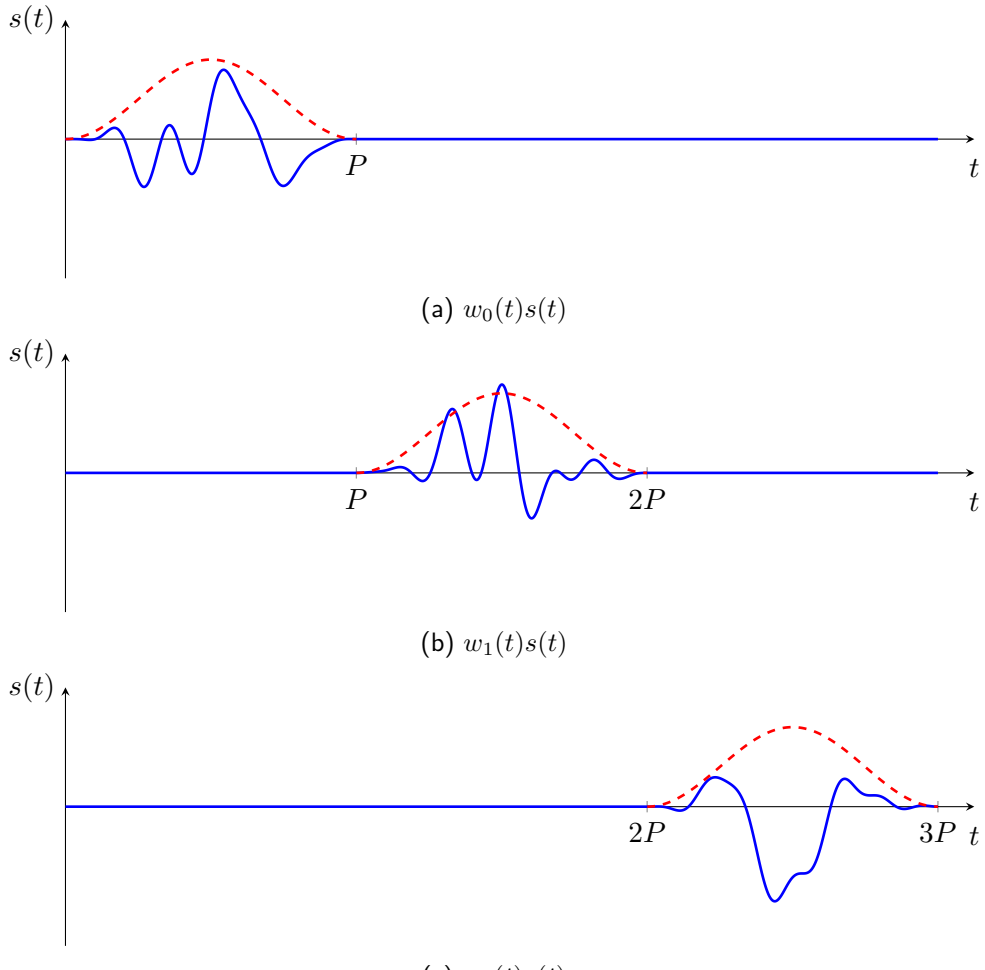
(c) $w_2(t)s(t)$

Figure 2.19: Application of the rectangular window function to a signal.

In practice we use a window function that gradually decreases to zero to avoid the error caused by the rectangular window function. One common window function is **Hanning window function** named after Julius von Hann (1839 – 1921) which is defined by

$$w_i(t) = \begin{cases} \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi t}{P}\right), & t \in [iP, (i+1)P], \\ 0, & \text{elsewhere.} \end{cases} \tag{36}$$

The application of the Hanning window function is shown in figure 2.20. Unlike in figure 2.19, the signal does not have the abrupt drop-off to zero at the boundaries of the window. The amplitude of the signal decreases as it approaches the window boundary but the frequency has not been changed. Since it is the frequency that is the most important factor of a signal and any errors are minimal.



(a) $w_0(t)s(t)$

(b) $w_1(t)s(t)$

(c) $w_2(t)s(t)$

Figure 2.20: Application of the Hanning window function to a signal.

2.6 The Fast Fourier Transform (FFT)

We have seen that the DFT is incredibly useful in calculating the amplitudes in the frequency domain. Unfortunately it is too computationally expensive to be applied to real world signals where the sampling frequency is measured in kHz. If we had N sample values then the computing time required to calculate the DFT is $O(N^2)$ meaning that if the sampling frequency doubles, then the time required will increase by a factor of $2^2 = 4$. The **Fast Fourier Transform (FFT)** is an algorithm that calculates the DFT in $O(N \log N)$ so is much quicker than calculating the DFT. The methods used in the FFT can vary and are outside the scope of this unit but fortunately MATLAB and the NumPy library for Python both have commands to calculate the FFT.

Example 2.8. An analogue signal with period $P = 1$ seconds is modelled by

$$s(t) = 0.2 + 0.6\cos(40\pi t) + 0.9\cos(100\pi t) + 0.3\cos(200\pi t)$$

using a sampling frequency of $f_s = 1000$Hz. Calculate the FFT of the signal and plot the single-sided frequency spectrum.

A plot of the signal and the frequency spectrum is shown in figure 2.21 (the code used to produce this plot can be found in chapter 4 for MATLAB and chapter 5 for Python). From the frequency spectrum we can see that $a_0/2 = 0.2$ and the harmonics with frequencies 20, 50 and 100Hz and amplitudes $A_1 = 0.6$, $A_2 = 0.9$ and $A_3 = 0.3$ respectively dominate which corresponds to the analogue signal from which the digital signal was sampled.

Here we have seen that we have been able to exactly determine the amplitudes of the 3 harmonics from the analogue signal since these are the only ones with non-zero amplitudes. This was because we used a sampling frequency higher than that of the Nyquist frequency (in this case $f_{\text{Nyq}} = 200$Hz) and the signal is exactly periodic over the period $P = 2$ seconds.

□

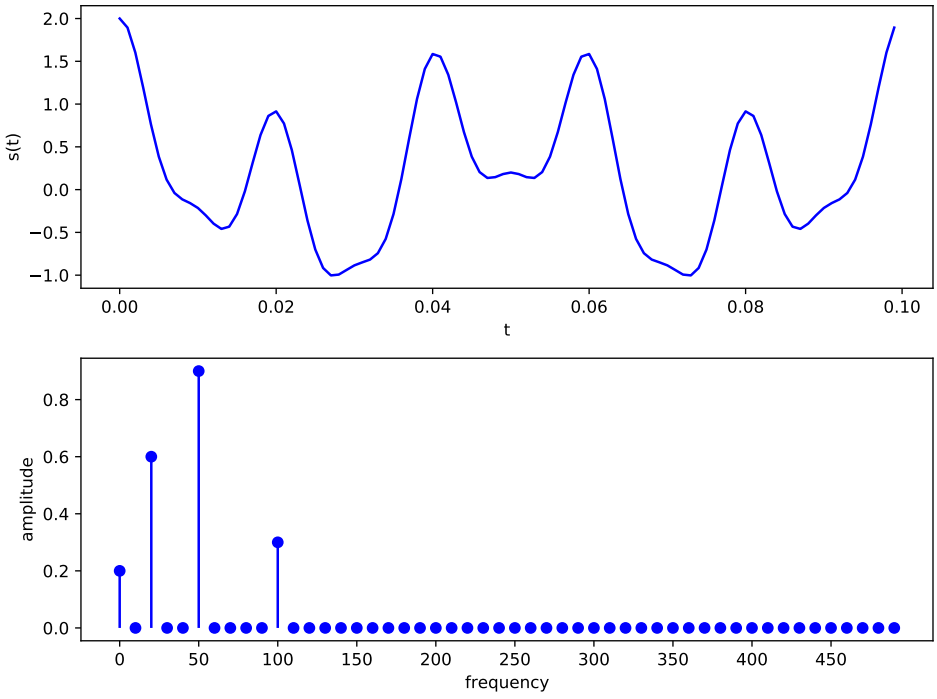


Figure 2.21: The analogue signal from example 2.8 and its frequency spectrum calculated using a FFT.

2.6.1 Cleaning up a noisy signal

The power of spectral analysis can be demonstrated by adding random noise to a signal and performing spectral analysis. A random noise was added to the signal from example 2.8 and the plots of the noisy signal and its frequency spectrum are shown in figure 2.22. The frequency spectrum shows that all of the harmonics have non-zero amplitudes but the same original 3 harmonics as seen in the pure signal still dominate.

We can attempt to remove the noise in the signal by setting all of the harmonics with an amplitude less than a certain threshold to zero. The signal was reconstructed using the Inverse Fast Fourier Transform (IFFT) and a plot of the cleaned up signal and its frequency spectrum is shown in figure 2.23. Note that

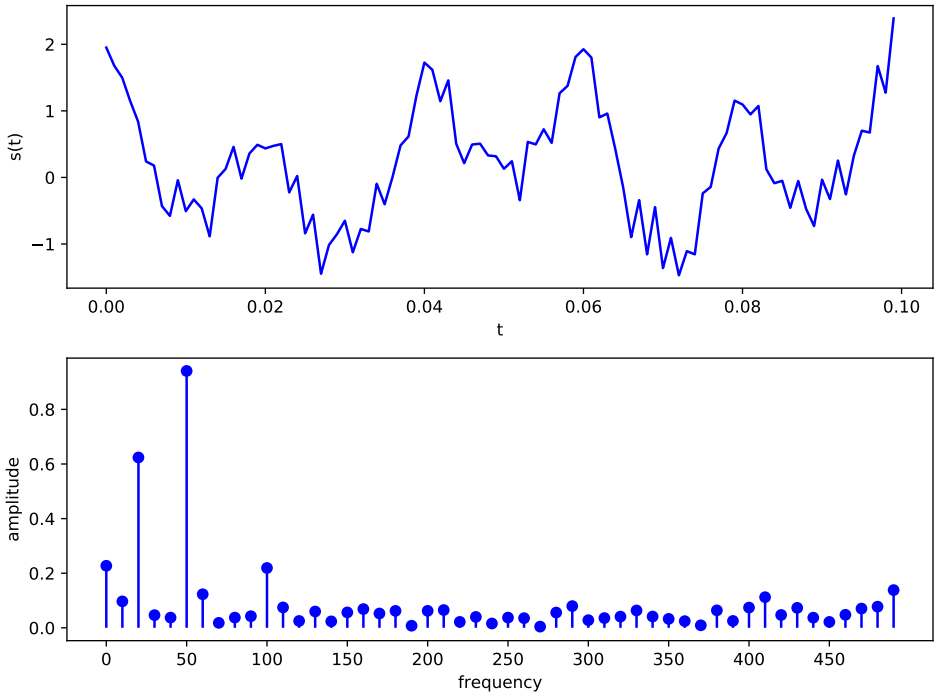


Figure 2.22: Noisy signal and its frequency spectrum.

the cleaned up signal is not exactly the same as the original signal in figure 2.21 since the noise has caused the amplitudes of the three dominant harmonics to change, but it is an significant improvement over the noisy signal.

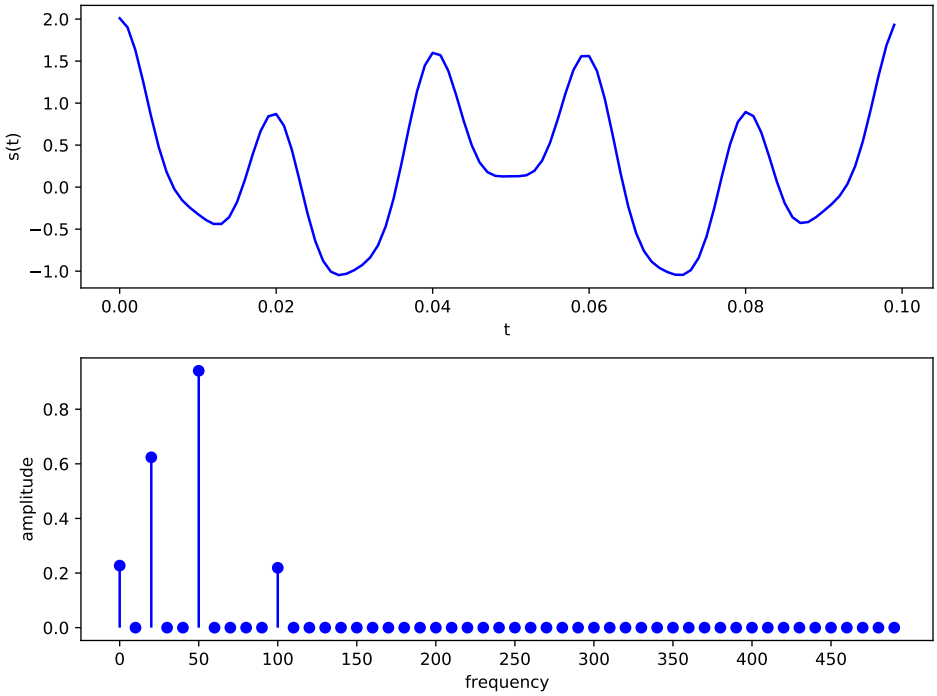


Figure 2.23: Cleaned up signal and its frequency spectrum.

2.7 Exercises

1. An analogue signal is modelled by the following sinusoids
   (a) $s(t) = 6\cos(8t)$;
   (b) $s(t) = 2 + 3\cos(4t) + 5\cos(8t + 2) + \cos(12t)$;
   (c) $s(t) = 1 + 3\cos(220t - 1) + \cos(440t + 2) + 0.5\cos(660t)$;

   For each signal:
   (i) identify the amplitudes, frequencies and phase angles for each harmonic;
   (ii) identify the fundamental frequency and period of the signal;
   (iii) what is the smallest sampling frequency that is needed to digitise $s(t)$;
   (iv) draw the frequency spectrum for $s(t)$.
2. A periodic signal $s(t)$ has the following Fourier series

$$F_n = \sum_{n=1}^{10} \frac{1}{n} \cos\left(\frac{\pi n t}{2}\right).$$

   (a) Find the fundamental frequency and period of the signal;
   (b) find the Nyquist frequency;
   (c) find the mean value of $s(t)$;
   (d) sketch the frequency spectrum of $s(t)$;
   (e) what is the smallest sampling interval needed to digitise $s(t)$?
3. A periodic signal $s(t)$ is defined by

$$s(t) = t, \qquad t \in [-\pi, \pi].$$

   (a) Sketch the signal over the range $t \in [-5\pi, 5\pi]$;
   (b) identify the period and fundamental frequency of the signal;
   (c) calculate a_0, a_n and b_n for the sine-cosine form of the the Fourier series;
   (d) write down the the sine-cosine form of the Fourier series for $s(t)$;
   (e) write down the amplitude-phase form of the Fourier series for $s(t)$;
   (f) write down the exponential Fourier series for $s(t)$;
   (g) sketch the frequency spectrum of this signal using the first 5 harmonics;
   (h) use MATLAB or Python to calculate the Fourier series approximation of this signal using the first 5 harmonics.
4. Prove that Fourier transform in equation (27) is a linear transformation.
5. A periodic signal is sampled using a sampling frequency of $f_s = 12$Hz and is given by

$$s = (3, 0, 0, 3, 0, 0).$$

   (a) Determine the sampling interval and period of the signal;
   (b) calculate the DFT matrix for this signal;
   (c) calculate the DFT of this signal;

(d) sketch the frequency spectrum;

(e) determine the exponential form of the Fourier series that approximates this signal;

(f) use MATLAB or Python to plot the Fourier series and the signal points on the same axes.

6. A periodic signal is sampled using a sampling frequency of $f_s = 5$Hz and is given by

$$s = (1, -1, -2, 0, -1).$$

Determine the amplitude-phase Fourier series that approximates this signal and using MATLAB or Python to plot the signal values and the Fourier series on the same set of axes.

7. A signal was sampled using a sampling frequency of $f_s = 4$Hz and the DFT was calculated resulting in the following values

$$F = (4, 3.74 - 1.54i, -0.74 + 0.36i, -0.74 - 0.36i, 3.74 + 1.54i).$$

(a) Calculate the signal values;

(b) plot the signal values against the Fourier series for this signal.

8. An analogue signal $m(t) = 5\cos(4\pi t)$ with period $P = 1.5$s is sampled N times and transmitted over the internet. After the DAC the signal received is $s(t) = m(t) + e(t)$ where $e(t) = 0.6\cos(32\pi t)$ is the error (or noise) in the signal.

Use MATLAB or Python to do the following:

(a) plot $s(t)$ over the range $t \in [0, 1.5]$ and identify the fundamental frequency of $s(t)$;

(b) calculate the FFT of the signal and plot the one-sided frequency spectrum;

(c) remove the high frequency noise by setting any harmonic with an amplitude less than 1 to zero. Use the IFFT to calculate the cleaned up signal and plot the cleaned up signal against the noisy signal.

The solutions to these exercises are given on page 86.

Chapter 3

Digital Music

Along with speech, music is the most common form of sound in human experience. With the development of digital computers and mathematical algorithms there has been a revolution in the transmission, representation and storage of music. Once a musical signal has been digitised it can be compressed (e.g. mp3), encoded and processed to create a variety of interesting and useful effects. By spectrally analysing music from different instruments it is possible to generate their waveforms from pure sinusoids and thus produce cheap instruments which mimic real ones e.g. digital piano.

3.1 Production of Musical Notes

Musical sounds are produced by vibrations of strings (e.g., guitar, violin, piano), air columns (e.g., trumpet, flute) and membranes (e.g., drum) and by the vibrations of the instruments themselves (e.g. body of an acoustic guitar). An instrument therefore produces a complicated waveform which is the result of the interaction of all its vibration patterns. The sound of an instrument is characterised by its physical parameters (e.g. length, material composition) and this is described by the time and frequency domain representations of its output.

In order to analyse the sound from musical instruments it is necessary to look at how their physical structure influences the notes that they produce. A key concept is that of the wavelength, λ (the Greek character 'lambda') a note of frequency fHz. The wavelength is the distance (in metres) between the centre of neighbouring compressions in a sound wave. It can be easily shown that for a sound speed of vm /s.

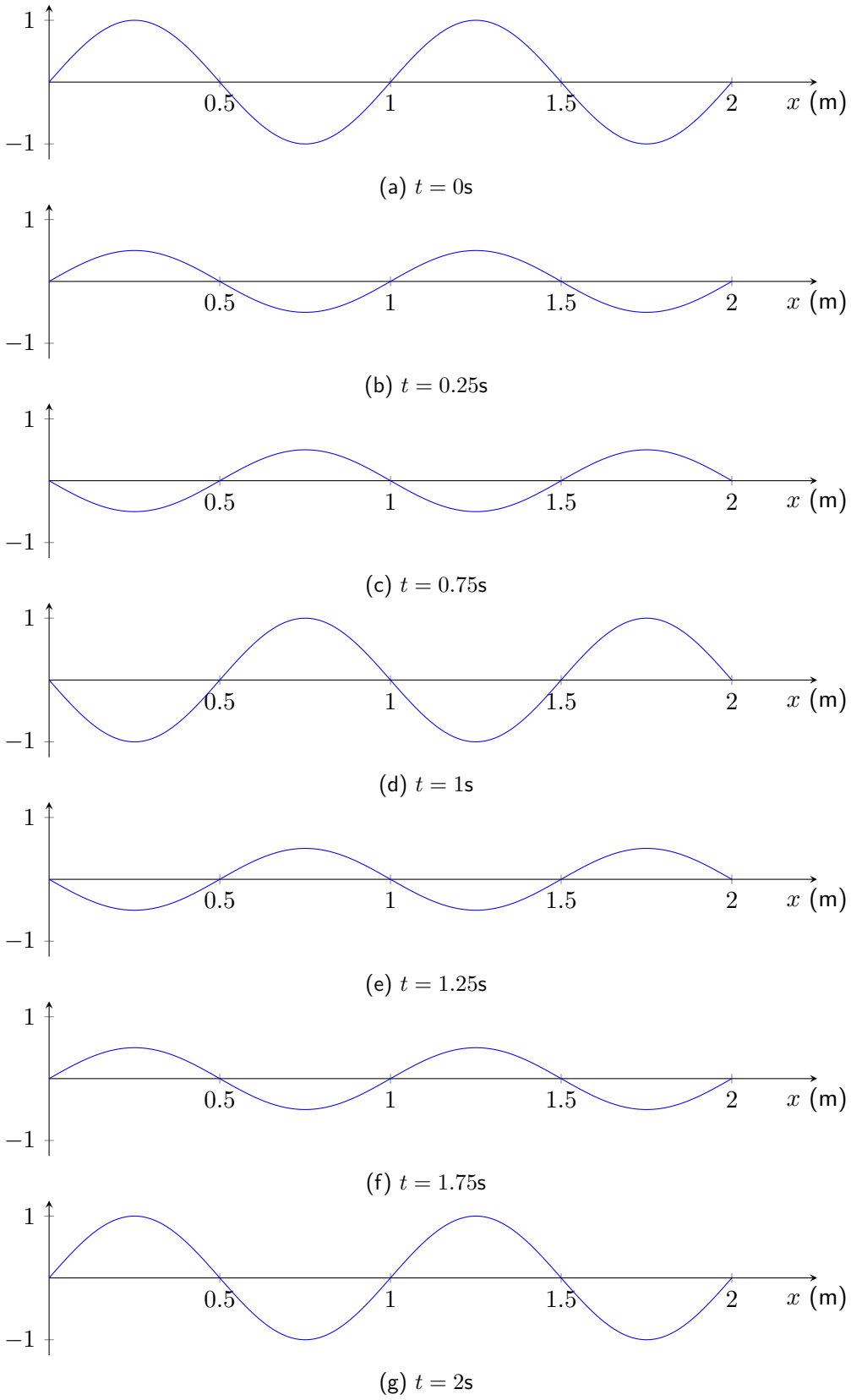
$$v = f\lambda, \tag{37}$$

Hence, by knowing the wavelength and the sound speed the frequency of the sound can be determined. Musical notes are determined by their frequency e.g. the note A over middle C has a frequency of 440Hz (see section 3.4).

3.2 Standing waves

Standing waves are vital to the production of musical sounds. Essentially a standing wave is a waveform in which the pattern of compressions and rarefactions is fixed in time. This is achieved by a balance between transmitted and reflected waves which cancels out any motion. The net effect is to produce a strong, sustained sound at a particular frequency which is transmitted to the air at that frequency.

A standing wave at various times is shown in figure 3.1. At $t = 0$s the wave follow a sinusoidal pattern with amplitude 1m and a wavelength of 1m At $t = 0.25$s the wave is still sinusoidal but its amplitude has decreased to 0.5m but its wavelength is still 1m At $t = 1$s the wave is the same shape as at $t = 0$s but inverted. As we move forward through time we see that the sinusoidal shape remains unchanged but the amplitude varies in time but remains constant in space. Furthermore, its wavelength remains constant which by equation (37) means the frequency must also be constant.



(a) $t = 0$s

(b) $t = 0.25$s

(c) $t = 0.75$s

(d) $t = 1$s

(e) $t = 1.25$s

(f) $t = 1.75$s

(g) $t = 2$s

Figure 3.1: Standing waves at different times on a string fixed at both ends.

The points in space where the wave amplitude remains at zero, i.e., $x = 0.5, 1, 1.5, 2$ in figure 3.1, are called **nodes** and the points where the deviation is greatest, i.e., $x = 0, 0.25, 0.75, 1.25, 1.75, 2$, are called the **antinodes**.

The difference between a standing wave and a travelling waves is that the position of a compression changes with time in a travelling waves but remains stationary (at the antinodes) in a standing wave.

3.3 Sound

Consider a string of a stringed instrument such as an acoustic guitar. When the string is plucked, a standing wave occurs because the string is fixed at both ends (the bridge and the nut). The vibration of the string causes compression waves which, amplified by the body of the guitar, reach our ears where we perceive the compression wave as sound.

The nodes of the standing wave produced by the string occur at half wave length intervals

$$x = \frac{n\lambda}{2}, \qquad n = 0, 1, 2, \ldots,$$

and the antinodes over at odd multiples of quarter wave length intervals

$$x = \frac{n\lambda}{4}, \qquad n = 1, 3, 5, \ldots$$

If we shorten the wavelength of the string, either by applying more tension to the string or decreasing the length between the fixed endpoints, then the frequency will increase by equation (37) and we will perceive a higher pitched sound.

3.3.1 Harmonics

Consider a string of length L. Assuming standing waves are produced when the string vibrates, there are only a few possibilities for the allowable wave lengths (and therefore frequencies) that can be produced. The first four possibilities are shown in figure 3.2.

The longest possible wavelength (and therefore the lowest frequency) is $\lambda = 2L$ (figure 3.2(a)). This is called the **first harmonic**. The next longest wavelength is $\lambda = L$ (figure 3.2(b)) which is called the **second harmonic**. In general, for a string of length L fixed at both ends the possible wavelengths are

$$\lambda = \frac{2L}{n}, \qquad n = 1, 2, \ldots, \tag{38}$$

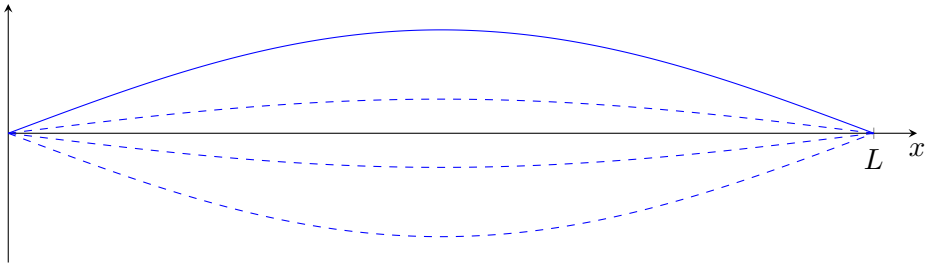
where n is called the **harmonic number**. The frequency corresponding to the n^{th} harmonic is denoted by f_n. From equation (37)

$$f_n = \frac{nv}{2L}, \qquad n = 1, 2, \ldots \tag{39}$$

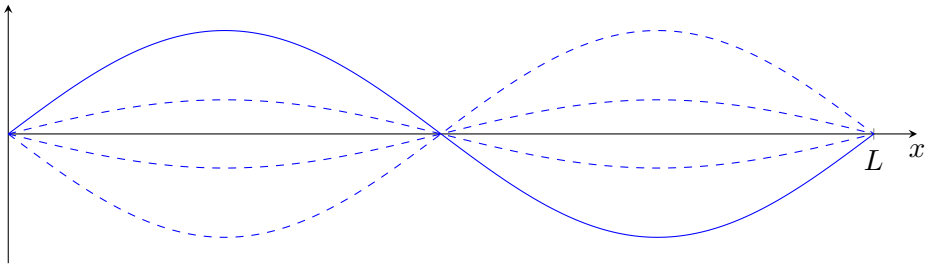
hence

$$f_n = nf_1, \qquad n = 1, 2, \ldots \tag{40}$$

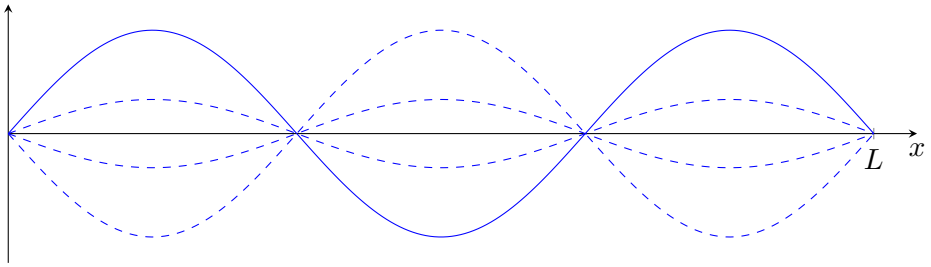
i.e., the frequency of the n^{th} harmonic is n times the fundamental frequency.



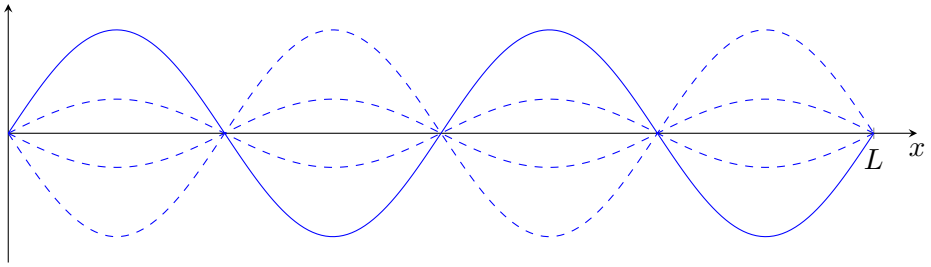
(a) first harmonic, $\lambda = 2L$



(b) second harmonic, $\lambda = L$



(c) third harmonic, $\lambda = 2L/3$



(d) fourth harmonic, $\lambda = 2L/4$

Figure 3.2: The first four harmonics of a standing wave.

3.4 Musical notation

Music is made up of notes which are sounds of a particular frequency and length. Modern western music is based on 12 notes which are denoted by

$$\text{A, A}\sharp\text{, B, C, C}\sharp\text{, D, D}\sharp\text{, E, F, F}\sharp\text{, G, G}\sharp.$$

Consider the diagram of a piano keyboard in figure 3.3. The white keys A, B, C, D, E, F and G are the **naturals**. The black keys A$\sharp$, C$\sharp$, D$\sharp$, F$\sharp$ and G$\sharp$ are the **sharps**. The sharps can also be referred to as **flats** and denoted by $\flat$, i.e., A$\sharp$ is the same note as B$\flat$.

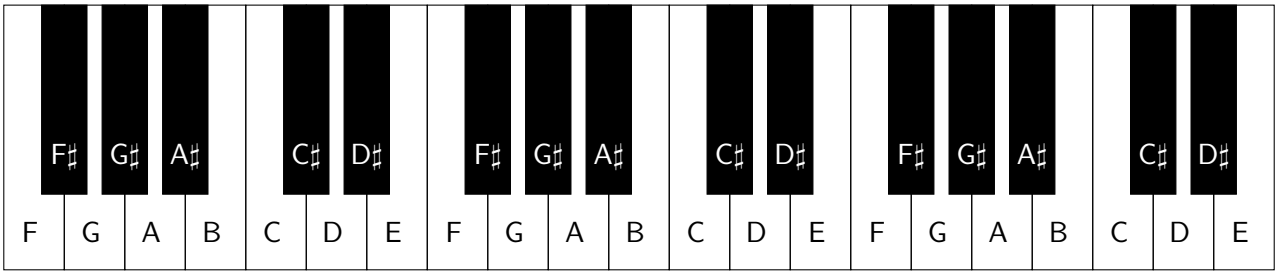


Figure 3.3: The notes on a piano keyboard.

These 12 notes increase in frequency and span an **octave**. So the next note in the sequence is another A note but this note will have double the frequency of the first A note in our sequence.

The frequencies assigned to each note depends on the musical system being used. In modern Western music the notes are space so that the ratio of neighbouring notes frequencies is constant. Let this constant be K, then

$$K = \frac{\text{frequency of A}\sharp}{\text{frequency of A}} = \frac{\text{frequency of B}}{\text{frequency of A}\sharp} = \frac{\text{frequency of C}}{\text{frequency of B}} = \cdots \tag{41}$$

The frequency spacing between neighbouring notes are called **semitones**, these are not constant frequency intervals. So to find the frequency of a note a semitone away from the base note we simply multiply the frequency of the base note by K. To find the frequency of a note 2 semitones above the base note we multiply its frequency by K^2 and so on.

Consider a note with frequency f and consider the same note in the next octave above with frequency $2f$. Our sequence of notes are 12 semitones apart so

$$2f = K^{12} f,$$

therefore

$$K = 2^{1/12} \approx 1.0595 \tag{42}$$

Given this value it is possible to calculate the frequencies of all notes. Since the frequency of A over middle C is 440Hz then

$$\begin{aligned}
\text{frequency of A}\sharp &= 1.05946 \times 440 = 466.16\text{Hz},\\
\text{frequency of B} &= 1.05946^2 \times 440 = 493.88\text{Hz},\\
\text{frequency of C} &= 1.05946^3 \times 440 = 523.25\text{Hz},\\
&\vdots
\end{aligned}$$

A modern piano keyboard has spans 7 octaves which begin at each C note. The leftmost keys (the lowest notes) are the A, A$\sharp$ and B notes which are in the octave below the first full octave so are referred to as being in octave 0 and denoted by A_0, $\text{A}_0\sharp$ and B_0. The remaining keys consist of the 7 octaves each

containing 7 naturals and 5 sharps and the rightmost key is the single note C_8 from the 8th octave. So in total a piano keyboard has 88 keys.

A modern reference value for A_4 (also known as 'A over middle C' where middle C is C_4) is 440Hz. Counting from the left, A_4 is the 49th key on the keyboard so the frequency of the n^{th} key, $f(n)$, can be calculated using

$$f(n) = 2^{(n-49)/12} \times 440\text{Hz}. \tag{43}$$

The frequencies for all 88 keys on a piano are shown in table 3.1. These can be used to help us produce digital sounds that can replicate a piano.

Table 3.1: The frequencies of the notes on a piano keyboard

Note \ Octave	0	1	2	3	4	5	6	7	8
C		32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C♯		34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	
D		36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	
D♯		38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	
E		41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	
F		43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	
F♯		46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	
G		49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	
G♯		51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	
A	27.50	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	
A♯	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	

3.5 Digital music

So we know that to create a sound we can generate a signal of a given frequency and amplitude. For example, if we wanted to produce a sound of the note A_4 that lasts for 2 seconds using a sampling frequency $f_s = 44100$Hz (this is the standard sampling frequency used for music) then we calculate the sampling interval $T = 1/f_s$ and the sampling times $t = (0, T, 2T, \ldots)$ and sample the sinusoid

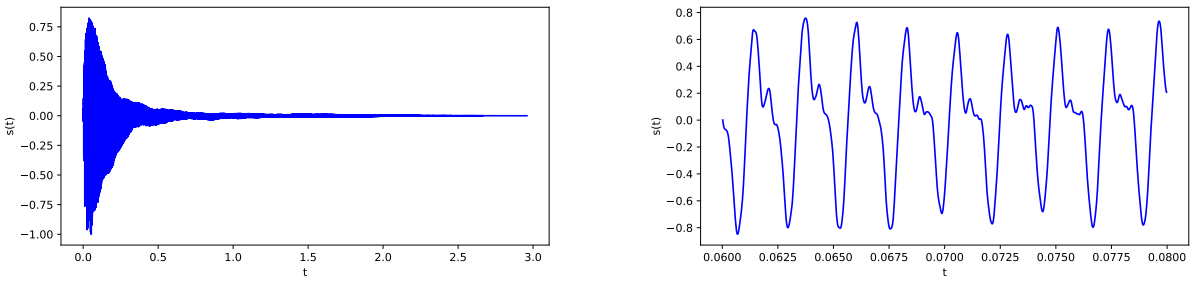
$$s(t) = \cos(2\pi f_0 t).$$

If we do this the result is not particularly pleasant to listen to (listen to the file `A4note.wav` from Moodle). This is because musical notes are a combination of harmonics whose amplitudes change over time and simply a pure sinusoid. So in order to computer generated instrumental sounds we need to look at the frequency spectra of waveforms generated using a real instrument.

3.5.1 Analysing a note played on a real instrument

An A_4 note played on a grand piano was recorded and loaded into a Python (this can also be done using MATLAB) program and the plot of the signal is shown in figure 3.4. We can see that the amplitude of the signal has an initial peak at around $t = 0.1$s before decreasing at around $t = 0.4$s and then continues to ring out until the end of the recording. The plot of the signal in the range $t \in [0.06, 0.08]$s shown in figure 3.4(b) shows that in this region the signal is periodic.

An FFT was calculated for the signal and the plot of the frequency spectrum is shown in figure 3.5. Here we can see that there are clear peaks which correspond to the frequencies 440, 880, 1320, 1760, 2210 and 2660Hz which means that the note consists of the first 6 harmonics where the first 2 harmonics dominate.



(a) $t \in [0, 3]$s (b) $t \in [0.06, 0.08]$s

Figure 3.4: The audio signal of the note A_4 played on a grand piano.

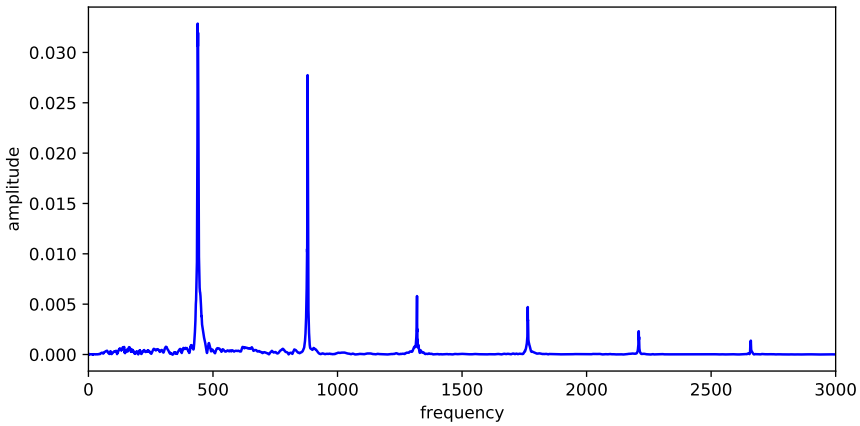


Figure 3.5: Frequency spectrum for a A_4 note played on a grand piano.

3.5.2 Creating a digital note

We calculate a Fourier series using the frequencies and amplitudes of the first 6 harmonics which would give a signal which has the same pitch as the recorded signal. However, we saw in figure 3.4 that the amplitude changes over time so the amplitudes of the harmonics will change over time. The frequency spectra for different time intervals over the length of the signal are shown in figure 3.6. Note that in the first interval $t \in [0, 0.5]$s the first two harmonics dominate with the first harmonic having the higher amplitude. In the second interval $t \in [0.5, 1]$s the second harmonic now dominate and the amplitude of the first harmonic has decreased relative to the second harmonic (note that the vertical scale has changed) and the amplitude of the third harmonic has increased.

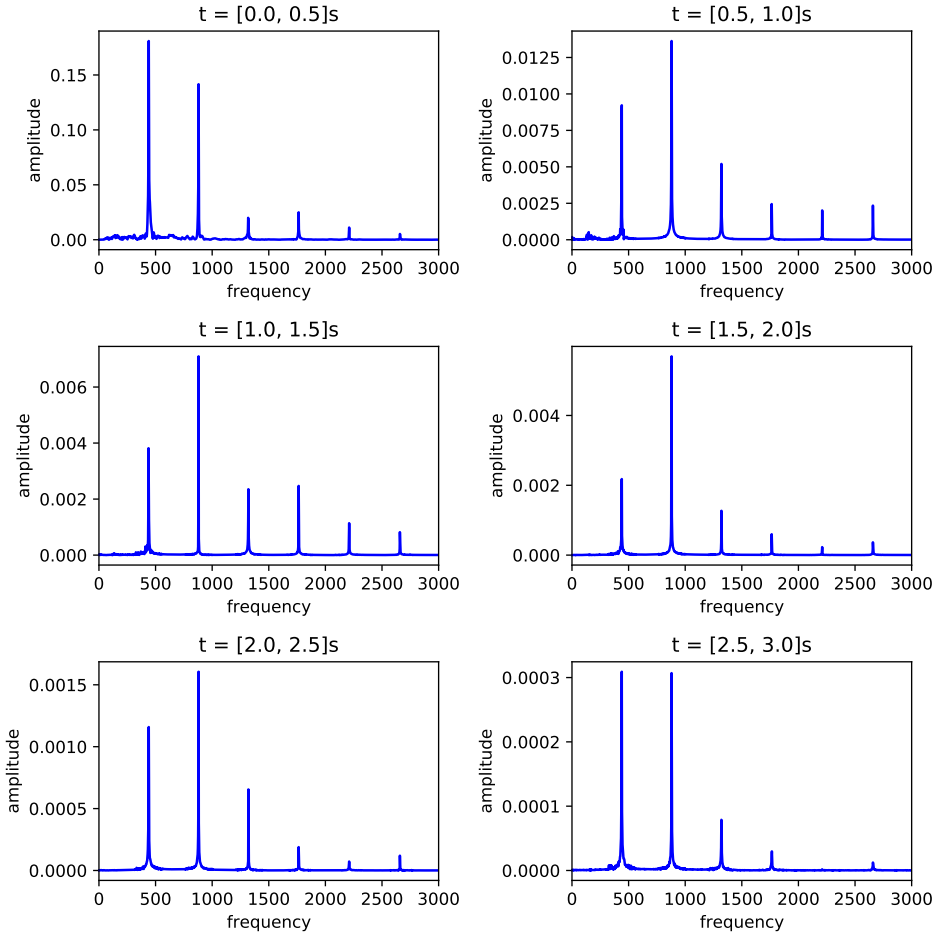


Figure 3.6: Frequency spectra of the A_4 note for different intervals of time.

So to replicate this note we need information on how the amplitudes in the frequency spectrum change over time. We do this by calculating the FFT for time intervals of the signal known as **frames** and examine the changes of the amplitudes over time. The signal was split up into frames each of duration $w = 0.02$s and the amplitudes of the harmonics have been plotted in figure 3.7.

Now we now how the amplitudes change over the duration of the signal we can use the Fourier series using the first 6 harmonics with the amplitudes modelled on the recorded note. A plot of the digital note is shown in figure 3.8 and the note can be heard by playing the file `A4piano_digital.wav` on Moodle. We can repeat this for any note played on different instruments to build a library of signals that can then be used replicate real instruments.

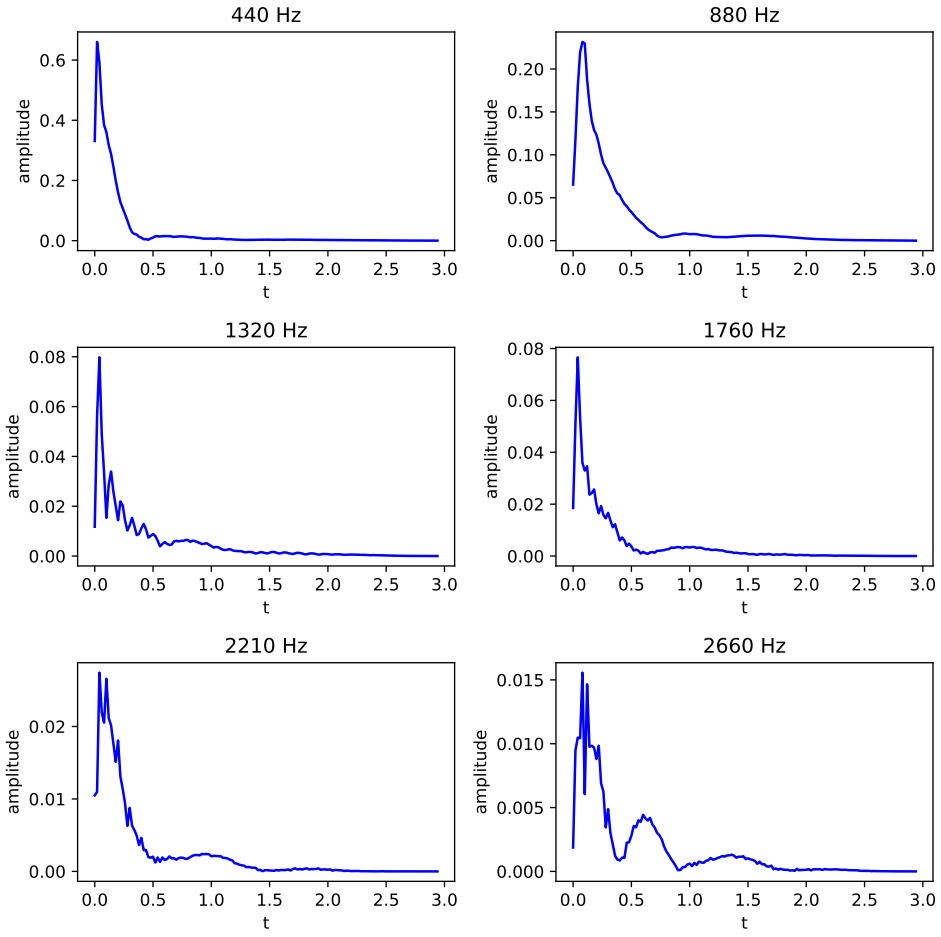


Figure 3.7: Plots of the amplitudes of the harmonics over time.

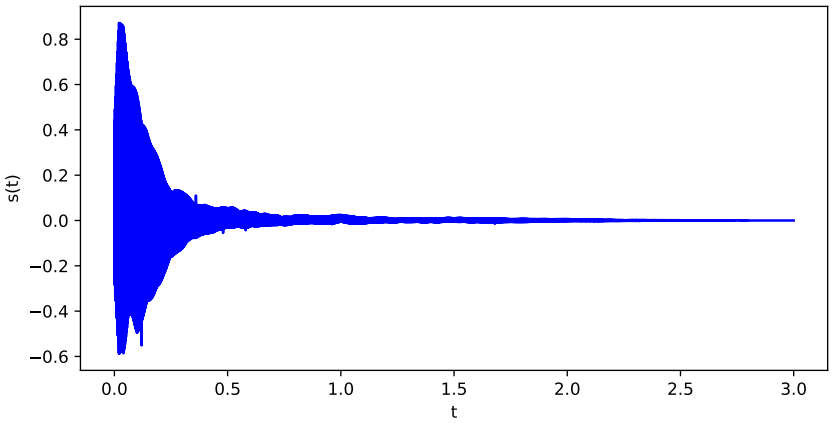


Figure 3.8: A plot of the digital note based on A_4 played on a grand piano.

3.6 Exercises

Answer the following questions using MATLAB and/or Python.

1. Write a program that generates an audio signal which has the same pitch as an G_3 note played on a piano and plays the audio signal. Your signal should be of length $L = 1$s and has sampling frequency $f_s = 44100$Hz.
2. Download the wav file `piano_note.wav` from the Moodle area for this unit which contains a recording of a single note played on a piano. Write a program that reads in the audio signal and plots the signal.
3. Calculate the FFT of the signal and plot its frequency spectrum.
4. Determine the frequencies of the harmonics and identify which note is being played.
5. Use a `for` loop to loop through each harmonic and calculate the amplitudes over time using frames of $w = 0.05$s in length. Plot the amplitudes against time for each harmonic.
6. Calculate a digital note based on the recorded note which has a length of $L = 4$s and using sampling frequency $f_s = 44100$Hz. Plot the signal and play the audio.

The solutions to these exercises are given on page 99.

Chapter 4

MATLAB

The various MATLAB commands used to compute the examples in these notes are presented in this chapter. MATLAB comes with a signal processing toolbox which is specifically designed to analyse signals, however, much of the underlying mathematics is hidden from the user. Therefore, these notes do not make use of the signal processing toolbox in order to demonstrate what is going on "under the hood" (with the exception of calculating the FFT and IFFT).

4.1 Sampling signals

4.1.1 Sampling an analogue signal

The sampling of an analogue signal defined using mathematical functions is a simple matter of calculating an array that contains the sampling times and then use that to calculate the signal. To do this we need the length, L, of the signal over which we are sampling and the sampling frequency, f_s, and from these we calculate the sampling interval and number of sample points using equations (2) and (4) which are

$$T = \frac{1}{f_s},$$
$$N = Lf_s.$$

The sampling times are then calculated by multiplying an array of numbers $(0, 1, 2, \ldots, N-1)$ by T using

```
t = (0 : N - 1) * T;
```

It is also useful to define a function that describes the analogue signal in case we need to re-sample. The MATLAB code in listing 4.1 samples the signal $s(t) = \cos(4t)$ of length $L = 2$s (taken from example 1.1 on page 6) using a sampling frequency of $f_s = 4$Hz.

Listing 4.1: Sampling an analogue signal in MATLAB.

```
% Define analogue signal
analogue_signal = @(t) cos(4 * t);

% Define signal parameters
L = 2;
fs = 4;
T = 1 / fs;
N = L * fs;

% Sample the analogue signal
t = (0 : N - 1) * T;
s = analogue_signal(t);
fprintf(['t = ', sprintf(repmat('%1.4f, ', 1, length(t)), t), ...
    '\ns = ', sprintf(repmat('%1.4f, ', 1, length(s)), s)])
```

```
t = 0.0000, 0.2500, 0.5000, 0.7500, 1.0000, 1.2500, 1.5000, 1.7500,
s = 1.0000, 0.5403, -0.4161, -0.9900, -0.6536, 0.2837, 0.9602, 0.7539,
```

4.1.2 Plotting analogue and digital signals

To plot an analogue signal we can simply use the standard `plot` command

```
plot(t, s, 'b')
```

which plots the signal array `s` against the time array `t` using a blue line. In order to represent the continuous nature of an analogue signal we need to make sure that we sample the `analogue_signal` function using lots of sample points (200 sampling points is common). This is done using the `linspace` command.

```
ta = linspace(0, L, 200);
```

which creates a 1×200 array of equally spaced values from 0 to L (which is the length of the signal).

To plot a digital signal we can use the `stairs` command

```
stairs([t, t(N) + t], [s, s(N)], 'r')
```

which plots a horizontal line from each sample point to the end of the sample interval representing the discontinuities in the digital signal. Note that the `t` and `s` arrays have been extended by appending `L` and `s(N)` (the last element in `s`) respectively so that the last sampling interval is included in the plot.

For example, the code in listing 4.2 plots the digital and analogue signal from listing 4.1 and the resulting plot is shown in figure 4.1.

Listing 4.2: Plotting signals in MATLAB

```
% Define analogue signal for plotting
ta = linspace(0, L, 200);
sa = analogue_signal(ta);

% Plot the analogue and digital signal
plot(ta, sa, 'b')
hold on
stairs([ t, t(N) + T ], [ s, s(N) ], 'r')
hold off
pbaspect([ 2, 1, 1 ])
axis padded
legend('analogue signal', 'digital signal', 'location', 'southwest')
xticks(t)
xlabel('time')
ylabel('s(t)')
```

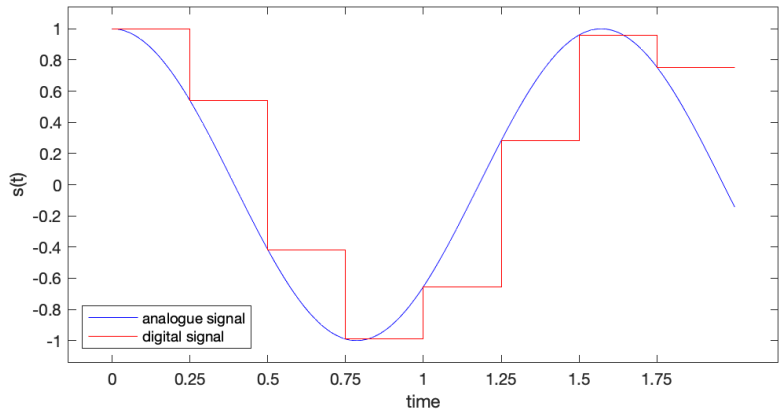


Figure 4.1: MATLAB plot of an analogue and digital signal produced using the MATLAB code in listing 4.2.

4.1.3 Quantising a sampled signal

To quantise a sampled signal we need the lower and upper bound of the signal, $s_{\min}$ and $s_{\max}$, and the bit depth M. Then it is simply a matter of calculating Δq and q using equation (5) and **??** which are

$$\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1},$$
$$q = \left\lfloor \frac{s - s_{\min}}{\Delta q} + \frac{1}{2} \right\rfloor.$$

To calculate the bit string for the quantised signal we can use the `reshape` and `dec2bin` commands

```
bitstring = reshape(dec2bin(q)', 1, N * M)
```

which converts the array `q` to an array of binary numbers before reshaping it into a single character string. The MATLAB code in listing 4.3 calculates the quantisation of the signal contained in the array `s` from listing 4.1 using lower and upper bounds $s_{\min} = -1$ and $s_{\max} = 1$ and a bit depth of $M = 2$ (taken from example 1.2 on page 8).

Listing 4.3: Quantising a sampled signal in MATLAB

```
% Define signal parameters
smin = -1;
smax = 1;
M = 2;

% Calculate quantisation values
Deltaq = (smax - smin) / (2 ^ M - 1);
q = floor((s - smin) / Deltaq + 1 / 2);
fprintf(['q = ', sprintf(repmat('%1i, ', 1, length(q)), q)])

% Convert q to a bit string
bitstring = reshape(dec2bin(q)', 1, N * M);
fprintf('bitstring = %s', bitstring)
```

```
q = 3, 2, 1, 0, 1, 2, 3, 3,
bitstring = 1110010001101111
```

4.1.4 Reconstructing a quantised signal

To reconstruct a quantised signal from a bit string we need to know the sampling frequency f_s, the lower and upper bound of the signal, $s_{\min}$ and $s_{\max}$, and the bit depth M. We first need to convert the bit string into a array of decimal numbers for the quantised values by looping through each sample point and

using the `bin2dec` command to convert a binary number to an integer. Then we reconstruct the signal s using equation (7) which is

$$s = q\Delta q - 1$$

The MATLAB code in listing 4.4 reconstructs the quantised signal calculated in listing 4.3 with the sampling frequency of $f_s = 4$Hz, lower and upper bounds of $s_{\min} = -1$ and $s_{\max} = 1$ and bit depth of $M = 2$ (taken from example 1.3 on page 10).

Listing 4.4: Reconstructing a digital signal from a bit string in MATLAB.

```
% Convert bitstring to decimal array
q = zeros(1, N);
for i = 1 : N
    q(i) = bin2dec(bitstring(1 + (i - 1) * M : i * M));
end
fprintf(['q = ', sprintf(repmat('%1i, ', 1, length(q)), q)])

% Reconstruct signal
Deltaq = 2 / (2 ^ M - 1);
s = q * Deltaq - 1;
t = (0 : N - 1) * T;
fprintf(['t = ', sprintf(repmat('%1.4f, ', 1, length(t)), t), ...
    '\ns = ', sprintf(repmat('%1.4f, ', 1, length(s)), s)])
```

```
q = 3, 2, 1, 0, 1, 2, 3, 3,
t = 0.0000, 0.2500, 0.5000, 0.7500, 1.0000, 1.2500, 1.5000, 1.7500,
s = 1.0000, 0.3333, -0.3333, -1.0000, -0.3333, 0.3333, 1.0000, 1.0000,
```

4.2 Spectral analysis

4.2.1 Calculating a Discrete Fourier Transform (DFT)

To calculate the DFT of a signal we need to use nested `for` loops to calculate the elements of the DFT matrix W using

$$[W]_{nk} = (e^{-i2\pi/N})^{(n-1)(k-1)},$$

and then apply the DFT using $F = W \cdot s$ before calculating the complex coefficients using

$$c_i = F_i/N, \qquad i = 0, 1, \ldots, n,$$

where $n = \lfloor (N-1)/2 \rfloor$. The Fourier series approximation of the signal can then be calculated using the values of c_i and f_0 using

$$s(t) = \sum_{i=0}^{n} c_i e^{i2\pi n f_0 t} + c_i^* e^{-i2\pi n f_0 t}.$$

The MATLAB code in listing 4.5 calculates the DFT of the sampled signal $s(t) = [2, 0, -1, 1, 0]$ (from example 2.6 on page 31) and plots the Fourier series approximation which is shown in **??**.

Listing 4.5: Calculating a DFT in MATLAB

```
% Define signal sampled signal
s = [2, 0, -1, 1, 0];
fs = 10;
P = 0.5;

% Calculate DFT matrix
N = length(s);
W = zeros(N);
for n = 1 : N
    for k = 1 : N
        W(n, k) = exp(-1i * 2 * pi / N) ^ ((n - 1) * (k - 1));
    end
end

% Calculate DFT
F = W * s';
c = F / N;
fprintf(['F = ',sprintf(repmat('%1.0f + %1.4fi, ', 1, length(F)),real(F),imag(F))])
```

```
F = 2 + 2.0000i, 2 + 2.0000i, 2 + 0.0000i, 1 + -1.9021i, 2 + -1.1756i,
```

4.2.2 Calculating the Fourier series from a DFT

Once the DFT has been calculated we can calculate the Fourier series approximation of the sampled signal. First we need to calculate the values of the complex coefficients c_i using equation (29) which is

$$c_i = \frac{F_i}{N},$$

where N is the length of the F array. Then the Fourier series approximation is calculated using equation (22) which is

$$s(t) = \sum_{n=0}^{m} c_i e^{i2\pi n f_0 t} + c_i^* e^{-i2\pi n f_0 t}.$$

where f_0 is the fundamental frequency and $m = \lfloor (N-1)/2 \rfloor$.

The MATLAB code in listing 4.6 calculates the Fourier series approximation using the DFT values from listing 4.5 (taken from example 2.6 on page 31) and plots the Fourier series approximation and the sampled signal which is shown in figure 4.2.

Listing 4.6: Plotting a Fourier series in MATLAB.

```
% Calculate signal parameters
f0 = 1 / P;
T = 1 / fs;
t = (0 : N - 1) * T;

% Calculate the Fourier series approximation
nmax = floor((N - 1) / 2);
ta = linspace(0, P, 100);
sa = c(1);
for n = 1 : nmax
    sa = sa + c(n+1) * exp(1i * 2 * pi * n * f0 * ta) ...
        + conj(c(n+1)) * exp(-1i * 2 * pi * n * f0 * ta);
end

% Plot Fourier series approximation and the sampled signal
plot(ta, sa, 'b-')
hold on
plot(t, s, 'ro', 'MarkerFaceColor', 'r')
hold off
pbaspect([2,1,1])
axis padded
legend('Fourier series', 'sampled signal', 'location', 'southwest')
xlabel('t')
ylabel('s(t)')
```

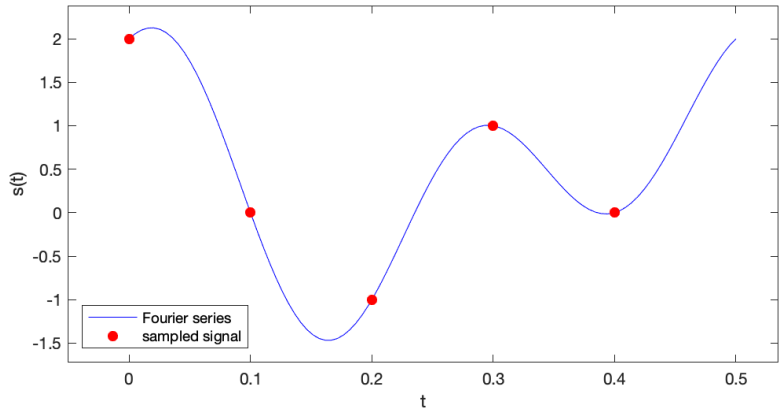


Figure 4.2: MATLAB plot of the Fourier series approximation of the sampled signal from listing 4.6.

4.2.3 Plotting a frequency spectrum

A frequency spectrum can be plotted using the `stem` command

```
stem(f, A, 'b', 'MarkerFaceColor', 'b')
```

which plots a single vertical line for each harmonic. The frequencies for each harmonic are calculated by multiplying the fundamental frequency, f_0, by the array of numbers $(0, 1, 2, \ldots, n_{\max})$ where $n_{\max}$ is the maximum harmonic using

```
f = (0 : nmax) * f0;
```

The DC and amplitudes of the harmonics are calculated from the complex coefficients, c_i, using

$$\frac{a_0}{2} = |c_0|,$$
$$A_i = 2|c_i|, \qquad i = 1, 2, \ldots, n_{\max}.$$

The MATLAB code in listing 4.7 calculate the amplitudes of the harmonics given the values of the complex coefficients and plots the frequency spectrum for the signal from example 2.6 (page 31) which is shown in figure 4.3. Note that it is common to use a standard `plot` command to plot the frequency spectrum when dealing with a large number of harmonics.

Listing 4.7: Plotting a frequency spectrum in MATLAB.

```
% Calculate the frequencies and amplitudes of the harmonics
f = (0 : nmax) * f0;
A = abs([c(1) ; 2 * c(2:nmax+1)]);
fprintf(['f = ', sprintf(repmat('%1.4f, ', 1, length(f)), f), ...
    '\nA = ', sprintf(repmat('%1.4f, ', 1, length(A)), A)])

% Plot signal and amplitude spectrum
stem(f, A, 'b', 'MarkerFaceColor', 'b')
pbaspect([2,1,1])
axis padded
xlabel('frequency')
ylabel('amplitude')
```

```
f = 0.0000, 2.0000, 4.0000,
A = 0.4000, 0.9280, 1.1040,
```

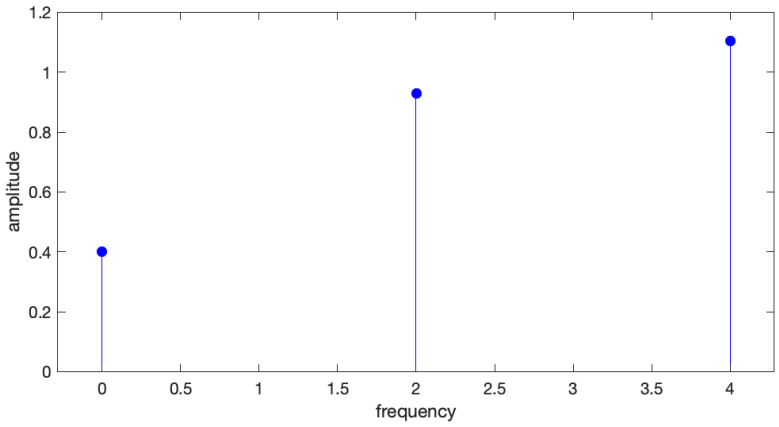


Figure 4.3: MATLAB plot of the frequency spectrum from listing 4.7.

4.2.4 Calculating a Inverse Discrete Fourier Transform (IDFT)

To calculate the IDFT we require the F_i values from the DFT of a signal. The IDFT matrix is calculated using nested `for` loop similar to what was used to calculate the DFT where the elements of the IDFT matrix are

$$[W^{-1}]_{nk} = \frac{(e^{i2\pi/N})^{(n-1)(k-1)}}{N},$$

and the signal values are calculated using $s = W^{-1} \cdot s$. The MATLAB code in listing 4.8 calculates the signal values from the DFT values from listing 4.5 (taken from example 2.7 on page 33).

Listing 4.8: Calculating an IDFT in MATLAB.

```
% Define F array
F = [2, 2 + 1.1756i, 2 - 1.9021i, 2 + 1.9021i, 2-1.1756i];

% Calculate IDFT matrix
N = length(F);
W = zeros(N);
for n = 1 : N
    for k = 1 : N
        W(n, k) = exp(1i * 2 * pi / N) ^ ((n - 1) * (k - 1)) / N;
    end
end

% Calculate IDFT
s = W * F';
fprintf(['s = ', sprintf(repmat('%1.4f, ', 1, length(s)), s)])
```

```
s = 2.0000, 0.0000, 1.0000, -1.0000, -0.0000,
```

4.2.5 Calculating the Fast Fourier Transform (FFT)

MATLAB comes with a FFT function `fft` which is implemented using

```
F = fft(s);
```

The MATLAB code in listing 4.9 calculates the FFT of the sampled signal from example 2.6 on page 31

$$s(t) = (2, 0, -1, 1, 0),$$

and plots the frequency spectrum which is shown in figure 4.4.

Listing 4.9: Calculating a FFT in MATLAB.

```
% Define sampled signal
s = [2, 0, -1, 1, 0];
P = 0.5;

% Calculate signal parameters
f0 = 1 / P;
N = length(s);
nmax = floor((N - 1) / 2);

% Calculate FFT
F = fft(s);
c = F / N;
fprintf(['F = ',sprintf(repmat('%1.0f + %1.4fi, ', 1,length(F)),real(F),imag(F))])

% Calculate the frequencies and amplitudes of the harmonics
f = (0 : nmax) * f0;
A = abs([c(1), 2 * c(2:nmax+1)]);
fprintf(['f = ', sprintf(repmat('%1.4f, ', 1, length(f)), f), ...
    '\nA = ', sprintf(repmat('%1.4f, ', 1, length(A)), A)])

% Plot signal and frequency spectrum
stem(f, A, 'b', 'MarkerFaceColor', 'b')
pbaspect([2,1,1])
axis padded
xticks(f)
xlabel('frequency')
ylabel('amplitude')
```

```
F = 2 + 2.0000i, 2 + 2.0000i, 2 + 0.0000i, 1 + -1.9021i, 2 + -1.1756i,
f = 0.0000, 2.0000, 4.0000,
A = 0.4000, 0.9280, 1.1040,
```

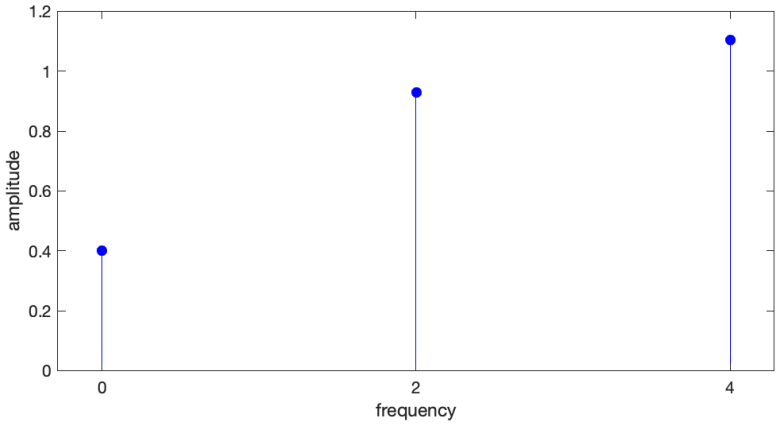


Figure 4.4: MATLAB plot of the signal and frequency spectrum from example 2.8.

4.2.6 Calculating the Inverse Fast Fourier Transform (IFFT)

The MATLAB command to calculate the IFFT is `ifft` which is implemented using

```
s = ifft(F);
```

The MATLAB code in listing 4.10 calculates the signal values from the FFT of the signal from listing 4.9 (taken from example 2.7 on page 33)

Listing 4.10: Calculating an IFFT in MATLAB.

```
% Define F array
F = [2, 2 + 1.1756i, 2 - 1.9021i, 2 + 1.9021i, 2-1.1756i];

% Calculate IFFT
s = ifft(F);
fprintf(['s = ', sprintf(repmat('%1.4f, ', 1, length(s)), s)])
```

```
s = 2.0000, -0.0000, -1.0000, 1.0000, 0.0000,
```

4.3 Reading, writing and playing audio files

4.3.1 Reading audio files

To read an audio file we can use the `audioread` command

```
[s, fs] = audioread(filename)
```

which reads the audio file specified by `filename` and stores the signal values in the $m \times n$ array `s` where m is the number of sample points and n is the number of channels (e.g., 1 channel is mono sound, 2 channel is stereo sound etc.) and the sampling frequency in `fs`. Supported audio formats include `.wav`, `.ogg`, `.flac` and `.mp3`. For example, the code in listing 4.11 reads in an audio file containing 8 seconds from Handel's Hallelujah Chorus, calculates the signal parameters and plots the signal which is shown in figure 4.5.

Listing 4.11: Reading an audio file in MATLAB.

```
% Read audio file
[s, fs] = audioread('handel.wav');

% Calculate signal parameters
[N, nchannels] = size(s);
T = 1 / fs;
L = N * T;
t = (0 : N - 1) * T;

% Plot audio signal
plot(t, s, 'b')

pbaspect([2, 1, 1])
axis padded
xlabel('time')
ylabel('s(t)')
```

```
N = 65536
nchannels = 1
fs = 8192Hz
T = 1.2207e-04 s
L = 8.0000 s
```

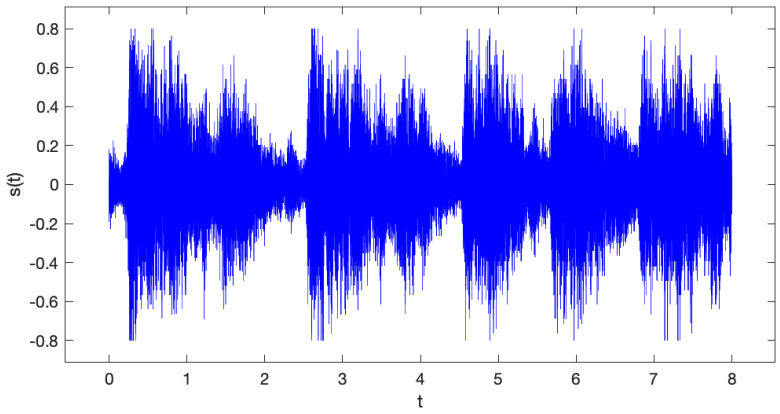


Figure 4.5: MATLAB plot of the first 8 seconds of Handel's Hallelujah Chorus.

4.3.2 Writing audio files

To write an audio file we can use the `audiowrite` command

```
audiowrite(filename, s, fs)
```

which writes the signal contained in the array `s` to the file specified by `filename` and the sampling frequency `fs` (this is necessary so that the audio is played at the correct tempo). The audio format used is determined by the file extension in `filename`. The MATLAB code in listing 4.12 creates a signal of 2 seconds of random noise and writes it to the wav file `random_noise.wav`.

Listing 4.12: Writing an audio file in MATLAB.

```
% Create signal (random noise)
L = 2;
fs = 44100;
N = L * fs;
s = rand(1, N);

% Write audio file
audiowrite('random_noise.wav', s, fs)
```

4.3.3 Playing audio files

To play an audio file we can use the `sound` command

```
sound(s, fs)
```

which plays the signal contained in the array `s` at a tempo determined by the sampling frequency `fs` (e.g., using `2 * fs` will double the tempo that the signal is played). MATLAB will continue to play the signal until it finishes or you use the command

```
clear sound
```

which is entered into the command window.

Chapter 5

Python

The various Python commands used to compute the examples in these notes are presented in this chapter. Python has become the most popular programming language in use today because it is freely available and it has been designed to be more intuitive than over languages such as C and MATLAB.

There are various ways in which you can use Python. We recommend that you download and install **Anaconda** (www.anaconda.org) which is a suite of software packages available for Windows, MacOS and Linux that includes Python (and the most common libraries), **Jupyter Notebook** which allows us to create documents that combines text and Python code (similar to MATLAB Live scripts) and **Spyder** which is a Integrated Developer Environment (IDE) which is similar to MATLAB.

These notes are not meant to serve as a 'how to' for programming in Python. If you would like to learn Python take a look at the Programming Skills materials on Moodle or search online for some excellent Python tutorials.

5.1 Python libraries

Python is a general purpose programming language which, in comparison to MATLAB, has very few commands. This is by design so that the language is simple to learn and does not suffer from the complications that can plague other languages. **Libraries** are collections of pre-combined codes that perform specific tasks. So instead of reinventing the wheel and starting from scratch, we can speed up our coding by making use of these libraries.

To use functions and commands from a particular library we first need to **import** it using

```
import libraryname as shortname
```

where `libraryname` is the name of the library and `shortname` is a shortened form of the name. The `as shortname` bit is optional but it allows us to use `shortname` as a prefix and helps to shorten our code.

All of the code shown in these notes assume that the libraries below have been imported. This is done using the following code

```
import numpy as np
import matplotlib.pyplot as plt
import soundfile as sf
import IPython.display as ipd
```

These libraries are briefly explained below.

5.1.1 Numpy

For our purposes we need to use Python to calculate mathematical functions and to perform operations on matrices so we can use the library **NumPy** (short for "Numerical Python") which we import using the

following code

```
import numpy as np
```

This is usually placed at the top of a Python program and allows us to use the NumPy commands using the prefix `np.`, e.g., `np.cos(x)` will calculate the cosine of `x`. We could use `import numpy` and use the full library name in the prefix, e.g., `numpy.cos(x)`, but it is common practice to define the abbreviation for brevity.

5.1.2 Matplotlib

Python does not have intrinsic plotting functions but thankfully there is a comprehensive library called **matplotlib** that provides MATLAB-like plot commands. To import matplotlib we use the following command

```
import matplotlib.pyplot as plt
```

which allows us to use the functions from **pyplot** which is a module that contains the plotting functions.

5.1.3 Soundfile

To read and write audio files we can use commands from the **soundfile** library which can be imported using

```
import soundfile as sf
```

Note that the soundfile library does not come packaged with most installations of Python and may need to be installed on your machine. This can be done by installing the librosa suite of music processing libraries, see https://librosa.org for instructions on how to do this.t

5.1.4 IPython

To play audio files in Jupyter Notebooks (all of the Python programs used in these notes were written in Jupyter) we need to import the **IPython.display** library using

```
import IPython.display as ipd
```

5.2 Sampling signals

5.2.1 Sampling an analogue signal

The sampling of an analogue signal defined using mathematical functions is a simple matter of calculating an array that contains the sampling times and then use that to calculate the signal. To do this we need the length, L, of the signal over which we are sampling and the sampling frequency, f_s, and from these we calculate the sampling interval and number of sample points using equations (2) and (4) which are

$$T = \frac{1}{f_s},$$
$$N = Lf_s.$$

The sampling times are then calculated by multiplying an array of numbers $(0, 1, 2, \ldots, N-1)$ by T using the `numpy.arange` command

```
t = np.arange(N) * T
```

It is also useful to define a function that describes the analogue signal in case we need to re-sample. The Python code in listing 5.1 samples the signal $s(t) = \cos(4t)$ of length $L = 2$s (taken from example 1.1 on page 6) using a sampling frequency of $f_s = 4$Hz.

Listing 5.1: Sampling an analogue signal in Python.

```
# Define analogue signal
def analogue_signal(t):
    return np.cos(4 * t)

# Define sampling parameters
L, fs = 2, 4
T = 1 / fs
N = L * fs

# Sample analogue signal
t = np.arange(N) * T
s = analogue_signal(t)
print('t = {}\ns = {}'.format(t, s))
```

```
t = [0.   0.25 0.5  0.75 1.   1.25 1.5  1.75]
s = [ 1.      0.5403 -0.4161 -0.99   -0.6536  0.2837  0.9602  0.7539]
```

5.2.2 Plotting analogue and digital signals

To create plots in Python we first need to create a figure window using

```
fig = plt.figure()
```

To plot an analogue signal we can simply use command `matplotlib.pyplot.plot` command

```
plt.plot(t, s, 'b')
```

which plots the signal array `s` against the time array `t` using a blue line. In order to represent the continuous nature of an analogue signal we need to make sure that we sample the `analogue_signal` function using lots of sample points (200 sampling points is common). This is done using the `numpy.linspace` command from the NumPy library.

```
t = np.linspace(0, L, 200)
```

which creates a 1×200 array of equally spaced values from 0 to L (which is the length of the signal).

To plot a digital signal we can use the `matplotlib.pyplot.step` command

```
plt.step(np.append(t, L), np.append(s, s[-1]), 'r', where='post')
```

which plots a horizontal line from each sample point to the end of the sample interval representing the discontinuities in the digital signal. Note that the `numpy.append` command has been used to append `L` and `s[-1]` (the last element in `s`) to the arrays `t` and `s` so that the last sampling interval is included in the plot.

For example, the code in listing 5.2 plots the digital and analogue signal from listing 5.1 and the resulting plot is shown in figure 5.1.

Listing 5.2: Plotting signals in Python.

```
# Calculate analogue signal
ta = np.linspace(0, L, 200)
sa = np.cos(4 * ta)

# Plot analogue and sampled signal
fig = plt.figure()
plt.plot(ta, sa, 'b-', label='analogue signal')
plt.step(np.append(t, L), np.append(s, s[-1]), 'r-', where='post', label='digital
    signal')
plt.legend()
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```

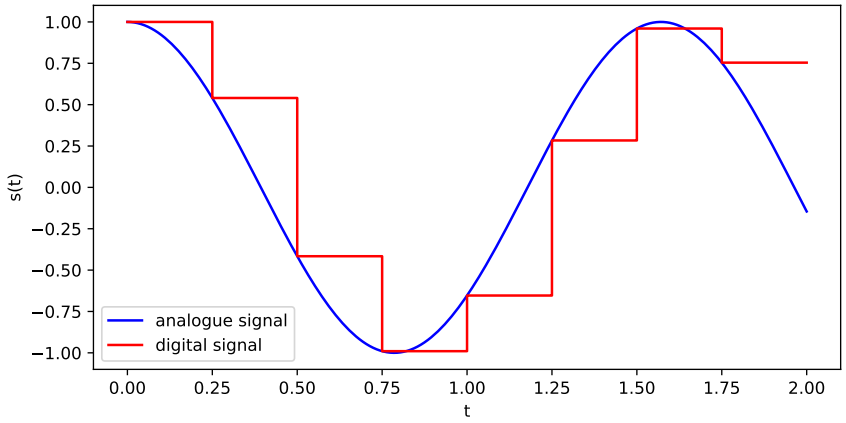


Figure 5.1: Python plot of an analogue and digital signal produced using the Python code in listing 5.2.

5.2.3 Quantising a sampled signal

To quantise a sampled signal we need the lower and upper bound of the signal, $s_{\min}$ and $s_{\max}$, and the bit depth M. Then it is simply a matter of calculating Δq and q using equations (5) and (6) which are

$$\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1},$$
$$q = \left\lfloor \frac{s - s_{\min}}{\Delta q} + \frac{1}{2} \right\rfloor.$$

To calculate the bit string for the quantised signal we can can use

```
bitstring = ''
for i in q:
    bitstring += ('{0:0'+str(M)+'b}').format(i)
```

which loops through each number in the `q` array, converts it to the equivalent binary number and appends it to the `bitstring` string. The Python function `quantise_signal` defined in listing 5.3 uses inputs of a signal array `s`, lower and upper bounds `smin` and `smax` and the bit depth `M` and outputs an array containing the quantised values `q` and the string `bitstring` which is the bit string representing the digital signal.

Listing 5.3: Python function to quantise a signal.

```
def quantise_signal(s, smin, smax, M):

    Deltaq = (smax - smin) / (2 ** M - 1)
    q = np.floor((s - smin) / Deltaq + 1 / 2).astype(int)
    bitstring = ''
    for i in q:
            bitstring += ('{0:0'+str(M)+'b}').format(i)

    return q, bitstring
```

The Python code in listing 5.4 uses the function `quantise_signal` to quantise the signal from listing 5.1 using lower and upper bounds of $s_{\min} = -1$ and $s_{\max} = 1$ and a bit depth of $M = 2$ (taken from example 1.2 on page 8).

Listing 5.4: Quantising a signal in Python.

```
# Quantise signal
smin, smax, M = -1, 1, 2
q, bitstring = quantise_signal(s, smin, smax, M)
print('q = {}\nbitstring = {}'.format(q, bitstring))
```

```
q = [3 2 1 0 1 2 3 3]
bitstring = 1110010001101111
```

5.2.4 Reconstructing a quantised signal

To reconstruct a quantised signal from a bit string we need to know the sampling frequency f_s, the lower and upper bound of the signal, $s_{\min}$ and $s_{\max}$, and the bit depth M. We first need to convert the bit string into a array of decimal numbers for the quantised values by looping through each sample point and using the `int` command to convert a binary number to an integer. Then we reconstruct the signal s using equation (7) which is

$$s_i = q_i \Delta q + s_{\min}.$$

The Python function `reconstruct_signal` defined in listing 5.5 uses inputs of the bit string `bitstring`, bit depth `M`, sampling frequency `fs` and lower and upper signal boundaries `smin` and `smax` and returns the signal values `s` and the sampling times `t`.

Listing 5.5: Python function to reconstruct a digital signal from a bitstring.

```
def reconstruct_signal(bitstring, M, fs, smin, smax):

    N = len(bitstring) // M
    T = 1 / fs
    t = np.arange(N) * T
    q = np.zeros(N)
    for i in range(0, N):
        q[i] = int(bitstring[M*i:M*i+M], 2)

    Deltaq = (smax - smin) / (2 ** M - 1)
    s = q * Deltaq + smin

    return s, t
```

The Python code in listing 5.6 uses the function `reconstruct_signal` to reconstruct the quantised signal from listing 5.4 with the sampling frequency of $f_s = 4$Hz, lower and upper bounds of $s_{\min} = -1$ and $s_{\max} = 1$ and bit depth of $M = 2$ (taken from example 1.3 on page 10).

Listing 5.6: Reconstructing a digital signal from a bit string in Python

```
# Reconstruct signal from bitstring
s, t = reconstruct_signal(bitstring, M, fs, smin, smax)
print('t = {}\ns = {}'.format(t, s))
```

```
q = [3. 2. 1. 0. 1. 2. 3. 3.]
t = [0.   0.25 0.5  0.75 1.   1.25 1.5  1.75]
s = [ 1.      0.3333 -0.3333 -1.     -0.3333  0.3333  1.      1.    ]
```

5.3 Spectral analysis

5.3.1 Calculating a Discrete Fourier Transform (DFT)

To calculate the DFT of a signal we need to use nested `for` loops to calculate the elements of the DFT matrix W equation (32) which is

$$[W]_{nk} = (e^{-i2\pi/N})^{(n-1)(k-1)},$$

and then apply the DFT using $F = W \cdot s$. The Python function `dft` defined in listing 5.7 calculates the DFT of the signal `s` and returns the array `F`.

Listing 5.7: Python function to calculate the DFT of a signal.

```
def dft(s):

    N = len(s)
    W = np.zeros((N, N)).astype(complex)
    for n in range(N):
        for k in range(N):
            W[n,k] = np.exp(-1j * 2 * np.pi / N) ** (n * k)

    F = np.dot(W, s)

    return F
```

The Python code in listing 5.8 uses the `dft` function to calculate the DFT of the signal $s(t) = (2, 0, -1, 1, 0)$ with period $P = 0.5$s and sampling frequency $f_s = 10$Hz (taken from example 2.6).

Listing 5.8: Calculating a DFT in Python.

```
# Define sampled signal
s = [2, 0, -1, 1, 0]
P, fs = 0.5, 10

# Calculate the DFT
F = dft(s)
N = len(s)
c = F / N
print('F =', F.round(4))
```

```
F = [2.+0.j     2.+1.1756j 2.-1.9021j 2.+1.9021j 2.-1.1756j]
```

5.3.2 Calculating the Fourier series from a DFT

Once the DFT has been calculated we can calculate the Fourier series approximation of the sampled signal. First we need to calculate the values of the complex coefficients c_i using equation (29) which is

$$c_i = \frac{F_i}{N},$$

where N is the length of the F array. Then the Fourier series approximation is calculated using equation (22) which is

$$s(t) = \sum_{n=0}^{m} c_i e^{i2\pi n f_0 t} + c_i^* e^{-i2\pi n f_0 t}.$$

where f_0 is the fundamental frequency and $m = \lfloor (N-1)/2 \rfloor$. The Python function `fourier` defined in listing 5.9 uses inputs of the DFT array `F` and the fundamental frequency `f0` and returns the Fourier series approximation of the sampled signal `s`.

Listing 5.9: Python function to calculate the Fourier series from the DFT.

```
def fourier(ta, c, f0):

    N = len(c)
    c = F / N
    nmax = (N - 1) // 2
    s = c[0]
    for n in range(1, nmax + 1):
        s += c[n] * np.exp(1j * 2 * np.pi * n * f0 * ta) \
            + np.conjugate(c[n]) * np.exp(-1j * 2 * np.pi * n * f0 * ta)

    return s
```

The Python code in listing 5.10 uses the function `fourier` to calculate the Fourier series approximation using the DFT values from listing 5.8 (taken from example 2.6 on page 31) and plots the Fourier series approximation and the sampled signal which is shown in **??**.

Listing 5.10: Plotting a Fourier series in Python.

```
# Calculate signal parameters
f0 = 1 / P
T = 1 / fs
t = np.arange(N) * T

# Calculate Fourier series approximation
ta = np.linspace(0, P, 100)
sa = fourier(ta, c, f0)

# Plot the signal and Fourier series approximation
fig = plt.figure()
plt.plot(ta, sa.real, 'b-', label='Fourier series')
plt.plot(t, s, 'ro', label = 'sampled signal')
plt.legend()
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```

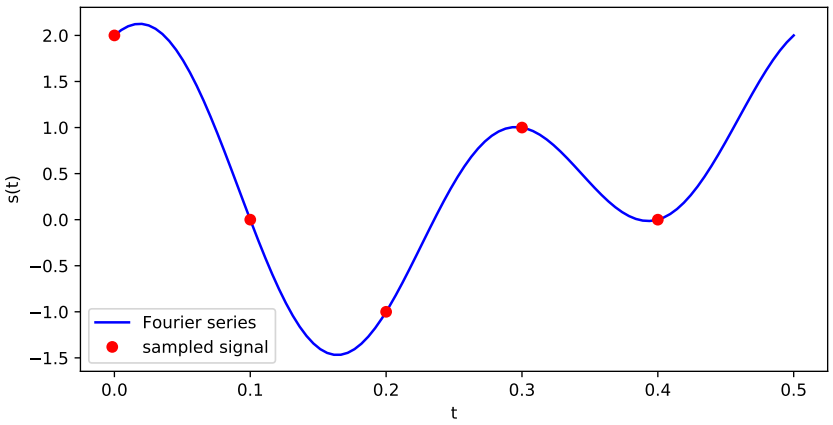


Figure 5.2: Python plot of the Fourier series approximation of the sampled signal from example 2.6.

5.3.3 Plotting a frequency spectrum

A frequency spectrum can be plotted using the `matplotlib.pyplot.stem` command from the matplotlib library

```
plt.stem(f, A, 'b', basefmt=' ', markerfmt='bo', use_line_collection = True)
```

which plots a single vertical line for each harmonic. The frequencies for each harmonic are calculated by multiplying the fundamental frequency, f_0, by the array of numbers $(0, 1, 2, \ldots, n_{\max})$ where $n_{\max}$ is the maximum harmonic using

```
f = np.arange(nmax + 1) * f0
```

The DC and amplitudes of the harmonics are calculated from the complex coefficients, c_i, using

$$\frac{a_0}{2} = |c_0|,$$
$$A_i = 2|c_i|, \qquad i = 1, 2, \ldots, n_{\max}.$$

so the Python commands `abs` and `numpy.append` are used to calculate an array containing the amplitudes

```
A = abs(np.append(c[0], 2 * c[1:nmax+1]))
```

The Python code in listing 5.11 calculates the amplitudes of the harmonics given the values of the complex coefficients for the signal from listing 5.8 and plots the frequency spectrum which is shown in figure 5.3 (from example 2.6 on page 31). Note that it is common to use a standard `matplotlib.pyplot.plot` command to plot the frequency spectrum when dealing with a large number of harmonics.

Listing 5.11: Plotting a frequency spectrum in Python.

```
# Calculate the frequencies and amplitudes of the harmonics
nmax = (N - 1) // 2
f = np.arange(nmax + 1) * f0
A = abs(np.append(c[0], 2 * c[1:nmax+1]))
print('f = {}\nA = {}'.format(f, A.round(4)))

# Plot the frequency spectrum
fig = plt.figure()
plt.stem(f, A, 'b', basefmt=' ', markerfmt='bo', use_line_collection = True, )
plt.xticks(f)
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.show()
```

```
f = [0. 2. 4.]
A = [0.4   0.928 1.104]
```

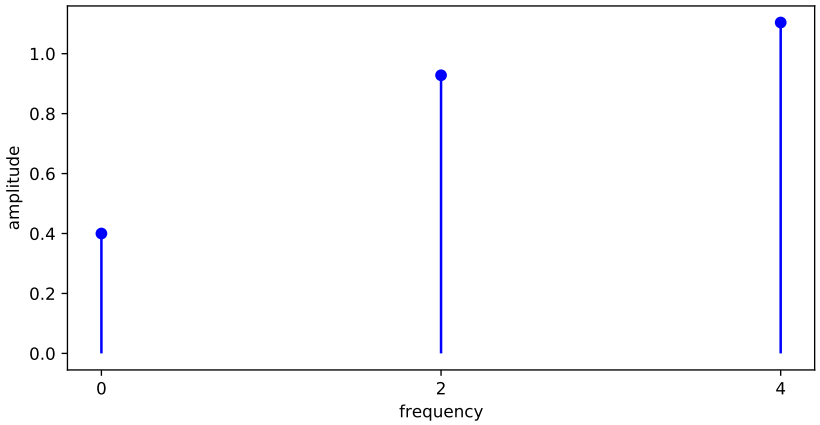


Figure 5.3: Python plot of the frequency spectrum from listing 5.11.

5.3.4 Calculating a Inverse Discrete Fourier Transform (IDFT)

To calculate the IDFT we require the F_i values from the DFT of a signal. The IDFT matrix is calculated using nested `for` loop, similar to how the DFT matrix was calculated, where the elements of the IDFT matrix are calculated using equation (34) which is

$$[W^{-1}]_{nk} = \frac{(e^{i2\pi/N})^{(n-1)(k-1)}}{N}.$$

The signal values are then calculated using $s = W^{-1} \cdot s$.

Listing 5.12: Python function to calculate the IDFT.

```
def idft(F):

    N = len(F)
    W = np.zeros((N, N)).astype(complex)
    for n in range(N):
        for k in range(N):
            W[n,k] = np.exp(1j * 2 * np.pi / N) ** (n * k) / N

    s = np.dot(W, F)

    return s
```

The Python code in listing 5.13 uses the `idft` function to calculate the signal values from the DFT values from listing 5.8 (from example 2.7 on page 33). Note that the `s` array is complex so the real parts have been printed to 4 decimal places using the `s.real.round(4)` command.

Listing 5.13: Calculating an IDFT in Python.

```
# Calculate IDFT
s = idft(F)
print('s =',s.real.round(4))
```

```
s = [ 2. -0. -1.  1.  0.]
```

5.3.5 Calculating the Fast Fourier Transform (FFT)

To calculate an FFT in Python we can use the `numpy.fft.ftt` command from the NumPy library which is implemented using

```
F = np.fft.fft(s)
```

The Python code in listing 5.14 calculates the FFT of the sampled signal from example 2.6 on page 31

$$s(t) = (2, 0, -1, 1, 0),$$

and plots the frequency spectrum which is shown in figure 5.4.

Listing 5.14: Calculating an FFT in Python.

```
# Define sampled signal
s = np.array([2, 0, -1, 1, 0])
P = 0.5;

# Calculate signal parameters
f0 = 1 / P
N = len(s)
nmax = (N - 1) // 2

# Calculate FFT
F = np.fft.fft(s)
c = F / N
print('F =', F.round(4))

# Calculate the frequencies and amplitudes of the harmonics
f = np.arange(nmax + 1) * f0
A = abs(np.append(c[0], 2 * c[1:nmax+1]))
print('f ={}\nA ={}'.format(f, A.round(4)))

# Plot signal and frequency spectrum
fig = plt.figure()
plt.stem(f, A, 'b', basefmt=' ', markerfmt='bo', use_line_collection = True, )
plt.xticks(f)
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.show()
```

```
F = [2.+0.j     2.+1.1756j 2.-1.9021j 2.+1.9021j 2.-1.1756j]
f = [0. 2. 4.]
A = [0.4   0.928 1.104]
```

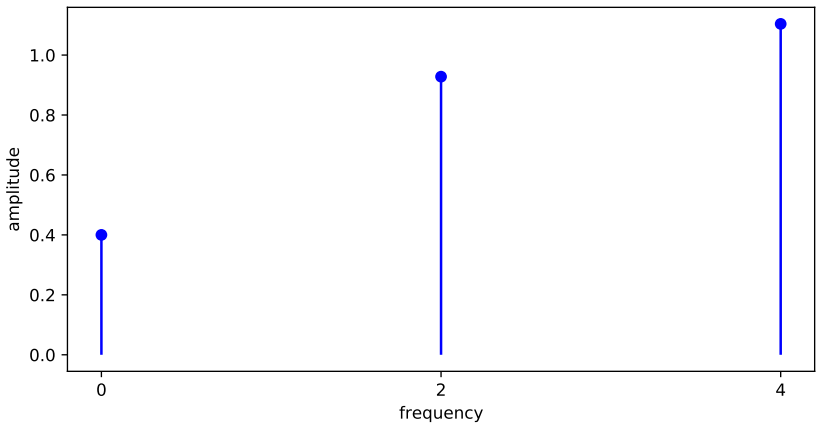


Figure 5.4: Python plot of the signal and frequency spectrum from example 2.8.

5.3.6 Calculating the Inverse Fast Fourier Transform (IFFT)

To calculate an IFFT in Python we can use the `numpy.fft.ifft` command from the NumPy library which is implemented using

```
s = np.fft.ifft(F);
```

The Python code in listing 5.15 calculates the signal values using the FFT of the signal from listing 5.14 (taken from example 2.7 on page 33)

Listing 5.15: Calculating an IFFT in Python.

```
# Calculate IFFT
s = np.fft.ifft(F)
print('s =', s.real.round(4))
```

```
s = [ 2. -0. -1.  1.  0.]
```

5.4 Reading, writing and playing audio files

5.4.1 Reading audio files

We can read in an audio file using the `soundfile.read` command from the soundfile library (see section 5.1.3)

```
s, fs = sf.read('handel.wav')
```

which reads the audio file specified by `filename` and stores the signal values in the $m \times n$ array `s` where m is the number of sample points and n is the number of channels (e.g., 1 channel is mono sound, 2 channel is stereo sound etc.) and the sampling frequency in `fs`. For example, the code in listing 5.16 reads in an audio file containing 8 seconds from Handel's Hallelujah Chorus, calculates the signal parameters and plots the signal which is shown in figure 5.5.

Listing 5.16: Reading an audio file in Python.

```
# Read audio file
s, fs = sf.read('handel.wav')

# Determine signal parameters
N = s.shape[0]
T = 1 / fs
L = N * T
t = np.arange(N) * T
print('N = {}\nnchannels = {}\nfs = {}Hz\nT = {:0.4e} s\nL = {:0.4f} s' \
      .format(N, 1, fs, T, L))

# Plot audio signal
fig = plt.figure()
plt.plot(t, s, 'b')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```

```
N = 65536
nchannels = 1
fs = 8192Hz
T = 1.2207e-04 s
L = 8.0000 s
```

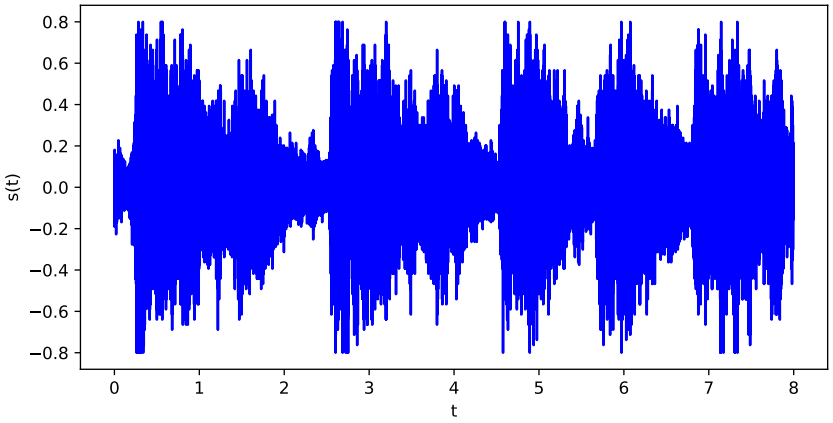


Figure 5.5: Python plot of the first 8 seconds of Handel's Hallelujah Chorus.

5.4.2 Writing audio files

To write an audio file we can use the `soundfile.write` command from the soundfile library

```
sf.write(filename, s, fs)
```

which writes the signal contained in the array `s` to the file specified by `filename` and the sampling frequency `fs` (this is necessary so that the audio is played at the correct tempo). The audio format used is determined by the file extension in `filename`. The MATLAB code in listing 4.12 creates a signal of 2 seconds of random noise and writes it to the wav file `random_noise.wav`.

Listing 5.17: Writing an audio file in Python.

```
# Create signal (random noise)
L, fs = 2, 44100
N = L * fs
s = np.random.rand(N)

# Write audio file
sf.write('random_noise.wav', s, fs)
```

5.4.3 Playing audio files

To play an audio file in a Jupyter Notebook we can use the `IPython.display.Audio` command from the **IPython** library

```
IPython.display.Audio('handel.wav')
```

which will produce the following widget allowing you to play the audio file.

Appendix A

Mathematical Fundamentals

To be completed

A.1 Trigonometry

A.1.1 Radians

Radians is a measure of an angle defined that the angle of 1 radian subtended from a unit circle (a circle of radius 1) produces an arc with arc length 1 (figure A.1). Since the circumference of a circle is $2\pi r$ then there are 2π radians in a circle. Radians are easier to work with than degrees so should be used wherever possible.

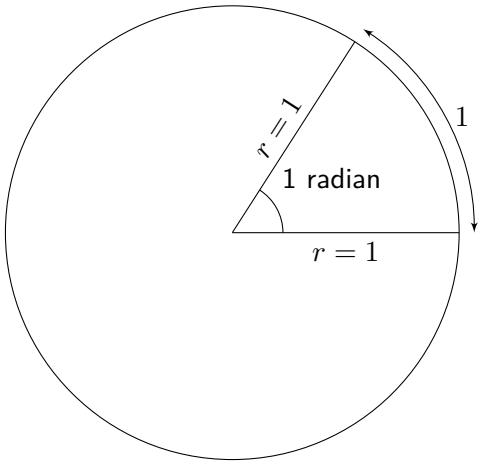


Figure A.1: 1 radian subtended from a unit circle produces arc with arc length 1.

A.1.2 Sine and cosine

Sine and **cosine** are trigonometric functions of an angle the values of which are related to the side lengths and angles of a right-angled triangle. Consider the right-angled triangle shown in figure A.2 where the sides are labelled 'adjacent', 'opposite' and 'hypotenuse' depending on their position relative to the angle θ, the sine and cosine functions are defined as

$$\sin(\theta) = \frac{\text{opposite}}{\text{hypotenuse}},$$
$$\cos(\theta) = \frac{\text{adjacent}}{\text{hypotenuse}}.$$

The sine and cosine functions have been plotted in figure A.3. Both functions have a period of 2π and the two functions are a phase shift of the other by $\pi/2$.

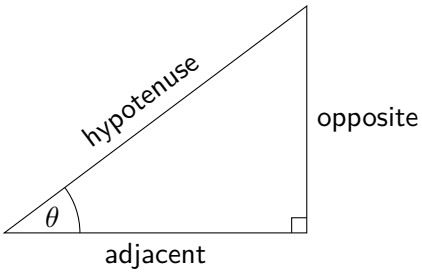


Figure A.2: A right-angled triangle.

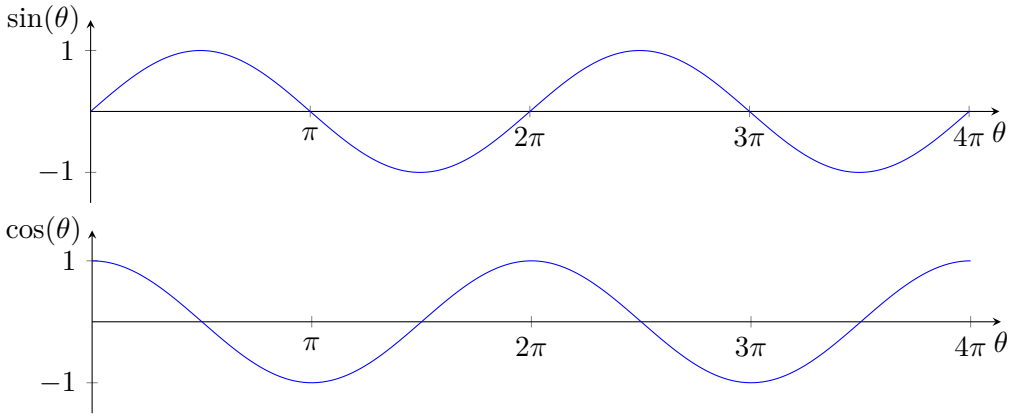


Figure A.3: Plots of the sine and cosine trigonometric functions.

A.1.3 atan2

The $\mathrm{atan2}(y, x)$ function calculates the angle θ between the positive x axis and the ray (x, y) and is limited to $(-\pi, \pi]$.

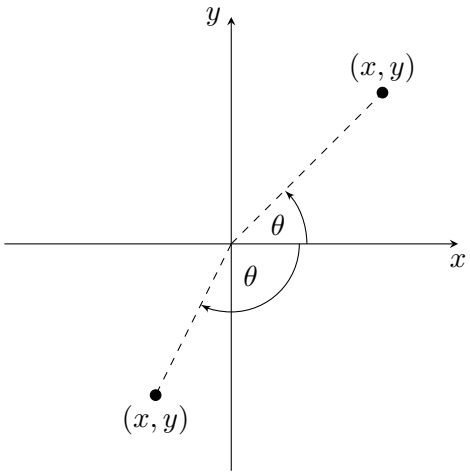


Figure A.4: The $\mathrm{atan2}(y, x)$ calculates the angle between the positive x axis and the ray (x, y).

$$\text{atan2}(y,x) = \begin{cases} \text{atan}\left(\frac{y}{x}\right), & x > 0, \\ \text{atan}\left(\frac{y}{x}\right) + \pi, & x < 0 \text{ and } y \geq 0, \\ \text{atan}\left(\frac{y}{x}\right) - \pi, & x < 0 \text{ and } y < 0, \\ \frac{\pi}{2}, & x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2}, & x = 0 \text{ and } y < 0, \\ \text{undefined}, & x = 0 \text{ and } y = 0. \end{cases}$$

A.2 Complex numbers

A **complex number** z is a number of the form $z = a + bi$ where $a, b \in \mathbb{R}$ and $i^2 = -1$. The value a is known as the **real** part of z which is denoted by $\text{Re}(z)$ and b is known as the **imaginary** part which is denoted by $\text{Im}(z)$. The set of all complex numbers is denoted by $\mathbb{C}$.

Complex numbers can represented on a **complex plane** which graphs a complex number using the horizontal axis for the real part and the vertical axis for the imaginary part (figure A.5).

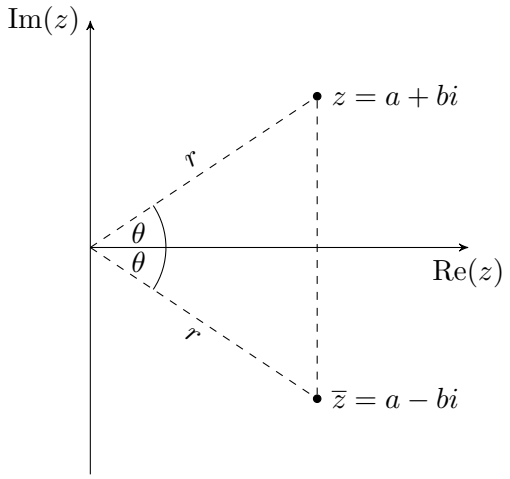


Figure A.5: The complex number $z = a + bi$ represented in the complex plane.

The **modulus** or absolute value of a complex number $z = a + bi$ is the distance of z from the origin of the complex plane which is calculated using Pythagoras' theorem, e.g.,

$$r = |z| = \sqrt{a^2 + b^2}.$$

Using the sine and cosine trigonometric functions we can represent the complex number $z = a + bi$ using **complex polar co-ordinates**

$$z = r(\cos(\theta) + \sin(\theta)i), \tag{44}$$

where r is the modulus of z and $\theta = \text{atan2}(b, a)$

The addition (and subtraction) of two complex numbers $z_1 = a + bi$ and $z_2 = c + di$ is achieved by adding the real and imaginary parts separately, e.g.,

$$z_1 + z_2 = a + bi + c + di = a + c + (b + d)i.$$

The multiplication of two complex numbers is achieved by multiplying out $(a + bi)(c + di)$, e.g.,

$$z_1 z_2 = (a + bi)(c + di) = ac + adi + bci + bdi^2 = ac - bd + (ad + bc)i.$$

A.2.1 Euler's formula

Euler's formula gives a relationship between the sine and cosine trigonometric functions and the exponential function. For any real number θ

$$e^{i\theta} = \cos(\theta) + i\sin(\theta). \tag{45}$$

So the complex polar form of a complex number, equation (44), can be representing using

$$z = re^{i\theta}.$$

A.2.2 Complex conjugate

The **conjugate** of a complex number $z = a + bi$ is denoted as $\overline{z}$ and is another complex number with the sign of the imaginary part reversed, e.g., $\overline{z} = a - bi$.

Appendix B

Exercise Solutions

B.1 Introduction to digital signals

These are the solutions to the exercises from the Introduction to Digital Signals chapter which are on page 14.

1. (a) $T = \frac{L}{N} = \frac{5}{10} = 0.5$s;

   (b) $f_s = \frac{1}{T} = \frac{1}{0.5} = 2$Hz;

   (c) $t = (0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5)$.

2. (a) $T = \frac{1}{f_s} = \frac{1}{8} = 0.125$s;

   (b) $L = N \cdot T = 5000 \cdot 0.125 = 625$s;

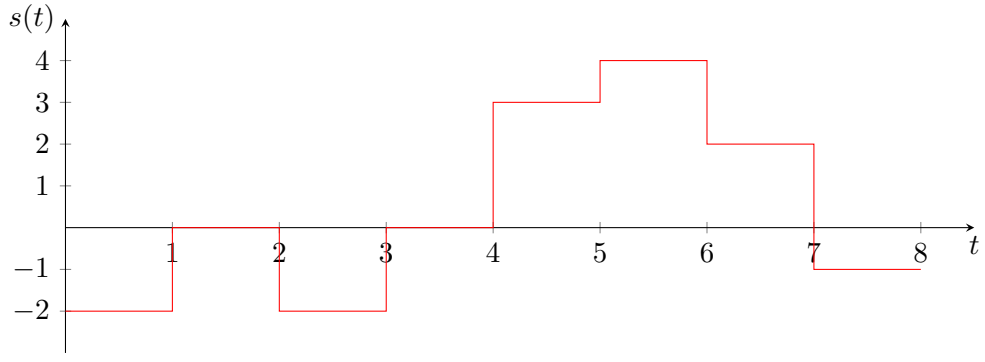
   (c) $t_{5000} = L - T = 625 - 0.125 = 624.875$s.

3. (a) $T = \frac{1}{f_s} = \frac{1}{1000} = 0.0001$s, therefore $t = (0, 0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007)$s;

   (b) $s_6 = 4$;

   (c) $t = 0.004$s;

   (d)



4. (a) $\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1} = \frac{5 - (-5)}{2^3 - 1} = \frac{10}{7} \approx 1.43$, therefore

$$q = (-5, -3.5714, -2.1429, -0.7143, 0.7143, 2.1429, 3.5714, 5)$$

(b)

$$q_0 = \left\lfloor \frac{-4.2-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 1, \qquad q_1 = \left\lfloor \frac{0.1-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 4,$$
$$q_2 = \left\lfloor \frac{-1.8-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 2, \qquad q_3 = \left\lfloor \frac{2.9-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 6,$$
$$q_4 = \left\lfloor \frac{4.9-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 7, \qquad q_5 = \left\lfloor \frac{-0.5-(-5)}{10/7} + \frac{1}{2} \right\rfloor = 3.$$

Therefore $q = (1, 4, 2, 6, 7, 3)$;

(c) bit string $= 001100010110111011$.

5. (a) $T = \frac{1}{f_s} = \frac{1}{5} = 0.2$s, therefore $t = (0, 0.2, 0.4, 0.6, 0.8)$;

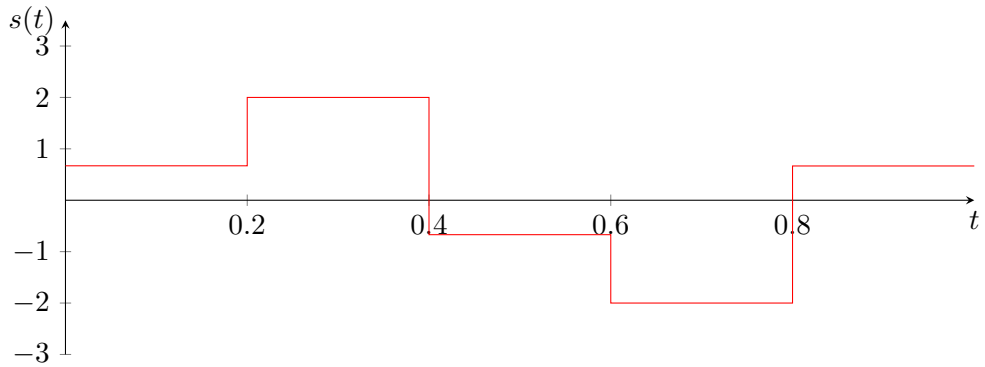
(b) $q = (10\ 11\ 01\ 00\ 10)_2 = (2, 3, 1, 0, 2)_{10}$;

(c) $\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1} = \frac{2-(-2)}{2^2-1} = \frac{4}{3} \approx 1.3333$

$$s_0 = 2 \cdot 1.3333 + (-2) = 0.6667, \qquad s_1 = 3 \cdot 1.3333 + (-2) = 2,$$
$$s_2 = 1 \cdot 1.3333 + (-2) = -0.6667, \qquad s_3 = 0 \cdot 1.3333 + (-2) = -2,$$
$$s_0 = 2 \cdot 1.3333 + (-2) = 0.6667,$$

therefore $s = (0.6667, 2, -0.6667, -2, 0.6667)$;

(d)



6. (a) $T = \frac{1}{f_s} = \frac{1}{6000} = 1.67 \times 10^{-4}$s;

(b) $t_{300} = (300-1)T = 299 \cdot 1.67 \times 10^{-4} = 0.0498$s;

(c) $L = \frac{\text{memory}}{\text{bit rate}} = \frac{360000 \cdot 8}{12000} = 240$s;

(d) $N = L \cdot f_s = 240 \cdot 6000 = 1440000$;

(e) $M = \frac{\text{bit rate}}{f_s} = \frac{12000}{6000} = 2$, therefore there are $2^M = 2^2 = 4$ quantisation levels.

7. (a) audio length $= \frac{\text{memory}}{f_s \cdot M \cdot \text{channels}} = \frac{750,000,000 \cdot 8}{8,000 \cdot 8 \cdot 2} = 56875$s or 781.25 minutes;

(b) audio length $= \frac{\text{memory}}{f_s \cdot M \cdot \text{channels}} = \frac{750,000,000 \cdot 8}{44,100 \cdot 16 \cdot 2} = 4251.7$s or 70.86 minutes.

8. $T = \frac{1}{f_s} = \frac{1}{1000} = 0.0001$ therefore $t = (0, 1, 2, 3, 4, 5)$ ms

   Estimate signal from graph $s(t) = (3, -0.2, 1, 0.5, 0.3, 3.1)$.

   $\Delta q = \frac{s_{\max} - s_{\min}}{2^M - 1} = \frac{4 - (-2)}{2^3 - 1} = \frac{6}{7}$ therefore

$$q_0 = \left\lfloor \frac{3 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 6, \qquad q_1 = \left\lfloor \frac{-0.2 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 2,$$
$$q_2 = \left\lfloor \frac{1 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 4, \qquad q_3 = \left\lfloor \frac{0.5 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 3,$$
$$q_4 = \left\lfloor \frac{0.3 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 3, \qquad q_5 = \left\lfloor \frac{3.1 - (-2)}{6/7} + \frac{1}{2} \right\rfloor = 6,$$

   so $q = (6, 2, 4, 3, 3, 6)$. Converting to a bit string gives 110 010 100 011 011 110.

B.2 Spectral Analysis

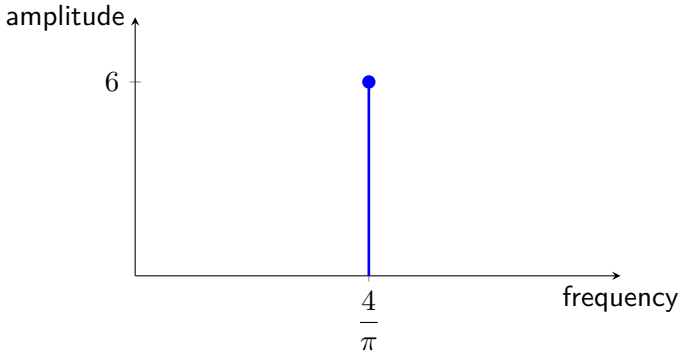
These are the solutions to the exercises from the Spectral Analysis chapter which are on page 40.

1. (a) (i) $A_0 = 0$, $A_1 = 6$, $f = \frac{8}{2\pi} = \frac{4}{\pi}$Hz, $\phi = 0$;

   (ii) $f_0 = \frac{4}{\pi}$, $P = \frac{1}{4/\pi} = \frac{\pi}{4}$;

   (iii) $f_{\text{Nyq}} = 2\left(\frac{4}{\pi}\right) = \frac{8}{\pi}$;

   (iv)

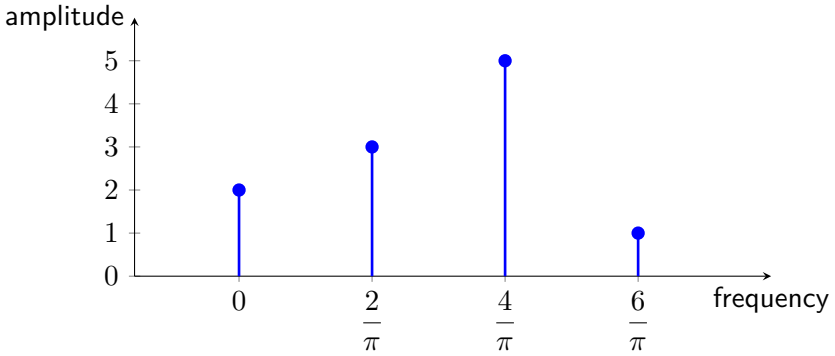


   (b) (i) $A_0 = 2$, $A_1 = 3$, $A_2 = 5$, $A_3 = 1$, $f = \left(\frac{2}{\pi}, \frac{4}{\pi}, \frac{6}{\pi}\right)$, $\phi = (2, 0, 0)$;

   (ii) $f_0 = \min(f) = \frac{2}{\pi}$, $P = \frac{1}{2/\pi} = \frac{\pi}{2}$;

   (iii) $f_{\text{Nyq}} = 2\max\left(\frac{2}{\pi}, \frac{4}{\pi}, \frac{6}{\pi}\right) = \frac{12}{\pi}$

   (iv)

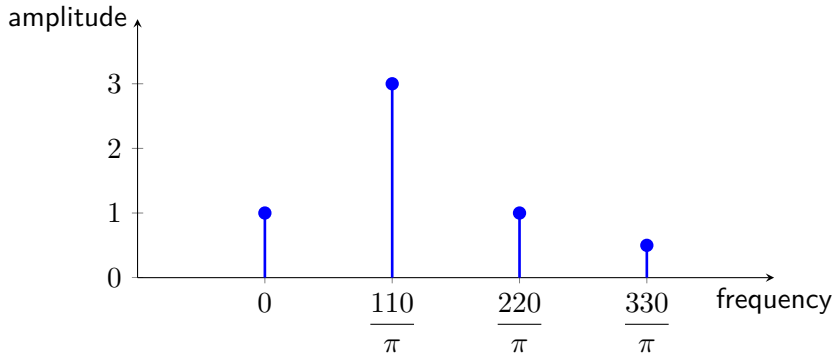


   (c) (i) $A_0 = 1$, $A_1 = 3$, $A_2 = 1$, $A_3 = 0.5$, $f = \left(\frac{110}{\pi}, \frac{220}{\pi}, \frac{330}{\pi}\right)$, $\phi = (0, 2, 0)$;

   (ii) $f_0 = \min(f) = \frac{110}{\pi}$, $T = \frac{1}{f_0} = \frac{\pi}{110}$;

   (iii) $f_{\text{Nyq}} = 2\max\left(\frac{110}{\pi}, \frac{220}{\pi}, \frac{330}{\pi}\right) = \frac{660}{\pi}$

   (iv)



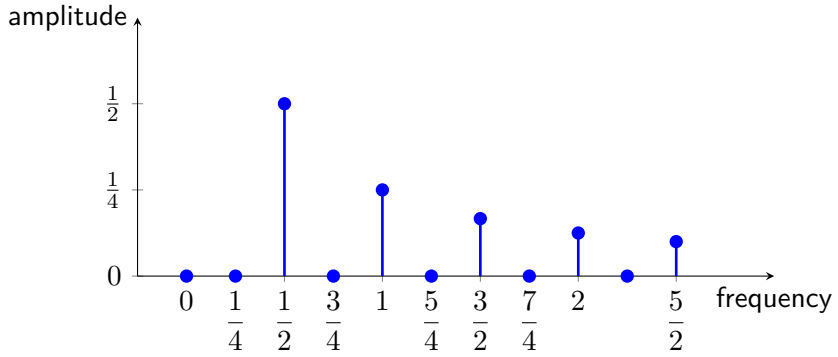
2. (a) $f_0 = \dfrac{\pi/2}{2\pi} = \dfrac{1}{4}$Hz;

   (b) $f_{\text{Nyq}} = 2f_{10} = 2\left(\dfrac{5}{2}\right) = 5$Hz;

   (c) $A_0 = 0$;

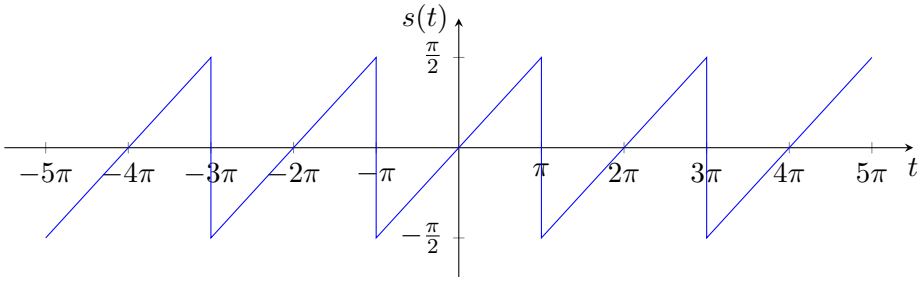
   (d) $f = \left(\dfrac{1}{4}, \dfrac{1}{2}, \dfrac{3}{4}, 1, \dfrac{5}{4}, \dfrac{3}{2}, \dfrac{7}{4}, 2, \dfrac{9}{2}, \dfrac{5}{2}\right)$,

   $A_{1,3,5,7,9} = 0$ and $A_2 = \dfrac{1}{2}$, $A_4 = \dfrac{1}{4}$, $A_6 = \dfrac{1}{6}$, $A_8 = \dfrac{1}{8}$, $A_{10} = \dfrac{1}{10}$.



   (e) $T = \dfrac{1}{f_{\text{Nyq}}} = \dfrac{1}{5}$s.

3. (a)



   (b) $P = \pi - (-\pi) = 2\pi$, $f_0 = \dfrac{1}{P} = \dfrac{1}{2\pi}$;

(c) Using equations (16) and (17) to calculate $a_0/2$, a_n and b_n

$$\frac{a_0}{2} = \frac{1}{\pi}\int_{-\pi}^{\pi} t\,\mathrm{d}t = \frac{1}{\pi}\left[\frac{t^2}{2}\right]_{-\pi}^{\pi} = 0,$$
$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} t\cos(nt)\,\mathrm{d}t = \frac{1}{\pi}\left[\frac{\cos(nt)}{n^2} + \frac{t\sin(nt)}{n}\right]_{-\pi}^{\pi} = 0,$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} t\sin(nt)\,\mathrm{d}t = \frac{1}{\pi}\left[-\frac{t\cos(nt)}{n} + \frac{\sin(nt)}{n^2}\right]_{-\pi}^{\pi} = -\frac{2\cos(n\pi)}{n} = \frac{2(-1)^{n+1}}{n}.$$

(d) Since $f_0 = 1/2\pi$ then the sine-cosine form of the Fourier series is

$$s(t) = \sum_{n=1}^{\infty} \frac{2(-1)^{n+1}}{n}\sin(nt)$$

(e) Using equations (18) and (19) to calculate the amplitude and phase angle

$$A_n = \sqrt{0^2 + \left(\frac{2(-1)^{n+1}}{n}\right)^2} = \frac{2(-1)^{n+1}}{n},$$
$$\phi = -\operatorname{atan2}\left(\frac{2(-1)^{n+1}}{n}, 0\right) = -\frac{\pi}{2},$$

therefore the amplitude-phase form of the Fourier series is

$$s(t) = \sum_{n=1}^{\infty} \frac{2(-1)^{n+1}}{n}\cos\left(nt - \frac{\pi}{2}\right)$$

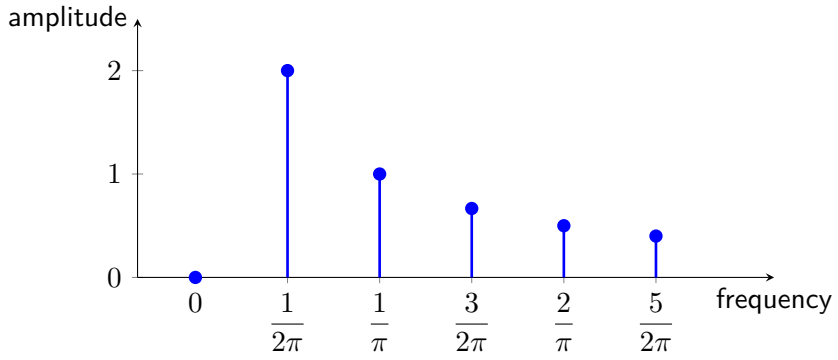
(f) Using equation (23) to calculate the complex coefficients

$$c_0 = \frac{a_0}{2} = 0,$$
$$c_n = \frac{1}{2}\left(0 - \frac{i2(-1)^{n+1}}{n}\right) = \frac{i(-1)^{n+1}}{n},$$
$$c_{-n} = -\frac{i(-1)^{n+1}}{n}$$

and therefore the exponential form of the Fourier series is

$$s(t) = \sum_{n=0}^{\infty} \frac{i(-1)^{n+1}}{n}e^{int} - \frac{i(-1)^{n+1}}{n}e^{-int}.$$

(g) $f_n = n/(2\pi)$, $A_0 = 0$, $A_n = 2/n$

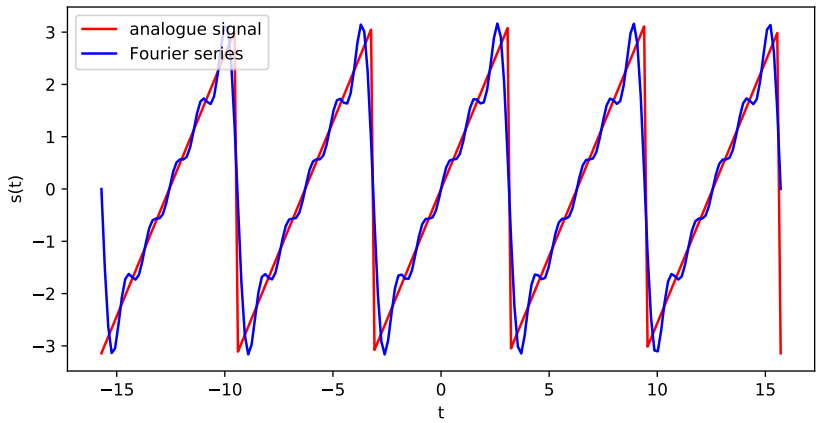


(h) MATLAB:

```
% Calculate Fourier series
t = linspace(-5 * pi, 5 * pi, 200);
s = 0;
for n = 1 : 5
    s = s + 2 * (-1) ^ (n + 1) / n * cos(n * t - pi / 2);
end

% Calculate analogue signal
sa = mod(t - pi, 2 * pi) - pi;

% Plot signal
plot(t, sa, 'r')
hold on
plot(t, s, 'b')
hold off
pbaspect([2, 1, 1])
axis padded
xlabel('t')
ylabel('s(t)')
```

Python:

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

# Calculate Fourier series
t = np.linspace(-5 * np.pi, 5 * np.pi, 200)
s = 0
for n in range(1, 6):
    s += 2 * (-1) ** (n + 1) / n * np.cos(n * t - np.pi / 2)

# Calculate analogue signal (optional)
sa = (t - np.pi) % (2 * np.pi) - np.pi

# Plot signal
fig = plt.figure(figsize=(8,4))
plt.plot(t, sa, 'r', label='analogue signal')
plt.plot(t, s.real, 'b', label='Fourier series')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.legend()
plt.show()
```



4. A transformation $T : U \to V$ is a linear transformation if the following are satisfied

1. $T(\alpha\mathbf{u}) = \alpha T(\mathbf{u})$;
2. $T(\mathbf{u}_1 + \mathbf{u}_2) = T(\mathbf{u}_1) + T(\mathbf{u}_2)$;

where $\mathbf{u}_1, \mathbf{u}_2 \in U$ and $\alpha \in \mathbb{C}$

Check criteria 1. Let $\mathbf{u} = s(t)$ where $s(t) \in \mathbb{C}$ and $\alpha \in \mathbb{C}$ then

$$\mathcal{F}[\alpha s(t)](\nu) = \int_{-\infty}^{\infty} \alpha s(t) e^{-i2\pi\nu t}\,\mathrm{d}t = \alpha \int_{-\infty}^{\infty} s(t) e^{-i2\pi\nu t}\,\mathrm{d}t = \alpha\mathcal{F}[s(t)](\nu)$$

Check criteria 2. Let $\mathbf{u}_1 = s(t)$ and $\mathbf{u}_2 = u(t)$ where $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{C}$ then

$$\begin{aligned}
\mathcal{F}[s(t) + u(t)](\nu) &= \int_{-\infty}^{\infty} (s(t) + u(t)) e^{-i2\pi\nu t}\,\mathrm{d}t \\
&= \int_{-\infty}^{\infty} s(t) e^{-i2\pi\nu t} + u(t) e^{-i2\pi\nu t}\,\mathrm{d}t \\
&= \int_{-\infty}^{\infty} s(t) e^{-i2\pi\nu t}\,\mathrm{d}t + \int_{-\infty}^{\infty} u(t) e^{-i2\pi\nu t}\,\mathrm{d}t \\
&= \mathcal{F}[s(t)](\nu) + \mathcal{F}[u(t)](\nu)
\end{aligned}$$

Therefore the Fourier transformation is a linear transformation.

5. (a) $T = \dfrac{1}{f_s} = \dfrac{1}{12}$s, $P = \dfrac{N}{f_s} = \dfrac{6}{12} = \dfrac{1}{2}$s;

   (b) Calculating the elements for the DFT matrix. Since $N = 6$ we only need to calculate the first $\lfloor (N+1)/2 \rfloor = \lfloor 7/2 \rfloor = 3$ rows (so the elements are $(e^{-i2\pi/3})^n$ where $n = 1, \ldots, 10$)

$$\begin{aligned}
e^{-i\pi/3} &= \cos\left(\frac{\pi}{3}\right) - i\sin\left(\frac{\pi}{3}\right) = \frac{1}{2} - \frac{\sqrt{3}}{2}i, \\
(e^{-i\pi/3})^2 &= \left(\frac{1}{2} - \frac{\sqrt{3}}{2}i\right)\left(\frac{1}{2} - \frac{\sqrt{3}}{2}i\right) = -\frac{1}{2} - \frac{\sqrt{3}}{2}i, \\
(e^{-i\pi/3})^3 &= (e^{-i\pi/3})^2 e^{-i\pi/3} = \left(-\frac{1}{2} - \frac{\sqrt{3}}{2}i\right)\left(\frac{1}{2} - \frac{\sqrt{3}}{2}i\right) = -1, \\
(e^{-i\pi/3})^4 &= (e^{-i\pi/3})^3 e^{-i\pi/3} = -\frac{1}{2} + \frac{\sqrt{3}}{2}, \\
(e^{-i\pi/3})^5 &= (e^{-i\pi/3})^3 (e^{-i\pi/3})^2 = \frac{1}{2} + \frac{\sqrt{3}}{2}i, \\
(e^{-i\pi/3})^6 &= (e^{-i\pi/3})^0 = 1, \\
(e^{-i\pi/3})^8 &= (e^{-i\pi/3})^2 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i, \\
(e^{-i\pi/3})^{10} &= (e^{-i\pi/3})^4 = -\frac{1}{2} + \frac{\sqrt{3}}{2}.
\end{aligned}$$

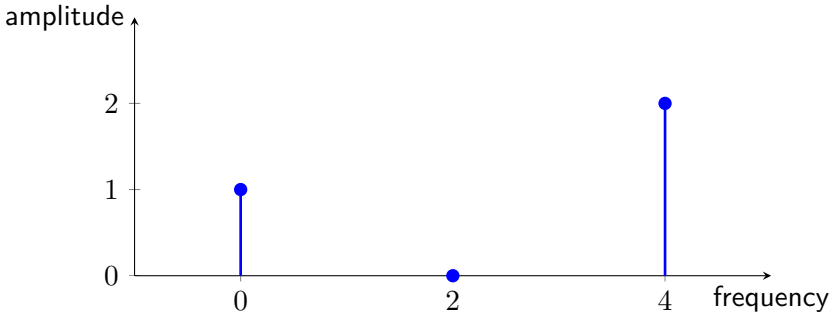
   The DFT matrix is

$$W = \begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & \frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -1 & -\frac{1}{2} + \frac{\sqrt{3}}{2}i & \frac{1}{2} + \frac{\sqrt{3}}{2}i \\
1 & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} + \frac{\sqrt{3}}{2}i & 1 & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} + \frac{\sqrt{3}}{2}i \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{pmatrix}.$$

(c) Calculating the DFT

$$\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -1 & -\frac{1}{2} + \frac{\sqrt{3}}{2}i & \frac{1}{2} + \frac{\sqrt{3}}{2}i \\ 1 & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} + \frac{\sqrt{3}}{2}i & 1 & -\frac{1}{2} - \frac{\sqrt{3}}{2}i & -\frac{1}{2} + \frac{\sqrt{3}}{2}i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} 3 \\ 0 \\ 0 \\ 3 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 6 \\ 0 \\ 6 \\ \vdots \end{pmatrix}.$$

(d) Since $\lfloor (6-1)/2 \rfloor = 2$ we can approximate up to the second harmonic. The fundamental frequency is $f_0 = 1/(1/2) = 2$Hz and $a_0/2 = |6/6| = 1$, $A_1 = 2|0/6| = 0$, $A_2 = 2|6/6| = 2$.



(e) Dividing F by $N = 6$ gives $c = (1, 0, 1)$ so the exponential form of the Fourier series is

$$\begin{aligned} s(t) &\approx \sum_{n=0}^{2} c_i e^{i2\pi nt} + c_i^* e^{i2\pi nt} \\ &= 1 + e^{i2\pi 2t} + e^{-i2\pi 2t} \end{aligned}$$

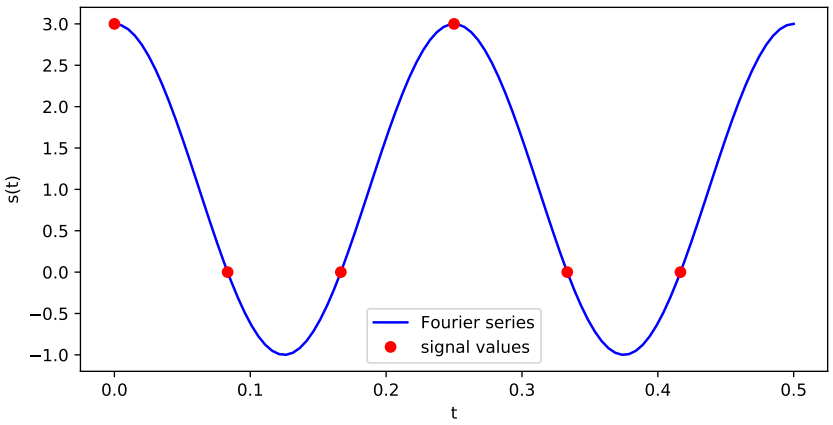
(f) MATLAB:

```
% Calculate Fourier series
t = (0 : N - 1) * T;
ta = linspace(0, P, 200);
sa = 1 + exp(-1i * 2 * pi * 2 * f0 * ta) + exp(1i * 2 * pi * 2 * f0 * ta);

% Plot Fourier series and signal values
plot(ta, sa)
hold on
plot(t, s, 'ro', 'MarkerFaceColor', 'r')
hold off
pbaspect([ 2, 1, 1 ])
axis padded
xlabel('t')
ylabel('s(t)')
legend('Fourier series', 'signal values', 'Location', 'south')
```

Python:

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

# Calculate Fourier series
t = np.arange(N) * T
ta = np.linspace(0, P, 100)
sa = np.real(1 + np.exp(-1j * 2 * np.pi * 2 * f0 * ta) \
             + np.exp(1j * 2 * np.pi * 2 * f0 * ta))
```

```
# Plot Fourier series and signal values
fig = plt.figure(figsize=(8,4))
plt.plot(ta, sa, 'b-', label='Fourier series')
plt.plot(t, s, 'ro', label='signal values')
plt.legend()
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```



6. Since $N = 5$ we only need to calculate the first $\lfloor (N+1)/2 \rfloor = \lfloor 6/2 \rfloor = 3$ rows (so the elements are $(e^{-i2\pi/3})^n$ where $n = 1, \ldots, 10$). Note that these are the same as those calculated in example 2.6.

$$
\begin{aligned}
e^{-i2\pi/5} &= \cos\left(\frac{2\pi}{5}\right) - i\sin\left(\frac{2\pi}{5}\right) = 0.31 - 0.95i, \\
(e^{-i2\pi/5})^2 &= (-0.81 - 0.95i)^2 = -0.81 + 0.59i, \\
(e^{-i2\pi/5})^3 &= e^{-i2\pi/5}(e^{-i2\pi/5})^2 = (0.31 + 0.95i)(-0.81 + 0.59i) = -0.81 - 0.59i, \\
(e^{-i2\pi/5})^4 &= (e^{-i2\pi/5})^2(e^{-i2\pi/5})^2 = (-0.81 + 0.59i)^2 = 0.31 + 0.95i, \\
(e^{-i2\pi/5})^6 &= e^{-i2\pi/5} = 0.31 - 0.95i, \\
(e^{-i2\pi/5})^8 &= (e^{-i2\pi/5})^3 = -0.81 + 0.59i.
\end{aligned}
$$

Calculating the DFT

$$
\begin{pmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0.31 - 0.95i & -0.81 - 0.59i & -0.81 + 0.59i & 0.31 + 0.95i \\ 1 & -0.81 - 0.59i & 0.31 + 0.95i & 0.31 - 0.95i & -0.81 + 0.59i \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -2 \\ 0 \\ -1 \end{pmatrix}
$$

$$
= \begin{pmatrix} -3 \\ 2 + 1.18i \\ 2 - 1.90i \\ \vdots \end{pmatrix}.
$$

We can approximate up to the $n = \lfloor (5-1)/2 \rfloor = 2$ harmonic so the complex coefficients are $c_0 = -3/5 = -0.6$, $c_1 = 0.4 + 0.24i$, $c_2 = 0.4 - 0.38i$, $c_{-1} = 0.4 - 0.24i$ and $c_{-2} = 0.4 + 0.38i$.

Calculating the values of the DC, A_n and ϕ_n for the amplitude-phase form of the Fourier series using

equations (18) and (19)

$$\begin{aligned}
\frac{a_0}{2} &= c_0 = -0.6, \\
A_1 &= 2|c_1| = 0.93, \\
\phi_1 &= -\operatorname{atan2}(-2\operatorname{Im}(c_1), 2\operatorname{Re}(c_1)) = -\operatorname{atan2}(-0.48, 0.8) = 0.53, \\
A_2 &= 2|c_2| = 1.1, \\
\phi_2 &= -\operatorname{atan2}(-2\operatorname{Im}(c_2), 2\operatorname{Re}(c_1)) = -\operatorname{atan2}(0.76, 0.8) = -0.76.
\end{aligned}$$

The fundamental frequency is $f_0 = 1/(5/2) = 2/5 = 0.4$so the exponential Fourier series that approximates this signal is

$$s(t) \approx -0.6 + 0.93\cos(0.8\pi t + 0.53) + 1.1\cos(1.6\pi t - 0.76).$$

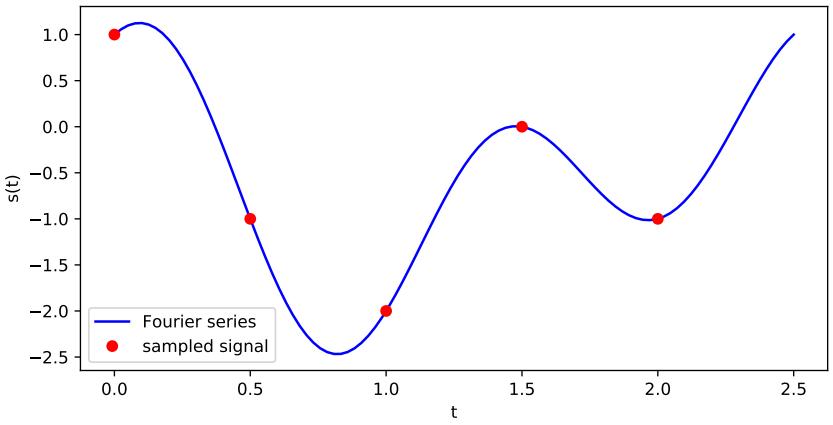


Figure B.1: Plot of the signal and Fourier series approximation.

7. (a) Here $N = 5$ so $P = 5/5 = 1$s. Calculate the IDFT (using the complex conjugate of the DFT matrix in question 6 and the symmetric property of the IDFT to fill in the missing rows)

$$s = \frac{1}{5}\begin{pmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 0.31 + 0.95i & -0.81 + 0.59i & -0.81 - 0.59i & 0.31 - 0.95i \\
1 & -0.81 + 0.59i & 0.31 - 0.95i & 0.31 + 0.95i & -0.81 - 0.59i \\
1 & -0.81 - 0.59i & 0.31 + 0.95i & 0.31 - 0.95i & -0.81 + 0.59i \\
1 & 0.31 - 0.95i & -0.81 - 0.59i & -0.81 + 0.59i & 0.31 + 0.95i
\end{pmatrix}$$

$$\begin{pmatrix}
4 \\
3.74 - 1.54i \\
-0.74 + 0.36i \\
-0.74 - 0.36i \\
3.74 + 1.54i
\end{pmatrix} = \begin{pmatrix}
2 \\
2 \\
0 \\
-1 \\
1
\end{pmatrix}$$

(b) The fundamental frequency is $f_0 = fs/N = 4/5 = 0.8$Hz and we can approximate up to the harmonic $n = \lfloor (5-1)/2 \rfloor = 2$. Dividing the DFT values by N to give c_n

$$\begin{aligned}
c_0 &= \frac{4}{5} = 0.8, \\
c_1 &= \frac{3.74 + 1.54i}{5} = 0.75 - 0.31i, \\
c_2 &= \frac{-0.74 - 0.36i}{5} = -0.15 + 0.07i,
\end{aligned}$$

so the Fourier series is

$$s(t) \approx 0.8 + (0.75 - 0.31i)e^{i2\pi t} + (0.75 + 0.31i)e^{-i2\pi t} + (-0.15 + 0.07i)e^{i4\pi t} + (-0.15 - 0.07i)e^{-i4\pi t}.$$

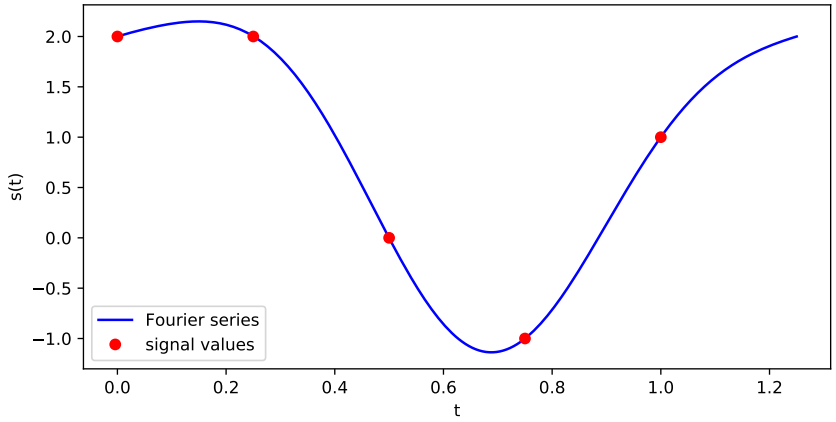
MATLAB:

```
% Define signal and F array
s = [2, 2, 0, -1, 1];
F = [ 4, 3.74 + 1.54i, -0.74 - 0.36i, -0.74 + 0.36i, 3.74 - 1.54i ]';
fs = 5;

% Calculate signal parameters
N = length(s);
T = 1 / fs;
L = N * T;
t = (0 : N - 1) * T;
f0 = fs / N;
c = F / N;

% Calculate Fourier series
ta = linspace(0, L, 200);
sa = c(1);
for n = 1 : 2
    sa = sa + c(n + 1) * exp(1i * 2 * pi * n * f0 * ta) ...
        + conj(c(n + 1)) * exp(-1i * 2 * pi * n * f0 * ta);
end

% Plot signal and fourier series
plot(ta, sa, 'b-')
hold on
plot(t, s, 'ro', 'MarkerFaceColor', 'r')
hold off
pbaspect([ 2, 1, 1 ])
axis padded
xlabel('t')
ylabel('s(t)')
legend('Fourier series', 'signal values', 'Location', 'southwest')
```

Python:

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

# Define signal and F array
s = np.array([2, 2, 0, -1, 1])
F = np.array([4, 3.74 - 1.54j, -0.74 + 0.36j, -0.74 - 0.36j, 3.74 + 1.54j])
fs = 4

# Calculate signal parameters
N = len(s)
T = 1 / fs
L = N * T
t = np.arange(N) * T
f0 = fs / N
c = F / N

# Calculate Fourier series
ta = np.linspace(0, L, 200)
sa = c[0]
for n in range(1, 3):
    sa += c[n] * np.exp(1j * 2 * np.pi * n * f0 * ta) \
        + np.conj(c[n]) * np.exp(-1j * 2 * np.pi * n * f0 * ta)

# Plot Fourier signal and signal values
```

```
fig, ax = plt.subplots(figsize=(8,4))
ax.plot(ta, sa.real, 'b-', label='Fourier series')
ax.plot(t, s, 'ro', label='signal values')
ax.legend(loc='lower left')
ax.set_xlabel('t')
ax.set_ylabel('s(t)')
plt.show()
```



8. (a) MATLAB:

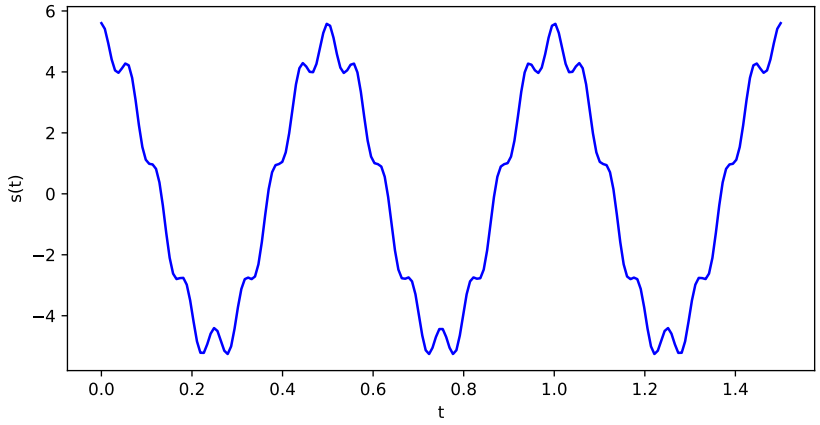
```
% Define signal period
P = 1.5;

% Calculate analogue signal
t = linspace(0, P, 200);
s = 5 * cos(4 * pi * t) + 0.6 * cos(32 * pi * t);

% Plot signal
plot(t, s, 'b-')
pbaspect([ 2, 1, 1 ])
axis padded
xlabel('t')
ylabel('s(t)')
```

Python:

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

# Define signal period
P = 1.5

# Calculate analogue signal
t = np.linspace(0, P, 200)
s = 5 * np.cos(4 * np.pi * t) + 0.6 * np.cos(32 * np.pi * t)

# Plot signal
fig = plt.figure(figsize=(8,4))
plt.plot(t, s, 'b')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()

# Output fundamental frequency
f0 = 1 / P
print('f0 = {:0.4f}Hz'.format(f0))
```

Output:

```
f0 = 0.6667Hz
```

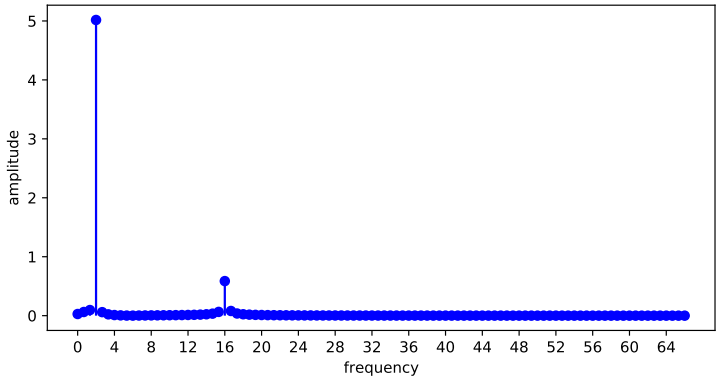


(b) MATLAB:

```
% Calculate FFT transform
F = fft(s);
N = length(s);
c = F / N;

% Calculate the frequency spectrum
nmax = floor((N - 1) / 2);
f = (0 : nmax) * f0;
A = abs([c(1), 2 * c(2:nmax + 1)]);

% Plot frequency spectrum
stem(f, A, 'b', 'LineWidth', 2, 'MarkerFaceColor', 'b')
pbaspect([ 2, 1, 1 ])
axis padded
xticks(f(1 : 6 : end))
xlabel('frequency')
ylabel('amplitude')
```

Python:

```
# Calculate FFT transform
F = np.fft.fft(s)
N = len(s)
c = F / N

# Calculate the frequency spectrum
nmax = (N - 1) // 2
f = np.arange(nmax + 1) * f0
A = abs(np.append(c[0], 2 * c[1:nmax+1]))

# Plot frequency spectrum
fig = plt.figure(figsize=(8,4))
plt.stem(f, A, 'b', basefmt=' ', markerfmt='bo', use_line_collection=True)
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.xticks(f[0::6])
plt.show()
```

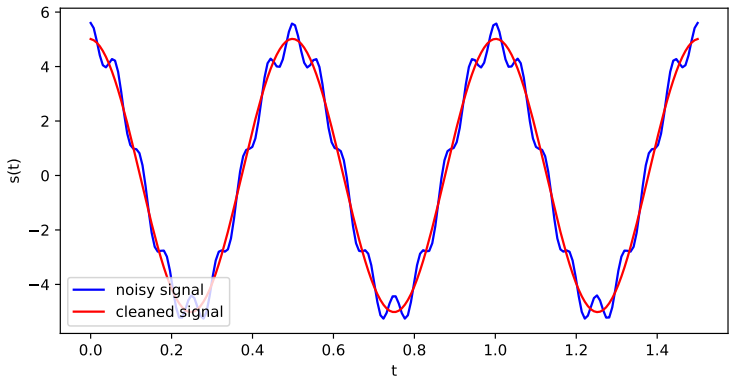


(c) MATLAB:

```
% Set harmonics with an amplitude less that 1 to zero
F(2 * abs(c) < 1) = 0;
c = F / N;

% Calculate IFFT
sclean = ifft(F);

% Plot the noisy signal and the cleaned up signal
plot(t, s, 'b-')
hold on
plot(t, sclean, 'r-')
hold off
pbaspect([ 2, 1, 1 ])
axis padded
xlabel('t')
ylabel('s(t)')
legend('noisy signal', 'cleaned signal', 'location', 'southwest')
```

Python:

```
# Set harmonics with an amplitude less than 1 to zero
F[2 * abs(c) < 1] = 0
c = F / N

# Calculate IFFT
sclean = np.fft.ifft(F)

# Plot noisy signal and the cleaned up signal
fig = plt.figure(figsize=(8,4))
plt.plot(t, s.real, 'b-', label='noisy signal')
plt.plot(t, sclean.real, 'r-', label='cleaned signal')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.legend(loc='lower left')
plt.show()
```


6
4
2
0
−2
−4
s(t)
noisy signal
cleaned signal
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
t

B.3 Digital Music

These are the solutions to the exercises from the Digital Music chapter which are on page 52.

1. MATLAB:

```
% Define signal parameters
L = 1;
fs = 44100;
f0 = 196;
N = L * fs;
T = 1 / fs;

% Define signal
t = (0 : N - 1) * T;
s = cos(2 * pi * f0 * t);

% Play signal
sound(s, fs)
```

Python:

```
import numpy as np
import IPython.display as ipd

# Define signal parameters
L, fs, f0 = 1, 44100, 196
N, T = L * fs, 1 / fs

# Define signal
t = np.arange(N) * T
s = np.cos(2 * np.pi * f0 * t)

# Play signal
ipd.Audio(s, rate=fs)
```

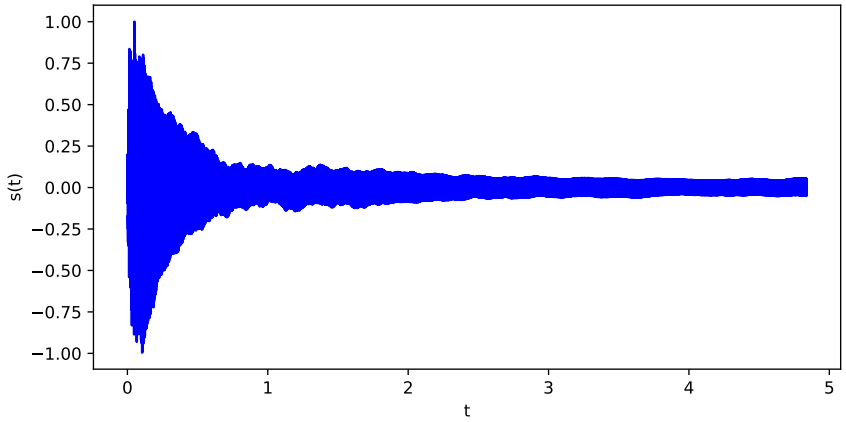
2. MATLAB:

```
% Read in audio file
[s, fs] = audioread('piano_note.wav');

% Calculate signal parameters
N = length(s);
T = 1 / fs;
t = (0 : N - 1) * T;

% Plot signal
plot(t, s, 'b')
pbaspect([2, 1, 1])
axis padded
xlabel('t')
ylabel('s(t)')
```

Python:

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

# Read in audio file
s, fs = sf.read('piano_note.wav')

# Calculate signal parameters
N, T = len(s), 1 / fs
t = np.arange(N) * T
```

```
# Plot signal
fig = plt.figure(figsize=(8, 4))
plt.plot(t, s, 'b')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```

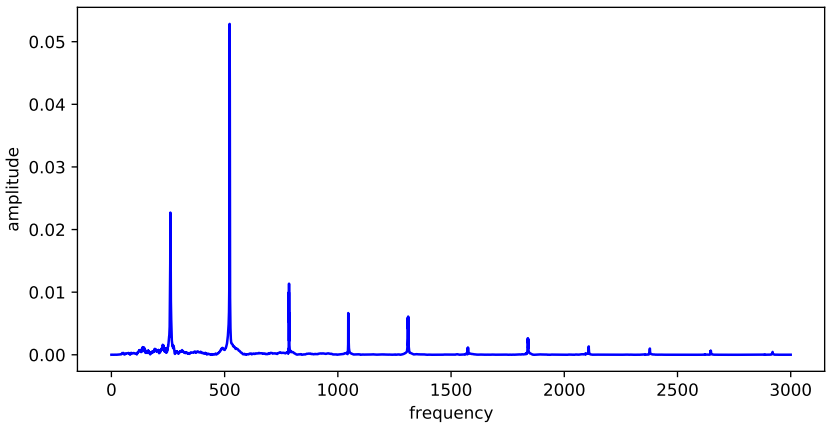


3. MATLAB:

```
% Calculate the FFT
F = fft(s);
c = F / N;

% Calculate amplitude and frequency
f0 = fs / N;
nmax = floor((N - 1) / 2);
f = (0 : nmax) * f0;
A = abs([c(1) ; 2 * c(2 : nmax + 1)]);

% Plot frequency spectrum
plot(f(f <= 3000), A(f <= 3000), 'b')
pbaspect([2, 1, 1])
axis padded
xlabel('frequency')
ylabel('amplitude')
```

Python:

```
# Calculate the FFT
F = np.fft.fft(s)
c = F / N

# Calculate amplitude and frequency
f0 = fs / N
nmax = (N - 1) // 2
f = np.arange(nmax + 1) * f0
A = abs(np.append(c[0], 2 * c[1:nmax+1]))

# Plot frequency spectrum
fig = plt.figure(figsize=(8,4))
plt.plot(f[f <= 3000], A[f <= 3000], 'b')
plt.xlabel('frequency')
plt.ylabel('amplitude')
plt.show()
```



4. MATLAB:

```
% Identify frequencies of the harmonics (this could be done by
% inspecting the frequency spectrum)
[~, i] = findpeaks(A, 'MinPeakProminence',0.002,'MinPeakDistance',100);
freq = f(i);
fprintf(['freq = ', sprintf(repmat('%1.2f, ', 1, length(freq)), freq)])
```

Python:

```
# Identify frequencies of the harmonics (this could be done by
# inspecting the frequency spectrum)
import scipy.signal as sp

peaks, _ = sp.find_peaks(A, height=0.002, distance=200)
freq = f[peaks]
print('frequencies = ', freq.round(2))
```

```
frequencies =  [ 260.39  522.02  784.48 1045.9  1310.63 1838.85]
```

The fundamental frequency for this signal is 260.39Hz so this note is C_4.

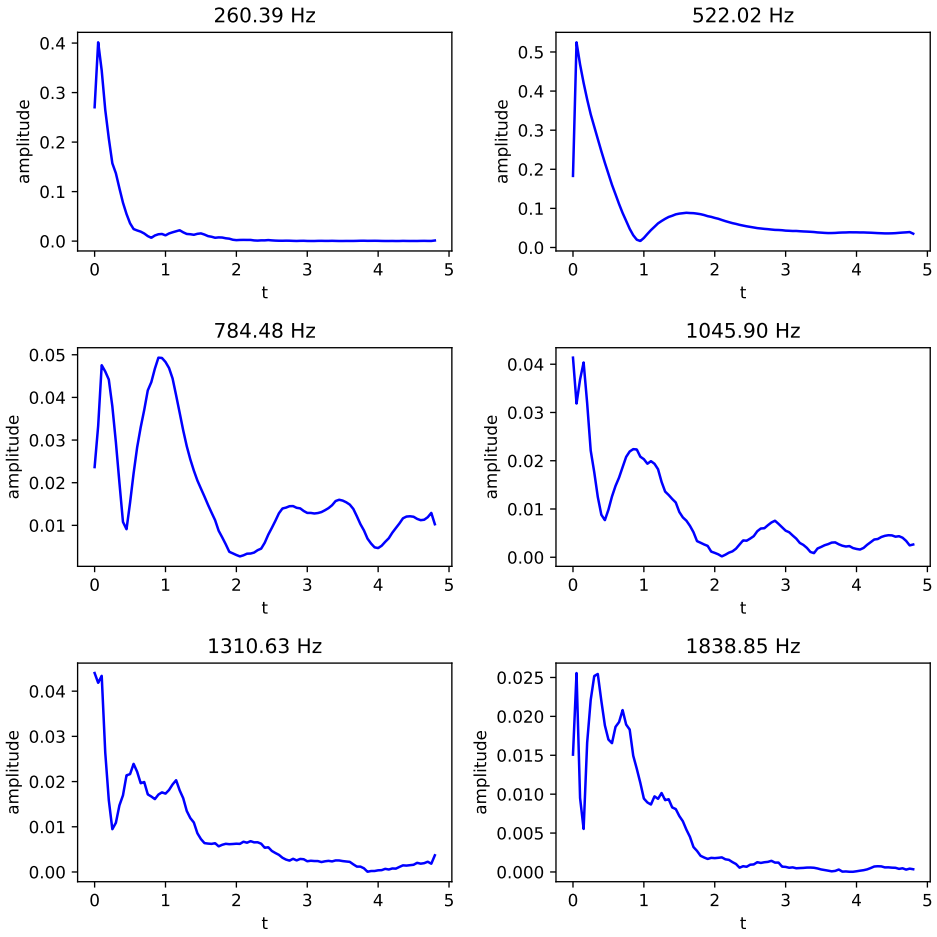
5. MATLAB:

```
w = 0.05;
L = T * N;
nframes = floor(L / w) + 1;
nfreq = length(freq);
time = (0 : nframes - 1) * w;
A = zeros(nfreq, nframes);

for harmonic = 1 : nfreq
    for frame = 0 : nframes - 1

        % Extract frame
        sf = s(t >= frame * w & t < frame * w + w);
        N = length(sf);

        % Calculate FFT
        F = fft(sf);
        c = F / N;

        % Calculate amplitude for current harmonic
        f0 = fs / N;
        nmax = floor((N - 1) / 2);
        f = (0 : nmax) * f0;
        [~, i] = min(abs(f - freq(harmonic)));
```

```
        A(harmonic, frame + 1) = 2 * abs(c(i));

    end

    % Plot the amplitude for the current harmonic
    subplot(floor(nfreq / 2), 2, harmonic)
    plot(time, A(harmonic, :), 'b')
    axis padded
    xlabel('t')
    ylabel('amplitude')
    title(sprintf('%1.2fHz', freq(harmonic)))

end
```

Python:

```
w = 0.05
L = T * N
nframes = int(L / w) + 1
nfreq = len(freq)
time = np.arange(nframes) * w
A = np.zeros((nfreq, nframes))

fig, ax = plt.subplots(nfreq // 2, 2, figsize=(8, 8))
for harmonic in range(nfreq):
    for frame in range(nframes):

        # Extract frame
        sf = s[(t >= frame * w) & (t < frame * w + w)]
        N = len(sf)

        # Calculate FFT
        F = np.fft.fft(sf)
        c = F / N

        # Calculate amplitude for current harmonic
        f0 = fs / N
        nmax = (N - 1) // 2
        f = np.arange(nmax + 1) * f0
        i = np.argmin(abs(f - freq[harmonic]))
        A[harmonic,frame] = 2 * abs(c[i])

    # Plot the amplitude for the current harmonic
    i, j = harmonic // 2, harmonic % 2
    ax[i,j].plot(time, A[harmonic,:].real, 'b')
    ax[i,j].set_title('{:0.2f} Hz'.format(freq[harmonic]))
    ax[i,j].set_ylabel('amplitude')
    ax[i,j].set_xlabel('t')

plt.tight_layout()
```



6. MATLAB:

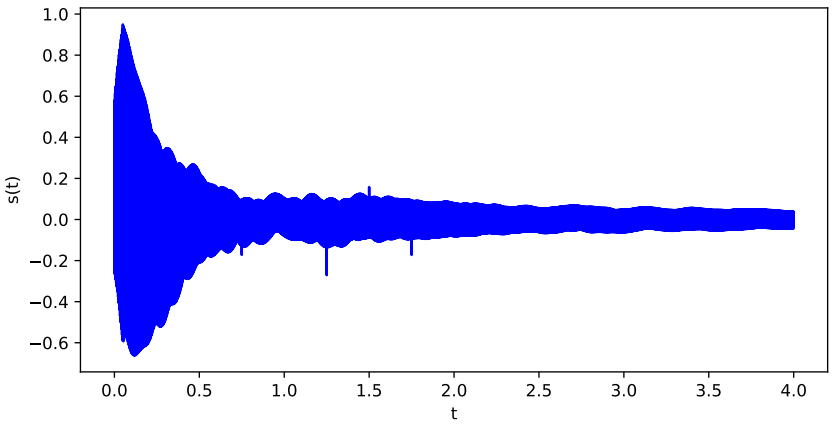
```
L = 4;
fs = 44100;
N = L * fs;
T = 1 / fs;
t = (0 : N - 1) * T;
sd = zeros(1, N);

% Calculate Fourier series for digital note
for harmonic = 1 : nfreq
    for frame = 0 : nframes - 1
        i = t >= frame * w & t < frame * w + w;
        a = interp1(time, A(harmonic, :), t(i));
        sd(i) = sd(i) + a .* cos(2 * pi * freq(harmonic) * t(i));
    end
end

% Plot signal
clf
plot(t, sd, 'b')
pbaspect([2, 1, 1])
axis padded
xlabel('t')
ylabel('s(t)')
```

```
% Play signal
sound(sd, fs)
```

Python:

```
L, fs = 4, 44100
N = L * fs
T = 1 / fs
t = np.arange(N) * T
sd = np.zeros(N)

# Calculate Fourier series for digital note
for harmonic in range(nfreq):
    for frame in range(nframes):
        i = (t >= frame * w) & (t < frame * w + w)
        a = np.interp(t[i], time, A[harmonic,:])
        sd[i] += a * np.cos(2 * np.pi * freq[harmonic] * t[i])

# Plot signal
fig = plt.figure()
plt.plot(t, sd, 'b')
plt.xlabel('t')
plt.ylabel('s(t)')
plt.show()
```

```
# Play audio
ipd.Audio(sd, rate=fs)
```



Listings

Index