

---

# Chapter 4

## Systems

---

### Essential reading

Eaton, J.W. *GNU Octave Manual*. (Bristol: Network Theory, 1996) [ISBN 0954161726]. (This is also available online in HTML form at <http://www.gnu.org/software/octave/doc/interpreter> and in texinfo source format in the Octave source code distribution.)

---

### Recommended reading

Oppenheim, A.V. and A.S. Willsky with S. Hamid Nawab *Signals and Systems*. (Upper Saddle River, N.J.; London: Prentice Hall, 1997) [ISBN 0138147574].

---

## 4.1 Introduction

The previous chapter discussed how increasingly complex signals can be constructed by summing fundamental signal building blocks called sinusoids. Digital signal processing is concerned with both the construction and transformation of signals to make new signals. The transformation processes are made from combinations of fundamental transformation building blocks; collectively, these transformations are called **systems**.

A system **responds** to an input signal to produce a new output signal. Examples of input signals and system responses include:

- the electromagnetic spectrum and the radio
- the hammer and the church bell
- the electric current and the dimmer switch
- the bow on the string and the violin body
- the bump on the road and the car's suspension
- the bat's chirp and the echo from an insect
- the lightning strike and the rolling thunder
- the glottis and the vocal tract (speech)
- the baseball on the bat and the 'pop'
- the penny dropped and the resulting ring.

In all of these examples an initial signal is transformed into another signal via a system. The system can be simple, as in the case of the dimmer switch which is a variable resistor, or very complex as in the case of rolling thunder (which is due to the acoustic properties of air and the geography and architecture of the region of the lightning strike).

## 4.2 LTI systems

Much of DSP is concerned with the mathematical analysis of the properties of systems; as such there are a very wide variety of systems and classes of systems that occupy different branches of mathematics. Examples are linear systems, non-linear systems, linear dynamical systems, non-linear dynamical systems and chaotic systems.

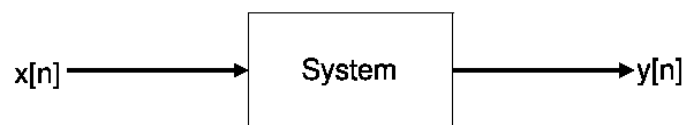
Of all the classes of system the most widely used are linear time-invariant (LTI) systems. LTI systems are characterised by three properties: linearity, time-invariance and complete characterisation by an impulse response.

In mathematical notation a system is represented as a function,  $T\{\cdot\}$ , that acts on an input signal,  $x[n]$  to produce an output signal  $y[n]$ :

$$y[n] = T\{x[n]\}$$

The output signal,  $y[n]$  is called the **system response** to system  $T\{\cdot\}$  acting on input signal  $x[n]$ . The independent variable,  $n$ , represents a discrete-time index; so the above notation means that at each discrete time instant,  $n$ , the signal  $y[n]$  at that instant is generated by the action of a system  $T\{\cdot\}$  acting upon an input signal  $x[n]$ . Using this notation the following sections describe the properties of LTI systems in greater detail.

Figure 4.1 shows a schematic diagram of a system. The input signal, the system and the output signal are shown as a cascading circuit with a box in the middle representing the system.



**Figure 4.1:** A system block diagram showing the flow of a signal from input,  $x[n]$ , through the system to produce the output signal,  $y[n]$ .

### 4.2.1 Linearity

A system,  $T\{\cdot\}$ , is said to be linear if it satisfies the property of **scaling**:

$$aT\{x[n]\} = T\{ax[n]\}$$

This means that scaling the system response to an input signal produces the same output as scaling the input signal and **then** applying the transformation. The order of the operations does not affect the result.

Linear systems also obey the property of superposition for sums of signals:

$$aT\{x[n]\} + bT\{y[n]\} = T\{ax[n] + by[n]\}$$

This property demonstrates one of the main utilities of linear systems; there are two transformations on the left-hand side of the equation and only one transformation on the right-hand side. Linear systems allow algebraic manipulation of signals and systems to find a more efficient way to implement the transformations. This principal is very important to the field of digital signal processing because, often, the goal is to perform complex tasks with limited computational means.

#### 4.2.2 Time invariance

The property of time invariance means that a system responds in the same manner to its input at all instants in time. So there is no dependence on the variable  $n$  in the action of the system.

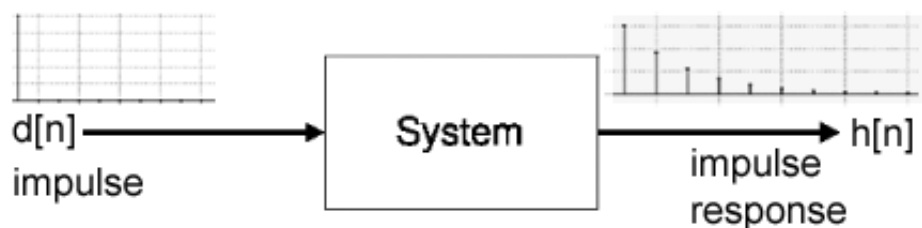
Formally, this concept is notated by:

$$y[n] = T\{x[n + d]\} \rightarrow y[n - d] = T\{x[n]\} \forall n$$

#### 4.2.3 Impulse response

LTI systems are completely characterised by their **impulse response**. The impulse response is the system response given the delta function (the impulse) as input.

By convention, we call the impulse signal  $d[n]$  and the impulse response of a system  $h[n]$ . Figure 4.2 shows the system block diagram of an impulse response.



**Figure 4.2:** The impulse response of a system is the system response to the delta signal input. The delta signal is called the unit impulse. The impulse response completely characterises an LTI system.

The fact that an impulse response characterises an LTI system means that once we know the impulse response of a system, we know **exactly** how the system will respond to **any** signal. The impulse response is a precise description of an LTI system.

This fact means that we can implement an LTI system by transforming a signal with the impulse response of a system. The operation that transforms a signal using an

impulse response is called **convolution**.

The impulse response is itself a discrete-time signal. This means that systems are implemented by convolving two signals. We will now discuss the convolution operator in more detail.

#### 4.2.4 Convolution

Convolution is represented by a special operator notated as  $*$ . This is not to be confused with the multiplication operator that uses the same symbol in most computer programming languages:

$$y[n] = x[n] * h[n]$$

The convolution operator implements the convolution sum, which is a widely-used operation in mathematics:

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

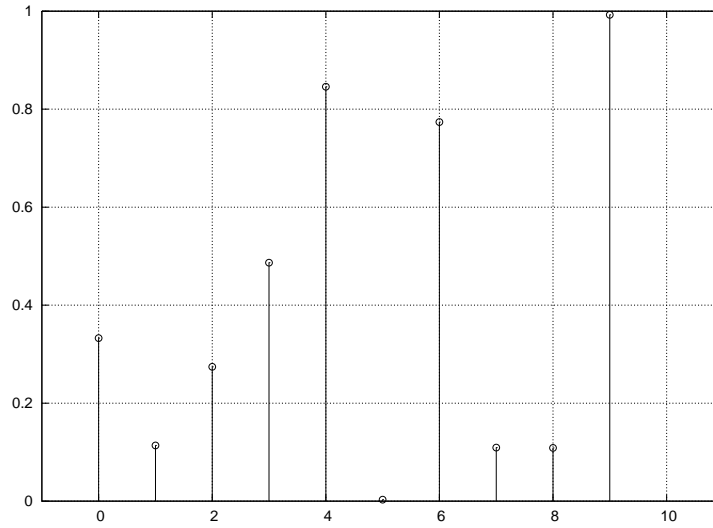
This notation informs us that the output at each time instant,  $n$ , is formed by adding the impulse response,  $h[n]$  to the current output time location,  $y[n]$  and scaling it by the current input value  $x[n]$ .

In Octave, the convolution operator is provided by a function call, `conv()`. The following example shows the use of the convolution operator to transform a uniform random signal, with length 10, using an impulse response of length 2:

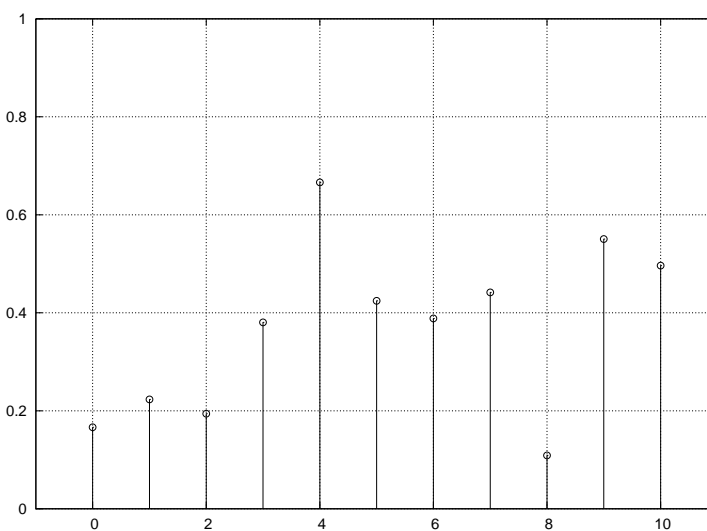
```
octave:>x = rand(1,10); % A random signal of length 10
octave:>h = [0.5 0.5]; % An impulse response
octave:>y = conv(x, h);% Convolution operation
octave:>subplot(211)
octave:>stem(0:length(x)-1, x, '*')
octave:>axis([-1 length(y) 0 1])
octave:>subplot(212)
octave:>stem(0:length(y)-1,y, '*')
octave:>axis([-1 length(y) 0 1])
```

The length of the input signal  $x$  is 10 and the impulse response  $h$  has length 2 in this case. The convolution operation produces a signal that is  $\text{length}(x) + \text{length}(h) - 1$ , so the length of the output signal  $y$  is therefore 11.

The signal in Figure 4.3 is transformed to that of Figure 4.4 via the system with impulse response  $h = [0.5 \ 0.5]$ . We often call the process of convolving a signal with an impulse response filtering, because it helps us to extract specific parts of a signal. In this case the system is a two-point averaging filter. As we shall see in the following chapter, this corresponds to a low-pass filter which means that high frequencies present in the signal are attenuated and low frequencies are passed through unchanged.



**Figure 4.3:** A signal of length 10 generated using Octave's `rand()` function.



**Figure 4.4:** The signal of Figure 4.3 convolved with the two-point impulse response  $[0.5 \ 0.5]$ . The convolved signal has length  $\text{length}(x) + \text{length}(h) - 1$  which is 11 samples.

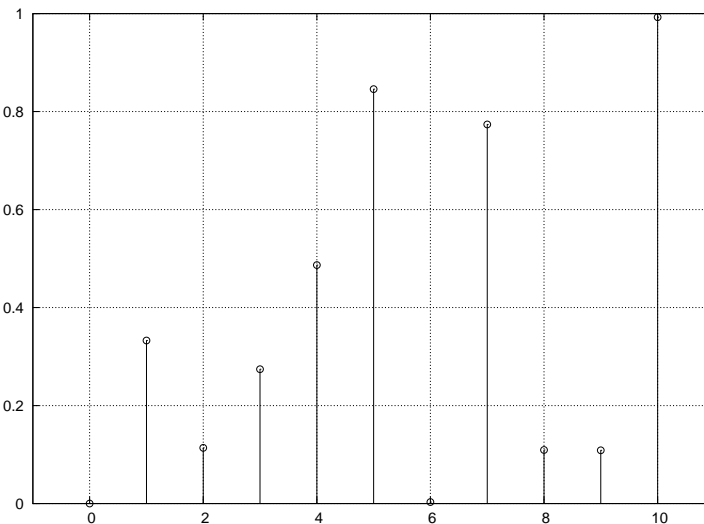
## 4.2.5 Unit impulse and unit delay systems

Convolution with the unit impulse produces the identity. This means that the signal is passed through the system unchanged regardless of its frequency content.

Convolution with a unit delay produces a delayed signal; the signal is passed through shifted by the delay time of the impulse:

```
octave:>y = conv(x, [0 1]); % Unit Delay signal
stem(0:length(y)-1, y, '*');
axis([-1 length(y) 0 1])
```

In this example the zero-time reference index is 1. So the signal is shifted by one sample relative to vector index 1.



**Figure 4.5:** The signal of Figure 4.3 convolved with a unit delay [0 1].

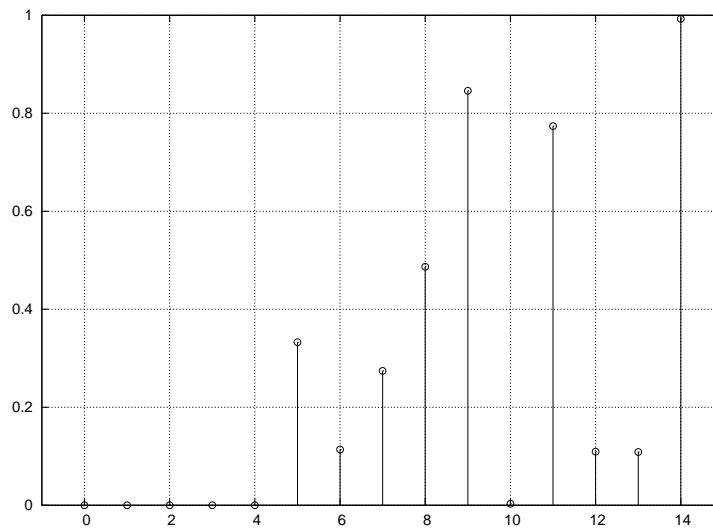
To delay the signal by multiple samples, we can either append more zeros to the beginning of the unit delay, thus shifting the signal further to the right, or we can re-convolve the signal with the unit delay. Convolution of the output signal with the unit delay  $N$  times produces a delay of  $N$  samples:

```
y = conv(x, [0 0 0 0 0 1]); % delay by five samples
stem(0:length(y)-1,y,'*');
axis([-1 length(y) 0 1])
```

Or equivalently:

```
y = conv(x, [0 1]);
y = conv(y, [0 1]);
y = conv(y, [0 1]);
y = conv(y, [0 1]);
y = conv(y, [0 1]);
stem(0:length(y)-1,y,'*');
axis([-1 length(y) 0 1])
```

These examples show that the convolution operation can be re-applied multiple



**Figure 4.6:** The signal of Figure 4.3 convolved multiple times with a unit delay  $[0 \ 1]$ . The result is a signal shifted by multiple samples; in this case five samples.

times; with each iteration taking the previous output and feeding it back into the system for the next input. This process of applying a system multiple times to a signal, via the system output, is called composition.

We can also compute system composition as a set of nested function calls in Octave:

```
d = [0 1]; % The unit delay
y = conv(conv(conv(conv(conv(x, d), d),
d), d), d);
```

## 4.2.6 Scaled delay

By the linearity of LTI systems, a signal can be scaled by multiplying the input by a scalar, multiplying the output by a scalar or multiplying the impulse response by a scalar. The following examples produce the same signal as output:

```
y1 = conv(0.5 * x, [0 1]);
y2 = conv(x, [0 0.5]);
y3 = 0.5 * conv(x, [0 1]);
```

You should verify that the three signals  $y_1$ ,  $y_2$  and  $y_3$  are the same.

## 4.2.7 Convolution revisited

Given the properties of linearity and superposition we discussed above, there is an alternative way to think of the convolution operation.

The impulse response can be thought of as consisting of delayed, scaled, unit impulses. For example, the impulse response  $[0.5 \ 0.25 \ 0.125 \ 0.125]$  consists of four scaled and shifted impulses. The first is scaled by 0.5 and shifted by zero samples. The second is scaled by 0.25 and shifted by one sample, the third is scaled by 0.125

and shifted by two samples, and the fourth is also scaled by 0.125 but shifted by three samples.

By the property of superposition, convolution can be calculated as shifting and scaling the input signal by each scaled and shifted impulse in the impulse response and summing the four system responses:

```
y = conv(x, [0.5 0 0 0]);
y = y + conv(x, [0 0.25 0 0]);
y = y + conv(x, [0 0 0.125 0]);
y = y + conv(x, [0 0 0 0.125]);
```

You should verify that the output of this set of operations is the same as convolution of the input signal with the impulse response [0.5 0.25 0.125 0.125].

This example illustrates a very important property of convolution, namely that the operation can be broken into a number of very simple steps:

- delay the signal
- scale the signal
- sum it with the output signal.

It was stated above that the output of the convolution operation is a signal that has length  $\text{length}(x) + \text{length}(h) - 1$ . For an input signal of length 10 and an impulse response of length 4 the resulting output will be length 13. We can produce the convolution operation without the use of the `conv()` function by padding the input signal with zeros at the end so that it is the correct length for the convolution operation, then performing the simple operations of delay and scale. For the convolution of the signal  $x$  with  $h = [0.5 \ 0.25 \ 0.125 \ 0.125]$  the signal needs to be padded with  $\text{length}(h) - 1 = 3$  zeros:

```
x1 = [x zeros(1,3)];
```

Now the convolution operation can be expressed as a sequence of delay, scale and sum operations on the input signal:

```
y = x1 * 0.5; % The first impulse
y = y + 0.25 * shift( x1, 1 ); % the second impulse
y = y + 0.125 * shift( x1, 2 ); % the third impulse
y = y + 0.125 * shift( x1, 3 ) % the fourth impulse y =
Columns 1 through 8:
0.16641 0.14002 0.20713 0.36773 0.59303 0.30806 0.55423 0.35422
Columns 9 through 13:
0.17872 0.63382 0.27538 0.13764 0.12407
```

These examples illustrate that the convolution operation is the result of a set of fundamental signal processing operations: scale, shift and sum. The properties of linearity and superposition are key to the ability to represent complex transforms as sequences of such simple unit transforms.

The prevalence of DSP in the world is due in large part to the ability to implement such systems out of these elementary building blocks.

## 4.3 Spectral analysis

### 4.3.1 Complex exponentials

The larger part of DSP theory is concerned with the representation of signals and systems as complex numbers. The reason is that many of the properties of systems are best represented in the complex domain rather than in the domain of real numbers.

Complex numbers are easily handled by Octave. A complex number is one that has both a real and an imaginary component. The real part is simply a real number and the imaginary part is a real number multiplied by  $\sqrt{-1}$ , the square-root of  $-1$ , denoted by  $i$ . Complex numbers, therefore, are written down as two parts:

$$a + ib$$

In Octave we can generate complex numbers by taking the square root of a negative number; or by multiplying a real number by the square root of  $-1$ :

```
octave:>sqrt(-1)
ans = 0 + 1i
octave:>(-1)^0.5
ans = 0 + 1i % If this says NaN then your version of Octave is out of date
octave:> 2 + 20i
ans = 2 + 20i
```

#### Complex arithmetic

The operation of summing two complex numbers involves summing the real parts and the imaginary parts separately:

$$(a + ib) + (c + id) = a + c + i(b + d)$$

In octave we perform addition on complex numbers in the same way as ordinary numbers:

```
3 + 4i + -2 + 3i
ans = 1 + 7i
```

The operation of subtracting two complex numbers requires the subtraction of the real parts and the imaginary parts as separate operations:

$$(a + ib) - (c + id) = a - c + i(b - d)$$

```
(3 + 4i) - (-2 + 3i)
ans = 5 + 1i
```

Note in the last example we were careful to use parentheses around each complex number. What happens if we don't use parentheses? Why does this happen?

Finally, multiplication of complex numbers follows the standard algebraic operation of multiplication, but with the complex element  $i = \sqrt{-1}$ :

$$(a + ib)(c + id) = ac - bd + i(ad + cb)$$

when multiplying complex numbers in octave the individual complex numbers must also be surrounded by parentheses:

```
(3 + 4i) * (-2 + 3i)
ans = -18 + 1i
```

These examples illustrate that arithmetic operations on complex numbers result in other complex numbers.

### Polar form of complex numbers

When using complex numbers it is often convenient to convert the rectangular coordinates  $(a, ib)$  to polar form consisting of a magnitude and phase component. This is performed with the following relations:

$$r = \text{abs}(a + ib) = |a + ib| = \sqrt{a^2 + b^2}$$

$$\theta = \text{angle}(a + ib) = \angle(a + ib) = \text{atan}\left\{\frac{b}{a}\right\}$$

Octave conveniently provides the `abs()` and `angle()` functions to convert rectangular complex coordinates into polar form:

```
octave:>abs(2 + 3i) ans = 3.6056 octave:>angle(2 + 3i) ans = ans =
0.98279
```

The polar representation can be converted back to complex coordinates using the relations:

$$a = r\cos(\theta)$$

$$b = r\sin(\theta)$$

The resulting complex number is  $a + ib$ :

```
octave:>x = 2 + 3i % define a complex number variable x
x = 2 + 3i octave:>r = abs(x) % take the magnitude
ans = 3.6056 octave:>theta = angle(x) % take the phase (angle)
ans = 0.98279 octave:>a = r*cos(theta) % get the real part from r and
theta
ans = 2.0 octave:>a = r*sin(theta) % get the imaginary part from r and
theta
ans = 3.0 octave:>a+ib % form the complex number
ans = 2.0000 + 3.0000i % back where we started
```

### Complex exponential function

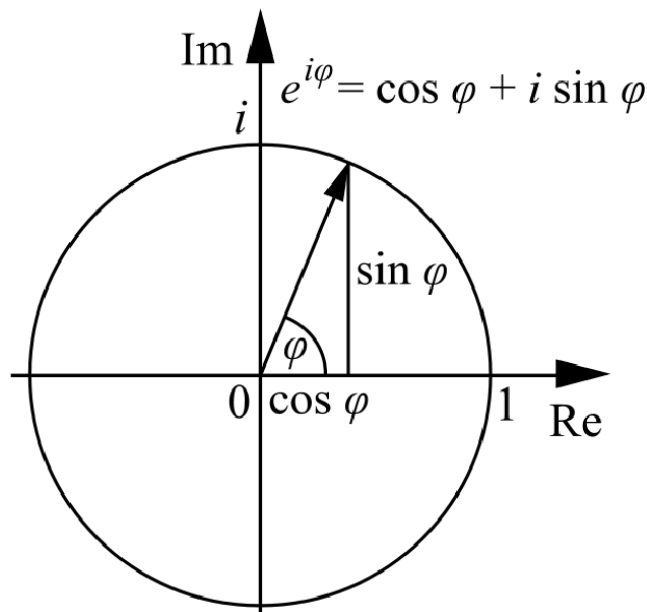
The Euler number  $e$  is the irrational 2.71828182845905... often truncated to 2.7183. This is the natural exponent; it has the special property that the slope of the curve  $e^x$  at a point  $x$  is  $e^x$  for all values of  $x$ .

One of the relations that is fundamental to DSP is that of the exponential function with a complex argument:

$$e^{ix} = \cos(x) + i\sin(x)$$

This is called Euler's formula due to its creator, Leonard Euler. In this formula, the complex exponential is understood as the sum of a real cosine and an imaginary sine value. If we set the sine value to 0 we see that all real sinusoids are actually complex exponentials, which is of profound importance to DSP and is the reason why DSP theory is full of complex exponentials.

Euler's formula can be visualised as the circumference of a unit circle in the **complex plane**. The complex plane is a two-dimensional graph with the x-axis representing the real axis and the y-axis representing the imaginary axis. Euler's formula is illustrated in the complex plane in Figure 4.7.



**Figure 4.7:** Euler's formula represented as a circle in the complex plane. The x-axis is the real part and the y-axis the imaginary part of the complex number represented by  $e^{i\theta}$ . The circle has radius 1 therefore it is called the unit circle.

One remarkable result of this property of the exponential function is the identity:

$$e^{i\pi} = -1$$

following from the relation above. This is called Euler's identity.

If we expand the identity:

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

we can see the derivation  $\cos(\pi) = -1$  and  $\sin(\pi) = 0$  hence  $e^{i\pi} = -1 + 0 = -1$ .

By re-arranging this equation we have a relationship between the five most important numbers in mathematics:  $e, i, \pi, 0$  and  $1$ :

$$e^{i\pi} + 1 = 0$$

We can see this in Octave using the built-in `exp()` function and the constants `i` and `pi`:

```
octave:>exp(i*pi)
ans = -1.0000e+00 + 1.2246e-16i
```

Note that the imaginary part is  $1.2246e - 16i$  which is  $1.2246 \times 10^{-16}i$ , a very small number that is essentially zero. The imaginary part is not exactly zero due to floating point roundoff error.

Often we wish to represent a real sinusoid using complex exponential notation. How can we make the exponential function generate a sinusoid that has only an imaginary part?

To do this we use the relation:

$$\cos(x) = \frac{A}{2}\{\cos(x) + i\sin(x)\} + \frac{A}{2}\{\cos(x) - i\sin(x)\}$$

This says that the cosine function can be understood as two complex exponentials, one with a positive imaginary part and the other with a negative imaginary part. By the process of summation for two complex numbers the two complex exponentials combine to make a single real cosine function. In complex exponential notation we write:

$$\cos(x) = \frac{A}{2}e^{ix} + \frac{A}{2}e^{-ix} = \frac{A(e^{ix} + e^{-ix})}{2}$$

Here we see that the real cosine function is the sum of two complex exponentials, one with a positive exponent and the other with a negative exponent with the same complex argument. We can construct sinusoids using this relation in Octave:

```
octave:>x=[0:pi/8:pi]
x =
Columns 1 through 8:
0.00000 0.39270 0.78540 1.17810 1.57080 1.96350 2.35619 2.74889
Column 9:
3.14159
octave:> cos(x)
ans =
Columns 1 through 5:
```

```

1.0000e+00 9.2388e-01 7.0711e-01 3.8268e-01 6.1232e-17
Columns 6 through 9:
-3.8268e-01 -7.0711e-01 -9.2388e-01 -1.0000e+00
octave:>exp(i*x)/2 + exp(-i*x)/2
ans =
Columns 1 through 5:
1.0000e+00 9.2388e-01 7.0711e-01 3.8268e-01 6.1232e-17
Columns 6 through 9:
-3.8268e-01 -7.0711e-01 -9.2388e-01 -1.0000e+00

```

### 4.3.2 Signal multiplication by complex exponentials

A very useful property of complex exponentials comes when they are multiplied by a signal. The product of a complex exponential and a signal is a measure of the amplitude and phase of the signal at the frequency of the complex exponential.

To illustrate this principle we can construct a signal in Octave and measure the presence of each frequency component by vector multiplication with a complex exponential at a given analysis frequency.

```

octave:>sr = 44100;
octave:>t = [ 0:400 ] / sr;
octave:>x = sin(2*pi*441*t) + sin(2*pi*882*t);
octave:>h = exp(i*2*pi*1000*t);
octave:>h*x'
ans = -3.0453 + 13.5648i
octave:>abs(h*x')
ans = 13.0902
octave:>h = exp(i*2*pi*882*t);
octave:>abs(h*x')
ans=200.00
octave:>angle(h*x')
ans = 1.5708

```

In this example we have constructed an audio signal consisting of one cycle of a 441Hz Helmholtz tone with two frequency components; one at the fundamental frequency and one at the second harmonic. We then constructed a complex exponential with a frequency of 1000Hz and computed the vector dot product (the vector multiplication seen in Chapter 3 of this guide) between the complex exponential and the Helmholtz tone and measured the magnitude. The resulting magnitude value is 13.902. Then we made a new complex exponential with frequency 882Hz corresponding with one of the frequencies in the Helmholtz tone; we took the vector dot product between the complex exponential and the Helmholtz tone and the resulting magnitude was 200.

Multiplication between the Helmholtz tone and the complex exponential produces a much higher magnitude value for frequencies that are present in the Helmholtz tone than when we multiply by a complex exponential having a frequency that is not present in the Helmholtz tone.

Hence, multiplication by complex exponentials is a way to analyse the frequency content of signals. This is a very widely used and important concept within DSP and you should spend time familiarising yourself with it. The learning activity which follows is an illustration of this principle for audio signals.

---

### Learning activity

- Make a complex tone audio signal of 441 sample duration by summing four sine waves with amplitudes of 0.25 and frequencies of 441Hz, 882Hz, 1323Hz and 1764Hz.
- Now make a new sinusoid with amplitude 1 and frequency 1000Hz. Calculate the dot product of this sinusoid with your complex tone using vector multiplication between the heterodyne and the transposed complex tone signal. (Find out what is meant by the term *heterodyne*.)
- Do the same for sinusoids with the following frequencies:
  - 441Hz
  - 882Hz
  - 1323Hz
  - 1764Hz.
- What do you notice about the value returned for 1000Hz versus the other frequency values?
- Repeat the above but for an inverted sinusoid (i.e.  $\pi$  radians phase offset).
- What do you notice about the value returned for the inverted sinusoid for each of the signals?
- Make a complex exponential at frequency 1000Hz and perform the same dot product on your complex tone as above. What is the magnitude of the resulting value? Hint: use `abs()`.
- What is the angle (phase) of the resulting value?
- Make a complex exponential at frequency 1323Hz and perform the dot product on your complex tone. What is the magnitude of the resulting value?
- What is the angle (phase) of the resulting value?

---

The process of multiplying a signal by a complex exponential yields two values: a magnitude and a phase of any sinusoidal component present in the complex tone. The magnitude measures the amount of the given frequency that is present in the signal and the phase measures the phase offset for a sinusoid at the given frequency to be present in the signal.

This operation of multiplication by a complex exponential is the basis for Fourier theory which measures the **spectrum** of signals.

### 4.3.3 Spectra of signals and systems

The spectrum of a signal is an analysis of the frequency content of the signal. As we saw in the last chapter, any signal can be constructed out of a set of sinusoidal components at the right amplitudes, frequencies and phases.

The process of extracting the sinusoidal parameters from a signal is called spectral analysis and it is most often performed using Fourier analysis. Fourier analysis multiplies the signal by a set of complex exponentials at different frequencies, providing estimates of the amplitudes (magnitudes) and phase offsets for each sinusoidal component.

To plot a spectrum we put frequency on the x-axis and make exponentials at a number of equally-spaced intervals starting at 0Hz and extending up to the highest frequency value in our signal. When we perform a Fourier analysis on a discrete-time

signal it is called a discrete Fourier transform (DFT). A magnitude spectrum represents the relative strengths of the sinusoidal components in our signal.

We can re-construct a signal by summing together sinusoids containing the relative strengths provided by the Fourier analysis. This reverse process is known as Fourier synthesis.

#### 4.3.4 Fast Fourier Transform (FFT)

The more densely we sample the frequency axis, the more time it takes to perform the Fourier analysis. For long signals a Fourier analysis can take more time than we have available. Fortunately, a clever algorithm was invented by Cooley and Tukey in the 1960s that performs a discrete Fourier transform much faster than the `for`-loop above.

Octave provides a function to compute the fast Fourier transform of a signal called `fft()`. This function takes a signal as an argument and performs the multiplication by a series of complex exponentials yielding a complex valued spectrum. We take the magnitude of the resulting spectrum to find the amplitudes of the sinusoidal components and we take the angle of the resulting spectrum to find the relative phase offsets.

##### Spectrum of a sine wave

Figure 4.8 shows the spectrum of a sine wave. The Octave code that generated the figure is:

```
t = [0:1/1000:1]; % 1s at 1000Hz sample rate
x = sin(2*pi*100*t); % 100Hz sine wave
n = 10; % how many frequency samples to take
stem([0:2/n:2-2/n],abs(fft(x(1:n))),'*')
axis([0 2 0 6])
```

This example first constructs a vector of time-points at which to sample a sine wave. The sampling rate is 1000Hz. Then a signal,  $x$ , is constructed by sampling a sine wave with frequency 100Hz which is angular frequency  $2\pi \cdot 100$ Hz. We define  $n$  the number of samples of the waveform to use to compute frequency estimates; this will be the number of frequency samples that are calculated by the FFT. The fourth line makes a stem plot of the magnitude (`abs()`) FFT of one cycle of the sine wave, i.e. 10 samples.

The number of samples in one cycle of a repeating waveform, such as a sine wave, is called the period:

$$p = sr/f$$

Notice that in our examples the period consists of a whole number of samples.

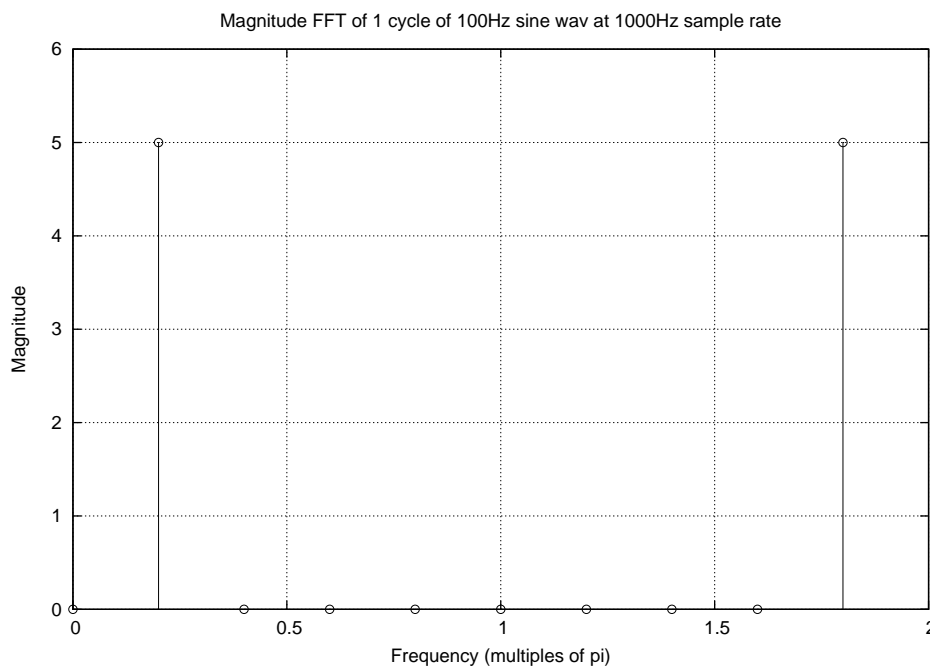
For a sample rate  $sr$  and signal of length  $n$ , the FFT takes frequency samples of the signal at angular frequency intervals  $sr/n$ ; in this case the frequency samples are spaced by  $1000/10 = 100$ Hz. The x-axis for the stem plot is determined by sampling

the angular frequencies  $0 \cdot 2 \cdot \pi / (1000/10)$  to  $10 \cdot 2 \cdot \pi / (1000/10)$  which simplifies to 0 to  $2 \cdot \pi$ . This way of labelling the frequency axis is called **normalised frequency**.

In normalised frequency the value  $2 \cdot \pi$  is the sample rate, the value  $\pi$  is the Nyquist frequency and all the frequency samples are spaced at angular frequency intervals of  $2 \cdot \pi / n$  for a signal of length  $n$ .

The spectrum for frequencies greater than the Nyquist frequency is the mirror image, about  $\pi$ , of the spectrum for frequencies less than the Nyquist frequency. This symmetry in the Fourier transform is present for all real-valued signals. It is usual to ignore the values greater than the Nyquist when interpreting a spectrum.

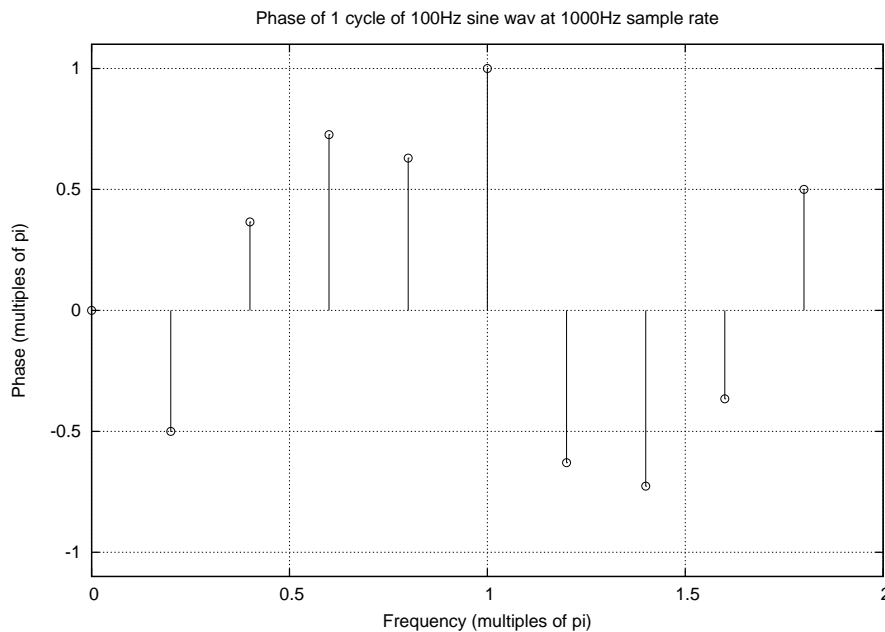
The most important feature of the spectrum of a sine wave is to recognise that there is only one frequency, below the Nyquist, that has non-zero magnitude. This means that a sine wave has exactly one frequency component. The sine wave is the basis for the spectrum; the spectrum tells us the amplitudes of the sine waves that we sample, at  $2 \cdot \pi / n$  intervals. **All** length- $n$  signals can be constructed by summing  $n$  sine waves with amplitudes corresponding to the magnitudes of the spectrum. This is the most remarkable fact of Fourier analysis: (almost) all signals can be represented and reconstructed as sums of sinusoids; this was discovered by the French mathematician Jean Baptiste Joseph Fourier in the late 1700s.



**Figure 4.8:** The magnitude spectrum of one cycle of a 100Hz sine with sample rate 1000Hz. The x-axis consists of 10 frequency components spanning angular frequencies 0 to  $2\pi$ . The spectrum is symmetric around its mid-point; only frequencies from 0 to the Nyquist frequency  $\pi$  are informative, the rest are due to mathematical symmetry in the Fourier transform of real signals.

To completely represent a signal in frequency each sinusoidal frequency component also needs a phase offset.

Figure 4.9 shows the phase spectrum of the same sine wave. The phase spectrum is computed by taking the `angle()` of the FFT of a signal. The phase values represent the phase offset of each frequency component and these values are in the range  $-\pi$  to  $\pi$ , spanning a  $2 * \pi$  range. In the figure the phase angle is shown in normalized units of  $\pi$ ; the y-axis is in the range  $-1$  to  $1$  which means  $-1 * \pi$  to  $1 * \pi$ .



**Figure 4.9:** The phase spectrum of the sine wave corresponding to Figure 4.8. The phase spectrum is anti-symmetric about its mid-point. The phases of frequencies  $\pi$  to  $2\pi$  are inverted values of the phases 0 to  $\pi$ .

From the phase spectrum we can see that the phase of the only non-zero magnitude sinusoidal component is  $-\pi/2$ . This is the Fourier phase of a sine wave with  $-\pi/2$  phase; it corresponds to a cosine wave.

### Spectrum of a cosine wave

Figure 4.10 shows the magnitude FFT of a cosine wave consisting of 10 samples corresponding to one full cycle of the cosine wave with frequency 100Hz at a sample rate of 1000Hz. Can you see a difference between Figure 4.10 and Figure 4.8?

### Spectrum of 10 cycles of a sine wave

We already know that a sine wave, or cosine wave, is periodic which means that it repeats and that the period of the repetition in samples is the sample rate divided by the frequency:  $sr/n$ . How does including more samples of the signal affect the Fourier analysis of the signal?

By setting  $n = 10 * sr/f = 100$  we can compute the Fourier transform of 10 cycles of the sine wave. Figure 4.12 shows the Fourier transform. How is the Fourier transform for 10 cycles different from the Fourier transform for one cycle?

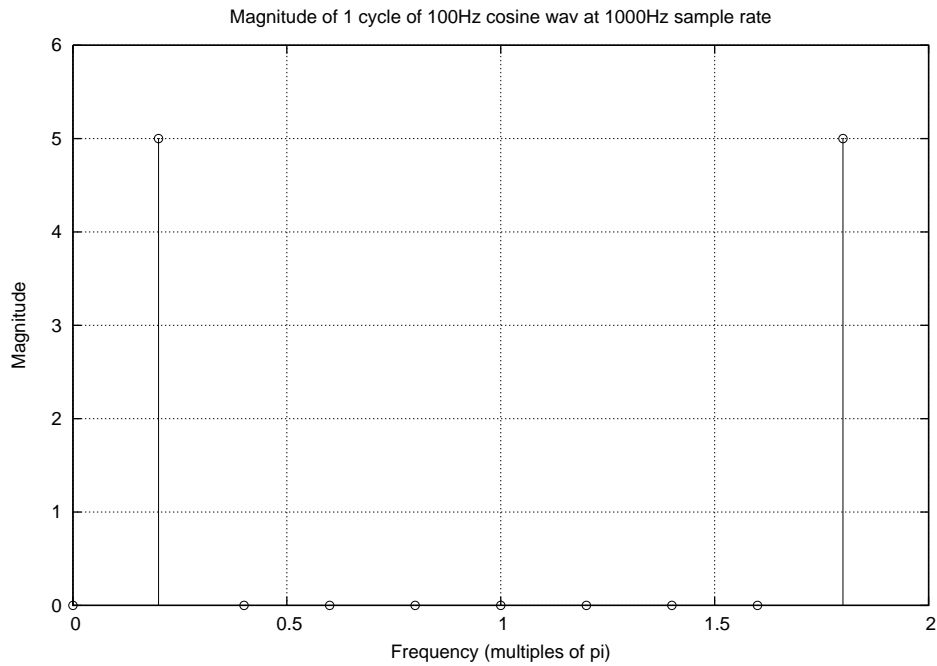


Figure 4.10: The magnitude FFT of a cosine wave with frequency 100Hz at a sampling rate of 1000Hz. The FFT consists of  $1000/100 = 10$  samples for one cycle of the cosine wave.

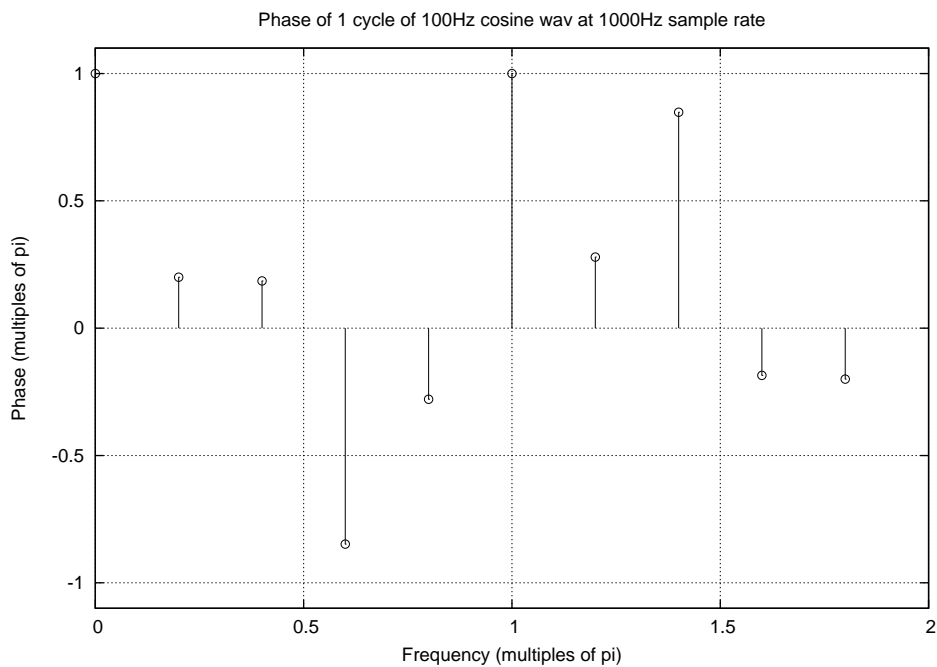
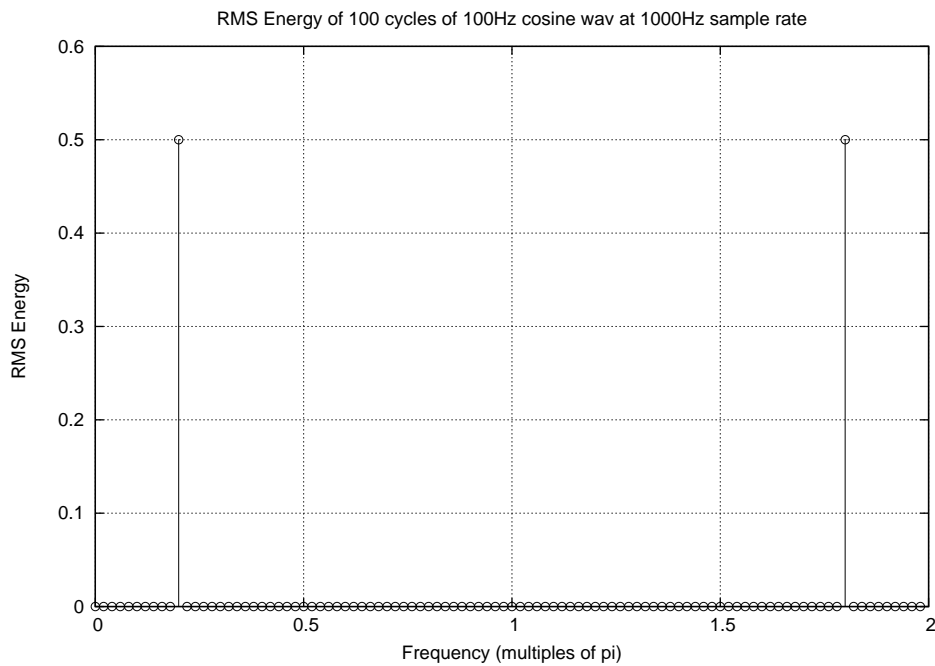


Figure 4.11: The phase of the same cosine wave as in the previous example. The phase of the frequency component below the Nyquist frequency having non-zero magnitude is  $\pi/4$ ; this means that Cosine waves are the basis of real signals in Fourier analysis.



**Figure 4.12:** Fourier transform of 10 cycles of the sine wave.

By increasing the length of the signal to 100 samples we have not altered the frequency content of the signal but we have altered the total energy in the signal because energy depends on the signal's length:

$$\text{Energy} = \sqrt{\sum_{n=1}^N x[n]^2}$$

To eliminate the dependency on the length of the signal we often use the root-mean-square (RMS) energy, which divides the total energy by the length of the signal:

$$\text{RMS} = \sqrt{\frac{1}{N} \sum_{n=1}^N x[n]^2}$$

If we do the same for the Fourier transform, then the Magnitude spectrum divided by the signal length will yield the same values for periodic signals with a length of any integer multiple of the period.

---

**Learning activity**

- Compute the magnitude Fourier transform of three cycles of a sine wave with frequency 400Hz and sample rate of 1000Hz. Make a stem plot of the magnitude Fourier transform.
  - Compute the magnitude Fourier transform of eleven cycles of a sine wave with frequency 400Hz and sample rate of 1000Hz. Make a stem plot of the magnitude Fourier transform.
  - Divide the two magnitude Fourier transforms by the length of the signals (3 and 11 cycles respectively) and make stem plots of the results. What is the difference between the two plots?
  - Use the Octave commands `xlabel`, `ylabel` and `title` to label your plots.
- 

By increasing the length of the signal we increase the number of frequency samples in the Fourier transform. This means that the spacing between frequency components that we analyse gets smaller; the Fourier analysis components are closer together in frequency.

If we choose an integer multiple of the period of the sine wave as the signal length then the values of the magnitude components do not change. What changes is the number of samples of the spectrum that we compute. Hence, for a single sine wave the same non-zero magnitude value is reported for multiple cycles (after dividing by the signal length) as for one cycle; but the position of this non-zero value in the sequence of frequency samples changes because the frequency samples are more dense. What we see are an increased number of samples of the same underlying magnitude spectrum.

This is also true for phase. The value of the phase for the component with non-zero magnitude remains the same but here we see many more phase values that are non-zero, as in Figure 4.13. These phase values are for components that do not exist in the signal so they can essentially be ignored. We are only interested in the phases of those components with non-zero magnitude. Again, what we see is not a different phase spectrum, but more samples of the same underlying phase spectrum.

**Spectrum of an impulse**

The Fourier transform of the unit impulse also shows a fundamental property of the Spectrum:

```
octave:>fft([1 0 0 0 0 0 0])
ans =
1 + 0i 1 + 0i 1 + 0i 1 + 0i 1 + 0i 1 - 0i 1 - 0i 1 - 0i
octave:>abs(ans)
ans =
1 1 1 1 1 1 1
```

Figure 4.14 shows the unit impulse signal; notice that the signal is offset in time. Figure 4.15 shows the spectrum of the unit impulse. These examples illustrate that the magnitude Fourier transform of the unit impulse is a sequence of 1s. The phase of each component in the Fourier transform is related to the time shift of the unit impulse.

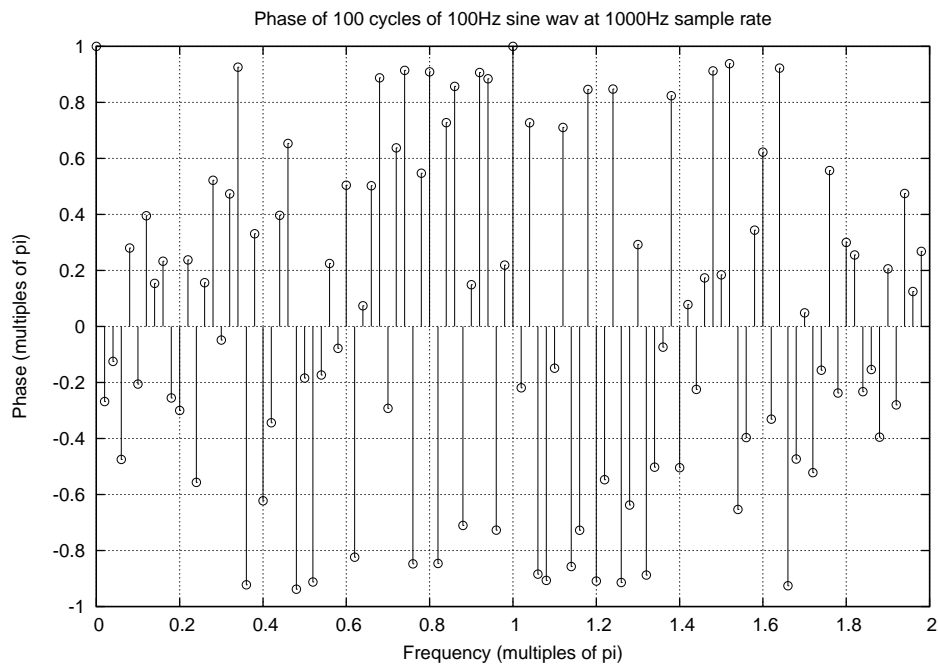


Figure 4.13

For the impulse in Figure 4.14 the phase is:

```
octave:>angle(fft([0 0 0 0 0 1 0 0 0 0 0]))
ans =
Columns 1 through 8:
0.00000 -2.85599 0.57120 -2.28479 1.14240 -1.71360 1.71360 -1.14240
Columns 9 through 11:
2.28479 -0.57120 2.85599
```

Note that the signal is of length 11; the phase in this example can be more conveniently represented in units of  $2 * \pi / 11$ :

```
ans/(2*pi/11)
ans =
0 -5 1 -4 2 -3 3 -2 4 -1 5
```

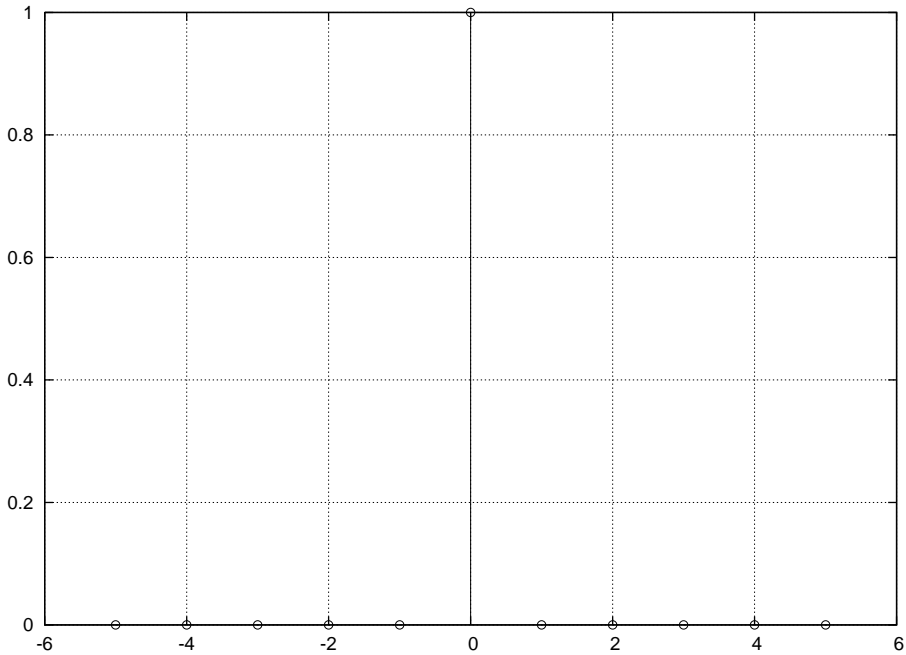


Figure 4.14: A unit impulse signal with zero-time reference at sample position 6.

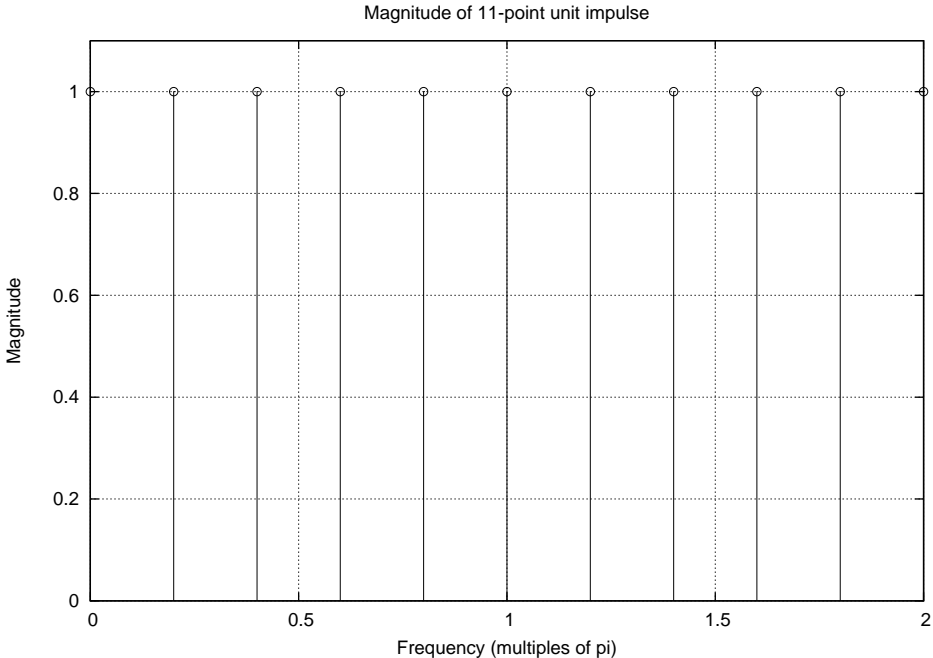


Figure 4.15: The RMS magnitude spectrum of the impulse demonstrating that the unit impulse consists of all frequencies with unit amplitude.

---

**Learning activity**

Make stem plots of the magnitude and phase spectra, using Fourier transforms, of one cycle of each of the following waveforms with a sample rate of 44.1kHz and a fundamental frequency of 441Hz (Hint: one cycle will be  $44100/441 = 100$  samples):

- ten harmonics with amplitude  $(1 - k/10)$
  - ten harmonics with amplitude  $k/10$
  - ten harmonics with amplitude  $\exp(0.5 * -k)$
  - ten harmonics with amplitude  $\exp(0.5 * k)$
  - the fundamental and even harmonics up to ten harmonics [ 1 2 4 6 8 10 ]
  - the fundamental and odd harmonics up to ten harmonics [ 1 3 5 7 9 ].
- 

### 4.3.5 Convolution by spectrum multiplication

A further very useful property of convolution is that when we take the spectrum of two signals, their element-wise product is the Fourier transform of their convolution. This means that taking the inverse Fourier transform, using `ifft()`, yields the convolution of the two signals.

Because the FFT is an efficient way to compute the Fourier transform, the process of computing the spectrum, performing multiplication and taking the inverse Fourier transform is more efficient and therefore takes less time than the process of convolution using `conv()`.

Here are some examples of convolution using FFTs. The first example is convolution of a signal by the unit delay.

```
sig = [1 2 3 4 5 6 5 4 3 2 1];
d = [0 1]; S = fft(sig, length(sig)+length(d)-1); D = fft(d,
length(sig)+length(d)-1); X = S .* D; x = ifft(X); sig
sig =
1 2 3 4 5 6 5 4 3 2 1
fix(real(x)) ans =
0 1 2 3 4 5 6 5 4 3 2 1
```

The second example illustrates the use of the FFT to perform low-pass filtering of a signal using the 2-point averaging filter [0.5 0.5]:

```
sig = rand(1,10)*2-1;
d = [0.5 0.5]; S = fft(sig, length(sig)+length(d)-1); D = fft(d,
length(sig)+length(d)-1); X = S .* D; x = ifft(X); sig = Columns 1
through 7:
0.660514 -0.287672 0.092979 0.954095 0.137672 0.845794 -0.120167
Columns 8 through 10:
0.920592 -0.715481 -0.512909
x x =
Columns 1 through 7:
0.330257 0.186421 -0.097346 0.523537 0.545883 0.491733 0.362814
```

```
Columns 8 through 11:  
0.400213 0.102555 -0.614195 -0.256454  
conv(sig,d) ans =  
Columns 1 through 7:  
0.330257 0.186421 -0.097346 0.523537 0.545883 0.491733 0.362814  
Columns 8 through 11:  
0.400213 0.102555 -0.614195 -0.256454
```

There are many uses for Fourier transforms: in this chapter we have used them for analysing spectra of signals to see their constituent frequency components and for an alternative implementation of the convolution operation.

Building upon the knowledge that you have gained about systems, convolution, Fourier representations and transformation by Fourier representations, there is a final observation that we can make.

LTI systems have the unique property that complex exponentials presented at the input are passed through the system and multiplied by the frequency components of the impulse response of the system. There can be no new frequency components added by a linear system; instead, existing frequency components can be attenuated, amplified or delayed. Therefore all discrete-time systems are combinations of the simple operation of scaling and delaying individual sinusoidal components that are summed together to form the system response to the input.

In other words, sinusoids presented at the input to a system are also present at the output; but they can be scaled and/or delayed. Because all signals can be represented as sums of sinusoids, all system responses can be decomposed into their individual scaled and delayed sinusoids that are summed back together to form the final signal. For this reason it is said that Complex Exponentials are Eigenfunctions of Linear Time Invariant Systems. An Eigenfunction is a signal, in this case a sinusoid, that passes through a system with its basic shape unchanged; and LTI systems do not change the shape of sinusoids, they simply scale them and shift them in time.

---

## 4.4 Summary and learning outcomes

This chapter first looked at linear time invariant (LTI) systems and their properties. The impulse response was introduced as a complete characterisation of LTI systems that is used to implement system transformations on signals using the convolution operator.

The convolution operator was shown to be a way to combine a signal with itself by delaying and scaling. This produces a transformation of the signal that is often called filtering. The operations of delay and low-pass filtering were introduced.

Representation of sinusoidal signals by complex exponentials was introduced leading to the representation known as the spectrum. The Fourier transform, using the FFT, was then discussed as a means for computing spectra as well as its implementation in the Octave programming language. It was then illustrated that the Fourier transforms of a signal and a system can be multiplied to produce the Fourier transform of the convolution of the two signals. By taking the inverse Fourier transform of this product the convolved, or filtered, signal is produced.

Not only is the Fourier transform a useful analysis tool for inspecting the sinusoidal components of signals and systems, but it can be a more efficient method for

performing convolution than computation by direct means (e.g. using `conv()`). For these reasons the Fourier transform, and the complex exponential representation of signals and systems, are very important in the study of signal processing.

With a knowledge of the contents of this chapter and its directed reading and activities, you should be able to:

- describe what an LTI system is
- discuss the impulse response and its importance in signal processing
- describe what convolution is and how it is used, as well as being able to perform convolution on signals using Octave
- describe the basic theory of Fourier analysis, and in particular the use of the Fast Fourier Transform in digital signal processing
- describe spectral analysis and how signals are made up of sinusoidal waveforms.

---

## 4.5 Exercises

In addition to the exercises listed below, you should attempt all of the learning activities throughout this chapter, to enhance your understanding of the material.

1. What is the difference between a signal and a system?
2. What is meant by the term ‘*impulse response*’? What is the impulse response of a room?
3. What is Fourier analysis? Discuss the use of Fourier analysis for both analogue and digital signals. In particular, describe how the analysis might differ for these two types of signals.
4. What is a Fourier Transform? Can a Fourier Transform be applied to both analogue and digital signals? What is a Fast Fourier Transform? Is this applicable to both analogue and digital signals?
5. What is convolution? What is the relationship between convolution and the FFT?

