Introduction to Fourier Transforms

5.1 Introduction

A time-varying signal such as a sound wave can either be described as amplitude as a function of time, or amplitude as a function of frequency. These are two equally valid ways of describing the signal. The same idea holds for spatially-varying signals too, where the independent variable is space rather than time. Fourier analysis is the technique of converting between these two different viewpoints, i.e. taking a signal and decomposing it into its frequency components. This operation is called a *Fourier transform*.

The basis functions of the frequency domain are sinusoids, which means that you can build up any function by combining sinusoids (each representing one frequency) of the appropriate amplitudes and phases. Jean Baptiste Joseph Fourier showed in 1807 that for continuous periodic functions, you only need to combine sinusoids with frequencies that are an integer multiple of the fundamental frequency. Non-periodic functions can also be built from sinusoids, but this requires that a continuous summation (i.e. an integral) be taken over them.

Frequency decomposition is an important concept, because there are many applications whose natural measurement space is the frequency domain of the quantity we are actually after. These include X-ray crystallography, Fourier transform spectroscopy, and radio and optical interferometry. For example in X-ray crystallography, the peaks in the X-ray diffraction pattern tell you which families of atomic planes (i.e. spatial frequencies) are present, and this can be used to deduce the arrangement of atoms in the crystal. The law that governs this is the famous *Bragg's law* of X-ray diffraction, named after William Henry Bragg and his son, William Lawrence Bragg, who in 1913 successfully applied this to analysing the atomic structure of sodium chloride.

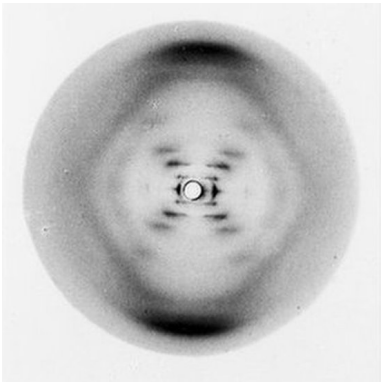


Figure 5.1: The historic "Photo 51", showing the X-ray diffraction pattern of DNA taken by Rosalind Frankin and Ray Gosling in 1952, from which the double-helix structure was deduced (source: `http://www.bbc.com/news/health-18041884`).

Since continuous functions cannot be exactly represented by digital computers (the signal must always be discretised and of finite length), computers perform a Discrete Fourier Transform (DFT) rather than a true continuous Fourier transform. DFTs can be evaluated efficiently on computers thanks to the Fast Fourier Transform (FFT) algorithm, developed by Gauss in 1805 and later rediscovered by Cooley and Tukey in 1965. In this lab we will explore how to do one-dimensional Fourier Transforms with MATLAB, and in the following labs extend this analysis to two dimensions.

5.2 Lab objectives

By the end of this lab session you should be able to:

1. Understand the concept of frequency decomposition.
2. Use **`fft`** to perform discrete Fourier transforms and be familiar with its output format.
3. Understand the importance of the Nyquist limit.

If you still don't know how to do any of these things once you have completed the lab, please ask your tutor for help. You should also use these lists of objectives when you are revising for the end of semester exam.

5.3 Fourier transforms

Suppose we have a function $y(x)$ defined along $-\infty < x < +\infty$, where x represents the independent variable (this could be space, time, etc.). The key concept of Fourier analysis is that this function can alternatively be described as a combination of frequencies $Y(f)$, where f denotes frequency. If x is a spatial coordinate then f is a spatial frequency (also called a wavenumber, or the number of oscillations per unit length). If x is a time coordinate then f is a temporal frequency (the number of oscillations per unit time). The complex amplitude of the sinusoid of frequency f that goes into building $y(x)$ is given by

$$Y(f) = \int_{-\infty}^{+\infty} y(x) \mathrm{e}^{-i2\pi f x} \, \mathrm{d}x \,, \tag{5.1}$$

where the units of f are the reciprocal of the units of x. For example, if x is measured in seconds, then f has units of hertz (s^{-1}). We say that $Y(f)$ is the *Fourier transform* of $y(x)$.

There are multiple conventions for defining $Y(f)$, sometimes with factors of $\frac{1}{2\pi}$ or $\frac{1}{\sqrt{2\pi}}$ out the front. The frequency variable f is also sometimes defined with the factor of 2π in the exponent absorbed into it. In such cases, the units of f will be slightly different, e.g. radians per second rather than hertz, if x is in seconds. However, we are going to stick with the definition of f without it absorbing the factor of 2π, because this makes it more straightforward to interpret the units as just plain reciprocals of the original units.

Computers cannot store and represent continuous functions over infinite domains, so they must instead make do with a discrete and finite version of Eq. (5.1). In MATLAB, the *discrete Fourier transform* or DFT is evaluated by the inbuilt **`fft`** function as the sum

$$Y(k) = \sum_{j=1}^{N} y(j) \mathrm{e}^{-i2\pi(j-1)(k-1)/N} \,, \tag{5.2}$$

where N is the number of measurements, and j and k are the indices of the arrays y and Y, respectively. An important point to note is that the individual data points must be evenly spaced for Eq. (5.2) to hold, i.e. the **`fft`** function can only be used to analyse data taken at regular intervals.

5.4 MATLAB's `fft` function

For an input signal described by a vector of data points **`y`**, its discrete Fourier transform **`Y`** can be computed using the **`fft`** function:

```
>> Y = fft(y);
```

The entries of **`Y`** will in general be complex numbers, encoding the amplitudes and phases of each frequency component present in the input vector **`y`**. Each entry of **`Y`** corresponds to one frequency.

It is important to understand the output format of the function **`fft`**, i.e. what frequencies each element of the array **`Y`** corresponds to. This differs depending on whether the input vector **`y`** contains an odd or an even number of elements.

Suppose that the values in **`y`** are sampled at a cadence/spacing of dx for a range 0 to L:

```
x=dx:dx:L; %Note that the first data point in the range is not included in the sample: x starts from dx (no zero)
```

and that you calculate the Fourier transform of **`y`** (**`Y=fft(y)`**). The first value of **`Y`** corresponds to the zero-frequency component, i.e. the constant offset. If **`y`** has an even number of elements, then

$$f_{\mathrm{even}} = \left[0, \, \frac{1}{L}, \, \frac{2}{L}, \, \cdots, \, \mathrm{Nq} - \frac{1}{L}, \, -\mathrm{Nq}, \, -\mathrm{Nq} + \frac{1}{L}, \, -\mathrm{Nq} + \frac{2}{L}, \, \cdots, \, -\frac{2}{L}, \, -\frac{1}{L}\right] , \tag{5.3}$$

if **`y`** has an odd number of elements, then

$$f_{\mathrm{odd}} = \left[0, \, \frac{1}{L}, \, \frac{2}{L}, \, \cdots, \, \mathrm{Nq} - \frac{1}{2L}, \, -\mathrm{Nq} + \frac{1}{2L}, \, -\mathrm{Nq} + \frac{3}{2L}, \, \cdots, \, -\frac{2}{L}, \, -\frac{1}{L}\right] . \tag{5.4}$$

The term Nq in both equation is known as the *Nyquist frequency* and it is defined as

$$\mathrm{Nq} = \frac{1}{2}\frac{1}{dx} = \frac{1}{2}f_s \tag{5.5}$$

where $f_s = \frac{1}{dx}$ is the sampling frequency.

Notice the abrupt jump from positive to negative frequencies near the middle of each array. This ordering stems from the way MATLAB implements its FFT algorithm. Equations (5.3) and (5.4) are summarised schematically in Fig. 5.2.

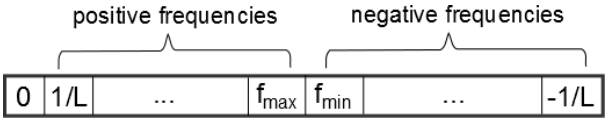


Figure 5.2: Output format of MATLAB's **`fft`** function, where the array elements are labelled with their associated frequencies. When the number of samples is even, $f_{\max} = \mathrm{Nq} - \frac{1}{L}$ and $f_{\min} = -\mathrm{Nq}$. When the number of samples is odd, $f_{\max} = \mathrm{Nq} - \frac{1}{2L}$ and $f_{\min} = -\mathrm{Nq} + \frac{1}{2L}$.

Negative frequencies appear naturally when we use complex exponentials to represent sinusoids. For example, a sine wave can be expressed as the linear combination of two complex exponentials with positive and negative frequencies $+f$ and $-f$:

$$\sin 2\pi f x = \frac{1}{2i}(\mathrm{e}^{i2\pi f x} - \mathrm{e}^{-i2\pi f x})\,. \tag{5.6}$$

Both positive and negative frequencies are necessary to fully describe functions if we use complex arithmetic. If the signal is purely real, then it turns out that the negative-frequency components are redundant (their amplitudes are complex conjugates of the corresponding positive-frequency components) so in this case they can just be ignored. In these labs we will not deal with any complex input signals.

None of the frequencies in Equations (5.3) and (5.4) exceed Nq in absolute value. As we will discuss in more detail later, Nq is the highest frequency that can be measured from an input signal given a fixed sampling rate. It is an important concept in the context of discrete signal processing, because it specifies the limit on the highest frequency than can be reconstructed accurately from any data.

Question 1
If a signal is sampled at a cadence of $dx = 2\,\mathrm{s}$ for a duration of $L = 10\,\mathrm{s}$, use Equation 5.4 to calculate (*by hand*) the frequencies corresponding to the elements of the array output by **`fft`**. There are an odd number of samples and therefore frequencies, so we will call this vector **`fodd`**. What are the correct units?

$f_{odd} = [0, 0.1, 0.2, -0.2, -0.1]$ **(units of Hz, or s^{-1})**
$\mathrm{Nq} = 1/2dx = 1/(2 \times 2) = 1/4 = 0.25$
Tutor note: this question is a by-hand calculation - you don't need fft()
Samples are taken at 2, 4, 6, 8 and 10 s. Note we do not include sample at 0 s

Question 2
If a signal is sampled at spatial intervals of $dx = 0.5\,\mathrm{m}$ over a length of $L = 4\,\mathrm{m}$, use Equation 5.3 to calculate (*by hand*) the frequencies corresponding to the elements of the array output by **`fft`**. There are an even number of samples and therefore frequencies, so we will call this vector **`feven`**. What are the correct units?

[0, 0.25, 0.5, 0.75, −1, −0.75, −0.5, −0.25] (units of m^{-1})
Tutor note: this question is a by-hand calculation - you don't need fft()
Again, note we do not include sample at 0s

5.5 MATLAB's `fftshift` function

If you wanted to plot the output of **`fft`** versus frequency, you would need to accommodate for the awkward ordering of frequencies. MATLAB has an inbuilt function called **`fftshift`** that is designed for this purpose. **`fftshift`** takes a vector output by **`fft`** and re-orders it such that the negative frequencies are placed to the left of the positive frequencies. After this operation, all frequencies will be in ascending order. The result is shown schematically in Fig. 5.3 (compare this to Fig. 5.2).

Figure 5.3: The resultant frequency ordering of a vector generated by **`fft`**, after it has been passed to **`fftshift`**. f_{min} and f_{max} refer to the same quantities mentioned in Fig. 5.2.

Question 3
Apply **`fftshift`** to the vectors of frequencies **`fodd`** and **`feven`** you found in Questions 1 and 2. Write down the resultant vectors.

```
fftshift([0, 0.1, 0.2, -0.2, -0.1])
ans = -0.2000   -0.1000    0    0.1000    0.2000

fftshift([0, 0.25, 0.5, 0.75, -1, -0.75, -0.5, -0.25])
ans = -1.0000   -0.7500   -0.5000   -0.2500  0   0.2500   0.5000   0.7500
```

5.6 How to compute and plot a Fourier transform

We will now put this together to calculate the Fourier transform of an input signal. Let's start with a sine wave (Equation 5.6) of frequency $f = 3$ sampled with $dx = 0.01$ over a range of $L = 2$. We can create the input signal and plot it like this:

```
dx = 0.01;
L = 2;
x = dx:dx:L;
f = 3;
y = sin(2*pi*f*x);
plot(x, y, 'o');
xlabel('x');
ylabel('y(x)');
```

resulting in the plot in the left panel of Figure 5.4. We can then calculate the Fourier transform, reorder the frequency components using **`fftshift`** and plot the results. In MATLAB this is:

```
Y = fft(y);
F = fftshift(Y);
Nq = 1/(2*dx);
freqs = -Nq : 1/L : Nq-1/L;
```

```
sqmod = abs(F).^2;
plot(freqs, sqmod);
xlabel('f');
ylabel('|F|^2');
```

which results in the plot in the right panel of Figure 5.4. The two spikes you see represent the two complex exponential terms appearing on the RHS of Equation (5.6), which have frequencies +3 and −3 for this example (the input sine wave, $\sin 2\pi f x$, had $f = 3$).

Since the output amplitudes are complex, it is usually the modulus or the squared modulus that is plotted as a function of frequency. Both the modulus and squared modulus of a complex number are real numbers. The more physically meaningful quantity in an optics context is the squared modulus, because the intensity of a diffraction pattern is directly proportional to the squared modulus of the Fourier transform of the aperture.

Because the input signal is real, the output complex amplitudes of the components with frequencies $+f$ and $-f$ for a given f found in the array **`Y`** are complex conjugates of one another. Their squared moduli are therefore equal. Hence if the input signal is real, the plot of the squared modulus as a function of frequency will be symmetric about $f = 0$. For this reason, you will sometimes see just the positive frequencies being plotted.

An important thing to do is to put the correct numbers on the frequency axis. The rules in Section 5.4 tell us that if **`y`** has an even number of elements, the minimum and maximum frequencies are $f_{\min} = -\mathrm{Nq}$ and $f_{\max} = \mathrm{Nq} - \frac{1}{L}$, otherwise if **`y`** has an odd number of elements, the minimum and maximum frequencies are $f_{\min} = -\mathrm{Nq} + \frac{1}{2L}$ and $f_{\max} = \mathrm{Nq} - \frac{1}{2L}$. In both cases, the frequency step size is $\frac{1}{L}$. Recall that $\mathrm{Nq} = \frac{1}{2}\frac{1}{dx}$ Applying these rules to this example, we constructed the vector of frequency values **`freqs = -Nq : 1/L : Nq-1/L`**.

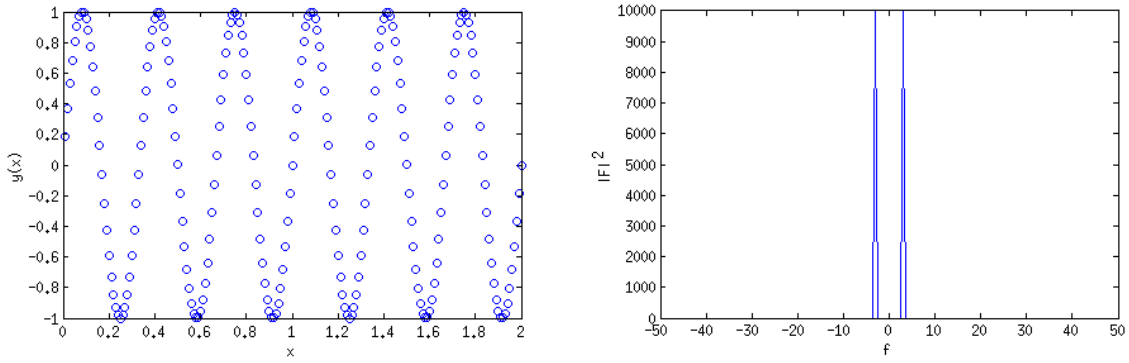


Figure 5.4: (a) A sine wave and (b) its Fourier transform.

Question 4
In Section 5.6 we constructed a vector of frequencies using the command **`freqs = -Nq : 1/L : Nq-1/L`**, where **`Nq`** is the Nyquist frequency and **`L`** is the range of the data. This expression is the **`fftshift`**ed version of Equation (5.3), and is relevant for cases where there are an even number of samples. Write down the command you would use to construct the frequency vector **`freqs`** if there were an odd number of samples.

```
freqs = -Nq+1/(2*L) : 1/L : Nq-1/(2*L)
```

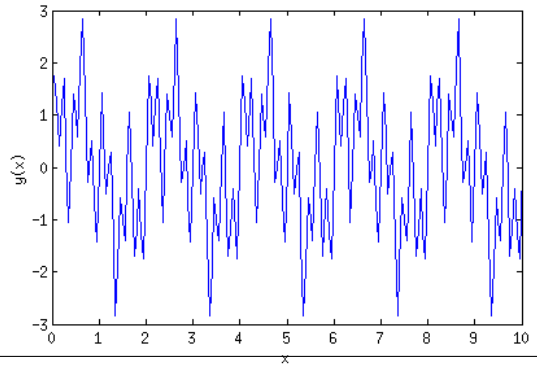
Question 5
Consider a waveform comprising a superposition of three sinusoids with frequencies f_1, f_2 and f_3, which has the functional form

$$y(x) = \sin(2\pi f_1 x) + \sin(2\pi f_2 x) + \sin(2\pi f_3 x)\,. \tag{5.7}$$

Set $f_1 = 0.5$, $f_2 = 2$ and $f_3 = 5$, and sample this at the points **`x = dx:dx:L`**, where **`dx = 0.05`** and **`L = 10`**.

Sketch the resulting signal below.

```
dx = 0.05;
L = 10;
x = dx:dx:L;
y = sin(2*pi*0.5*x) + sin(2*pi*2*x) + sin(2*pi*5*x);
plot(x, y)
xlabel('x')
ylabel('y(x)')
```



Question 6

Now compute and plot the squared modulus of its Fourier transform. Use Equation (5.3) to work out the frequencies that should go on the horizontal axis. Verify that the peaks appear at the correct frequencies.

```
Y = fft(y);
F = fftshift(Y);
Nq = 1/(2*dx);
freqs = -Nq : 1/L : Nq-1/L;
sqmod = abs(F).^2;
plot(freqs, sqmod);
xlabel('f');
ylabel('|F|^2');
```

The peaks should occur at f = 0.5, 2, 5.

Checkpoint 1:

5.7 Aliasing

We have already said that for purposes of computation a function must be represented by a series of data points taken over a finite interval. Clearly the representation is more accurate if the data points are close together.

How close must the data points be if we are to represent a sine wave of period T? Run the code

```
T = 1;
dt = 0.01;
t = dt:dt:20*T;
y = sin(2*pi*t/T);
plot(t,y)
```

The cadence **dt** is small and the representation is good. Now increase the cadence. Try dt = 0.1, 0.2, 0.5, 1, 1.1 and 1.2. You can see that if $dt \geq 0.5$ T, the waveform appears to have the wrong period. This is *aliasing*. In other words, the sampling frequency f_s must be at least two times greater than the frequenct of the wave we are interested in:

$$f_s \geq 2\, f,$$

where f is the frequency of the signal. For example if the signal is 5 HZ, then you must sample at a frequency greater than 10 Hz. The highest frequency that we can obtain without aliasing is the *Nyquist frequency* (Nq).

5.7.1 A spinning wheel

The **Nyquist limit** is the highest frequency that can be reconstructed from a discretely-sampled signal. Imagine you are filming a spinning object, which could be a wheel or a propeller, at some given frame rate, e.g. 30 frames per second. If the object were to rotate at exactly 30 revolutions a second, then the video would appear to show the object completely still and non-rotating. Someone watching the video would incorrectly conclude that the frequency of rotation was zero. For a video demonstration of this effect, see

`http://www.youtube.com/watch?v=VNftf5qLpiA`

What's happening is that above a certain threshold, the sampling rate is not high enough to "catch" the rotation of the object. The object may appear on film to rotate more slowly than it actually is, or even rotate in the opposite direction. This effect where measured frequencies differ from actual frequencies because of too low a sampling rate is known as *aliasing*, and the threshold signal frequency above which this occurs (for a given sampling rate) is called the Nyquist frequency. Figure 5.5 shows a schematic of how aliasing for a rotating wheel occurs.

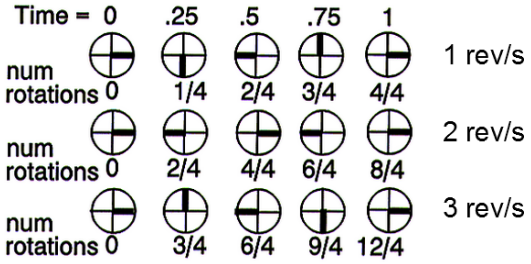


Figure 5.5: Snapshots of a clockwise-rotating wheel at a fixed sampling rate and different rotation speeds. In the third row, it is not possible to distinguish this from an anticlockwise rotation of 1 rev/s with 1/4 of a rotation between snapshots. (Source: `http://derek.dkit.ie/graphics/aliasing/aliasing.html`).

We are now going to explore what happens when we sample a given signal at different rates.

Question 7
Consider a signal oscillating at a frequency of 20 Hz. At what cadence dt must we sample in order to accurately reconstruct the waveform? Give your answer in seconds.

To reconstruct the signal accurately, we need to sample at at least 40 Hz, which corresponds to a cadence of $dt = 0.025$ **s.**

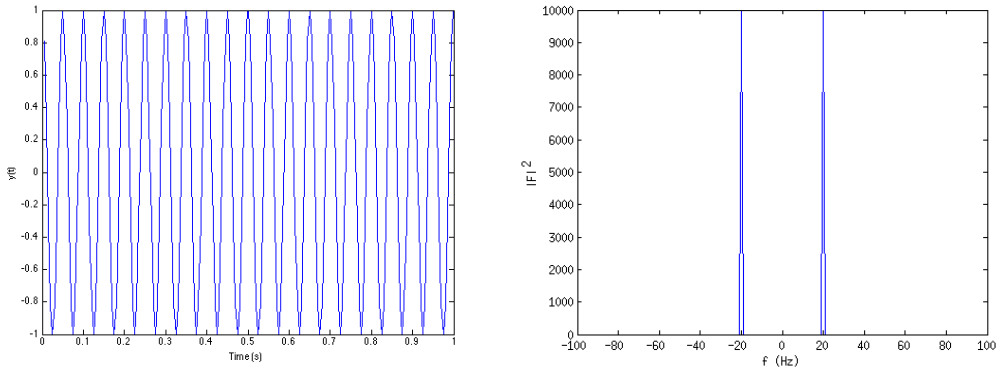
Question 8

Consider a sinusoidal wave of f = 20 Hz over a L = 1 s interval. Sample this waveform every 0.005 s.

a) How does the sampling frequency compare to the wave frequency?

The sampling frequency is 10 times the wave frequency

b) Sketch the original waveform and its Fourier transform below.



c) Are the frequency components what you expect? Is there aliasing?

The peaks appear at ±20 Hz.

Question 9

Now sample the waveform of Question 8 every 0.01 s, 0.02 s , 0.025 , 0.033 and 0.04 s.

a) What is the sampling frequency in each case and how does it compare to the wave frequency?

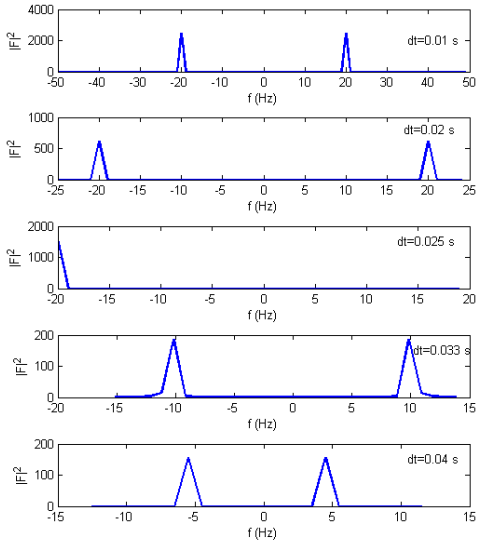
Wave Frequency: 20 Hz. Sampling Frequency:
0.01 s 100 Hz 5 times
0.02 s 50 Hz 2.5 times
0.025 s 40 Hz 2.0 times
0.033 s 30.3 Hz 1.52 times
0.04 s 25.0 Hz 1.25 times

b) What is the Nyquist frequency in each case?

Tutor note: Even though there is always an even number of samples, let the students think on how to deal with the case of even or odd number of points (see solutions).
0.01 s 50.0 Hz
0.02 s 25.0 Hz
0.025 s 20.0 Hz
0.033 s 15.15 Hz
0.04 s 12.5 Hz

c) For each case sketch the Fourier transform of the sampled waveform.

TUTOR: You may want to show them how to use subplots, but producing multiple plots is fine.



d) Are the frequency components what you expect? Is there aliasing?

The peaks appear at 20 Hz for $dt = 0.01s$ **and** $0.02s$ **but at** 10 **Hz for** $dt = 0.033s$ 5 **Hz for** $dt = 0.04s$**. So the frequency is measured accurately for the first two values but there is aliasing when** $dt = 0.033dt$ **and** $dt = 0.04dt$**.**

5.8 Fourier transform of a noisy signal (PHYS2911)

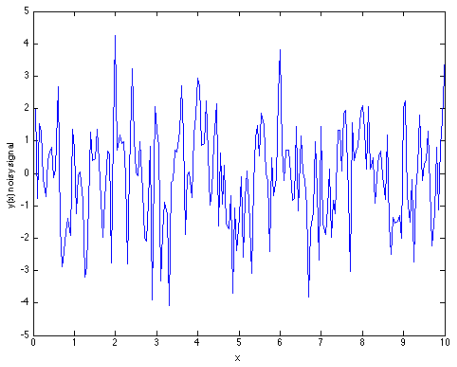
These questions are optional for PHYS2921 and PHYS2011

Signals measured in real life are often corrupted by noise. A common type of noise is white Gaussian noise. "Gaussian" means that the perturbation from the true signal at each measurement point is a random number drawn from a Gaussian distribution, and "white" means that no two perturbation values in the series are correlated.

We will now investigate what happens to the Fourier transform of a signal corrupted by white Gaussian noise, using the waveform you constructed in Question 5. A vector of uncorrelated, Gaussian-distributed random numbers can be generated by `g = randn(size(x));` where `x` is the vector of data points. By default, `randn` generates Gaussian noise with a characteristic amplitude of 1.

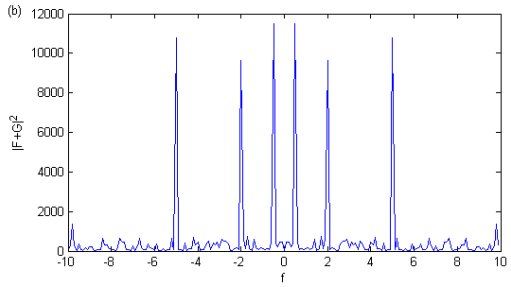
Question 10 (PHYS2911)
Generate a vector of Gaussian noise using the above command, add this on to the signal you created in Question 5, and plot the result. Sketch your plot below. If you didn't already know that the signal contained three periodic components, would you have been able to guess just by looking at it?

The waveform is now extremely corrupted and it is not possible to visually identify the presence of periodic components. By eye, it looks indistinguishable from random chaotic noise.



Question 11 (PHYS2911)
Now compute the Fourier transform of this signal and plot the squared modulus of the result. How does it differ from your answer to Question 6? Can you suggest why Fourier transforms are useful for analysing noisy signals?

Although the waveform is so corrupted, the spectral peaks are still prominent. This demonstrates that Fourier transforms can easily recover periodic signals that have been corrupted by white Gaussian noise, even when the noise levels are significant. They are very useful for identifying periodic signals buried within noise.



5.9 The Fourier transform phase (PHYS2921)

These questions are optional for PHYS2911 and PHYS2011

A complex number z can in general be expressed in terms of an amplitude r and phase θ, as $z = re^{i\theta}$. When we take the squared modulus of a complex number, this is the operation of multiplying it by its complex conjugate, i.e. $|z|^2 = z\bar{z} = re^{i\theta} \times re^{-i\theta} = r^2$. Since $|z|^2$ does not depend on θ, the operation of taking the squared modulus removes information about the phase θ. While phases are not important for some applications, they turn out to be a crucial pieces of information if we want to reconstruct the original waveform. We will now explore the consequences of modifying the phases of a Fourier transform.

The example we are going to use is a sawtooth function, which can be generated using the inbuilt MATLAB function **`sawtooth`**, e.g.

```
>> x = 0.2:0.2:6*pi;
>> s = sawtooth(x);
```

The resultant function looks like the one shown in Fig. 5.6.

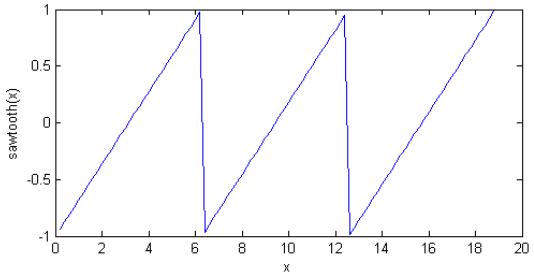
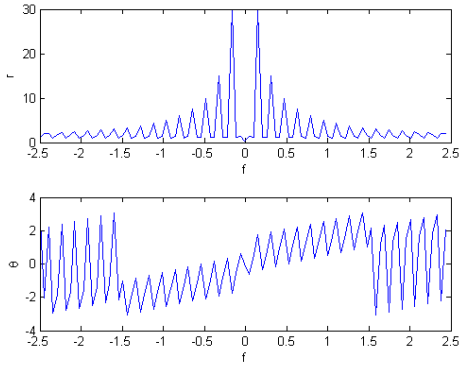


Figure 5.6: A sawtooth function.

Question 12 (PHYS2921)
Generate a sawtooth function using the above commands and compute its Fourier transform. Decompose this into amplitude and phase values, which you can achieve using the MATLAB functions **`abs`** and **`angle`**, respectively. Store these in two variables r and θ. Plot r and θ to see what they look like.



The MATLAB function for taking the inverse Fourier transform to recover the original function is **`ifft`**. This is an exact inverse of **`fft`**, in that any input **`x`** should be equal to **`ifft(fft(x))`**. If the original signal is real, then its Fourier transform should be Hermitian and taking the inverse Fourier transform should again recover the purely real input. However, internal roundoff error can sometimes lead to the appearance of small imaginary components in the inverse-Fourier-transformed signal. This can be suppressed by invoking the **`ifft`** function with the **`'symmetric'`** option, e.g.

```
>> X = ifft(x, 'symmetric');
```

An important point to note is that `ifft` expects input of the format output by `fft`, i.e. with the strange frequency ordering. If you used `fftshift` to reorder the array, you need to first put it back using `fftshift`'s inverse function, `ifftshift`, before you pass it to `ifft`. E.g. if you started off with some data `a`, the commands

```
>> A = fftshift(fft(a));
>> b = ifft(ifftshift(A), 'symmetric');
```

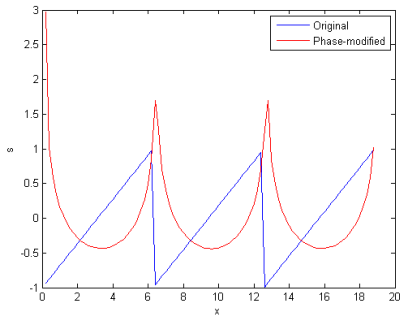
will recover a vector `b` that is identical to `a`.

Question 13 (PHYS2921)
Use the `ifft` function with the `'symmetric'` option to compute the inverse Fourier transform of

1. $re^{i\theta}$, i.e. the unmodified Fourier transform, and
2. $re^{i0} = r$, i.e. the Fourier transform of the sawtooth function with all phases set to zero.

Overplot these and comment on the similarities/differences. The waveform you obtain for part 1 should look identical to the one plotted in Fig. 5.6.

The two functions have the same overall periodicities and peak in the same places, but their shapes are very different.



Question 14 (PHYS2921)
What information is encoded in the phase of a sinusoidal wave? Why would changing the phases modify the shape of a periodic waveform but preserve its overall periodicity?

The phase of a sinusoid encodes its horizontal displacement. Changing the phases of the various sinusoidal components changes their positions relative to each other, and therefore where they constructively/destructively add to produce different features of the waveform. The dominant period is not changed because the relative amplitudes of the sinusoids don't change when the phase is modified, so the frequency component that had the largest amplitude in the original signal will still be the one with the largest amplitude after the phases have been changed.

Checkpoint 2: