10 Fourier Analysis

10.1 Introduction

In this chapter we present an introduction to the Fourier series and the Fourier transform. In particular, we will concentrate on computational algorithms for the Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT). It is expected that the student has at least heard of the concept of the Fourier series and transform, but this chapter is written at a basic level so that no previous knowledge is required.

The Fourier transfrom is an extremely useful digital tool for dealing with signals of all types; it allows the user to determine the presence of periodicities in the signal, it also allows the user to filter the signal to reduce the noise, or to autocorrelate noisy data to recover the original signal.

We begin our exploration of the Fourier transform with the Fourier Series.

10.2 The Fourier Series

Fourier series are used to represent period functions, defined as

$$h(t) = h(t+T) \tag{1}$$

where T is the period of the function $h(t)$. Any periodic signal in t can be expanded into a trigonometric series of sine and cosine functions, as long as it satisfies the following conditions:

1. $h(t)$ has a finite number of maxima and minima within T
2. $h(t)$ has a finite number of discontinuities within T, and
3. $h(t)$ has a finite power such that $\int_0^T |h(t)|dt < \infty$.

The trigonometric expansion of $h(t)$ can be written as an infinite series

$$h(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + \sum_{m=1}^{\infty} b_m \sin(m\omega t) \quad \text{where} \quad \omega = \frac{2\pi}{T} \tag{2}$$

To obtain values of the coefficients, a_0, the a_n's, and b_m's, we exploit the orthogonality properties of sinusoids:

$$\int_{-T/2}^{T/2} \cos(n\omega t)\cos(m\omega t) = 0 \qquad n \neq m \tag{3a}$$

$$\int_{-T/2}^{T/2} \sin(n\omega t)\sin(m\omega t) = 0 \qquad n \neq m \tag{3b}$$

$$\int_{-T/2}^{T/2} \sin(n\omega t)\cos(m\omega t) = 0 \qquad n \neq m,\ n = m \tag{3c}$$

$$\int_{-T/2}^{T/2} \cos^2(m\omega t)dt = \int_{-T/2}^{T/2} \sin^2(m\omega t)dt = \frac{T}{2} \tag{3d}$$

First, we multiply each termin in Eqn (2) by $\cos(n\omega t)$ and integrate over one period – using the conditions (3a – 3d), we arrive at an expression for the a_n's:

$$a_n = \frac{2}{T}\int_{-T/2}^{T/2} h(t)\cos(n\omega t)dt \tag{4}$$

Similarly, we multiply each term in Eqn (2) by $\sin(m\omega t)$ and integrate over one period. Using the conditions (3a – 3d), we arrive at an expression for the b_m's:

$$b_m = \frac{2}{T}\int_{-T/2}^{T/2} h(t)\sin(m\omega t)dt \tag{5}$$

Finally, to obtain a_0, every term in Eqn (2) is directly integrated over one period and all terms on the right hand side vanish except that containing a_0:

$$a_0 = \frac{1}{T}\int_{-T/2}^{T/2} h(t)dt \tag{6}$$

This simply states that a_0 is the average of the function $h(t)$. Equation (2) is known as the *Fourier Series* expansion of the time-dependent function $h(t)$. In practive, we need to restrict ourselves to a finite number of harmonics, which we discuss later.

Note that we said at the outset that the Fourier series represents periodic functions. However, with a little trickery, we can apply the Fourier series

expansion to any non-periodic function. We assume that the width of the domain of the non-periodic function may be considered a period (or an integral fraction of a period) of a periodic function that repeats itself outside of our domain of interest. A Fourier series may be written for the periodic function which will give us the desired values in the interval defined by our non-periodic function. Although it also gives us the values outside of the interval of interest, we don't care since our non-periodic function is not defined there (see Figure 1).

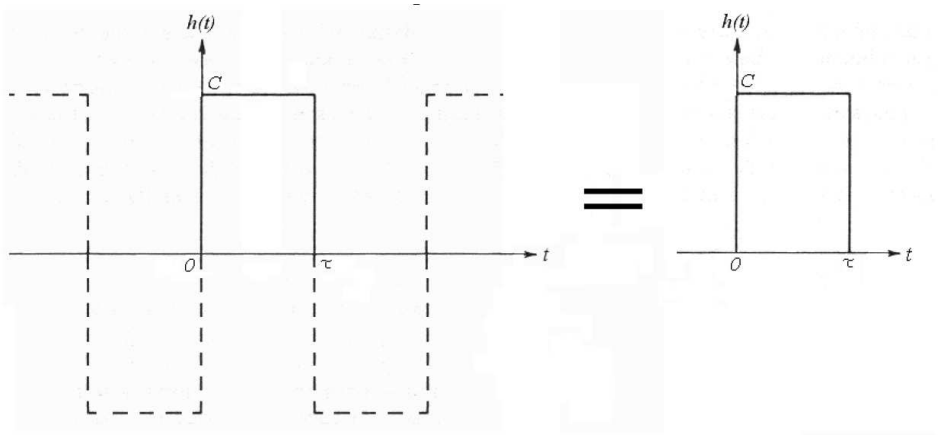


Figure 1: The periodic function with amplitude C defined on the left, when restricted to the domain interval $0 < t < \tau$, represents the non-periodic square function on the right with the same Fourier series expansion.

The domain interval is commonly selected as a half period (as in Figure 1) since the function extended outside the interval may be made either *even* or *odd*, and the corresponding Fourier series would then have only either cosine or sine terms alone, respectively. Figure 1 shows an odd function (anti-symmetric about $t = 0$), for which the Fourier series becomes:

$$h(t) = \sum_{m=1}^{\infty} b_m \sin(m\omega t) = \sum_{m=1}^{\infty} b_m \sin\left(m\pi \frac{t}{\tau}\right) \tag{7}$$

where we have used the fact that $\tau = T/2$ and, from inspection, $a_0 = 0$ (average is zero). In this interval, $h(t) = C$ and the coefficients are found via Eqn (5), noting that the contribution to the integral from the negative and

positive intervals are equal:

$$b_m = \frac{2}{\tau}\int_0^{\tau} C\sin\left(m\pi\frac{t}{\tau}\right)dt = \frac{2C}{m\pi}\left[-\cos\left(m\pi\frac{t}{\tau}\right)\right]_0^{\tau} \tag{8}$$

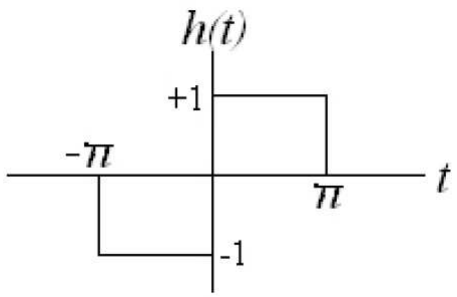
and the series is

$$h(t) = \frac{2C}{\pi}\left[2\sin\frac{\pi t}{\tau} + \frac{2}{3}\sin\frac{3\pi t}{\tau} + \frac{2}{5}\sin\frac{5\pi t}{\tau} + \ldots\right] \tag{9}$$

This series is based on the requirement that $h(t) = C$ over the interval $0 < t < \tau$ but also represents the dashed portion of the waveform in Figure 1 outside of that interval.

An important note here is that the same mathematical formulism that we have applied to the time domain applies equally well to the spatial domain; all that is needed is to replace, in all of the equations, ωt with kx, where k is the wavenumber defined as $k = 2\pi/\lambda$ and λ is the spatial period (wavelength).

Exercise 10.1: To knock some rust off of our math, let's work another example by hand. First of all, derive equations (4) – (6) in the text from equations (2) and (3), showing all steps. Next, write out the Fourier series for the following function defined as:



$$h(t) = -u(-t) + u(t)$$

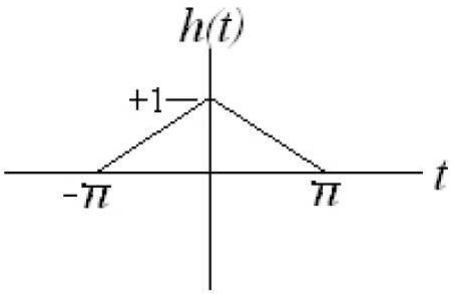
where $u(t)$ is the *unit step function*. The function is defined in the domain $-\pi \leq t \leq \pi$. Another way of writing this function is:

$$h(t) = -1 \quad t < 0$$

$$h(t) = +1 \quad t > 0$$

You should find that the a_n terms vanish and only the b_m terms in which m is odd survive. Comment on this result – why does it make sense for the function given? Write a program that calculates $h(t)$ from $-\pi$ to π using a series with 5 terms. Your program should plot the results to the screen using gnuplot.

Exercise 10.2: Obtain the program `bms.c` from the course website or the `/home/shared` directory on the comp1 server. This program determines the Fourier series coefficients from Exercise 10.1 numerically out to index M. Compile and run this code and see the effects of setting the number of coefficients, M, to 5, 10, and 50. Plot the resulting $h(t)$ in file `fs.out` for each M value using `gnuplot` for your comparison. Now, modify `bms.c` (this is what



you will turn in) to generate the Fourier coefficients and the $h(t)$ function out to 50 terms for the following function:

$$h(t) = 1 + \frac{t}{\pi} \quad t < 0$$

$$h(t) = 1 - \frac{t}{\pi} \quad t > 0$$

Note that this is now a *symmetric* function. what does this tell you about the Fourier series?

10.3 The Fourier Transform

In some problems, the function of interest is defined over the entire range and is aperiodic. An example is a square function of amplitude C in the range

$-\tau \leq x \leq \tau$ and zero everywhere else (see Figure 2A - this is different from the assumptions illustrated in Figure 1 in that we care about the function being zero outside of the range $-\tau \leq x \leq \tau$). This situation could be considered the limiting case of a periodic signal whose period goes to infinity. The spacing of the components $(n+1)\omega - n\omega = 2\pi/T$ becomes vanishingly small as the period T goes to infinity. In that limit, the spectrum of component sinusoidal waves becomes a continuum. In the limiting, aperiodic case, the series in Equation (2) is replaced by the integral:

$$h(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} H(\omega) e^{-i\omega t} d\omega \tag{10}$$

and the coefficients (equations 10.4 – 10.6) are replaced by

$$H(\omega) = \int_{-\infty}^{+\infty} h(t) e^{i\omega t} dt \tag{11}$$

Equations 10 and 11 go well beyond the apparent generalization of the Fourier series to continuous, aperiodic functions: together, they act as the means by which we transform functions to and from the time-domain and the frequency-domain. Why should we care? Sometimes we may find it difficult or tedious to analyze and manipulate data and/or functions in the time or spatial domain. In many applications such as electronics, signal processing, and communications, computations in frequency space not only make our lives easier, but may shed light onto system behaviors that may otherwise go undiscovered. Equation 11 is known as the Fourier transform of $h(t)$ – the Fourier transform of a function in the time domain, $h(t)$, yields its representation in the frequency domain, $H(\omega)$. Likewise, equation 10 is the inverse Fourier transform of $H(\omega)$. Together, equations 10 and 11 are known as the transform pair. Figure 2 shows an example transform pair. The Fourier transform and inverse Fourier transform of a function are often written as:

$$H(\omega) = \mathbb{F}[h(t)] \tag{12}$$

and

$$h(t) = \mathbb{F}^{-1}[H(\omega)], \tag{13}$$

respectively. For the Fourier transform to be applicable, the function has to be continuous or have a finite number of finite discontinuities in any finite interval, and the function must be absolutely integrable, that is:

$$\int_{-\infty}^{+\infty} |h(t)|dt < \infty \tag{14}$$

These statements don't really place strong limitations on the utility of the transform pair. Note that the statement "...or have a finite number of discontinuities" allows for a much broader range of powerful applications (e.g. impulse functions, a.k.a. the *Dirac delta function*) and leads us to discretization for numerical implementation (more on this later).

Certain symmetry properties within the time domain lead to accompanying symmetry properties within the frequency domain (which we are more than happy to exploit in our applications). Some transform pairs of interest are:
Time scaling:

$$\frac{1}{a}H\left(\frac{\omega}{a}\right) = \mathbb{F}[h(at)] \tag{15}$$

Frequency scaling:

$$\frac{1}{b}h\left(\frac{t}{b}\right) = \mathbb{F}^{-1}[H(b\omega)] \tag{16}$$

Time shifting:

$$H(\omega)e^{-i\omega t_0} = \mathbb{F}[h(t-t_0)] \tag{17}$$

Frequency shifting

$$h(t)e^{i\omega t_0} = \mathbb{F}^{-1}[H(\omega-\omega_0)] \tag{18}$$

Another useful property of the Fourier transform is that of linearity which states that the transform of the sum of two functions is the sum of the transforms: $\mathbb{F}[h(t)+g(t)] = \mathbb{F}[h(t)] + \mathbb{F}[g(t)]$.

For an example of the Fourier transform, consider the function $h(t) = C$ for the interval $-\tau \leq x \leq \tau$ and zero elsewhere (Figure 2A). The Fourier transform of $h(t)$ is:

$$H(\omega) = \mathbb{F}[h(t)] = \int_{-\infty}^{+\infty} Ce^{i\omega t}dt \tag{19}$$

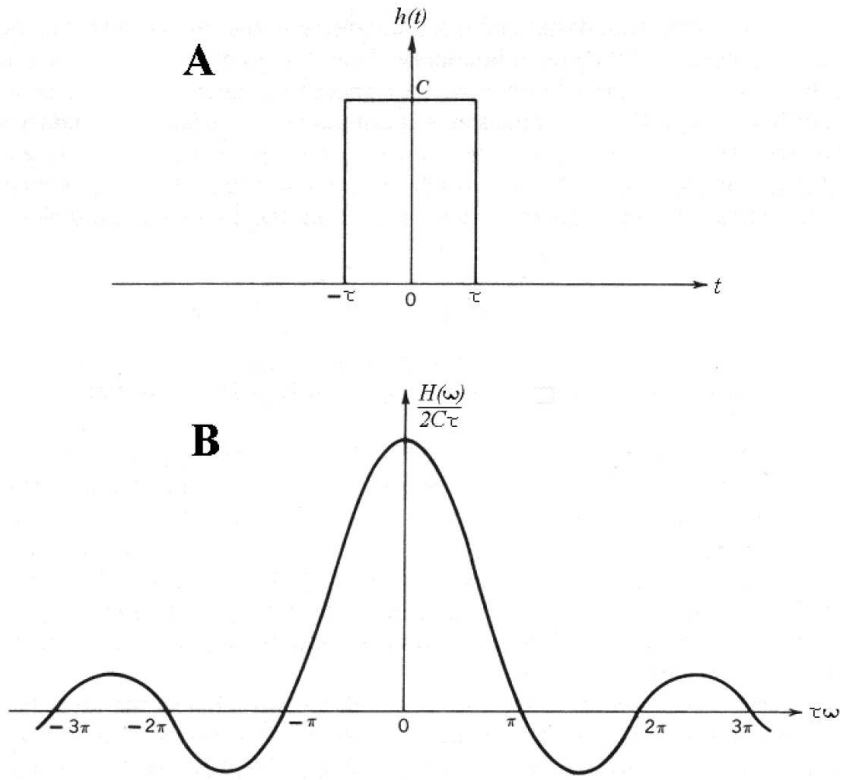


Figure 2: (A) A square aperiodic function in the time domain that has a value of C inside and a value of zero outside of the range $-\tau \leq x \leq \tau$. (B) The Fourier transform of the function in (A). This is a very common transform pair that comes up frequently in practice.

Since we know that the function is zero outside of the range $-\tau \leq x \leq \tau$, and using the principle of linearity, we can write:

$$H(\omega) = \mathbb{F}\left[h(t)|_{-\infty}^{-\tau} + h(t)|_{-\tau}^{+\tau} + h(t)|_{+\tau}^{+\infty}\right] = \mathbb{F}\left[h(t)|_{-\infty}^{-\tau}\right] + \mathbb{F}\left[h(t)|_{-\tau}^{+\tau}\right] + \mathbb{F}\left[h(t)|_{+\tau}^{+\infty}\right] \tag{20}$$

giving

$$H(\omega) = \int_{-\infty}^{-\tau} 0e^{i\omega t}dt + \int_{-\tau}^{+\tau} Ce^{i\omega t}dt + \int_{+\tau}^{+\infty} oe^{i\omega t} = \int_{-\tau}^{+\tau} Ce^{i\omega t}dt = C\frac{e^{i\omega t}}{i\omega}\Bigg|_{-\tau}^{+\tau} \tag{21}$$

Using Euler's expansion:

$$\sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i} \tag{22}$$

we arrive at:

$$H(\omega) = 2C\tau\left(\frac{\sin\omega\tau}{\omega\tau}\right) = 2C\tau\,\mathrm{sinc}(\omega\tau) \tag{23}$$

This is the $\mathrm{sinc}(\theta)$ function illustrated at the bottom of Figure 2. You will most likely see this again.

10.4 Sampling and the Discrete Fourier Transform (DFT)

To analyze and manipulate an analog signal (continuous function) using a digital computer, we need to take discrete *samples* of the function at fixed intervals in time and store these sampled values as a time series. There are many sampling techniques out there but all have in common a similar approach. For example, the signal in Figure 3A is sampled using an 8-bit sampler – the value at time t_n is represented by 8 bits, the leading bit for $\pm$ and the remaining 7 bits for the value (e.g. voltage). This allows for 128 unique positive values to be assigned. For example, the value of 1.00 may be assigned the representation of 255 (11111111; FF Hexadecimal), zero the representation of 128 (10000000; 80 Hexadecimal) and -1.0 the representation of 127 (01111111; 7F Hexadecimal), allowing for a resolution of 0.0078125 along the y-axis.

The time interval $t_n - t_{n-1}$ is designated the *sampling interval*, Δ, and $1/\Delta$ is the *sampling rate* – a large sampling interval leads to a small sampling rate. There exists a special frequency, called the *Nyquist critical frequency*:

$$f_c \equiv \frac{1}{2\Delta} \quad \left(\text{e.g.} \quad \Delta \equiv \frac{\lambda_c}{2} = \frac{T(\text{period})}{2}\right) \quad \text{NOTE}: \quad f = \frac{\omega}{2\pi} \tag{24}$$

which determines the maximum allowable frequency that a signal can contain and expect to be correctly sampled. [Note that we will be using f to represent frequencies ($f = \omega/2\pi$) in the remainder of this section– we do this to have a clearer understanding of the relationship between frequency and Δ.] For example, if the sine wave being sampled has a period $T = 1$ sec, then, for critical sampling,

$$\frac{1}{\Delta} = \frac{2}{T} = 2 \quad \text{samples per second and} \quad \Delta = \frac{1}{2}\text{sec} = 0.5\text{sec} \tag{25}$$

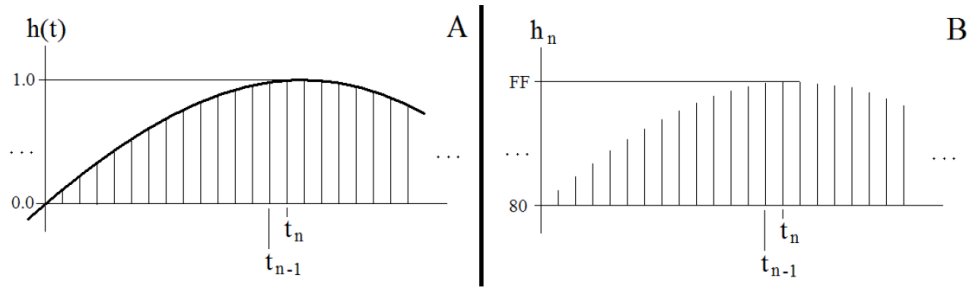


Figure 3: The function $h(t)$ (A) is sampled at a rate of $1/\Delta$, where $\Delta = t_n - t_{n-1}$. The resulting digital representation (B) is stored as a time series of values ranging from -128 (00) to 0 (80) to $+128$ (FF), with a resolution of $1/128$ (0.0078125).

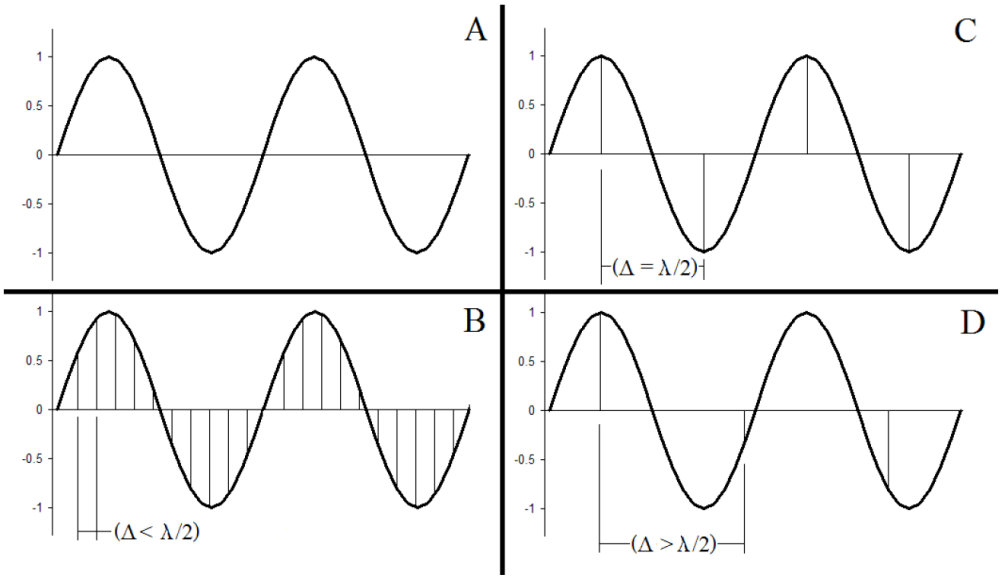


Figure 4: The sine wave in (A) is over sampled at a rate higher than the Nyquist frequency of this sine wave (B), sampled at the Nyquist frequency (C), and under sampled (D). The sampled values from (D) would not recover the original sine wave.

The sine wave at the Nyquist critical frequency will be sampled first at its peak, then at its trough, etc.– a sine wave can be recovered from the sampled data if it is originally sampled at least twice per period. Another way of thinking about it is that the choice of sampling interval will determine

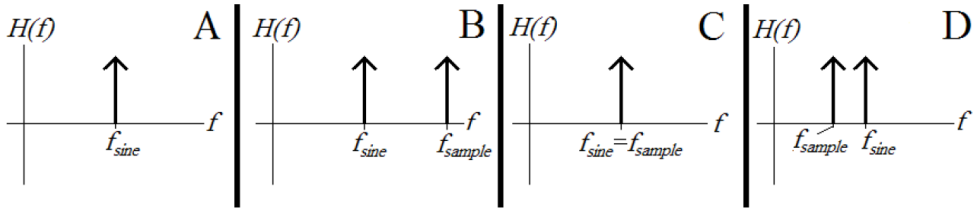


Figure 5: The frequency-space representation of Figures 4 A–D. In (B) the sample frequency is higher than the signal frequency and in (C), the sampling frequency equals the signal frequency, both cases meet Nyquist criteria. In (D), the signal frequency lies outside the sampling frequency, leading to erroneous signal representation.

the range of frequencies that can be accurately sampled. If the sampling rate is less than the highest Fourier harmonic sine wave frequency, the signal can not be recovered accurately (see Figure 4D). Figure 5 shows the frequency space representation of the signals found in Figure 4.

If a complicated signal composed of a sum of weighted sine waves (e.g. Equation 7) is band limited such that the signal does not contain frequencies greater than the Nyquist critical frequency, e.g.

$$H(f) = 0 \quad \text{for all} \quad |f| \geq f_c \tag{26}$$

then the function $h(t)$ is completely determined! This amazing result is known as the *sampling theorem*. In general, when a continuous signal is "under sampled", in which case there exist frequencies outside of the range $-f_c < f < f_c$ (see Figure 4D), then the original signal is *aliased* – frequencies outside of the range $-f_c < f < f_c$ are "forced" into this range by the intrinsic nature of the sampling procedure, resulting in a discrete representation of the signal that may be quite different from the original signal. In general, we will want to impose an analog band-pass filter to signals we wish to analyze prior to input into a sampler – the boundaries of the filter would be determined by the limits we wish to impose onto our sampling rate.

We are now equipped to describe the Fourier transform of a continuous function from the series of sampled discrete points.

Lets sample a continuous function $h(t)$ at N consecutive time intervals such that the discrete representation of that function is:

$$h_k = h(t_k) \quad \text{where} \quad t_k = k\Delta \quad \text{and} \quad k = 0, 1, 2, \ldots, N$$

Recall that the sampling interval is Δ. It is also assumed that the function is zero outside of the range $k = 0 \ldots N-1$; if not, we must assume that the function within the sampling range is indicative of the function outside of our range. The Fourier transform $H(f)$ will lie within the frequency range $-f_c$ to $+f_c$ and since we have N independent inputs (time samples), we can expect no more than N independent outputs at the following discrete frequencies:

$$f_n = \frac{n}{N\Delta} \quad \text{where} \quad n = -\frac{N}{2}, \ldots, \frac{N}{2}. \tag{27}$$

The limits on n correspond exactly to the boundaries of our Nyquist frequency, $-f_c$ to $+f_c$. Note that $H(-f_c) = H(+f_c)$ since there are N independent output values (not $N+1$, as is implied by equation 10.22). So, for example, if we sample $h(t)$ that spans a time interval of $T = 10$ seconds, and we sampled it $N = 1024$ times, then...

$$n = -\frac{1024}{2}, \ldots, \frac{1024}{2} = -512, \ldots, 512 \tag{28}$$

$$f_n = \frac{n}{N\Delta} = \frac{n}{N(T/N)} = \frac{n}{T} \tag{29}$$

so

$$f_1 = \frac{1}{10s} = 0.1\text{Hz} \ \ldots \ f_{10} = \frac{10}{10s} = 1.0\text{Hz} \ \ldots \ f_{f12} = \frac{512}{10s} = 51.2\text{Hz} \tag{30}$$

Recall, f_{512} is the Nyquist frequency, which can be computed as...

$$f_c = \frac{1}{2\Delta} = \frac{1}{2(T/N)} = \frac{N}{2T} = \frac{1024}{2(10)} = 51.2\text{Hz} \tag{31}$$

In other words, if the original signal contains frequencies higher than 51.2Hz, they will be aliased. To increase f_c , you should increase N (or decrease T if possible).

An Aside: What are negative frequencies?

The sign ($\pm$) of the frequency comes from the fact that, in angular motion, something can rotate either clockwise or counterclockwise (in other words, the sign determines its helicity). An observable signal such as that expressed as $\cos(\omega t)$, is actually comprised of both the negative and positive frequency contributions, as seen via:

$$\cos(\omega t) = \frac{e^{i\omega t} + e^{-i\omega t}}{2} = \frac{e^{i(+\omega)t} + e^{i(-\omega)t}}{2}. \tag{32}$$

In general, when we obtain the Fourier transform function, $H(\omega)$ (or $H(f)$), we generally think only in terms of the positive half of the spectrum.

We now approximate the Fourier transform integral (equation 10.11) as (note, here we are replacing ω with $2\pi f$ for clarity):

$$H(f_n) = \int_{-\infty}^{+\infty} h(t) \exp(i2\pi f_n t) dt \rightarrow \sum_{k=0}^{N-1} h_k \exp(2\pi f_n t_k)\Delta = \Delta \sum_{k=0}^{N-1} h_k \exp(i2\pi kn/N) \tag{33}$$

The *discrete Fourier transform* is simply the sum at the end of equation 33:

$$H_n \equiv \sum_{k=0}^{N-1} h_k \exp(i2\pi kn/N) \tag{34}$$

such that

$$H(f_n) = \Delta H_n. \tag{35}$$

This operation transforms N sampled points h_k into N values representing the Fourier transforms, H_n. The *discrete inverse Fourier transform* is then:

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n \exp(-i2\pi kn/N) \tag{36}$$

It should be noted that we imposed the range $0 \ldots N-1$ on both n and k. This means that the negative frequencies in the range $-f_c \ldots 0$ are indexed as $n = (N/2)+1 \ldots N-1$ (the positive frequencies are indexed as $n = 1 \ldots (N/2)-1$). Zero frequency (e.g. DC voltage), is indexed as $n = 0$ and the frequencies $-f_c$ and $+f_c$ are both indexed by $n = N/2$. The same symmetry properties for the continuous Fourier transform (15 – 18) apply here as well.

10.5 Fast Fourier Transform (FFT)

Although the DFT could be solved directly, there are more efficient ways of doing so. For example, to compute the DFT directly as described in the previous section, it would take N^2 calculations to complete (where N is the number of sampling points). The fast Fourier transform (FFT) is a computational algorithm that realizes the discrete Fourier transform utilizing

matrix symmetry and bit reversal to minimize the number of computer operations involved in the transform. The number of computations is decreased to $N \log_2 N$ which, for $N = 1024$, is an improvement of about 100x. What do we mean by matrix symmetry? If we allow the following definition:

$$W = \exp(i2\pi/N) \tag{37}$$

then we can rewrite equation 34 in matrix notation as:

$$H_n \equiv \sum_{k=0}^{N-1} W^{nk} h_k. \tag{38}$$

This resulting H_n can be written as the sum of two terms – one containing only the even-numbered points of the original N and the other containing only the odd-numbered points of the original N. It can be shown that, for an arbitrary function f for which:

$$F_k = \sum_{j=0}^{N-1} \exp(i2\pi jk/N) f_j \quad \text{(from equation 34)} \tag{39}$$

then

$$F_k = F_k^e + W^k F_k^o \tag{40}$$

Where F_k^e and F_k^o are the constituent matrices of the transform formed from the even components and odd components of the originally sampled function, respectively:

$$F_k^e = \sum_{j=0}^{N/2-1} \exp(i2\pi jk/(N/2)) f_{2j} \tag{41}$$

$$F_k^o = \sum_{j=0}^{N/2-1} \exp(i2\pi jk/(N/2)) f_{2j+1} \tag{42}$$

In other words, we have broken the original Fourier transform operation on N samples into two Fourier transforms operating on $N/2$ samples. But there is nothing stopping us from applying this same even-odd symmetry technique recursively to F_k^e and F_k^o again, and again, and again, until all we are left

with is a set of single-point transforms! This method was documented by Danielson and Lanczos in 1942 (termed the Danielson-Lanczos lemma) and more details about how it works is discussed in detail in *Numerical Recipes* (2nd Edition, Chapter 12). To use this method, the only restriction that we must respect is that we require N to be an integer power of 2 . For example, if we have 500 samples, we would use a value of $N = 2^9 = 512$ and fill the remaining 12 time interval values with zeros – this is called "zero padding".

Exercise 10.3: A Rudimentary Low-Pass Filter

Fourier transforms can be used to filter high-frequency noise from a signal. Consider Figure 6. Figure 6 shows a stellar spectrum (top) of the star UHA21 obtained at the Dark Sky Observatory. The fuzz in the spectrum is noise due to the fact that the star is faint, too faint to get a good signal to noise (S/N) ratio in the time available for exposure. However, the noise in the spectrum is generally high frequency whereas the spectral information (the shape of the continuum, the strength and profile of the absorption lines) is lower frequency.

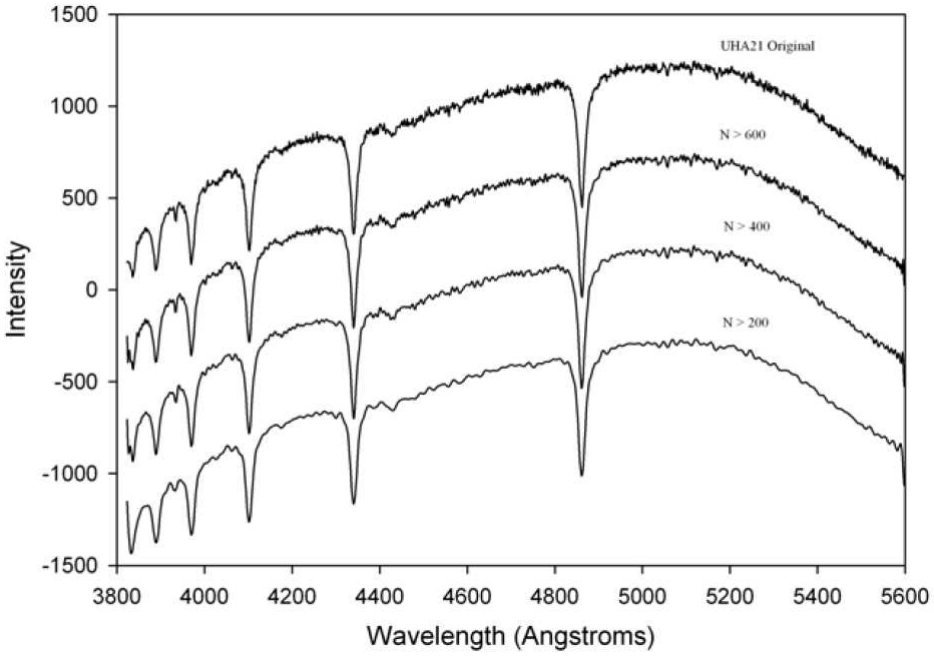


Figure 6:

This suggests a strategy for getting rid of the noise in the spectrum: take the Fourier transform of the spectrum, zero out the high-frequency tail and then take the inverse transform. The result should be a spectrum with less noise. This strategy actually works. Figure 7 shows the Fourier transform of the spectrum:

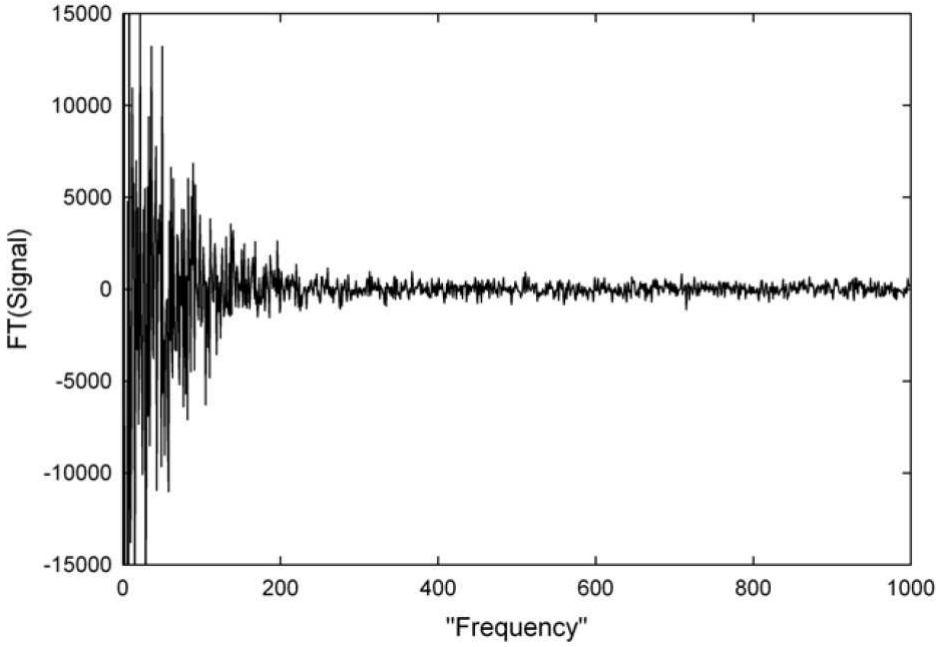


Figure 7:

The spikes in the low frequency part of this diagram represent the information content of the spectrum and the long tail going out to higher frequencies represents, for the most part, noise in the stellar spectrum. Your task for this exercise is as follows:

1) Read in the stellar spectrum UHA21.nor with wavelength in one vector (call it `wave`) and the intensity in another vector (call it `y`).
2) Zero-pad `y` to some integer power of 2.
3) Use `realft` to take the Fourier transform of `y`.
4) Using a value of `N` input by the user, zero out the Fourier transform of `y` for `i > N`.
5) Take the inverse transform using `realft`.

Unfortunately, this simple scheme runs into a problem. When ever you zero-out part of a Fourier transform, you lose power or amplitude in the original signal. To compensate for this, you can integrate over the entire original spectrum, and then do the same for the processed spectrum. Take a ratio of the two integrals and multiply the processed spectrum by this ratio to restore the original power. The following bits of code will do the trick:

Before using `realft` to obtain the Fourier transform:

```
for(j=1;j<=m;j++) sum += y[j];
```

After using `realft` to obtain the inverse Fourier transform:

```
for(j=1;j<=m;j++) sum1 += y[j];
for(j=1;j<=m;j++) y[j] *= sum/sum1;
```

Your results, for different values of `N` should look like Figure 6.
Here is the program description for function `realft` from *Numerical Recipes*:

`void realft(float data[], unsigned long n, int isign)`
Calculates the Fourier transform of a set of n real-valued data points. Replaces this data (which is stored in array `data[1..n]`) by the positive frequency half of its complex Fourier transform. The real-valued first and last components of the complex transform are returned as elements `data[1]` and `data[2]` , respectively. `n` must be a power of 2. This routine also calculates the inverse transform of a complex data array if it is the transform of real data. (Result in this case must be multiplied by `2/n`).

10.6 Convolution

The FFT is a powerful tool, but by itself falls short of solving many different types of problems that arise in physics and engineering. Here, we introduce three additional tools that employ the analytical and computational methodologies of the Fourier transform for specific types of scientific and engineering applications: convolution, correlation, and auto-correlation.

The first of these is the *convolution* function. The convolution of two functions is defined as:

$$h(t) * r(t) = \int_{-\infty}^{\infty} h(\tau) r(t-\tau) d\tau \tag{43}$$

Normally, $h(t)$ is an arbitrary signal and $r(t)$ is what is known as a "response function" (to be described momentarily). Since the functions $h(t)$ and $r(t)$ have Fourier transforms $H(\omega)$ and $R(\omega)$ respectively, then the *Convolution Theorem* states that:

$$H(\omega)R(\omega) = \mathsf{F}[h(t) * r(t)] \tag{44}$$

In other words, the Fourier transform of the convolution of two functions is simply the product of their individual Fourier transforms.

In general, the convolution acts to smear out the input function $h(t)$ in time. For example, if the input function has numerous impulse functions or discontinuities imbedded within it, these could be smoothed out as dictated by the nature of $r(t)$. Figure 8 illustrates an input signal with discontinuities. The response function is chosen to be a short, symmetrical function with a smoothly peaked center. The convolution of these functions yields a signal that still contains the original information, but it is in a form that may be more conducive to the application of signal analysis techniques.

How does the Fourier transform come into play? To answer that, lets look at an example from electrical engineering. We may need to know the response of a linear, time-invariant system to an arbitrary input signal. Here, the function $h(t)$ would represent the input signal and $r(t)$ would represent the linear system's "unit impulse response" (a.k.a. Greens function or characteristic function). The convolution of $h(t)$ with $r(t)$ would then provide the system's response to $h(t)$. However, most signals and response functions are non-trivial, leading to a difficult integral to evaluate (per equation 1) at best. If we were to take the Fourier transform of $h(t)$ and $r(t)$ first, we could simply multiply $H(\omega)$ and $R(\omega)$ per equation 44 in the frequency domain. All that would remain would be to apply the inverse Fourier transform to their products. This is also non-trivial analytically, but with the FFT as a tool for numerical evaluation of the Fourier- and inverse Fourier transforms, the procedure is greatly simplified.

Here the differential equation describing the circuit is:

$$\frac{dv_{\text{out}}}{dt} + \frac{v_{\text{out}}}{RC} = \frac{v_{\text{in}}}{RC}$$

Solving for v_{out}, where v_{in} is non-zero from $0 \leq \tau \leq t$, we get:

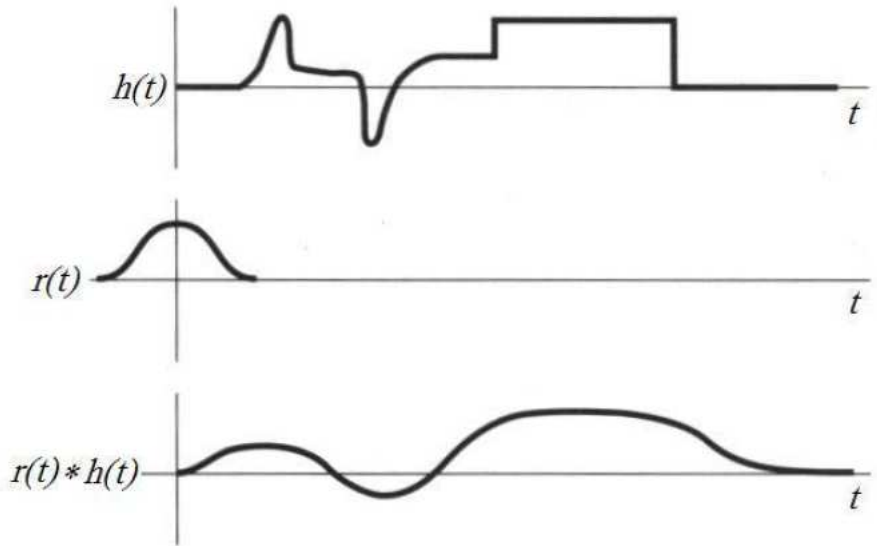


Figure 8: The input signal is convolved with the response function, yielding a function that is "smeared out" in time. The edges of discontinuities in the input signal are smoothed. Adapted from *Numerical Recipes*.

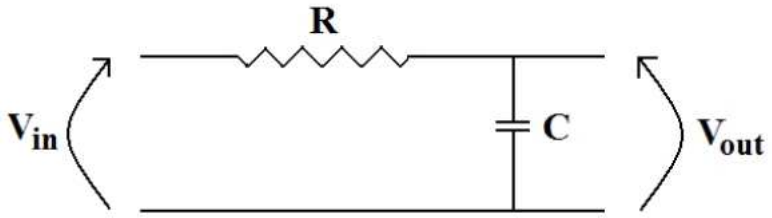


Figure 9: A simple RC circuit used in the example. The input voltage can be any aperiodic signal. The characteristic function is derived in the text.

$$v_{\rm out}(t) = \int_0^t v_{\rm in}(\tau)\exp[-(t-\tau)/RC]d\tau$$

But we recognize this as:

$$v_{\rm out}(t) = v_{\rm in}(t) * r(t)$$

where r(t) is the circuit's characteristic function

$$r(t) = e^{-t/RC} \tag{45}$$

which has a Fourier transform of (this transform is so common, it is usually listed in tables within related literature, e.g. Oppenheim et al., 1983):

$$R(\omega) = \frac{1}{1/RC - i\omega}$$

Then we can take the inverse Fourier transform via FFT of $V_{\text{out}}(\omega)$ to obtain $v_{\text{out}}(t)$.

Exercise 10.4: Fortunately, our job is simplified even more by the existence of available convolution algorithms. One such routine is the Numerical Recipes C function `convlv`. This routine performs all of the Fourier transforms for us – all that is needed is the time series of $h(t)$ and $r(t)$, their array lengths, a control parameter (`isign`) to indicate whether we want the convolution (or deconvolution), and a predefined array (`ans`) to store the result:

```
void convlv (float data[], unsigned long n, float respns[],
unsigned long m, int isign, float ans[]);
```

The Numerical Recipes description of this function is as follows:

Convolves or deconvolves a real data set `data[1..n]` (including any user-supplied zero-padding) with a response function `respns[1..m]`. The response function must be stored in wrap-around order in a real array of length $m \leq n$. Wrap-around order means that the first half of the array `respns` contains the impulse response function at positive times, while the second half of the array contains the impulse response function at negative times, counting down from the highest element `respns[m]`. On input `isign` is $+1$ for convolution, -1 for deconvolution. The answer is returned in the first `n` components of `ans`. However, `ans` must be supplied in the calling program with dimension `[1..2*n]`, for consistency with `twofft`. `n` MUST be an integer power of 2.

Use the `convlv` routine to find the output voltage time series of the RC circuit in Figure 8 with $R = 1\text{k}\Omega$ and $C = 1\text{mF}$ ($RC = 1$). Define $h(t) = V_{\text{in}}$

as 1.0V in the range $0 \leq t < 1$ and the response function, $r(t)$, from equation 3. Set $n = m = 128$ and choose $dt = 0.1$s such that your time range goes from -6.4 to +6.4 seconds. Plot the answer $V_{\rm out}$ using Gnuplot.

Exercise 10.5: It should not be surprising that astronomy offers up many examples of the need to perform convolution. One excellent example is that of a spectrograph. A spectrograph observes the spectrum of a star, but the output spectrum is a convolution of that spectrum and the response function (called the line-spread function) of the spectrograph. Essentially, because of the wave nature of light, the spectrograph convolves a Gaussian function with the spectrum. The Gaussian function is given by:

$$r(\lambda) = \exp(-\lambda^2/2\sigma)$$

where σ = FWHM/2.355 where FWHM is the full width of the Gaussian function at half maximum. In the shared directory, you can find a very high-resolution spectrum of the sun, `solar6100.dat` (that file has two columns, wavelength in Ångstroms in the first and intensity in the second). Take the intensity to be $h(\lambda)$ and the above Gaussian function to be $r(\lambda)$. Using FWHM = 0.2Å, convolve $h(\lambda)$ with $r(\lambda)$ and plot both the original and the convolved spectrum out to the screen using gnuplot. Note: to get the original and final spectra to agree in continuum height (the continuum height on the original spectrum is normalized to 1.0), you will have to do the same sort of normalizing procedure that you did in Ex 10.3.

10.7 Correlation and Auto-correlation

The *correlation* between two functions is a measure of their similarity – if two functions are very similar, then they have a high correlation. The correlation is defined as:

$$\mathrm{Corr}(p, q) = \int_{-\infty}^{\infty} p(\tau + t) q(\tau) d\tau \tag{46}$$

This corresponds to a transform pair:

$$\mathrm{Corr}(p, q) \leftrightarrow P(\omega) Q^*(\omega) \tag{47}$$

The correlation is similar to the convolution but simpler in a fundamental way: the functions in the correlation do not represent a signal and response as

in the convolution, but rather just two functions that are usually suspected of having some measurable similarity. For example, a transmitted radar signal is expected to have some similarity to its corresponding time-delayed received signal. Figure 10 shows an example of a received signal (time goes left to right) – at first glance it appears to be just a noisy signal of no significance. Figure 11 shows the correlation between the transmitted signal and the received signal (time delayed), obtained analytically. Figure 12 shows the same correlation but with an actual received signal that includes scattering effects from a target. As is apparent, the correlation operator is an invaluable tool for signal recovery for active systems such as radar and sonar, but can also be applied to any two or more signals whose change with respect to time may be of interest.

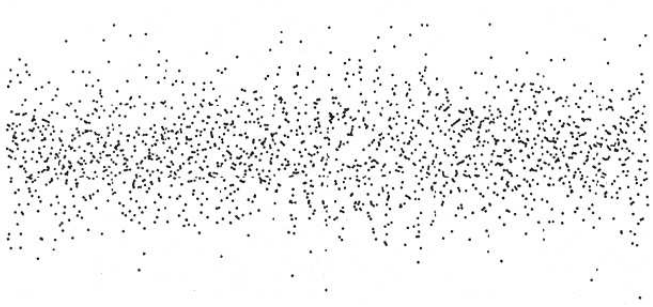


Figure 10: A received radar signal (sampled at a fixed rate) – without processing, this signal appears to contain no information.

Autocorrelation is simply the correlation of a function with itself. The same mathematical techniques and numerical routines that apply to correlation apply to autocorrelation. Why would we want to perform an autocorrelation? First of all, autocorrelation helps to illustrate an important property of the Fourier transform. The "Weiner-Khinchin Theorem" states:

$$\text{Corr}(p, p) \leftrightarrow |P(\omega)|^2 \tag{48}$$

The total power of a signal is the same, whether we compute it in the time domain or the frequency domain:

$$\text{Power} = \int_{-\infty}^{\infty} |p(t)|^2 dt = \int_{-\infty}^{\infty} |P(\omega)|^2 d\omega \tag{49}$$

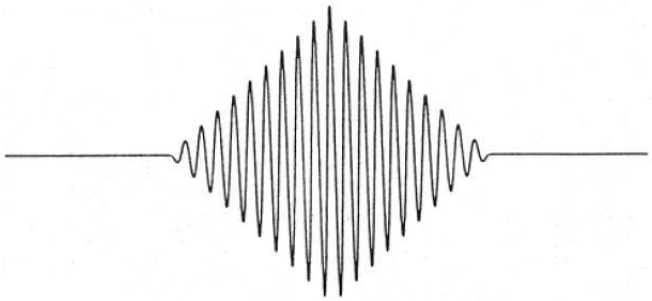


Figure 11: The correlation between the transmitted and received signals, obtained analytically.

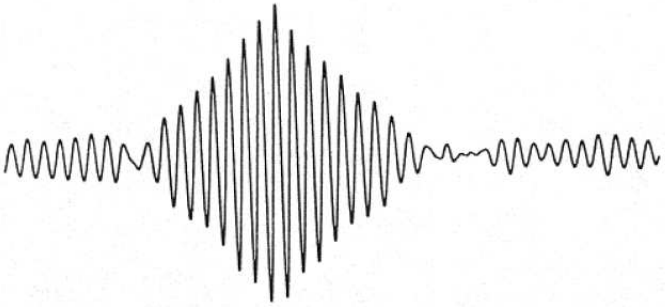


Figure 12: The correlation between the transmitted and received signals, indicating a scattering pattern off of a target.

Autocorrelation provides another way of computing total power of a signal, especially if the time-domain representation is cumbersome.

In signal processing, the autocorrelation method is commonly used for analyzing functions or series of values, such as time domain signals. The autocorrelation represents the "strength" of a relationship between two (usually consecutive) observations. Autocorrelation is useful for finding repeating patterns in a signal, such as determining the presence of a periodic signal which has been buried under noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies. In Figure 13, the signal on the left is a scatter plot of noisy data. The autocorrelation of the signal, seen on the right, produces a clean representation of the oscillatory signal.

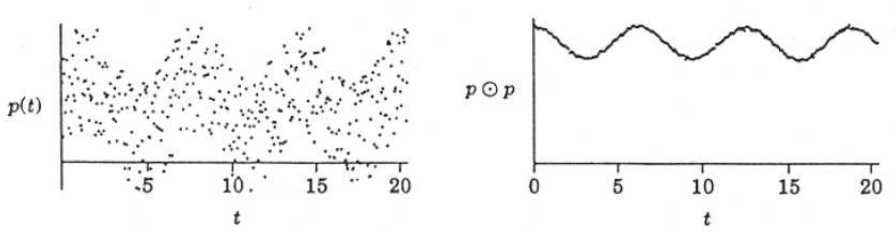


Figure 13: On the left, a scatter plot of noisy data. On the right, its autocorrelation.

The *Numerical Recipes* function `correl` enables us to find the correlation between two signals. The definition for the use of `correl` is given by:

```
void correl(float data1[], float data2[], unsigned long n,
float ans[])
```
Computes the correlation of two real data sets `data1[1..n]` and `data2[1..n]` (including any user-supplied zero padding). `n` MUST be an integer power of two. The answer is returned as the first `n` points in `ans[1..2*n]` stored in wrap-around order, i.e. correlations at increasingly negative lags are in `ans[n]` on down to `ans[n/2+1]`, while correlations at increasingly positive lags are in `ans[1]` (zero lag) on up to `ans[n/2]`. Note that `ans` must be supplied in the calling program with length at least `2*n`, since it is also used as working space. Sign convention of this routine: if `data1` lags `data2`, i.e., is shifted to the right of it, then `ans` will show a peak at positive lags.

Exercise 10.6: Write a program using the function `correl` (found in `comphys.c`) that will find the correlation of any two datasets in the collection `cdata1.dat, cdata2.dat, cdata3.dat` which can be found in the `shared` directory or on the class website. The data files all contain two columns, the first column representing the "time", and the second column the signal. The program should prompt the user for the names of the two data files to be tested for correlation. Two of the data sets are, indeed, strongly correlated, whereas the third data set is not. The program should use gnuplot to plot out the correlation (using the prescription in the description of `correl` above. Find out which two datasets are correlated. Choose one of them and use your program to find its autocorrelation.

References:

Cooley, J.W. and J.W. Tukey, 1965, An algorithm for the machine calculation of complex Fourier series, Mathematics of Computation, 19(90), 297-301

Oppenheim, A.V, A.S. Willsky, I.T. Young, 1983, Signals and Systems, Prentice-Hall, Englewood Cliffs, New Jersey

Press, W.H., W.T. Vetterling, S.A. Teulolsky, B.P. Flannery, 1996, Numerical Recipes in C, The Art of Scientific Computing, 2nd Edition, Cambridge University Press

Ramo, S., J.R. Whinnery, T. Van Duzer, 1965, Fields and Waves in Communication Electronics, 2nd Edition, John Wiley & Sons, New York

Roland, C., C.S. Thaxton, 1998, Personal notes and correspondence on computational physics, available upon request

Su, K.L., Fundamentals of Circuits, Electronics, and Signal Analysis, 1978, Houghton Mifflin Co. Boston