Chapter 1

Discrete Fourier Transform

We usually think about processes around us as functions of time. However, it is often useful to think about them as functions of frequencies. We naturally do this without giving it a second thought. For example, when we listen to someone's speech, we distinguish one person from another by the pitch, i.e. dominating frequencies, of the voice. Similarly, our eyes do a similar time-to-frequency transformation, as we can distinguish different light colors, and the colors themselves are directly connected to the frequency of light. When we tune a radio, we select a particular subset of frequencies to listen in the time varying signal of electromagnetic waves. Even when we talk about salary, it is often enough to know that it will come every two weeks or a month, i.e. we are concerned with the period and, thus, frequency of the wages.

The transformations from the time domain to the frequency domain and back are called the forward[1] Fourier and inverse Fourier transforms respectively. The transformation is general and can broadly be applied to any variable, not just a time variable. For example, we can do the transformation from spatial coordinates to the spatial coordinates' frequencies, which is the base of the JPEG image compression algorithm.

The Fourier transform provides the basis for many filtering and compression algorithms. It is also an indispensable tool for analyzing noisy data. Many differential equations can be solved much easier in the periodic oscillation basis. Additionally, the calculation of convolution integrals of two functions is very fast and simple when it is done in the frequency domain.

1.1 Fourier series

It is natural to think that a periodic function, such as the example shown in fig. 1.1, can be constructed as the sum of other periodic functions. In Fourier series, we do it in the basis of

[1]The word 'forward' is often omitted.

sines and cosines, which are clearly periodic functions.

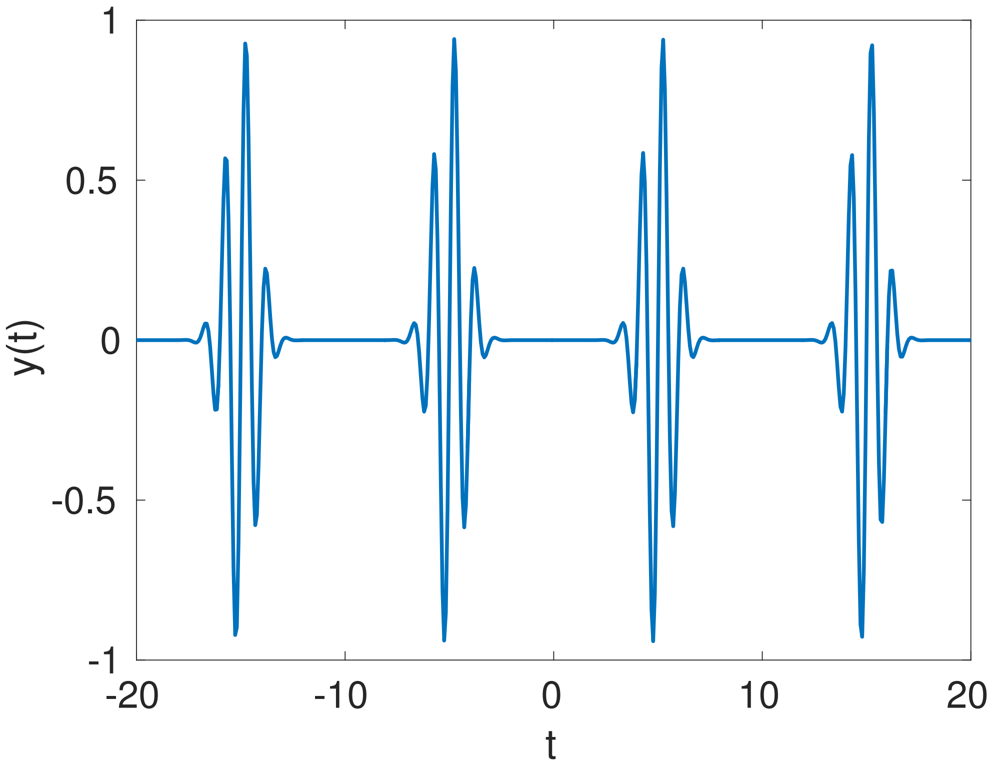

Figure 1.1: Example of a periodic function with the period of 10.

A more mathematically solid definition of a function that can be transformed is: any periodic single value function $y(t)$ with a finite number of discontinuities and for which $\int_0^T |f(t)|dt$ is finite can be presented as

Fourier series

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} (a_n \cos(n\omega_1 t) + b_n \sin(n\omega_1 t)) \tag{1.1}$$

where T is the period, i.e. $y(t) = y(t+T)$, and $\omega_1 = 2\pi/T$ is the fundamental angular frequency, constant coefficients a_n and b_n can be calculated according to the following formula

$$\begin{pmatrix} a_n \\ b_n \end{pmatrix} = \frac{2}{T} \int_0^T \begin{pmatrix} \cos(n\omega_1 t) \\ \sin(n\omega_1 t) \end{pmatrix} y(t)dt \tag{1.2}$$

At a discontinuity, the series approaches the midpoint, i.e.

$$y(t) = \lim_{\delta \to 0} \frac{y(t-\delta) + y(t+\delta)}{2} \tag{1.3}$$

Note that for any integer n, $\sin(n2\pi/Tt)$ and $\cos(n2\pi/Tt)$ have the period of T.

The calculation of a_n and b_n according to eq. (1.2) is called forward Fourier transformation and construction of the $y(t)$ via series of eq. (1.1) is called inverse Fourier transform.

The validity of the transformation can be shown by the use of the following relationship

$$\frac{2}{T}\int_0^T \sin(nw_1 t)\cos(mw_1 t)dt = 0, \text{ for any integer } n \text{ and } m \tag{1.4}$$

$$\frac{2}{T}\int_0^T \sin(nw_1 t)\sin(mw_1 t)dt = \delta_{nm}, \tag{1.5}$$

$$\frac{2}{T}\int_0^T \cos(nw_1 t)\cos(mw_1 t)dt = \begin{cases} 2, & \text{for } n = m = 0 \\ \delta_{nm}, & \text{otherwise} \end{cases} \tag{1.6}$$

Note that $a_0/2$, according to eq. (1.2), is equal

$$\frac{1}{2}a_0 = \frac{1}{2}\frac{2}{T}\int_0^T \cos(0w_1 t)y(t)dt = \frac{1}{T}\int_0^T y(t)dt = \overline{y(t)} \tag{1.7}$$

Thus, $a_0/2$ has a special meaning: it is the mean of the function over its period, i.e. the base line, the DC offset, or the bias.

Also, a_n coefficients belong to cosines, thus they are responsible for the symmetrical part of the function (after offset removal). Consequently, b_n coefficients are in charge of the asymmetrical behavior.

Since each a_n or b_n coefficient corresponds to the oscillatory functions with the frequency $n\omega_1$, the set of a and b coefficients is often called *spectrum* when it is shown as the dependence on frequency.

Example: Fourier series for |t|

Let's find the Fourier series representation of the following periodic function

$$y(t) = |t|, \quad -\pi < t < \pi$$

Since the function is symmetrical, we can be sure that all $b_n = 0$. The a_n coefficients are found by applying eq. (1.2):

$$\begin{cases} a_0 = \pi, & \\ a_n = 0, & \text{for even } n \\ a_n = -\frac{4}{\pi n^2}, & \text{for odd } n \end{cases}$$

Their values are shown in fig. 1.2. As we can see $a_0 = \pi$ is twice the mean of the $|t|$ on the $(-\pi, \pi)$ interval.

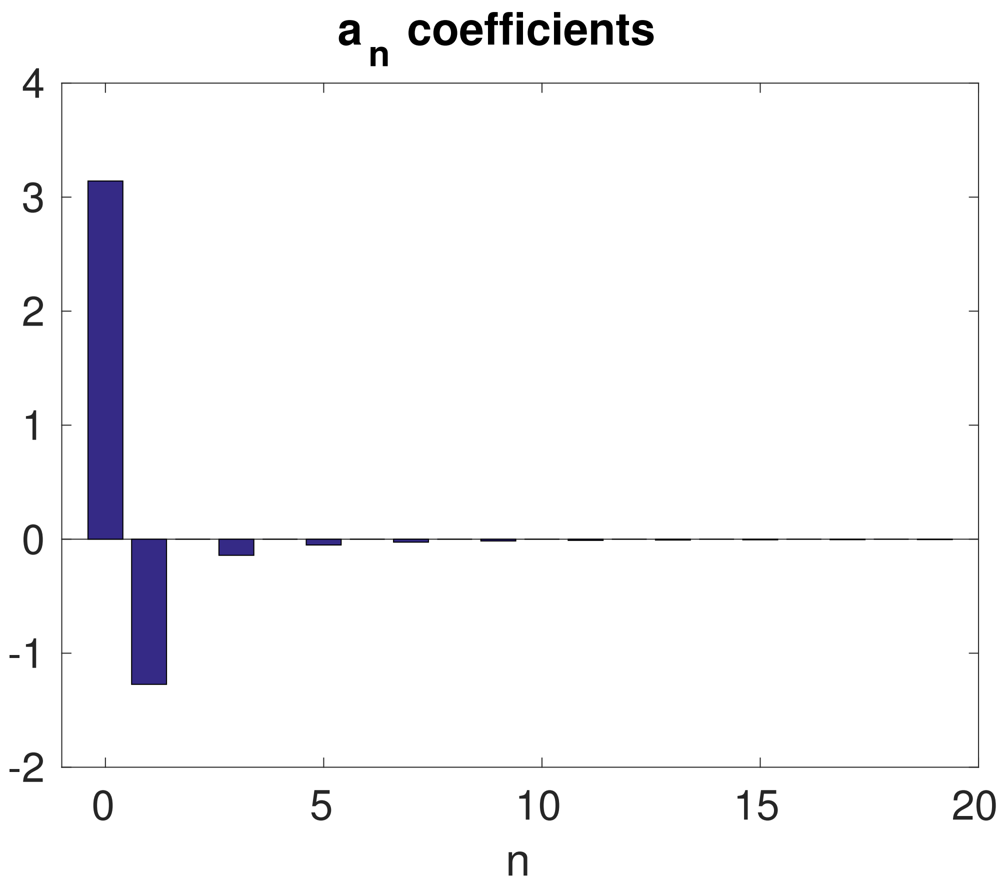

Figure 1.2: The first 20 a_n coefficients of the $|t|$ Fourier transform.

We can notice from fig. 1.3 that the a_n coefficients decrease very quickly as n grows, implying that the contribution of the higher n terms vanishes very quickly. Therefore, we can get quite a good approximation of $|t|$ with a truncated Fourier series.

This observation provides a basis for information compression. It is enough to know only the first 11 coefficients (by the way, half of them are zero) of the Fourier transform to reconstruct our function with minimal deviations from its true values at all possible times. If we need better precision, we can increase the number of the Fourier series coefficients.

Example: Fourier series for the step function

Let's find the Fourier series for the step function defined as

$$\begin{cases} 0, & -\pi < x < 0, \\ 1, & 0 < x < \pi \end{cases}$$

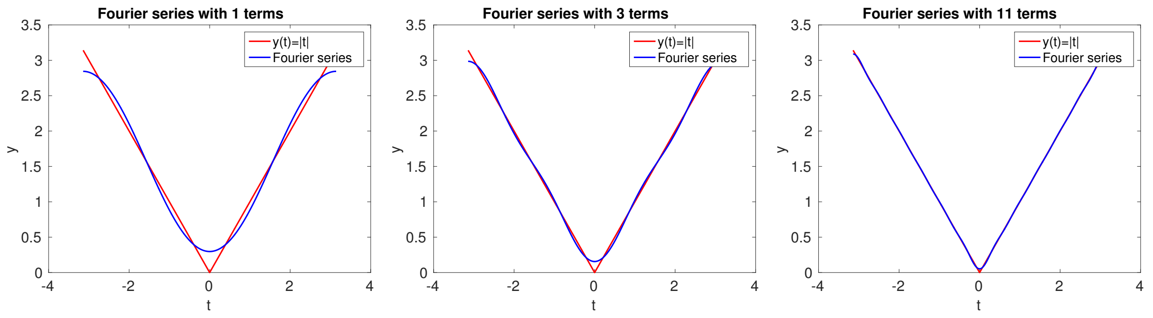

Figure 1.3: Approximation of the $|t|$ function by truncating the Fourier series at maximum $n = 1$ (left), $n = 3$ (middle), $n = 11$ (right).

This function is asymmetric (with respect to its mean value 1/2), thus all $a_n = 0$ except the $a_0 = 1$. The b_n coefficients are

$$\begin{cases} b_n = 0, & n \text{ is even} \\ b_n = \frac{2}{\pi n}, & n \text{ is odd} \end{cases}$$

The values of the b_n coefficients are shown in fig. 1.4.

The b_n coefficients decrease at an inversely proportional rate to n. Therefore, we can hope that we can truncate the Fourier series and still get a good approximation of the step function. The result of truncation is shown in fig. 1.5. The coefficients do not drop as quickly as in the previous example; thus, we need more members in the Fourier series to get a good approximation. You may notice that at discontinuities where $t = -\pi, 0, \pi$ the approximation passes through midpoints $y = 1/2$ as we promised in section 1.1. You may also notice a strange overshot near the discontinuities (ringing-like behavior). You may think this is the result of having a small number of members in the Fourier series, but it will not go away if we increase the number of expansion terms. This is known as the Gibbs phenomenon. Nevertheless, we have a good approximation of the function with a very small number of coefficients.

1.1.1 Complex Fourier series representation

Recall that

$$\exp(i\omega t) = \cos(\omega t) + i \sin(\omega t)$$

It can be shown that we can rewrite eqs. (1.1) and (1.2) in the much more compact and symmetric notation without the eye-catching 1/2 for the zero's coefficient:

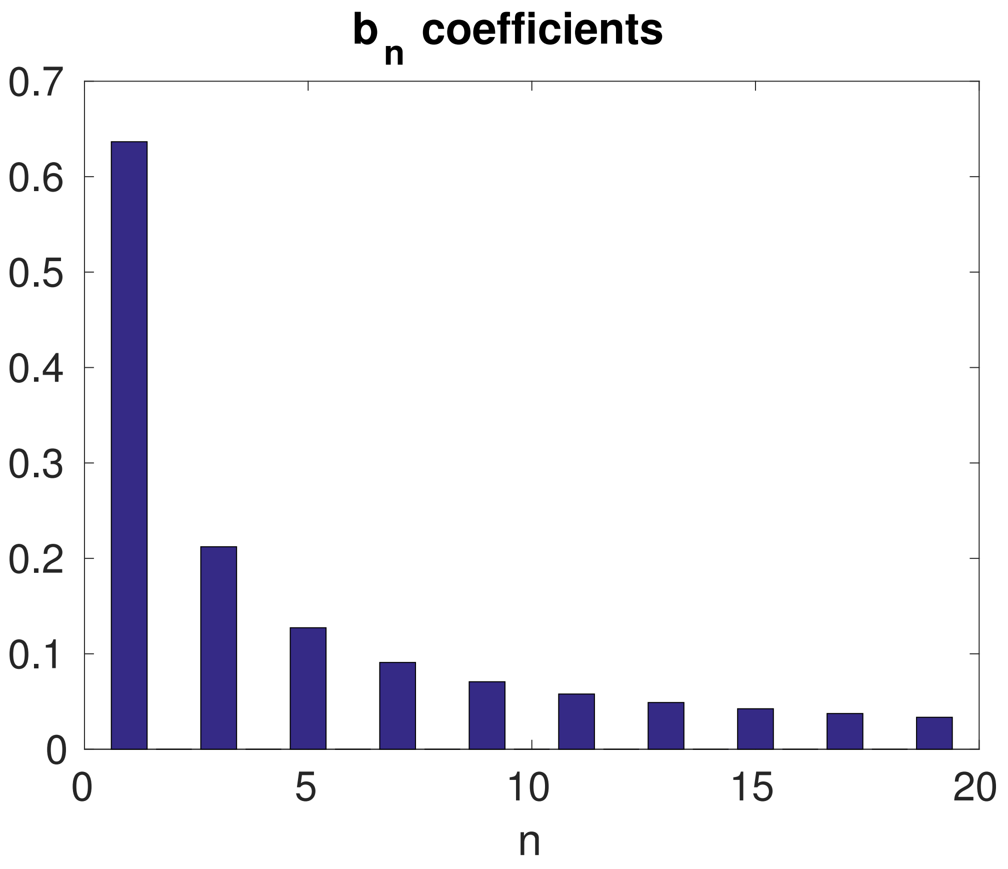

Figure 1.4: The first 20 b_n coefficients of the step function transform.

Complex Fourier series

$$y(t) = \sum_{n=-\infty}^{\infty} c_n \exp(in\omega_1 t) \tag{1.8}$$

$$c_n = \frac{1}{T}\int_0^T y(t)\exp(-i\omega_1 nt)dt \tag{1.9}$$

The c_0 has the special meaning: the average of the function over the period or bias or offset.

The following connection exists between a_n, b_n, and c_n coefficients:

$$a_n = c_n + c_{-n} \tag{1.10}$$

$$b_n = i(c_n - c_{-n}) \tag{1.11}$$

You might ask yourself: what are those ‘negative’ frequencies components for which c_{-n} is

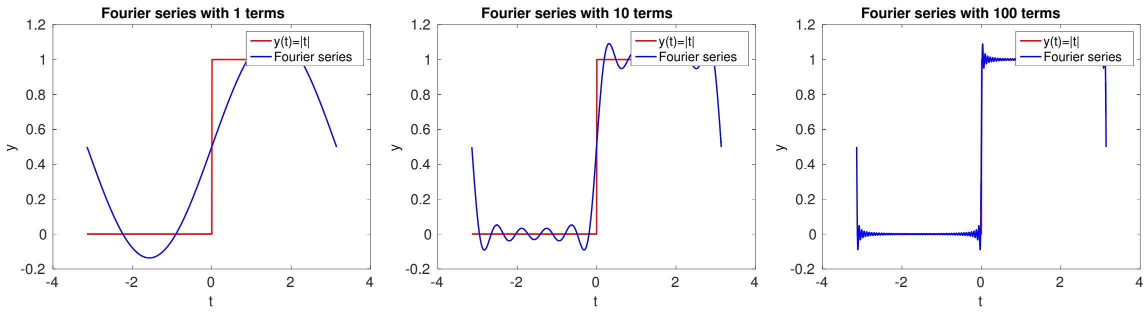

Figure 1.5: Approximation of the step function by truncating the Fourier series at maximum $n = 1$ (left), $n = 10$ (middle), $n = 100$ (right).

in charge? This does not look physical. It should not worry us too much, since $\cos(-\omega t) = \cos(\omega t)$ and $\sin(-\omega t) = -\sin(\omega t)$, i.e. it is just a flip of the sign of the corresponding sine coefficient. We do this to make the forward and inverse transforms look alike.

1.1.2 Non periodic functions

What to do if function is not periodic? We need to pretend that it is periodic but on a very large interval, i.e. $T \to \infty$. Under such an assumption, our discrete transform becomes a continuous one, i.e. $c_n \to c_\omega$. In this case, we can approximate the sums in eqs. (1.8) and (1.9) with integrals[2]

Continuous Fourier transform

$$y(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} c_\omega \exp(i\omega t) d\omega \tag{1.12}$$

$$c_\omega = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} y(t) \exp(-i\omega t) dt \tag{1.13}$$

The above requires that $\int_{-\infty}^{\infty} y(t)dt$ exists, and it is finite.

Note that the choice of the normalization $1/\sqrt{2\pi}$ coefficient is somewhat arbitrary. In some literature, the forward transform has the overall coefficient of 1 and the inverse transform has coefficient of $1/2\pi$. The opposite is also used. Physicists like the symmetric form shown above in eqs. (1.12) and (1.13).

[2]Here, we do the opposite to the rectangle integration method discussed in section 9.2.

1.2 Discrete Fourier transform (DFT)

Our interest in the above material is somewhat academic only. In real life, we cannot compute the infinite series, since it takes an infinite amount of time. Similarly, we cannot compute the exact integrals required for the forward Fourier transform, as doing so requires knowledge of the $y(t)$ at every point of time, which necessitates that we need an infinite amount of data to do the calculation. You might say that we have quite good methods to approximate the value of integrals covered in chapter 9, but then we automatically limit ourself to a finite set of points in time where we do the measurement of the $y(t)$ function. In this case, we do not need the infinite sum, it is enough to have only N coefficients of the Fourier transform to reconstruct N points of $y(t)$.

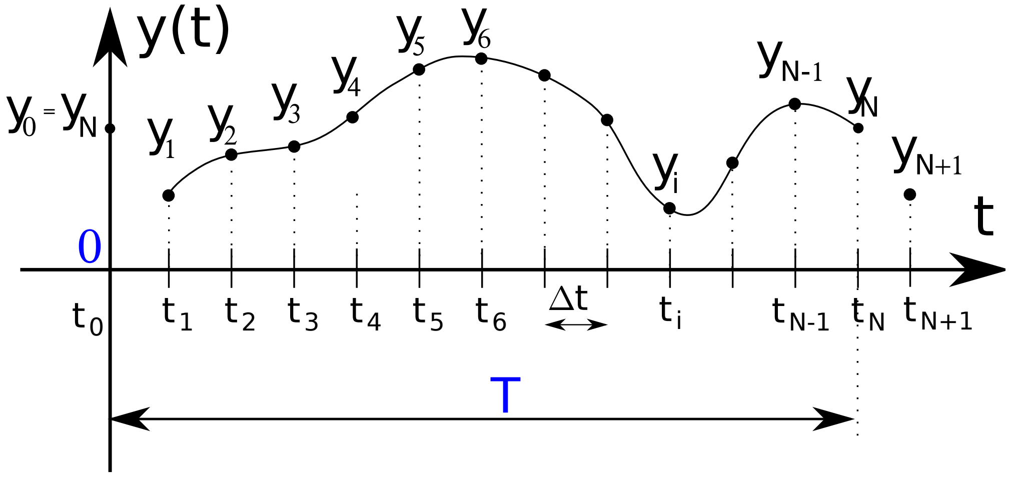

Figure 1.6: An example of discretely sampled signal $y(t)$ with period T.

Assuming that $y(t)$ has a period T, we took N **equidistant** points such that spacing between them is $\Delta t = T/N$ (see fig. 1.6). The periodicity condition requires

$$y(t_{k+N}) = y(t_k) \tag{1.14}$$

We use the following notation $t_k = \Delta t \times k$ and $y_k = y(t_k)$ and define (see for example [1])

Discrete Fourier Transform (DFT)

$$y_k = \frac{1}{N} \sum_{n=0}^{N-1} c_n \exp(i\frac{2\pi(k-1)n}{N}), \text{ where } \; k = 1, 2, 3, \cdots, N \tag{1.15}$$

$$c_n = \sum_{k=1}^{N} y_k \exp(-i\frac{2\pi(k-1)n}{N}), \text{ where } n = 0, 1, 2, \cdots, N-1 \tag{1.16}$$

Notice that the above equations do not have time in them at all! Strictly speaking, the DFT uniquely connects one periodic set of points with another; the rest is in the eye of the beholder. The notion of the spacing is required when we need to decide what is the corresponding frequency of the particular c_n: $f_1 \times n$, where $f_1 = T/N$ is the spacing (Δf) in the spectrum between two nearby c coefficients (see fig. 1.7). The other meaning of f_1 is the *resolution bandwidth* (RBW), i.e. by construction, we cannot resolve any two frequencies with a spacing smaller than the RBW. The $f_s = 1/\Delta t$ is called the *sampling frequency* or the *acquisition frequency*. The Nyquist frequency $f_{Nq} = f_s/2 = f_1 N/2$ has a very important meaning which we will discuss later in section 16.1.

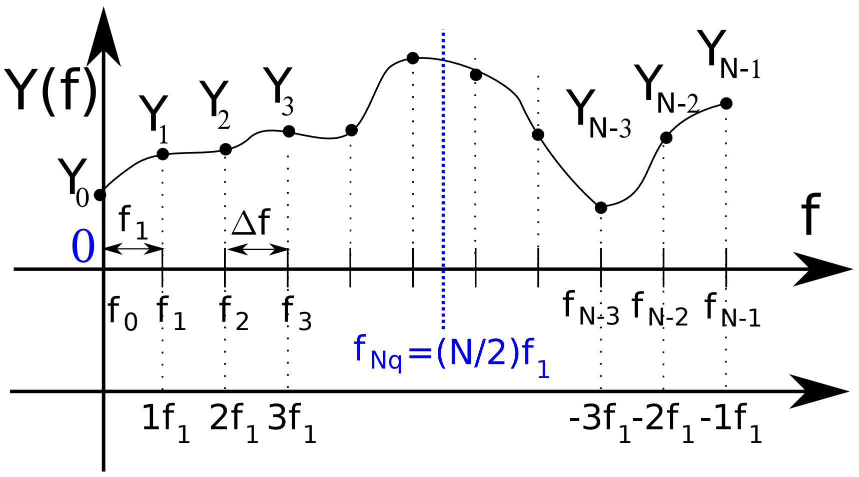

Figure 1.7: Sample spectrum: Fourier transform coefficient vs. frequency. Y_k is the same as c_k

Note the canonical placement of the normalization coefficient $1/N$ in eq. (1.15) instead of eq. (1.16). With this definition, c_0 is not the average value of the function anymore; it is N times larger[3]. Unfortunately, pretty much every numerical library implements the DFT in this particular way, and MATLAB is not an exception.

There are several properties of the c_n coefficient; the proof of which is left as exercise for the reader. The c coefficients are periodic

$$c_{-n} = c_{N-n} \tag{1.17}$$

A careful reader would notice that c_n and c_{-n} coefficients are responsible for the same absolute frequency $f_1 \times n$. Therefore, the spectrum is often plotted from -$N/2 \times f_1$ to $N/2 \times f_1$. It also has an additional benefit: if all y_k have no imaginary part, then

$$c_{-n} = c_n^* \tag{1.18}$$

[3] This is what happens when mathematicians are in charge; they work with numbers and not with underlying physical parameters.

i.e. they have the same absolute value $|c_{-n}| = |c_n| = |c_{N-n}|$. This in turn means that for such real $y(t)$ the absolute values of the spectrum are symmetric either with respect to 0 or to the $N/2$ coefficient. Consequently, **the highest frequency** of the spectrum is **the Nyquist frequency** (f_{Nq}) and not the $(N-1) \times f_1$, which is $\approx f_s$ for the large N.

1.3 MATLAB's DFT implementation and Fast Fourier Transform (FFT)

If someone implements eq. (1.16) directly, it would take N basic operations to calculate each c_n, and thus N^2 operations to do the full transform. This is extremely computationally taxing. Luckily, there is an algorithm, aptly named the *Fast Fourier Transform* (FFT) which does it in $\mathcal{O}(N \log N)$ steps [1], drastically speeding up the calculation time.

Matlab has built-in FFT realizations

- `fft(y)` for the forward Fourier transform;
- `ifft(c)` for the inverse Fourier transform.

Unfortunately (as we discussed in section 1.2) , MATLAB does FFT the same way as it is defined in eq. (1.16), i.e. it does not normalize by N. So if you change number of points, the strength of the same spectral component will be different, which is unphysical. To get Fourier series coefficients (c_n) normalized, you need to calculate `fft(y)/N`. Nevertheless, the round trip normalization is maintained, i.e. `y = ifft( fft(y) )`.

There is one more thing, which arises from the fact that MATLAB indexes arrays starting from 1. The array of the forward Fourier transform coefficients `c=fft(y)` has the shifted by 1 correspondence to c_n coefficients, i.e. `c(n)`$= c_{n-1}$.

1.4 Compact mathematical notation for Fourier transforms

The forward Fourier transform is often denoted as $\mathcal{F}$ and the coefficients of the forward transformation as $Y = (Y_0, Y_1, Y_2, \ldots, Y_{N-1}) = (c_0, c_1, c_2, \ldots, c_{N-1})$. In this notation, we refer to Y_n coefficients, instead of c_n coefficients. So, the spectrum of the time domain signal $y(t_k)$ is

$$Y = \mathcal{F}y \tag{1.19}$$

The inverse Fourier transform is denoted as $\mathcal{F}^{-1}$:

$$y = \mathcal{F}^{-1}Y \tag{1.20}$$

1.5 DFT example

Let's consider a very simple example which will help us to put together the above material. We will sample and calculate the DFT for the following function

$$y(t) = D + A_{one}\cos(2\pi f_{one}t) + A_{two}\cos(2\pi f_{two}t + \pi/4) \tag{1.21}$$

where $D = -0.1$ is the displacement, offset, or bias of the function with respect to zero; $A_{one} = 0.7$ is the amplitude of the cosine with frequency $f_{one} = 10$ Hz; and $A_{two} = 0.2$ is the amplitude of the $\pi/4$ shifted cosine with frequency $f_{two} = 30$ Hz. For the reasons which we explain later in the chapter 16, we chose the sampling frequency $f_s = 4f_{two}$. We run the following code to prepare time samples y_k for the time data set governed by eq. (1.21) and to calculate the corresponding DFT components $Y_n =$`fft(y)`

Listing 1.1: `two_cos.m` (available at `http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/code/two_cos.m`)

```
%% time dependence governing parameters
Displacement=-0.1;
f_one=10; A_one=.7;
f_two=30; A_two=.2;
f= @(t) Displacement + A_one*cos(2*pi*f_one*t) + A_two*cos(2*pi*f_two*t+pi
   /4);

%% time parameters
t_start=0;
T  = 5/f_one; % should be longer than the slowest component period
t_end = t_start + T;

% sampling frequency should be more than twice faster than the fastest
   component
f_s = f_two*4;
dt = 1/f_s; % spacing between sample points times
N=T/dt; % total number of sample points

t=linspace(t_start+dt,t_end, N); % sampling times
y=f(t); % function values in the sampled time

%% DFT via the Fast Fourier Transform algorithm
Y=fft(y);
Y_normalized = Y/N; % number of samples independent normalization
```

Let's first plot the time domain samples $y_k = y(t_k)$ and the underlying eq. (1.21)

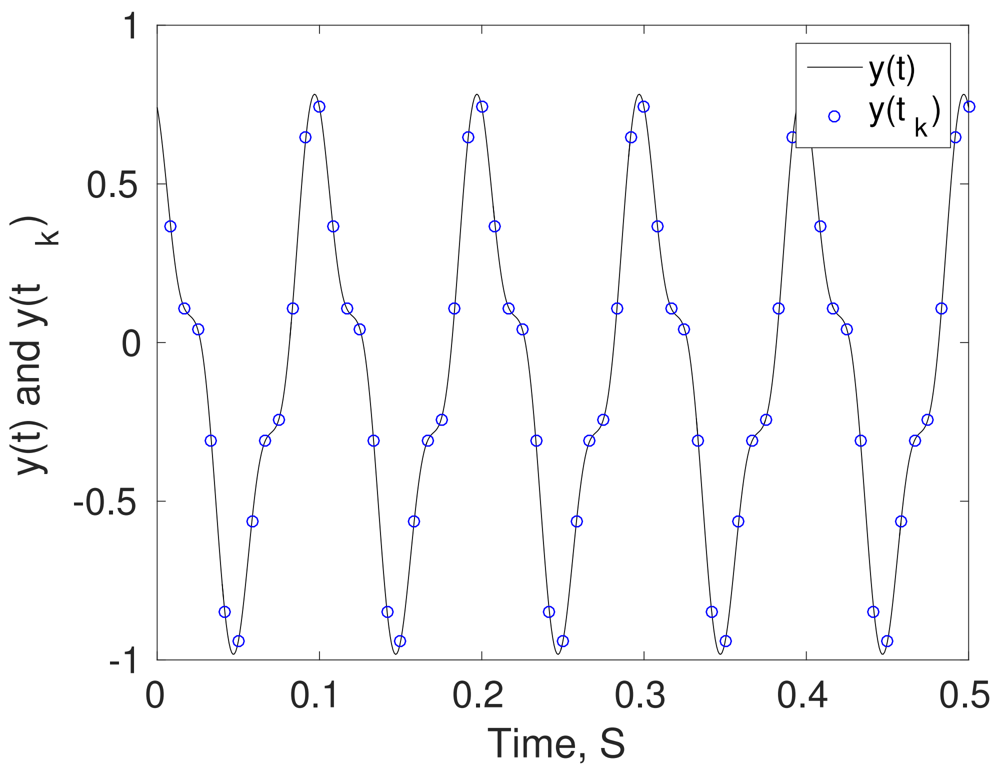

Figure 1.8: 60 time domain samples and the underlying eq. (1.21).

Listing 1.2: plot_two_cos_time_domain.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/code/plot_two_cos_time_domain.m)

```
two_cos;

%% this will be used to provide the guide for the user
t_envelope=linspace(t_start, t_end, 10*N);
y_envelope=f(t_envelope);

plot( t_envelope, y_envelope, 'k-', t, y, 'bo' );
fontSize=FontSizeSet; set(gca,'FontSize', fontSize );
xlabel('Time, S');
ylabel('y(t) and y(t_k)');
legend('y(t)', 'y(t_k)');
```

The result is shown in fig. 1.8. We can see 5 periods of the $y(t)$, though the function no longer resembles sine or cosine anymore due to combination of two cosines. Note that the $y(t)$ is shown as the guide to the eye only. The DFT algorithm has no access to it other than the 60 sampled points.

Let's now draw the $|Y_n|$ calculated by `fft(y)`

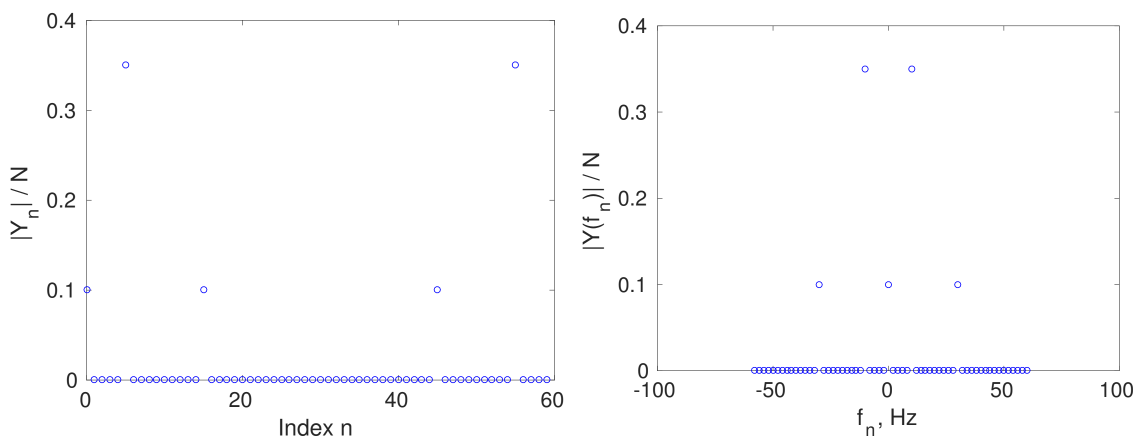

Figure 1.9: The normalized DFT coefficients for the time samples shown in fig. 1.8. The left panel depicts coefficient values versus their index (starting from 0). The right panel shows the spectrum, i.e. coefficient values versus their corresponding frequency.

Listing 1.3: plot_two_cos_fft_domain.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/code/plot_two_cos_fft_domain.m)

```
two_cos;

n=(1:N) - 1; % shifting n from MATLAB to the DFT notation

plot( n, abs(Y_normalized), 'bo' );
fontSize=FontSizeSet; set(gca,'FontSize', fontSize );
xlabel('Index n');
ylabel('|Y_n| / N');
```

The result is shown in the left panel of fig. 1.9. Note that we normalized the result of the DFT by number of points $N = 60$. This allows us to see the true nature of the Fourier transform coefficients: recall (see section 1.2) that the Y_0 coefficient normalized by N corresponds to the mean or displacement of the function, which is -0.1 as we set it above for eq. (1.21). We might wonder why there are 4 more non zero coefficients if we have only two cosines with two distinct frequencies. This is due to reflection property of the DFT, i.e. Y_{-n} and Y_{N-n} corresponds to the same frequency. This is better shown if we plot the spectrum, i.e. Y_n versus the corresponding frequency. This is done with

Listing 1.4: plot_two_cos_freq_domain.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/code/plot_two_cos_freq_domain.m)

```
two_cos;

freq = fourier_frequencies(f_s, N); % Y(i) has frequency freq(i)

```

```
plot( freq, abs(Y_normalized), 'bo' );
fontSize=FontSizeSet; set(gca,'FontSize', fontSize );
xlabel('f_n, Hz');
ylabel('|Y(f_n)| / N');
```

the Y_n index transformation to the frequency is done with the helper function

Listing 1.5: fourier_frequencies.m (available at http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/code/fourier_frequencies.m)

```
function spectrum_freq=fourier_frequencies(SampleRate, N)
        %% return column vector of positive and negative frequencies for DFT
        % SampleRate - acquisition rate in Hz
        % N - number of data points

        f1=SampleRate/N; % spacing or RBW frequency

        % assignment of frequency,
        % recall that spectrum_freq(1) is zero frequency, i.e. DC component
        spectrum_freq=(((1:N)-1)*f1).';  % column vector

        NyquistFreq= (N/2)*f1; % index of Nyquist frequency i.e. reflection
           point

        %let's take reflection into account
        spectrum_freq(spectrum_freq>NyquistFreq) =-N*f1+spectrum_freq(
           spectrum_freq>NyquistFreq);
end
```

Now we have the canonical spectrum shown in the right panel of fig. 1.9. Notice that the spectrum of the absolute values of Y_n is fully symmetric (i.e. mirrored around $f = 0$, as it predicted by eq. (1.18). Now, we see that the spectrum has only two strong frequency components at 10 Hz and 30 Hz. This is in complete accordance with eq. (1.21) and the values of $f_{one} = 10$ Hz and $f_{two} = 30$ Hz. Now, let's examine the components' values: $Y(10)$ Hz $= 0.35$, which is exactly half of the $A_{one} = 0.70$. A similar story is seen for the other frequency component $Y(30 \text{ Hz}) = 0.1$. This is due to eqs. (1.10) and (1.11) and the fact that $y(t)$ has no imaginary part. Note that Y_n themselves can be complex even in this case.

1.6 Self-Study

Problem 1: Have a look at the particular realization of the N point forward DFT with the omitted normalization coefficient:

$$C_n = \sum_{k=1}^{N} y_k \exp(-i2\pi(k-1)n/N)$$

Analytically prove that the forward discrete Fourier transform is periodic, i.e. $c_{n+N} = c_n$. Note: recall that $\exp(\pm i2\pi) = 1$.

Does this also prove that $c_{-n} = c_{N-n}$?

Problem 2: Use the proof for the previous problem's relationships and show that the following relationship holds for any sample set which has only real values (i.e. no complex part)

$$c_n = c^*_{N-n}$$

Where * depicts the complex conjugation.

Problem 3: Load the data from the file `'data_for_dft.dat'`[4] provided at the book's web page. It contains a table with y vs t data points (the first column holds the time, the second holds y). These data points are taken with the same sampling rate.

(a) What is the sampling rate?

(b) Calculate forward DFT of the data (use MATLAB built-ins) and find which 2 frequency components of the spectrum (measured in Hz not rad^{-1}) are the largest. Note: I refer to the real frequency of the sin or cos component, i.e. only positive frequencies.

(c) What is the largest frequency (in Hz) in this data set which we can scientifically discuss?

(d) What is the lowest frequency (in Hz) in this data set which we can scientifically discuss?

[4]The file is available at http://physics.wm.edu/programming_with_MATLAB_book/ch_dft/data/data_for_dft.dat

Bibliography

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 3 edition, 2007.