Chapter 5

Fourier Transforms

Outline of Section

- Fourier Transforms—recap
- Discrete FTs (DFTs)
- Sampling & Aliasing
- Fast Fourier Transform (FFT) algorithm
- Pseudocode

In the previous chapter, we looked at how to take a set of data points, and produce a function that returns a value for any point in the range of these data points, interpolating between them. We now turn towards another way in which such a set of data points can be processed, which is to perform a Fourier Transform on them, to extract the frequency-based properties of the data set.

We focus on how to numerically calculate the Fourier transform of *regular,* discrete, samples of a function. The **Discrete Fourier Transform** (DFT) is used for this, and is the discrete equivalent of the Fourier transform. The sampled function could be the function found by solving an ODE or PDE using finite difference methods. Or it might be data sampled from, e.g., an oscilloscope or a microphone. Considerations and limitations caused by the sampling procedure will be discussed.

The **Fast Fourier Transform** (FFT)—an efficient implementation on the DFT—is used extensively, e.g., for

- Noise suppression—Data analysis
- Image compression—JPEG formats
- Audio and video compression—MPEG formats
- Medical imaging
- Interferometric imaging
- Modelling of optical systems
- Solution of periodic boundary value problems

to name a few examples.

"Pseudocode" is a concept that is very useful and is widely-used in discussions of computer algorithms. While this has no direct connection with Fourier Transforms, we make the use of the opportunity to describe the Fast Fourier Transform algorithm in this chapter to introduce the concept of writing and using pseudocode.

5.1 Fourier Transforms—Recap

The continuous Fourier Transform (FT), both forward and backward, of a function $f(t)$ are defined as

$$\tilde{f}(\omega) = \int_{-\infty}^{+\infty} e^{i\omega t} f(t)\,\mathrm{d}t = \mathcal{F}(f(t)), \tag{5.1}$$

$$\text{and} \qquad f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-i\omega t}\tilde{f}(\omega)\,\mathrm{d}\omega = \mathcal{F}^{-1}(\tilde{f}(\omega)), \tag{5.2}$$

for the case of time t and angular frequency $\omega = 2\pi\nu$ (where ν is frequency). Time t and ω are reciprocal 'coordinates' or variables. $\tilde{f}(\omega)$ is the (angular) frequency spectrum of the function $f(t)$. The FT is an expansion on plane waves $\exp(-i\omega t)$ which constitute an orthonormal basis, and $\tilde{f}(\omega)$ can be thought of as the "coefficients" of each component wave of angular frequency ω. (Notation note: do not confuse this with the tilde '$\sim$' notation mentioned elsewhere in the course, such as to denote numerical approximation or for scaled variables!) In general $\tilde{f} \in \mathbb{C}$, even for a real function $f(t)$, with $|\tilde{f}(\omega)|$ being the amplitude and $\arg(\tilde{f}(\omega))$ the phase of the component wave $\exp(-i\omega t)$. For $f(t) \in \mathbb{R}$ the **reality condition** applies in reciprocal space (also referred to as 'Fourier space')

$$\tilde{f}(-\omega) = \tilde{f}^*(\omega), \tag{5.3}$$

so that the negative frequencies are degenerate; there is no extra information in them. However, for an arbitrary $f(t) \in \mathbb{C}$, $\tilde{f}(-\omega)$ is unrelated to $\tilde{f}^*(\omega)$.

The $f(t)$ and $\tilde{f}(\omega)$ are different representations of the same thing in the time and frequency domain, respectively, and the relationship between such FT pairs is often written as

$$f(t) \rightleftharpoons \tilde{f}(\omega). \tag{5.4}$$

$\mathcal{F}^{-1}(\tilde{f}(\omega))$ is often called the inverse Fourier transform. Note that there are different conventions for defining the FT operations, e.g., which operation gets the $1/2\pi$ factor or whether both share it (i.e. $1/\sqrt{2\pi}$ in front of each), and the sign in the transform kernel (i.e. $\exp(-i\omega t)$ or $\exp(i\omega t)$) , etc.

For spatial FTs in one dimension (e.g. x) the reciprocal variables become $t \to x$ and $\omega \to k$ where $k = 2\pi/\lambda$ is the wavenumber and λ is wavelength. In multiple spatial dimensions the wavenumber becomes a wave vector

$$\vec{x} \equiv (x, y, z) \qquad \leftrightarrow \qquad \vec{k} \equiv (k_x, k_y, k_z)\,, \tag{5.5}$$

and the FTs are modified accordingly:

$$\tilde{f}(\vec{k}) = \int_{-\infty}^{+\infty} e^{i\vec{k}\cdot\vec{x}} f(\vec{x})d\vec{x} = \mathcal{F}(f(\vec{x})), \tag{5.6}$$

$$f(\vec{x}) = \frac{1}{(2\pi)^3}\int_{-\infty}^{+\infty} e^{-i\vec{k}\cdot\vec{x}}\tilde{f}(\vec{k})d\vec{k} = \mathcal{F}^{-1}(\tilde{f}(\vec{k})). \tag{5.7}$$

Derivatives and FTs

FTs can be very useful in manipulating differential equations. This is because spatial or time derivatives become algebraic operations in Fourier space. As an example consider the following equation in real space

$$\frac{\mathrm{d}}{\mathrm{d}x}u(x) = v(x).$$

Taking the FT of this equation yields

$$-ik\tilde{u}(k) = \tilde{v}(k). \tag{5.8}$$

This can be shown as follows: replace $u(x)$ and $v(x)$ by their inverse FTs

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-ikx}\tilde{u}(k)\,\mathrm{d}k\right) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-ikx}\tilde{v}(k)\,\mathrm{d}k\,. \tag{5.9}$$

The only bits containing x are the transform kernel (i.e. the plane wave part $\exp(-ikx)$). Therefore using Leibnitz's integral rule yields

$$\frac{1}{2\pi}\int_{-\infty}^{+\infty} -ike^{-ikx}\tilde{u}(k)\,\mathrm{d}k = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-ikx}\tilde{v}(k)\,\mathrm{d}k\,.$$

Equating the integrands on both sides we have (5.8).

This can be generalised to higher derivatives:

$$\frac{d^n}{dx^n}f(x) \rightleftharpoons (-ik)^n\tilde{f}(k). \tag{5.10}$$

It can be generalised to 3D too:

$$\begin{aligned} \nabla f(\vec{x}) &\rightleftharpoons -i\vec{k}\tilde{f}(\vec{k}) \quad , & \nabla^2 f(\vec{x}) &\rightleftharpoons -|\vec{k}|^2\tilde{f}(\vec{k}) \quad , \\ \nabla\cdot\vec{E}(\vec{x}) &\rightleftharpoons -i\vec{k}\cdot\tilde{\vec{E}}(\vec{k}) \quad , & \nabla\times\vec{E}(\vec{x}) &\rightleftharpoons -i\vec{k}\times\tilde{\vec{E}}(\vec{k}), \end{aligned} \tag{5.11}$$

where $\nabla^2 f = \nabla\cdot(\nabla f)$ has been used.

5.2 Discrete FTs

DFTs are the equivalent of complex Fourier series, which are defined on a finite length domain, but for a **discretely sampled function** rather than a continuous function. We illustrate here with time and angular frequency.

Time domain—We assume the function is sampled by N equally spaced samples on a time domain of length $T = N\Delta t$ as illustrated in figure 5.1 ;

$$f_n \equiv f(t_n) \qquad \text{with} \qquad n = 0, 1, 2, ..., N-1 \qquad \text{and} \qquad t_n = n\Delta t. \tag{5.12}$$

The function $f(t)$ is ***assumed to be periodically extended*** beyond the domain $0 \le t \le T$ so that $f(t+mT) = f(t)$ for integer m. For the sampled function, periodicity means $f_N = f_0$. This does not mean that the value of the function at $t \to T$ has to be the same as that at $t = 0$; but it needs to be single-valued, and for discretising it one would choose the value for $t = 0$.

If one is only focused on the time domain $0 \le t < T$, the periodicity of the function does not affect things directly.

Frequency domain— Figure 5.2 illustrates the frequency domain and discrete spectrum of the signal sampled in figure 5.1. The longest wave that can fit exactly into the time domain has an angular frequency $\omega_{min} = 2\pi/T \equiv \Delta\omega$. The shortest wave that can be *recognised* by the grid and fits periodically into the time domain has duration $2\Delta t$ (i.e. one peak and one trough in just two time samples) so that $\omega_{max} = 2\pi/(2\Delta t) = \pi/\Delta t$. This maximum frequency is known as the **Nyquist frequency**

$$\boxed{\omega_{max} = \frac{\pi}{\Delta t} = \Delta\omega\frac{N}{2}.} \tag{5.13}$$

Allowing for negative angular frequencies too, these frequencies define the discrete, finite, angular frequency grid on which $\tilde{f}$ is sampled;

$$\boxed{\tilde{f}_p \approx \frac{1}{\Delta t}\tilde{f}(\omega_p) \quad \text{with} \quad p = -\frac{N}{2}, \ldots, 0, \ldots, \frac{N}{2} \quad \text{and} \quad \omega_p = p\Delta\omega = p\frac{2\pi}{N\Delta t}.} \tag{5.14}$$

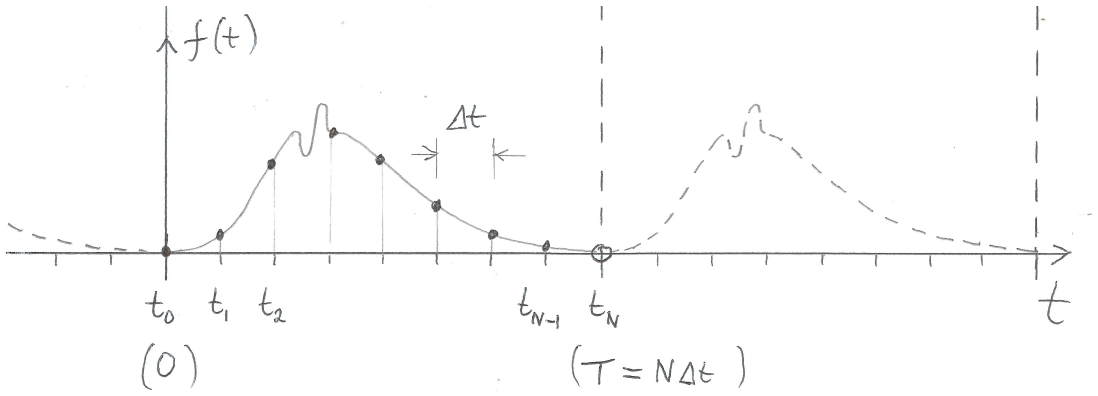

Figure 5.1: Illustration of a signal in the time domain. $N = 8$ here. The signal/function is assumed to be periodically extended beyond $0 \le t \le T$.

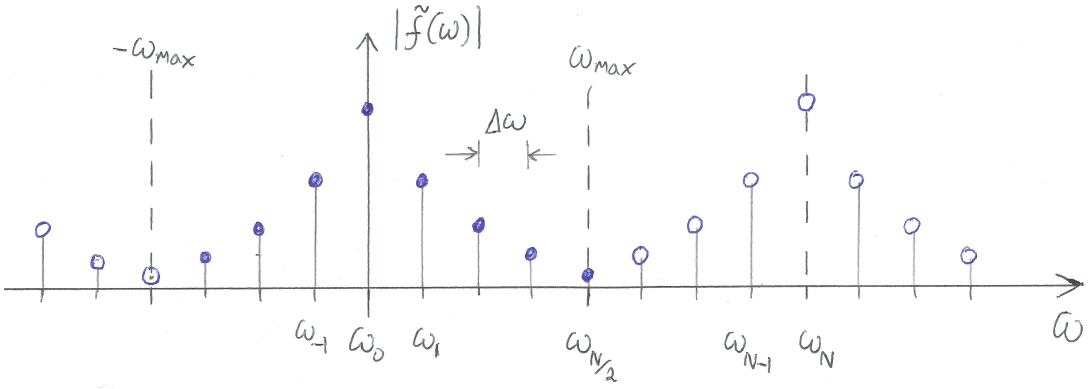

Figure 5.2: Frequency domain and discrete spectrum corresponding to figure 5.1. The discrete spectrum is also periodically extended (beyond $|\omega_{max}|$ in this case). The modulus of $\tilde{f}(\omega)$ is depicted.

Each discrete frequency corresponds to a wave that fits exactly an integer number of times 'p' into the finite domain. Note that the integer index p seems to run over $N + 1$ values but in fact the $-N/2$ and $N/2$ samples are degenerate, i.e.,

$$\tilde{f}_{-N/2} \equiv \tilde{f}_{N/2}.$$

so there are only N degrees of freedom. (See section 5.3.)

Why does $\tilde{f}_p \approx \tilde{f}(\omega_p)/\Delta t$ rather than $\tilde{f}_p = \tilde{f}(\omega_p)/\Delta t$ in equation (5.14) above? This is because $\tilde{f}_p$ represents the DFT which is not always exactly $\tilde{f}(\omega)$, the true spectrum of $f(t)$, simply picked out at discrete frequencies ω_p. We assume that $f(t)$ *is* exactly, discretely sampled. If there is no aliasing (see 5.3) then $\tilde{f}_p = \tilde{f}(\omega_p)/\Delta t$. If there is aliasing, then the DFT is a distorted, discrete, version of the true spectrum. (The $1/\Delta t$ factor is just the convention used in the definition of the DFT, see below.)

The DFT is the analogue of a continuous FT, but with the continuous integral $\int \ldots \mathrm{d}t$ replaced by a finite sum $\sum \ldots \Delta t$:

$$\boxed{\tilde{f}_p = \sum_{n=0}^{N-1} f_n e^{i\omega_p t_n} = \sum_{n=0}^{N-1} f_n e^{i2\pi pn/N},} \tag{5.15}$$

where $\omega_p t_n = \left(\frac{2\pi p}{N\Delta t}\right)(n\Delta t)$ has been used to obtain the second form. The backwards transform is

similarly defined as

$$f_n = \frac{1}{N} \sum_{p=-N/2+1}^{N/2} \tilde{f}_p e^{-i2\pi pn/N}. \tag{5.16}$$

The different normalisation factor of $1/N$ accounts for the discrete sampling. This comes from $\mathrm{d}t \to \Delta t$ and $d\omega \to \Delta\omega = 2\pi/(N\Delta t)$. (Note that in going from the FT (5.1) to the DFT (5.15) a factor of Δt has been absorbed into $\tilde{f}_p$, i.e., $\tilde{f}_p \approx \tilde{f}(\omega_p)/\Delta t$.) The backward DFT (5.16) is usually defined as

$$\boxed{f_n = \frac{1}{N} \sum_{p=0}^{N-1} \tilde{f}_p e^{-i2\pi pn/N},} \tag{5.17}$$

which is more symmetrical with the forward DFT. Why this is possible will be seen in section 5.3.

Relation to complex Fourier series – Discrete FTs (DFTs) are intimately related to complex Fourier series (CFS).

$$c_k = \frac{1}{T} \int_0^T f(t) e^{i\omega_k t} \,\mathrm{d}t\,, \tag{5.18}$$

$$f(t) = \sum_{k=-\infty}^{+\infty} c_k e^{-i\omega_k t}. \tag{5.19}$$

Complex Fourier series represent the discrete spectrum of a ***continuous function*** $f(t)$, also on a domain of finite length. The allowed angular frequencies have the same spacing $\Delta\omega$, but are now infinitely many.

FFTs—Fast Fourier transforms

The DFTs above, equations (5.15) and (5.16), are easily implemented on a computer, however the naive method of implementing it requires $\mathcal{O}(N^2)$ operations. With typical samples volumes of $N \sim 10^6$ in modern applications this becomes prohibitive even on the fastest computers. The DFT can actually be done in $\boldsymbol{\mathcal{O}(N \log_2 N)}$ **operations**; the algorithm for this is known as the **fast Fourier transform** or just **FFT**. For $N = 10^6$, the speed-up factor $N/log_2 N$ is approximately 50,000. FFTs work best when $N = 2^m$ with integer m, and works by recursively splitting the DFT into separate sums over only the even or odd indices. ***The FFT implementation of the DFT is what is used in practice.*** If $N \neq 2^m$, then it is usual to ***pad out*** the samples with zeros until the size is $N = 2^m$.

DFT in 2D

For a function $f_{n,m} = f(x_n, y_m)$ defined on a 2D grid $x_n = m\Delta x$ for $0 \leq n \leq N-1$ and $y_m = m\Delta y$ for $0 \leq n \leq M-1$ its DFT is given by

$$\tilde{f}_{p,q} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f_{n,m} e^{i2\pi pn/N} e^{i2\pi qm/M}. \tag{5.20}$$

The backward DFT is

$$f_{n,m} = \frac{1}{NM} \sum_{p=0}^{N-1} \sum_{q=0}^{M-1} \tilde{f}_{p,q} e^{-i2\pi pn/N} e^{-i2\pi qm/M}. \tag{5.21}$$

Going to 3-dimensions just adds another nested sum (with a new, independent index) and another exponential wave factor (incorporating the new dimension) into the transform kernel.

5.3 Sampling & Aliasing

The discrete sampling of the function to be FT'd, given by equation (5.12), introduces some sampling effects and care has to be taken in allowing for them. There is a minimum sampling rate that needs to be used so that we do not lose information. If the function to be sampled fits into the finite length time (or spatial) domain of length T, and is **bandwidth-limited** to below the Nyquist frequency then all frequency content of the function is exactly captured by the discrete sampling process. If the function has $\tilde{f}(\omega) \neq 0$ for $|\omega| > \omega_{max}$ then the frequency content for $|\omega| > \omega_{max}$ will be **aliased** down into the range $|\omega| < \omega_{max}$ and distort the DFT compared to the exact FT of $f(t)$. If the original function is bandwidth-limited but is longer than T, then it will be ***clipped*** which will introduce a sharp cutoff at $t = 0$ and/or T. This will effectively increase the bandwidth of the clipped function and aliasing will occur again during sampling of it.

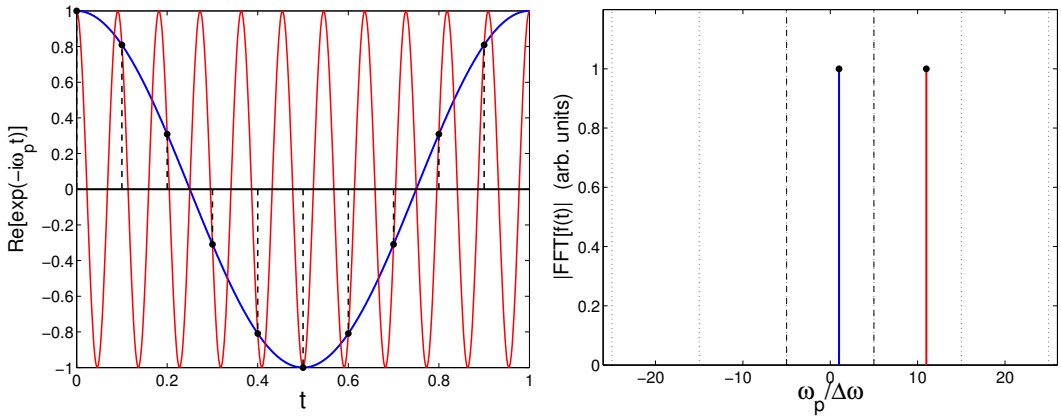

Figure 5.3: (Left) Indistinguishable harmonic waves below and above the Nyquist frequency, for $N = 10$. Blue line: $\omega = \omega_1 = \Delta\omega$. Red line: $\omega = \omega_{N+1} = \Delta\omega + 2\omega_{max}$. (Waves are periodically extended beyond range shown.) (Right) Corresponding (angular) frequency spectrum. The vertical dashed lines lie at $\pm$ the Nyquist frequency.

Aliasing & indistinguishable frequencies – For any wave with a frequency ω_p lying in the frequency domain captured by the DFT, there are higher frequency waves with $\omega_{p'} = \omega_p + m\Omega$ (where $m \in \mathbb{Z}$ and $\Omega = 2\omega_{max}$ is the width of the frequency domain) that look exactly the same to the discrete time-sampling grid. This is illustrated in figure 5.3. This can be understood by considering the kernel of the DFT, i.e., $\exp\left(i\omega_{p'}t_n\right)$.

$$\begin{aligned}\exp\left[i(\omega_p + m\Omega)t_n\right] &= \exp\left[i\left(\frac{2\pi pn}{N} + m\frac{2\pi}{\Delta t}n\Delta t\right)\right] = \exp\left(i\frac{2\pi pn}{N} + i2\pi mn\right) \\ &= \exp\left(i\frac{2\pi pn}{N}\right) = \exp\left(i\omega_p t_n\right).\end{aligned}$$

This has the following implications;

(a) It explains why $\tilde{f}_{-N/2} \equiv \tilde{f}_{N/2}$.

(b) It also explains why equation (5.17) for the backwards DFT 'works'. The "negative" frequency part of the spectrum $p = \left\{-\frac{N}{2}+1, \ldots, -1\right\}$ maps to the positive spectrum at $p' = p + N = \left\{\frac{N}{2}+1, \frac{N}{2}+2, \ldots, N-1\right\}$ lying beyond the Nyquist frequency. In other words, although the $\sum_{p=N/2+1}^{N-1}$ parts of (5.17) are above the Nyquist frequency, they happen to capture the desired negative frequencies within the Nyquist range.

(c) Aliasing:

$$\tilde{f}_p = \sum_m \mathcal{F}(f)|_{\omega_p + m\Omega}, \tag{5.22}$$

i.e., the DFT (the LHS) folds in the Fourier components from the *exact FT of the continuous function* from the indistinguishable set of waves $\omega'_p = p + m\Omega$.

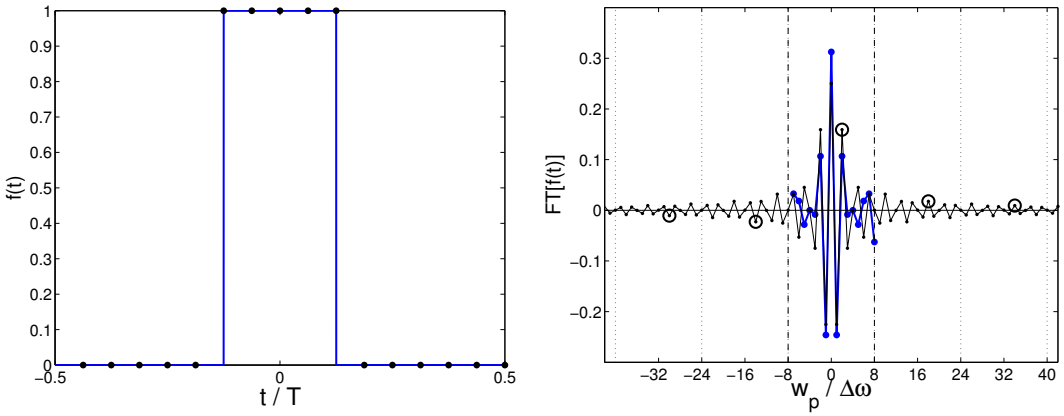

Figure 5.4: Aliasing: (Left) $f(t)$ in the time domain; centred "rect" function of width $T/4$. Sampled with $N = 16$. (Right) The DFT (blue, solid line + markers), and exact FT (black, thin solid line) in the frequency domain. The open circles denote Fourier components outside the range $-\omega_{max} \le \omega \le \omega_{max}$ that are aliased into that range during the DFT. The real part of the DFT & FT are shown. The DFT been divided by Δt.

Figure 5.4 shows the phenomenon of aliasing for a top-hat function

$$f(t) = \begin{cases} 1 & |t| < a/2 \\ 0 & |t| > a/2 \end{cases} .$$

5.4 Fast Fourier Transforms—FFTs

The FFT algorithm for doing a discrete Fourier transform in $\mathcal{O}(N \log N)$ operation was described by Daniel & Lanczos in 1942, and earlier versions existed in the 1800s (indeed, the first example of the algorithm was given by Gauss in 1805), but was rediscovered and popularised in the computer age by Cooley & Tukey in 1965.

Directly coding the DFT (as we did last time) results in an $\mathcal{O}(N^2)$ algorithm. But this can be reduced to $\mathcal{O}(N \log_2 N)$ with the Fast Fourier Transform, for $N = 2^m$. The central idea is that an N-sample DFT can be split into two $N/2$-sample ones:

$$\begin{aligned} \tilde{f}_p &= \sum_{n=0}^{N-1} f_n e^{i2\pi pn/N} \\ &= \sum_{n=0,2,\ldots}^{N-2} f_n e^{i2\pi pn/N} + \sum_{n=1,3,\ldots}^{N-1} f_n e^{i2\pi pn/N} \\ &= \tilde{f}_p^{\text{even}} + e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}} \end{aligned}$$

Each half of these is an $\mathcal{O}(\frac{N}{2} \log \frac{N}{2})$ problem, and this can be done recursively.

Here, $\tilde{f}_p^{\text{even, odd}}$ are $N/2$-sample DFTs of the even and odd samples of the original f_n. These are each periodic such that $\tilde{\boldsymbol{f}}_{\boldsymbol{p+N/2}}^{\text{even, odd}} = \tilde{\boldsymbol{f}}_{\boldsymbol{p}}^{\text{even, odd}}$ This results in, for $0 \le p < N/2$:

$$\begin{aligned} \tilde{f}_p &= \tilde{f}_p^{\text{even}} + e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}} \\ \tilde{f}_{p+N/2} &= \tilde{f}_p^{\text{even}} - e^{i2\pi p/N} \times \tilde{f}_p^{\text{odd}}. \end{aligned}$$

This leads to a simple algorithm for implementation, which we can discuss using pseudocode.

5.4.1 Pseudocode

We often need to explain an algorithm to each other, in a way that clearly shows how you would code it up (in any language). In this case, it helps not to be burdened by the (language-specific) overhead that real code brings with it (as well as a need to be syntactically perfect). This to say that it helps to omit constructs such as:

"`import numpy as np`"

"`tarray = np.zeros(N)`"

"`#include <stdio.h>`"

"`COMMON/CRZT/IXSTOR,IXDIV,IFENCE(2),LEV,LEVIN,BLVECT(LURCOR)`"

and the positioning of colons and semicolons etc., and allow ourselves to focus on the logic of the algorithm that is being presented. We call this "Pseudocode"; it is intended to be for humans, but has a code-like structure because of the logic of algorithms. While attempts have been made in the past to standardise pseudocode, these have failed because this runs counter to the benefits of using pseudocode; therefore there is no formal syntax; it is up to the author to make things clear for their purpose and their audience. One is allowed to use language-specific elements if that is not confusing (e.g., logic statements from Python or C etc.).

Examples of pseudocode of all sorts of styles are readily available online.

5.4.2 Fast Fourier Transforms in Pseudocode

The following is an example of the FFT algorithm, in pseudocode. The recursive nature of the algorithm is made clear through "calls" to the FFT() function itself:

```
def FFT(f): # f is an array
    N = size of array f
    if N == 1:
        return f[0] # for a sample size of 1, the FT is the value itself
    if N is not a power of 2, exit

    # recursive calls
    farray_even = FFT(even entries of f) # size N/2
    farray_odd  = FFT(odd  entries of f) # size N/2

    return array made up of farray_even[n]
                          + exp(i2pi n) * farray_odd[n] (for n = 0..N/2-1)
           and of farray_even[n]
                          - exp(i2pi n) * farray_odd[n] (for n = N/2..N)
```

Note the use of expressions such as a) `N = size of array f`, b) `if N is not a power of 2, exit` and c) `odd entries of f`. These are of course not syntactically correct in any programming language, but are easy to parse for a human being, without being confusing for people who do not speak a particular language. These examples could be written as a) `N = $#f + 1;`, b) `if(x&(x-1)) exit;` and c) `f[1::2]`, none of which can necessarily be said to be easy to understand if one is not aware of the programming language that is in use.

The conversion of the above pseudocode into real code is left for the problem sheets.

Recursive function calls, whilst being logically clear, can require substantial overheads when the code is executed. When I was young, coding in BASIC and FORTRAN, recursive function calls were not even part of the standards for the languages.

In addition to the above, further shortcuts exist which can speed up the code by sorting the elements of the calculations in a clever way.

Contracting the "even" and "odd" superscripts in Equation 5.23 to "e" and "o" yields:

$$\begin{aligned} \tilde{f}_p &= \tilde{f}_p^{\mathrm{e}} + e^{i2\pi p/N} \times \tilde{f}_p^{\mathrm{o}} \\ \tilde{f}_{p+N/2} &= \tilde{f}_p^{\mathrm{e}} - e^{i2\pi p/N} \times \tilde{f}_p^{\mathrm{o}}. \end{aligned}$$

Iterating once again, the quantity $\tilde{f}^{\mathrm{e}}$ can be written, for $p = 0, \ldots, N/4 - 1$:

$$\begin{aligned} \tilde{f}_p^{\mathrm{e}} &= \tilde{f}_p^{\mathrm{ee}} + e^{i2\pi p/(N/2)} \times \tilde{f}_p^{\mathrm{eo}} \\ \tilde{f}_{p+N/4}^{\mathrm{e}} &= \tilde{f}_p^{\mathrm{ee}} - e^{i2\pi p/(N/2)} \times \tilde{f}_p^{\mathrm{eo}}. \end{aligned}$$

Assuming $N = 2^m$ (i.e. even), after m even/odd splitting we are left with N functions of period 1 which are each trivially Fourier Transformed; the transform of each is the same as itself

$$\tilde{f}_p^{eooeeoeoeo...e} = f_n, \tag{5.23}$$

for some n.

All we have left to do is identify which combination of *eoeeo...e* corresponds to each n—this can be done by assigning the binary value $e \equiv 0$ and $o \equiv 1$ to each element in the sequence and then **reversing** the sequence. The series of 1s and 0s obtained is a binary representation of n.

This can be seen empirically in the following, where an input function represented by a series of 2^3 samples is broken up in to two even and odd series, continuing recursively into single-sample elements:

$$\begin{array}{cccccccc} ||f_0, & f_1, & f_2, & f_3, & f_4, & f_5, & f_6, & f_7|| \\ ||f_0, & f_2, & f_4, & f_6|| & ||f_1, & f_3, & f_5, & f_7|| \\ ||f_0 & f_4|| & ||f_2 & f_6|| & ||f_1 & f_5|| & ||f_3 & f_7|| \\ ||f_0|| & ||f_4|| & ||f_2|| & ||f_6|| & ||f_1|| & ||f_5|| & ||f_3|| & ||f_7|| \end{array}$$

This results in the sequence of indices: 0, 4, 2, 6, 1, 5, 3, 7. This sequence can be compared with the original sequence, in binary form:

Original		Transformed	
Decimal	Binary	Decimal	Binary
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

Here, the binary representation is made of of bits that are reversed between the original and transformed order of indices. The "bit-reversing" procedure may not be particularly fast in software, but it is easy to implement directly in signal-processing chips etc.

The complete method requires

$$N \times p = \frac{N \log N}{\log 2} \sim \mathcal{O}(N \log N), \tag{5.24}$$

operations. As an example consider a problem with $N \sim 10^4$ samples, in this case the FFT is faster than the naive DFT by a factor of about 750.

The FFT algorithm leads naturally to the Fast Convolution of functions, Filtering, and finding the Correlation between functions etc.