An FFT Primer for physicists

Thomas Kaiser

Institue of Applied Physics, Abbe School of Photonics
Friedrich-Schiller-Universität Jena, Germany

V 1.0
– DRAFT –
29 June 2014

Abstract

This document is a draft. If you find typos (there are probably a lot, I type drafts fast without error correction) or have remarks, please contact: thomas.kaiser.1@uni-jena.de

The aim of this document is to clarify many beginners' uncertainties that occur when using FFT algorithms to obtain physical solutions to the Fourier transform using MATLAB. The "correct" use of `fft`, `ifft`, `fftshift`, and `ifftshift`, finding the correct frequency vector as well as issues regarding even/odd data length, sampling/aliasing and truncation/padding are discussed.

Contents

1 Introduction 2

2 Two simple examples, a naive attempt and its failure 2

3 Why FFT does not perform a Fourier Transform 4
3.1 Fourier series expansion vs. Fourier Transform 5
3.2 Sampled data considerations 6
3.3 Nyquist-Shannon sampling theorem, aliasing 6

4 Frequency bandwidth, negative frequencies 7
4.1 Real vs. complex Fourier series 8

5 Even vs. odd data length, truncation and padding 9

6 "Physical Use" of the FFT 12

A Fourier transform of a periodic function with finite repetitions 14

1 Introduction

Fourier transforms are a concept regularly appearing to the researcher doing natural sciences. In the processing of sampled data, they play an important role as well as in many numerical techniques.

2 Two simple examples, a naive attempt and its failure

Assume your task is to find the Fraunhofer diffraction pattern of a slit. From your basic optics course you know that this corresponds to finding the Fourier transform of the aperture. We want to solve this problem using MATLAB. First of all, we define the slit function. In our example, we use a computational window of 10 µm, sampled with 1000 points and a centered slit with 2 µm width.

```
x = linspace(-5, 5, 1000);   % create the x vector
f = zeros(size(x));          % create a vector of same size full of zeros
f(abs(x)<1) = 1;             % set f to one in the interval [-1 1]
```

The result is plotted in Fig. 1(a) and shows a nice rectangular function. We want to compare the results of the computation we are going to implement with the analytic result. Thankfully, this is an easy task and the analytic Fourier transform of a unit rectangle of width a reads as

$$F_{\text{analyt}}(k) = \frac{1}{2\pi}\int_{-a/2}^{a/2} \exp(-ikx)\,\mathrm{d}x = \frac{1}{-i\pi k}\sinh\left(-ik\frac{a}{2}\right) = \frac{a}{2\pi}\operatorname{sinc}\left(k\frac{a}{2}\right). \quad (1)$$

One may naively think that finding the Fourier Transform corresponds to simply executing `fft`. The result of this attempt is shown in Fig. 1(b). The expected sinc function is not reproduced. A glimpse into the `fft` manual explains that this is due to the specific order in which the coefficients come out of the `fft`. It starts with the zero frequency component, then positive ones and the negative ones in reverse order. One can read that the 'natural' order is restored by `fftshift`. Fig. 1(c) shows the result. In principle, a sinc-like behavior can be seen, but the result is not what one would expect. From the analytic solution we see that the maximum value we expect to appear is $1/\pi \approx 0.3$. The numeric solution, however, shows much higher values. Especially the zoomed central region in Fig. 1(d) shows another issue. While the analytic solution is purely real, the imaginary part of the analytic solution shows oscillatory non-zero values in the order of 10^0.

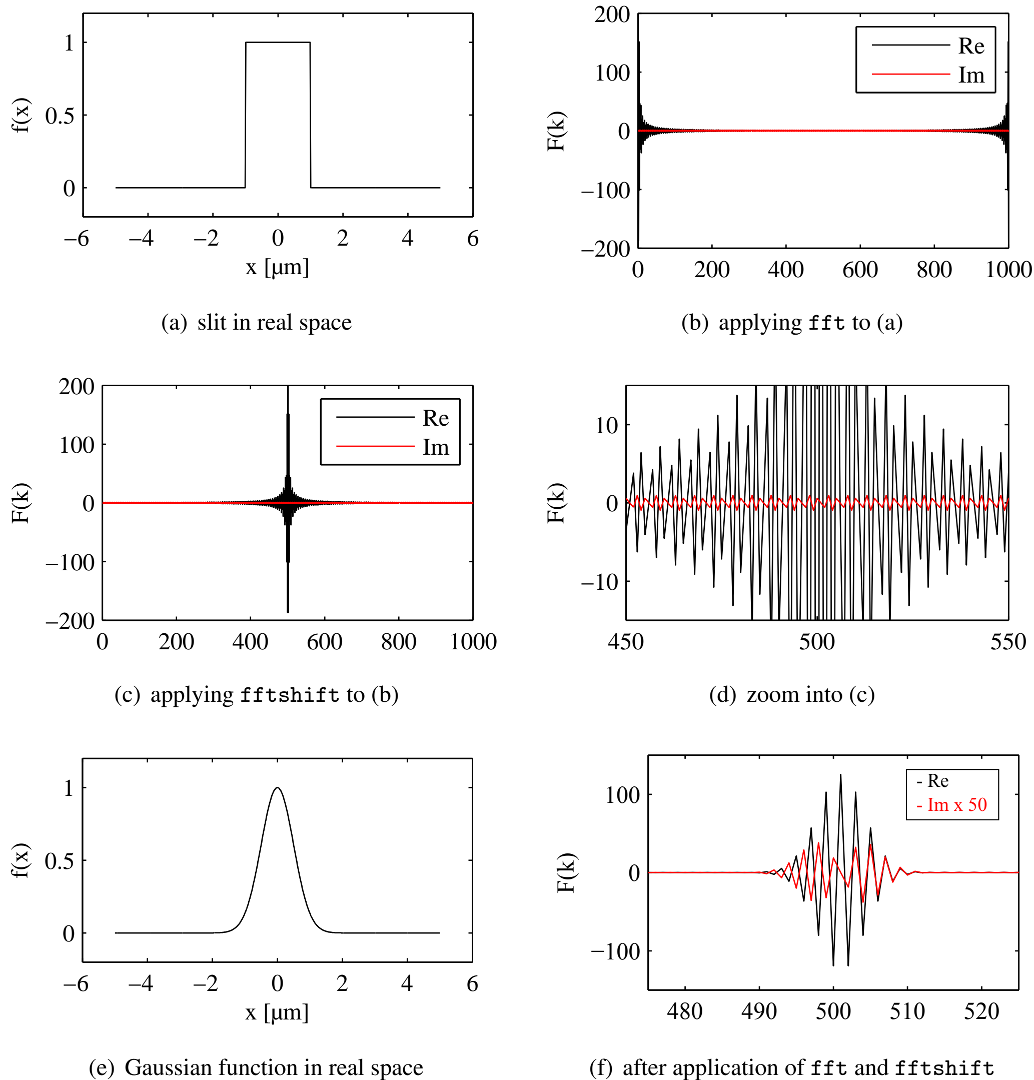
The situation becomes even worse when we use a Gaussian function as input instead of the rectangle. The Fourier transform of a (real) Gaussian is known to be a real Gaussian. So here is what it looks like. We put now

```
f = exp(-x.^2/0.5);
```

and get the function shown in Fig. 1(e) – a nice, zero-centered Gaussian. Look at the disastrous output after we applied `fft` and `fftshift` in Fig. 1(f). The output oscillates, is complex and if you have a close look not even centered at the point with index 500, which is the middle of the Fourier space (or not since there is an even number of points?). So what is going wrong here?

One may feel appealed to attribute this behavior to normalization issues, numerical errors, sampling inaccuracies and so on. But I will show that this is not true. It is about understanding what `fft` does – and what not. One may also be semi-successful by just considering the *absolute* value of F (which looks sinc-like and Gaussian, respectively, indeed). But doesn't it feel bad not to know why the absolute value seems to fit (except for scaling) while real and imaginary part by themselves are 'wrong'? What if phases play a role (which is not uncommon for optics) and real *and* imaginary part are important?

Also several other questions arise with this example. For a Fourier transform, one performs an integration over x, so the x values need to be explicitly known. How can `fft` then perform this transform? It does neither require the input of x, nor is it even possible to pass it as an argument. How are the sampling values for k found to assign an k-axis to the spectral plots in Fig. 1(b)-(d)? How to 'normalize'



(a) slit in real space

(b) applying `fft` to (a)

(c) applying `fftshift` to (b)

(d) zoom into (c)

(e) Gaussian function in real space

(f) after application of `fft` and `fftshift`

Figure 1: Naive use of `fft` to evaluate the Fourier transform of a slit. (a) A slit function in real space. (b) The direct use gives a result where the first argument is the zero frequency and the second half of the computed spectrum corresponds to negative frequencies. (c) The negative frequency components can be shifted to the left side of the spectrum using `fftshift`. (d) However, when zooming into the result it becomes obvious that the result has to be somehow 'wrong', for instance because of the non-zero imaginary components. (e) The use of a Gaussian example shows the problems even more prominently. (d) Its 'computed spectrum' is not Gaussian and not purely real as it needs to be.

the result properly so that it gives back the analytically expected values? Why do the (wrong) imaginary components occur and how to overcome this?

To answer all these questions, we have to work a little deeper into the math behind it. However since this is not supposed to be a math paper, I will not go into details unless necessary and try to keep the math at a practical level, so that everyone struggling with uncertainties using `fft` will be able to apply it with confidence to his problems after reading this document.

3 Why FFT does not perform a Fourier Transform

The first important point is to know what `fft`, `ifft`, `fftshift` and `ifftshift` actually *do*. This is pretty easy to find out. `fft` and `ifft` simply perform the following summations

$$\texttt{fft:} \quad F_m = \sum_{n=0}^{N-1} f_n \exp\left(-i\frac{2\pi}{N}mn\right), \qquad m = 0\ldots N-1 \tag{2a}$$

$$\texttt{ifft:} \quad f_n = \frac{1}{M}\sum_{m=0}^{M-1} F_m \exp\left(-i\frac{2\pi}{M}mn\right), \qquad n = 0\ldots M-1 \ . \tag{2b}$$

Not more, not less. The clue about the `fft` algorightm is that it can perform these sums with incredible *efficiency*, but this shall not be our concern here. I have intentionally changed the span of the indices m and n compared to the original definition given in the online help. The reason is the way MATLAB indexes arrays. It starts with 1 and ends at $N = \texttt{length(f)}$, in contrast to other languages. While such a convention seems understandable to some extend from a numerical logic point of view, this leads to problems with physics where the origin usually plays an important role. The reason why I have chosen to introduce apparently more complexity than necessary is that this notation is much more consistent with the math, which is more important to understand here. It is simply intuitive that F_0 corresponding to $m = 0$ contains the zero-order Fourier component. This comes at the expense that in your implemented program, you will of course find this component at a *different* position, but since the output of the FFT will usually undergo some shifting, we will have to make up our mind where it is anyway. For the examples discussed in Fig. 1 for instance, we will see that F_0 will be found at index 501 in the vector, so I see no point in using "MATLAB-notation" here.

Concerning the shifting. The effect of `fftshift` and `ifftshift` on a vector is fairly easy. They just exchange the left and right half. So how can their use be struggling though? The reason lies in different array *lengths*. Let us first consider a sample array with 4 components, so an even number of numerical entries

```
a = [0 1 2 3]
```

than the effect of the two functions will be

```
fftshift(a)  = [2 3 0 1]
ifftshift(a) = [2 3 0 1]
```

so there is no difference between them. However, when the array has an *odd* number of elements, see what happens

```
a            = [0 1 2 3 4]
fftshift(a)  = [3 4 0 1 2]
ifftshift(a) = [2 3 4 0 1]
```

To understand this behavior is important to understand where the "middle" the Fourier transform goes. We will come back to this issue when we have understood what the coefficients coming out of the FFT mean.

3.1 Fourier series expansion vs. Fourier Transform

It is now time to define the thing we are interested in, namely the (inverse) Fourier transform

$$F(k) = \mathscr{F}\{f(x)\} \quad = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)\, e^{-ikx}\,\mathrm{d}x \tag{3a}$$

$$f(x) = \mathscr{F}^{-1}\{F(k)\} = \int_{-\infty}^{\infty} F(k)\, e^{ikx}\,\mathrm{d}x \ . \tag{3b}$$

First we want to consider a point that will be helpful in understanding what follows. What is the Fourier transform of a periodic function? Let therefor $f(x)$ be composed of several shifted repetitions of a function $\tilde{f}(x)$, which is just defined within the interval $[0,L]$

$$f(x) = \cdots + \tilde{f}(x+L) + \tilde{f}(x) + \tilde{f}(x-L) + \tilde{f}(x-2L) + \cdots = \sum_{n=-\infty}^{\infty} \tilde{f}(x-nL). \tag{4}$$

To compute the Fourier transform, we make use of the **Fourier theorem** to evaluate it. It states that every periodic function can be expanded into a **Fourier series**

$$f(x) = \sum_{m=-\infty}^{\infty} c_m \exp\left(imx\frac{2\pi}{L}\right), \qquad m \in \mathbb{Z}, \tag{5}$$

with expansion coefficients

$$c_m = \frac{1}{L}\int_0^L f(x)\exp\left(-imx\frac{2\pi}{L}\right)\mathrm{d}x \tag{6a}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{f}(x')\exp(-imx')\,\mathrm{d}x'$$

$$= \tilde{F}\left(k = m\frac{2\pi}{L}\right). \tag{6b}$$

The Fourier transform of the remaining exponentials in (5) are a series of Dirac δ-peaks

$$\mathscr{F}\left\{\exp\left(imx\frac{2\pi}{L}\right)\right\} = \delta\left(k - m\frac{2\pi}{L}\right), \tag{7}$$

so we find

$$F(k) = \tilde{F}(k)\sum_{m=-\infty}^{\infty} \delta\left(k - m\frac{2\pi}{L}\right) \ . \tag{8}$$

Let us take a second to interpret this result. A periodic function $f(x)$ can be seen as an endless repetition of a 'kernel' function $\tilde{f}(x)$ which is identiacal to $f(x)$ in the core interval, and zero otherwise. Although the Fourier transform of $\tilde{f}(x)$ is *continous*, its Fourier spectrum consists of *discrete peaks* at spatial frequencies $2\pi/L$. The 'height' of this peak is identical to $\tilde{F}(k)$ at this particular frequencies.

Why is this consideration important? When we want to compute a Fourier transform of a (continuous) function numerically, we will have to *sample* this function and to *truncate* it. Both can have subtle consequences you might not have in mind when naively applying the `fft` algorithm. The Fourier transform $F(k)$ we want wo compute will consequently be sampled and truncated as well.

Both aspects are the reason why the FFT should better be viewed in connection to Fourier series expansion than Fourier transform! The internal mechanism of FFT will always *assume* that you pass to it sampled data of the function over one period – and that this function is then repeated infinitely many times! What you get out is consequently a number of coefficients of a truncated Fourier series. As was shown above, these coefficients are *not* directly $F(k)$, but $\tilde{F}(k = m\cdot 2\pi/L)$, so the sampled Fourier transform of $\tilde{f}(x)$, i.e the function truncated to the interval. This interval is the numerical window you use and you see immediately that a proper choice needs to be made in order to get the correct result.

3.2 Sampled data considerations

I want to unravel here the mystery why `fft` does not need the input of any vector containing the sampling in x to “compute the Fourier transform”. As I said earlier, FFT will *always* assume a periodic function as input. Computing the sampled Fourier transform corresponds consequently to finding the Fourier series expansion coefficients c_m. In **first order approximation**, the Fourier integral can be represented by a sum. This yields

$$c_m = \frac{\Delta x}{L} \sum_{n=0}^{N-1} f_n \, \exp\left(-i\frac{2\pi}{L}\Delta x \cdot mn\right) . \tag{9}$$

Here, we find a specialty of the FFT sampling. When you sample data in N points, this corresponds to sampling the *interval* into $(N-1)$ segments. In contrast, The FFT samples the interval into N points *and* has N sampling points for the function. The reason is the assumed periodicity, which makes the last datapoint in the interval equal to the first datapoint of the next interval $f(L) = f(0)$, so that this point is literally *not* part of this interval. If we insert $L = N \cdot \Delta x$, this yields

$$c_m = \frac{1}{N} \sum_{n=0}^{N-1} f_n \exp\left(-i\frac{2\pi}{N}mn\right) . \tag{10}$$

Comparing to (2), we find

$$c_m = \frac{1}{N} \, \texttt{fft}[f_n] \ . \tag{11}$$

Note that all information about the spatial sampling has canceled completely. What remains is just the f-data in its “natural” space, i.e., integer sampling.

One question may arise at this point. I already mentioned that a FFT of data given in N sampling points yields also N points in the Fourier domain. It is not obvious why the index m in (10) should automatically run from $0 \ldots (N-1)$. Any value for m – including non-integer values in principle to compute *any* Fourier frequency – could be inserted as it seems. There are two reasons for that. First of all, `fft` is only *efficient* when m is an integer since it is based on using the mth complex square root of 1 and splitting the calculations. The second reason that it cannot *exceed* a certain value lies in the **Nyqusit-Shannon sampling theorem** which I want to loose just a few words on now.

3.3 Nyquist-Shannon sampling theorem, aliasing

There are some important questions related to sampled data and sampled Fourier transform that need to be addressed here. As we have seen in the last section, one might think that FFT can give back frequency components c_m for any value of m, i.e. for arbitrary high frequencies. The reason why this is not true is the Nyquist-Shannon sampling theorem. Its deeper foundation lies in the signal theory and is rather theoretical. As promised, I will therefore not address it here but focus on the outcome for our case.

I present here in short the main essence of the theorem in a form that is relevant to us.

The sampling rate of data determines the maximum frequency component that is resolvable. This ***Nyquist frequency*** *is equal to*

$$k_{max} = \frac{\pi}{\Delta x} = N \cdot \frac{\pi}{L} = \frac{N}{2} \cdot \Delta k \ . \tag{12}$$

Knowing this, not less people try to assign a frequency axis to the output of `fft` by using a command like `k = linspace(0, k_max, N/2)` which is not wrong, but in my eyes not instructive. The reason is that N could be an odd number, leading to confusion and struggling how to interpret the FFT result in terms of frequency. A better and easier way is to view the output of `fft` just in terms of frequency sampling and negative frequencies as I will show in the next section.

Before we do so, I want to emphasize another important consequence related to the sampling theorem which is **aliasing**. This problem relates more to the original question asked by Nyquist, Shannon and others. How many sampling points does a signal need to have so that its Fourier components are 'resolved'? In other words: If you want to find the Fourier transform of a function numerically, how many sample points or what sampling spacing do you have to use *at least*?

Imagine you want to perform this task for a narrow function, so that its Fourier transform is broad and has high frequency components. Now imagine that your chosen sampling is so that $k_{\max}$ is smaller than this high components. Consequently, you cannot compute them with your code, but what will happen instead? Your best hope might be that the Fourier transform might just be cut and is at last correct within the limited frequency range given by your sampling – but its worse. The low sampling rate will lead to contributions at *lower* frequencies which act as *alias* for the frequency components you cannot resolve any more. Sloppy speaking, FFT "works with the limited bandwidth it has" to represent your signal in Fourier space. This leads to *artificial* frequency components within the limited bandwidth which can be viewed as an unwanted numerical artifact. It is important though to use a sufficiently narrow sampling for a function in order to ensure that its Fourier spectrum fits into the computational window determined by that sampling.

4 Frequency bandwidth, negative frequencies

How to obtain the frequency vector for the FFT? As already said, this question can best be answered when we consider the fact that FFT will first of all lead to a constant sampling rate in the Fourier space. If we compare (2) and (5), having in mind (11), it becomes obvious that this frequency spacing is given by

$$\Delta k = \frac{2\pi}{L} \quad . \tag{13}$$

This is a first important result: If you want to increase frequency *resolution*, you have to use a larger computational window, not necessarily more sampling points as long as aliasing is not an issue!

Let us now assume we want to calculate the Fourier expansion coefficients c_m. Recalling again (11), we thus implement

```
c_m = 1/length(f) * fft(f)
```

since `length(f)` will give us N. An examination of (10) also tells us to which frequencies the result corresponds. Since m starts at 0 and runs in integer steps, the corresponding frequency vector must build up like

$$k_m = 0, 1\cdot\frac{2\pi}{L}, 2\cdot\frac{2\pi}{L}, 3\cdot\frac{2\pi}{L}, \ldots \tag{14}$$

If we insert $L = N\cdot\Delta x$, we have

$$k_m = 0, \left(\frac{2}{N}\right)\cdot\frac{\pi}{\Delta x}, \left(\frac{4}{N}\right)\cdot\frac{\pi}{\Delta x}, \ldots, \left(\frac{2m}{N}\right)\cdot\frac{\pi}{\Delta x}. \tag{15}$$

One may now come into confusion considering the Nyquist frequency. This "maximum meaningful" frequency $k_{\max}$ is reached when $m = N/2$, but as already mentioned m runs from 0 to $(N-1)$.

At this point, I have to say that the usual manual explanations can be misleading for FFT beginners, especially if they come from an optics background. In the MATLAB manual, examples of time data analysis are given where the "spectrum is truncated to the first half" since the other half would "not contain any information" but it is not clear what is actually meant. Is the second half of the FFT vector useless unphysical data?

The reason that this problem occurs is the different scientific background in the examples. The MATLAB examples – the "noisy sine", the "blue whale call", and the "asteroids observed by Carl Friedrich

Gauß" – solely care for *real* data. Phases play no role. Even in the only optical example – the "2D diffraction pattern" of a circular aperture – only the intensity is given at the end. No frequency scale, no words on the absolute scaling, no phases. Have you ever tried to implement the full diffraction formula with all prefactors like this when you care about absolute power?[1]

4.1 Real vs. complex Fourier series

For soleyly real data, the Fourier story is somewhat easier. First of all, a real function has a nice property. Its Fourier spectrum has a hermitian symmetry

$$f(x) \in \mathbb{R} \quad \Leftrightarrow \quad F(-k) = F^*(k). \tag{16}$$

This is no numeric mystery but a simple mathematical fact. It means that the real part of $F(k)$ is a *symmetric* (also called *even*) function while the imaginary part is *antisymmetric* (*odd*). It is thus only necessary to know one half of the spectrum *if* you already know that $f(x)$ is real. Indeed, the second half "does not carry new information" about the function in this case.

What is the effect on the Fourier series representation of $f(x)$? If we rewrite (5)

$$f(x) = \sum_{m=-\infty}^{\infty} c_m \exp\left(imx\frac{2\pi}{L}\right) \tag{17a}$$

$$= c_0 + \sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} c_m \cos\left(mx\frac{2\pi}{L}\right) + i \sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} c_m \sin\left(mx\frac{2\pi}{L}\right), \tag{17b}$$

and use the hermitian symmetry of the Fourier coefficients

$$c_{-m} = c_m^*, \tag{18}$$

we find

$$f(x) = \frac{a_0}{2} + \sum_{m=1}^{\infty} a_m \cos\left(mx\frac{2\pi}{L}\right) + \sum_{m=1}^{\infty} b_m \sin\left(mx\frac{2\pi}{L}\right), \tag{19}$$

with[2]

$$a_m = 2 \cdot \mathrm{Re}(c_m) \in \mathbb{R} \tag{20a}$$

$$b_m = 2 \cdot \mathrm{Im}(c_m) \in \mathbb{R}. \tag{20b}$$

This is the "sine-cosine" representation of the Fourier series. The coefficients a_m and b_m are real, and only *positive* indices m are necessary. This is why only one half of the FFT output is necessary to represent a *real* function in Fourier space.

But what about *complex* functions $f(x)$? In this case, it is obvious the whole spectrum will play a role, including the coefficients with *negative* indices m. The real sine-cosine representation will no longer be possible and we have to stick to the exponential form in (5).

How to calculate the negative frequency coefficients? Well, this is fairly easy, we just have to insert some negative value for m. Actually, there is a better way when we consider FFT. You may have already wondered that in (11) only positive frequencies seem to play a role, but I will show you that this is not the case. The key essence is the simple observation that you can calculate c_{-m} by calculating $c_{(N-m)}$

$$c_{(N-m)} = \frac{1}{N} \sum_{n=0}^{N-1} f_n \underbrace{\exp(-2\pi i \cdot n)}_{=1} \exp\left(i\frac{2\pi}{N}mn\right) = c_{-m}. \tag{21}$$

[1] It is not even explained why they choose the parameters like this. They use a numerical window for instance that is ≈ 100 times larger than the hole. Why not, say 10,000? Or, as you would perhaps more intuitively use 10? I recommend that you try it if you did not do it already. Let R become larger and smaller. Surprise, surprise! Your result will only barely look like the Airy function you were expecting and you wander how the result can depend on the sampling.

[2] Note that for real $f(x)$, c_0 is always real since it is in fact the function average.

These considerations shine a some new light on the relation between Fourier series and the FFT.

If you would like to represent a *real* function by a numerical truncated Fourier series, you can do so by using the *real* Fourier series representation given in (19). It will just involve just *positive* indices m and the series is truncated at the Nyquist frequency as result of finite sampling. Numerically, you can obtain the a_m and b_m coefficients by the `fft`, but only *half* of the computed spectrum really contains the information. This is why you will usually "throw away" the second half. In short, this means

$$a_m = \frac{2}{N}\,\texttt{real}(\texttt{fft}[f_n]), \qquad m = 0 \ldots \frac{N}{2}, \quad N \text{ even} \tag{22a}$$

$$b_m = \frac{2}{N}\,\texttt{imag}(\texttt{fft}[f_n]), \qquad m = 0 \ldots \frac{N}{2}, \quad N \text{ even} \;. \tag{22b}$$

Obviously, there is the question what happens if N is odd, but we will address this later.

You can also represent the same real function by a *complex* Fourier series representation (5). In this case, you have to consider also negative frequencies. This can be intuitively understood since sine and cosine are related to the exponential

$$\cos x = \frac{1}{2}(e^{ix} + e^{-ix}) \tag{23a}$$

$$\sin x = -\frac{i}{2}(e^{ix} - e^{-ix}). \tag{23b}$$

For a real function $f(x)$, the spectrum will possess the hermitian symmetry discussed earlier which shows that the complex form "does not contain more information" than the real form in this case. For a complex function, you *have to* use the complex form of the Fourier series in any case. In this representation, negative frequencies are necessary and also contain "real" information if $f(x)$ is complex. The *absolute* value of the frequency is bound by the Nyquist limit and consequently, frequencies can occur from $-k_{\max}$ to k_{max}. This is the range which is actually covered by the `fft`. The first half of the spectrum will contain the positive frequency components (including c_0), the second half will contain the negative ones. This translates to

$$c_m = \frac{1}{N}\,\texttt{fft}[f_n], \qquad m = 0,1,2\ldots\frac{N}{2}-1, -\frac{N}{2}, -\frac{N}{2}+1 \cdots -2,-1, \quad N \text{ even}, \tag{24}$$

which is a more precise version of (11). At this point, I still haven't explained what happens if N is odd. You might also wonder about the $\frac{N}{2}-1$. Now that we have understood why the `fft` output contains also negative frequencies, we will continue to examine the implications of even and odd length input.

5 Even vs. odd data length, truncation and padding

We now want to see in detail what happens exactly when we change N, especially understand the difference between an even and an odd input vector to `fft`. Even though it is useless in practice, we will imagine what the FFT output vector would contain if we gradually increase N, beginning with one.

Although a single value is hard to imagine as "data", there is no conceptual or mathematical difficulty. If f_n is just a single value, this corresponds simply to a constant function, since the FFT would regard the function as periodic. The Fourier transform of a constant function is a Dirac peak at the center of the frequency space and indeed, if we plug the single value into (10), we see that N^{-1} `fft`(f) yields c_0.

As we now increase the total number of points N in the data vector f_n, more and more Fourier orders will appear. For $N = 2$, we will have two values in the resulting vector. The first entry will be again c_0. The delicate question is now whether the second entry is c_1 or c_{-1}? This unveils the key question for even data input N. Since the Nyquist frequency can just be reached *once* for an even number of data points, will this correspond to a positive or negative frequency? I must admit that I haven't thought it through to the very end whether the reason is a convention or a numerical necessity associated with the

N	result of N^{-1} `fft`(f_n)	result of N^{-1} `fftshift`$($`fft`$(f_n))$	range for m
1	$[c_0]$	$[c_0]$	0
2	$[c_0, c_{-1}]$	$[c_{-1}, c_0]$	-1 ... 0
3	$[c_0, c_{+1}, c_{-1}]$	$[c_{-1}, c_0, c_{+1}]$	-1 ... 1
4	$[c_0, c_{+1}, c_{-2}, c_{-1}]$	$[c_{-2}, c_{-1}, c_0, c_{+1}]$	-2 ... 1
5	$[c_0, c_{+1}, c_{+2}, c_{-2}, c_{-1}]$	$[c_{-2}, c_{-1}, c_0, c_{+1}, c_{+2}]$	-2 ... 2
6	$[c_0, c_{+1}, c_{+2}, c_{-3}, c_{-2}, c_{-1}]$	$[c_{-3}, c_{-2}, c_{-1}, c_0, c_{+1}, c_{+2}]$	-3 ... 2
⋮	⋮	⋮	⋮

Table 1: Resulting vector for different length of the input vector. This scheme should clarify how the `fft` results should be interpreted in terms of Fourier components.

implementation of `fft`, but since we didn't want to go into unnecessary numerical detail, I can briefly give the answer. It is a *negative* frequency component and the resulting vector contains $[c_0, c_{-1}]$.

For $N = 3$, things are clearer since it is obvious that the resulting vector will contain, in addition to the center zero frequency component, an equal number of positive and negative frequencies. This means the result must be $[c_0, c_1, c_{-1}]$. However, note that this means that the Nyquist frequency, which would correspond loosely speaking to "$c_{\pm 1.5}$" in this case, is reached *nowhere* in the vector, but all frequencies are *lower* in that case!

Further increasing N would reproduce the same behavior explained above, just that more and more frequency components enter the game. It becomes clear that `fftshift` will now restore the "natural" order of the Fourier coefficients. Fig. 1 summarizes these results to clarify the scheme. **As a take-home message, it is important to note that for odd input, the center frequency will be in the middle, while even input causes one more negative data point and c_0 is found to the right of the middle after applying `fftshift`.**

It is now straightforward to determine the frequency vector from these considerations. It will simply be given by $m \cdot \Delta k$, where the spatial frequency spacing is given by (13). The range for m will be slightly different for even or odd vector lengths as explained above. However these differences can still be brought into a single formula when using the implementations `ceil` (rounding "towards the ceiling", i.e. next higher integer) and `floor` (rounding "towards the floor", i.e. next smaller integer). The result is

$$k_m = \left(-\texttt{floor}\left[\frac{N}{2}\right] : \texttt{ceil}\left[\frac{N}{2}\right] - 1 \right) \cdot \frac{2\pi}{L} \quad . \tag{25}$$

Truncation. An important point is the issue of truncating a Fourier series properly. Imagine that you have sampled a function in, say, 1000 points as in our initial example. Usually, the "useful information" about the spectrum is contained in a narrow range around the center frequency as demonstrated in Fig. 1. You want to use just this part for further calculation, not the high frequency components which are essentially zero. This is important for advanced numerical methods in optics like the Fourier Modal Method for instance, where you just use a couple of dozen Fourier orders since the system does not fit into your memory otherwise or the whole calculation takes forever.

How do we select out of this 1000 results for the Fourier coefficients the 30 (or 45, or any other small number) that are of interest to us? If the new intended length of our truncated series is given by M, we want to keep the zero frequency component in the middle of our truncated result and have $M/2$ points to each side of it. The remaining part of the original vector is thrown away.

Again we see that even or odd M would make a difference if we want to obtain consistent data. Moreover, we now have to distinguish between four cases since N and M can be even or odd independently. I will now go through the different cases as above, assuming you want to determine the indices to cut out M Fourier orders out of a resulting vector N^{-1}`fftshift`$($`fft`$(f_n))$ of length $N > M$. What you have to do is finding the "MATLAB indices" (i.e. positions in the result vector) of the Fourier indices you need to extract. We want to save them as a variable `range`. A difficulty arises here because we have to mix

case (example)	example data vector	MATLAB index range
MATLAB generic (5)	$[a_1, a_2, a_3, a_4, a_5]$	$1 \dots 5$
N odd (7), M even (4)	$[c_{-3}, \boxed{c_{-2}, c_{-1}, c_0, c_1}, c_2, c_3]$	$\frac{N+1}{2} - \frac{M}{2} : \frac{N+1}{2} + \frac{M}{2} - 1$
N odd (7), M odd (5)	$[c_{-3}, \boxed{c_{-2}, c_{-1}, c_0, c_1, c_2}, c_3]$	$\frac{N+1}{2} - \frac{M-1}{2} : \frac{N+1}{2} + \frac{M+1}{2} - 1$
N even (6), M even (4)	$[c_{-3}, \boxed{c_{-2}, c_{-1}, c_0, c_1}, c_2]$	$\frac{N}{2} + 1 - \frac{M}{2} : \frac{N}{2} + 1 + \frac{M}{2} - 1$
N even (6), M odd (3)	$[c_{-3}, c_{-2}, \boxed{c_{-1}, c_0, c_1}, c_2]$	$\frac{N}{2} + 1 - \frac{M-1}{2} : \frac{N}{2} + 1 + \frac{M+1}{2} - 1$

Table 2: Truncating a Fourier vector of length N to a shorter length M. The `range` is written in this unusual way to clarify where the index comes from and understand the resulting formula summarizing this table. The generic MATLAB indexing of any vector with positive integers is shown for comparison.

Fourier indices and the “generic” MATLAB vector indices here, that start at 1 with the first element in the vector.

Tab. 2 summarizes these considerations for the four cases, including an example for every case. It would be highly inconvenient if the index range would have to be determined each time separately in a program, depending on whether the numbers are even or odd. However, it turns out that if you rewrite $(M+1)/2$ as $(M-1)/2+1$ and make use of the fact that

$$\texttt{floor}\left(\frac{N}{2}\right) = \begin{cases} \frac{N}{2} & N \text{ even} \\ \frac{N-1}{2} & N \text{ odd} \end{cases} \tag{26a}$$

$$\texttt{ceil}\left(\frac{N}{2}\right) = \begin{cases} \frac{N}{2} & N \text{ even} \\ \frac{N+1}{2} & N \text{ odd} \end{cases}, \tag{26b}$$

it becomes possible to combine the range into one expression. The result is given by

$$\texttt{range} = \texttt{floor}\left(\frac{N}{2}\right) + 1 - \texttt{floor}\left(\frac{M}{2}\right) : \texttt{floor}\left(\frac{N}{2}\right) + \texttt{ceil}\left(\frac{M}{2}\right) \quad . \tag{27}$$

This allows you to extract M values out of a Fourier coefficient vector of original length N and thus consistently truncate your Fourier series. Of course the new frequency vector will now be shorter and its indices are determined by the new length M as

$$k_m = \left(-\texttt{floor}\left[\frac{M}{2}\right] : \texttt{ceil}\left[\frac{M}{2}\right] - 1\right) \cdot \frac{2\pi}{L}. \tag{28}$$

Note that the frequency spacing $\Delta k = 2\pi/L$ is *still* determined by the (physical) length of the computational window that was the original input to the `fft` function! Truncation simply means that you use a smaller subset of Fourier components and neglect all others.

Padding. The exercise of padding data is the inverse problem to truncating data. Imagine you have computed a Fourier transform and truncated it to 50 points since all other Fourier components were

essentially zero. Now you want to get back the original function from this and perform an inverse Fourier transform. Your result will have 50 points and thus look very badly sampled and ugly. What you would like to have is a function which is again sampled in, say, 1000 points. How to achieve this? The first and most intuitive method would be to recall that you actually compute Fourier coefficients c_m when using FFT. From their knowledge, it is straightforward to reconstruct the function using (5). There you see that you can basically sample your x vector into as many points as you like!

This is a point that often causes confusion. It is widely believed by freshmen that 50 points in Fourier space would "just be good for" 50 points in real space and vice versa. This is not the essence of the Fourier business. The correct version is that 50 points in Fourier space are good for 50 *informations* in real space! I try to explain this elusive sentence a little bit.

Your 50 points in Fourier space tell you which sine and cosine wave (or complex exponential) of which frequency is how strong in the "harmonic composition" of your signal. Numerically, you have to end your series at what we called the Nyquist frequency, the maximum resolvable frequency in your signal. *If* your signal has substantial contributions from higher frequencies (corresponding to sharp details in real space), you have killed these with your numerics. But if this is *not* the case, no matter how many points you draw on your reconstructed function, the main details are already resolved. Adding points cannot and will not make any details sharper or add any new information. Loosely speaking you are just interpolating the essential data (and those "generic" 50 points in real space are the essence of the function!) which is justified since you already *know* that between two points, there *cannot* be a sharp variation since this would correspond to a frequency contribution higher than the Nyquist frequency.

In the context of FFT, there is the possibility to pad data with zeros to both sides until you reach your desired number, and then transform it back. If you add 475 zeros to each side of your data vector and then perform the back transform, you will obtain a nice smooth 1000 point sampled original function. You did not add any information, there is no mystery involved.

I will not reproduce the considerations above regarding where to add the zeros in the different cases of even or odd data since the problem of finding the generic indices does not exist in that case. Basically, if you have a sharp look onto Tab. 2 again, you see what you have to do, just be sure to have the center frequency at the right place. If you want to pad your vector with length N to a length M, its best to consider the difference $M - N$. If this difference is even, it is trivial to add half of it to each side. If it is odd, the question is where to put the "remaining" zero. This will depend on N. If N is even, add the additional zero on the right, if it is odd, on the left.

A word of caution. MATLAB comes with some implemented functionality to truncate or pad `fft` data. You may not resist the seduction of performing the aforementioned task by implementing `fft`$(f_n, 50)$ and `ifft`$(F_n, 1000)$, respectively. The result will be vectors of the length you want, but the result may be far from expected. The reason is the way MATLAB performs the truncation and padding. When it truncates, it does *not* truncate the Fourier series anyhow, but *simply cuts your data input vector*! Accordingly, it just adds zeros to the end of the vector to perform the padding. While this behavior may be of limited use when you perform a time series analysis of time data that has tens of thousands of points and you select a smaller window, doing so in a physical context is, carefully speaking, unwise, and, strictly speaking, screwball. It is unnecessary to say that simply cutting your input data basically screws up all your data. The data length L is changed, a part of the data is missing, nothing is going well as expected. The same story happens for the padding. The single-sided adding will shift your input data and consequently modulate your output. I cannot think of any situation where the internal truncation and padding implementation could be of any use in this context. I strongly recommend that you implement this manually.

6 "Physical Use" of the FFT

After we have now understood the different aspects associated to the use of the FFT, it is the aim of this section to derive the implementation of a "physical" Fourier transform, i.e. a sampled realization of the

Fourier integral with all the prefactors involved.

We use the definitions (3) above and perform a first order approximation in the sampled data. This yields

$$\begin{aligned} F(k) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x)\, e^{-ikx}\,\mathrm{d}x \\ &\approx \frac{\Delta x}{2\pi}\left[\sum_{n=0}^{N-1} f_n\,\exp\left(-ikx_n\right)\right]. \end{aligned} \tag{29}$$

The spacial sampling x_n can be expressed using the sampling distance Δx and the absolute shift of the x data x_0 (which is the first element of x)

$$x_n = n\cdot\Delta x + x_0. \tag{30}$$

This yields

$$F(k) = \frac{\Delta x}{2\pi}\,\exp\left(-ikx_0\right)\left[\sum_{n=0}^{N-1} f_n\,\exp\left(-ik\Delta x\cdot n\right)\right]. \tag{31}$$

Up to now we did not consider that the data for $F(k)$ will be sampled as well. We could still insert any k in the Nyquist interval. If we take into account what we have learned so far about the FFT, we write the sampled frequencies as

$$k_m = m\cdot\Delta k = m\cdot\frac{2\pi}{L} = m\cdot\frac{2\pi}{\Delta x\cdot N}, \tag{32}$$

to get

$$F(k_m) = F_m = \frac{\Delta x}{2\pi}\,\exp\left(-i\frac{2\pi}{L}\cdot x_0\cdot m\right)\underbrace{\left[\sum_{n=0}^{N-1} f_n\,\exp\left(-i\frac{2\pi}{N}\cdot mn\right)\right]}_{\texttt{fft}(f_n)}. \tag{33}$$

The remaining task is to get the indices into a “physical” order using `fftshift`. The end result is thus

$$F_m = \frac{\Delta x}{2\pi}\,\exp\left(-i\frac{2\pi}{L}\cdot x_0\cdot m\right)\cdot\texttt{fftshift}(\texttt{fft}(f_n))\ , \tag{34}$$

where m is as explained above given as

$$m = \left(-\texttt{floor}\left[\frac{N}{2}\right] : \texttt{ceil}\left[\frac{N}{2}\right] - 1\right)\ . \tag{35}$$

The exponential prefactor ensures that the relation between the spatial sampling in x, constituting an absolute coordinate system, and the generic indexing by the `fft` implementation does not introduce an unwanted modulation.

A special and often used case is when the x vector is *zero-centered*, i.e. the origin is either in the middle (odd length) or right from the middle (even length). In this case, the correcting factor can be canceled if the origin is shifted to the beginning of the data vector before issuing `fft`. This can be done by `ifftshift` and yields

$$F_m = \frac{\Delta x}{2\pi}\,\texttt{fftshift}(\texttt{fft}(\texttt{ifftshift}(f_n))) \qquad (x \text{ vector zero-centered}). \tag{36}$$

Appendix

A Fourier transform of a periodic function with finite repetitions

I give here the result for the Fourier transform of a periodic function with *finite* extent, since the result is useful in optics (gratings with a finite number of finite periods etc.). Starting from (4), we set[3]

$$f(x) = \sum_{n=0}^{N-1} \tilde{f}(x - nL), \qquad a, b \in \mathbb{Z}. \tag{37}$$

To get the Fourier transform, we make use of the Fourier shifting theorem

$$\mathscr{F}[g(x - x_0)] = e^{-ikx_0}\, G(k). \tag{38}$$

This yields

$$\mathscr{F}[f(x)] = \tilde{F}(k) \sum_{n=0}^{N-1} e^{-iknL} = \tilde{F}(k) \sum_{n=0}^{N-1} \left[e^{-ikL}\right]^n. \tag{39}$$

The sum is a complex geometric one with the value

$$\sum_{n=0}^{N-1} q^n = \frac{1 - q^N}{1 - q}. \tag{40}$$

This gvies

$$\begin{aligned} F(k) &= \tilde{F}(k) \frac{1 - e^{-iNkL}}{1 - e^{-ikL}} \\ &= \underline{\underline{\tilde{F}(k) \frac{\sin(NkL/2)}{\sin(kL/2)}\, e^{ikL(1-N)/2}}}. \end{aligned} \tag{41}$$

[3]It must be noted that this assumes the *absolute* position of $\tilde{f}(x)$ in the first interval. If the function is shifted, a phase factor needs to be applied.