PH20105, Hand-in-exercise 2020-2021

This year's hand-in exercise is about Fourier transforms. Fourier transforms play a large role in many areas of physics, including the analysis of sound, electrodynamics, fiber optics, quantum mechanics, field theory etc. You will write C code capable of reading a file from disk, performing a Fourier transformation, identifying the key waveforms that make up the signal and reverse transforming back to the original signal. Details of how to hand in the exercise and about the marking scheme are provided at the end.

Introduction: discrete Fourier transforms

The basic theory of Fourier transforms is as follows. Say we have a periodic function $h(t)$ with period T (during the exercise, we will actually use $T = 2\pi$ for simplicity). This could be the total amplitude $h(t)$ of sound at a time t, with the values at all times together making up a complete song and shorter time intervals individual notes (the *periodic* part does not matter as much as long as the song begins and ends with the same value, e.g. zero). Its Fourier transform is given by

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

which for the example of sound would amount to decomposing the sound into its harmonics. Now $H(\omega)$ is the amplitude of the wave of angular frequency ω (related to frequency ν according to $\nu = \omega/(2\pi)$. The separate waves are represented by

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

Here we use the complex representation of waveforms, which allows for the possibility that $h(t)$ and $H(\omega)$ are complex functions (t and ω remain real). This is not needed for sound waves, which are real functions. But in quantum mechanics, the wave functions $\Psi(x,t)$ are complex. All waveforms can be combined again to jointly form the original function, according to

$$h(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H(\omega) e^{i\omega t} \mathrm{d}\omega. \tag{3}$$

Transforming a function back and forth should yield the same function again, i.e.

$$h'(t') = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(t) e^{i\omega(t'-t)} \mathrm{d}t \mathrm{d}\omega = \int_{-\infty}^{\infty} h(t) \delta(t'-t) \mathrm{d}t = h(t'). \tag{4}$$

This uses that the *Dirac delta function* can be represented using

$$\delta(t'-t) \equiv \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i\omega(t'-t)} \mathrm{d}\omega. \tag{5}$$

Delta functions are deceptively subtle, and this representation might not be immediately obvious. For the purpose of this exercise, it suffices to realize that this

implies that two waveforms with different ω will cancel each other out when integrated over t:

$$\omega \neq \omega' \rightarrow \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{-it(\omega'-\omega)}\mathrm{d}t = 0. \tag{6}$$

In terms of mathematical jargon, the waveforms can be said to form an orthogonal basis in the space of functions. Which is why when hearing a recording of an instrument playing a certain note at a given pitch, you will not all of a sudden identify it as a completely different note the next time your listen to the recording: your mental decomposition of the sound into waveforms is unique.

The same principle applies to quantum mechanics, where a given state of the system gets decomposed into waveforms of a particular type through the act of measurement. Of course, the two bizarre things in quantum mechanics are (1) a measurement also implies that you take *one* of those waveforms and discard the others, and (2) measuring a second, different quantity (e.g. momentum, rather than position) means using a different *type* of waveforms that is not generally aligned with the first and might itself be built up out of many waveforms of the first type when decomposed that way.

Getting back to numerics, the first thing to realize is that we work with a *discrete* system, rather than a *continuous* one. Instead of using an integral, let us therefore introduce the *discrete* Fourier transform:

$$H_n(\omega_n) = \sum_{k=0}^{N-1} h_k(t_k) e^{-i2\pi nk/N}. \tag{7}$$

This transformation is applied to a function $h(t_k)$ that has been *sampled* N times in the interval T, with a sampling interval $\Delta = T/N$ and a sampling rate $1/\Delta$. Everything that we know about $h(t)$ is what is provided to us through the N values $h_k = h(t_k)$, where $t_k = k\Delta$ and $k = 0, \ldots, N-1$. The periodicity requirement on the function implies that $h(N\Delta) = h(0)$, and the last point included in the sample is h_{N-1}. The angular frequencies ω_n are given by $\omega_n = n2\pi/N\Delta$. The Fourier transform generates N values H_n, with $n = 0, \ldots, N-1$, measuring the relative contributions of their respective ω_n's to the original function sampled through h_k. The inverse discrete Fourier transform is given by

$$h_k(t_k) = \frac{1}{N}\sum_{n=0}^{N-1} H_n(\omega_n) e^{i2\pi nk/N}. \tag{8}$$

The exercise

Question 1: Fourier transform theory

Although this is a programming exercise, we start with a theory question to help you on your way. Suppose we have a function $h(t) \equiv \exp[i\omega_1 t]$, with $\omega_1 = 1$. For $N = 4$, please compute the values of H_0, H_1, H_3 and H_3 by hand and interpret the outcome.

Question 2: The outline and approach of your program

You will need to write a program that performs a number of tasks to be detailed in question 3. For question 2, read question 3 in its entirety and describe how you plan to approach this, choice of data types, program structure, etc. Focus on being complete and readable, but not overlong (ie about one page). Keep in mind that you will also be using your program to demonstrate your understanding of a broad range of C language concepts, which does not necessarily leads to exactly the same design choices everywhere as when you aim to write the most efficient code and/or the most concise source code etc. It is therefore important that you *motivate* the particular design choices that you are making.

Question 3. The program

Please implement the items below. All your C code should be contained in a single `.c` source file, with a single `main` routine. You can use any method discussed during the lectures to generate your plots. Along with the answers to the questions, your pdf report *must* include a listing of the full source code and images of your plots.

a Define two C functions that respectively implement two mathematical functions, $h_1(t) = \exp[it] + \exp[i5t]$, and $h_2(t) = \exp[(t-\pi)^2/2]$. Both mathematical function returns are *complex* numbers (even though h_2 only has a real part), so you will need to find a way to define your C functions such that, when provided with a real number t, they are able to provide you with what amounts to a complex number (i.e. *two* real numbers worth of information rather than one). You are *not* allowed to use any complex number functionality from an existing library routine.

b Sample these functions $N = 100$ times each over a time period $T = 2\pi$, and write the results to two text files for plotting. There is no need to include the text files themselves in your report.

c (outside of your C program) generate plots for both functions and include these in your report. These should be combined with the plots requested in question 3h below.

d Use a discrete Fourier transform to obtain $H_1(\omega)$ and $H_2(\omega)$ from $h_1(t)$ and $h_2(t)$ respectively.

e Print the results for H_1 and H_2 onto the screen.

f Apply an Inverse Fourier transform to H_1 and H_2 to obtain h_1' and h_2'. But for H_1 skip $n = 1$, and for H_2 skip $n = 0$.

g Write the outcomes of the inverse Fourier transforms to text files for plotting. Again, no need to include the text files themselves in your report.

h (outside of your C program) generate plots for both h_1' and h_2'. Please overplot these on the plots for h_1 and h_2 respectively.

i Load the sampling data from the file `h3.txt` (provided on Moodle) into memory. This file contains $N = 200$ samplings of a function h_3.

j Apply a discrete Fourier transform to h_3 in order to obtain H_3.

k Apply an inverse discrete Fourier transform to H_3 in order to obtain h_3'. But only include the *four* terms out of the N terms for $H_3(\omega_i)$ with the largest amplitude, $|H_3(\omega_i)|$.

l Write the outcome for h_3' to a text file. No need to include your text file in the report.

m (outside of your C program) overplot h_3' (from your own text file) and h_3 (from the provided file).

In summary, when run, your program should produce a number of separate (5) text files: for h_1, h_2, h_1', h_2' and h_3', and show the results for H_1 and H_2 on screen.

Question 4. Interpretation

Please briefly interpret the key features of the three figures (h_1 and h_1'; h_2 and h_2'; h_3 and h_3') that you have produced, as well as the outcomes for H_1 and H_2.

Marking of the exercise

The marking scheme for the exercise is as follows

Proper hand in (5/100)

You will need to submit *two* files on-line for marking:

- A `.c` file containing your program in its entirety. This will be checked, so should compile and run without problems or requiring edits. Therefore do NOT put this in a word document or anything else other than plain text. Filename: `23456.c` (replace with your student number)
- Your report in *pdf* format, with filename `23456.pdf` (replaced with your student number). The pdf file should contain:
  - Answer to question 1
  - Answer to question 2
  - Answer to question 3
  - The plots: h_1 vs h_1', h_2 vs h_2' and h_3 vs h_3'.

– The source code. Note that I therefore get your source code *twice*. Once in a C file that I can compile and run, and once as part of the report so that I can annotate it if needed.

A failure to follow these instructions *exactly* implies zero points. Reports handed in after the deadline will in principle *not* be marked.

Question 1 (5/100)

Answering question 1 can get you up to 5 points.

Questions 2 and 3 (80/100)

A completely working program that correctly achieves all goals is worth 20 points, fewer points for programs with mistakes. The formatting and lay-out of the program are worth up to an additional 30 points; this includes annotation, consistent indentation, proper use of empty lines and reasonable variable names, as well as overall structure (functions). The annotation of your program also serves as a means for me to establish your submission as something genuinely your own. The different tasks performed should consistently be properly signposted and commented upon, but without undue verbiage.

A further 30 points can be earned by demonstrating an appropriate use of non-trivial C programming concepts, e.g. structs & pointers and by describing and motivating your code well. This is done in part in your answer to question 2, so make sure your program does indeed follow your outline. Regarding the text of question 2, the more sensible and better motivated your approach, the more points. The clarity of your writing is included in the assessment.

Question 4 (10/100)

Ten points can be obtained by demonstrating an understanding of the features of the plots and of H_1 and H_2.

HJvE & demonstrators, Feb 2021