PH20105 C Programming hand-in exercise

Candidate Number: —–
Department of Physics, University of Bath
Bath BA2 7AY, UK

Question 1: Outline and approach of the program

You will need to write a program that performs a number of tasks to be detailed in question 2. For question 1, read question 2 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.

To start with, the program will have this overall structure: header files imports, constants, function headers, the main() function, where the functions will be executed to answer question 2 step by step, and function definitions. The definition of the functions will go at the end of the program so that the main() function is accessible, closer to the top.

The exercise is about Fourier transforms of complex functions, so a way to handle complex numbers has to be found. In this program, a structure resembling a complex number (CN) will be used, which will store the real and imaginary part of a complex number.

The most important functions of the program will be the Discrete Fourier Transform ($DFT()$) and Inverse Discrete Fourier Transform ($IDFT()$). These will use dynamic memory as an efficient way of handling data and also as a demonstration of how pointers are used. Dynamic memory is passed to these functions both for reading values and for writing them. When passing dynamic arrays to write values, the arrays will be passed with a & to access the memory address rather than the values themselves, which enables modifying the arrays.

Before using these, $h1()$ will use static arrays to demonstrate how they are used to store function results. C does not allow to return static arrays from functions, at least with the resources we learnt in the module, since it does not allow to return pointers to local variables. Being that so, functions can only modify external arrays. Using arrays in this program is sensible because the array size is fixed from the start.

In an attempt to return arrays using what was taught in the module, $h2()$ returns a structure object which stores an array of complex numbers. The $h2\_result$ structure will be defined for this purpose. This could be useful in a case where modifying external arrays can produce errors, for example when many arrays are needed or an array has to be modified many times. In this program it is just used as a demonstration.

Regarding encapsulation, a function will be used to filter the needed frequencies coefficients in the transforms. This function will check if a given number is in an array. It will check if the index of a transform coefficient is in a given array of indexes to skip. In the case of $H3$, the four elements with largest modulus have to be skipped in the inverse transform. For this, there will be another function, which will iterate over an array of magnitudes (calculated with another function) and retrieve the indexes of the maximum four by iterating and comparing elements. By putting repeated tasks and long tasks into functions, the main() program will look more concise and easy to understand.

The output files will be comma separated for readability and easiness of handling in future. Also, the files will be opened with dynamic memory, since the length of the file is difficult to know beforehand.

Respecting errors arisen from using functions and especially handling files, they will be handled throughout the program using conditional statements (e.g. if()) to check for invalid parameter values or file opening failures. Error messages will be printed to the console to indicate issues during program execution.

Most of the values or variables apart from the $h1\_values$, $h2\_values$, $h3\_values$ values and their transforms will be stored in static arrays for simplicity, since they will be just a means, temporary variables, to calculate the ones in dynamic memory. When using static arrays, there is no need to manage memory manually and that also makes them less prone to errors. Also, they are stored in the stack, the most accessible memory.

Constants will be defined at the start so that they are immutable. Double type variables will be used to calculate the transforms. These are preferred because the transform functions imply many changes, many calculations and if the precision at each of those changes is lower, then the accumulative effect is a bigger error. Integers will be used for step numbers and sizes, since C only accepts integers as index. The character type *char* will be used to manage files because this is the most general type, without any specification of sign, appropriate when no restrictions want to be put to the files.

Question 2: 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.

1. *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.*
2. *Sample these functions N=100 times each over a time period T=2π, and write the results to two text files for plotting. There is no need to include the text files themselves in your report.*
3. *(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.*
4. *Use a discrete Fourier transform to obtain* $H_1(\omega)$ *and* $H_2(\omega)$ *from* $h_1(t)$ *and* $h_2(t)$ *respectively.*
5. *Print the results for* H_1 *and* H_2 *onto the screen.*
6. *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.*
7. *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.*
8. *(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.*
9. *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.
10. *Apply a discrete Fourier transform to* h_3 *in order to obtain* H_3.
11. *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,* $|H_3(\omega_i)|$.
12. *Write the outcome for* h_3' *to a text file. No need to include your text file in the report.*
13. *(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.*

The source code is included as an appendix.

The plots:

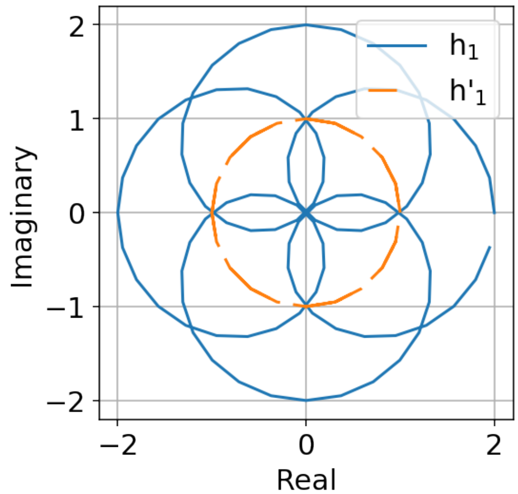

Figure 1: h_1, the original function and the IDFT.

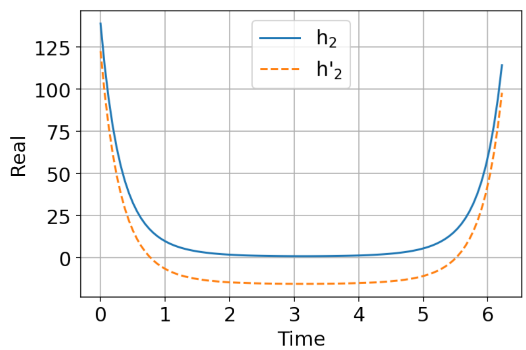

Figure 2: $h_2(t)$, real part of the original function and the IDFT versus time. This and not the function itself is plotted because, having no imaginary part, the function itself is just a constant line in the real axis, and it is not as informative as the real part curve.

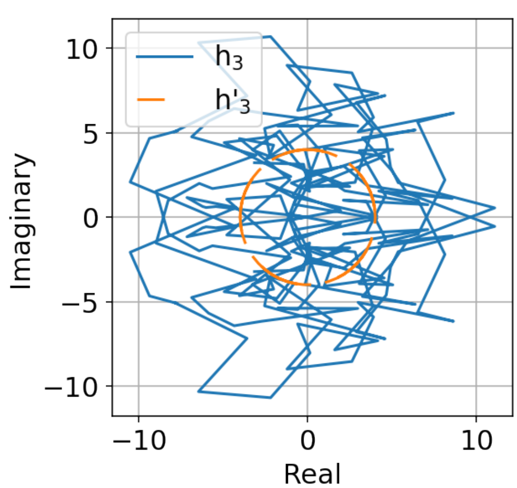

Figure 3: h_3, the original function and the IDFT.

Question 3: 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.

In the case of $h_1(t)$, $H_1(\omega)$ shows that $h_1(t)$ is composed of signals of only two different frequencies and the contribution of one of them, the corresponding one to $n = 1$, is removed. That makes the $h'_1(t)$ a perfect sinusoidal signal, with constant amplitude, for both the real and imaginary part. That is why the shape of h'_1 is a circle as seen in Figure 1.

In the case of $h_2(t)$, $H_2(\omega)$ shows that $h_2(t)$ has the contribution of many frequencies and the one by the frequency with $n = 0$ is a constant since $\omega_0 = 0$. When the contribution of this frequency is removed, about 16 is subtracted from all the signal values. This difference can be checked graphically in Figure 2 or by a calculation in the C program. The difference is most noticeable around the minimum of the curve as function of time, as can be seen in Figure 2, where the slope is not as high and the two curves clearly do not overlap. With regards to the magnitude, it gets larger around the minimum, since the real part of the original $h_2(t)$ is about zero there and also the magnitude, but shifting the lower limit to a negative value makes the magnitude larger.

In the case of $h_3(t)$, the effect of just taking into account the contributions of the frequencies for which the transform coefficient $H_3(\omega)$ is one of the four maxima in magnitude is making the magnitude of the signal constant. This makes $h'_3(t)$ a circle as seen in Figure 3 as in the case of $h'_1(t)$. The original signal had small contributions of many frequencies apart from the main ones, making h_3 to have a very irregular shape. By only keeping the most significant frequencies, this irregularities go away.

None of the $H_1(\omega)$ and $H_2(\omega)$ values has an imaginary part, so their phase is zero. This means the offset of the sinusoidal component of all frequencies relative to the origin (considering the original signal starts at the origin) is zero as well.