PH20105 C Programming hand-in exercise

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

June 25, 2024

Answer to question 2

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 3 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.

Answer to question 3

The source code is included as an appendix.

The plots:

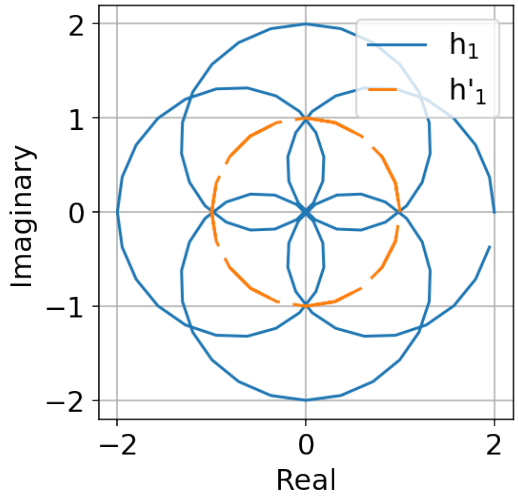

Figure 1: h_1, the original function and the IDFT.

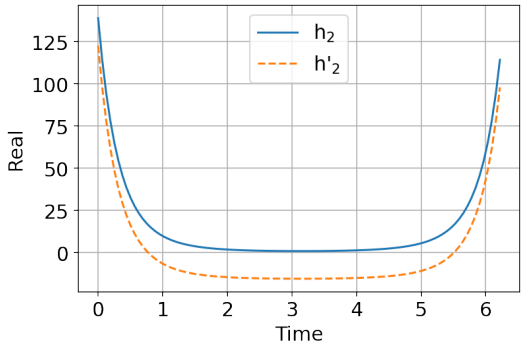

Figure 2: $h_2(t)$, real part of the original function and the IDFT versus time. This and not the function itself is plotted because, having no imaginary part, the function itself is just a constant line in the real axis, and it is not as informative as the real part curve.

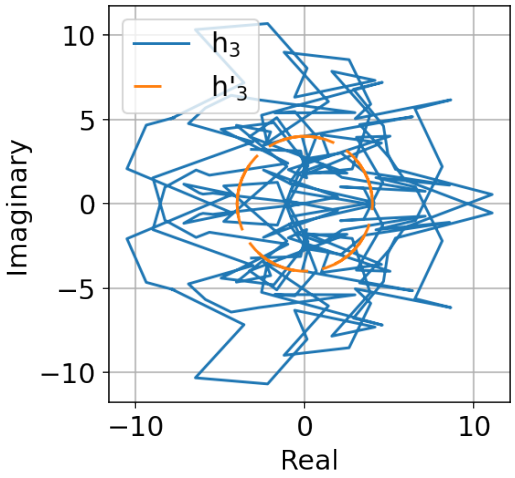

Figure 3: h_3, the original function and the IDFT.

Answer to question 4

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.