Pre-lab: Understanding the Fourier Transform with Applications to NMR and IR Spectroscopy

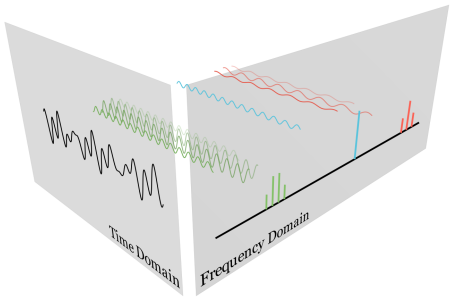

Figure 1: A sample FID in the time domain (black) and its component sine waves (green, blue, red). Each of these individual components correspond to a specific frequency in the frequency domain, which, when plotted, resembles an ^{1}H NMR spectrum.

Introduction

It is often said that the Fourier transformation (FT) takes your collected data, e.g., the free-induction decay (FID), from the "time domain" to the "frequency domain". But what does this really mean? And how can we gain a better understanding of this mathematical tool while learning more about the chemistry that it helps to elucidate? The goal of this pre-lab is to develop the tools needed to begin this journey. We will start by refreshing some familiar mathematical concepts and then learn how to use them by writing Python code in a Google Colab notebook.

Part 1: Mathematical Preliminaries

In this lab, we will model the free-induction decay (FID) as a variation of a sinusoid, so we should refresh their anatomy. As an example, a sinusoid is often written as $x(t) = A\cos(\omega t + \varphi)$ where x is the dependent variable with units of length (meters) and is a function of the independent variable t with units of time (seconds). Here, A is the amplitude of the wave (in meters), ω is the angular frequency (in radians/second), and φ is the phase (in radians). Please see the figure below for an illustration of these concepts.

1. Consider the following sinusoid:
$$x(t) = 4\cos(2t). \tag{1}$$
   a) What is the amplitude of $x(t)$?
   b) What is the frequency of $x(t)$?
   c) What is the phase of $x(t)$?
   d) Is $x(t)$ an even or an odd function?
   e) Draw $x(t)$ on the interval $[0, 2\pi]$.

Imaginary numbers play a critical role in the mathematics of the FT and so we will use this exercise as a gentle introduction or review of imaginary numbers and their properties. A complex number is a number that is written as $z = a + ib$, where a and b are *real* numbers and $i = \sqrt{-1}$ is the imaginary unit. Let's look at the complex numbers $z_1 = 1 + i$ and $z_2 = 4 + \sqrt{2}i$. We can add and subtract them as follows

$$\begin{aligned} z_1 + z_2 &= 5 + (1 + \sqrt{2})i \\ z_1 - z_2 &= -3 + (1 - \sqrt{2})i. \end{aligned} \tag{2}$$

We can also multiply them using the algebra rules that we are familiar with (using $i^2 = -1$):

$$z_1 \cdot z_2 = (1 + i)(4 + \sqrt{2}i) = (4 - \sqrt{2}) + (\sqrt{2} + 4)i. \tag{3}$$

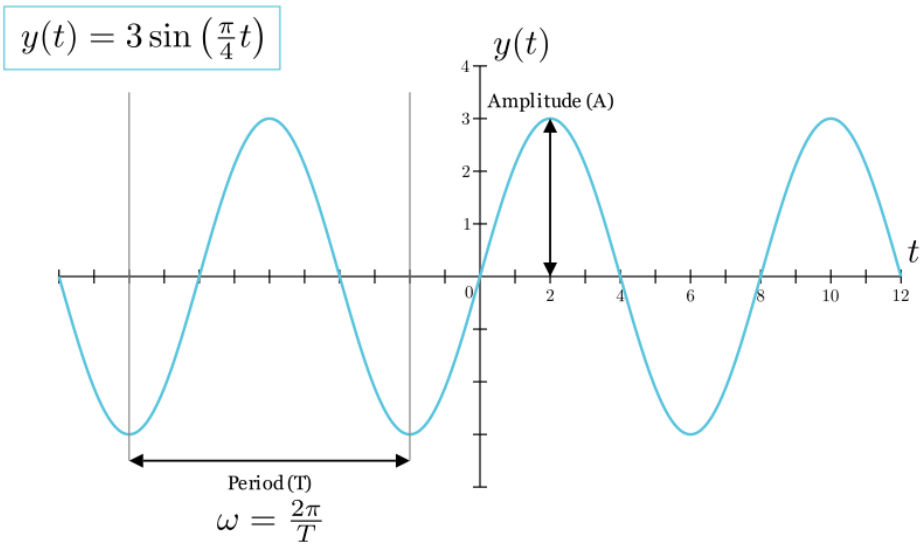

Figure 2: A graph of $y(t) = 3\sin\left(\frac{\pi}{4}t\right)$ with the components labeled.

2. Let's practice some manipulations with $z_1 = 2 + 3i$ and $z_2 = -1 - 5i$.

   a) Compute the sum and differences $z_1 + z_2$ and $z_1 - z_2$.

   b) Compute $z_1 \cdot z_2$.

Additionally, we define the complex conjugate of a complex number $z = a + bi$ to be $z^* = a - bi$, i.e., every i gets replaced with $-i$. For example, if $z = 4 + 3i$ then the complex conjugate of z is $z^* = 4 - 3i$.

3. Again, we will practice with $z_1 = 2 + 3i$ and $z_2 = -1 - 5i$.

   a) Compute z_1^* and z_2^*.

   b) Compute $z_1 \cdot z_1^*$.

Now that we have a grasp on complex numbers, we can look at one of the key ingredients to the FT: the complex exponential e^{ix}. At first glance, this seems strange... You may ask, "what does it even mean to have an imaginary unit in the exponent?" It turns out that we can use Euler's identity to rewrite this complex exponential as something a bit more manageable:

$$e^{ix} = \cos(x) + i\sin(x). \tag{4}$$

4. Here, we invite you to get a sense for a few properties of the complex exponential so that you can feel comfortable when we later see it in action.

   a) Show that $e^{i\pi} = -1$.

   b) Show that $e^{-ix} = \cos(x) - i\sin(x)$. *Hint*: Use the definitions of even and odd functions.

   c) Show that $\frac{1}{2}(e^{ix} + e^{-ix}) = \cos(x)$. *Hint*: Expand both complex exponentials in terms of trigonometric functions and then simplify.

Since the FT acts as a bridge between the time domain and the frequency domain, both frequency and time will be present in our final function. The next example contains a function of both x and y and reviews how to integrate a function of two variables with respect to one.

5. Consider the function $f(x, y) = y\sin(x)$. If we aim to compute the integral of this function with respect to the variable x, then we are allowed to treat any variable that *is not* x as a constant. To see this, we will integrate $f(x, y)$ with respect to x from on the interval $[0, \pi]$:

$$\int_0^\pi y\sin(x)\,dx = y\int_0^\pi \sin(x)\,dx = -y\cos(x)\Big|_0^\pi = -y\cos(\pi) - [-y\cos(0)] = 2y. \tag{5}$$

   In this example, we observe that integration of $f(x, y)$ with respect to x returns an output that is **only** a function of y. Mathematically, we can write $g(y) = \int_0^\pi y\sin(x)\,dx$.

a) Using the above example as a guide, try to compute $\int_0^\pi f(x,y)\,\mathrm{d}y$?

b) What do you observe? Is the solution a function of x or y?

c) Now suppose that you are given some new function $h(\omega,t) = e^{i\omega t}f(\omega)$. If we integrate h with respect to t, we will get a new function $\hat{h}$. Will $\hat{h}$ be a function of ω or t?

6. In each of the previous exercises we have practiced working with the components of the FT. We are now ready to define it. The FT of a function $f(t)$ is given by

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} e^{i\omega t} f(t)\,\mathrm{d}t. \tag{6}$$

a) Using the results of Question 5, justify the fact that the left-hand-side of Eq. (6) is *not* a function of time, t.

b) Let's get back to the idea of FT-NMR. Ultimately, our goal is to understand how we transform our FID signal from the 'time domain' to the 'frequency domain'. As a first step, let's label the following inputs and outputs on Eq. (6): frequency, time, complex exponential, FID (time-domain), frequency spectrum.

Part 2: Google Colab

Setting Up Google Colab

You may find this introduction to Colab notebooks to be useful.

1. Download the "Prelab_Exercise.ipynb" file provided from your instructor and save it in a known location.
2. Sign into a Google account and navigate to Google Colab (`https://colab.research.google.com/`).
3. Under the "Welcome To Colab" title in the top left, select "File," then "Upload Notebook".
4. Select "Browse," then navigate to where the you saved the "Prelab_Exercise.ipynb" file and select it.
5. The template for Exercises 3 and 4 should now be loaded into Google Colab. *Note:* the following instructions will be displayed both here in the prelab and also in the "Prelab_Exercises.ipynb" file in Google Colab.

Part 3: Python Fundamentals

1. One of the first exercises everyone writes in a new programming language is a line of code to print "hello world" to the screen. To do this, we will click into the first coding cell and type the following into the it:

```
print("hello world")
```

To run this code, we can either press the "play" button that is directly to the left of the cell, or we can press **Shift** and **Enter** at the same time on our keyboard. You should see the phrase `hello world` print just below the cell. Here, we have to include quotation marks around this phrase in the `print` command so that Python interprets the words as a `string`, rather than a numerical value.

Code written following a `#` will not be executed when the cell is run. You can use this to leave yourself comments as to what your code is doing. *Note:* If you get a message indicating that the notebook was not authored by Google, you can safely ignore it and select "Run anyway".

2. We can also run basic math calculations in these cells. For example, try computing the following:

a) 4 + 2

b) 4 * 2

c) 4**2

d) 4 / 2

Hint: If you type all of these operations into one cell together, your output will only show the result of the last line. To view the outputs of each operation at once, we recommend using the `print` command by putting each math operation into a separate `print` statement.

3. Before we start, we will need to setup a few things. We will need to perform math operations beyond those demonstrated above and plot our results. We will enable these functionalities by importing "libraries", or freely available, pre-written helper-programs. In a new cell, type the following:

```
import numpy as np
import matplotlib.pyplot as plt
```

   Again, we can run this code using either the "play" button that is directly to the left of the cell or the **Shift + Enter** on our keyboard.

   To familiarize yourself with `numpy` (pronounced num-pi) we will first start with some simple mathematical operations. Python is capable of basic operations such as addition, subtraction, multiplication, and division, but for things such as exponentiation, complex numbers, or trigonometric functions, you must use `numpy`.

4. Create a new cell by clicking the "+ Code" button at the top, then enter `np.power(2,3)` and, after pressing play, the answer should be displayed below the code block.

   a) What does `np.power` do?

   b) Does changing the order of the arguments (the numbers in parenthesis) change the output? If so, how does the position of the arguments relate to the result of the mathematical operation being calculated?

5. `numpy` also provides quick access to mathematical constants such as π and e, as well as trigonometric functions. Use `numpy` to calculate the cosine (cos) of π, as well as the sine (sin) of $8\pi/9$. Remember, all uses of `numpy` are preceded with `np.` as above in `np.power`.

   a) What code did you use to calculate the cosine of π?

   b) What value did you get for the sine of $8\pi/9$?

6. To plot values as a function of time in python, we will need to first create an array of those time values to serve as our x-axis. To do this, use the `np.arange` function, which has arguments (capitalized) listed below.

```
np.arange(START, STOP, STEP)
```

   Use this to create an array from 0 to 2 with a step size of 0.1. Pay special attention to the final value of the array. Is it what you expect it to be?

7. When working with time values, it is common to define the `STEP`, dt, as a variable since it will likely be used in other places in the code. Defining a variable allows you to use the same value in many places without having to remember that value, and it provides you with a method to quickly adjust that value if needed. To define a variable, follow the below template.

```
name = VALUE
```

   Note that it is customary in python to keep variable names lowercase and with multiple words separated by underscores.

   a) Write code to assign the value 0.1 to a variable named `dt`.

   b) Variables can represent things other than numbers. Write code which stores your time array from before as `time_array`, and use the `dt` value from part a as your `STEP`.

   c) Determine the length of the time array by using the `len(ARRAY)` command.

8. To display variables, use the `print` function as shown below. Print your time array and confirm that it displays the same values as before.

```
print(time_array)
```

9. The final concept which is necessary to implement the Fourier transform is a for-loop. A for-loop is used to repeat a certain block of code a fixed number of times. In python, a for-loop is written as shown below:

```
for i in range(5):
    print(i)
```

   Note that indentation is important, so the code which is repeated *must* be indented from the initial line of the for-loop. Type this code into a new cell in your Colab Notebook to see what it prints.

10. In a for-loop `range` is a function which specifies how many times the for-loop will iterate and `i` represents the "counter" of the for loop; it is the variable which is incremented by the number specified in `range`. `range` can take additional arguments as shown below.

```
for i in range(0, 10, 2):
    print(i)
```

    a) Run this code in your Colab Notebook in a new cell. What do each of the three values of the `range` function represent?

11. A common use for for-loops is to implement a Σ-sum. As a reminder, below is an example of a Σ-sum and how you would evaluate it.

$$\sum_{k=0}^{5} 2^k = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 = 63$$

    Notice how for each value of k from 0 to 5, that value is plugged into the expression, evaluated, and then added to the total. A for-loop can be used to do something similar. First you initialize a variable to zero *outside* the for-loop, then you define the lower- and upper-limits of the Σ-sum as the `START` and `STOP` parameters of the `range` function, and your expression goes inside the for loop. A for-loop which implements the same expression as above is shown below.

```
1 total = 0
2 for k in range(0, 6, 1):
3     total = total + 2**k
4 print(total)
```

    Remember that the `range` function's upper-limit is *not* inclusive, so to get values from 0 to 5, we need to set the upper-limit to 6. Also, all expressions involving variables (such as line `3` above) evaluate the right side first, then assign it to the variable on the left. This effectively takes the old value of `total`, adds 2^k to it, and then replaces `total` with that new value.

    a) Write code to implement the following Σ-sum and show that the answer is 99.

$$\sum_{k=0}^{10} (2k - 1) = 99$$

Part 4: Plotting a Cosine Wave

1. We will now use `matplotlib` to as well as our new knowledge of python to plot the cosine wave, $x(t) = 4\cos(2t)$, from **Part 1** in the range 0 to 2π. All of our values will be stored as variables to allow for easy modification.

```
start = 0
duration = 2 * np.pi
dt = 0.1
amplitude = ?
frequency = ?
phase = ?
```

    In python, the right side of a variable assignment always happens before the left is updated, so you can update a variable as follows:

```
var = var + 1
```

    This will increment `var` by 1 and then replace the old value of `var` with the incremented one. This concept will be helpful in updating our time array and cosine wave.

    a) Use `np.arange` as before to create the `time_array` for this cosine wave.

    b) In python, if you include an array in a mathematical operation, that operation is applied to all values of an array. Since our function takes the cosine of $2t$, multiply the time array by 2, then store this variable as `updated_time_array`.

    c) Now take the cosine of all the time values and store that as a variable named `cosine_wave`.

d) Finally, multiply the entire cosine wave by 4.

2. You should now have an array called `cosine_wave` which stores the discrete values of the function $x(t) = 4\cos(2t)$. Plot these values as shown below, replacing the arguments with your **original** time array and cosine wave:

```
1         plt.plot(X_VALUES, Y_VALUES)
2         plt.show()
```

   Does this plot have the same characteristics as you expected from $x(t)$ of **Part 1** above?