Lab assignment #5: Fourier Transforms

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)
TA & Marker: Alex Cabaj (alex.cabaj@mail.utoronto.ca)

Due Friday, 16 October 2021, 5 pm

Following lab 4, it appears that there is no need for a second room. Should it be incorrect and the room overflows, N.G. will open up the room he had for the previous labs (grab the link on the old question documents).

Room (also for office hour) Alex Cabaj and Nicolas Grisouard,
`https://gather.town/app/vmCKPZTSEKgGYlZI/PHY407AC1`
PWD: `tr0p0sph3r3`

General Advice

- **Work with a partner!**
- Today's topics revolve around applications of the Fourier transform, and their numerical computations by computers. Compared to other labs, coding proficiency will be less crucial, and we will instead focus on physical applications.
- Read this document and do its suggested readings to help with the pre-labs and labs.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Carefully check what you are supposed to hand in for grading in the section "Lab Instructions".
- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step.
- Test your code as you go, **not** when it is finished. The easiest way to test code is with `print('')` statements. Print out values that you set or calculate to make sure they are what you think they are.
- Practice modularity. It is the concept of breaking up your code into pieces that as independent as possible from each other. That way, if anything goes wrong, you can test each piece independently.
- One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
    """A header that  explains the function
    INPUT:
    argument [float] is the angle in rad
    OUTPUT:
    res [float] is twice the argument"""
    res = 2.*argument
    return res
```

Place them in a separate file called e.g. `MyFunctions.py`, and call and use them in your answer files with:

```
import MyFunctions as fl2  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl2.MyFunc(ZehValyou)
```

Computational background

Discrete Fourier Transforms This lab focusses on applications of the Fourier Transform. See §§ 7.1 and 7.2 of the textbook for general statements about the Discrete Fourier Transform (DFT). In particular, the DFT finds the coefficients c_k for a set of discrete function values y_n through

$$c_k = \sum_{n=1}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) \tag{1}$$

If the function if real, then the coefficients in the range $N/2 \rightarrow N$ are just the complex conjugates of the coefficients in the range $0 \rightarrow N/2$, i.e.,

$$c_{N-i} = c_i^* \tag{2}$$

This means we only have to calculate $\sim N/2$ coefficients for a real function with N values.

Since the Fourier coefficients are complex, we usually plot their absolute value vs. k to make a Fourier Transform plot. The c_k's with large magnitudes represent periodicities with k values which dominate the signal. The k value is related to the period n_{cyc} of the signal through:

$$\frac{2\pi k n_{cyc}}{N} = 2\pi \quad \Rightarrow \quad n_{cyc} = \frac{N}{k} \tag{3}$$

The DFT requires $O(N^2)$ evaluations of $\exp\left(-i\frac{2\pi kn}{N}\right)$. This is a lot and would not make the DFT useful for many operations. Luckily, the Fast Fourier Transform (FFT) is an algorithm that rearranges the DFT to make it much faster. When this faster algorithm is implemented, you require only $N\log_2 N$ evaluations of $\exp\left(-i\frac{2\pi kn}{N}\right)$. This is significantly fewer and hence significantly faster. Note that the FFT gives the same result as the DFT, it is not because it is faster that it is an approximation!

Although doable, you should never really code an FFT yourself. While a good FFT algorithm does not need to exceed a few lines, the number of mistakes one can do is dizzying. Instead, there are standard library functions in Python (as well as other programming languages) for the FFT.

Using the `.wav` sound file format The `.wav` format is a standard digital sound format used by computers. Typically if you click on a `.wav` format an audio player will open up and play the sound in the file. The particular sample that you will be processing in one of the questions is a stereo file with two channels, Channel 0 and Channel 1. The data format is `int16`, which is a 16 bit integer format. When you write to the new file you will need to write to that format. Here is some code to read and write that file — adapt it to your needs.

```
""" The scipy.io.wavfile allows you to read and write .wav files """
from scipy.io.wavfile import read, write
from numpy import empty
# read the data into two stereo channels
# sample is the sampling rate, data is the data in each channel,
#  dimensions [2, nsamples]
sample, data = read('input_file.wav')
# sample is the sampling frequency, 44100 Hz
# separate into channels
channel_0 = data[:, 0]
channel_1 = data[:, 1]
N_Points = len(channel_0)

# ... do work on the data...

# this creates an empty array data_out with the same shape as "data"
# (2 x N_Points) and the same type as "data" (int16)
data_out = empty(data.shape, dtype = data.dtype)
# fill data_out
data_out[:, 0] = channel_0_out
data_out[:, 1] = channel_1_out
write('output_file.wav', sample, data_filt)
```

In the code above, `sample` is the sampling rate of the data in Hz, which is typically 44100 Hz for audio data, and we have converted the data to the `int16` datatype in the two lines before the write statement.

Loading data from a file Recall from lab #2 that if you have data in a text file you can load the entire file into an array using a command like:

```
y = numpy.loadtxt("filename.txt")
```

where `filename.txt` is the name of your file.

If the data consists of `N` rows and `M` columns, then `y` will be an array with `N` rows and `M` columns. If you want to work with the first column of the array, you can refer to it as:

```
y[:,0]
```

The colon (`:`) means take all the rows, and using the index `0` in place of it means in the first column. Similarly, if you wanted the 5^{th} column you would use `y[:, 4]` or if you wanted the 3^{rd} row you would use `y[2, :]`. All of these options give you 1D arrays, essentially picking out the column or row of interest to you.

If you want to know the shape of the array (i.e., how many rows and columns) you can use the `shape` attribute. For example, for an $N \times M$ array,

```
print(y.shape)
```

would return (N, M). This is an array itself. If you wanted to just know the number of rows in y you could use

```
print(y.shape[0])
```

which picks out the first element of the shape array. Similarly, if you wanted to know the number of columns you would call the element indexed by `[1]`.

Numerical computations of Fourier transforms The page

https://numpy.org/doc/stable/reference/routines.fft.html

is full of useful functions. You might use more than one in this lab.

Physics background

A relativistic particle on a spring We revisit the relativistic particle on a spring of Lab 03, this time starting from the equations of motion. In relativistic mechanics, Newton's second law for one dimensional motion on the x axis for a particle experiencing a force $F(x,t)$ is

$$\dot{p} = F, \tag{4}$$

where the dot indicates a time derivative and the relativistic momentum is

$$p = \frac{m\dot{x}}{\sqrt{1-\dot{x}^2/c^2}}, \tag{5}$$

where m is the particle mass and c is the speed of light.

From eqns. (4)–(5), you can show (but don't need to show for this assignment) that the equation of motion for the relativistic spring is

$$m\ddot{x} = -kx\left(1-\frac{\dot{x}^2}{c^2}\right)^{3/2}. \tag{6}$$

To turn this into a system that you can integrate numerically, define the equivalent first order ODE system using the notation in eqn. (1) of Lab 01: define $u_1 \equiv x, u_2 \equiv \dot{x}$ and thus

$$\begin{aligned}\dot{u}_1 &= u_2,\\ \dot{u}_2 &= -\frac{k}{m}u_1\left(1-\frac{u_2^2}{c^2}\right)^{3/2}\end{aligned}$$

This is an interesting nonlinear ODE system that is hard to solve exactly but easy to solve numerically. The classical mass-on-spring system can be recovered for small velocities with $|u_2/c| \to 0$.

Signal analysis and filtering In this lab we will take a sample sound file, look at its Fourier spectrum, filter it for desired audio characteristics, and write the results to a new file in order to listen to the impact of our filtering. The sound file is in the `.wav` format described in the computational background. We will employ a very simple filtering method in which we will zero out the components that we want to suppress. In particular, we produce a low-pass filter such that all Fourier

coefficients above a cutoff frequency f_c will be set to zero: if $\hat{s}(f)$ represents the Fourier coefficients of a time series $s(t)$, the low-pass filter is here defined as

$$\hat{s}_{lp}(f) = \begin{cases} 0, & f > f_c, \\ \hat{s}(f), & f \leq f_c \end{cases} . \tag{7}$$

This is actually not a very good method for good quality sound filtering (you may see or may have seen better ones in PHY408), but will give you a flavour of how filtering might work.

Looking at atmospheric data In atmospheric physics, the *sea level pressure* (SLP) is the surface air pressure that would be found once the Earth's surface was raised or lowered to the geoid (the notional mean sea level). Daily maps of SLP show the location of low pressure centres (associated with cloudy weather and rain) and high pressure centres (associated with fair or sunny conditions). Such centres typically propagate eastward in the temperate latitude. SLP data (in the form of a kind of melded data and modeling product) are available from many research centres around the world. We have extracted some SLP data for you to analyze:

- Quantity: SLP with a mean value subtracted
- Units: hPa (hectopascals, i.e. 100 N m^{-2})
- Coordinates:
  - Latitude: 50°S.
  - Longitude: [0°, 360°], sampled every 2.5°.
  - Time: The first 120 days of 2015, sampled every day.
- Source: NCEP Reanalysis

To keep the analysis simple, we have extracted SLP around one latitude circle located in the Southern Hemisphere. Weather there is often observed to progress in coherent wave trains and you will use Fourier analysis to extract these wave trains. You will use the idea that SLP can be decomposed into a series of waves of the form

$$A(t)\cos(m\lambda + \phi(t)), \tag{8}$$

where m is the longitudinal wavenumber, λ is the longitude, and $A(t)$ and $\phi(t)$ are time-dependent phase factors. The dataset shows that different wavenumbers m are characterized by different propagation characteristics, as we'll describe in the corresponding problem. Some of these differences are predicted by theories in atmospheric dynamics.

Questions

1. **[20% Various examples of Fourier transforms**

   (a) Do Newman Exercise 7.1 (p. 303). Also plot the functions to make sure you have the right ones. For the Fourier coefficients, and in the three cases suggested, I find the `plt.bar()` way of plotting nicer than the `plt.plot()` function, which the question suggests when they recommend to emulate Figure 7.4.

HAND IN YOUR CODE, YOUR PLOTS, AND ANY EXPLANATORY NOTES.

(b) Do Newman Exercise 7.8 (p. 321). There are mentions of Exercise 5.19. You don't need to do it, but you can compare your results with the results obtained with one of the integration methods we saw in lectures 2 and 3 if you want.

HAND IN YOUR CODE, YOUR PLOTS, AND ANY EXPLANATORY NOTES.

2. **[30%] Fourier Transforms in the time domain**

(a) **Re-revisiting the relativistic spring:** The purpose of this exercise is to practice extracting a frequency spectrum from a time series you generate from one of your simulations. Do the following:

- Using the Euler-Cromer method (see Lab 01; you may re-use either your code or N.G.'s solution as a starting point, or you can start from scratch), simulate the relativistic spring system (cf. lab 03) in the case of zero initial velocity and for three initial positions:
  - $x_0 = 1$ m,
  - $x_0 = x_c$ and
  - $x_0 = 10x_c$,

  with x_c defined in Lab 03 as the initial stretch, a non-relativistic spring would need to have for its maximum velocity to be that of the speed of light, c. To get a good estimate of the spectrum, generate a long time series that contains several oscillations (at least 10 periods for each case). Make sure the time step is small enough to get stable solutions — even though Euler-Cromer is more stable than forward Euler, it can still be unstable in the relativistic regime for too long a time step.

HAND IN YOUR PLOTS FOR THESE SIMULATIONS AND BRIEF DESCRIPTIONS OF THE PARAMETERS YOU USED, AS RELEVANT.

- Then find the Fourier transform of the solutions for position $x(t)$. This will give Fourier components which can be written $\hat{x}(\omega)$, where the hat indicates a Fourier component and ω is the angular frequency. To plot the three cases on the same plot, given that the amplitude of the oscillation is very different in each case, plot the scaled quantity, $|\hat{x}(\omega)|/|\hat{x}(\omega)|_{\max}$, where the max indicates the maximum value of the amplitude. This quantity has a maximum value of 1 for each spectrum.

  Describe the differences in the spectrum between the three cases. Do they make sense to you?

- Now do the same for velocity $v(t)$ instead of position (i.e., plot the spectrum of the velocity Fourier coefficients). Are there any qualitative differences between the spectra for $v(t)$ and for $x(t)$?

- Quantitatively compare these results obtained from the simulation to the predicted period of the oscillations using Gaussian quadrature integration, as in Lab 03, Equation (7) (see Lab03 Q2). In other words, you now have two estimates of the characteristic frequencies, one obtained from the spectrum, and another obtained from Equation (7) of Lab03. The simplest way to do this is to plot a vertical line at the

location of the frequency $1/T$, where T is the period obtained from Equation (7) of Lab03.

FOR ALL BULLET POINTS SINCE THE LAST INSTRUCTIONS, HAND IN PLOTS, WRITTEN ANSWERS, AND CODE.

(b) **Filtering an audio recording.**

In this exercise I will ask you to low pass filter the sound in the sound file provided with this exercise. Do the following:

- Play the sound file `GraviteaTime.wav` to make sure your computer's audio can process it.[1] If you are having trouble with this please seek help on Discord or Piazza.
- Now, adapting the code provided in the background material, produce a plot of the data in the file as a function of time. Produce one plot for each channel, and mark the time axis in seconds, accounting for the sampling rate given by `sample` in Hz (recall $\omega = 2\pi f$, with f the frequency) as described in the background material.

HAND IN YOUR PLOT.

- Now focus down this plot to a small segment of the total time series about 20-50 ms long. This will let you better see the impact of filtering. You do not need to hand in this particular plot.
- Now implement a filter in which all frequencies greater than 880 Hz are set to zero. Do this by Fourier transforming each signal to the frequency domain, setting the right coefficients to zero, then Fourier transforming the result back to the time domain. Plot the amplitude of the original Fourier coefficients, the amplitude of the filtered Fourier coefficients, the original time series over the short interval described above, and the filtered time series for this interval. Do this for both channels and use `subplot` to reduce the number of separate figures. The filtered time series should be a smoothed version of the original time series.

HAND IN YOUR PLOTS.

- Now output the resulting time series, for each channel, into a new `.wav` file, as shown in the background material. For our reference, call the file `GraviteaTime_lpf.wav`. You should hear a bass-heavy version of the tune.[2] Your marker will get to listen to your creation!

HAND IN YOUR FILE.

3. **[20%] Analysis of sea level pressure in the Southern Hemisphere for summer 2015.**

Your job will be to Fourier-decompose in the longitudinal direction the sea level pressure at 50°S, which is provided in the file `SLP.txt`. The files `lon.txt` and `times.txt` provide the longitudes λ (in degrees) and the times (in days, starting from January 1, 2015) for this data.

[1] The music file is from Joel Franklin's *Computational Methods for Physics*. Once you get to a point where you can't get it out of your head, you may share your best rendition on TikTok.

[2] We are in good company: https://youtu.be/KsfxIgOtROk

This dataset can be read and initially plotted by the following commands, as long as all the files are in the same folder as your code:

```
from numpy import loadtxt
from matplotlib.pyplot import contourf, xlabel, ylabel, title, colorbar

SLP = loadtxt('SLP.txt')
Longitude = loadtxt('lon.txt')
Times = loadtxt('times.txt')

contourf(Longitude, Times, SLP)
xlabel('longitude(degrees)')
ylabel('days since Jan. 1 2015')
title('SLP anomaly (hPa)')
colorbar()
```

The plot you obtain will show the time evolution of the SLP at this latitude for the first 120 days of 2015.

(a) Extract the component of SLP corresponding to Fourier wavenumber $m = 3$ and to $m = 5$ for this data. Create filled contour plots in the time-longitude domain for the data thus extracted, and hand in these plots. What are some important qualitative characteristics of these plots? The theory of atmospheric wave propagation suggests that wave disturbances can propagate in a dispersive manner (i.e., the phase speed depends on the wavelength, in this case shorter waves travelling eastward faster than longer waves). Is this roughly consistent with what you see?

HAND IN THESE PLOTS.

(b) Now estimate the speed and direction of propagation for the case $m = 5$. To get the speed in $m\,s^{-1}$, the circumference of the latitude is given by $2\pi r_E \cos(\theta)$, where θ is latitude and r_E is the radius of the Earth. Note that these results will be different for other times of year and other years. Your estimate can be graphical or more quantitative — just state which method you use.[3]

HAND IN CODE, ANY PLOT YOU END UP USING, AND YOUR WRITTEN ANSWERS.

4. **[30%] Newman 7.9: Image deconvolution (p. 322).**

Note that I find Newman a bit misleading. The technique he describes only works for pictures that are out-of-focus, in the sense that all of the information is present in the image, just scrambled. This method cannot fix problems related to resolution limits. I.e., if the details you are trying to retrieve are smaller than what a pixel of the picture, you are stuff out of luck. So, when you see forensic analysts making up a face from 3×3 pixels, that's fiction.

HAND IN CODE AND THE THREE FIGURES.

Hints:

- *I recommend using "imshow" to plot the data with a gray colormap.*
- *There is some important information at the top of page 326. Make sure to turn the page!*

[3] This question focuses on the largest differences between the Fourier components. You can explore other wavenumbers m and you will find that the behaviour does not follow a simply predicted trend across scales.

- *You don't have to do part (d).*