PHY407 Formal Lab Report 3

Fall 2024

Computational Background

Discrete Fourier Transforms This lab focuses 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 $N/2 \to N$ are just the complex conjugates of the coefficients in the range $0 \to N/2$, i.e., $c_{N-i} = c_i^*$. 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: $2\pi k n_{cyc} = 2\pi N$. Therefore, $n_{cyc} = \frac{N}{k}$.

The Fast Fourier Transform (FFT) is an algorithm that rearranges the DFT to make it much faster; it gives the same result as the DFT.

Numerical computations of Fourier transforms 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. `https://numpy.org/doc/stable/reference/routines.fft.html` is full of useful functions, which you may use in this lab.

One common problem in using FFTs to analyze scientific data is finding the *correct* amplitude and phase. For example, when you Fourier transform a wave with amplitude of 2 units, you need the Fourier transform to return a wave with an amplitude of 2 unit (not 2π units, or $2/N$ units, or some other scalar multiple). As there is often ambiguity about which scale factor is used by which built-in FFT routine it is a good idea to check the results with simple functions. For example, to check the output of the `fft` function against the `ifft` function make a sine function and FFT that. Compare your *known* input parameters (amplitude, phase) against the output of the FFT.

```
import numpy as np
def cosine_wave(time, amplitude, period, phase):
    """Generate a sine wave with known amplitude, period, and phase"""
    return amplitude * np.cos((time/period + phase) * 2 * np.pi)
t = np.arange(0, 100, .1)
y = cosine_wave(t, 2, 20, 0.)

fft1 = np.fft.fft(y)
fft2 = np.fft.ifft(y)
fft3 = np.fft.rfft(y)
fft4 = np.fft.irfft(y)
```

```
amp1 = np.abs(fft1)
amp2 = np.abs(fft2)
amp3 = np.abs(fft3)
amp4 = np.abs(fft4)
# Are any of the following numbers what you expect?
print(amp1.max(), amp2.max())
print(amp3.max(), amp4.max())
```

You should find that one FFT command includes a scale factor equal to the length of the array, while the other doesn't. There are options to the FFT function that allow you to change this behaviour. **If you change it, make sure you are consistently using the options**. You will also find that the "amplitude" only includes half of the input signal. This happens because there are positive and negative phase components and each carries some of the signal – in this case half. The distinction between the positive and negative components can sometimes be ignored for one dimensional FFTs, in which case the rFFT functions are useful. For two dimensional FFTs the two components contain information about the *direction* the signal is travelling.

Finding the phase can also be ambiguous. The Fourier transform in `numpy.fft` returns a complex number $r_k + ij_k$, and the phase can be found from trigonometry using the `numpy.arctan2` function. You should experiment to make sure you understand what phase is being returned by numpy.fft .

Using the `.wav` sound file format The `.wav` format is a standard digital sound format. The scipy.io.wavfile allows you to read and write files in this format. Typically if you click on a `.wav` file on a computer, 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 16-bit integers (`int16`). 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.

```
from scipy.io.wavfile import read, write
from numpy import empty
# read the data into two stereo channels, labelled 0 and 1
# sample is the sampling rate (typically 44100 Hz)
# data is the data in each channel, dimensions [2, nsamples]
sample, data = read('input_file.wav')
channel_0 = data[:, 0]
channel_1 = data[:, 1]
nsamples = len(channel_0)
# ... work with the data to create new arrays channel_0_out and channel_1_out,
# each of length nsamples, containing values convertible to int16
# create & fill empty output array data_out with the same shape and datatype as "data"
data_out = empty(data.shape, dtype = data.dtype)
data_out[:, 0] = channel_0_out
data_out[:, 1] = channel_1_out
# write the output array to a new .wav file
write('output_file.wav', sample, data_out)
```

Drawing contour plots The `contour` and `contourf` methods in `matplotlib.pyplot` , which you should use in this lab, are documented at: `https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.contourf.html` and `https://matplotlib.org/stable/gallery/images_contours_and_fields/contourf_demo.html`.

Physics background

Signal analysis and filtering In this lab we will take a sample file, look at its Fourier spectrum, filter it for desired characteristics, and write the results to a new file in order to observe the impact of our filtering. 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

$$\hat{s}_{\ell p}(f) = \begin{cases} 0, & f > f_c, \\ \hat{s}(f), & f \leq f_c \end{cases} . \tag{2}$$

This is a common method for reducing noise in a dataset. It's actually not the best method for high-quality sound filtering, but will give you a flavour of how filtering works in general.

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; here we use data from NCEP Reanalysis. We have extracted some SLP data for you to analyze, where the mean SLP value has been subtracted out, and the units are hPa (hectopascals, i.e. 100 $\mathrm{N/m^2}$. The coordinates are:

- Latitude: 50°S.
- Longitude: $[0°, 360°]$, sampled every 2.5°.
- Time: The first 120 days of 2015, sampled every day.

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

$$A(t)\cos(m\lambda + \phi(t)), \tag{3}$$

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. **Audio filtering** In this exercise you will apply a "low pass filter" to the sound file provided. Play the sound file `GraviteaTime.wav` to make sure your computer's audio can process it.[1]

   (a) 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. **Submit your plots.**

   (b) 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 appropriate coefficients to zero, then Fourier transforming the result back to the time domain. For both channels: plot the amplitude of the original Fourier coefficients, the amplitude of the filtered Fourier coefficients, the original

[1] The music file is from Joel Franklin's *Computational Methods for Physics*.

time series, and the filtered time series. *(Hint: you may want to use* `subplot` *to reduce the number of separate figures.)* The filtered time series should be a smoothed version of the original time series. **Submit your plots.**

(c) Plot the original time series and the filtered time series again, but over just a small segment of the total time series (about 30 ms long, starting at any time of your choice) to better show the smoothing impact of the filtering. **Submit your plots.**

(d) Output the filtered time series you obtained in Part b, for both channels, into a new `.wav` file, as shown in the background material. You should hear a bass-heavy version of the tune. Name the new file `GraviteaTime_filtered.wav`. Your marker will get to listen to your creation! **Submit your .wav file.**

2. **Getting rich on the stock market** If your prof's research budget is running short, she might need to play the stock market in order to fund her next dark matter experiment. The file `sp500.csv` contains the opening value (second column) for each business day (first column) from late 2014 until 2019 of the S&P 500 stock index (a measure of the growth of the largest 500 companies in the United States).

   (a) Write code to load the second column of data from `sp500c.csv` (opening value), and plot it against "business day number" starting at 0. (The data in the file only exists for "business days", so it skips weekends and holidays; therefore if you plot the "actual day number", there will be gaps that mess up your analysis.) **Submit your code.**

   (b) Write code to calculate the coefficients of the FFT of the data (opening value vs business day) using the `rfft` function from `numpy.fft`. Test that inverting the results with `irfft` returns the original data. **Submit your code, and explanation and output of whatever test you implemented.**

   (c) Suppose we want to analyze long-term trends in the market, by *removing* any variation with a period of 6 months or shorter. Write code to set some of the Fourier coefficients (figure out yourself how many, and which ones) to zero, and perform the inverse FFT. This is another example of a "low–pass filter". Plot the result on the same graph as the original (opening value vs business day) data; adjust the linestyle as needed to make the two lines clear, and add a legend. Explain the shape of the filtered data (the first and last datapoints may be weird, but you can ignore these.) **Submit your code, graph, and explanation.**

3. **Analysis of sea level pressure** 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. This dataset can be read by the following commands, as long as all the files are in the same folder as your code:

```
from numpy import loadtxt

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

   (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. **Submit your plots.**

   (b) 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? **Submit your written answers.**