PHY407-Lab05

Due October 14 2022

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.,

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

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: $2\pi k n_{cyc} = 2\pi N$. Therefore, $n_{cyc} = \frac{N}{k}$.

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!

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. The page

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

is full of useful functions. You may use them in this lab.

Using the `.wav` sound file format The `.wav` format is a standard digital sound 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 `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.

```python
""" 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.

Drawing contour plots The `contour` and `contourf` methods in `matplotlib.pyplot` 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

Relativistic particle on a spring, again 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

$$\dot{p} = F, \tag{3}$$

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

$$p = \frac{m\dot{x}}{\sqrt{1-\dot{x}^2/c^2}}, \tag{4}$$

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

From eqns. (3)–(4), you can show (but don't need to show for this assignment) that the equation of motion for the relativistic spring is

$$m\ddot{x} = -kx\left(1-\frac{\dot{x}^2}{c^2}\right)^{3/2}. \tag{5}$$

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

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

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

$$\hat{s}_{\ell p}(f) = \left\{ \begin{array}{cc} 0, & f > f_c, \\ \hat{s}(f), & f \leq f_c \end{array} \right. . \tag{6}$$

This is actually not a very good 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. We have extracted some SLP data for you to analyze:

- Quantity: SLP with a mean value subtracted
- Units: hPa (hectopascals, i.e. $100\ \mathrm{N\,m^{-2}}$)
- 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

$$A(t)\cos(m\lambda + \phi(t)), \tag{7}$$

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. **[35%] Revisiting the relativistic spring**

   The purpose of this exercise is to practice extracting a frequency spectrum from a time series generated from a simulation.

   (a) Using the Euler-Cromer method (see Lab 01; you may either re-use your code as a starting point, or start from scratch), simulate the relativistic spring system from 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.

Submit your pseudocode, code, plots, and brief descriptions of the parameters you used.

(b) 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, $|\hat{x}(\omega)|/|\hat{x}(\omega)|_{\text{max}}$, 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?

Submit your pseudocode, code, plots, and written answers.

(c) 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.

Submit your pseudocode, code, plots, and written answers.

(d) **Optional** Plot the spectrum of the velocity Fourier coefficients, i.e. do part (b) for velocity $v(t)$ instead of position $x(t)$. Are there any qualitative differences between the spectra for $v(t)$ and for $x(t)$?

2. **[30%] Audio filtering** In this exercise you will apply a "low pass filter" to the sound file provided.

(a) Play the sound file `GraviteaTime.wav` to make sure your computer's audio can process it.[1]

Nothing to submit.

(b) 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 code, pseudocode, and plot.

(c) **Optional** 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.

(d) 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 time series over the short interval described above, and the filtered time series for this interval. *(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 code, pseudocode, and plots.

(e) Now output the resulting time series, for each channel, into a new `.wav` file, as shown in the background material. You should hear a bass-heavy version of the tune. Name all the file `GraviteaTime_lpf.wav`.

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

Your marker will get to listen to your creation!

Submit the .wav file.

3. [**35%**] **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 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()
```

(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 pseudocode, code, and 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.