Computational Lab 5 : Fourier Transforms

Due October 11th, 2019 (@ 17:00)

- 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.
- 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 form 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
  ```

  For example, in this lab, you will probably use Fourier transforms more than once. You may want to write generic functions for these rules (or use the piece of code, provided by the textbook online resources), place them in a separate file called e.g. `functions_lab02.py`, and call and use them in your answer files with:

  ```
  import functions_lab02 as fl2  # make sure file is in same folder
  ZehValyou = 4.
  ZehDubble = fl2.MyFunc(ZehValyou)
  ```

Physics Background

S&P 500 and Dow Jones index: An entirely optional introduction can be found on the NPR Planet Money podcast: https://www.npr.org/sections/money/2017/01/04/508261371/episode-443-dont-believe-the-hype

Surface level air pressure: The atmosphere of Mars, much like Earth and Venus, is heated daily by the Sun as it passes overhead, but the heating is not a simple function of time. While there is a smooth increase and decrease in heating during the day (approximating a sine function with local noon at the peak) the night–time 'heating' is effectively constant — once the Sun has set, the same amount of sunlight reaches the ground until morning (none). As a result of this heating and cooling pattern, the surface temperature on Mars doesn't cycle in a single 24 hour sine function, but as a series of sine functions with periods of 24,12,8,4,3 hours and so on, all integer factors of 24 hours. Similarly the surface pressure on Mars goes through a similar daily cycle composed of waves of shorter periods.

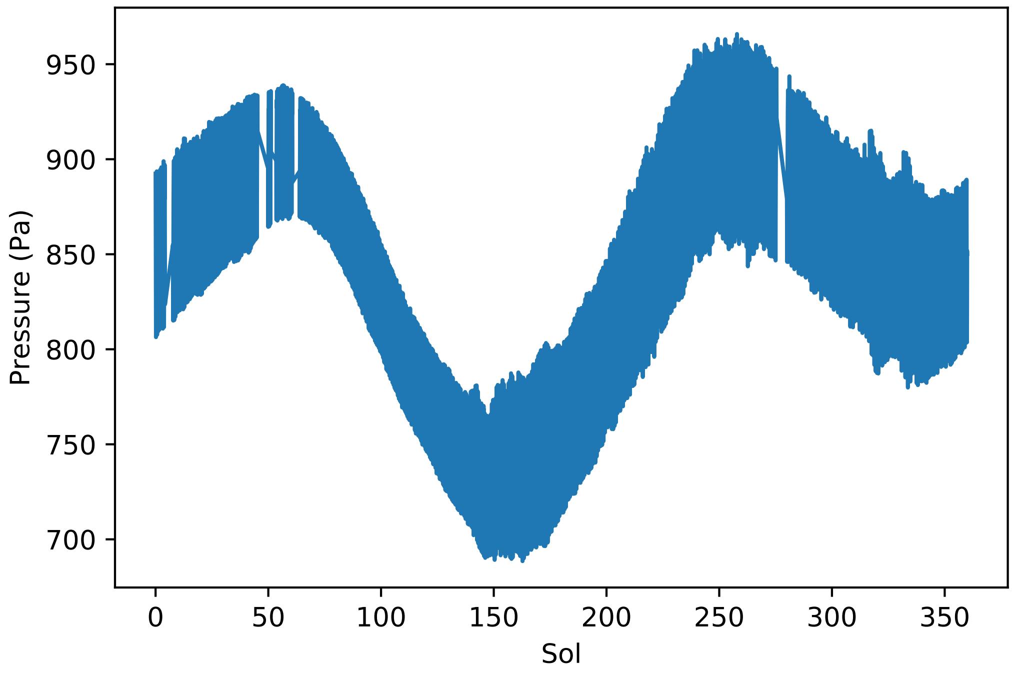

Figure 1: Surface pressure (Pa) measured by the Curiosity Rover on Mars for 1 year.

As an added complication, the air on Mars is 95% carbon dioxide, a gas that freezes at temperatures that are found on Mars during winter. A third of the Martian atmosphere literally freezes out of the sky twice a year. The annual pressure cycle and daily pressure cycle measured from the Mars Science Laboratory Rover Environmental Monitoring Stations (a.k.a. MSL–REMS) is shown in figure 1

Time on Mars The data file you are given contains a measurement 24 times each Mars day. Since the Mars day is slightly longer than an Earth day (by 2.7%) it is usually called a Sol, and 1/24 of a Sol is an "hour" (even though this hour has almost 100 extra seconds). In the dataset the times are given in days (since some relatively arbitrary time) such that the integers times are local midnight. Sometimes, as in the figure, the Mars year is broken up into Heliocentric longitude, measured as the number of degrees around the orbit since the start of the Martian year at $L_s = 0$.

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 $N/2 \to N$ 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:

$$\left(\frac{2\pi k n_{cyc}}{N}\right) = 2\pi \Rightarrow n_{cyc} = \frac{N}{k} \tag{3}$$

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.
Although doable, you should never really code an FFT yourself, because the amount of things to keep track of is dizzying. Instead, there are standard library functions in python (as well as other programming languages) for the FFT.

Numerical computations of Fourier transforms The page
https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.fft.html#module-numpy.fft
is full of useful functions. You might use more than one in this lab.

ifft or fft?: 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 FFT routine it is necessary 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.

```python
import numpy as np
def cosine_wave(time, amplitude, period, phase):
    """Generate a sine wave with known amplitude, period, and phase"""
    # What do you mean by "phase" exactly?
    return amplitude * np.cos((time/period + phase) * 2 * np.pi)

t = np.arange(0, 100, .1)
y = cosine_wave(t, 2, 20, 0.)

#plt.plot(t, y)
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)

print(amp1.max(), amp2.max())
print(amp3.max(), amp4.max()) # Are any of these numbers what you expect?
```

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 $r_k + i j_k$, 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 .

Gradients using Fourier transforms: Using the discrete method from earlier labs you calculated the gradient as

$$\frac{dz}{dx} \approx \frac{z_{i+1} - z_i}{\delta x}. \tag{4}$$

Fourier transforms can also be used to calculated gradients in data by calculating the analytical differential of the Fourier coefficient and inverting the Fourier transform to complete the process. For example for a given function $z(x)$

$$z(x) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i \frac{2\pi kx}{L}\right), \tag{5}$$

(equation 7.5 in the text book). The differential is then

$$\begin{aligned} \frac{dz(x)}{dx} &= \frac{d}{dx}\left(\sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i\frac{2\pi kx}{L}\right)\right), & (6)\\ &= \sum_{k=-\infty}^{\infty} \gamma_k \frac{d}{dx}\left(\exp\left(i\frac{2\pi kx}{L}\right)\right), & (7)\\ &= \sum_{k=-\infty}^{\infty} \gamma_k \frac{i2\pi k}{L} \exp\left(i\frac{2\pi kx}{L}\right). & (8) \end{aligned}$$

Computationally, this means you can calculate the gradient of a field in with the following pseudo–code

```
ft_data = fft(data)
diff_coeff = i * 2 * pi * k_coeff #where k_coeff is a suitably defined np.arange
ft_diff = ft_data * diff_coeff
grad = ifft(ft_diff)
```

Padding data: Given an array with N points, it is sometimes necessary to increase the length of the array to M points with zeros padded on either side of the real data. One way to do this is

```
N, M = 10, 20  # example data
real_data = np.arange(N)  # example data
new_data = np.zeros(M)
# The padding
pad = (M-N)//2
new_data[pad:pad+N] = real_data
```

Fourier algorithm optimization: The Fast Fourier transform was first implemented for arrays that are a power of 2 in length (2,4,8,16,32,etc.), because the algorithm involves repeatedly halving the dataset into odd and even components. Later optimizations added fast code for powers of small primes, including 3,5,7. The optimizations for higher primes is not used. As a result, FFT codes are typically fast for numbers that can be factored into $2^l 3^m 5^n$ for integer l,m,n and slower for other numbers. The further a number is from a product of these primes, the worse the performance.

Questions

1. **[30%]** Fourier filtering and smoothing

   Modified Exercise 7.4 from the textbook

   The file `sp500.csv` contains the closing value for each business day 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 a program to load the data from `sp500c.csv` and plot the data on a graph. How should you handle the lack of data on the weekends and holidays?

   SUBMIT WITH LATER PARTS

(b) Calculate the coefficients of the discrete/fast Fourier transform of the data using the `rfft` function from `numpy.fft` and make sure that inverting with `irfft` returns you to the original data.

NOTHING TO SUBMIT

(c) Set all but the first 10% of the Fourier coefficients to zero, inverse the FFT and plot the new dataset on the same graph as the original data. You can adjust the linestyle as needed to make the two lines clear, and add a legend. What happened to the processed dataset when you set Fourier coefficients to zero?

SUBMIT THE PLOT, AND WRITTEN ANSWERS TO THE QUESTION

(d) Modify the program to *remove* any variation with a period of 6 months or shorter but keep all other variability. This is a low–pass filter, keeping all frequencies *lower* that a threshold. Explain the shape of the filtered data, including the interior shape and behaviour at the boundary.

SUBMIT THE PLOT AND CODE USED TO MODIFY THE DATASET AS NEEDED, AND THE WRITTEN ANSWER

(e) Modify the program to *keep* variations faster than 1 week. This is a high–pass filter, keeping all frequencies above a minimum frequency. Plot the filtered data, and plot the histogram of the filtered data. What property does this histogram suggest for the S&P 500 index ? Describe the behaviour of the stock market over short time periods.

SUBMIT THE PLOT AND CODE USED TO MODIFY THE DATASET AS NEEDED. SUBMIT THE WRITTEN DESCRIPTION

2. **[35%] Analysing air pressure measurements on Mars** This question uses the data file *msl_rems.csv* that includes surface air pressure measurements from the Mars Science Laboratory over 2 Mars years from 2014 to 2018. The data has been preprocessed for you to make this question feasible, but the data are essentially measurements from a nuclear–powered semi–autonomous rover driving up a mountain within a crater on Mars!

   The data file contains five pressure columns, `A`,`B`,`C`,`D`, and `E`. Datasets `A` through `D` are modified in various ways to let you explore the data, dataset `E` is the cleaned–up pressure data.

   (a) Write a function that will take a data series with constant time intervals and will perform a Fourier analysis on this data. The function should calculate the FFT of the dataset and return the amplitude, period, and phase of waves in the dataset. Test this function on a simple dataset by generating a cosine wave of known amplitude, period, and phase.

   SUBMIT YOUR CODE AND AN EXAMPLE TESTING THE CODE TO SHOW CORRECTNESS

   (b) Perform a Fourier analysis of the data found in pressure field `A`. Plot the amplitude and phase as a function of period (a log period scale is needed) and find the amplitude and phase of the dominant wave. You should see a single peak in the FFT at a period of 1 Sol. What is the amplitude of this peak. It may be helpful to plot the data to make sure you see a wave.

   NOTHING TO SUBMIT

   (c) Repeat the Fourier analysis for pressure field `B`. This dataset has 3 peaks. Find the amplitude, phase, and period of these waves. What does your phase tell you about the location of the wave in time?

   SUBMIT THE AMPLITUDE, PHASE, AND PERIOD OF THE WAVES.

   (d) Dataset `C` contains cleaned pressure data[1] using only the diurnal (once per day) and sub–diurnal waves. Find the periods and amplitudes of the waves.

   SUBMIT THE PLOTS AND WAVE PROPERTIES

   (e) Dataset `D` contains a less clean pressure data that `C` but is essentially the same. Repeat the Fourier analysis for this data. No artifical noise has been added to this data. Can you extract the same information from dataset `D` as dataset `C`? If you plot the time series of dataset `D` you may notice that the amplitude of the fast waves changes over the dataset (how do you know?). What did the FFT do with this information? How

[1] some signals have been removed, gaps have been filled

could you extract this new information from the dataset? (you don't need to do this, only explain how you could do it).

This is where real data becomes a problem for you. Naively sorting the data and looking at the largest amplitudes will probably give you 4 waves with the same period. You need to find waves with different periods.

SUBMIT THE PLOTS AND WAVE PROPERTIES. ALSO SUBMIT THE EXPLANATION AND ANY FURTHER WAVE ANALYSIS YOU PERFORM.

(f) Dataset `E` contains the real pressure data including the seasonal pressure cycle, short term 'weather' cycles, and diurnal tides (it is dataset `D` with more information retained). Find the 4 largest waves with periods longer than 100 sols, these are the seasonal cycles, responding the summer and winter on Mars. Find the 4 largest waves with periods shorter than 1 sols (or maybe just above 1 sol), these are the tides, responding to the rotation of Mars and the heating from the Sun.

For the two sets of waves you found, what relationship do the periods have with each other? For the waves with a period of less than 1 sol, at what local time do they peak?

SUBMIT THE PLOTS AND WAVE PROPERTIES.

3. **[35%] Gradients using Fourier transforms and profiling FFT times**

(a) For this question you will be re–using data and code from a previous lab. First copy the code you wrote for that lab into a python file (e.g. called lab03_mapping.py) so that you can `import` you code into the new program instead of copying and pasting code, **which can lead to bugs**.

SUBMIT THE PYTHON FILE INCLUDING THE DOCUMENTED FUNCTION YOU WILL BE USING FROM LAB 3

(b) Load the Hawa'ii island data as in lab 2, and calculate the gradients using a discrete method. You should not need to write new code for this step. Plot the gradients you calculated.

NOTHING TO SUBMIT

(c) Write a function to calculate the gradient in a 2D dataset along either axis using a Fourier transform to calculate the first derivative. Use this function to calculate the longitudinal ($\frac{dw}{dx}$) and latitudinal ($\frac{dw}{dx}$) derivatives, as before. Are the gradients the same as calculated with the discrete method, if not, what caused the difference?

SUBMIT YOUR CODE, AND THE PLOTS OF THE GRADIENTS. SUBMIT YOUR WRITTEN ANSWERS TO THE COMPARISON OF FOURIER AGAINST THE DISCRETE METHOD

(d) If your code is working as expected, you might have found that the Fourier transform is slow. Write a function that takes the original data and adds padding along the edge of the original data before performing the gradient calculation, and correctly reverses the process to return only the original size. Time this code for padding sizes from 1 to 100 in steps of 1. Plot the timing of your function against the total array size. What is causing the variation in timing?

SUBMIT THE TIMING PLOT, AND AN EXPLANATION OF THE TIMING VARIATION

Question:	1	2	3	Total
Points:	30	35	35	100