Lab assignment #5: Computing Fourier Transforms

Nico Grisouard (Instructor), Alex Cabaj and Vince Pascuzzi (TAs),
Haruki Hirasawa (Marker)

Due Friday, 12 October 2018, 5 pm

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:

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

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. 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, 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.

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 " : " means take all the rows, the "`0`" means in the first column. Similarly, if you wanted the 5th column you would use `y[:,4]` or if you wanted the 3rd 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 `[1]` element.

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.

Image deconvolution is one example of the use of cross-correlation techniques to get info from data that is jumbled up (I suppose the technical term is "convoluted").

Pages 322-325 take you through the mathematics of how it works, so you should read through that to get an idea of what is going on, but to do the lab, you really only need the result near the bottom of page 324:

$$\tilde{b}_{kl} = KL\tilde{a}_{kl}\tilde{f}_{kl}, \tag{4}$$

where $\tilde{b}_{kl}$ are the Fourier coefficients of the convoluted image, $\tilde{a}_{kl}$ are the Fourier coefficients of the pure image, $\tilde{f}_{kl}$ are the Fourier coefficients of the point spread function that convoluted the image and K and L are the length and width of the image.

So if you take the Fourier transform of the convoluted image and divide it by the Fourier transform of the point spread function, then you get the Fourier transform of the pure image. You can then inverse-transform the pure image Fourier Transform to get the pure image.

Shifting the Gaussian Here is some code to produce the gaussian image in the question bout image deconvolution. In the code below, '`rows`' is the number of rows in the data and '`cols`' is the number of columns. These will have to be defined beforehand from the shape of the blurred image.

```
for i in range(rows):
    ip = i
    if ip > rows/2:
        ip -= rows  # bottom half of rows moved to negative values
    for j in range(cols):
        jp = j
        if jp > cols/2:
            jp -= cols  # right half of columns moved to negative values

        gauss[i, j] = exp(-(ip**2 + jp**2)/(2.*sigma**2))  # compute gaussian
```

Physics background

The sunspot cycle This is for your information mostly, but sunspots are cool, and a former colleague of mine (Sabine Stanley) wrote this interesting description.

- Sunspots appear as dark patches on the surface of the sun. They are caused by strong magnetic activity which inhibits convection in the area, hence making the region appear darker.
- Sunspots demonstrate some periodic behaviour that is referred to as the sunspot cycle. During one cycle, the location, number and polarity of sunspots varies. Plots of sunspot fractional area create the infamous "butterfly diagram" (since each cycle looks like the wings of a butterfly), shown in fig. 1.
- The sunspot cycle has lasted for over 400 years (it was observed first by Galileo in 1610).
- I show an image of the sun with sunspots and some close ups of sunspots in fig. 2.

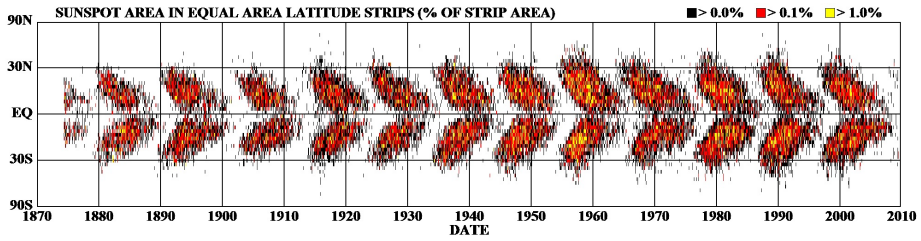

Figure 1: Butterfly diagram of the sunspot cycle.

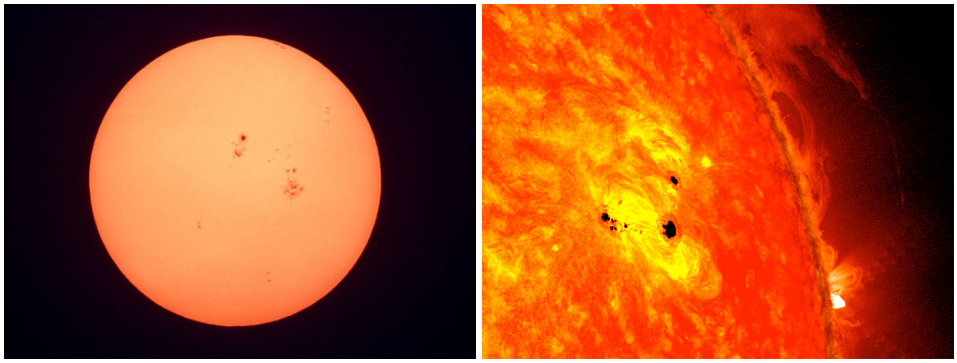

Figure 2: Left: Sun spots in the visible from NASA. Right: Zoom in of sunspots: The bottom two black spots on the sun, known as sunspots, appeared quickly over the course of Feb. 19-20, 2013. These two sunspots are part of the same system and are over six Earths across. This image combines images from two instruments on NASA's Solar Dynamics Observatory (SDO): the Helioseismic and Magnetic Imager (HMI), which takes pictures in visible light that show sunspots and the Advanced Imaging Assembly (AIA), which took an image in the 304 Angstrom wavelength showing the lower atmosphere of the sun, which is colorized in red. Credit: NASA/SDO/AIA/HMI/Goddard Space Flight Center.

Musical frequencies Like many sensory experiences (e.g. light or sound intensity), we perceive frequencies logarithmically. That is, if three successive frequencies are separated two-by-two by the same multiplicative factor (e.g., 50 Hz, 100 Hz, 200 Hz), then we will perceive them as equally "different" two-by-two. On the other hand, in a linear frequency progression (e.g., 50 Hz, 100 Hz, 150 Hz), we will have more difficulty telling the last two frequencies apart than the first two frequencies.

As a result[1], most civilizations, which combined to form our current musical system, have used instruments that are tuned such that the frequency interval between two successive notes has the same ratio (which is called an equal temperament).

In the West, the most widely-used system has been the twelve-tone equal temperament for quite a while now. In this temperament, all frequencies are tuned based on $A_4 = 440$ Hz. From this frequency, generating intervals are defined as octaves. Each octave contains all the frequencies between one frequency and twice its frequency (i.e., between $A_4 = 440$ Hz and $A_5 = 880$ Hz). Then, each octave is split into 12 intervals, in a logarithmic progression. E.g., after A_4 comes $A_4\sharp/B_4\flat = 440 \times 2^{1/12} \approx 466.164$ Hz, and after that, $B_4 = 440 \times 2^{2/12} \approx 493.883$ Hz.

For most musical instruments, the musician makes one or many oscillator(s) resonate, and try to excite all of the frequencies allowed by that oscillator. This is actually not a trivial task to make music like this! Indeed, the most basic instrument you could think of, that is, the vibrating string, has a frequency spectrum that is $\nu_n = (n/L)\sqrt{T/\mu}$, where $n \in \mathbb{N}^*$, T is the tension in the string and μ is the mass per unit length ($\sqrt{T/\mu}$ is the phase speed of the wave). This is a linear progression of the frequency, and most simple oscillators (strings, tubes, boxes) would exhibit such a progression in the resonance frequency spectrum. In order to reproduce the logarithmic progression of an equal temperament, one then needs to use various contraptions, like many strings (piano), ways to adjust the length of the string (guitar), puncture holes in a tube (flute) or add even weirder things like mouthpieces, bells, and rely on non-linear effects to create extra frequencies (trumpet).

My mention of the trumpet is not innocent: you will see that for the trumpet to create one note, it actually needs to excite on a whole spectrum of frequencies.

Questions

1. **[60%] Basic applications of the Fourier transform**

   (a) Newman 7.2.

   HAND IN PLOTS AND WRITTEN ANSWERS TO THE QUESTIONS. THE CODE WILL BE VERY SIMILAR TO THAT OF THE OTHER QUESTIONS, AND IS NOT REQUIRED.

[1] though we should not forget that is also the result of preference, philosophy, and fashion in a given period.

Hints:

- *Making the estimate just involves eye-balling the period from your graph;*
- *for the FT plot, you will notice that the most relevant information is located in some corner of the graph. You can experiment with removing the mean from the signal before Fourier-transforming, or try semi-log or log-log axes;*
- *you don't have to do anything fancy to determine the k that has the maximum peak. I recommend using a subplot to zoom in on the region of interest in the plot to isolate the specific k value where the max peak occurs.*

(b) Newman 7.4.[2]

HAND IN ONE PLOT OF ALL OF THE TIME SERIES REQUESTED IN THE QUESTION, THE CODE, AND WRITTEN ANSWER TO THE QUESTIONS.

Note: the warning about making sure that the curves have different colours relates to an old version of Matplotlib. As of October 2018, Matplotlib's latest version takes care of it automatically. However, you may want to play with the line style (solid, dashed, etc.).

(c) Newman 7.6.

HAND IN ONE PLOT FOR ALL OF THE TIME SERIES REQUESTED, AND WRITTEN ANSWERS TO THE QUESTIONS (NO CODE).

Notes:

- *The "artifacts" Newman talks about can fit under what is called the "Gibbs phenomenon". Look it up, you have seen it before.*
- *Newman mentions a few drawbacks for the DCT compared with the DFT, but there are more. In particular, you may notice that away from the Gibbs oscillations, the inverse smoothed DCT "struggles" more at following the raw data than the inverse smoothed DFT. That is because the spectrum of a DCT is usually less compact than the spectrum of the corresponding DFT. When you discarded 98% of the DCT coefficients, you discarded a lot more information than when you discarded 98% of the DFT coefficients.*

(d) Newman 7.3: musical instruments.

HAND IN ONE PLOT FOR EACH WAVE FORM AND FOURIER TRANSFORM, AND WRITTEN ANSWERS (NO CODE).

2. **[40%] Newman 7.9: Image deconvolution** Note that I find Newman a bit misleading. The technique he describes only works for pictures that are out-of-focus, in the

[2] A short (and completely optional) cultural introduction:
https://www.npr.org/sections/money/2017/01/04/508261371/episode-443-dont-believe-the-hype

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