**MIT-WHOI JP Summer Math Review**
Notes from Class
Topic: **Signal Processing & Time Series Analysis**
Instructor: **Alec Bogdanoff** (alecb@mit.edu)
*Based on Julia Hopkins's notes.*

## Fourier Transforms

Definition: formula to translate any function into a sum of sines and cosines

- Properties of sinusoidal functions make them easier to handle when solving equations or performing operations on the equations
- Used heavily in signal processing to identify dominant frequencies and phases in a time series. Many of the examples below will use the concepts of time and frequency to explain, more intuitively, the formulaic math.

*Basic Form*

Forwards Fourier Transformation

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dx$$

Backwards Fourier Transformation

$$f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi i k x} dk$$

Intuitively, one can think of the forward transform as moving into the frequency domain, while the backwards transform takes the function to the time or spacial domain.

*Applications*

- Time series spectral analysis (focus of many data-driven oceanographic studies)
- Quantum mechanics (ask your favorite quantum physicist up at MIT for more)
- Solving differential equations (multiply the Fourier transform by $2\pi i k$ and take the inverse)

$$\frac{d}{dx} f(x) = 2\pi i \int_{-\infty}^{\infty} k F(k) e^{-2\pi i k x} dk$$

- Convolution (convolving two functions in the time domain is the same as multiplying them in the Fourier domain)

$$f * g(x) = \int_{-\infty}^{\infty} F(k) G(k) e^{-2\pi i k x} dk$$

*Examples*

Time Series Analysis

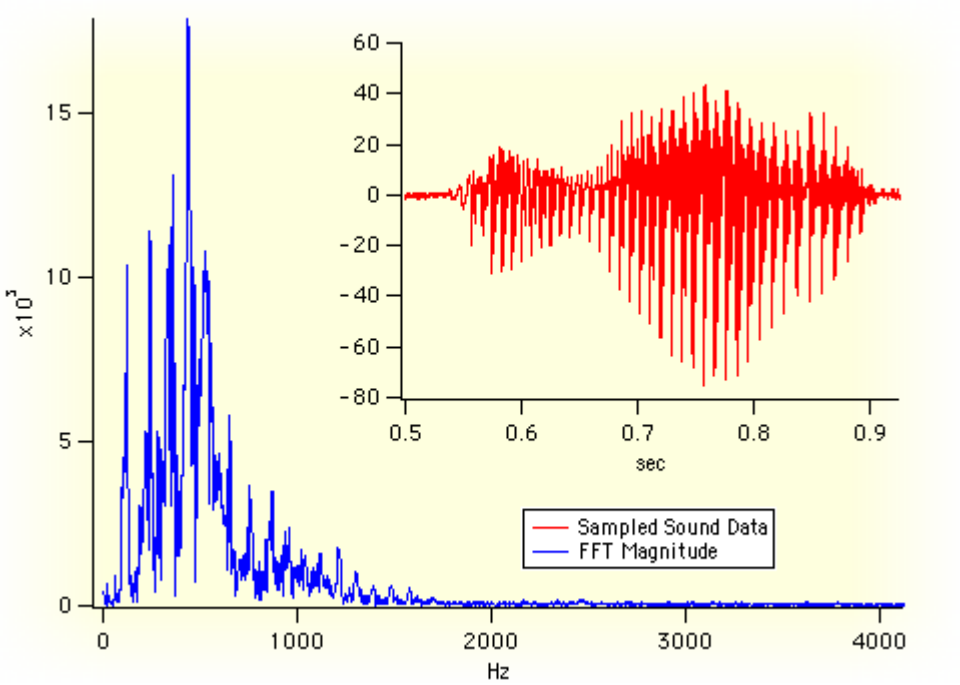



Image from http://www.wavemetrics.com/products/igorpro/dataanalysis/signalprocessing.htm

Transforming a Cosine Function

$$f(x) = \cos(2\pi s x)$$

Transforming a Square Function

$$f(x) = \begin{cases} 1 & \text{if } -a/2 < x < a/2 \\ 0 & \text{else} \end{cases}$$

Other Common Transforms

| Spatial Domain | | Frequency Domain | |
|---|---|---|---|
| **Function Name** | **Formula** | **Function Name** | **Formula** |
| Cosine | $\cos(2\pi sx)$ | Deltas | $\frac{1}{2}[\delta(k+s)+\delta(k-s)]$ |
| Sine | $\sin(2\pi sx)$ | Deltas | $\frac{1}{2}[\delta(k+s)-\delta(k-s)]$ |
| Constant | $a$ | Delta | $\delta(k)$ |
| Delta | $\delta(x)$ | Unit | $1$ |
| Square | $\begin{cases}1 & \text{if} -\text{a}/2 < x < \text{a}/2 \\ 0 & \text{else}\end{cases}$ | Sinc | $sinc(a\pi k)$ |
| Triangle | $\begin{cases}1-\lvert x\rvert & \text{if} -\text{a} < x < \text{a} \\ 0 & \text{else}\end{cases}$ | $\text{Sinc}^2$ | $sinc^2(a\pi k)$ |
| Gaussian | $e^{-\pi x^2}$ | Gaussian | $e^{\pi k^2}$ |

*Discrete Fourier Transform (DFT)*

Useful for looking at data in particular. Simply convert the integrals in the Fourier transform equations above into their discrete counterparts: summations.

Forwards Fourier Transformation

$$F_k = \sum_{n=0}^{N-1} f_n e^{-2\pi ikn/N}$$

Backwards Fourier Transformation

$$f_n = \frac{1}{N}\sum_{n=0}^{N-1} F_k e^{2\pi ikn/N}$$

Think about this as a least squares problem for a second.

*Fast Fourier Transform (FFT)*

DFT takes about $2N^2$ computations to calculate, where $N$ is the number of data points. The more efficient FFT takes $2Nlog_2N$ computations. A DFT can be computed as an FFT if $N$ is a power of 2 (if not, there are ways of padding your data to make it possible to use an FFT, such as adding 0's to the end).
There are many different algorithms used to compute an FFT. The basic premise behind all of them takes advantage of the ideally periodic or repetitive nature of the signal. The most commonly used FFT algorithm is the Cooley-Turkey algorithm, which simply divides the computation of the DFT into separate computations of the DFT for all odd-indexed points and all even-indexed points. This division can be performed recursively (continuously dividing the series of points) to get the desired efficiency and computation speed.

There have been many improvements to the algorithm over time, and the version you are most likely to use will not be as simple as what has been described above. This class will not go into

the rigorous math behind these algorithms. Instead, it will present some idea of how to work with the FFT.

*MATLAB and the FFT*

Let y be the discrete timeseries you wish to transform to the frequency series Y.

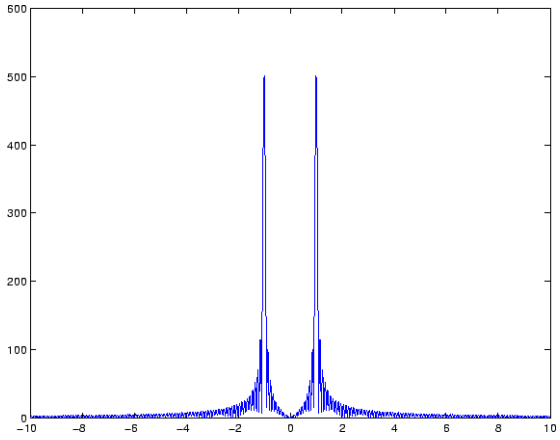
Forward transform:

```
Y = fft(y); %it is literally that simple
```

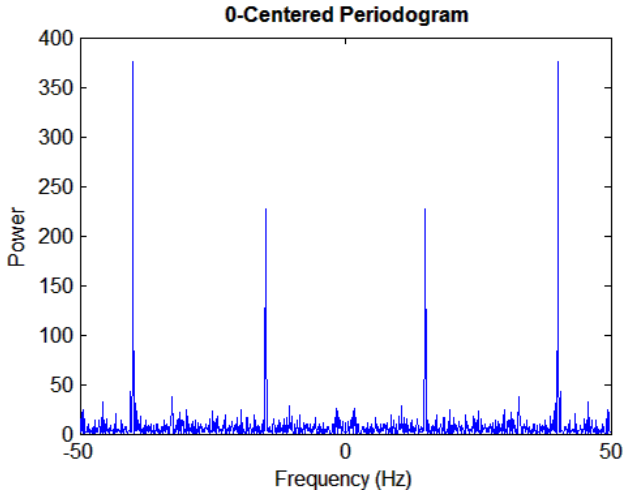
Backward transform:

```
y = ifft(Y); %...or is it?
```

MATLAB has a bit of an issue with the fft function. You might have noticed that the DFT formula divides by a factor of N for the inverse Fourier transform. You would expect MATLAB to do the same. For the most part, it does this division somewhere in the depths of the code for `fft` and `ifft`, but if you do anything more than implement the two formula above immediately after one another you need to be careful about that constant factor and precisely where in this sequence MATLAB decides to use it.
Another issue is that MATLAB calculates the transform a frequency domain which includes negative frequencies – so the result ends up often looking like this:



Or this:




The way to control that is with the `fftshift.m` function. This shifts the 0 frequency to the first position in your vector and rearranges everything correctly behind it. Once again, though, be careful of the inverse transform if you shift the function. That constant factor (or lack thereof) could come back to bite you.

*Filtering a Signal*

Signals can have noise or otherwise unwanted frequency components which make analysis difficult. Fourier transforms can be used to help filter or smooth a signal.

Disclaimer: there are many types of filters. Each is appropriate for a different type of analysis. This lesson will not give an exhaustive list of these filters. The book **Data Analysis in Physical Oceanography by Emery and Thompson**, on the other hand, will give an exhaustive list. Ask any upper class PO student to lend you a copy.

Moving Average Filter

One of the most common filters applied to data, it averages together a set number of neighboring points at every instance in the signal. Mathematically, it can be understood as a convolution of your data with a rectangular function (which only spans the number of points you want to average).

Weighted Moving Average Filter

Exactly the same as the moving average filter, except that the points averaged are given different weights. This could take the form of a convolution with a triangle function, or a parabolic function, or a particular sinusoidal function.

**NOTE**: In general, a function that performs well in the spacial domain will do the opposite in the frequency domain. High resolution in the spatial domain, for instance, corresponds to low resolution in

the frequency domain and vice versa. It is important to be aware of these trade-offs as you begin to analyze data, and certainly to determine whether the frequency or spacial signal is more important for your analyses before proceeding with any kind of filter or smoothing scheme.

Examples:

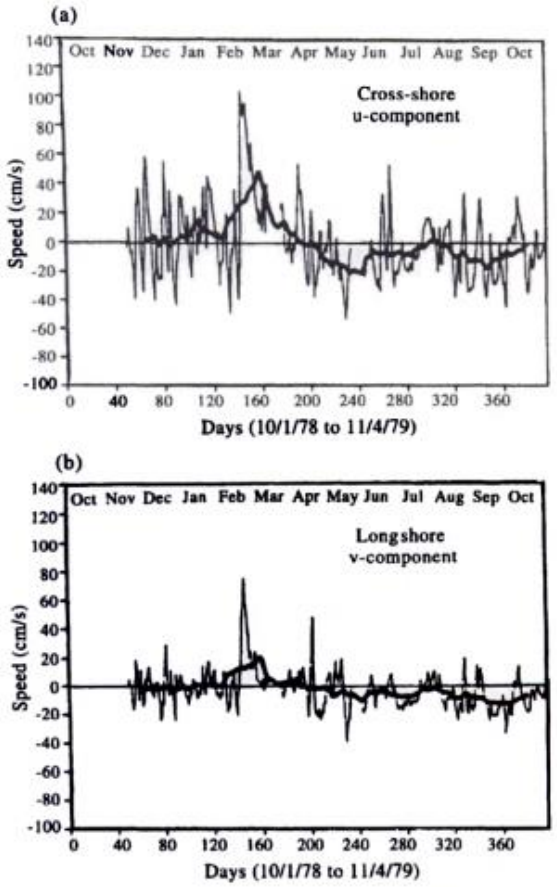



*Figure 5.10.9. Daily mean time series of cross-shelf (top) and longshelf (bottom) near-surface currents off Cape Romain in the South Atlantic Bight for the period 10 January 1979 to 11 April 1979. Thin line: Daily average data. Thick line: 30-day running-mean values. (From McClain et al., 1988.)*

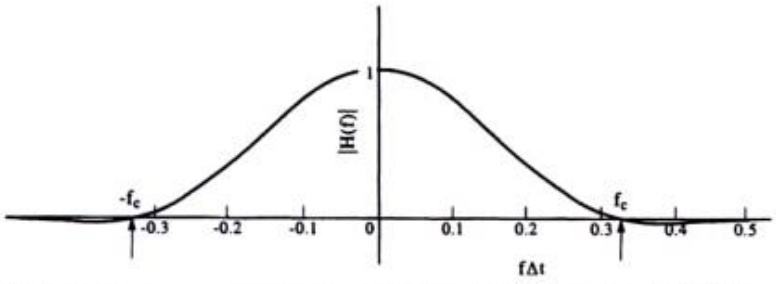



*Figure 5.10.10. The frequency-response function, $|H(f)|$, for the Godin-type filter $A_2^2 A_3/(2^2 3)$ used to smooth 30-min data to hourly values. The horizontal axis has units $f\Delta t$, with $f_N \Delta t = 0.5$; $f_c$ is the cut-off frequency. (From Godin, 1972.)*

http://www.mathworks.com/help/signal/functionlist.html
*https://onlinecourses.science.psu.edu/stat510/*