

Data X

LTI and Spectral Information

Data X: A Course on Data, Signals, and Systems

Ikhlaq Sidhu
Chief Scientist & Founding Director,
Sutardja Center for Entrepreneurship & Technology
IEOR Emerging Area Professor Award, UC Berkeley

Jean Baptiste Joseph Fourier (1768-1830)

- Had crazy idea (1807):
- Any periodic function can be rewritten as a weighted sum of **Sines** and **Cosines** of different frequencies.
- Don't believe it?
 - Neither did Lagrange, Laplace, Poisson and other big wigs
 - Not translated into English until 1878!
- But it's true!
 - called **Fourier Series**
 - Possibly the greatest tool used in Engineering

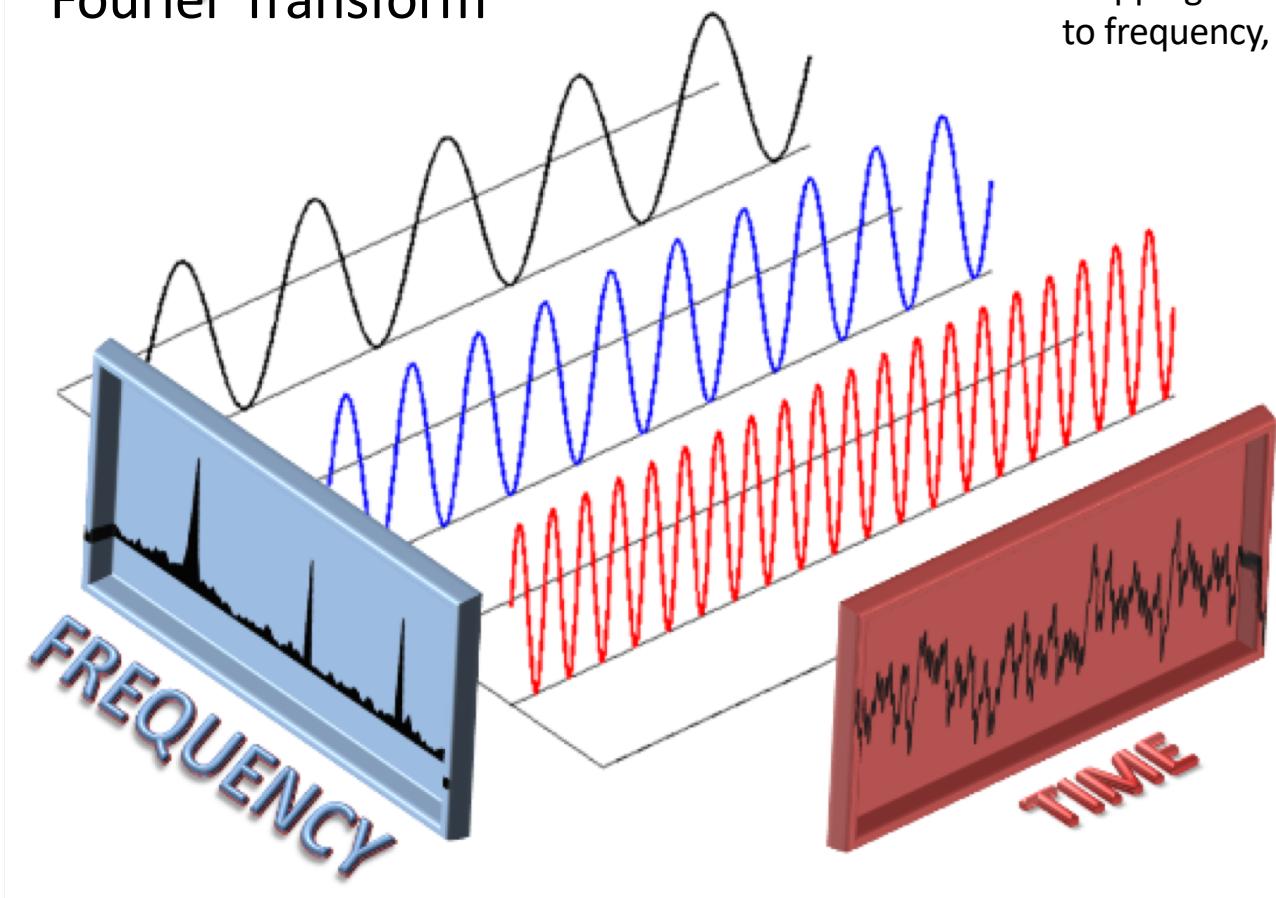


S. Narasimhan, Computer Vision



Fourier Transform

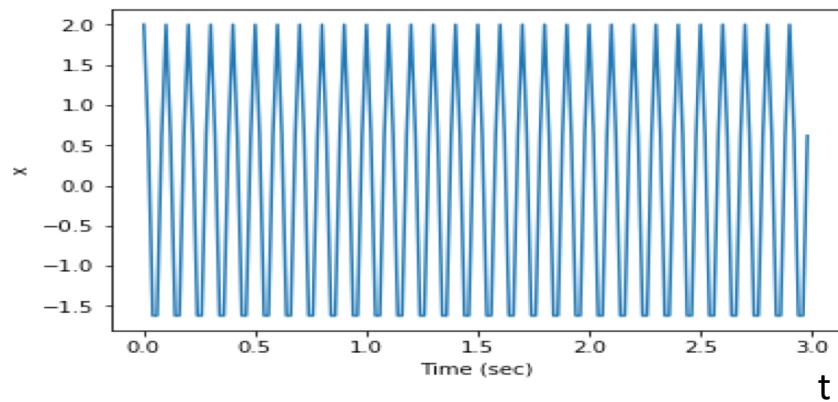
Mapping from a time
to frequency, and back.



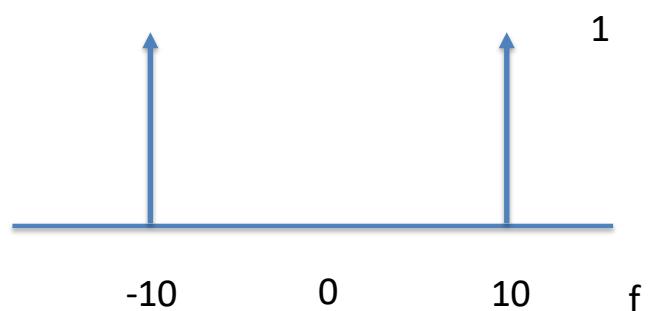
groups.csail.mit.edu/netmit/wordpress/projects/sparse-fourier-transform/



Intuition



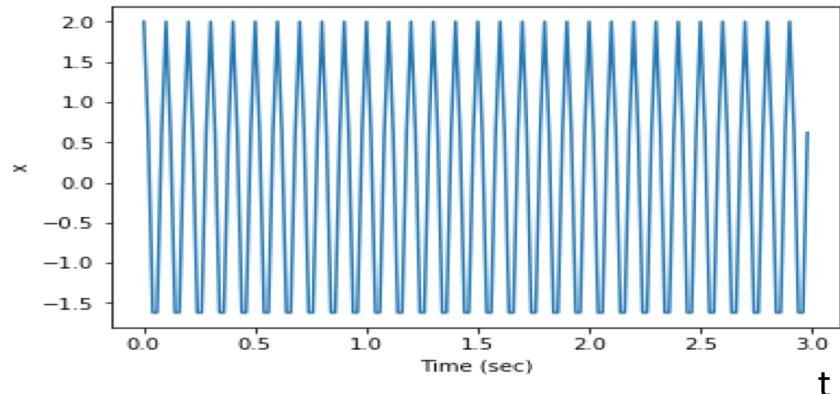
$$x(t) = 2 \cos(2 \pi 10 t)$$



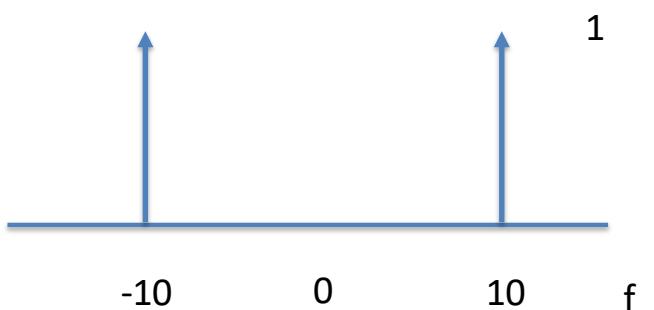
$$X(f) = \delta(f+10) + \delta(f-10)$$



Intuition



$$x(t) = 2 \cos(2 \pi 10 t)$$



$$X(f) = \delta(f+10) + \delta(f-10)$$

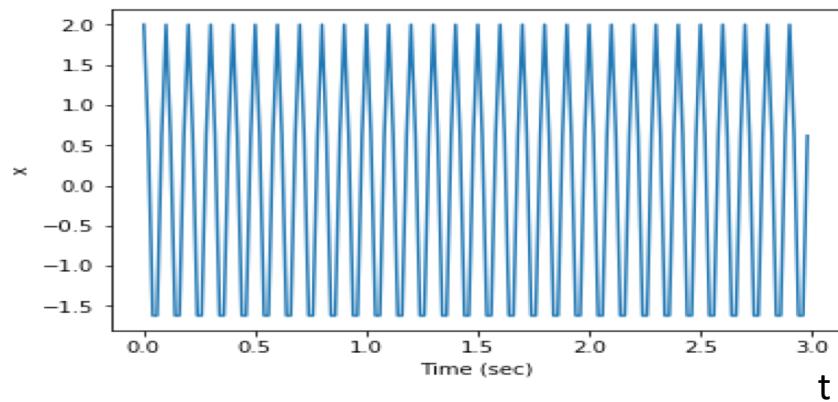
For Reference

$$X(f) = \int_{-\infty}^{\infty} x(t) \times e^{-i2\pi ft} dt \quad f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi sx} ds$$

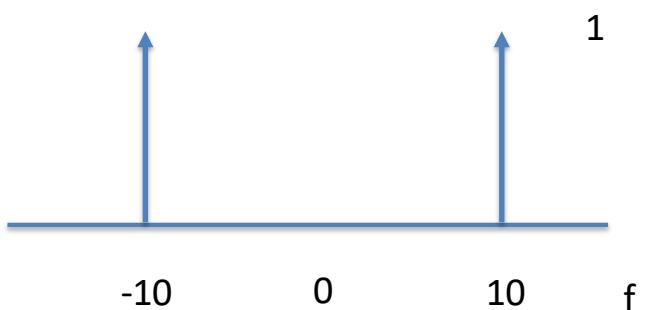
$$e^{-j2\pi ft} = \cos(2\pi f x) - j \sin(2\pi f x)$$



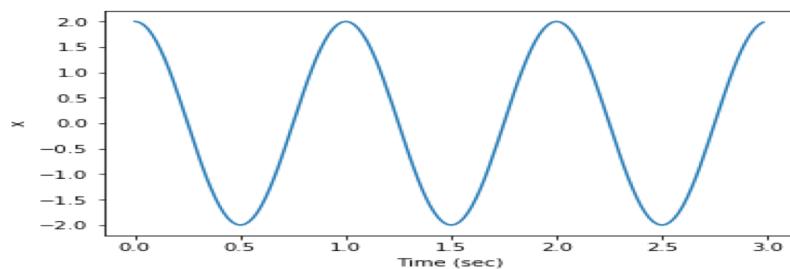
Intuition



$$x(t) = 2 \cos(2 \pi 10 t)$$



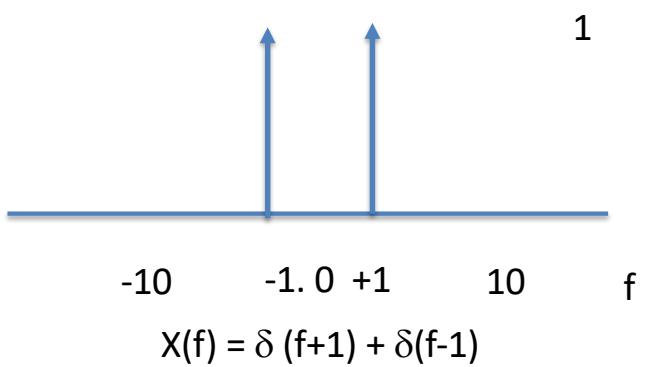
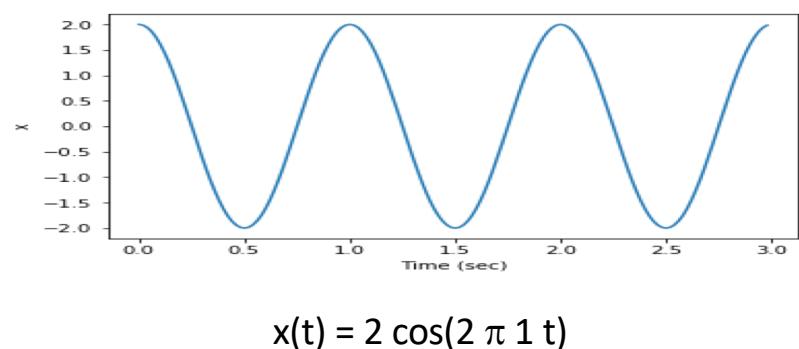
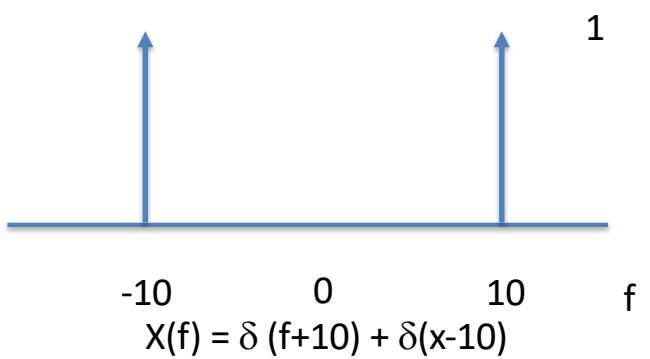
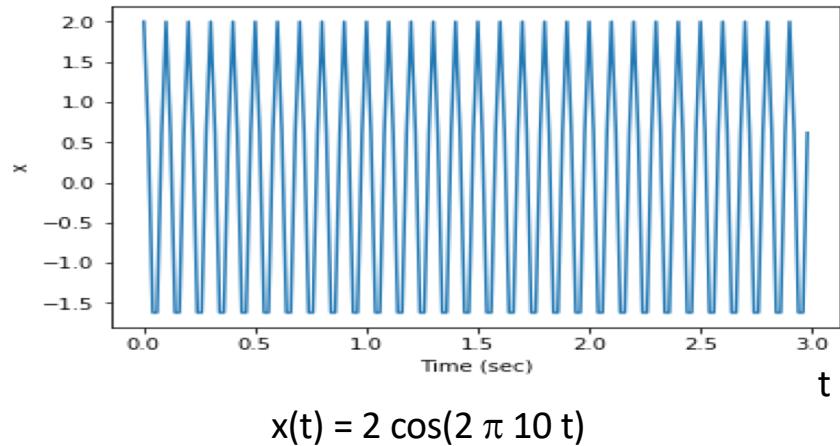
$$X(f) = \delta(f+10) + \delta(f-10)$$



$$x(t) = 2 \cos(2 \pi 1 t)$$



Intuition



Stock Market Data Signal

Cisco Stock of 1 year

- Downloaded CSV from Yahoo Finance
- Read into 'x'
- Pandas -> Numpy

```
df0 = pd.read_csv('cisco1yr.csv')
print(df0.head())
```

Date	Open	High	Low	Close	Adj Close	Volume	
0	2017-11-13	33.860001	34.209999	33.830002	33.950001	32.948601	16441500
1	2017-11-14	33.860001	34.160000	33.799999	34.040001	33.035950	17468500
2	2017-11-15	33.970001	34.310001	33.750000	34.110001	33.103882	30829600
3	2017-11-16	36.040001	36.669998	35.830002	35.880001	34.821678	61177100
4	2017-11-17	35.900002	36.320000	35.810001	35.900002	34.841087	27988000

```
x=df0['Open'].as_matrix()
print (x[0:10])
[ 33.860001 33.860001 33.970001 36.040001 35.900002 35.93 36.75 36.700001
 36.41 36.509998]
```



Stock Market Data Signal

Cisco Stock of 1 year

- Numpy -> FFT

Notice: Complex

Absolute Value for Magnitude

What do we do with this?

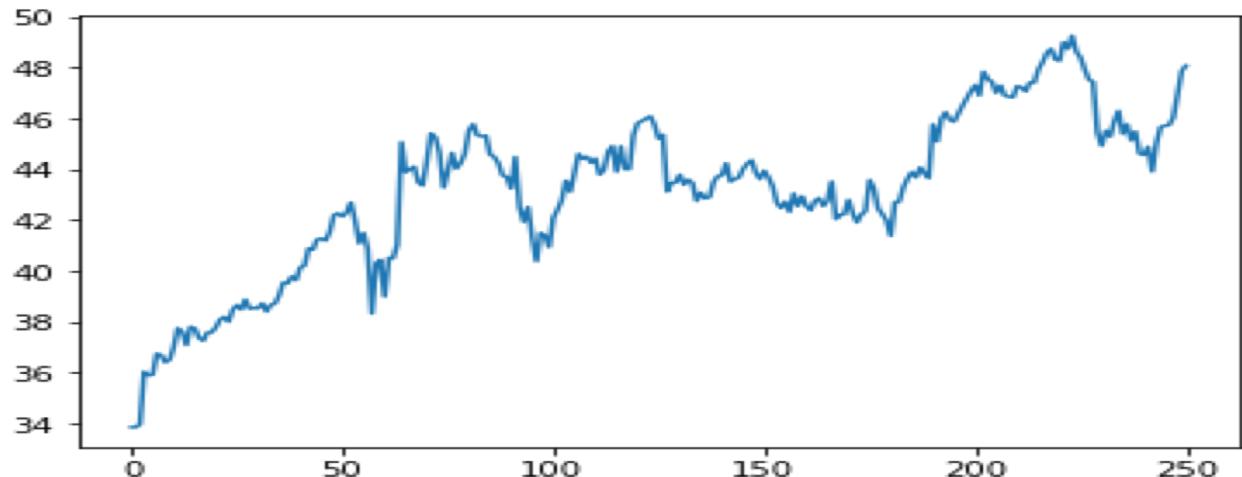
```
fft_x = np.fft.fft(x)
print (fft_x [0:10])
print (abs(fft_x [0:10]))  
  
n = len(x)
print (n)  
  
[ 1.08330600e+04 +0.j -1.07898092e+02+243.66563655j -
 7.89148905e+01+342.12255602j -1.23044892e+02+175.27530998j -
 1.63652374e+01 +18.27080103j -4.08661037e+01 +44.76386467j
 2.87257607e+01+104.2437998j 3.87999056e+00 -24.25520849j -3.21137204e+01
 +48.40457951j 2.10073911e+01 +58.12600158j]  
  
[ 10833.059996 266.48628599 351.10597158 214.15293517 24.52841547
 60.61222661 108.12927045 24.56358007 58.08867668 61.80568373]
```

251



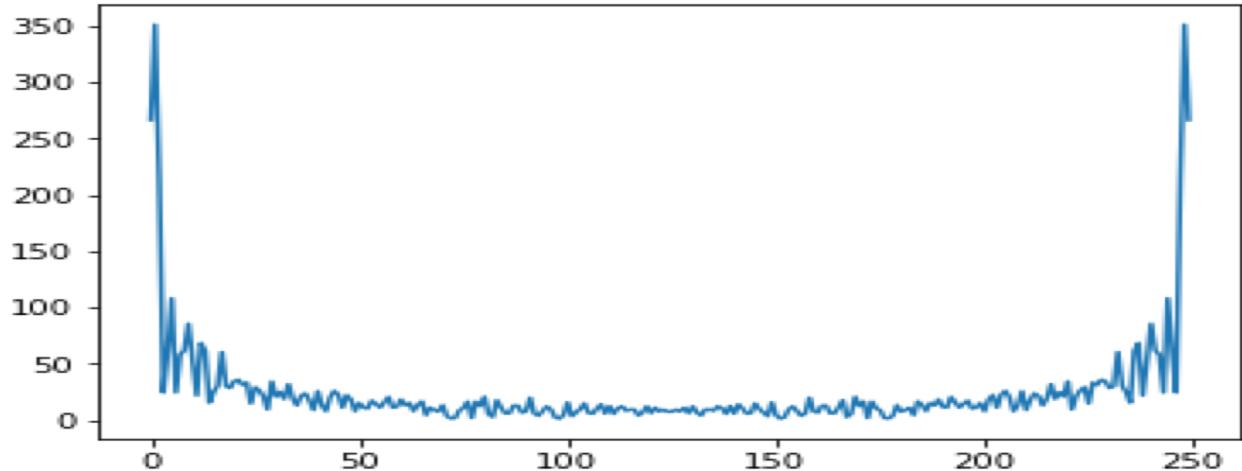
Stock Market Data Signal

- Data Signal in frequency
- Magnitude is plotted
- `plt.plot(x)`
- `plt.plot(abs(fft_x)[1:])`



Questions:

- What does the FFT mean?
- What is the frequency of each point?

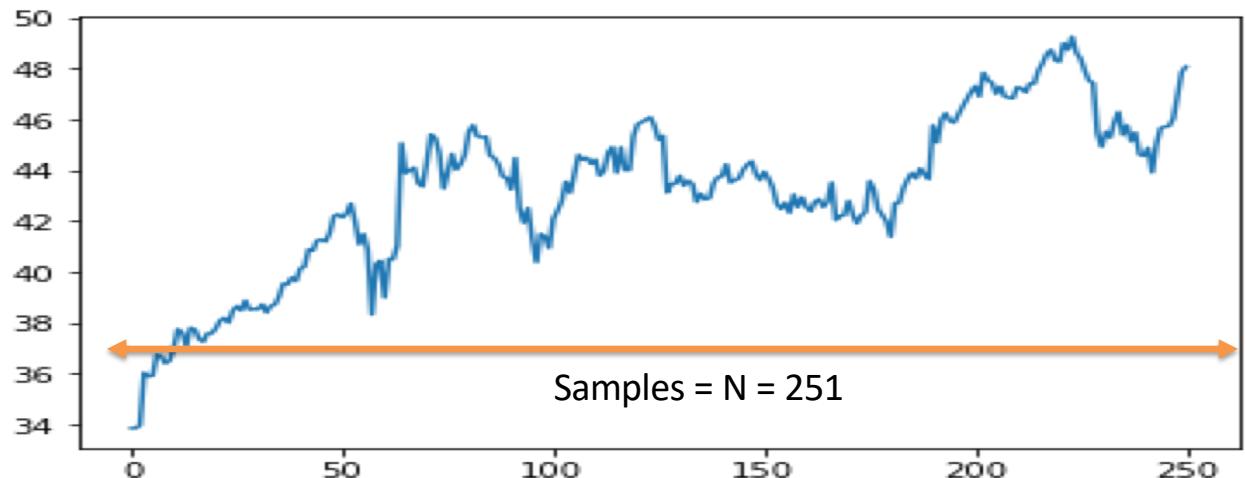


Stock Market Data Signal

Basic FFT Rules:

- N points of Data
- ->
- N points in FFT array

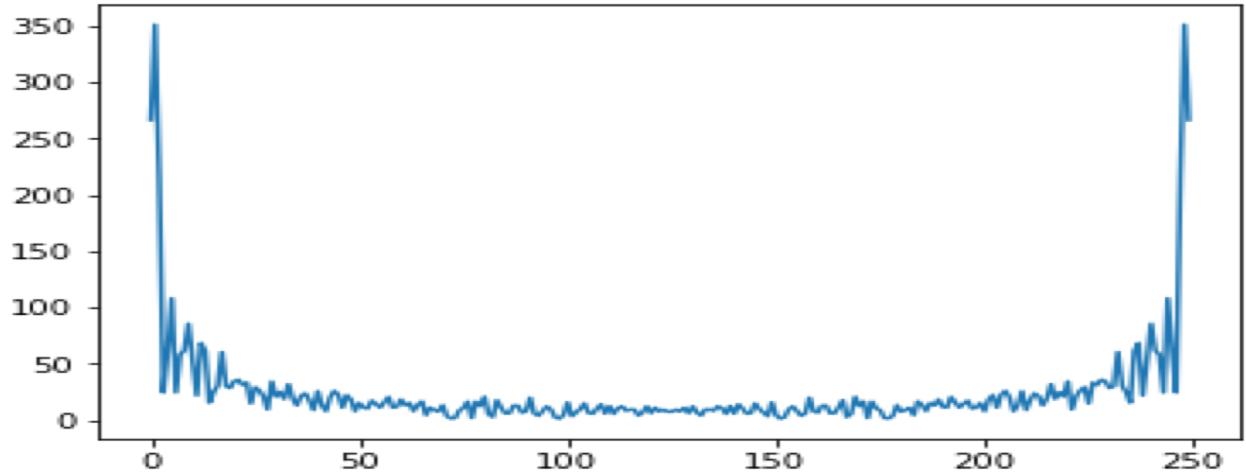
X



In this case:

- len(x) was 251
- ->
- len(fft_x) is 251

fft_X



Stock Market Data Signal

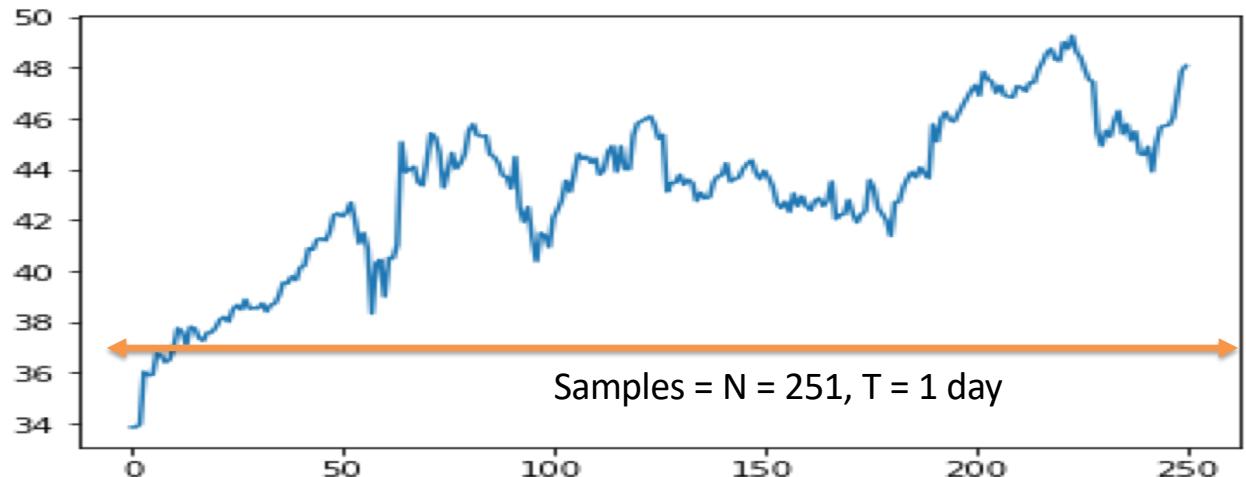
Basic FFT Rules:

- N points of Data ->
- N points in FFT array

In this case:

- `len(x)` was 251
- ->
- `len(fft_x)` is 251
- Each fft bin
(a bin is a freq. point)
Is fs/N
 $(1 \text{ cycle/day})/251$
 $=0.004 \text{ Cycles/day/bin}$

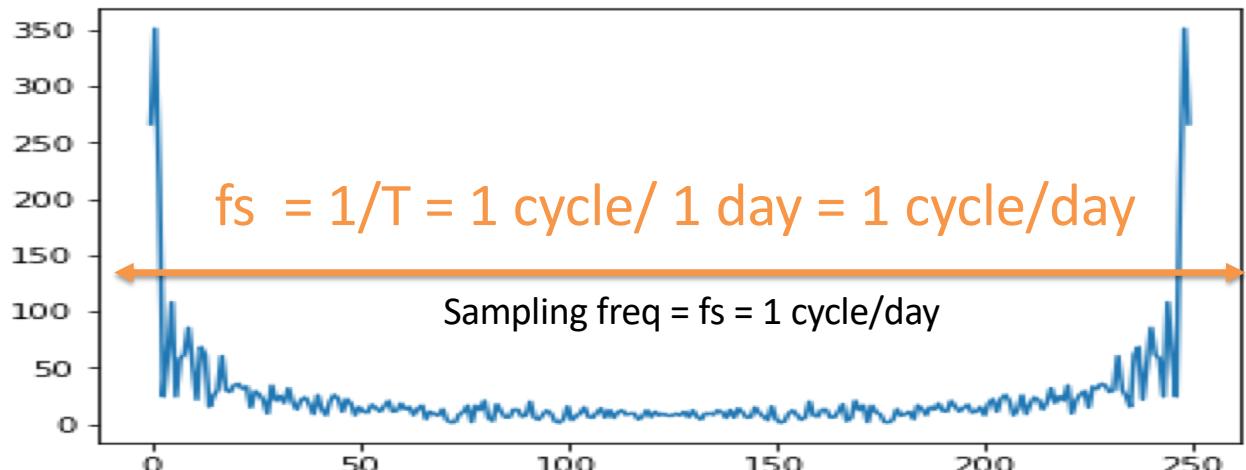
X



fft_x

$$fs = 1/T = 1 \text{ cycle} / 1 \text{ day} = 1 \text{ cycle/day}$$

Sampling freq = $fs = 1 \text{ cycle/day}$



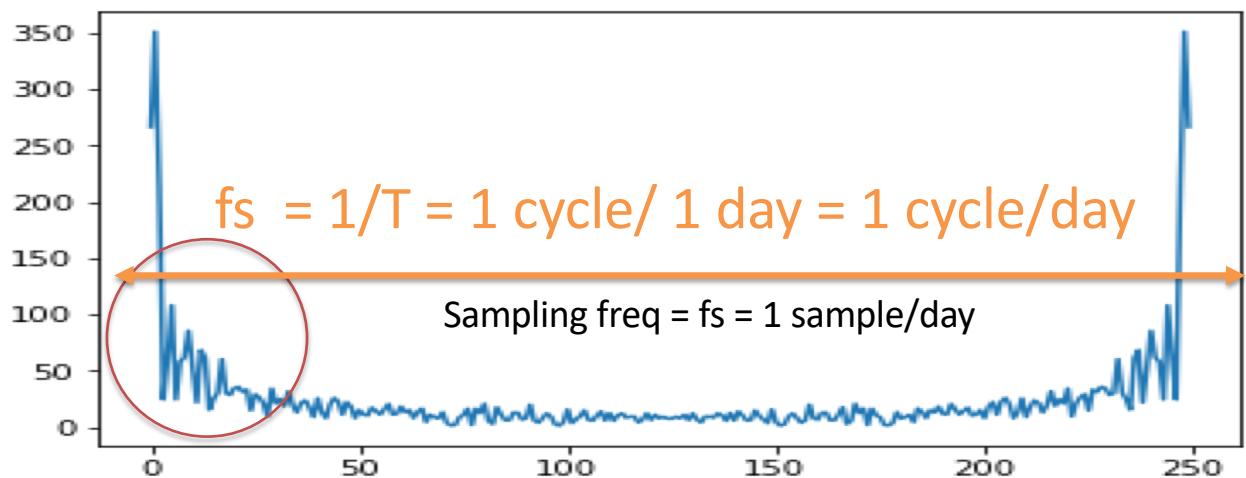
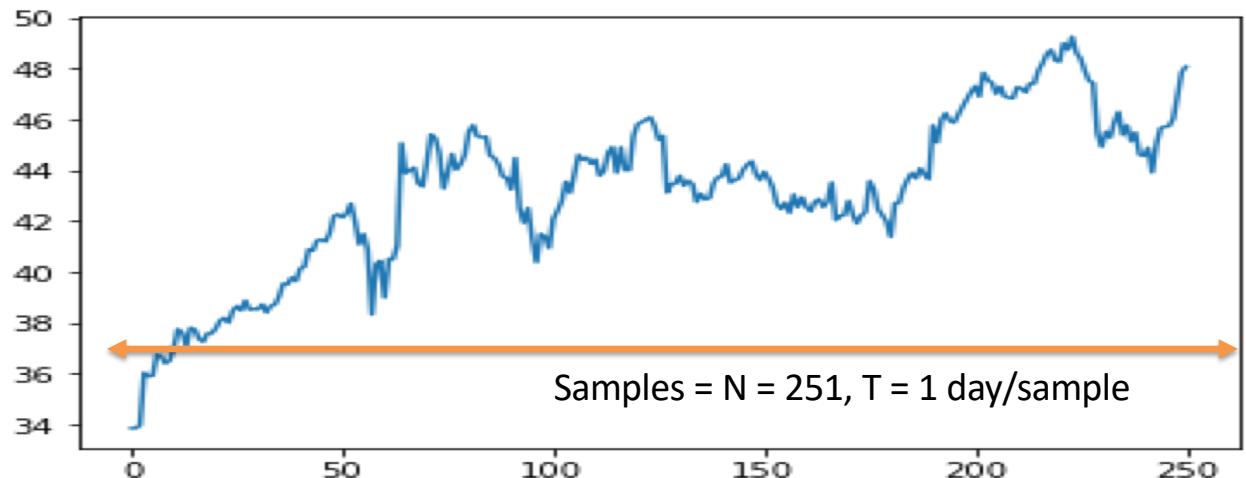
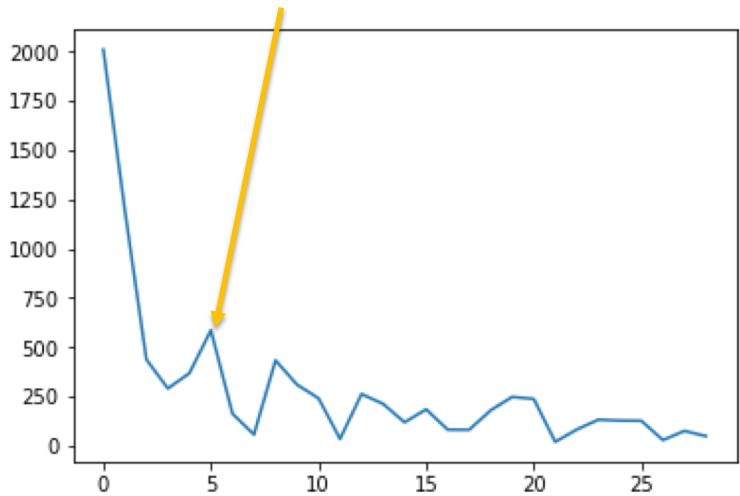
Data X

Stock Market Data Signal

Each fft bin $s \frac{fs}{N}$
 $(1 \text{ sample/day})/251$
 $=0.004 \text{ Cycles/day/bin}$

- Close Up: First 25 points

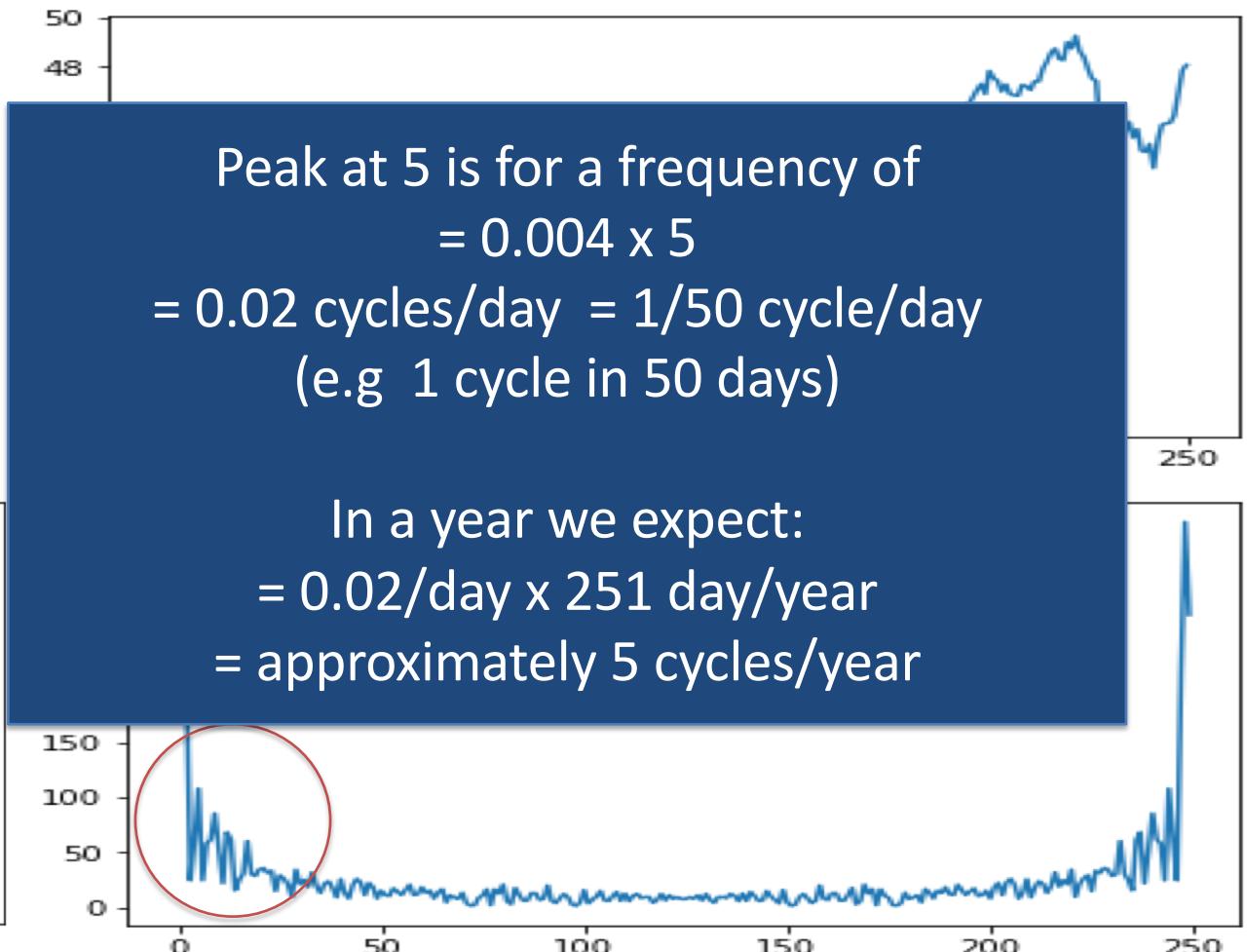
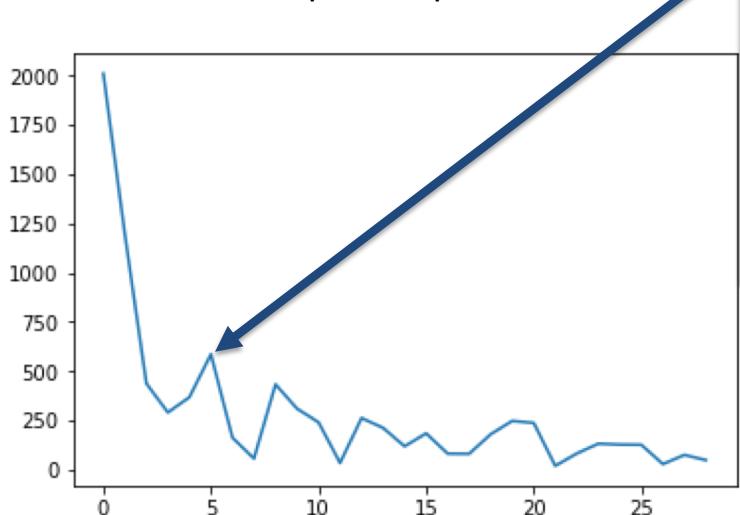
What is this peak at point 5?



Stock Market Data Signal

- Data Signal in frequency
- Magnitude is plotted
- Close Up: First 25 points

What is this peak at point 5?



Same Calculation Units Matter

Basic FFT Rules:

- N points of Data ->
- N points in FFT array

Sampling rate $fs = 251/\text{year}$

Each fft bin:

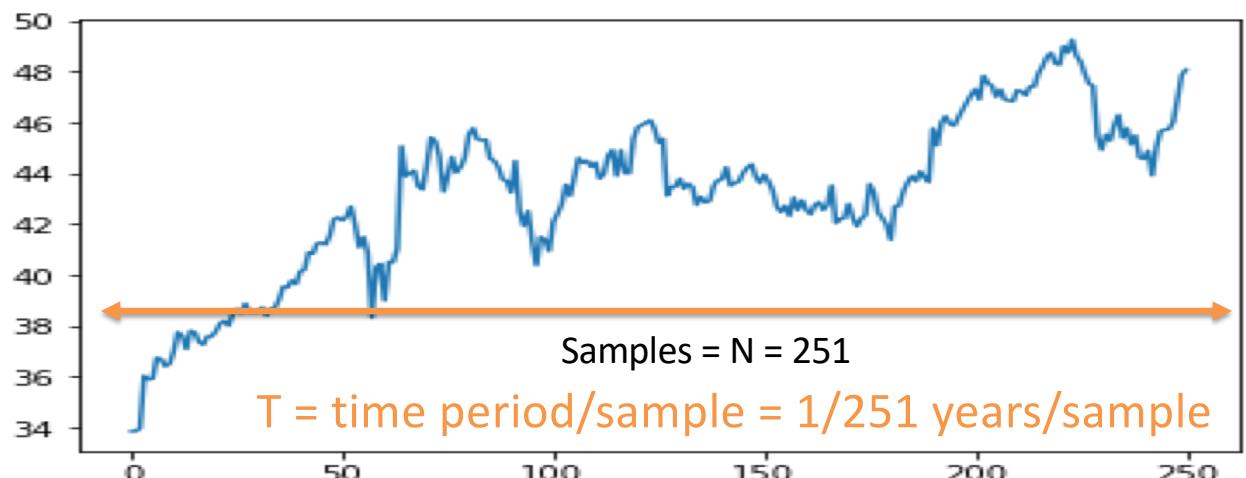
$$\text{Is } fs/N$$

$$= (251/\text{year})/251$$

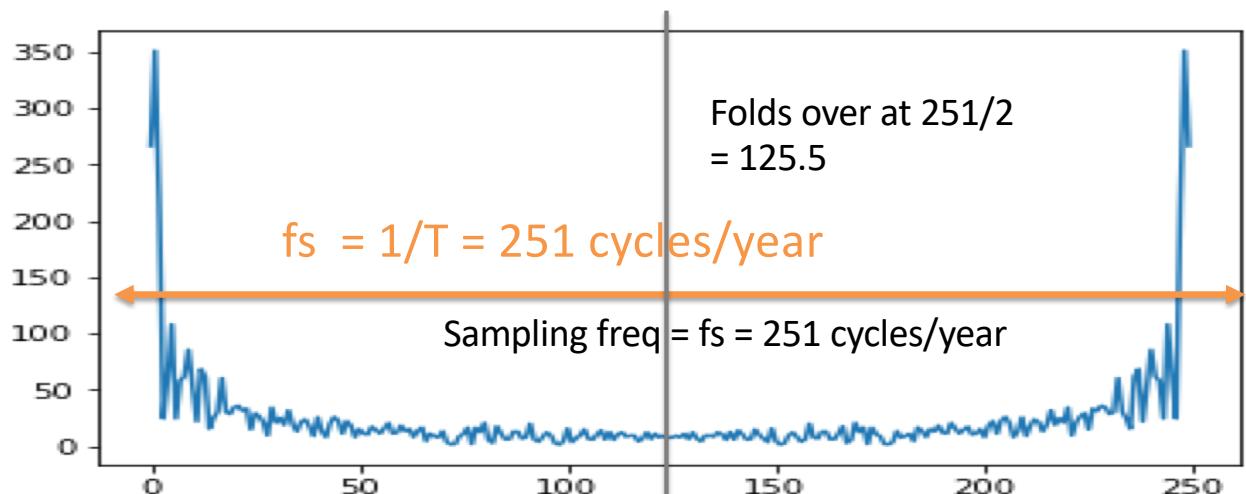
$$= 1 \text{ Cycles/year/bin}$$

- So point 5 is for 5 cycles/year

X

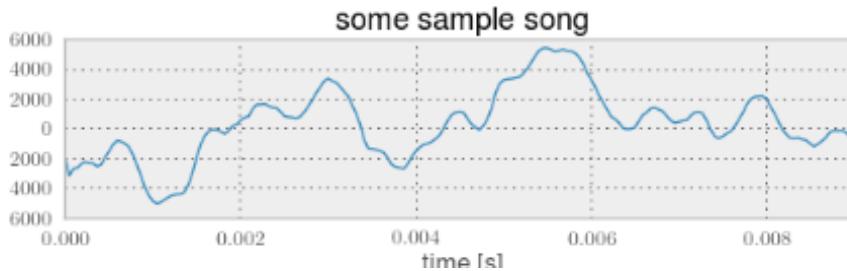


fft_x

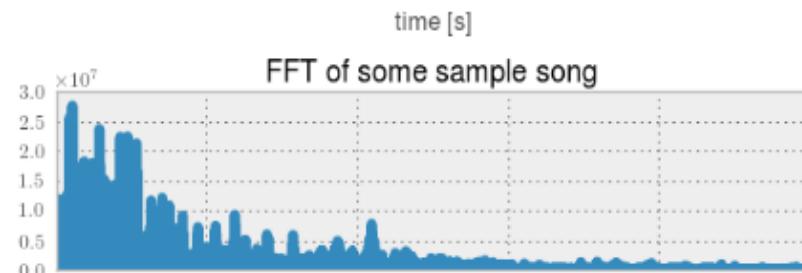


Data X

Try it Yourself



3 seconds of time
24000 samples of audio in an array



How many samples in FFT?

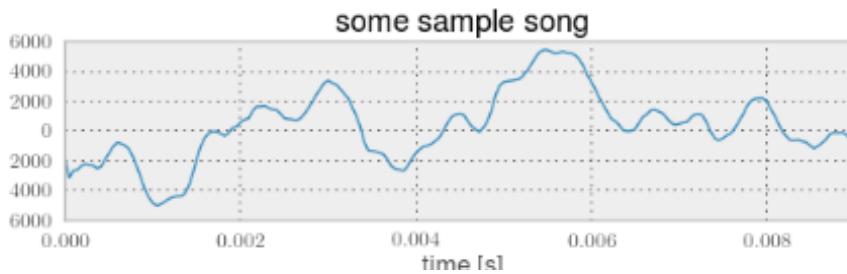
What is the full spectrum width of the FFT?

What is each point (bin) in Hz

What is the freq range between point 1000 and 2000?



Try it Yourself

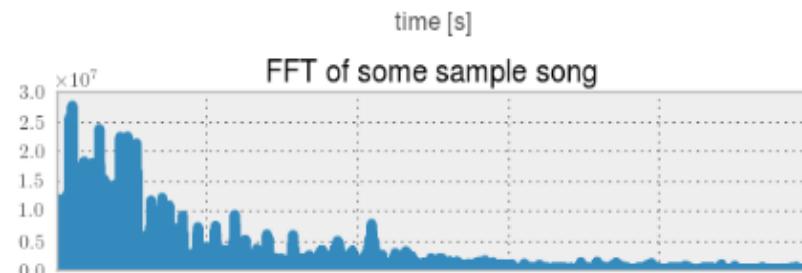


3 seconds of time

24000 samples of audio in an array

$T = 8000/\text{samples/sec}$

$T = 1/8000 \text{ sec}$



How many samples in FFT?

24000

What is the spectrum width of the FFT?

8000 Hz

What is each point (bin) in Hz

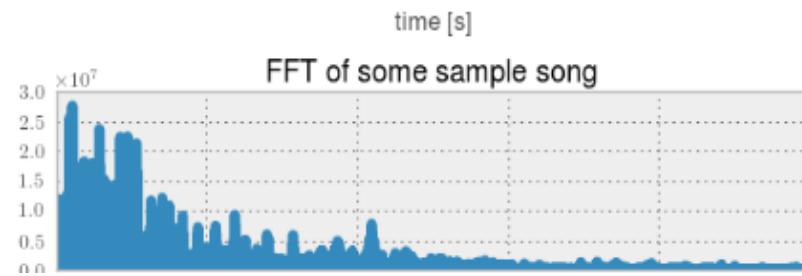
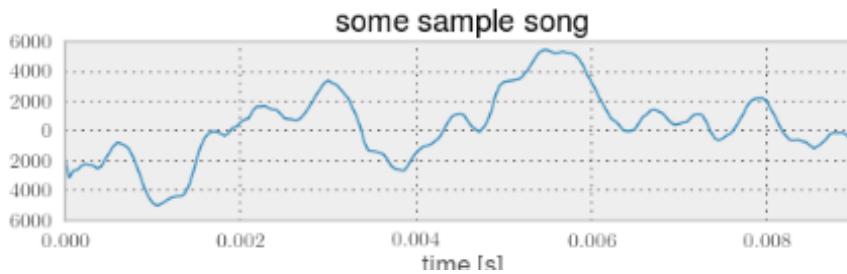
$8000/24000 = 1/3 \text{ Hz}$

What is the freq range between point 1000 and 2000?

333.3 Hz to 666.6 Hz



Try it Yourself



3 seconds of time

24000 samples of audio in an array

How many samples in FFT **of first 100 mS (ie 1/10 of sec)?**

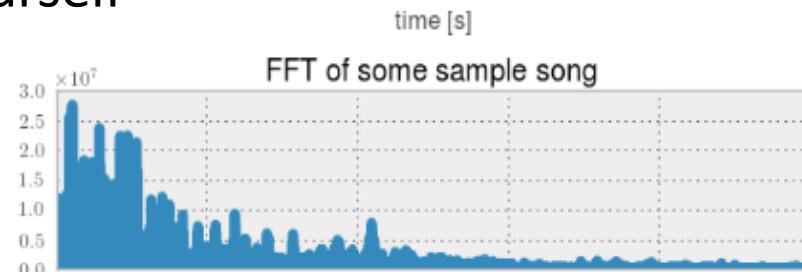
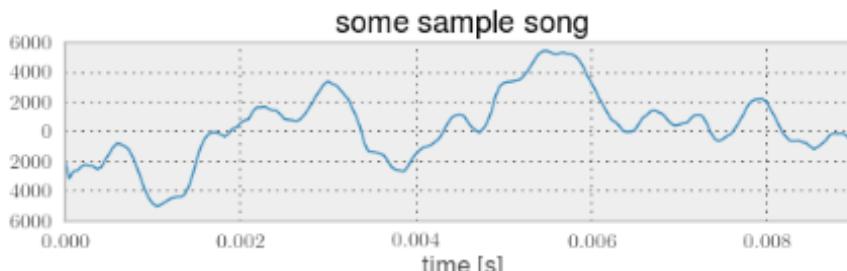
What is the spectrum width of the FFT?

What is each point (bin) in Hz

What is the freq range between point 10 and 20?



Try it Yourself



3 seconds of time

24000 samples of audio in an array

T still = 1/8000

100 mS has $8000 \times 1/10 = 800$ samples

How many samples in FFT of first 100 mS (ie 1/10 of sec)?

800

What is the spectrum width of the FFT?

8000

What is each point (bin) in Hz

$8000/800 = 10$ Hz

What is the freq range between point 100 and 200?

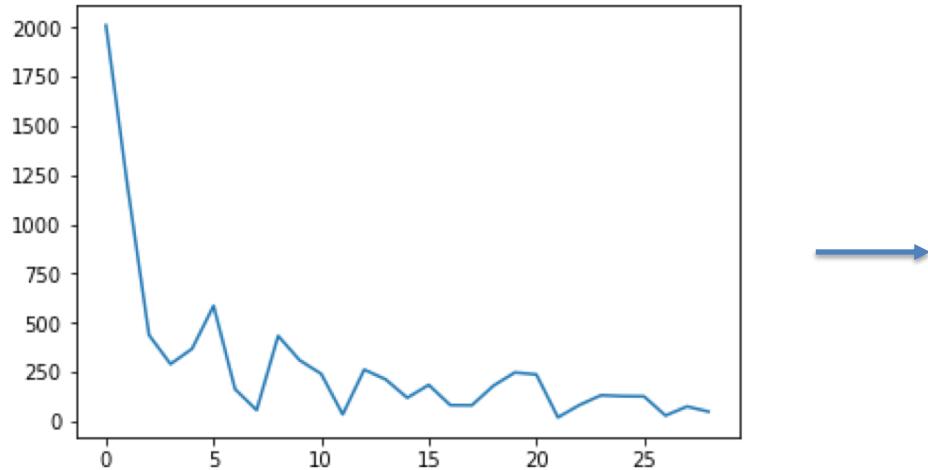
1000 Hz to 2000Hz



How can we use Spectrum / Fourier in ML?



Using FFT / Spectrum in ML



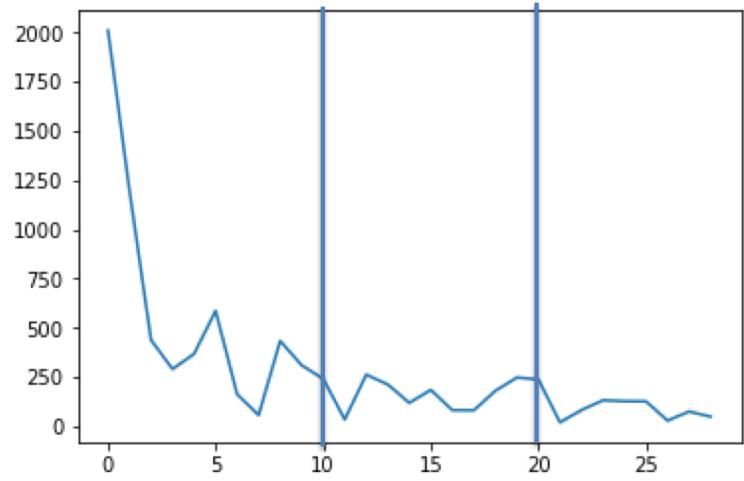
Features		Labels
X1 = highest peak	X2 = next peak	Y
5	8	label
		label
		label

One method is list highest peaks as features

Data X

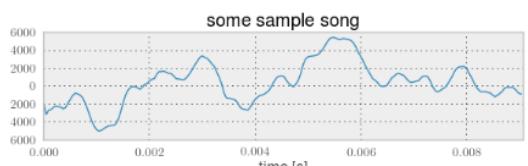
Using FFT / Spectrum in ML

Another method is to use the energy in each band as a feature

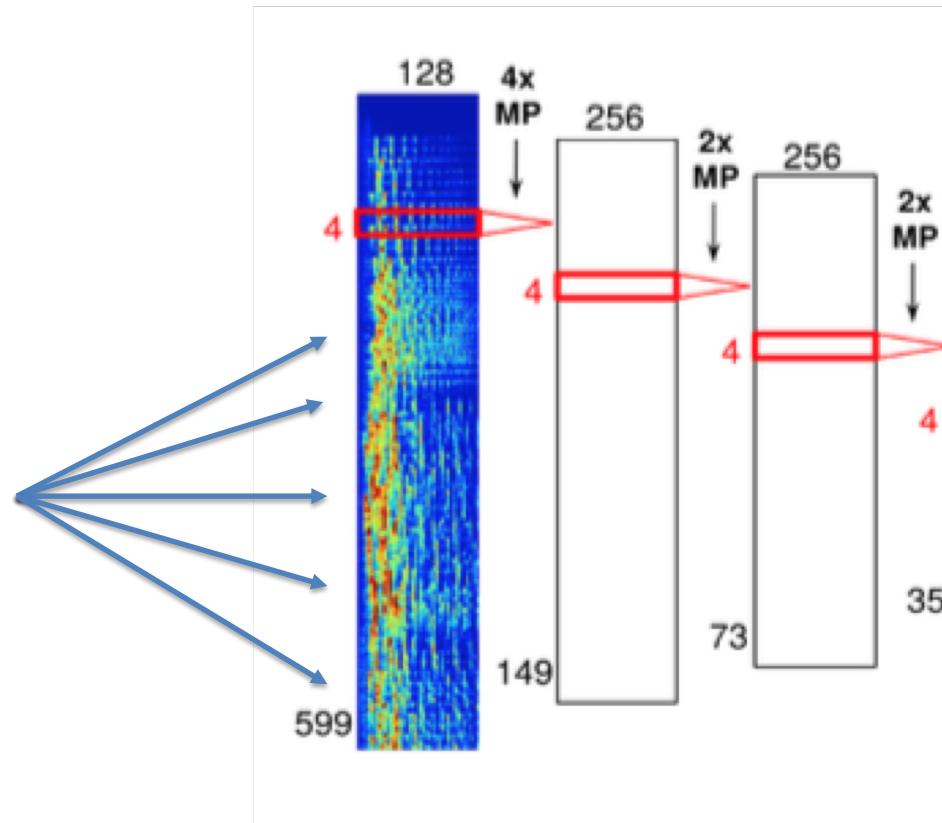


Data X

Using Data Signal in ML: Creating a picture of a song

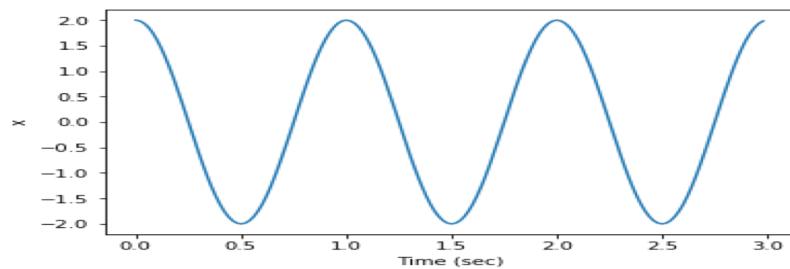


- Take song signal $x(n)$
- FFT of windows of 50 ms
- Stack on top of each other
- Classify with CNN

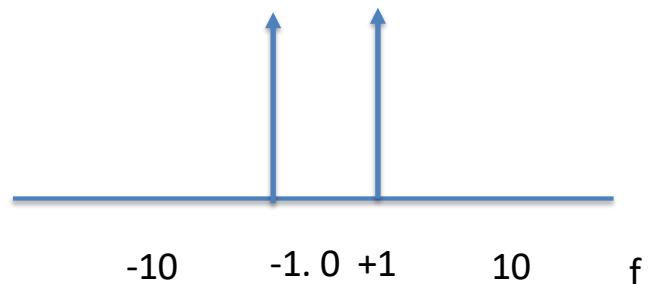


Intuition

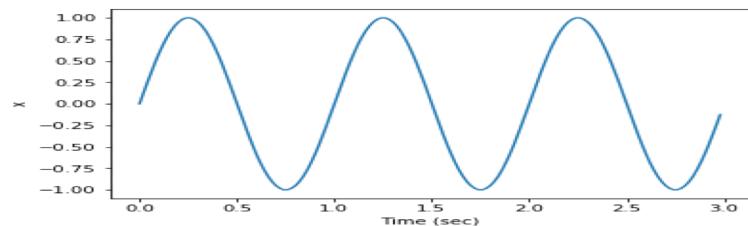
Magnitude is same
Phase is different



$$x(t) = 2 \cos(2 \pi 1 t)$$



$$X(f) = \delta(f+1) + \delta(f-1)$$



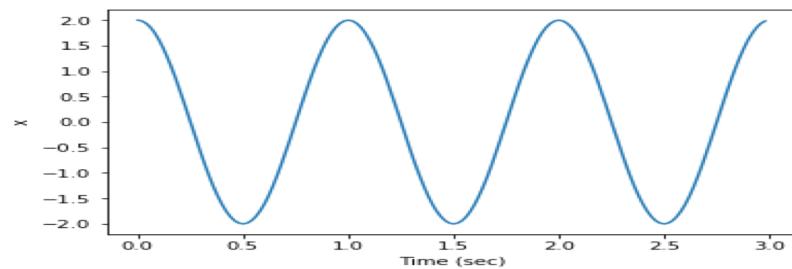
$$x(t) = \sin(2 \pi 1 t)$$



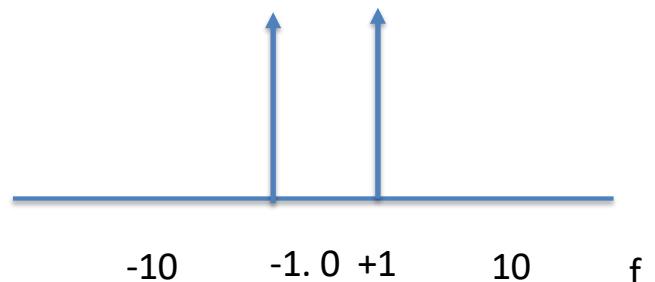
Data X

Intuition

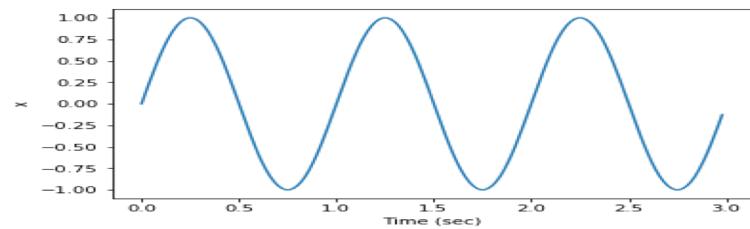
Magnitude is same
Phase is different



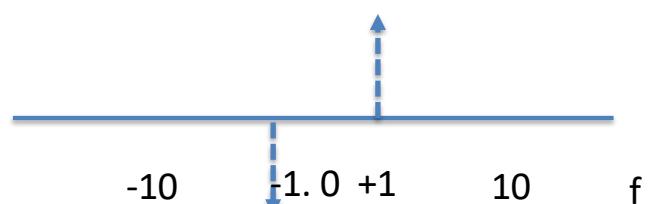
$$x(t) = 2 \cos(2 \pi 1 t)$$



$$X(f) = \delta(f+1) + \delta(f-1)$$



$$x(t) = \sin(2 \pi 1 t)$$

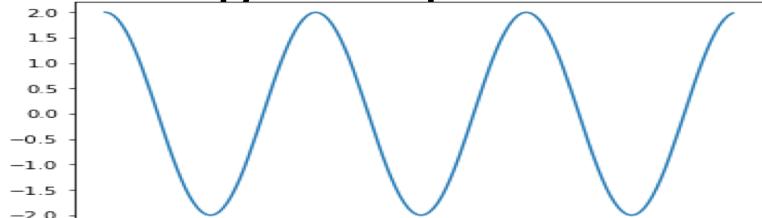


$$X(f) = -j/2 \delta(f+1) + j/2 \delta(f-1)$$

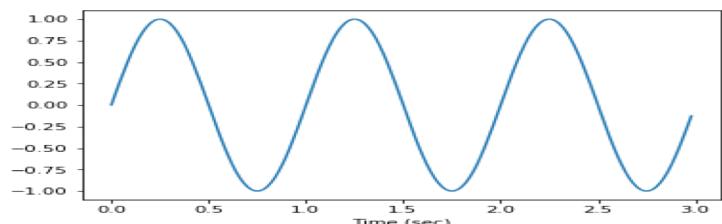
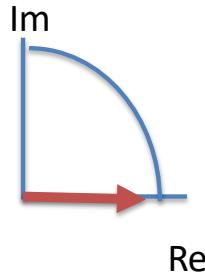


Data X

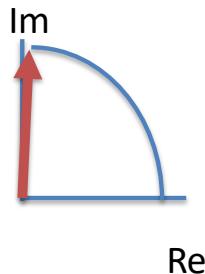
Data Signal Representation



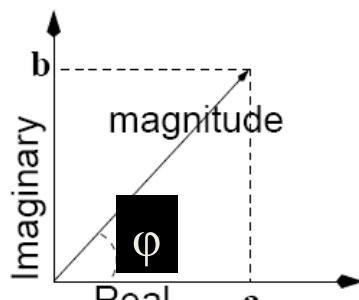
$$x(t) = 2 \cos(2 \pi 1 t + 0)$$



$$\begin{aligned} x(t) &= \sin(2 \pi 1 t) \\ &= \cos(2 \pi 1 t - \pi/2) \end{aligned}$$



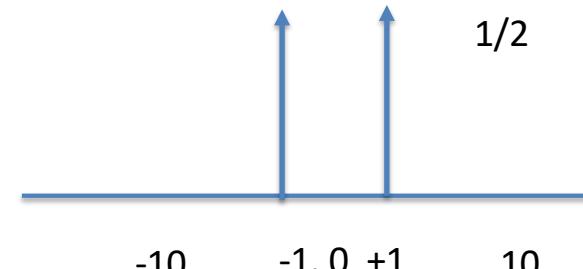
Any sinusoid: $x(t) = A \cos(2 \pi f_0 t + \varphi)$



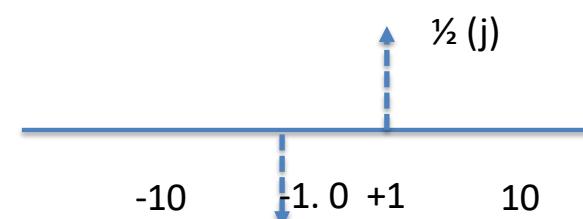
Can be represented as a phasor (vector):

$$x(t) = A e^{j\varphi} \quad A = |x(t)|$$

$$x(t) = A e^{j\varphi} = A \cos(\varphi) + j A \sin(\varphi)$$



$$X(f) = \delta(f+1) + \delta(f-1)$$

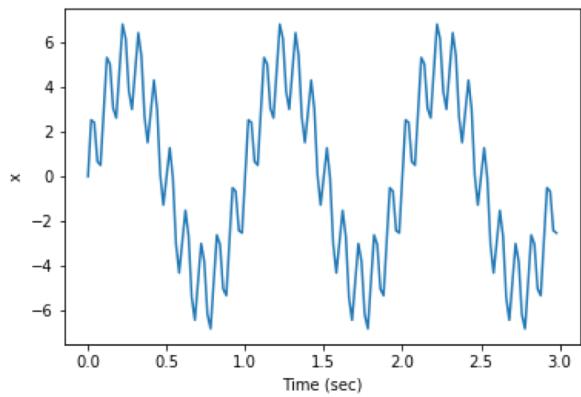


$$X(f) = -j/2 \delta(f+1) + j/2 \delta(f-1)$$

$$X(f) = \int_{-\infty}^{\infty} x(t) \times e^{-j2\pi ft} dt$$



Intuition

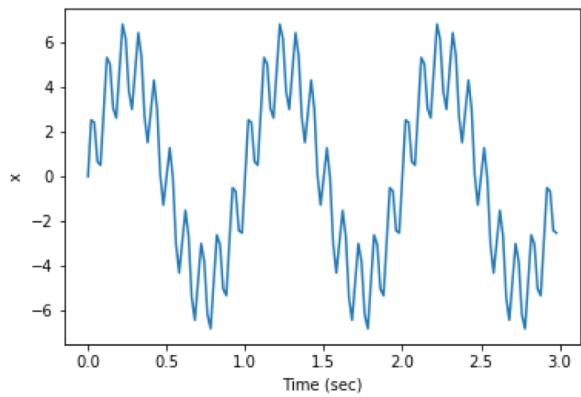


$$x(t) = 5 \cos(2 \pi 10 t) + 2 \cos(2 \pi 1 t)$$

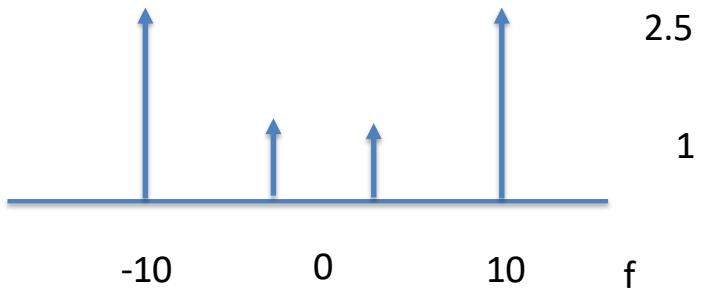
Data X

A blurred image of a data matrix with binary values (0s and 1s) representing the signal $x(t)$ over time. The matrix has approximately 20 columns and 10 rows, showing the discrete samples of the signal at regular intervals.

Intuition



$$x(t) = 5 \cos(2 \pi 10 t) + 2 \cos(2 \pi 1 t)$$



$$X(f) = 2.5[\delta(f+10) + \delta(f-10)] + 1[\delta(f+1) + \delta(f-1)]$$



Working with Numpy FFT Results: Scaling and Folding

Imports

```
In [1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

NumPy FFT Example 1

For an FFT to make sense, we need to know:

1. The data (list) = x
2. The sample rate = 50 Hz

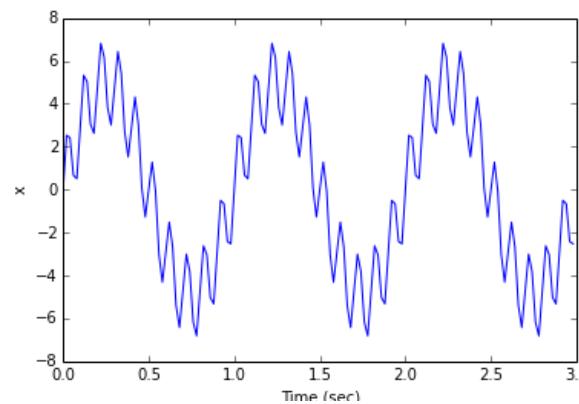
Create a Test Signal

f_s is the sampling frequency, while f is a base frequency for the signal content. We create a signal that contains components at a couple of multiples of this base frequency. Note the amplitudes here since we will be trying to extract those correctly from the FFT later.

```
In [2]: f_s = 50.0 # Hz
f = 1.0 # Hz
time = np.arange(0.0, 3.0, 1/f_s)
x = 5 * np.sin(2 * np.pi * f * time) + 2 * np.sin(10 * 2 * np.pi * f * time)
```

```
In [3]: plt.plot(time, x)
plt.xlabel("Time (sec)")
plt.ylabel("x")
```

```
Out[3]: <matplotlib.text.Text at 0x6e22b10>
```



 jedludlow / ipython_fft_example.ipynb
<https://gist.github.com/jedludlow/3919130>



NumPy FFT Example 1

Compute the FFT

The FFT and a matching vector of frequencies

len
sampling rate
fftfreq(n,Ts) returns an array
of the frequency bins

fs = 50
n = 3 * 50
fs/n = 1/3

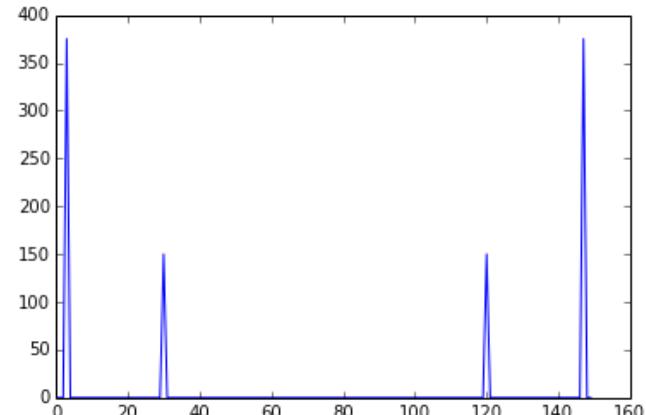
```
In [4]: fft_x = np.fft.fft(x)
n = len(fft_x)
freq = np.fft.fftfreq(n, 1/f_s)
print n
print freq
```

150

0.	0.33333333	0.66666667	1.	1.33333333
1.66666667	2.	2.33333333	2.66666667	3.
3.66666667	4.	4.33333333	4.66666667	5.
5.66666667	6.	6.33333333	6.66666667	7.
7.66666667	8.	8.33333333	8.66666667	9.
9.66666667	10.	10.33333333	10.66666667	11.
11.66666667	12.	12.33333333	12.66666667	13.
13.66666667	14.	14.33333333	14.66666667	15.
15.66666667	16.	16.33333333	16.66666667	17.
17.66666667	18.	18.33333333	18.66666667	19.
19.66666667	20.	20.33333333	20.66666667	21.
21.66666667	22.	22.33333333	22.66666667	23.
23.66666667	24.	24.33333333	24.66666667	-25.
-24.33333333	-24.	-23.66666667	-23.33333333	-23.
-22.33333333	-22.	-21.66666667	-21.33333333	-21.
-20.33333333	-20.	-19.66666667	-19.33333333	-19.
-18.33333333	-18.	-17.66666667	-17.33333333	-17.
-16.33333333	-16.	-15.66666667	-15.33333333	-15.
..
-14.33333333	-14.	-13.66666667	-13.33333333	-13.
-12.33333333	-12.	-11.66666667	-11.33333333	-11.
-10.33333333	-10.	-9.66666667	-9.33333333	-9.
-8.33333333	-8.	-7.66666667	-7.33333333	-7.
-6.33333333	-6.	-5.66666667	-5.33333333	-5.
-4.33333333	-4.	-3.66666667	-3.33333333	-3.
-2.33333333	-2.	-1.66666667	-1.33333333	-1.
-0.33333333				

1017

```
In [5]: plt.plot(np.abs(fft_x))
Out[5]: [
```



150 points in the original data
150 points in the fft result



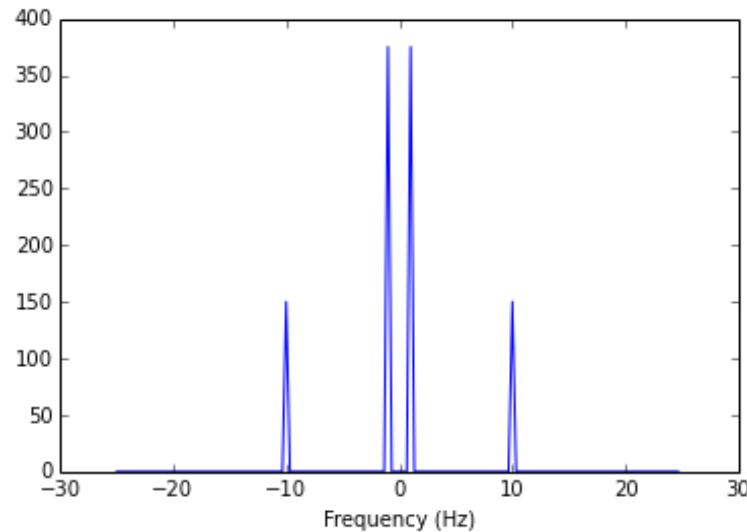
Swap Half Spaces

Note that frequencies in the FFT and the freq vector go from zero to some larger positive number then from a large negative number back toward zero. We can swap that so that the DC component is in the center of the vector while maintaining a two-sided spectrum.

```
In [6]: fft_x_shifted = np.fft.fftshift(fft_x)
freq_shifted = np.fft.fftshift(freq)
```

```
In [7]: plt.plot(freq_shifted, np.abs(fft_x_shifted))
plt.xlabel("Frequency (Hz)")
```

```
Out[7]: <matplotlib.text.Text at 0x70a2b50>
```



NumPy FFT Example 1



NumPy FFT Example 1

$$\begin{aligned}fs &= 50 \\n &= 3 * 50 \\fs/n &= 1/3\end{aligned}$$

Fold Negative Frequencies and Scale

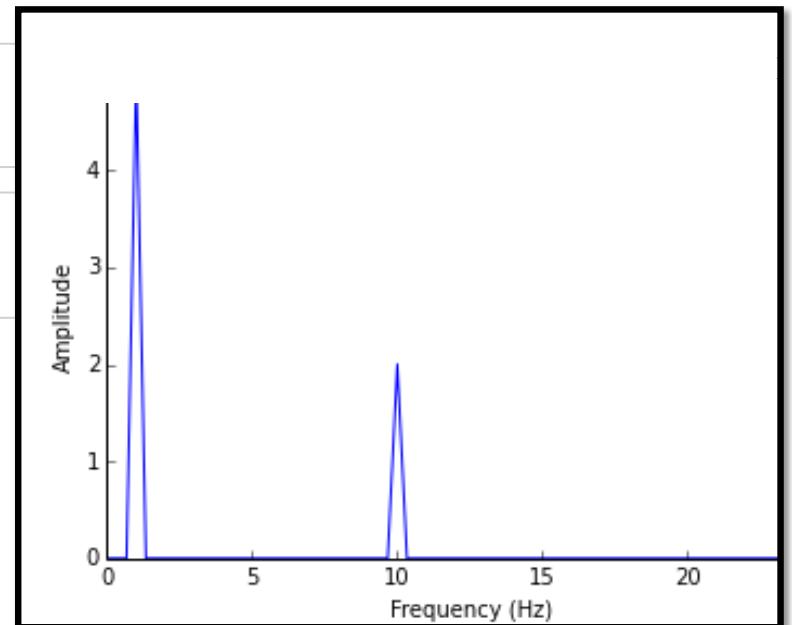
It's actually more common to look at just the first half of the unshifted FFT and frequency vectors and fold all the amplitude information into the positive frequencies. Furthermore, to get amplitude right, we must normalize by the length of the original FFT. Note the factor of $2 / n$ in the following which accomplishes both the folding and scaling.

```
In [8]: half_n = np.ceil(n/2.0)
        fft_x_half = (2.0 / n) * fft_x[:half_n]
        freq_half = freq[:half_n]
```

```
In [9]: plt.plot(freq_half, np.abs(fft_x_half))
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude")
```

Out[9]: <matplotlib.text.Text at 0x6edc150>

This is just slicing of the arrays.



Data X

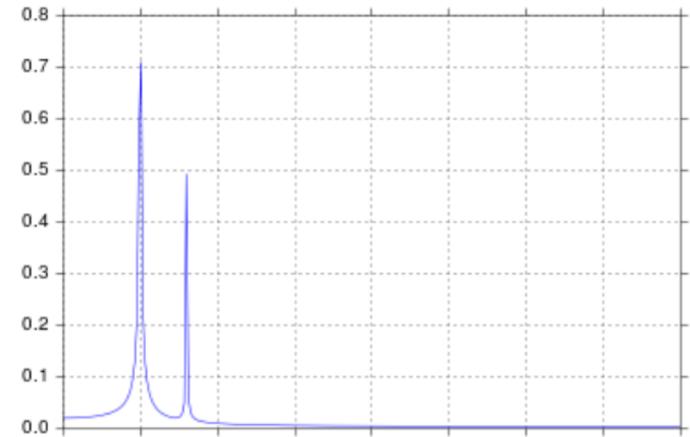
A SciPy Example

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack

# Number of samplepoints
N = 600
# sample spacing
T = 1.0 / 800.0
x = np.linspace(0.0, N*T, N)
y = np.sin(50.0 * 2.0*np.pi*x) + 0.5*np.sin(80.0 * 2.0*np.pi*x)
yf = scipy.fftpack.fft(y)
xf = np.linspace(0.0, 1.0/(2.0*T), N/2)

fig, ax = plt.subplots()
ax.plot(xf, 2.0/N * np.abs(yf[:N//2]))
plt.show()
```

I get what I believe to be very reasonable output.



0 0 0 1 0 1 1 1 1 1
1 0 1 1 0 0 0 0 0 0
1 Data X 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

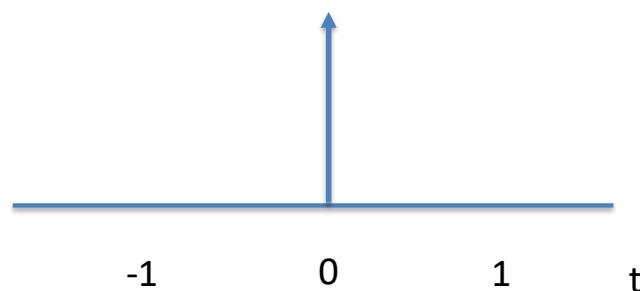
Why Does the FFT Fold over at $f_s/2$?



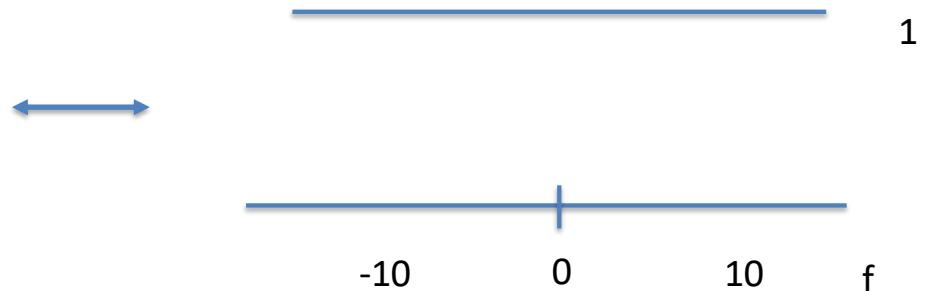
Example: impulse or “delta” function

- FT of delta function:

$$F(\delta(x)) = \int_{-\infty}^{\infty} \delta(x) e^{-j2\pi fx} dx = e^0 = 1$$



Like a very narrow pulse

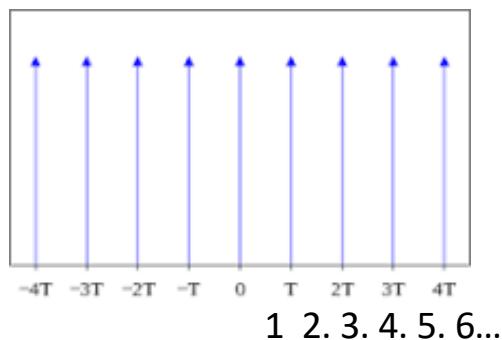


Results in a ‘very’ broad spectrum sinc



But wait, data is discrete, but Fourier continuous

This is the
comb
function:



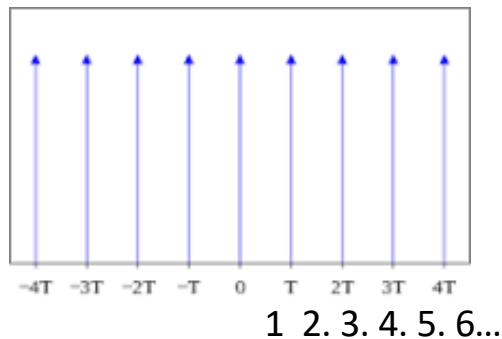
$$\text{III}(t) = \sum_{k=-\infty}^{\infty} \delta(t-k)$$

What would the Fourier transform of this function be?



But wait, data is discrete, but Fourier continuous

This is the
comb
function:



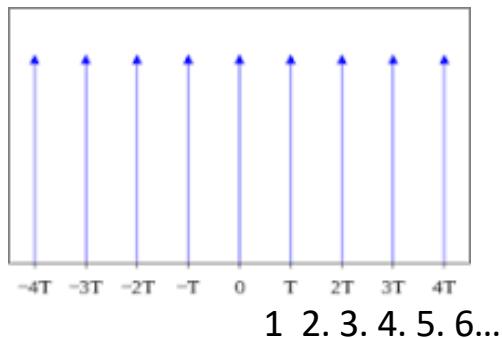
$$\text{III}(t) = \sum_{k=-\infty}^{\infty} \delta(t-k)$$

Answer: Fourier of $\text{Comb}(t) = \text{Comb}(f)$



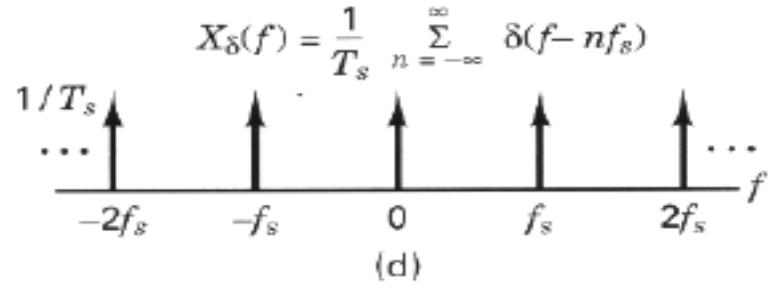
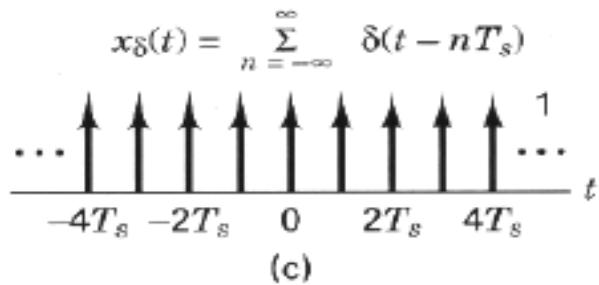
But wait, that was discrete, and Fourier was continuous

This is the comb function:



Answer: Fourier of Comb(t) = Comb(f)

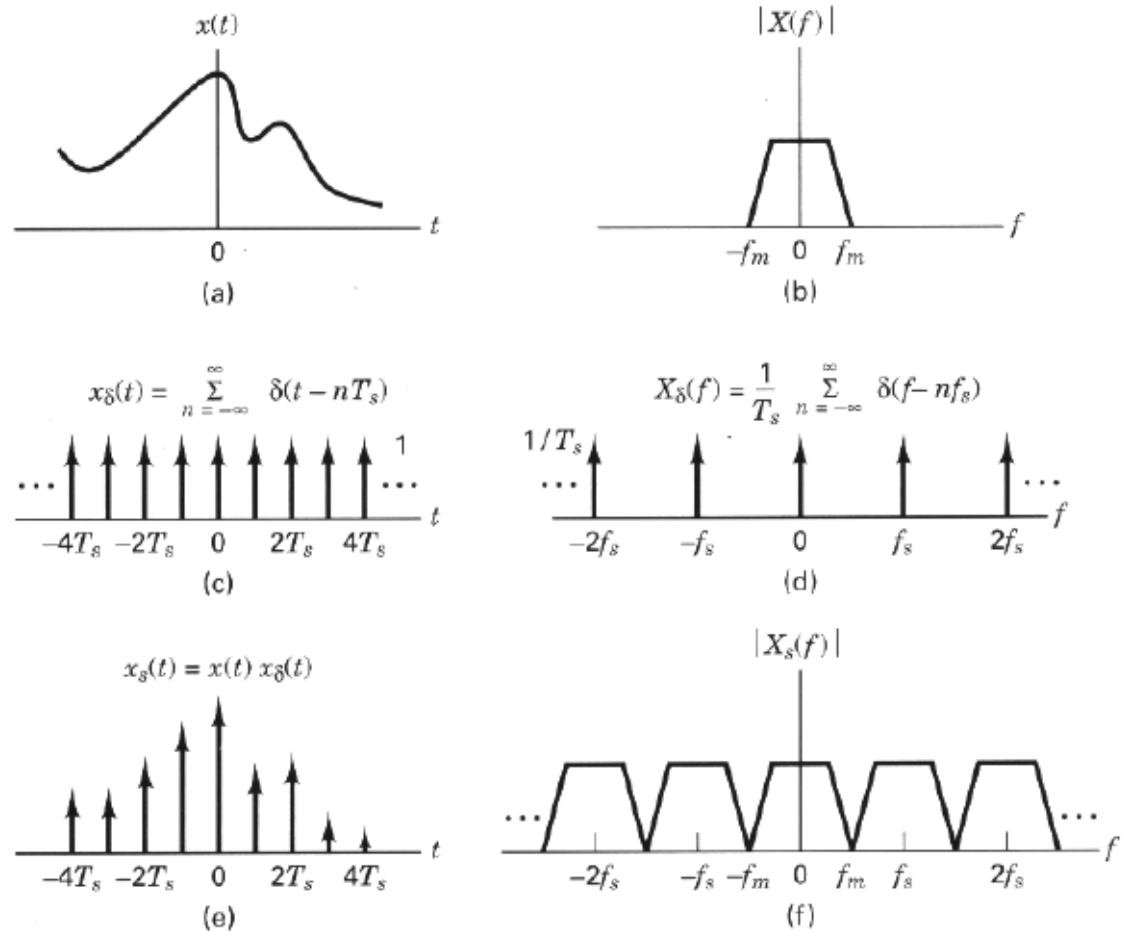
$$\text{III}_T(t) \stackrel{\text{def}}{=} \sum_{k=-\infty}^{\infty} \delta(t - kT) = \frac{1}{T} \text{III}\left(\frac{t}{T}\right)$$



By understanding the properties of the Comb, we can map between the time signal and the frequency samples

Sampled with time T, and with N samples means:

Each frequency bin in the FFT = F_s/N



<http://docs.exdat.com/docs/index-44055.html>



Rectangular shape code sample

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.array([1,1,1,1,0,0,0,0,0,0,0,0,0,0,0])
x = x/16.0

print (x)
print (np.fft.fft(x))

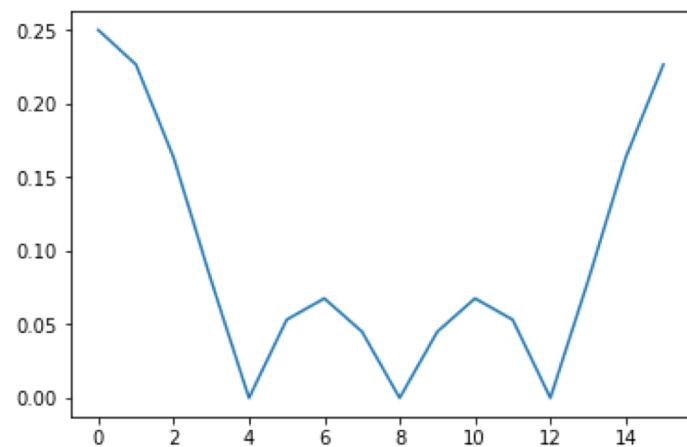
plt.plot(abs(np.fft.fft(x)))
```

[0.0625 0.0625 0.0625 0.0625 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0]

[0.25000000+0.j 0.18835436-0.12585436j 0.06250000-0.15088835j -0.01551893-0.07801893j
0.00000000+0.j 0.05213058+0.01036942j 0.06250000-0.02588835j 0.02503399-0.03746601j
0.00000000+0.j 0.02503399+0.03746601j 0.06250000+0.02588835j 0.05213058-0.01036942j
0.00000000+0.j -0.01551893+0.07801893j 0.06250000+0.15088835j 0.18835436+0.12585436j]



1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.

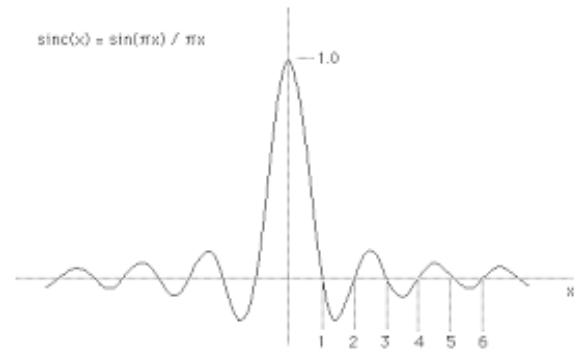
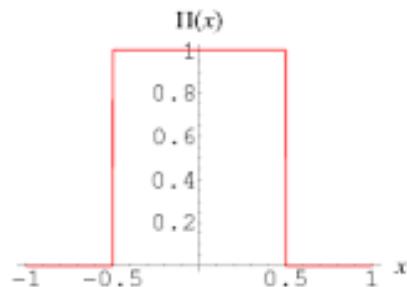


What does this mean?



A Pulse In Time

This example is the famous $\text{rect}(x)$ which transforms to $\text{Sinc}(f)$



$$\text{rect}(t) = \Pi(t) := \begin{cases} 0 & \text{if } |t| > \frac{1}{2} \\ \frac{1}{2} & \text{if } |t| = \frac{1}{2} \\ 1 & \text{if } |t| < \frac{1}{2}. \end{cases}$$

Which is right?

$$\text{sinc}(x) = \frac{\sin(x)}{x} .$$

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

Zero crossing

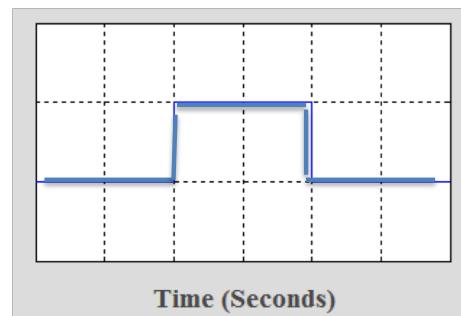
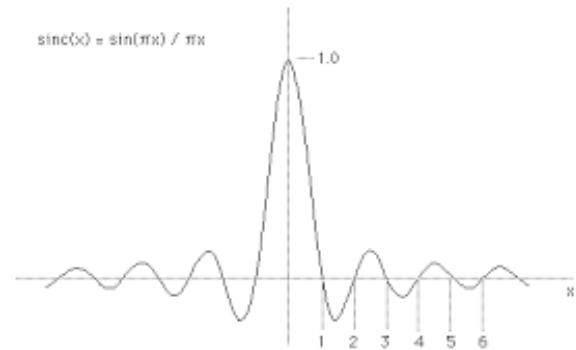
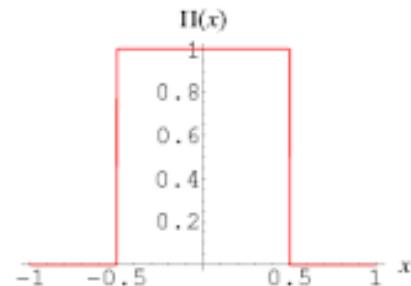
Pi radians

integers

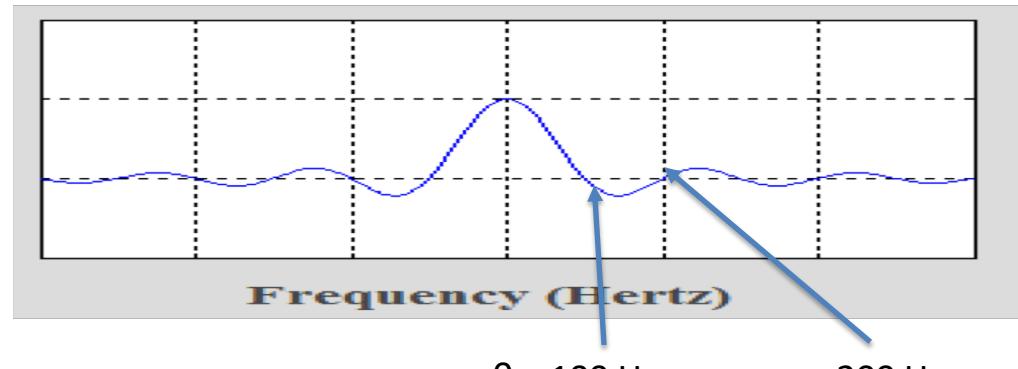
Data X

A Pulse In Time

This example is the famous $\text{rect}(x)$
which transforms to $\text{Sinc}(f)$

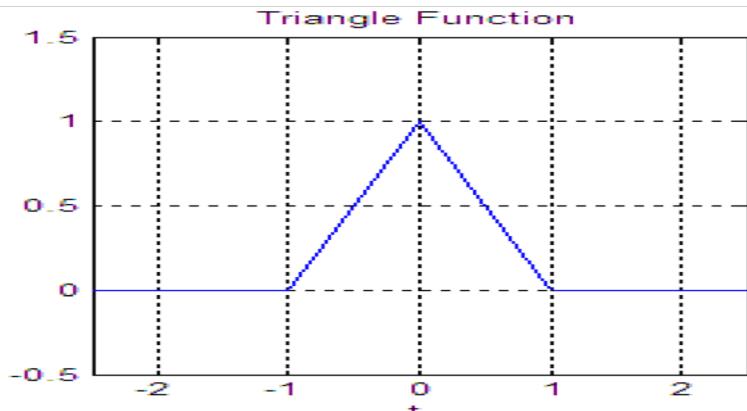


Width = 0.01 seconds

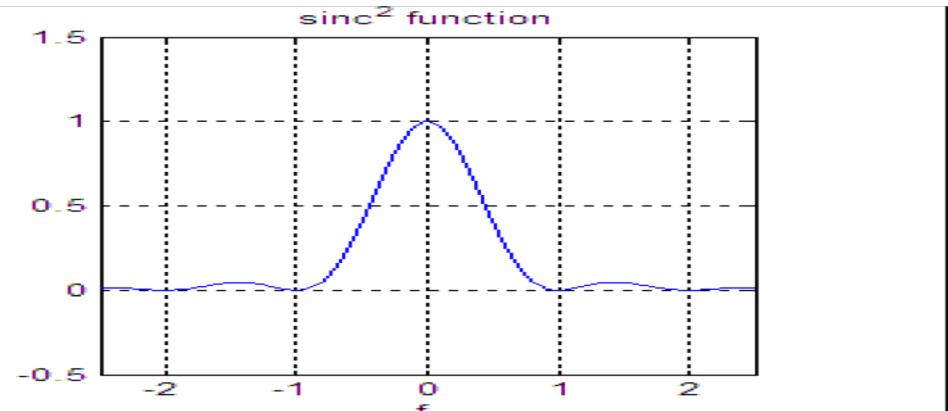


Another Common Example

Fourier of $\Delta(t)$



1 S

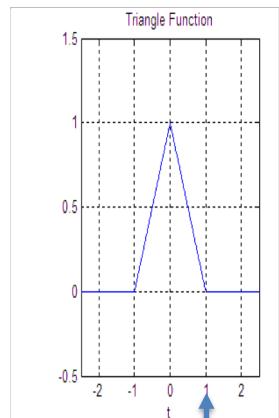


1 Hz

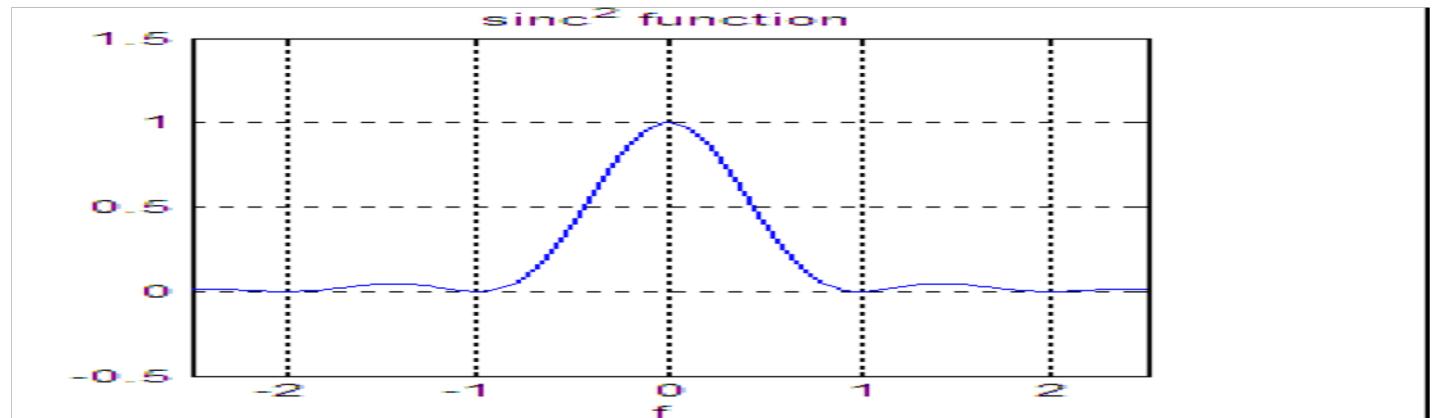


Another Common Example

What is Fourier of $\Delta(t)$



1 mS

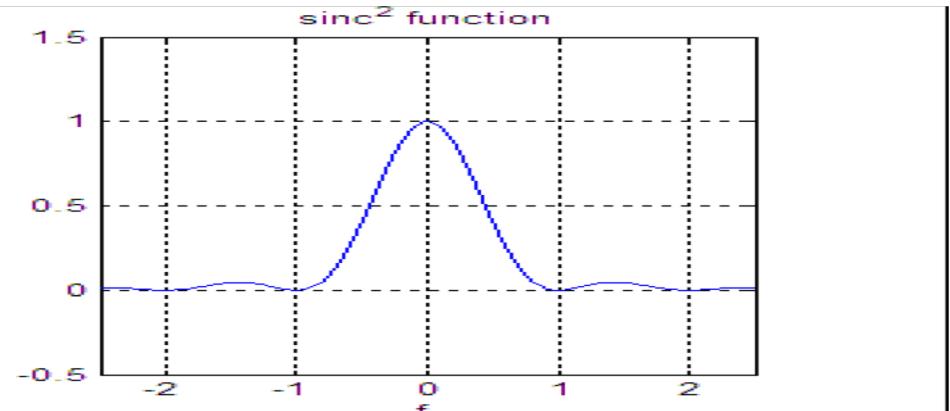
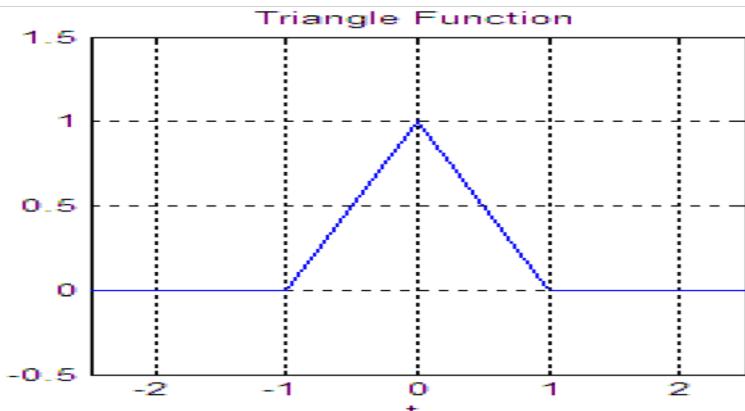


1000 Hz



Another Common Example

Fourier of $\Delta(t)$



1 mS

1000 Hz

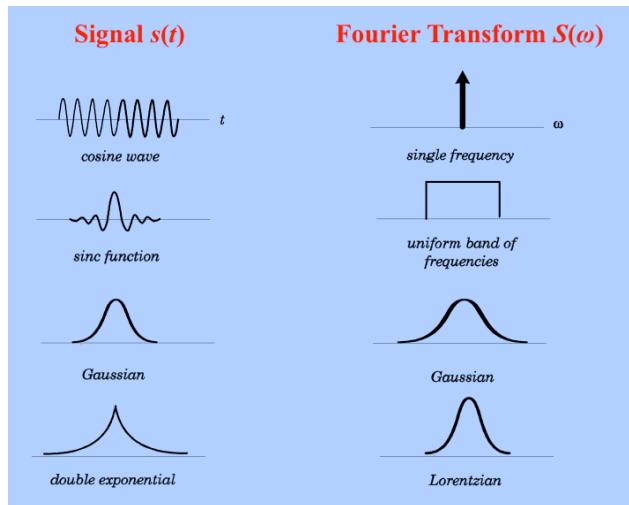
Scaling $f(ax)$

$$\frac{1}{|a|} F\left(\frac{u}{a}\right)$$



Properties of Fourier Transform

Common Closed Form Transforms and Properties



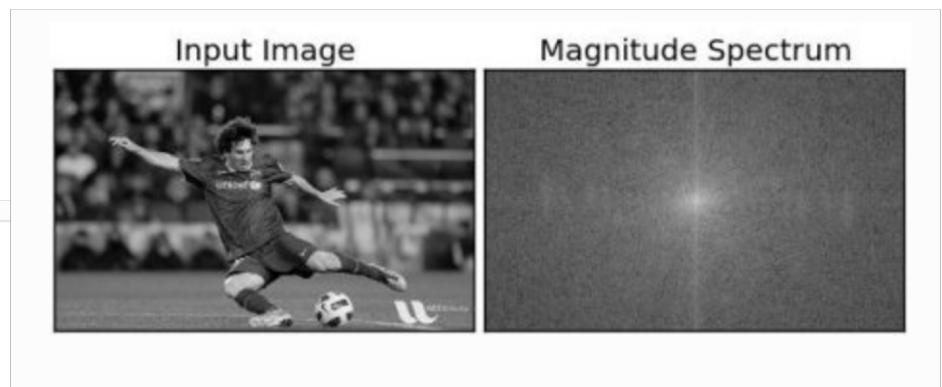
| | Spatial Domain (x) | Frequency Domain (u) |
|-----------------|-------------------------|---|
| Linearity | $c_1 f(x) + c_2 g(x)$ | $c_1 F(u) + c_2 G(u)$ |
| Scaling | $f(ax)$ | $\frac{1}{ a } F\left(\frac{u}{a}\right)$ |
| Shifting | $f(x - x_0)$ | $e^{-i2\pi u x_0} F(u)$ |
| Symmetry | $F(x)$ | $f(-u)$ |
| Conjugation | $f^*(x)$ | $F^*(-u)$ |
| Convolution | $f(x) * g(x)$ | $F(u)G(u)$ |
| Differentiation | $\frac{d^n f(x)}{dx^n}$ | $(i2\pi u)^n F(u)$ |

Getting image data and 2-d FFTs are possible with NumPy and Open CV

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

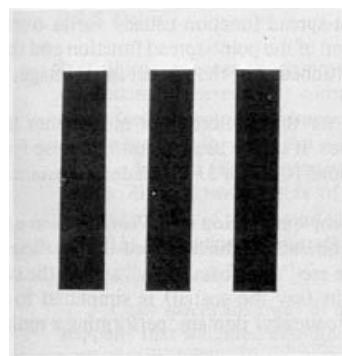
img = cv2.imread('messi5.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

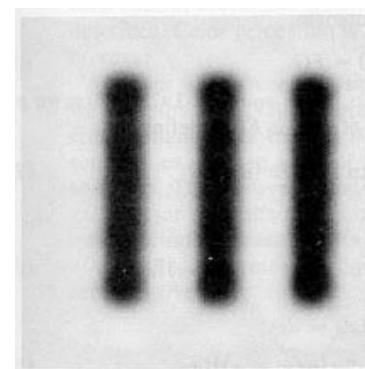


How do frequencies show up in an image?

- Low frequencies correspond to slowly varying information (e.g., continuous surface).
- High frequencies correspond to quickly varying information (e.g., edges)



Original Image

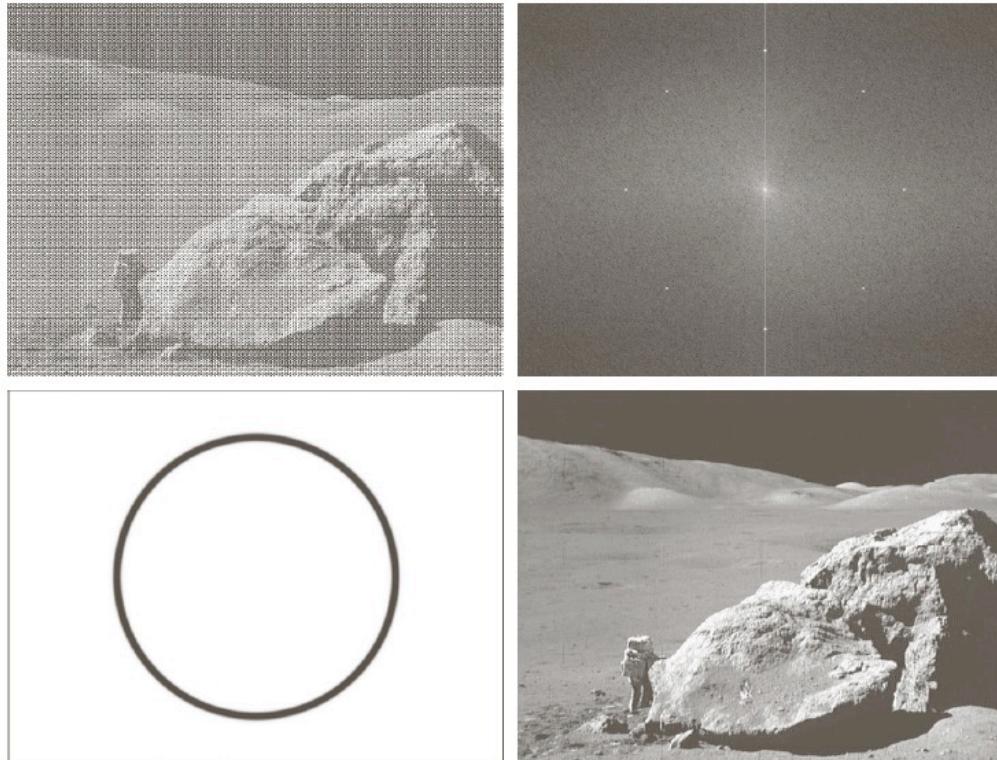


Low-passed

Naveen Sihag



Example of noise reduction using FT

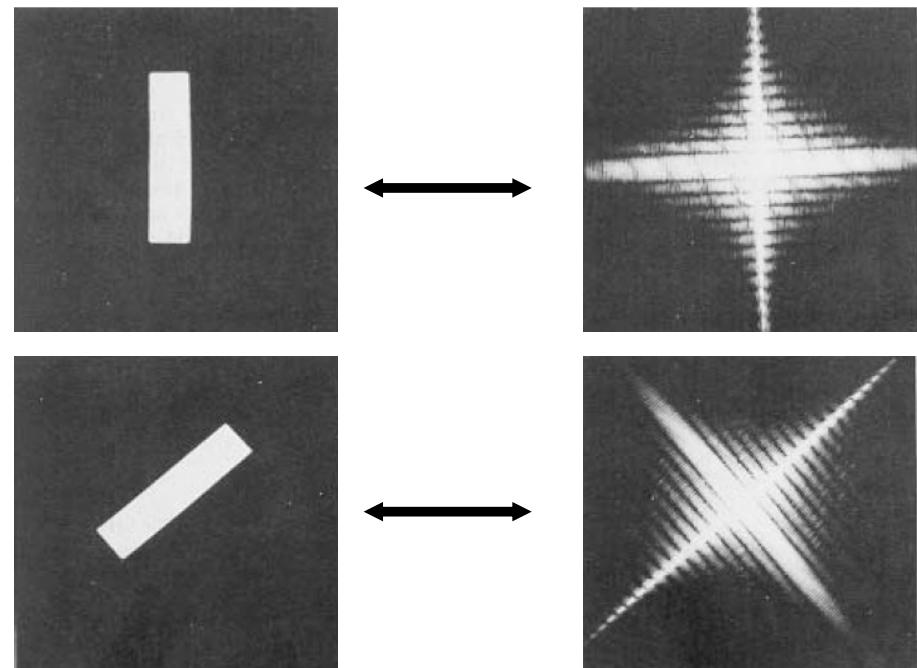


Naveen Sihag



DFT Properties: (5) Rotation

- Rotating $f(x,y)$ by θ rotates $F(u,v)$ by θ

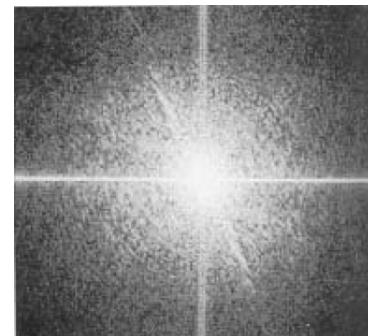


Naveen Sihag

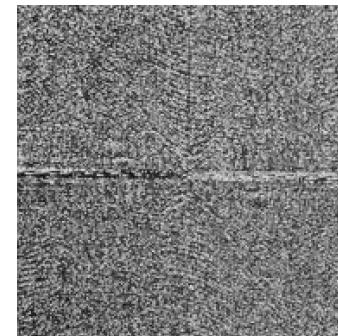


Magnitude and Phase of DFT

- What is more important?



magnitude



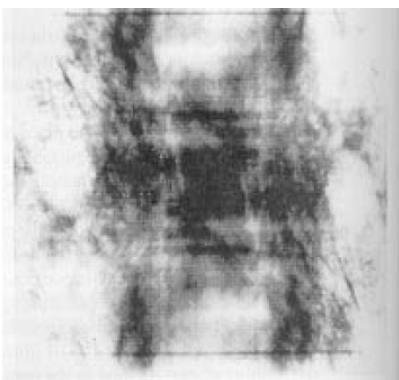
phase

- **Hint:** use inverse DFT to reconstruct the image using magnitude or phase only information

Naveen Sihag



Magnitude and Phase of DFT (cont'd)



Reconstructed image using
magnitude only
(i.e., magnitude determines the
contribution of each component!)

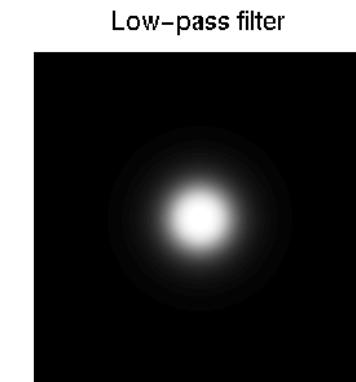
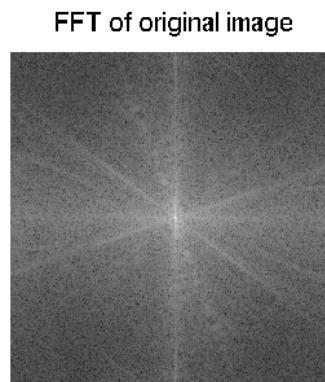
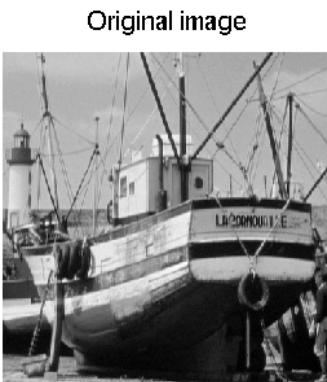


Reconstructed image using
phase only
(i.e., phase determines
which components are present!)

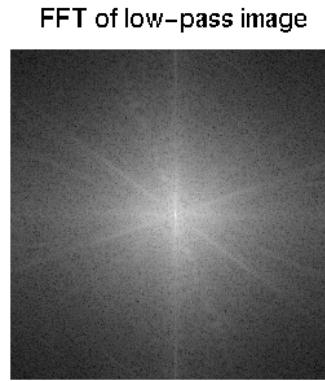
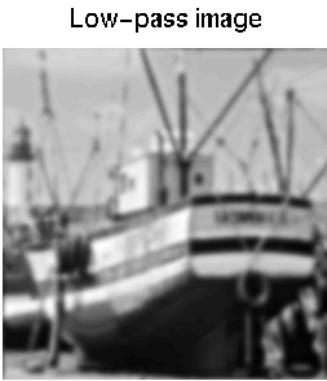
Naveen Sihag



Low-pass Filtering



Let the low frequencies pass and eliminating the high frequencies.



Generates image with overall shading, but not much detail

S. Narasimhan,
Computer Vision

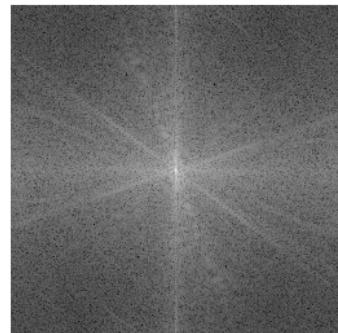


High-pass Filtering

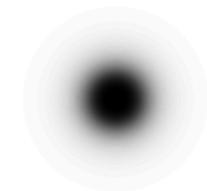
Original image



FFT of original image



High-pass filter

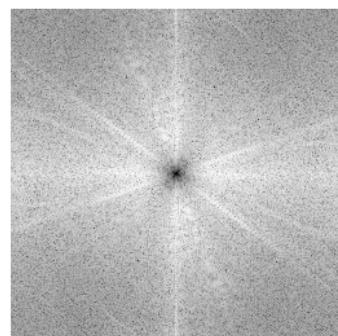


Lets through the high frequencies (the detail), but eliminates the low frequencies (the overall shape). It acts like an edge enhancer.

High-pass image



FFT of high-pass image



S. Narasimhan,
Computer Vision

Data X

End of Section

Data X