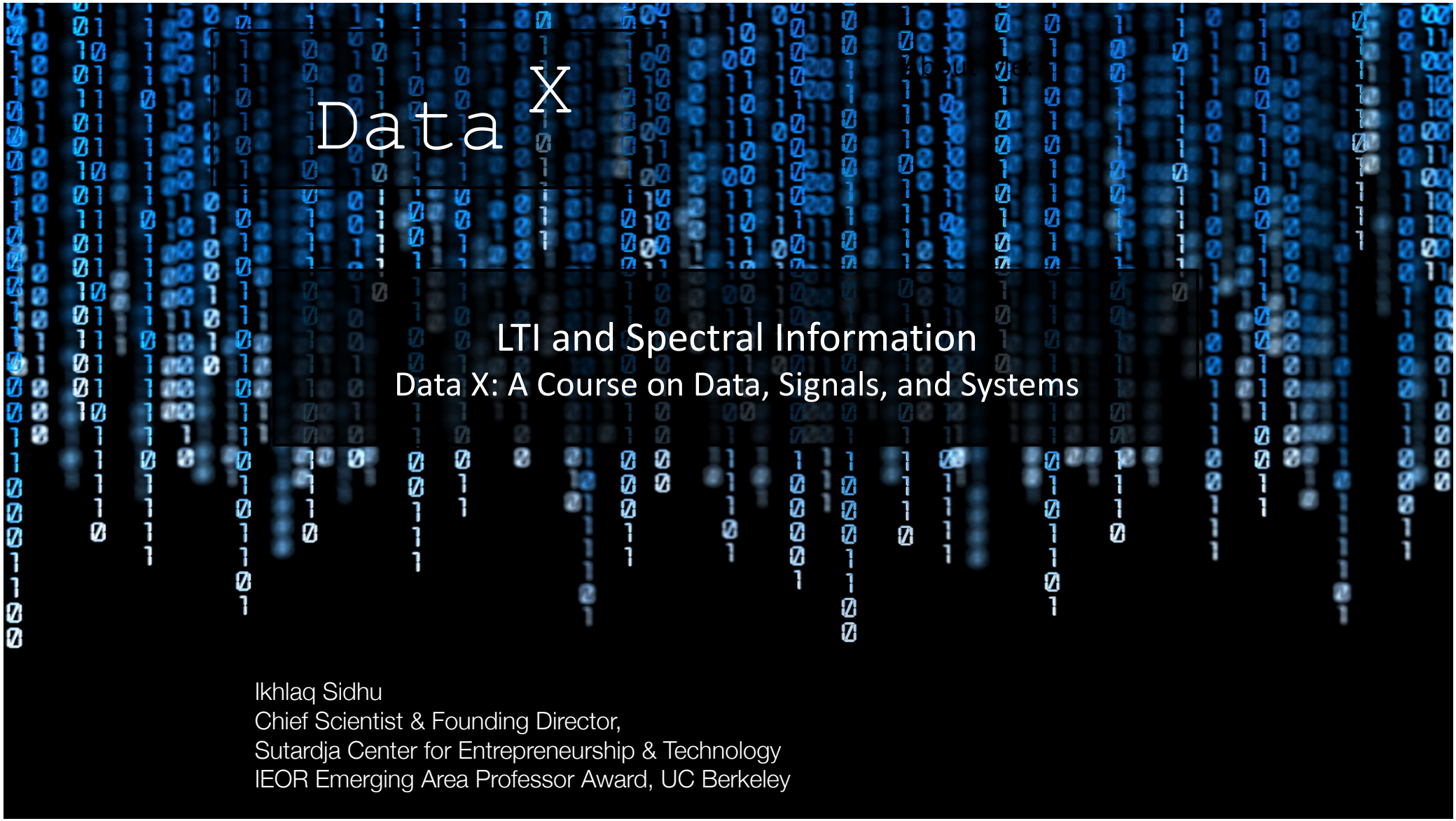


LTI and Spectral Information

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

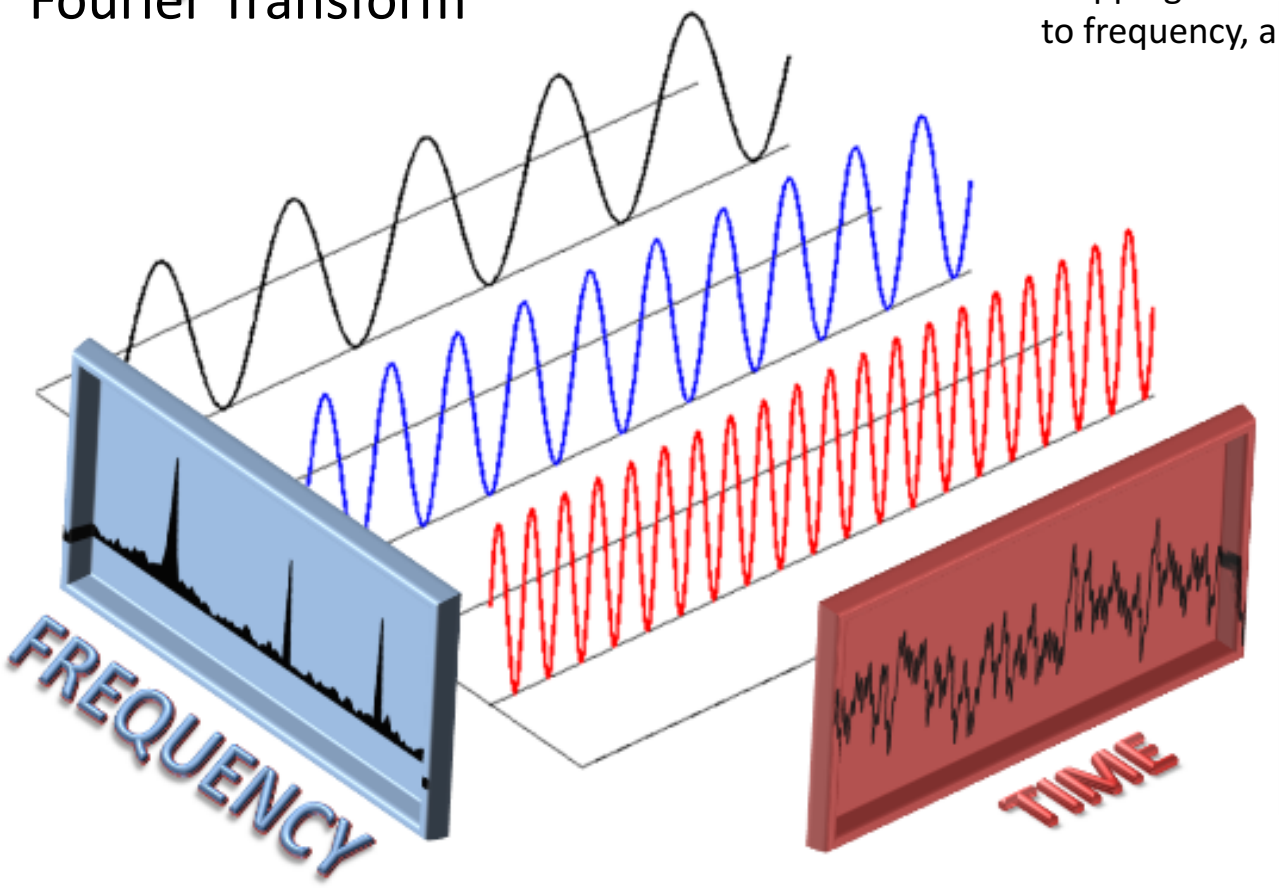
Jean Baptiste Joseph Fourier (1768-1830)

- 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

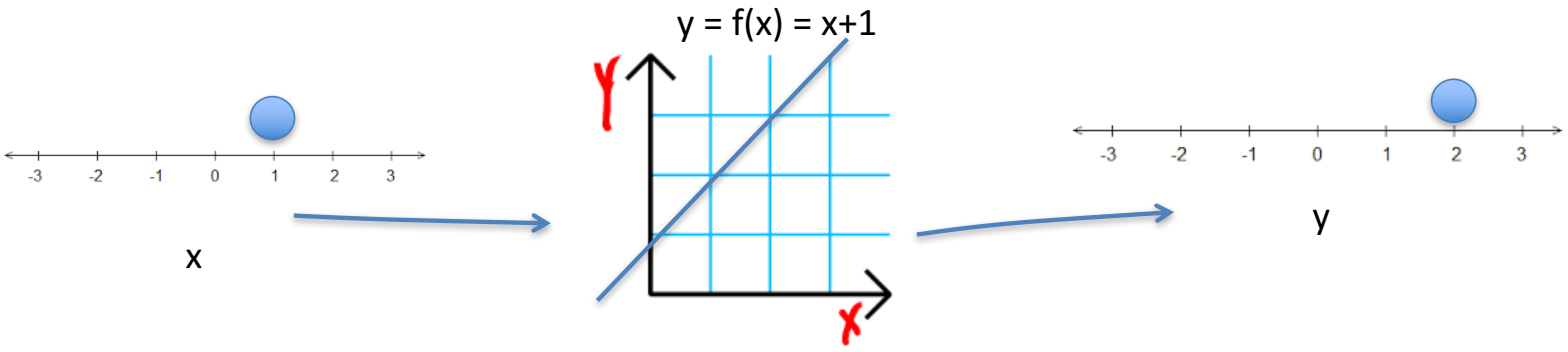
Fourier Transform

Mapping from a time
to frequency, and back.

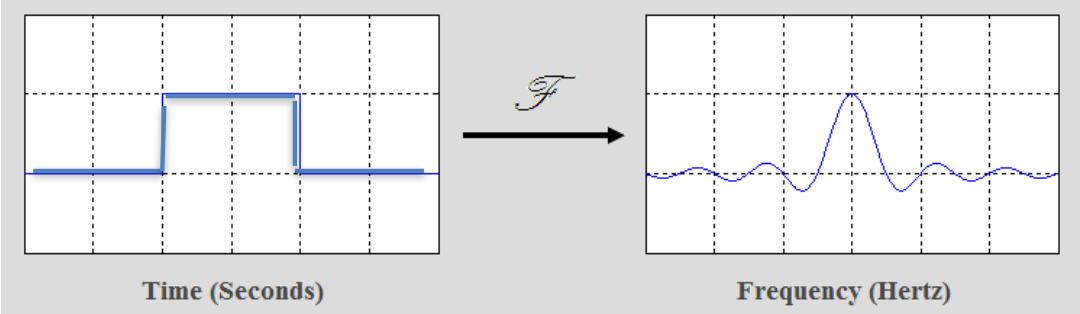


groups.csail.mit.edu/netmit/wordpress/projects/sparse-fourier-transform/

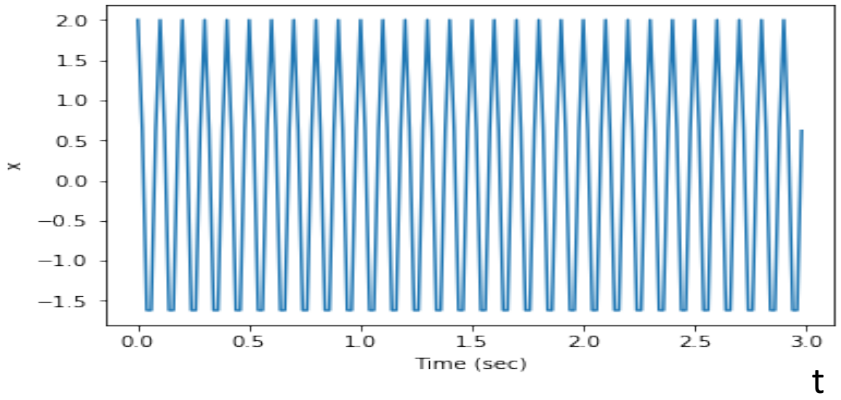
Fourier Transform
The most famous of mathematical transforms



A Transform maps a function to another function



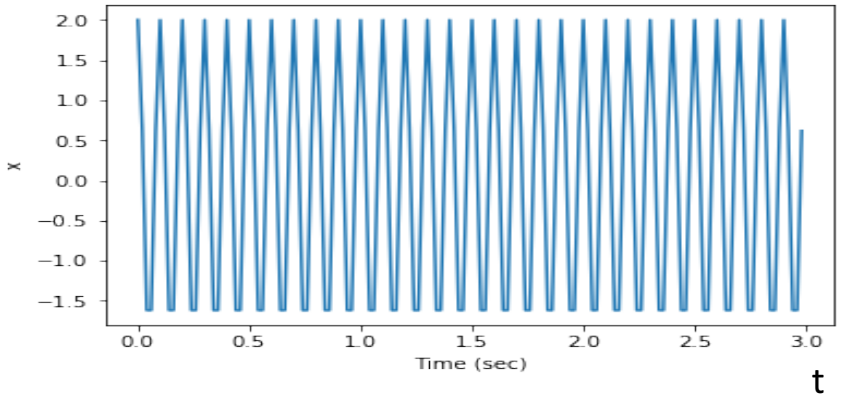
Intuition



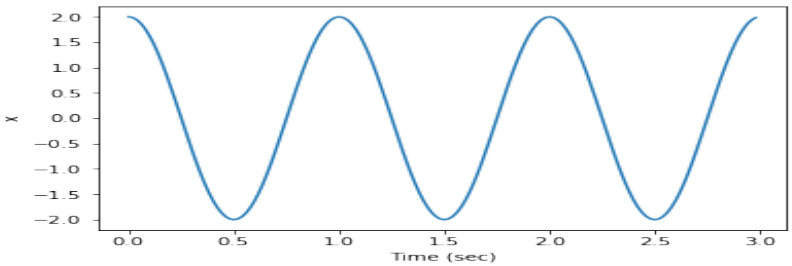
$x(t) = 2 \cos(2 \pi 10 t)$

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

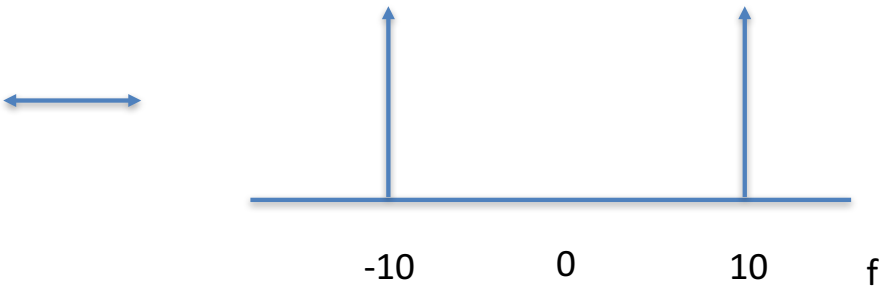
Intuition



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

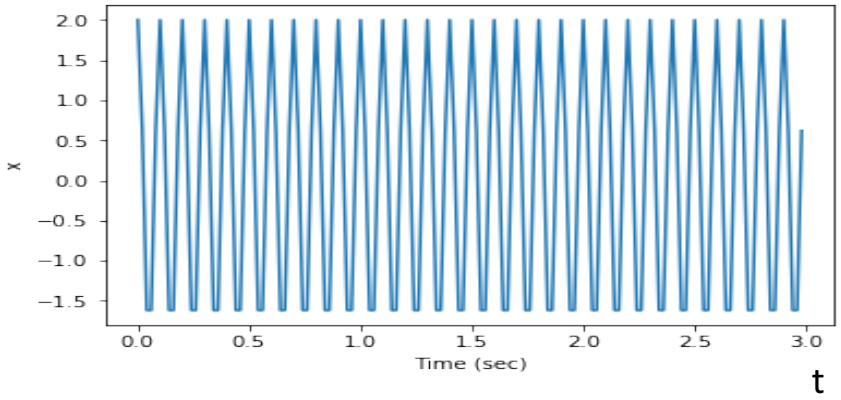


$x(t) = 2 \cos(2 \pi 1 t)$

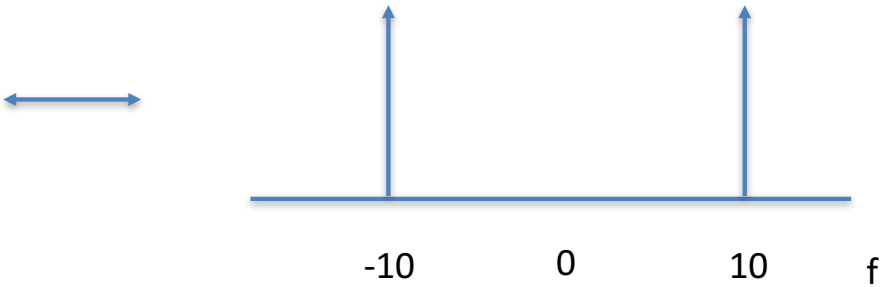


$X(f) = \delta (f+10) + \delta(x-10)$

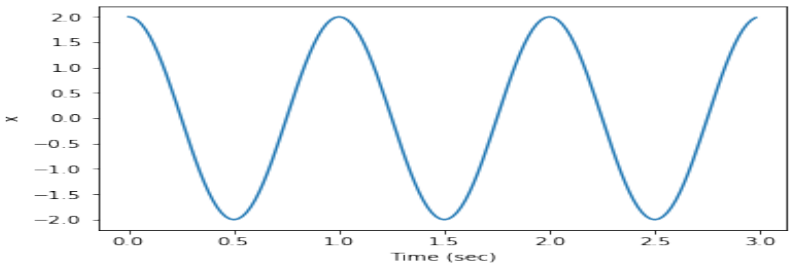
Intuition



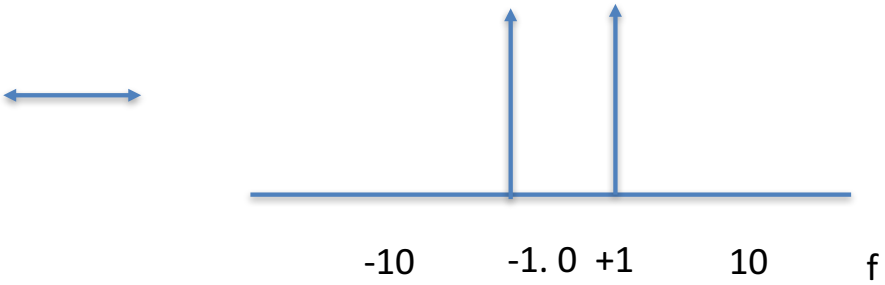
$x(t) = 2 \cos(2 \pi 10 t)$



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

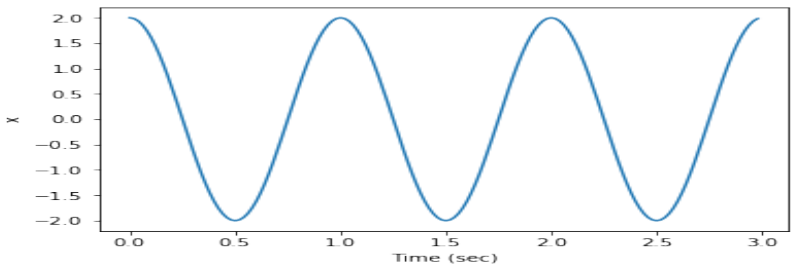


$x(t) = 2 \cos(2 \pi 1 t)$

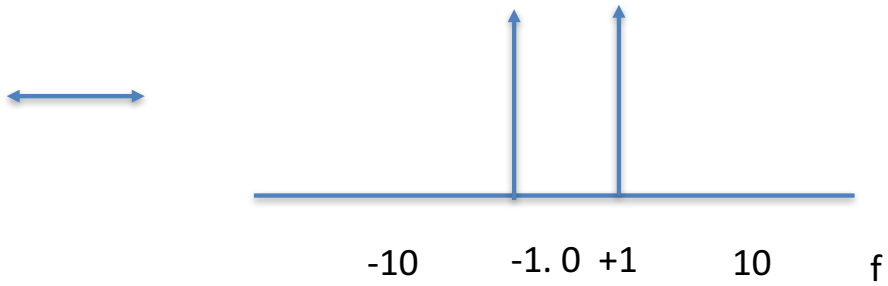


$X(f) = \delta (f+1) + \delta(x-1)$

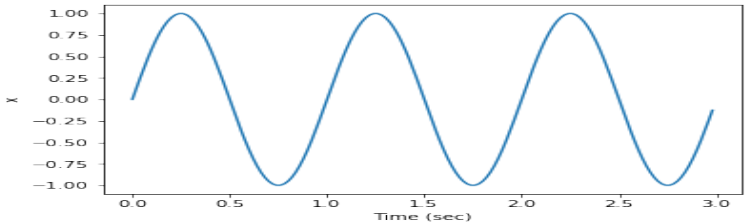
Intuition



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



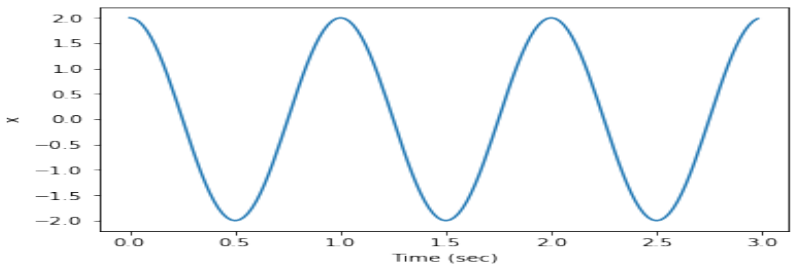
$X(f) = \delta (f+1) + \delta(x-1)$



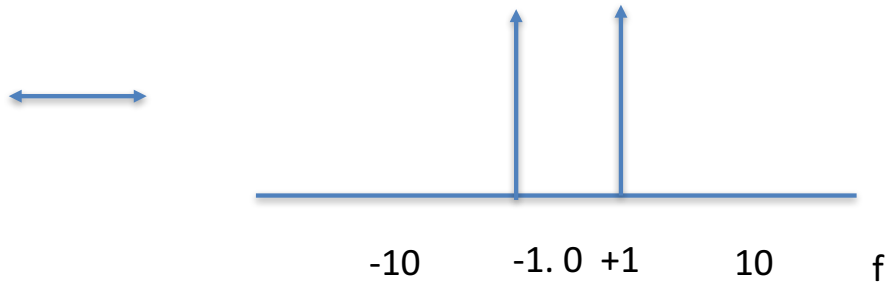
$x(t) = \sin (2 \pi 1 t)$

Intuition

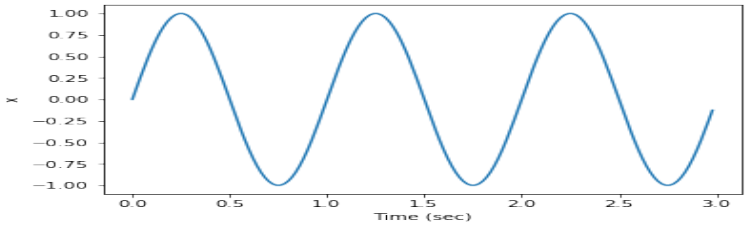
Magnitude is same
Phase is different



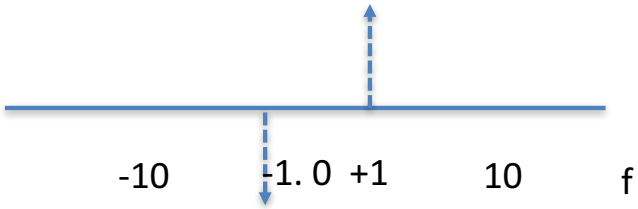
$x(t) = 2 \cos(2 \pi 1 t)$



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

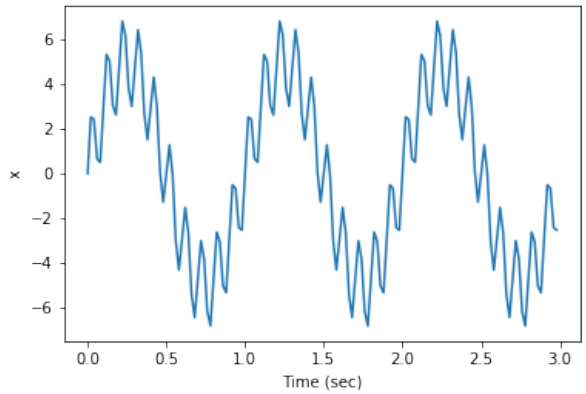


$x(t) = \sin(2 \pi 1 t)$



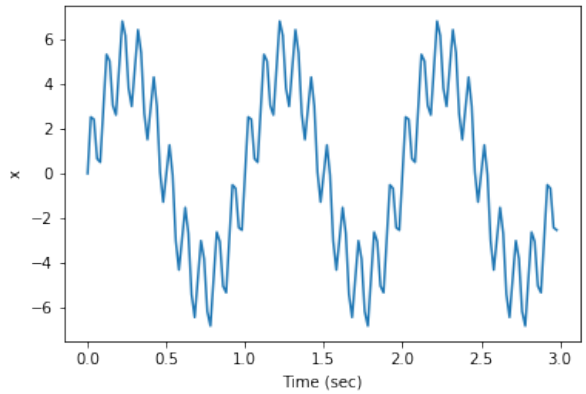
$X(f) = -j/2\ \delta (f+1) + j/2\ \delta(x-1)$

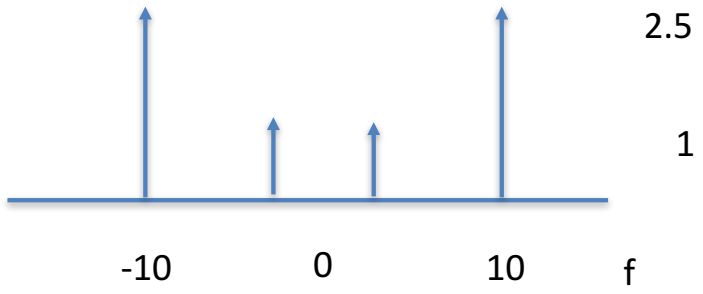
Intuition



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

Intuition





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

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

How do we get these transformations?

$$X(f) = \int_{-\infty}^{\infty} x(t) \times e^{-i2\pi ft}\, dt$$

- x or t = time, f(x) or x(t) is a function of time
- i or j are for imaginary numbers. Note X and x are both complex.
- f or s is often used for frequency, so F(s) is a function in frequency
- Integrate (average) over all time (for every given frequency)

$$f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi sx} ds$$

Remember that

$$cos(\theta) + i \times sin(\theta) = e^{i\theta}$$

$$cos(2\pi ft) + i \times sin(2\pi ft) = e^{i2\pi ft}$$

And this is the unit circle

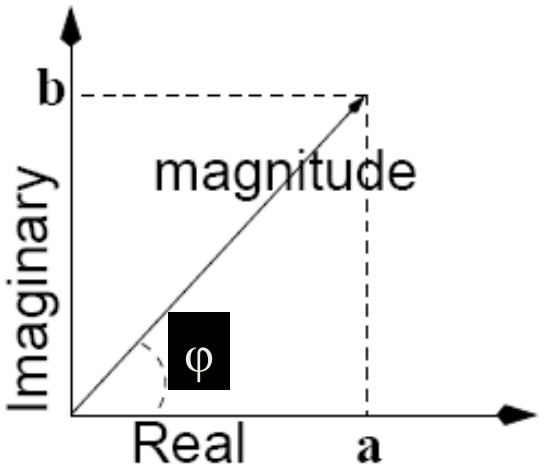
$$e^{-j2\pi ft} = \cos(2\pi fx) - j\sin(2\pi fx)$$

Mathematical Background: Complex Numbers

- Magnitude-Phase (i.e.,vector) representation

$$x = a + jb, \text{ where } j = \sqrt{-1}$$

Magnitude:



$$|x| = \sqrt{a^2 + b^2}$$

$$\phi(x) = \tan^{-1}(b/a)$$

Phase – Magnitude notation:

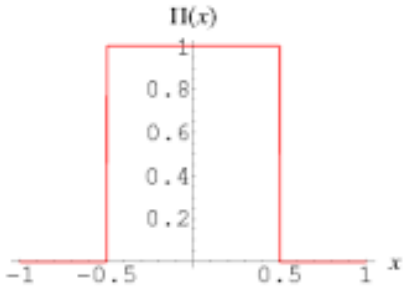
$$x = |x|e^{j\phi(x)}$$

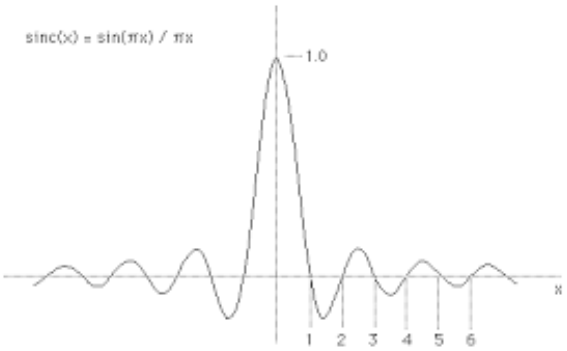
And this is the unit circle

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

A Pulse In Time

This example is the famous rect(x)
which transforms to Sinc(f)





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

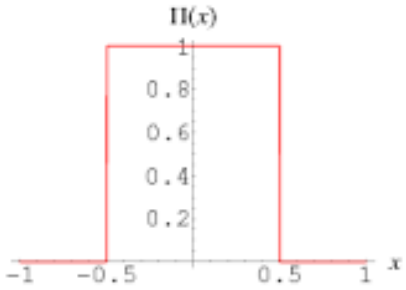
Which is right?

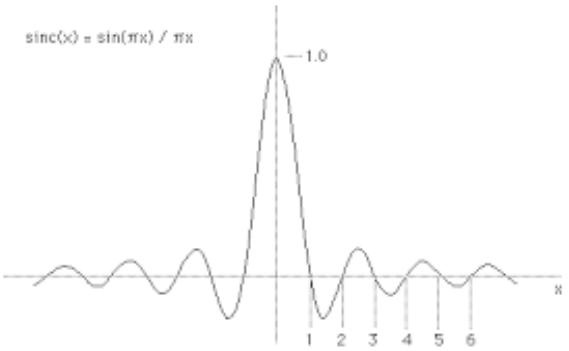
$$\mathrm{sinc}(x) = \frac{\sin(x)}{x}\,.$$

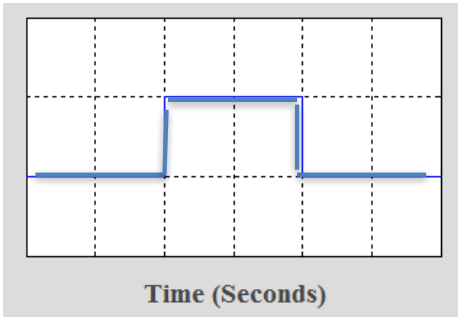
$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}\,.$$

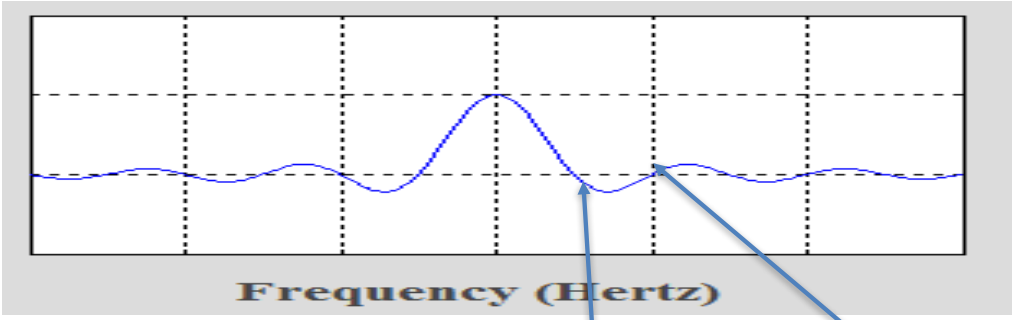
A Pulse In Time

This example is the famous rect(x) which transforms to Sinc(f)





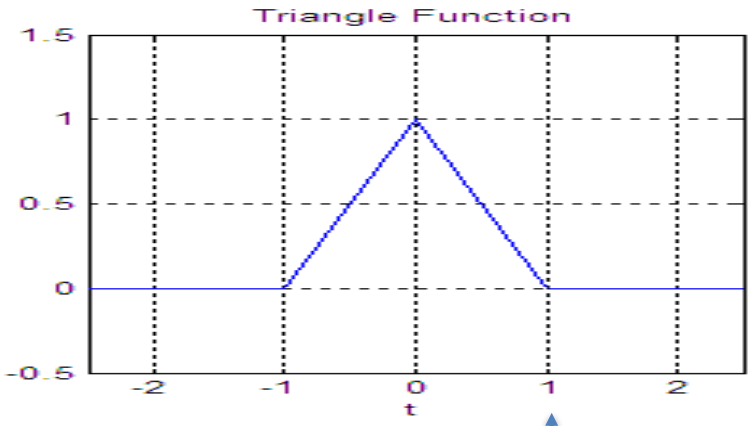




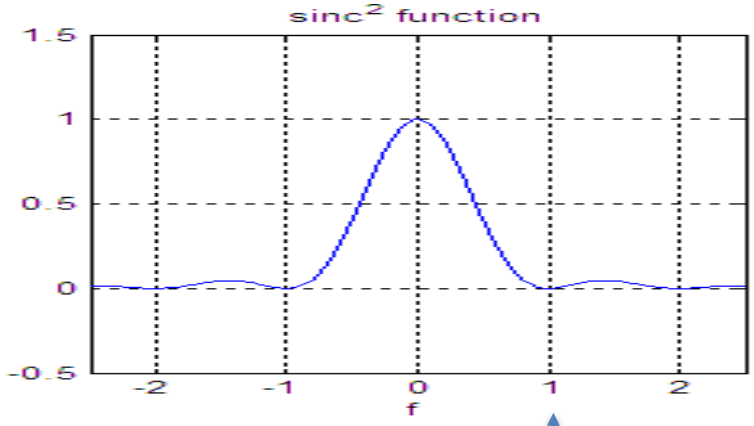
Width = 0.01 seconds

Another Common Example

Fourier of △(t)



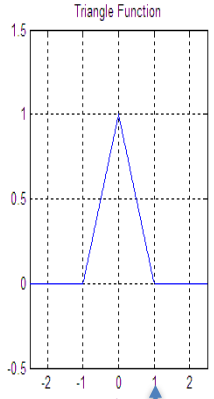
1 S



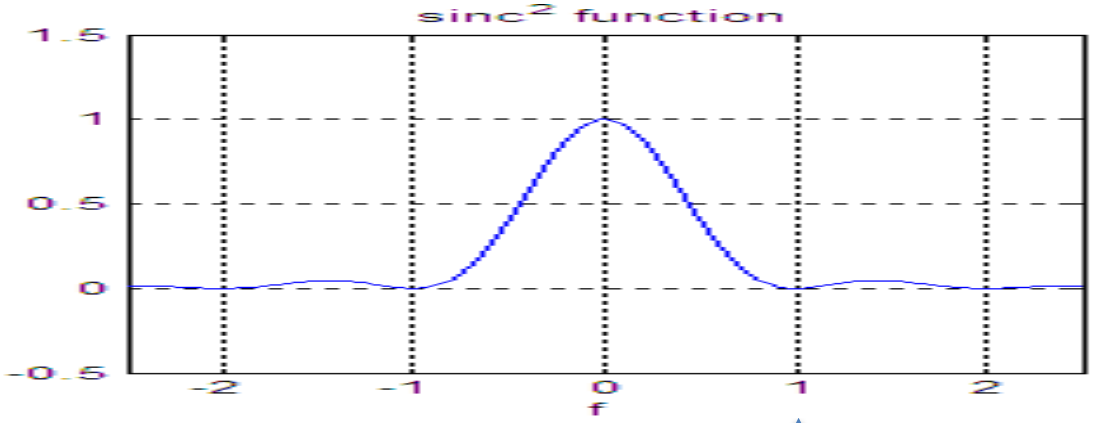
1 Hz

Another Common Example

What is Fourier of △(t)



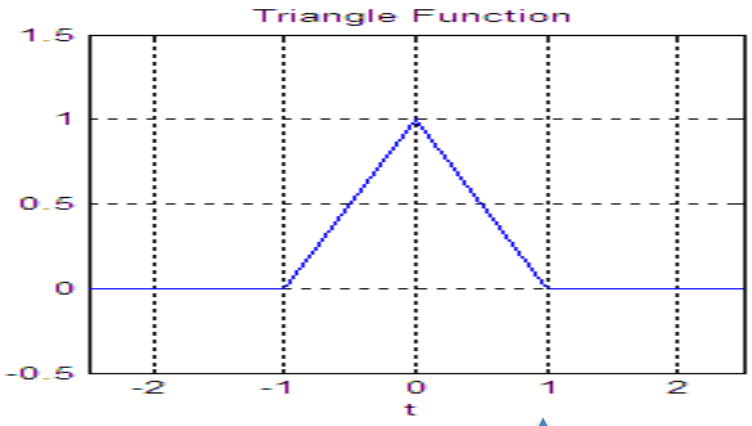
1 mS

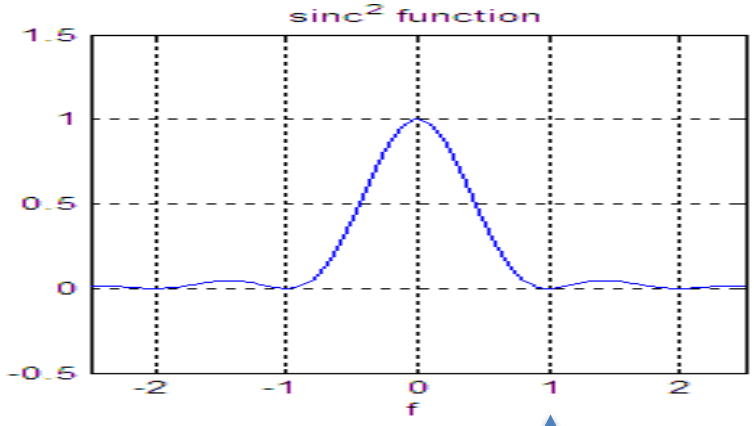


1000 Hz

Another Common Example

Fourier of $\triangle(t)$





1 mS

1000 Hz

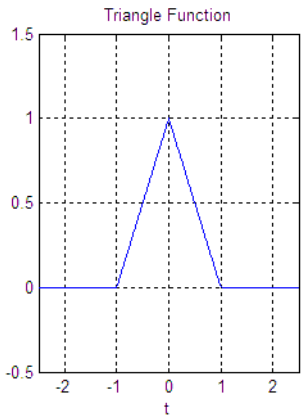
Scaling $f(ax)$ $\frac{1}{|a|}F\left(\frac{u}{a}\right)$

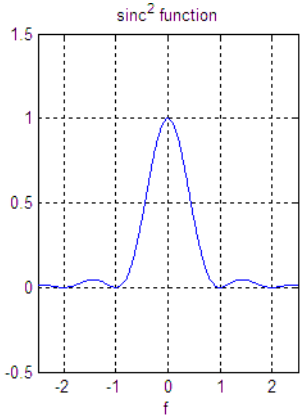
Property Examples

Time Shift

What is Fourier of △(t-a)

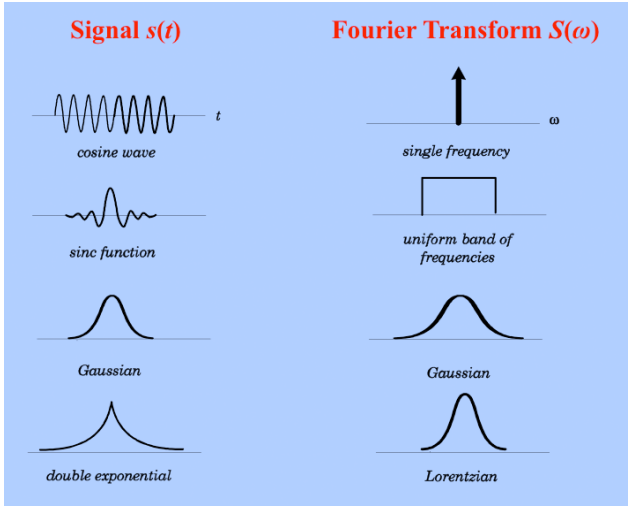
Shifting $f(x-x_0) \quad e^{-i2\pi ux_0}F(u)$





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	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{u}{a}\right)$
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	$\frac{d^n f(x)}{dx^n}$	$(i2\pi u)^n F(u)$

S. Narasimhan, Computer Vision

Common Closed Form Transforms and **More Properties**

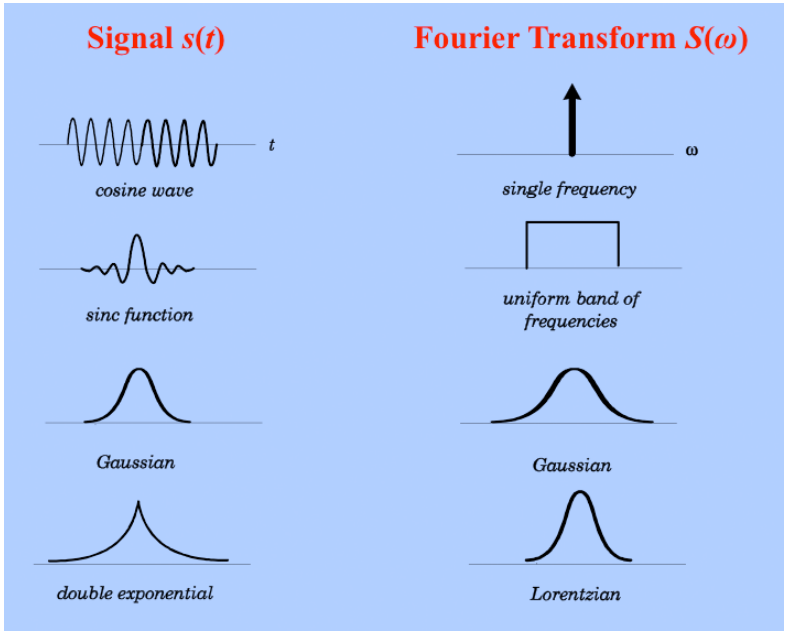


TABLE 4-1
Fourier Transform Theorems[a]

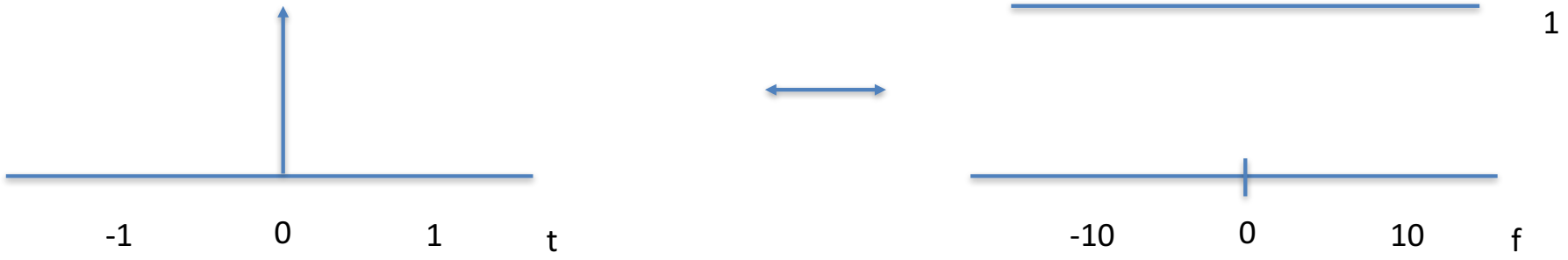
Name of Theorem		
1. Superposition (a_1 and a_2 arbitrary constants)	$a_1x_1(t) + a_2x_2(t)$	$a_1X_1(f) + a_2X_2(f)$
2. Time delay	$x(t - t_0)$	$X(f)e^{-j2\pi ft_0}$
3a. Scale change	$x(at)$	$\lvert a\rvert^{-1}X\left(\frac{f}{a}\right)$
b. Time reversal	$x(-t)$	$X(-f) = X*(f)$
4. Duality	$X(t)$	$x(-f)$
5a. Frequency translation	$x(t)e^{j\omega_0 t}$	$X(f - f_0)$
b. Modulation	$x(t)\cos\omega_0 t$	$\frac{1}{2}X(f - f_0) + \frac{1}{2}X(f + f_0)$
6. Differentiation	$\frac{d^n x(t)}{dt^n}$	$(j2\pi f)^n X(f)$
7. Integration	$\int_{-\infty}^{t} x(t')\,dt'$	$(j2\pi f)^{-1}X(f) + \frac{1}{2}X(0)\delta(f)$
8. Convolution	$\int_{-\infty}^{\infty} x_1(t - t')x_2(t')\,dt'$	$X_1(f)X_2(f)$
	$= \int_{-\infty}^{\infty} x_1(t')x_2(t - t')\,dt'$	
9. Multiplication	$x_1(t)x_2(t)$	$\int_{-\infty}^{\infty} X_1(f - f')X_2(f')\,df'$
		$= \int_{-\infty}^{\infty} X_1(f')X_2(f - f')\,df'$

[a] $\omega_0 = 2\pi f_0$; $x(t)$ is assumed to be real in 3b.

Example: impulse or “delta” function

- FT of delta function:

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

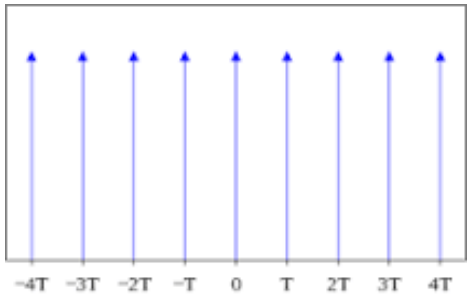


Like a very narrow pulse

Results in a ‘very’ broad spectrum sinc

But wait, data is discrete, but Fourier continuous

This is the comb function:



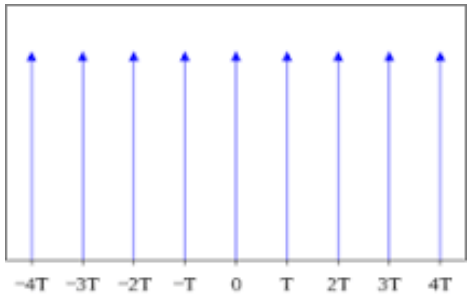
1 2. 3. 4. 5. 6…

What would this be?
III (t/T) =

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

But wait, data is discrete, but Fourier continuous

This is the comb function:



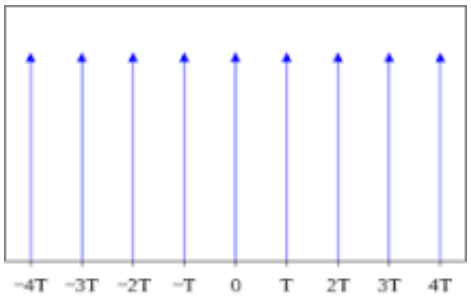
1 2. 3. 4. 5. 6…

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

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

But wait, that was discrete, and Fourier was continuous

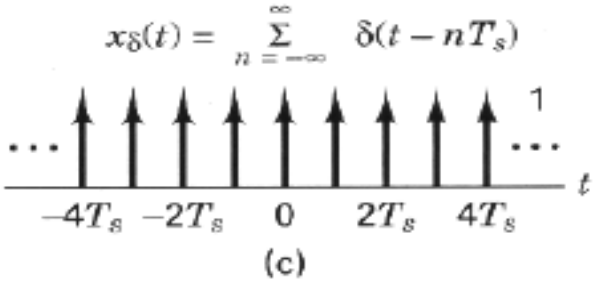
This is the comb function:



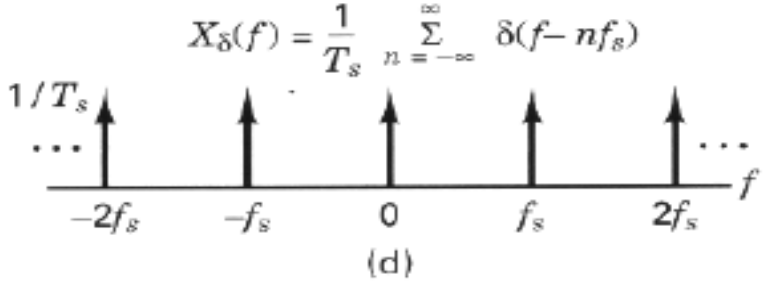
1 2. 3. 4. 5. 6…

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

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



(c)

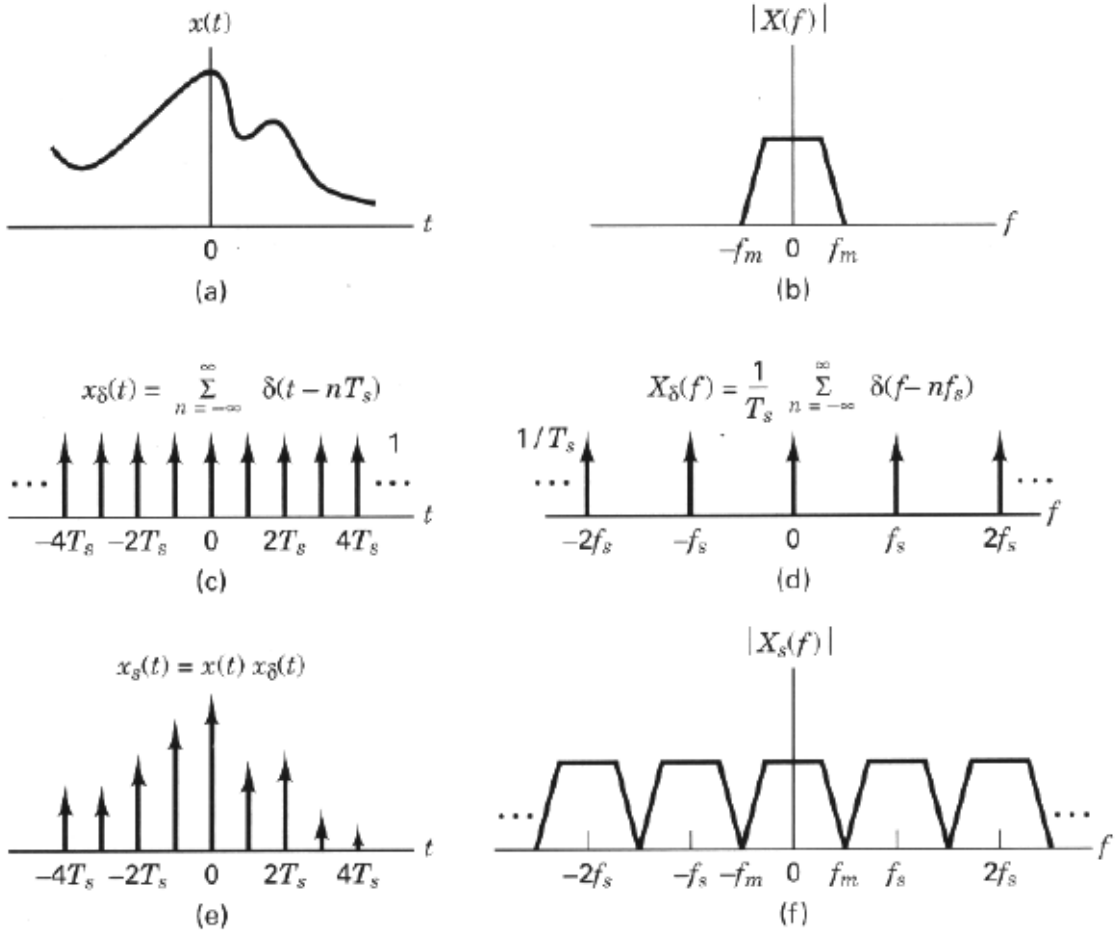


(d)

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 = 1/(T*N)



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

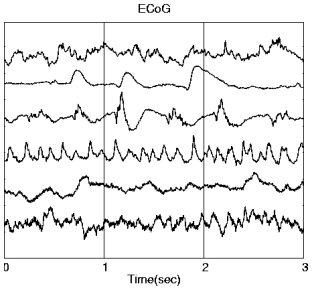
So that’s all great, but what about data?

```
>>> np.fft.fft(np.exp(2j * np.pi * np.arange(8) / 8))
array([ -3.44505240e-16 +1.14383329e-17j,
        8.00000000e+00 -5.71092652e-15j,
        2.33482938e-16 +1.22460635e-16j,
        1.64863782e-15 +1.77635684e-15j,
        9.95839695e-17 +2.33482938e-16j,
        0.00000000e+00 +1.66837030e-15j,
        1.14383329e-17 +1.22460635e-16j,
        -1.64863782e-15 +1.77635684e-15j])
```

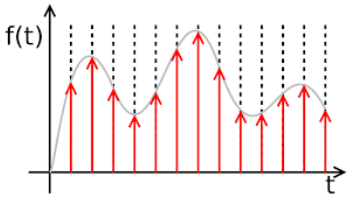
So that's all great, but what about data?

With data, we need to know 2 things:

1. What is the list of numbers (data) that was taken from the reading? Call is x[n]
2. What was the sampling frequency (fs) or sampling period (Ts) from the original source?



Continuous signals
x(t)



Sampled signals (data)
x(nTs)

Rec	Observed
1	60.323
2	61.122
3	60.171
4	61.187
5	63.221
6	63.639
7	64.989
8	63.761
9	66.019
10	67.857
11	68.169
12	66.513
13	68.655
14	69.564
15	69.331
16	70.551

Discrete data
x_n = x1, x2, x3, …

(might lose time reference)

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

Working with Numpy FFT Results: Scaling and Folding

Imports

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

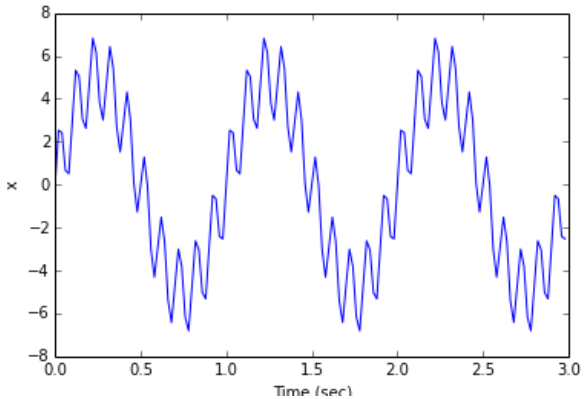
Create a Test Signal

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

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

Compute the FFT

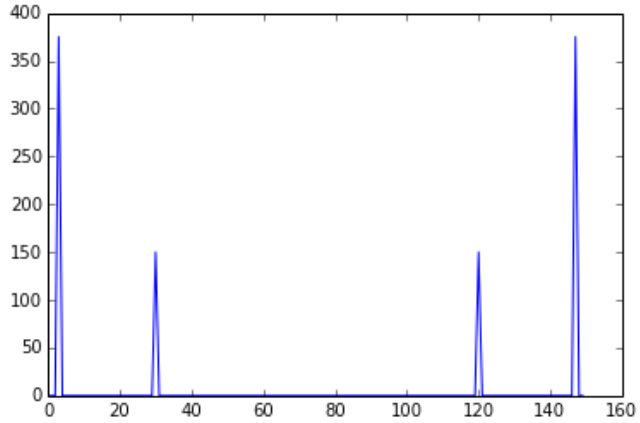
The FFT and a matching vector of frequencies

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

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

```
Out[5]: [<matplotlib.lines.Line2D at 0x70986f0>]
```



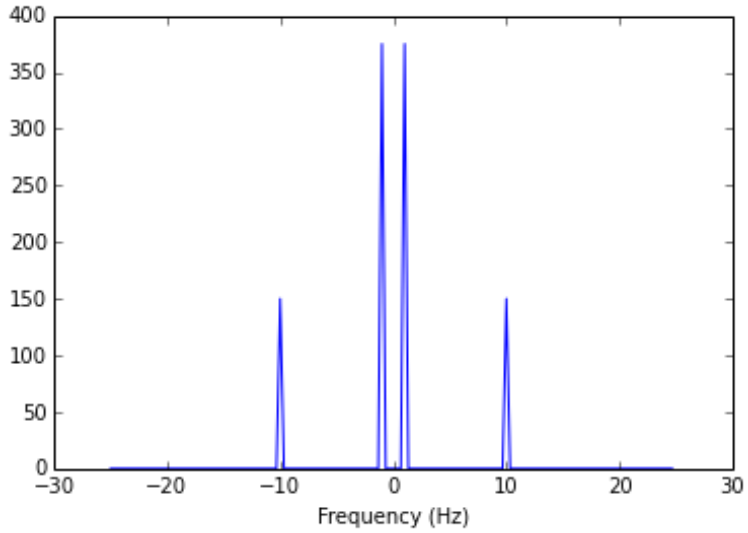
Swap Half Spaces

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

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

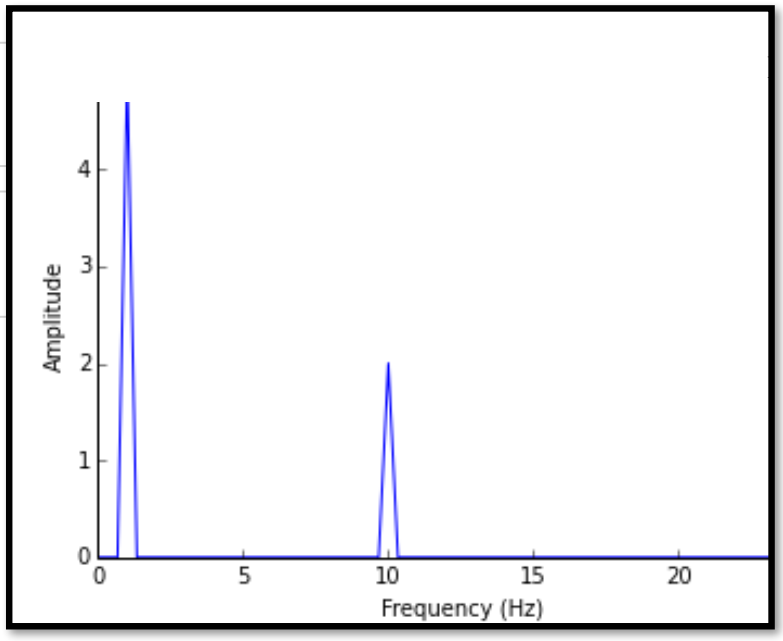
Fold Negative Frequencies and Scale

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 ampltude 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.

FFT Example in NumPy with complex input signal

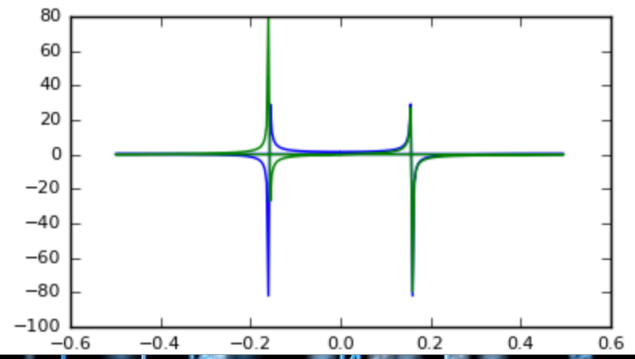
```
>>> np.fft.fft(np.exp(2j * np.pi * np.arange(8) / 8))

array([ -3.44505240e-16 +1.14383329e-17j,
        8.00000000e+00 -5.71092652e-15j,
        2.33482938e-16 +1.22460635e-16j,
        1.64863782e-15 +1.77635684e-15j,
        9.95839695e-17 +2.33482938e-16j,
        0.00000000e+00 +1.66837030e-15j,
        1.14383329e-17 +1.22460635e-16j,
        -1.64863782e-15 +1.77635684e-15j])
```

```
>>> np.fft.fft(np.exp(2j * np.pi * np.arange(8) / 8))
array([ -3.44505240e-16 +1.14383329e-17j,
         8.00000000e+00 -5.71092652e-15j,
         2.33482938e-16 +1.22460635e-16j,
         1.64863782e-15 +1.77635684e-15j,
         9.95839695e-17 +2.33482938e-16j,
         0.00000000e+00 +1.66837030e-15j,
         1.14383329e-17 +1.22460635e-16j,
         -1.64863782e-15 +1.77635684e-15j])
```

In this example, real input has an FFT which is Hermitian, i.e., symmetric in the real part and anti-symmetric in the imaginary part, as described in the numpy.fft documentation:

```
>>> import matplotlib.pyplot as plt
>>> t = np.arange(256)
>>> sp = np.fft.fft(np.sin(t))
>>> freq = np.fft.fftfreq(t.shape[-1])
>>> plt.plot(freq, sp.real, freq, sp.imag)
[<matplotlib.lines.Line2D object at 0x...>, <matplotlib.lines.Line2D object at 0
x...>]
>>> plt.show()
```

(Source code, png, pdf)



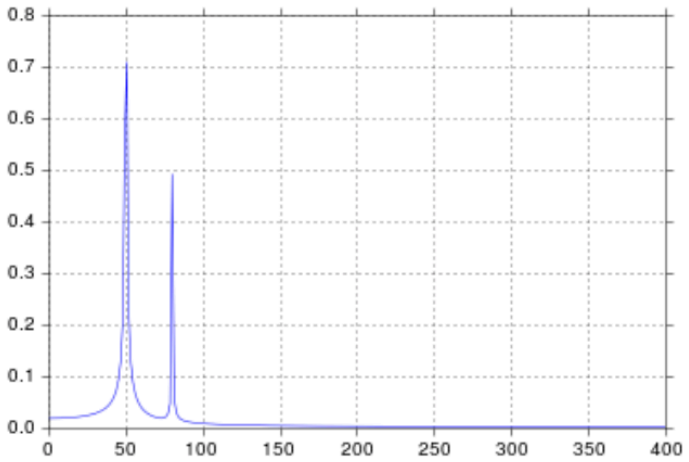
With code and data, we don’t have to worry about closed form solutions

```
>>> from scipy.fftpack import fft, ifft
>>> x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
>>> y = fft(x)
>>> y
array([ 4.50000000+0.j        ,  2.08155948-1.65109876j,
       -1.83155948+1.60822041j, -1.83155948-1.60822041j,
        2.08155948+1.65109876j])
>>> yinv = ifft(y)
>>> yinv
array([ 1.0+0.j,  2.0+0.j,  1.0+0.j, -1.0+0.j,  1.5+0.j])
```

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()
```

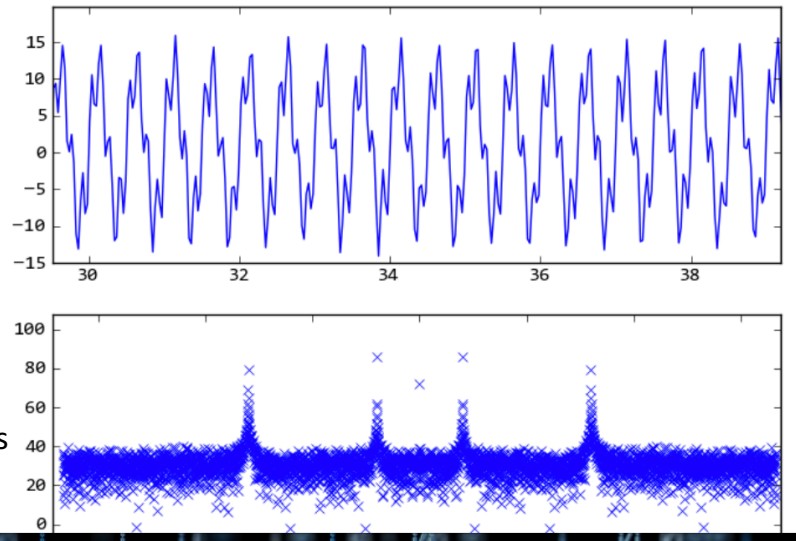
I get what I believe to be very reasonable output.



FFT Tool Example with SciPy

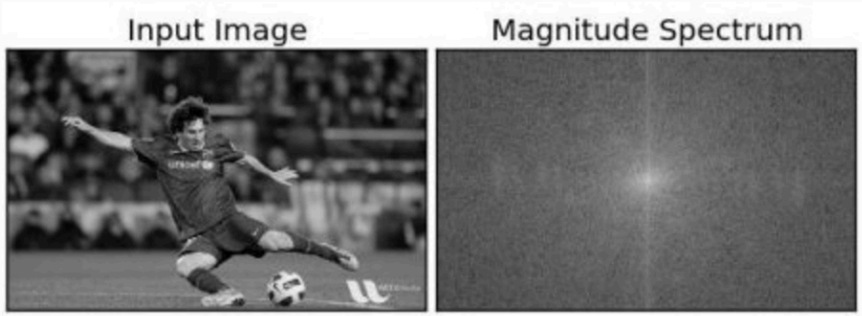
```
import scipy
import scipy.fftpack
import pylab
from scipy import pi
t = scipy.linspace(0,120,4000)
acc = lambda t: 10*scipy.sin(2*pi*2.0*t) + 5*scipy.sin(2*pi*8.0*t) + 2*scipy.random.

signal = acc(t)

FFT = abs(scipy.fft(signal))
freqs = scipy.fftpack.fftfreq(signal.size, t[1]-t[0])

pylab.subplot(211)
pylab.plot(t, signal)
pylab.subplot(212)
pylab.plot(freqs,20*scipy.log10(FFT),'x')
pylab.show()
```

from the graph you can see there are two peak at 2Hz and 8Hz.



See stack overflow:
http://stackoverflow.com/questions/9456037/scipy-numpy-fft-frequency-analysis

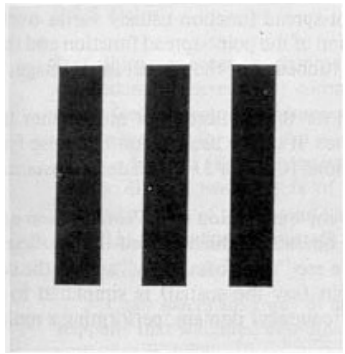
As we have seen before, getting image data and 2-d FFTs are possible with NumPy and Open CV



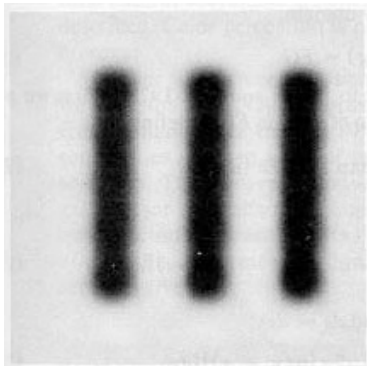

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

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)



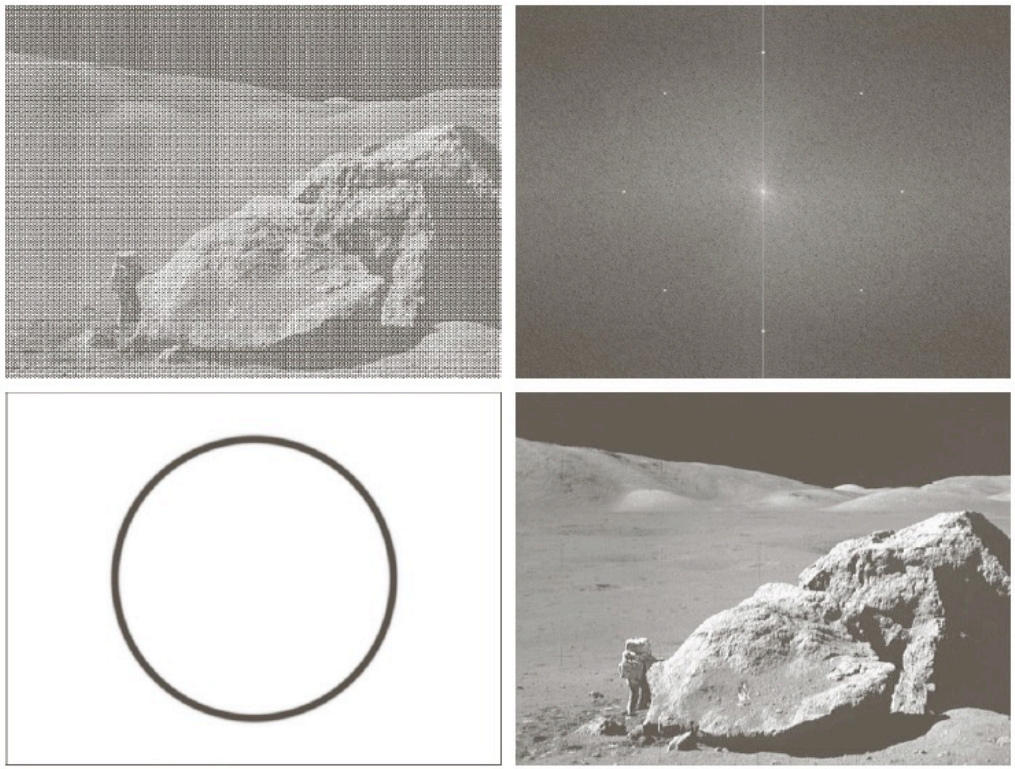
Original Image



Low-passed

Naveen Sihag

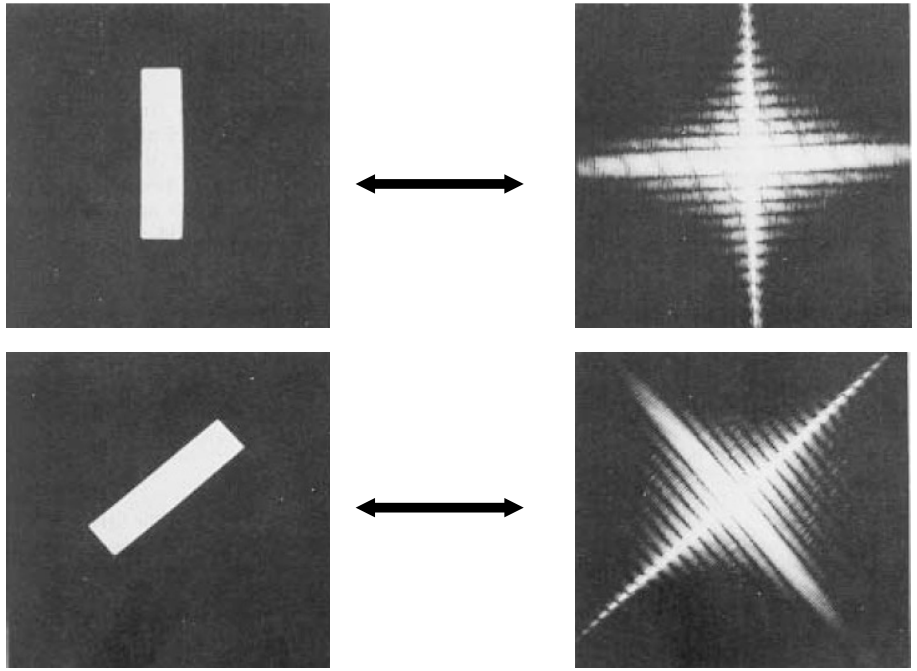
Example of noise reduction using FT



Naveen Sihag

DFT Properties: (5) Rotation

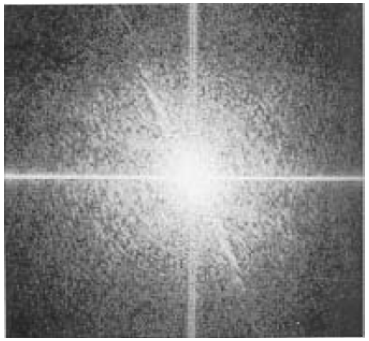
- Rotating f(x,y) by θ rotates F(u,v) by θ



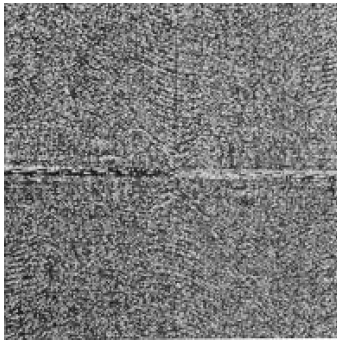
Naveen Sihag

Magnitude and Phase of DFT

- What is more important?



magnitude

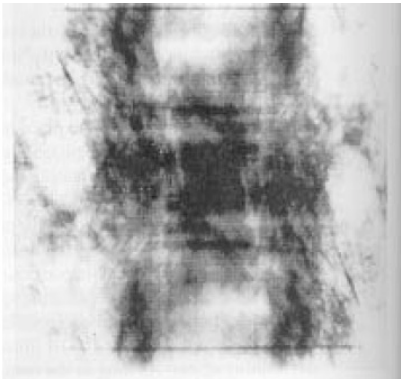


phase

- **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!)

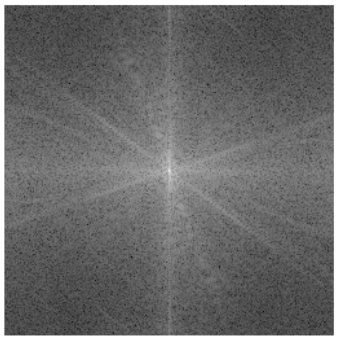
Reconstructed image using
phase only
(i.e., phase determines which components are present!)

Naveen Sihag

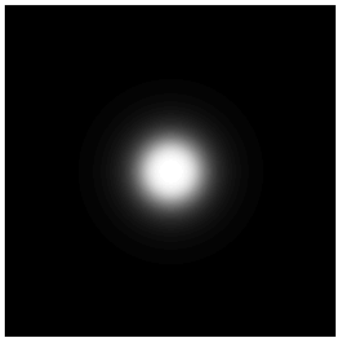
Low-pass Filtering

Original image

FFT of original image



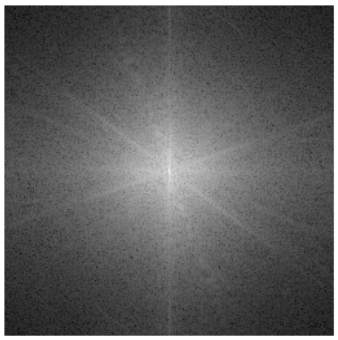
Low-pass filter



Let the low frequencies pass and eliminating the high frequencies.

Low-pass image

FFT of low-pass image

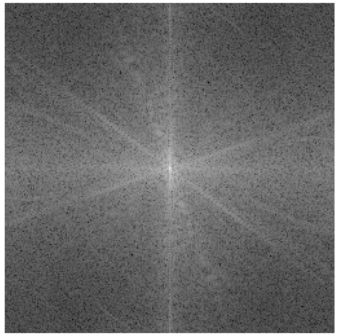


Generates image with overall shading, but not much detail

High-pass Filtering

Original image

FFT of original image

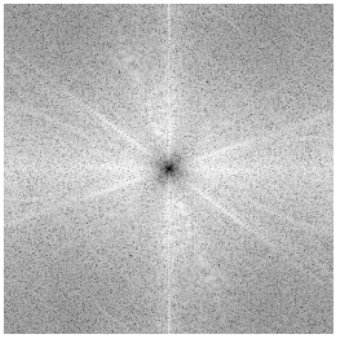


High–pass filter

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

End of Section