Tutorial 5 - Discrete Fourier Transform (DFT)

DFT is the process of decomposition if a signal into the frequencies that make it up
• Aperiodic-Continuous → Fourier Transform
• Periodic-Continuous → Fourier Series
• Aperiodic-Discrete → Discrete Time Fourier Transform
• Periodic-Discrete → Discrete Fourier Transform ←
Problem
In the real world, we only deal with finite signals, how do we apply DFT?

Solution
We define the signal as looping over itself
Euler's formula

\[ e^{i\theta} = \cos \theta + i \sin \theta \]

\[ \cos \omega = \frac{e^{i\omega} + e^{-i\omega}}{2} \quad \sin \omega = \frac{e^{i\omega} - e^{-i\omega}}{2i} \]
Given

\[ f(\theta) = e^{\alpha i \theta} = \cos \theta + i \sin \theta \]

In [3]: interact(plot1,
   a=widgets.IntSlider(min=1,max=10,step=1,value=1, layout=Layout(width= '100%')))

Out[3]: <function __main__.plot1(a)>
On the left graph, the time domain is along the $Z$ axis, the real and imaginary components are on $X$ and $Y$ respectively.
Decomposition

Seperating a signal into seperate components.

Synthesizes

Retrieving the original signal from the components.
DFT

Given a discrete signal defined as \([x_0, x_1, \ldots x_{N-1}]\) The DFT of the signal will be \([X_0, X_1, \ldots X_{n-1}]\) where

\[
X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn} \quad \leftrightarrow \quad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{\frac{i2\pi kn}{N}}
\]

Decomposition (Forward transform)

Synthesizes (Inverse transform)
In Numpy

A fast Fourier transform (FFT) is an algorithm that computes DFT $O(n \log n)$ for signal with length that is a power of 2.

In numpy, it works with arrays of different lengths, but is less efficient.

```python
from numpy import fft
fft.fft  #Decomposition
fft.ifft  #Synthesizes
```
In [7]: plot1()
In [8]:

def nth_component(fourier, n):
    fcpy = np.zeros_like(fourier)
    fcpy[n] = fourier[n]
    inv = fft.ifft(fcpy)
    return (np.real(inv), np.imag(inv))
```python
from itertools import chain

components = [nth_component(fourier, n) for n in range(len(signal))]

plot_components(x, components)
```
**Partial sum**

In [10]:
```python
def sum_x(x, components, n):
    re_sum = reduce(add, [re for re, _ in components[:n]])
    im_sum = reduce(add, [im for _, im in components[:n]])
    fig = plt.figure(figsize=(12, 6))
    plt.stem(x, re_sum)
    plt.ylim(-11, 11)
```

In [11]:
```python
components = components.copy()
def draw(n):
    return sum_x(np.arange(0, 16), components, n)
```
In [12]:
    interact(draw,
              n=widgets.IntSlider(min=1, max=16, step=1, value=1, layout=Layout(width='100%')))

Out[12]: <function __main__.draw(n)>
Even and odd signals

- Even signals \((f(x) = f(-x))\) will be symmetric around \(\frac{N}{2}\)
- Odd signals \((f(x) = f(-x))\) will be asymmetric around \(\frac{N}{2}\)
What will $\cos(x)$ look like?

In [14]: cos_plot()
What will $\sin(x)$ look like?

In [16]: `sin_plot()`
DFT in 2D

What does a frequency in 2D mean? We need to address a frequency and a direction. Generalization of 1D Fourier transform.

An element at \((u, v)\) represents a basis function at a corresponding direction.

Convolution to DFT relation

Given signals \(f, h\) and the matching Fourier transforms \(F, H\), then...
1. \( f \star h \leftrightarrow F \cdot H \)
2. A DFT of a gaussian is a gaussian
3. \( \{1 + 2\} \Rightarrow \) Convolve with a gaussian is the same as multiplying with a gaussian in the frequency domain
In [18]:
img = cv2.imread('messi.jpg',0)
img_fft = fft.fft2(img)
fshift = fft.fftshift(img_fft)

show_img_fft(img, fshift, fft.ifft2(fft.ifftshift(fshift)))
Low pass filter

Removing high frequencies from the image.

Trivial to do using fft
In [19]:

```python
img = cv2.imread('messi.jpg', 0)
img_fft = fft.fft2(img)
fshift = fft.fftshift(img_fft)  # makes frequencies be around center of image
```
In [20]:
rows, cols = img.shape
crow, ccol = rows//2, cols//2

mask_size = 60
kernel = zeros_like(img)
kernel[crow - mask_size:crow + mask_size, ccol - mask_size: ccol + mask_size] = 1
fshift *= kernel
In [21]: show_img_fft(img, fshift, fft.ifft2(fft.ifftshift(fshift)), kernel)
Only a fraction of the data is used, even though most of the detail remain.

A similar technique is used in image compression. (e.g. jpeg)
Noise reduction

Remainder - Gaussian:

$$f(x) = ae^{-\frac{x^2}{2c}}$$

**Important Gaussian quality** - A fourier transform of a gaussian is a gaussian!

We will perform a low pass filter using a gaussian instead of a box
In [24]:

```python
img = cv2.imread('messi.jpg', 0)
img_gaussian = img + np.random.normal(size=img.shape) * 10
img_fft = fft.fft2(img_gaussian)
fshift = fft.fftshift(img_fft) * G

rows, cols = img.shape
crow, ccol = rows//2, cols//2

mask_size = 120
```
In [25]:
```
show_img_fft(img_gaussian, fshift, fft.ifft2(fft.ifftshift(fshift)), G)
```
High pass filter
```python
In [26]:
def plot_high():
    img = cv2.imread('coins.jpg',0)
    img_fft = fft.fft2(img)
    fshift = fft.fftshift(img_fft)

    rows, cols = img.shape
    crow, ccol = rows//2, cols//2

    mask_size = 150

    kernel = np.ones_like(img)
    kernel[mask_size:-mask_size, mask_size:-mask_size] = 0

    fshift *= kernel

    show_img_fft(img, fshift, fft.ifft2(fft.ifftshift(fshift)), kernel)
```
In [27]: plot_high()