MAS212 Scientific Computing and Simulation

#10: The Discrete Fourier Transform

Key resources:

- Lec 10: `http://sam-dolan.staff.shef.ac.uk/mas212/docs/l10.pdf`
- Data sets : `http://sam-dolan.staff.shef.ac.uk/mas212/data/`.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline   # include plots in ipython notebook
```

Theory: The Fourier coefficients $\tilde{X}_k$ of a data set $x_j = [x_0, x_1, \ldots, x_{n-1}]$ are found by applying a Discrete Fourier Transform (DFT),

$$\tilde{X}_k = \sum_{j=0}^{n-1} x_j \exp(-i\, 2\pi jk/n), \tag{1}$$

where $k \in \mathbb{Z}$ and $\tilde{X}_{k \pm n} = \tilde{X}_k$. The data set may be reconstructed from $\tilde{X}_k$ by applying the inverse DFT:

$$x_j = \sum_{k=0}^{n-1} \tilde{X}_k \exp(+i\, 2\pi jk/n). \tag{2}$$

---

1. Finding the needle in the haystack. A noisy data set contains a hidden periodic signal!

(a) Download the first data set (`dft1.txt`), open it with `np.loadtxt()` and plot:

```
ts, xs = np.loadtxt('dft1.txt')
plt.plot(ts, xs, '-')
```

(b) Now take the DFT of the data, using functions in the `numpy.fft` module.

```
X_tilde = np.fft.fft(xs)
```

(c) To seek the signal, plot $\tilde{X}_k$ as a function of angular frequency ω_k, where $\omega_k = k\Delta\omega$ and $\Delta\omega = 2\pi/(n\Delta t)$. At what frequencies do 'spikes' appear?

```
n = len(xs)         # Number of data points
dt = ts[1]-ts[0]    # Delta t, the time interval
dw = 2*np.pi/dt  # Delta omega, the frequency interval
ws = np.fft.fftfreq(n, d=1/dw)  #  Get the frequency values, for x-axis.
plt.xlim(0, 20)
# Plot the square magnitude of Fourier coefficients
plt.plot(ws, X_tilde.real**2 + X_tilde.imag**2, '-')
```

(d) Add axis labels and a title to your plot.

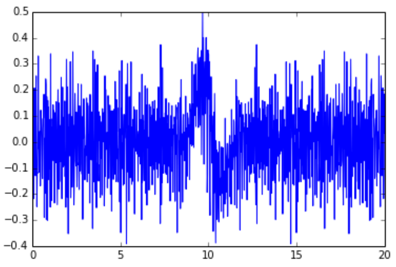
---

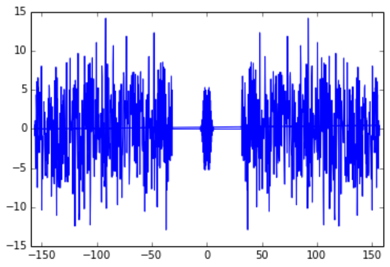
2. Filtering. Another signal is hidden in the second data set (`dft2.txt`). This data contains a low-frequency signal and high-frequency noise. Fortunately, we can separate the two by applying a frequency filter to $\tilde{X}_k$.

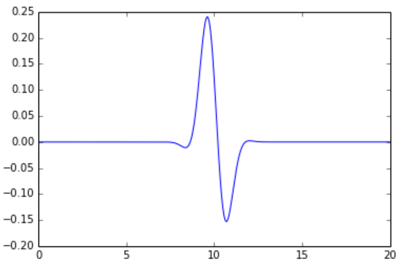
(a) Load the data set `dft2.txt` and plot.

(b) Take the DFT of the data, and plot this. What do you notice?

(c) Eliminate the high frequency part of $\tilde{X}_k$. Now take the **inverse** DFT to reconstruct the signal, using `np.fft.ifft`. Your signal should look something like the right-hand plot below.







---

3. Parseval's theorem.

Parseval's theorem states that, for any data set,

$$\sum_{j=0}^{n-1} |x_j|^2 = \frac{1}{n}\sum_{k=0}^{n-1}\left|\tilde{X}_k\right|^2$$

(a) By direct computation, demonstrate that the two data sets (`dft1.txt` and `dft2.txt`) satisfy Parseval's theorem to high numerical accuracy.

(b) Show that Parseval's theorem holds for a *complex* data set x_j of your own creation.

---

4. Fast Fourier Transforms.

Using the definition (1), write your own function to compute the DFT of a 1D numpy array of data. Now, using `%timeit` and a randomly-generated data set with $n = 10^4$ data points, compare the speed of your function and the numpy function `np.fft.fft()`. How many times faster is the numpy function? Try this again for $n = 10^5$ and $n = 10^6$.

Read the Wikipedia article on the Cooley-Tukey Fast Fourier Transform (FFT) algorithm to find out how this efficiency is achieved : `http://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm`. Can you implement a Fast Fourier Transform algorithm in Python? How does its efficiency compare with `np.fft.fft()`?