Contents

1 Plotting the DFT

1.1 With Python

1.1.1 Plotting the DFT

In this notebook we will look at the practical issues associated to plotting the DFT and in particular the DFT of real-world signals. We will examine how to map the DFT coefficients to real-world frequencies and we will investigate the frequency resolution of the DFT and the effects of zero padding.

As a quick reminder, the definition of the DFT for a length-N signal is:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi}{N}nk},\ k = 0,1,..N-1$$

> **FFT Module**
>
> ⌇ In Python, we will use the fft module in Numpy to compute the DFT

```python
# First the usual bookkeeping
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import IPython
from scipy.io.wavfile import write
```

```python
# Sets the size of the output plots
plt.rcParams["figure.figsize"] = (14,4)
```

Typically, we will take a vector of data points, compute the DFT and plot the magnitude of the result. For instance, consider the DFT of a linear ramp:

```python
x = np.arange(0, 10.2, 0.2) - 5
#draw figure
plt.stem(x);
```

```python
X = np.fft.fft(x);
plt.stem(abs(X));
```

1.1.2 Positive and negative frequencies

The coefficient number k indicates the contribution (in amplitude and phase) of a sinusoidal component of frequency

$$\omega_k = \frac{2\pi}{N}k$$

Because of the rotational symmetry of complex exponentials, a positive frequency ω between π and 2π is equivalent to a negative frequency of $\omega - 2\pi$; this means that half of the DFT coefficients correspond to negative frequencies and when we concentrate on the physical properties of the DFT it would probably make more sense to plot the coefficients centered around zero with positive frequencies on the right and negative frequencies on the left.

The reason why this is not usuall done are many, including

- convenience

- since we are manipulating finite-length signals, the convention dictates that we start at index zero

- when dealing with real-valued data, the DFT is symmetric in magnitude, so the first half of the coefficients is enough

- if we're looking for maxima in the magnitude, it's just easier to start at zero.

There is also another subtle point that we must take into account when shifting a DFT vector: **we need to differentiate between odd and even length signals**. With $k = 0$ as the center point, odd-length vectors will produce symmetric data sets with $(N1)/2$ points left and right of the oring, whereas even-length vectors will be asymmetric, with one more point on the positive axis; indeed, the highest positive frequency for even-length signals will be equal to $\omega_{N/2} = \pi$. Since the frequencies of π and $-\pi$ are identical, we can copy the top frequency data point to the negative axis and obtain a symmetric vector also for even-length signals.

Here is a function that does that:

```python
def dft_shift(X):
    N = len(X)
    if (N % 2 == 0):
        # even-length: return N+1 values
        return np.concatenate((X[(N/2):], X[:(N/2)+1]))
    else:
        # odd-length: return N values
        return np.concatenate((X[int((N+1)/2):], X[:int((N-1)/2)]))
```

```python
plt.stem(abs(dft_shift(X)));
```

While the function does shift the vector, the indices are still from zero to N−1N−1. Let's modify it so that we return also the proper values for the indices:

```python
def dft_shift(X):
    N = len(X)
    if (N % 2 == 0):
        # even-length: return N+1 values
        return (np.arange(-int(N/2), int(N/2) + 1),
                np.concatenate((X[int(N/2):], X[:int(N/2)+1])))
    else:
        # odd-length: return N values
        return (np.arange(-int((N-1)/2), int((N-1)/2) + 1),
                np.concatenate((X[int((N+1)/2):], X[:int((N+1)/2)])))
```

```python
n, y = dft_shift(X)
plt.stem(n, abs(y));
```

1.1.3 Mapping the DFT index to real-world frequencies

The next step is to use the DFT to analyze real-world signals. As we have seen in previous examples, what we need to do is set the time interval between samples or, in other words, set the "clock" of the system. For audio, this is equivalent to the sampling rate of the file.

```python
# import IPython   not used in orgmode
from scipy.io import wavfile
Fs, x = wavfile.read("./sound/piano.wav")
# IPython.display.Audio(x, rate=Fs) not used in orgmode
```

Play piano sound

In order to look at the spectrum of the sound file with a DFT we need to map the digital frequency "bins" of the DFT to real-world frequencies.

The $k-th$ basis function over $\mathbb{C}^N$ completes $k-periods$ over N samples . If the time between samples is $1/Fs$, then the real-world frequency of the $k-th$ basis function is periods over time, namely $k(F_s/N)$.

Let's remap the DFT coefficients using the sampling rate:

```python
def dft_map(X, Fs, shift=True):
    resolution = float(Fs) / len(X)
    if shift:
        n, Y = dft_shift(X)
    else:
        Y = X
        n = np.arange(0, len(Y))
    f = n * resolution
    return f, Y
```

```python
# let's cut the signal otherwise it's too big
x = x[:32768]
X = np.fft.fft(x);
f, y = dft_map(X, Fs)
plt.plot(f, abs(y));
```

The plot shows what a spectrum analyzer would display. We can see the periodic pattern in the sound, like for all musical tones. If we want to find out the original pitch we need to zoom in in the plot and find the first peak. This is one of the instances in which shifting the DFT does not help, since

4

we'll be looking in the low-frequency range. So let's re-plot withouth the shift, but still mapping the frequencies:

```python
X = np.fft.fft(x);
f, y = dft_map(X, Fs, shift=False)
plt.plot(f[:2000], abs(y[:2000]));
```

We can see that the first peak is in the vicinity of 200Hz; to find the exact frequency (to within the resolution afforded by this DFT) let's find the location

```python
dft_resolution = float(Fs)/ len(x)
print("DFT resolution is", dft_resolution, "Hz")

# let's search up to 300Hz
max_range = int(300 / dft_resolution)
ix = np.argmax(abs(y[:max_range]))
pitch = f[ix]
print("the note has a pitch of", pitch, "Hz")
```

```
('DFT resolution is', 0.48828125, 'Hz')
```

> **Concert Pitch**
> ↯ So the note is a A, half the frequency of concert pitch.

1.1.4 Zero-padding

Since the resolution of a DFT depends on the length of the data vector, one may erroneously assume that, by artificially extending a given data set, the resulting resolution would improve. Note that here we're not talking about collecting more data; rather, we have a data set and we append zeros (or any other constant value) to the end of it. This extension is called zero-padding.

The derivation of why zero-padding does not increase the resolution is detailed in the book. Here we will just present a simple example.

```python
N = 256
Delta = 2*np.pi / N
n = np.arange(0, N)

# main frequency (not a multiple of the fundamental freq for the space)
omega = 2*np.pi / 10

x = np.cos(omega * n) + np.cos((omega + 3*Delta) * n)
plt.plot(abs(np.fft.fft(x))[:100]);
```

we can tell the two frequencies apart and, if you zoom in on the plot, you will see that they are indeed three indices apart. Now let's build a signal with two frequencies that are less than $\Delta\Delta$ apart:

```python
x = np.cos(omega * n) + np.cos((omega + 0.5*Delta) * n)
plt.plot(abs(np.fft.fft(x))[:100]);
```

The two frequencies cannot be resolved by the DFT. If you try to increase the data vector by zero padding, the plot will still display just one peak:

```python
xzp = np.concatenate((x, np.zeros(2000)))
plt.plot(abs(np.fft.fft(xzp))[:500]);
```

1.2 With Matlab/Octave

```matlab
N=128;
fo1=21/128;
fo2=21/127;
n=0:N-1;
x1=cos(2*pi*fo1*n);
x2=cos(2*pi*fo2*n);
X1=fft(x1);                               # Compute the dft of X1 using FFT algorit
X2=fft(x2);                               # Compute the dft of X1 using FFT algorit

# Graphik
figure( 1, "visible", "off" )            # Do not open the graphic window in org
subplot(1,2,1),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid on;
xlabel("Sample");
ylabel("X_1[k]");
```
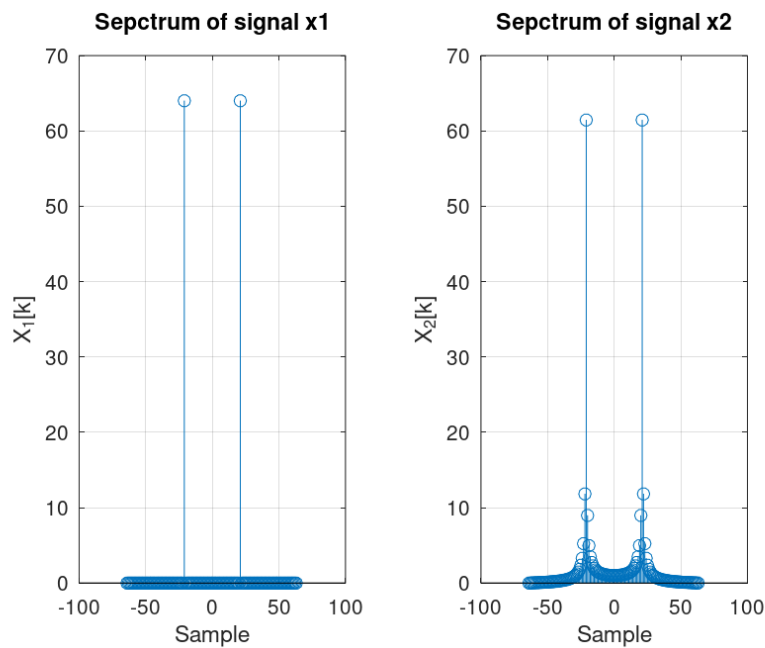
```
title("Sepctrum of signal x1");
subplot(1,2,2), stem(n-N/2,fftshift(abs(X2)))
grid on;
xlabel("Sample");
ylabel("X_2[k]");
title("Sepctrum of signal x2");

# Org-Mode specific setting
print -dpng ./image/oct_fft.png;
ans = "./image/oct_fft.png";
```


1.2.1 Homework 4

```
N=64;
n=0:N-1;

x1=ones(N,1);
X1=fft(x1);                                    # Compute the dft of X1 using FFT algorit
```

```
# Graphik
figure( 1, "visible", "off" )                      # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))   # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw4a_fft.png;
ans = "./image/hw4a_fft.png";
```
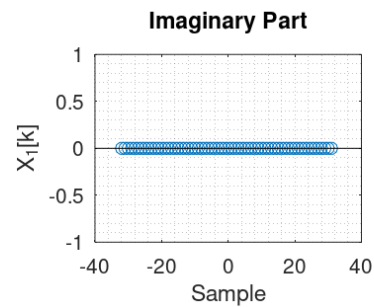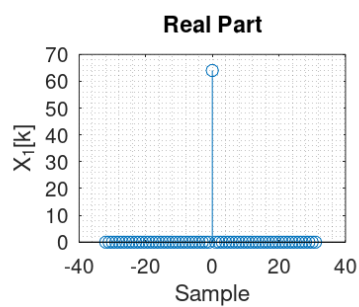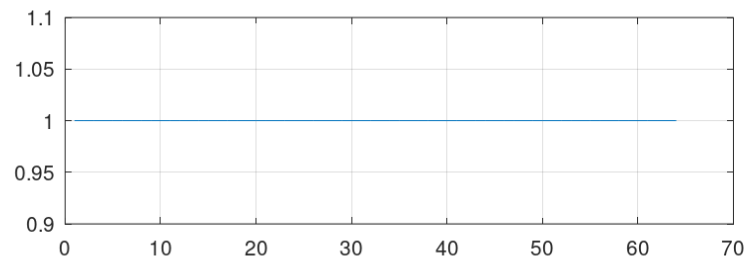
```
x1=[1 2 3 4 5]
X1=fft(x1);                                    # Compute the dft of X1 using FFT algorith
X2=fft(X1);


ans = X2/5;
```

$$1 \quad 5 \quad 4 \quad 3 \quad 2$$

1.2.2 Homework 6

```
N=64;
fo1=4*60/64;
n=0:N-1;

x1=(-1)
X1=fft(x1);                                    # Compute the dft of X1 using FFT algorith

# Graphik
```

```
figure( 1, "visible", "off" )                    # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))   # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6a_fft.png;
ans = "./image/hw6a_fft.png";
```

```
N=64;
fo1=8;
n=0:N-1;

x1=0.5*sin(2*pi/N*fo1*n);
X1=fft(x1);                                 # Compute the dft of X1 using FFT algorit

# Graphik
figure( 1, "visible", "off" )               # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1);
grid on;
#stem(n-N/2,fftshift(x1))
```
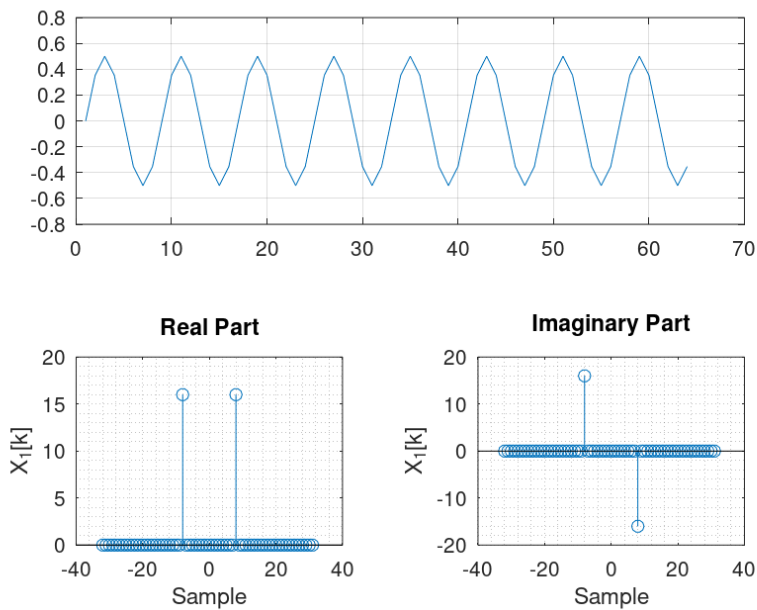
```
subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))   # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6b_fft.png;
ans = "./image/hw6b_fft.png";
```



```
N=64;
fo1=4;
```

```octave
n=0:N-1;

x1=2*cos(2*pi/N*fo1*n);
X1=fft(x1);                              # Compute the dft of X1 using FFT algorit

# Graphik
figure( 1, "visible", "off" )            # Do not open the graphic window in org

subplot(2,2,[1,2])
plot(x1)
grid on;
#stem(n-N/2,fftshift(x1))

subplot(2,2,3),stem(n-N/2,fftshift(abs(X1)))  # Move frequency 0 to the center
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Real Part");

subplot(2,2,4),stem(n-N/2,fftshift(imag(X1)))
grid minor;
xlabel("Sample");
ylabel("X_1[k]");
title("Imaginary Part");


# Org-Mode specific setting
print -dpng ./image/hw6d_fft.png;
ans = "./image/hw6d_fft.png";
```