W8 DFT (continued)

March 1, 2024

1 W8 4th Hour with TA (Opal)

How to get started with Python? A guide to installation and setting up a virtual enviroment: https://www.python.org/about/gettingstarted/. If you are struggling with downloading Python, I am happy to help you with the installation during TA office hours. An alternative approach is to use Jupyter Lab and avoid downloading Python.

[1]:
```
# import essential Python packages 'numpy' and 'librosa' and 'matplotlib'
import numpy as np
import librosa
import IPython
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
# import plotting package 'matplotlib' in Python
import matplotlib.pyplot as plt
import matplotlib

# set up preferred fontsize etc...
font = {'family' : 'serif',
        'size'   : 12}

matplotlib.rc('font', **font)
matplotlib.rc('xtick', labelsize=12)
matplotlib.rc('ytick', labelsize=12)
```

2 Discrete Fourier Transform

The discrete Fourier transform of a signal $x[n]$ with period N_0 is

$$X[k] = \frac{1}{N_0} \sum_{n=0}^{N_0-1} x[n] e^{-i2\pi \frac{k}{N_0} n}$$

where x_k are the discrete Fourier transform coefficients and the inverse Fourier transform is

$$x[n] = \sum_{k=0}^{N_0-1} X[k] e^{i2\pi \frac{k}{N_0} n}$$

2.1 Example 1

Consider the continuous signal $x(t) = \sin(2\pi t) + \cos(42\pi t) + \cos(100\pi t)$

a. Sample the signal at different at sampling rates $f_s = 11, 41, 101, 501$ samples/sec. Plot and save your results from $t \in [0, 1)$ sec.

[4]:
```
# example signal
def cont_signal(t):
    return np.sin(2*np.pi*t) + np.cos(42*np.pi*t) + np.cos(100*np.pi*t)
```

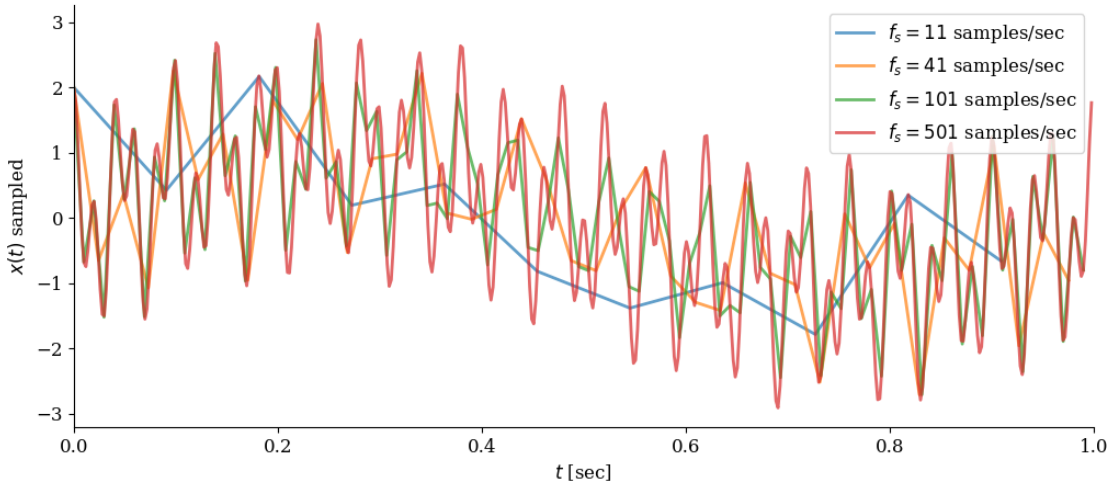
[5]:
```
T = 1 # final time in seconds
```

[6]:
```
# create discrete signals
sig_11 = cont_signal(np.linspace(0, T, int(T*11), endpoint=False))
sig_41 = cont_signal(np.linspace(0, T, int(T*41), endpoint=False))
sig_101 = cont_signal(np.linspace(0, T, int(T*101), endpoint=False))
sig_501 = cont_signal(np.linspace(0, T, int(T*501), endpoint=False))
```

[7]:
```
fig, ax = plt.subplots(figsize=(12, 5))
ax.plot(np.linspace(0, T, int(T*11), endpoint=False), sig_11, alpha=0.7,␣
 ↪linewidth=2, label="$f_{s} = 11$ samples/sec")
ax.plot(np.linspace(0, T, int(T*41), endpoint=False), sig_41, alpha=0.7,␣
 ↪linewidth=2, label="$f_{s} = 41$ samples/sec")
ax.plot(np.linspace(0, T, int(T*101), endpoint=False), sig_101, alpha=0.7,␣
 ↪linewidth=2, label="$f_{s} = 101$ samples/sec")
ax.plot(np.linspace(0, T, int(T*501), endpoint=False), sig_501, alpha=0.7,␣
 ↪linewidth=2, label="$f_{s} = 501$ samples/sec")

ax.set_xlabel("$t$ [sec]")
ax.set_ylabel("$x(t)$ sampled")
ax.legend()
ax.set_xlim(0, T)

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
```



b. Compute the discrete Fourier transform (DFT) of the discrete signals sampled at $f_s = 11, 41, 101, 501$ samples/sec. Plot the discrete Fourier coefficients. Explain the symmetry and magnitude difference. Are all the sampled signals able to recover the spectrum of the continuous signal?

[8]:
```
fig, ax = plt.subplots(nrows=2, figsize=(10, 7))
ax[0].scatter(np.arange(-int(T*11)//2 + 1, int(T*11)//2  + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_11)).real, alpha=0.9, label="$f_{s} = 11$ samples/
 ↪sec")
ax[0].scatter(np.arange(-int(T*41)//2 + 1, int(T*41)//2 + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_41)).real, alpha=0.9, label="$f_{s} = 41$ samples/
 ↪sec")
ax[0].scatter(np.arange(-int(T*101)//2 + 1, int(T*101)//2  +1), np.fft.
 ↪fftshift(np.fft.fft(sig_101)).real, alpha=0.9, label="$f_{s} = 101$ samples/
 ↪sec")
ax[0].scatter(np.arange(-int(T*501)//2 + 1, int(T*501)//2 + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_501)).real, alpha=0.9, label="$f_{s} = 501$ samples/
 ↪sec")

ax[1].scatter(np.arange(-int(T*11)//2 + 1, int(T*11)//2  + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_11)).imag, alpha=0.9, label="$f_{s} = 11$ samples/
 ↪sec")
ax[1].scatter(np.arange(-int(T*41)//2 + 1, int(T*41)//2 + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_41)).imag, alpha=0.9, label="$f_{s} = 41$ samples/
 ↪sec")
ax[1].scatter(np.arange(-int(T*101)//2 + 1, int(T*101)//2  +1), np.fft.
 ↪fftshift(np.fft.fft(sig_101)).imag, alpha=0.9, label="$f_{s} = 101$ samples/
 ↪sec")
```

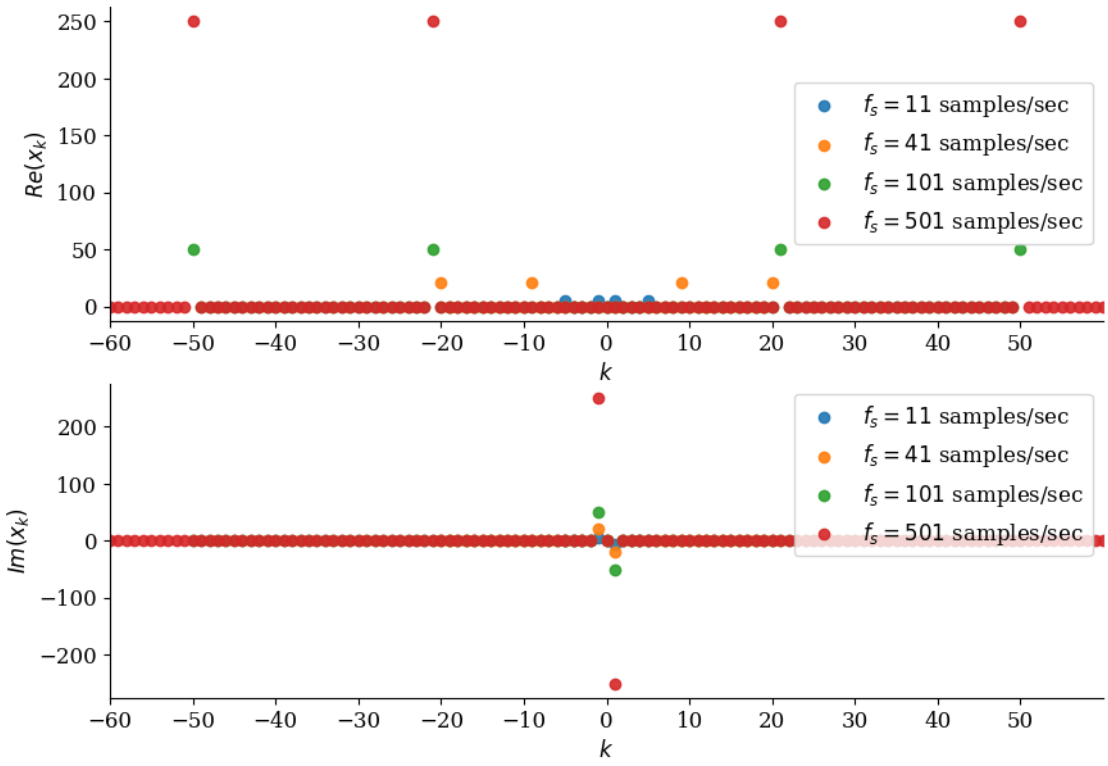
```
ax[1].scatter(np.arange(-int(T*501)//2 + 1, int(T*501)//2 + 1), np.fft.
 ↪fftshift(np.fft.fft(sig_501)).imag, alpha=0.9, label="$f_{s} = 501$ samples/
 ↪sec")

ax[0].set_xlabel("$k$")
ax[0].set_ylabel("$Re(x_{k})$")
ax[0].legend()
ax[0].set_xlim(-60, 60)
ax[0].set_xticks(np.arange(-60, 60, 10))

ax[1].set_xlabel("$k$")
ax[1].set_ylabel("$Im(x_{k})$")
ax[1].legend()
ax[1].set_xlim(-60, 60)
ax[1].set_xticks(np.arange(-60, 60, 10))

ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)

ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)
```



3 Example 2

Read into MATLAB/Python the file **song.mp3**. Tip: In Python, install the librosa package.

a. Play the song. Do you recognize it?

[32]:
```
y, sr = librosa.load('song_W8.mp3')
```

```
[src/libmpg123/parse.c:skip_junk():1276] error: Giving up searching valid MPEG
header after 65536 bytes of junk.
```

[33]:
```
print(r"sampling rate fs= " + str(sr) + " Hz (or samples per second)")
```

```
sampling rate fs= 22050 Hz (or samples per second)
```

[34]:
```
IPython.display.Audio(y, rate=sr)
```

[34]:
```
<IPython.lib.display.Audio object>
```

b. Plot the audio signal and its discrete Fourier transform magnitude.

[12]:
```
len(y) # this is a very large data set > 4 million samples
```

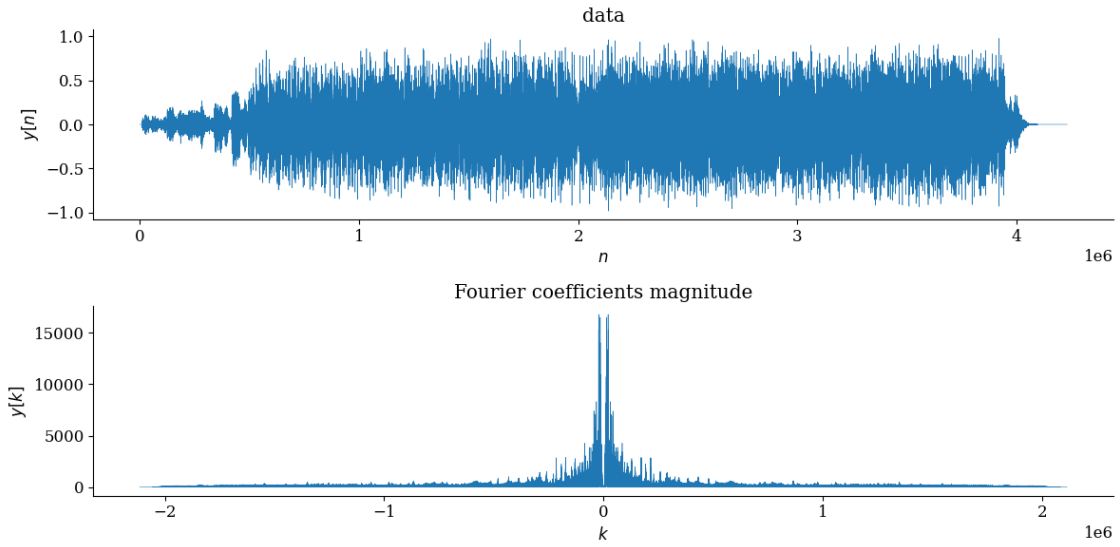
[12]:
```
4229632
```

[14]:
```
# compute the DFT
fft_y = np.fft.fft(y)
```

[15]:
```
fig, ax = plt.subplots(nrows=2, figsize=(12, 6))
ax[0].plot(y, linewidth=0.4)
ax[-1].plot(np.arange(-len(y)//2, len(y)//2), np.abs(np.fft.fftshift(fft_y)),␣
 ↪linewidth=0.4)
ax[0].set_xlabel("$n$")
ax[0].set_ylabel("$y[n]$")
ax[0].set_title("data")
ax[1].set_xlabel("$k$")
ax[1].set_ylabel("$y[k]$")
ax[1].set_title("Fourier coefficients magnitude")

ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)

ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)
plt.tight_layout()
plt.savefig("p4_b.png", dpi=600)
```



c. Apply a *ideal* low-pass filter on the audio data. Listen to the filtered song with $k_c = 10^4, 10^5, 10^6$.

[16]:
```
def low_pass_filter(y_fft, k_c):
    y_fft_filtered = np.zeros(len(y_fft), dtype="complex128")
    y_fft_filtered[len(y)//2 - k_c: len(y)//2 + k_c] = np.fft.
↪fftshift(y_fft)[len(y)//2 - k_c: len(y)//2 + k_c]
    return np.fft.ifft(np.fft.ifftshift(y_fft_filtered))
```

[17]:
```
y_filtered_low_4 = low_pass_filter(fft_y, k_c=int(1e4))
y_filtered_low_5 = low_pass_filter(fft_y, k_c=int(1e5))
y_filtered_low_6 = low_pass_filter(fft_y, k_c=int(1e6))
```

[18]:
```
IPython.display.Audio(y_filtered_low_4, rate=sr)
```

[18]:
```
<IPython.lib.display.Audio object>
```

[19]:
```
IPython.display.Audio(y_filtered_low_5, rate=sr)
```

[19]:
```
<IPython.lib.display.Audio object>
```

[20]:
```
IPython.display.Audio(y_filtered_low_6, rate=sr)
```

[20]:
```
<IPython.lib.display.Audio object>
```

d. Apply a high-pass-filter on the audio data. Listen to the filtered audio with $k_c = 10^4, 10^5, 10^6$.

[21]:
```
def high_pass_filter(y_fft, k_c):
    y_fft_filtered = np.fft.fftshift(y_fft)
    y_fft_filtered[len(y)//2 - k_c: len(y)//2 + k_c] = np.zeros(2*k_c)
```

```
    return np.fft.ifft(np.fft.ifftshift(y_fft_filtered))
```

[22]:
```
y_filtered_high_4 = high_pass_filter(fft_y, k_c=int(1e4))
y_filtered_high_5 = high_pass_filter(fft_y, k_c=int(1e5))
y_filtered_high_6 = high_pass_filter(fft_y, k_c=int(1e6))
```

[23]:
```
IPython.display.Audio(y_filtered_high_4, rate=sr)
```

[23]:
```
<IPython.lib.display.Audio object>
```

[24]:
```
IPython.display.Audio(y_filtered_high_5, rate=sr)
```

[24]:
```
<IPython.lib.display.Audio object>
```

[25]:
```
IPython.display.Audio(y_filtered_high_6, rate=sr)
```

[25]:
```
<IPython.lib.display.Audio object>
```

e. Change the sampling rate to twice and half the current rate of the song. Analyze/discuss your results.

[26]:
```
# twice the sampling rate
IPython.display.Audio(y, rate=sr*2)
```

[26]:
```
<IPython.lib.display.Audio object>
```

[27]:
```
# half the sampling rate
IPython.display.Audio(y, rate=sr//2)
```

[27]:
```
<IPython.lib.display.Audio object>
```

f. Subs-sample the low pass filtered audio with $k_c = 10^6$ by (1) saving every other data point and (2) saving every sixth datapoint. Analyze/discuss your results.

[30]:
```
# twice the sampling rate
IPython.display.Audio(y_filtered_low_6[::2], rate=sr//2)
```

[30]:
```
<IPython.lib.display.Audio object>
```

[31]:
```
# twice the sampling rate
IPython.display.Audio(y_filtered_low_6[::6], rate=sr//6)
```

[31]:
```
<IPython.lib.display.Audio object>
```

[ ]:
```

```