Review: Complex numbers

Motivation: functions like the FT (Fourier Transform) return complex numbers.

Most likely you first ran into imaginary numbers when finding roots or taking the square-root of a negative number:

\[ i = j = \sqrt{-1} \]

A complex number has a real part and an imaginary part:

\[ x = 3 + 4j \]

You can think of real numbers as just having zero imaginary parts!

\[ >> \text{real}(x) \]
\[ = 3 \]
\[ >> \text{imag}(x) \]
\[ = 4 \]

Complex numbers provide a way of holding two pieces of information. For example, consider converting number from Polar to Cartesian coordinates.

\[ x = r \cos(\theta) \]
\[ y = r \sin(\theta) \]

This information could be stored as: \( z = x + yi \)

In which case \( r \) and \( \theta \) are easily recovered:

\[ r = |x + yi| = \sqrt{x^2 + y^2} \]
\[ \theta = \arctan(y / x) \text{ \scriptsize \textit{In Matlab}} \]
\[ \text{angle}(x + yi) \text{ \scriptsize \textit{In Matlab}} \]
Amplitude: \( c_n = \sqrt{a_n^2 + b_n^2} \)
Phase: \( \Phi_n = \arctan\left(\frac{b_n}{a_n}\right) \)

Euler's Formula

\[
e^{-j\phi} = \cos(\phi) - j\sin(\phi)
\]
\[
e^{j\phi} = \cos(\phi) + j\sin(\phi)
\]

\[
\cos(\phi) = \frac{e^{j\phi} + e^{-j\phi}}{2}
\]
\[
\sin(\phi) = \frac{e^{j\phi} - e^{-j\phi}}{2j}
\]

Consider a discrete signal in the TIME DOMAIN

DFT = Discrete Fourier Transform
FFT = Fast Fourier Transform (efficient implementation of FFT)
iDFT or iFFT = Inverse Fourier Transform
DFT = Discrete Fourier transform
FFT = Fast Fourier Transform (efficient implementation of FFT)
iDFT or iFFT = Inverse Fourier transform

\[ x(t) = \sum_{n=0}^{\infty} [a_n \cos\left(\frac{2\pi n}{T} t\right) + b_n \sin\left(\frac{2\pi n}{T} t\right)] \]

Where \( T = \text{period} \)
\( t = \text{time} [0, \ldots, T] \)

\[ x(m) = \sum_{n=0}^{N/2} [a_n \cos\left(\frac{2\pi n}{N} m\right) + b_n \sin\left(\frac{2\pi n}{N} m\right)] \]

\( m = \text{the sample number} \)
\( m = 0, 1, \ldots, N-1 \)
Q: How do you find these magical coefficients?

A: By projection the time series on to each of the harmonic component signals

\[ a_n = \sum_{m=0}^{N-1} x(m) \cos(2\pi nm / N) \]

\[ b_n = \sum_{m=0}^{N-1} x(m) \sin(2\pi nm / N) \]

where \( m=[0\ldots N-1] \)

\( N \) is the total # of samples

Any band-limited signal can be represented as (decomposed into) a set of sines and cosines

Time Domain

\[ x[m] \]

N samples

Frequency Domain

Forward DFT

\[ X[k] \]

N/2 + 1 samples (other wave amplitudes)

In matlab:

\[ \text{Mag} X[k] = (\text{Re} X[k]^2 + \text{Im} X[k]^2)^{1/2} \]

\[ \text{Phase} X[k] = \arctan\left( \frac{\text{Im} X[k]}{\text{Re} X[k]} \right) \]

In matlab:

\[ \text{abs}(X) = \text{mag.} \]

\[ \text{angle}(X) = \text{phase} \]

Stored as a complex number:

\[ X = a + j b \]

\[ M = (A^2 + B^2)^{1/2} \]

\[ \theta = \arctan(B/A) \]
Discrete Fourier Transform

**Time domain signal**

**nth Harmonics**

\[ X(n) = \sum_{m=0}^{N-1} x(m)\{\cos\left(\frac{2\pi nm}{N}\right) - j\sin\left(\frac{2\pi nm}{N}\right)\} \]

**Spectral amplitudes**

(Frequency Domain)

\( X(n) \) in the frequency domain will have a real part (projection of time series on to cosine harmonic \( n \)) and an imaginary part (projection of time series on to sine harmonic \( n \)).

---

**Discrete Fourier Transform**

(\text{alternative complex # formula})

\[ X(n) = \sum_{m=0}^{N-1} x(m)e^{-j\frac{2\pi nm}{N}} \]

**Spectral amplitudes**

(Frequency Domain)

**N total points long**

---

**Example**

- \( x=[6 \ 4 \ 9 \ 0 \ 1 \ 5 \ 2 \ 7] \)

\[
\begin{array}{c|c|c}
\text{Columns} & 1 \text{ through } 3 & 4 \text{ through } 6 \\
\hline
34.0000 & 9.2426 - 1.3431i & -4.0000 - 2.0000i \\
0.7574 + 12.6569i & 2.0000 - 0.0000i & 0.7574 - 12.6569i \\
-4.0000 + 2.0000i & 9.2426 + 1.3431i &
\end{array}
\]

\( \text{DC component} \)

**Note:** There is some additional normalization that needs to be done... We'll get to this in a moment.
Consider only zero harmonic (n=0)

\[ X(n) = \sum_{m=0}^{N-1} x(m)\{\cos\left(\frac{2\pi nm}{N}\right) - j\sin\left(\frac{2\pi nm}{N}\right)\} \]

Last row gives the sum (or amplitude) of each harmonic
\[ x = [6 \ 4 \ 9 \ 0 \ 1 \ 5 \ 2 \ 7] \]

\[ X(n) = \sum_{m=0}^{N-1} x(m) \{ \cos\left(\frac{2\pi nm}{8}\right) - j\sin\left(\frac{2\pi nm}{8}\right) \} \]

**Consider only first harmonic (n=1)**

<table>
<thead>
<tr>
<th>m samples</th>
<th>n=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.0 + 0.0j</td>
</tr>
<tr>
<td>1</td>
<td>2.8 - 2.8j</td>
</tr>
<tr>
<td>2</td>
<td>0.0 - 9.0j</td>
</tr>
<tr>
<td>3</td>
<td>0.0 + 0.0j</td>
</tr>
<tr>
<td>4</td>
<td>-1.0 - 0.0j</td>
</tr>
<tr>
<td>5</td>
<td>-3.5 + 3.5j</td>
</tr>
<tr>
<td>6</td>
<td>-0.0 + 2.0j</td>
</tr>
<tr>
<td>7</td>
<td>4.9 + 4.9j</td>
</tr>
<tr>
<td>( \Sigma )</td>
<td>9.2 - 1.3j</td>
</tr>
</tbody>
</table>

\[ 4 \times \cos\left(\frac{2\pi \cdot 1 \times 1}{8}\right) - 4 \times j\sin\left(\frac{2\pi \cdot 1 \times 1}{8}\right) \]

\[ 2 \times \cos\left(\frac{2\pi \cdot 1 \times 6}{8}\right) - 2 \times j\sin\left(\frac{2\pi \cdot 1 \times 6}{8}\right) \]

**Consider only third harmonic (n=3)**

<table>
<thead>
<tr>
<th>m samples</th>
<th>n=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6.0 + 0.0j</td>
</tr>
<tr>
<td>1</td>
<td>-2.8 - 2.8j</td>
</tr>
<tr>
<td>2</td>
<td>0.0 + 9.0j</td>
</tr>
<tr>
<td>3</td>
<td>0.0 + 0.0j</td>
</tr>
<tr>
<td>4</td>
<td>-1.0 - 0.0j</td>
</tr>
<tr>
<td>5</td>
<td>3.5 + 3.5j</td>
</tr>
<tr>
<td>6</td>
<td>0.0 - 2.0j</td>
</tr>
<tr>
<td>7</td>
<td>-4.9 + 4.9j</td>
</tr>
<tr>
<td>( \Sigma )</td>
<td>0.8 + 12.7j</td>
</tr>
</tbody>
</table>

\[ 9 \times \cos\left(\frac{2\pi \cdot 3 \times 2}{8}\right) - 9 \times j\sin\left(\frac{2\pi \cdot 3 \times 2}{8}\right) \]

\[ 7 \times \cos\left(\frac{2\pi \cdot 3 \times 7}{8}\right) - 7 \times j\sin\left(\frac{2\pi \cdot 3 \times 7}{8}\right) \]

**Fast Fourier Transform**

**USE THIS – NOT THE MUCH SLOWER DFT**

```matlab
>> x=fft(x)
Columns 1 through 3
34.0000 -1.34311 -4.0000 -2.000001
Columns 4 through 6
0.7574 +12.6569i 2.0000 0.7574 -12.6569i
Columns 7 through 8
-4.0000 +2.00001 9.2426 +1.34311
```

**Columns:** 1 through 8
**Values:** Complex numbers representing the Fourier Transform of the input signal.
Example (on this slide we make a time series with known sinusoids)

```matlab
m = 0:99; % number of points N = 100
fs = 200; % sampling frequency
Ts = 1/fs; % sampling period
x = cos(2*pi*20*m*Ts + pi/4) +
    3*cos(2*pi*40*m*Ts - 2*pi/5) +
    2*cos(2*pi*60*m*Ts + pi/8);
t=linspace(0,99*Ts,length(x)); % for plotting
X = fft(x); m = 0:length(X)-1; N=length(X)
% Plot magnitudes
subplot(2,1,1);
f=m*fs/N; % freq axis
stem(f,abs(X));
% Plot phase angles
subplot(2,1,2);
stem(f,angle(X));
```

Now transform into the frequency domain using the FFT!

```matlab
X = fft(x); m = 0:length(X)-1; N=length(X)
% Plot magnitudes
subplot(2,1,1);
f=m*fs/N; % freq axis
stem(f,abs(X));
% Plot phase angles
subplot(2,1,2);
stem(f,angle(X));
```

Three Problems with this plot

1. We have 100 (N) frequencies spanning 0-200 Hz, but we only need 51 (N/2+1) spanning 0-Nyquist (100 Hz)
2. The Amplitudes are all wrong!
3. Phase plot is messy. We don’t care about the phase for frequencies with no amplitude.
\[ X = \text{fft}(x); \quad m = 0:\text{length}(X)-1; \quad N=\text{length}(X) \]
\[ \text{half}_m=\text{ceil}(N/2)+1; \quad \% \ 51 \]
\[ f=m(1:\text{half}_m)*fs/N; \quad \% \ \text{frequency axis} \]
\[ X=X(1:\text{half}_m); \quad \% \ \text{first half of the FFT output} \]
\[ \text{stem}(f,\text{abs}(X),’b’); \]

**FFT is symmetric, so just take the first half**

**Now rescale the amps**

\[ X=X^*2/N; \quad \% \ \text{multiple by 2 (since we took } 1/2 \text{ the data and divide by the total number of points).} \]
\[ X(1)=X(1)/2; \quad \% \ \text{undouble the first DC term} \]
\[ \text{stem}(f,\text{abs}(X),’b’); \]

\[ \text{tolerance} = 0.00001; \quad \% \]
\[ X2 = \text{ceil}(\text{abs}(X) - \text{tolerance}); \quad \% \ \text{flag small phase’s as 0} \]
\[ X3 = \text{round}(X2 ./ (X2+1)); \quad \% \ X3 \ \text{is now 0 or 1} \]
\[ \text{stem}(f,\text{angle}(X).*X3, ’b’); \quad \% \ \text{multiple } X3 \ \text{by phase} \]

% Example #2 in lecture 14
% Set up an example signal
m = 0:99; \% number of points
fs = 200; \% sampling frequency Hz
Ts = 1/fs; \% sampling period sec

\[ x = \cos(2\pi*20*m*T) + \pi/4) + \ldots \]
\[ 3\cos(2\pi*40*m*T - 2\pi/5) + \ldots \]
\[ 2\cos(2\pi*60*m*T + \pi/8); \]

% x is our example signal
\[ t=\text{linspace}(0,99*Ts, \text{length}(x)); \quad \% \ \text{time axis for plotting} \]
figure; plot(t,x,’LineWidth’,2); xlabel(’time sec’); ylabel(’amplitude’);
% Do the FFT
X = fft(x);
m = 0:length(X)-1; N=length(X);

disp(sprintf('Freq resolution is every %5.2f Hz',... fs/length(X)));

% Plot magnitudes
figure; subplot(2,1,1);
stem(m*fs/length(X),abs(X), 'b');
ylabel('magnitude'); xlabel('frequency (Hz)');
title('Frequency magnitude response');

% Plot phase angles
subplot(2,1,2);
stem(m*fs/length(X),angle(X), 'b');
ylabel('phase angle'); xlabel('frequency (Hz)');
title('Phase angle plot');

% take the first half of the FFT
half_m=ceil(N/2)+1; % only keep the first half
f=m(1:half_m)*fs/N; % frequency axis
X=X(1:half_m); % take first half of the FFT output

%Now rescale the amps
figure; subplot(2,1,1);
X=X*2/N;
X(1)=X(1)/2; % don't double the first
stem(f,abs(X),'b');
ylabel('Amplitude');xlabel('frequency (Hz)');
title('Frequency magnitude response - correctly normalized');

% The next 3 lines allow us to ignore phases with small amp
tolerance = 0.00001;
X2 = ceil(abs(X) - tolerance);
X3 = round(X2 ./ (X2+1)); % X3 is zeros and ones

subplot(2,1,2);
stem(f,angle(X).*X3, 'b');
title('phases'); xlabel('Frequency (Hz)');ylabel('Phase (rad)');
title('Phase spectrum with small amps zeroed');

Other issues….

Zero Padding with FFT

For efficiency the length of the time series is typically taken to be $2^{power}$.

If data are shorter, then add zeroes to the ends.

This has no effect on the result, but does change the freq resolution:

Frequency interval is $fs/NFFT$

Where $NFFT$ stands for number of points in FFT.
>> a = [1 2 3 4];
b = padarray(a,[0 2])
b =
0 0 1 2 3 4 0 0

OR

npow=nextpow2(length(a)) % = 3
NFFT=2^npow % NFFT = 8
fft(x,NFFT)

Other issues....
Spectral Leakage – spreading out of the spectra

Caused by:

• Frequency bins may not line up exactly with the content of the signal
• The fact that the signal ends abruptly (i.e., the signal is rectangularly windowed)

fs1 = 1000; % 1000 Hz sample rate
Ts1 = 1/fs1;
m1 = 0:99;
x1 = 3*cos(2*pi*200*m1*Ts1 - 7*pi/8) +...
2*cos(2*pi*300*m1*Ts1) +...
cos(2*pi*400*m1*Ts1 + pi/4);
% plot
t1=linspace(0,m1(end)*Ts1,length(x1));
figure; subplot(2,1,1); plot(t1,x1);

X1 = 2*abs(fft(x1))/length(x1);
N1=length(x1)
half_m1=ceil(N1/2)+1; % only keep the first half
f1=(0:half_m1-1)*fs1/N; % frequency axis
X1=X1(1:half_m1); % take first half of the FFT output
Now change the sample rate a bit:
fs2 = 1013;
Ts2 = 1/fs2;
n2 = 0:98;

N2=length(x2);
NFFT2=2^nextpow2(N2); % # points in FFT
X2 = 2*abs(fft(x2,NFFT2))/N2;
half_m2=ceil(NFFT2/2)+1; % only keep the first half
f2=(0:half_m2-1)*fs2/NFFT2; % frequency axis
X2=X2(1:half_m2); % take first half of the FFT output

Smaller freq bins = better frequency resolution