EEE411: Advanced Signal Processing

Signal Processing in MATLAB: Instructional Examples

- **Fourier Transforms**

Every signal can be written as a sum of sinusoids with different amplitudes and frequencies. The MATLAB command to compute the Fourier Transform and its inverse are respectively **fft** and **ifft**, for example:

```
>> x = rand(1,10); % suppose 10 samples of a random signal
>> y = fft(x);     % Fourier transform of the signal
>> iy = ifft(y);   % inverse Fourier transform
>> x2 = real(iy);  % chop off tiny imaginary parts
>> norm(x-x2);     % compare original with inverse of transformed
```

The **fft** is the abbreviation of **F**ast **F**ourier **T**ransform. This algorithm implements the discrete Fourier transform to transform data from time into the frequency domain. The study of this algorithm is normally covered in a good linear algebra course. First we give an example of the meaning of the Fourier transform before showing how Fourier transforms can be used to filter noise from signals.

- **Waveform and Amplitude Spectrum**

Suppose we sample a signal during 4 seconds, at a sampling rate of 0.01:

```
>> dt = 1/100;                              % sampling rate
>> et = 4;                                  % end of the interval
>> t = 0:dt:et;                             % sampling range
>> y = 3*sin(4*2*pi*t) + 5*sin(2*2*pi*t); % sample the signal
```

A natural plot is that of amplitude versus time:

```
>> subplot(2,1,1);                          % first of two plots
>> plot(t,y); grid on                       % plot with grid
>> axis([0  et  -8  8]);                    % adjust scaling
>> xlabel('Time (s)');                      % time expressed in seconds
>> ylabel('Amplitude');                     % amplitude as function of time
```

With the Fourier Transform we can visualize what characterizes this signal the most. From the Fourier transform we compute the amplitude spectrum:

```
>> Y = fft(y);                              % compute Fourier transform
>> n = size(y,2)/2;                         % 2nd half are complex conjugates
>> amp_spec = abs(Y)/n;                     % absolute value and normalize
```

To visualize the amplitude spectrum, we execute the following commands

```
>> subplot(2,1,2);                          % second of two plots
>> freq = (0:79)/(2*n*dt);                  % abscissa viewing window
>> plot(freq,amp_spec(1:80)); grid on       % plot amplitude spectrum
>> xlabel('Frequency (Hz)');                % 1 Herz = number of cycles/second
>> ylabel('Amplitude');                     % amplitude as function of frequency
```

On the amplitude spectrum we see two peaks: at 2 and 4. The location of the peaks occurs at the two frequencies in the signal. The heights of the peaks (5 and 3) are the amplitudes of the sines in the signal.

- **Filtering Noise from Signals**

We will see now how to use **fft** and **ifft** to filter out the noise from signals. First we add random noise to our signal and compute the amplitude spectrum.

```
>> noise = randn(1,size(y,2));        % random noise
>> ey = y + noise;                    % samples with noise
>> eY = fft(ey);                      % Fourier transform of noisy signal
>> n = size(ey,2)/2;                  % use size for scaling
>> amp_spec = abs(eY)/n;              % compute amplitude spectrum
```

To interpret these calculations we make a plot of the waveform and amplitude spectrum:

```
>> figure                             % plots in new window
>> subplot(2,1,1);                    % first of two plots
>> plot(t,ey); grid on                % plot noisy signal with grid
>> axis([0  et  -8  8]);              % scale axes for viewing
>> xlabel('Time (s)');                % time expressed in seconds
>> ylabel('Amplitude');               % amplitude as function of time
>> subplot(2,1,2);                    % second of two plots
>> freq = (0:79)/(2*n*dt);            % abscissa viewing window
>> plot(freq,amp_spec(1:80)); grid on % plot amplitude spectrum
>> xlabel('Frequency (Hz)');          % 1 Herz = number of cycles per second
>> ylabel('Amplitude');               % amplitude as function of frequency
```

On the first plot we recognize the shape of the signal. In the plot of the amplitude spectrum, the peaks and their heights are the same as on the plot of the amplitude spectrum of the original signal. The wobbles we see around the peaks show that the amplitude of the noise is less than that of the original signal. We can visualize the output of the Fourier transforms:

```
>> figure                             % new window for plot
>> plot(Y/n,'r+')                     % Fourier transform of original
>> hold on                            % put more on same plot
>> plot(eY/n,'bx')                    % Fourier transform of noisy signal
```

Via the inverse Fourier transform, we filter out the noise. The command **fix** rounds the elements of its argument to the nearest integers towards zero. For this example, we use **fix** to set all elements in eY less than 100 to zero:

```
>> fY = fix(eY/100)*100;              % set numbers < 100 to zero
>> ifY = ifft(fY);                    % inverse Fourier transform of fixed data
>> cy = real(ifY);                    % remove imaginary parts
```

The vector `cy` contains the corrected samples. So, finally we plot this corrected signal:

```
>> figure                             % new window for plot
>> plot(t,cy); grid on                % plot corrected signal
>> axis([0  et  -8  8]);              % adjust scale for viewing
>> xlabel('Time (s)');                % time expressed in seconds
>> ylabel('Amplitude');               % amplitude as function of time
```

Here we filtered out noise of low amplitude. Note we can also remove noise of high frequency.

Additional Materials:

1. FIR filter design: https://www.mathworks.com/help/signal/ug/fir-filter-design.html
2. IIR filter design: https://www.mathworks.com/help/signal/ug/iir-filter-design.html
3. Signal Processing Examples: https://www.mathworks.com/examples/signal