EE1/EIE1: Introduction to Signals and Communications

MATLAB Experiments

Professor Kin K. Leung

January 2018

Experiment 1: Fourier Transform of Rectangular Pulse Signal

Introduction: The discrete Fourier transforms (DFTs), which are Fourier transforms of a collection of signal samples (e.g., those obtained by sampling a continuous-time signal), is the powerful tool of digital signal processing. DFTs are often computed by a technique named fast Fourier transforms (FFTs), which is designed to compute DFTs with reduced execution time. The MATLAB® software provides the `fft` and `ifft` functions to compute the discrete Fourier transforms and their inverse (both based on the FFT technique), respectively. These two functions will be used in our experiments in the following.

Experiment 1.1 – Convert signal from time to frequency domain

In this experiment, we generate a rectangular pulse signal *f(t)* in time domain and then take the Fourier transform of it. First, copy and paste the following code to the MATLAB command window in order to generate and visualize the rectangular pulse signal in time domain.

```
clear;clc                       % clear command history and all variables
T = 20;                         % tunable parameter for the signal width
dt=.001;                        % increment
t=[-(10+T):dt:(10+T)];          % range of the signal
x=sign(t+T)-sign(t-T);          % generate the rectangular pulse signal
plot(t,x);                      % visualize the signal in time domain
title('Pulse signal');          % title of the plot
xlabel('Time (msec)');          % label x-axis
ylabel('Signal f(t)');          % label y-axis
axis([-(30+T) (30+T) 0 3]);     % set display range of x- and y-axis
```

Use the following MATLAB code to use the `fft` function to perform Fourier transform on the generated rectangular pulse signal and visualize the magnitude of the rectangular pulse signal in frequency domain.

```
y=fftshift(fft(x));             % apply Fourier transform and move zero
                                frequency component to the center
N=length(y);                    % measure frequency range
```

```
n=-(N-1)/2:(N-1)/2;            % evenly divide frequency range around zero
                               frequency
f=sqrt(y.*conj(y));            % calculate amplitude of the frequency signal
plot(n,f);                     % visualize the signal in time domain
title('Frequency spectrum amplitude for the rectangular pulse');
                               % set title of the plot
xlabel('Frequency (Hz)');      % label x-axis
ylabel('Frequency spectrum amplitude');  % label y-axis
axis([-50 50 0 70000]);        % set display range of x- and y-axis
```

Observation: Change the tunable parameter **T** highlighted in red given above, therefore change the width of the time-domain signal. By cut and paste the revised code to the MATLAB command window, repeat the experiment and observe the changes of the corresponding FFTs in frequency domain.

Experiment 1.2 – Convert from frequency to time domain with all frequency components

We apply inverse Fourier transforms in frequency domain to recover the signal in time domain. The following script using MATLAB function `ifft` can achieve the inversion.

```
Y=ifft(y);                     % take the inverse Fourier transformation
plot (t,abs(Y));               % visualize the signal in time domain
title('Reconstruct the pulse signal from Fourier series');
                               % set title of the plot
xlabel('Time(ms)');            % label x-axis
ylabel('Recovered f(t)');      % label y-axis
axis([-(30+T) (30+T) 0 3]);    % set display range of x- and y-axis
```

Experiment 1.3 – Convert from frequency to time domain with ideal low-pass filter (i.e., loss of high-frequency signal components)

Instead of using all frequency-domain components to reconstruct the time-domain signal, we only select and use a range of low-frequency components to reconstruct the time-domain signal.

In order to achieve this, we construct a filter that only lets low-frequency signal components to pass through, while blocking high frequency components. Use the following MATLAB code to realize the low-pass filter.

```
w = 50;                            % tunable parameter for filter bandwidth
fil=sign(n+w)-sign(n-w);           % specify the low-pass filter
plot(n,fil);                       % visualize the filter response
title('Ideal low-pass filter');    % set title of the plot
xlabel('Frequency (Hz)');          % label x-axis
ylabel('Amplitude');               % label y-axis
axis([-(w+20) (w+20) 0 3]);        % set display range of x- and y-axis
```

It can be seen that the filter is a rectangular pulse signal around zero frequency in frequency domain. Then we apply this filter to the frequency-domain signal and perform the inverse Fourier transform on the filtered frequency-domain signal.

```
w = 50;                                % tunable filter bandwidth
fil=sign(n+w)-sign(n-w);               % specify the low-pass filter
f2=fil.*y;          % f2 is the filtered output of original frequency signal
Y1=ifft(f2)/2;      % apply inverse Fourier transform to the filtered signal
plot (t,abs(Y1)); % reconstructed filtered signal in time domain
hold on;            % hold for another plot
plot (t,abs(Y));  % reconstructed unfiltered signal in time domain
legend('Reconstructed signal with loss (filtered)', 'Reconstructed signal
without loss');    % legend of two plots
title('Compare signal reconstructed from filtered Fourier series with the
original signal');% set title of the plot
xlabel('Time (msec)');                  % label x-axis
ylabel('Original or reconstructed');  % label y-axis
axis([-(10+T) (10+T) 0 3]);             % set display range of x- and y-axis
```

Observation 1: Compare the two reconstructed time-domain signals using all frequency-domain components and partial frequency-domain components, respectively. What are the differences?

Observation 2: Vary the tunable parameter **w** highlighted in red in the code above, therefore change the bandwidth width of the low-pass filter (in frequency domain). Repeat the experiment and observe the changes in the corresponding reconstructed time-domain signals.

Experiment 2: Fourier Transform of Audio Signals

This section experiments the Fourier transform of audio signals. Specifically, Experiment 2.1 converts several downloaded audio signals in time domain to frequency domain, whereas experiment 2.2 gives you the opportunity to record and analyse your own voice signal.

Experiment 2.1 - Fourier transform of downloaded audio signals

In this experiment, we perform Fourier transform of the following audio sources:

Male human voice http://www.kozco.com/tech/LRMonoPhase4.wav
Piano http://www.kozco.com/tech/piano2.wav

Snare drum http://audio.routledge-interactive.s3.amazonaws.com/9780240821535/mixing_home/gated_kick.mp3

Please click the above URL links to download the audio files. Make sure that these audio files are downloaded into your current MATLAB directory. First, use the following MATLAB code to load and visualize the signal in time domain.

```
s = audioread('LRMonoPhase4.wav');
s = s(:,1); % extract one sound track only
            % load different audio source by changing the file name in ('')
plot(s);    % visualize the signal in time domain
```

Next, perform Fourier transform and visualize the audio signal in frequency domain. Use the following MATLAB code.

```
Fs = 44100;                     % sample rate of the audio signal
N = length(s);                  % the number of samples of the audio signal
transform = fft(s,N);           % apply Fourier transform
magTransform = abs(transform); % magnitude of the FFT
faxis = ((-0.5:1/N:0.5-1/N)*Fs).';  % frequency range of the signal
plot(faxis,fftshift(magTransform));
xlabel('Frequency (Hz)');
ylabel('Spectrum magnitude');
xlim([-1000 1000]);
```

Using the following code, you can plot all three signals in a single plot to compare the differences of their frequency-domain components among the audio signals.

```
s1 = audioread('LRMonoPhase4.wav');
s2 = audioread('piano2.wav');
s3 = audioread('gated_kick.mp3');
Fs = 44100;
N1 = length(s1);
N2 = length(s2);
N3 = length(s3);
transform1 = fft(s1,N1);
transform2 = fft(s2,N2);
transform3 = fft(s3,N3);
magTransform1 = abs(transform1);
magTransform2 = abs(transform2);
magTransform3 = abs(transform3);
faxis1 = ((-0.5:1/N1:0.5-1/N1)*Fs).';
faxis2 = ((-0.5:1/N2:0.5-1/N2)*Fs).';
faxis3 = ((-0.5:1/N3:0.5-1/N3)*Fs).';
plot(faxis1,fftshift(magTransform1(:,1)),'m-+');
hold on
plot(faxis2,fftshift(magTransform2(:,1)),'b-s');
plot(faxis3,fftshift(magTransform3),'k-p');
xlim([-2000 2000]);
lgd = legend('male human voice','piano','snare drum');
lgd.FontSize = 10;
set(gca,'fontsize',10)
grid on
xlabel('Frequency (Hz)');
```

Experiment 2.2 - Fourier transforms of your own voice signal

In this experiment, you have the opportunity to record and analyse your own voice signal. Please make sure that the microphone is turned on and connected to your computer. The audio recording is done via the `audiorecorder` function in MATLAB. This function is used to create the `audiorecorder` object, which may take the following parameters:

1) `Fs` corresponds to the sampling frequency (in Hz) that will be applied to your voice signal. You need to choose one of the standard values: 8000, 11025, 22050, or 44100. Remember that from the sampling theorem, the sampling frequency should be at least twice the maximum frequency of the signal. You may choose 44100 Hz, if you want to work on a signal with a wide frequency range.
2) `nbits` corresponds to the number of bits used to represent each sample. The standard values are 8, 16, 24 or 32 bits where the last two are only available on 24-bit and 32-bit sound devices. A reasonable choice is 16 bits to represent each sample, which will be used in this experiment.
3) `channels` is the number of channels used in the recording. Possible values are 1 (for mono) and 2 (for stereo). If you are recording using only one microphone, you just need 1 channel;
4) `id` corresponds to the DeviceID of the device that is being used. Here, we can use the value found in `audiodevinfo`.

The following setting serves as an example:

```
Fs = 44100;
nbits = 16;
dev_id = getfield(getfield(audiodevinfo, 'input'), 'ID');
                              % obtain the ID of the computer's soundcard
arec = audiorecorder(Fs, nbits, 1, dev_id);
                              % create the audiorecorder object
```

After creating the `audiorecorder` object, you can start recording your voice by using a microphone connected to the mic input of a computer. To start, pause, resume and terminate the recording, use the following commands, respectively.

```
record(arec)
pause(arec)
resume(arec)
stop(arec)
```

Then, use the following command to extract the recorded data.

```
data = getaudiodata(arec);
```

Finally, use the recorded voice signal as input and repeat the steps in Experiment 2.1, use the following code.

```
Fs = 44100;                      % sample rate of the audio signal
N = length(data);                % the number of audio signal samples
transform = fft(data,N);         % apply Fourier transform
magTransform = abs(transform); % magnitude of the FFT
faxis = ((-0.5:1/N:0.5-1/N)*Fs).';  % frequency range of the signal
plot(faxis,fftshift(magTransform));
xlabel('Frequency (Hz)');
ylabel('Spectrum magnitude');
```

Experiment 3: Modulation

You will experiment signal modulation in MATLAB. Two modulation methods, namely Amplitude Modulation (AM) and Frequency Modulation (FM), will be explored here. In general, any type of modulation method typically involves three main steps: (1) generate the modulating signal, (2) generate the carrier signal, and (3) modulate the carrier signal with the modulating signal. These steps will first be demonstrated separately in the AM experiment below.

Experiment 3.1 - Amplitude modulation (AM)

First, generate and visualize the modulating signal using the following MATLAB code.

```
Am=2;                            % set the amplitude of modulating signal
fa=2000;                         % set the frequency of modulating signal
Ta=1/fa;                         % set the time period of modulating signal
t=0:Ta/999:6*Ta;                 % set the total time for simulation
ym=Am*sin(2*pi*fa*t);            % synthesize the modulating signal
plot(t,ym);                      % visualize the modulating signal
grid on;                         % include grid in the plot
title ('Modulating signal');
xlabel ('Time');
ylabel ('Modulating signal m(t)');
ylim([-5 5]);
```

Next, generate and visualize the carrier signal using the following MATLAB code.

```
Ac=3;                            % set the amplitude of carrier signal
fc=fa*10;                        % set the frequency of carrier signal
```

```
Tc=1/fc;                          % set the time of carrier signal
yc=Ac*sin(2*pi*fc*t);             % synthesize the carrier signal
plot(t,yc);                       % visualize the carrier signal
grid on;                          % include grid in the plot
title ('Carrier signal');
xlabel ('Time (sec)');
ylabel ('Carrier signal fc(t)');
ylim([-5 5]);
```

Finally, modulate the carrier signal with the modulating signal, and visualize the modulated signal using the following MATLAB code.

```
A=3;                                          % AM carrier offset
y=(A+Am*sin(2*pi*fa*t)).*sin(2*pi*fc*t);  % Full AM with carrier offset A
plot(t,y);
grid on;
title ('Amplitude modulated signal');
xlabel ('Time (sec)');
ylabel ('Signal');
axis([0 2*10^(-3) -7 7]);
```

In order to compare the modulating signal, the carrier signal, and the modulated signal in time domain, you can plot them on top of each other in the same figure using MATLAB function `subplot`.

```
subplot(3,1,1);plot(t,ym);
ylim([-6 6]);
ylabel('Signal');xlabel('Time');title('Modulating signal');
subplot(3,1,2);plot(t,yc);
ylim([-6 6]);
ylabel('Signal');xlabel('Time');title('Carrier signal');
subplot(3,1,3);plot(t,y);
ylim([-6 6]);
ylabel('Signal');xlabel('Time');title('Full AM modulated signal');
```

Also, it is helpful to visualize the frequency-domain equivalents of the three time-domain signals using the following MATLAB code.

```
Fs = 10000;
y1=fftshift(fft(ym,length(ym)));
N1=length(y1);
n1=(-0.5:1/N1:0.5-1/N1)*Fs;
f1=sqrt(y1.*conj(y1));
y2=fftshift(fft(yc,length(ym)));
N2=length(y2);
n2=(-0.5:1/N2:0.5-1/N2)*Fs;
f2=sqrt(y2.*conj(y2));
y3=fftshift(fft(y,length(y)));
N3=length(y3);
n3=(-0.5:1/N3:0.5-1/N3)*Fs;
f3=sqrt(y3.*conj(y3));
subplot(3,1,1);plot(n1,f1);  xlim([-150 150]);
ylabel('Amplitude');xlabel('Frequency');title('Modulating signal');
subplot(3,1,2);plot(n2,f2);  xlim([-150 150]);
ylabel('Amplitude');xlabel('Frequency');title('Carrier signal');
subplot(3,1,3);plot(n3,f3);  xlim([-150 150]);
ylabel('Amplitude');xlabel('Frequency');title('Full AM signal');
```

Experiment 3.2 - Frequency modulation (FM)

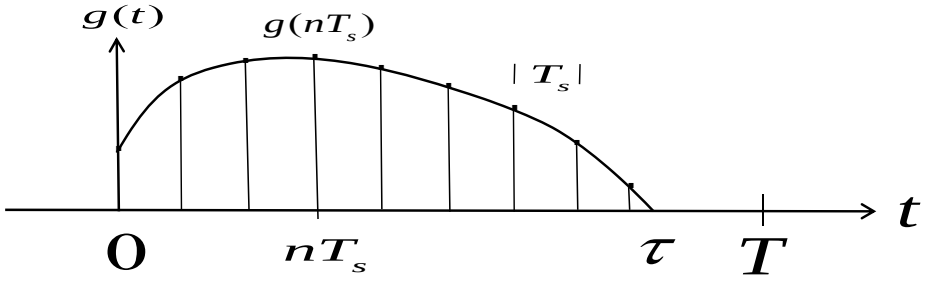
To experiment with FM in MATLAB, the three steps given above are applicable. However, instead of using pre-determined parameters, this time you will be given the opportunity to set experiment parameters, including the carrier frequency and the modulating signal frequency, by pop-up prompt in the command window. Use the following MATLAB code to achieve this.

```
clear;clc
fm=input('Message Frequency = (recommended value: 25)');
fc=input('Carrier Frequency = (recommended value: 400)');
mi=input('Modulation Index = (recommended value: 10)');
t=0:0.0001:0.11;
m=sin(2*pi*fm*t);
c=sin(2*pi*fc*t);
y=sin(2*pi*fc*t+(mi.*-cos(2*pi*fm*t)));
                          % note that the integral of sin(x) is -cos(x);
figure;
subplot(3,1,1);plot(t,m); ylim([-2 2]);
ylabel('Signal');xlabel('Time');title('Modulating signal');
subplot(3,1,2);plot(t,c); ylim([-2 2]);
ylabel('Signal');xlabel('Time');title('Carrier signal');
subplot(3,1,3);plot(t,y); ylim([-2 2]);
ylabel('Amplitude');xlabel('Time');title('FM signal');
Fs = 10000;
y1=fftshift(fft(m,length(m)));
N1=length(y1);
n1=(-0.5:1/N1:0.5-1/N1)*Fs;
f1=sqrt(y1.*conj(y1));
y2=fftshift(fft(c,length(c)));
N2=length(y2);
n2=(-0.5:1/N2:0.5-1/N2)*Fs;
f2=sqrt(y2.*conj(y2));
y3=fftshift(fft(y,length(y)));
N3=length(y3);
n3=(-0.5:1/N3:0.5-1/N3)*Fs;
f3=sqrt(y3.*conj(y3));
figure;
subplot(3,1,1);plot(n1,f1); ylim([0 600]); xlim([-1000 1000]);
ylabel('Amplitude');xlabel('Frequency');title('Modulating signal');
subplot(3,1,2);plot(n2,f2); ylim([0 600]); xlim([-1000 1000]);
ylabel('Amplitude');xlabel('Frequency');title('Carrier signal');
subplot(3,1,3);plot(n3,f3); ylim([0 600]); xlim([-1000 1000]);
ylabel('Amplitude');xlabel('Frequency');title('FM signal');
```

Similar to Experiment 3.1, the above code generates one figure showing the modulating signal, the carrier signal and the frequency modulated signal, as well as a second figure showing their frequency-domain equivalence. You can re-run this experiment with the modulating and carrier signals of different frequencies.

Discrete Fourier Transform (DFT)

- Discrete Fourier Transform (DFT)
  - Numerical computation of Fourier Transform
- Given a signal $g(t)$, which is non-zero for $0 \le t \le \tau$
  - Select $T \ge \tau$ and set $g(t) = 0$ for $\tau \le t \le T$



$$G(\omega) = \int_0^T g(t) e^{-j\omega t}\, dt$$

$$= \lim_{T_s \to 0} \sum_{n=0}^{N-1} g(nT_s) e^{-j\omega nTs} T_S \qquad \text{where} \quad N = T / T_S$$

Discrete Fourier Transform (DFT) and Fast Fourier Transform (FTT)

$$G(\omega) = \lim_{T_s \to 0} \sum_{n=0}^{N-1} g(nT_s) e^{-j\omega nTs} T_S \quad \text{where} \quad N = T / T_S$$

$$G_m = G(m\omega_0) = \sum_{n=0}^{N-1} g(nT_S) T_s e^{-jm\omega_0 nTs} \quad \text{where} \quad m = 0,1,...,N-1 \text{ and } \omega_0 = 2\pi / T$$

$$G_m = \sum_{n=0}^{N-1} g_n e^{-jmn\omega_0 T_S} \quad \text{where} \quad g_n \equiv g(nT_S) T_S$$

One can derive for *n=0,1, …, N-1*

$$g_n = \frac{1}{N} \sum_{m=0}^{N-1} G_m e^{jmn\omega_0 T_S}$$

$$\{g_0, g_1, ..., g_{N-1}\} \Leftrightarrow \{G_0, G_1, ..., G_{N-1}\}$$

- DFT computation complex $O(N^2)$
- Fast Fourier transform (FFT) by Tukey and Cooley 1965
- FFT computation complex $O(N \log N)$