COMM.SYS.300
COMMUNICATION THEORY

Matlab Exercise #1

Sampling, Fourier transform, Spectral illustrations, and Linear filtering

1 SAMPLING

The modeled signals and systems in this course are mostly analog (continuous-time). However, the computer based simulation environment is always digital (discrete-time), of course.

Thus, in Matlab, we have certain constraints from the DSP (Digital Signal Processing) world, such as

- A continuous time signal $x_c(t)$ is represented by a vector, $[x_c(t_0)\ x_c(t_1)\ ...\ x_c(t_{N-1})]$, whose length is N, i.e. the signal is truncated and sampled
- Time interval $t_{N-1} - t_0$ and sampling interval $T_s = t_{i+1} - t_i$ depend on the signal
- Time interval is chosen large enough to obtain enough information
- Sampling frequency is typically chosen much higher than the Nyquist frequency, $2 \cdot f_{max}$
    - Sampling frequency $F_s = 1/T_s$ is chosen enough large so that the sampled signal “looks like” an analog signal in Matlab. The rule of thumb is at least 10 times the maximum frequency component

EXAMPLE: RUN THE FOLLOWING STATEMENTS (IN THE COMMAND WINDOW OR IN A SCRIPT FILE) TO CREATE A 50 HZ SINUSOIDAL SIGNAL:

```
>> Fs=500;              % sampling frequency = 10*50Hz
>> Ts=1/Fs;             % sampling interval
>> t=0:Ts:0.1;          % sampling time instants [s]
>> x=sin(2*pi*50*t);    % signal vector
>> figure               % Opens a new figure
>> plot(t,x)            % plot in time domain
```

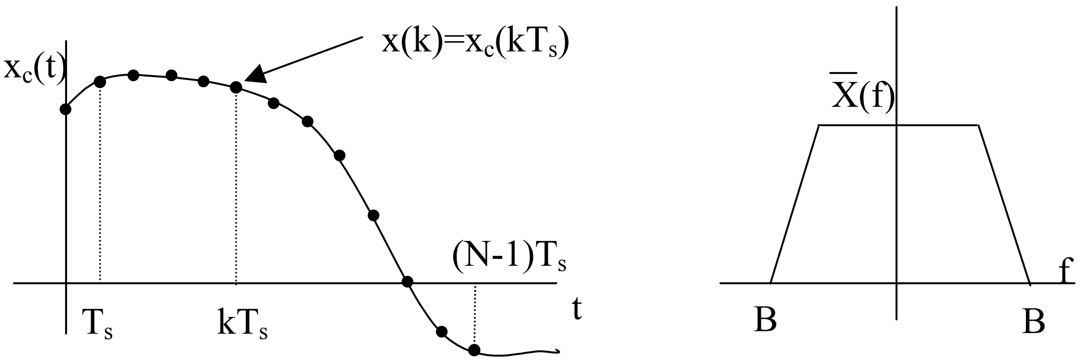
2 FOURIER TRANSFORM (FT)

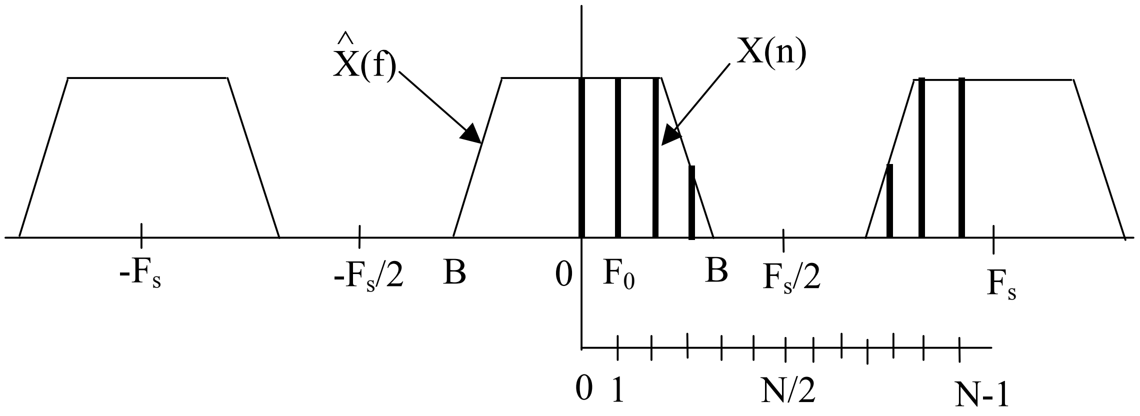
$$\overline{X}(j\omega)=\int_{-\infty}^{\infty} x_c(t)e^{-j\omega t}dt$$ Continuous time FT, CTFT

$$\hat{X}(e^{j\omega})=\sum_{k=-\infty}^{\infty} x(k)e^{-j\omega k}$$ Discrete time FT, DTFT

$$X(n)=\sum_{k=0}^{N-1} x(k)e^{-j2\pi kn/N},\quad n=0,1,\ldots,N-1$$ Discrete FT, DFT

Above and below $x(k)$ is the sampled signal from the $x_c(t)$





In Matlab, DFT can be calculated with the function "fft".

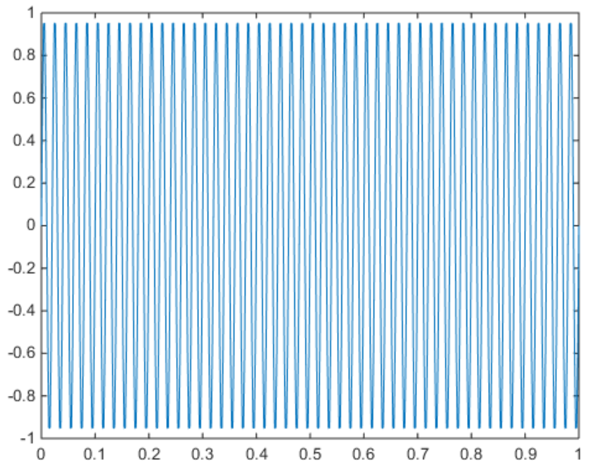
FFT calculation is not related to the actual sampling frequency F_s, but it is given as $0 \ldots ((N-1)/N)\cdot 2\pi$ where π refers to half the sample rate. For plotting you have to generate the vector of frequency points. Frequency resolution F_0 depends on sampling frequency and the length of the signal vector. With real signals all the information is between $0 - F_s/2$, but during this course we prefer to plot the spectrum in the two-sided format ($-F_s/2 \ldots F_s/2$).

Example: Run the following statements (in the command window or in a script file) to Plot the DFT of the 50 Hz sinusoidal signal:

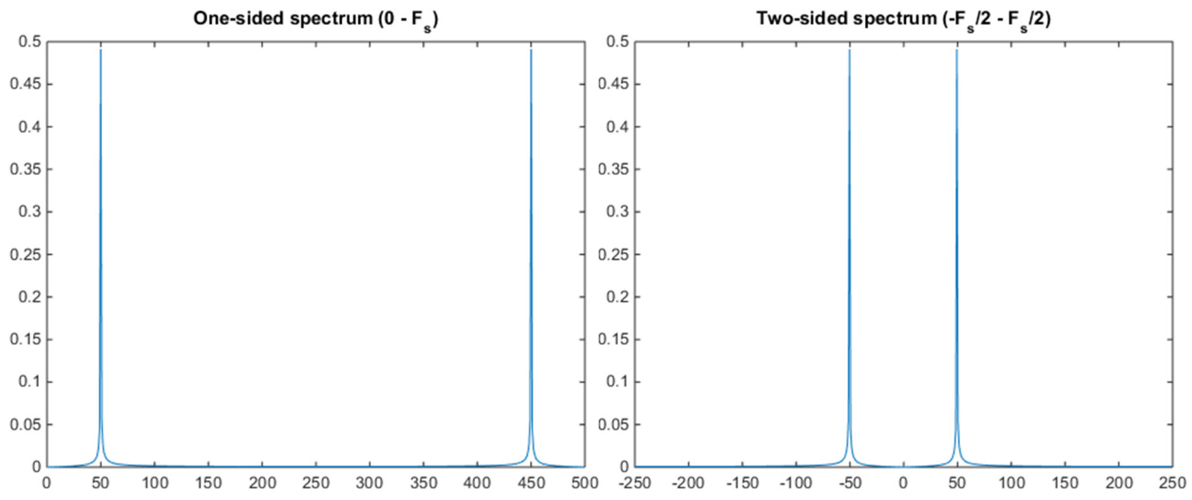
```
>>F_x=fft(x);                          % DFT of X, saved to Fx
>>Nx=length(x);
>>Fo=1/(Ts*Nx);                        % frequency resolution.
>>freq1=0:Fo:(Nx-1)*Fo;                % One-sided frequency Axis
>>figure
>>plot(freq1,abs(F_x)/Nx)              % One-sided amplitude Spectrum
>>freq2=-Nx/2*Fo:Fo:(Nx/2-1)*Fo;       % Two-sided frequency Axis
>>figure
>>plot(freq2,fftshift(abs(F_x)/Nx))    % Two-sided amplitude Spectrum
```

Scaling by Nx is required in Matlab in order to get correct amplitude values in the frequency domain. However, often it is more important to see the frequency content in general (and especially relative amplitudes between separate frequencies), in which case the scaling is not as essential. Fftshift is used to generate the two-sided spectrum plot ($-F_s/2 \ldots F_s/2$).

Time domain



Frequency domain



3 SPECTRAL ILLUSTRATIONS

At this point, if not yet done, you should create a new Matlab script-file (.m) and write there the rest of the Matlab commands in this exercise (do the same also for the following Matlab exercises). Below you'll find a few tips, which might be useful when working with the scripts:

- It can be useful to start the script as "`clear; close all; clc`", which automatically clears the old variables from the desktop (clear), closes all the open figures (close all), and clears the command window from the text (clc), when starting to run the script.
- Note that in the script you can "comment" any line or part of a line by adding the %-sign (note that the shortcut to comment the whole line at once is ctrl+r). After the %-sign, the following statements in the same line are ignored when the script is executed.
- You can execute/run a small part of the script by highlighting text, and then pressing F9. Now only the highlighted statements are executed.
- Remember also the Run-button in the editor window to run the whole script at once (this will also save the script)
- You can create a code section by adding two %-signs as %%. This might help in creating a more logical structure for the script (for example dividing the script into sections based on the exercise task numbers). You can run one section by selecting the section and pressing ctrl+Enter, or from the editor window by "Run section".

3.1 SIGNAL GENERATION

Generate a f_c = 800 MHz sinusoidal-signal (this is actually a frequency used in LTE (4G) networks in Finland) with sampling frequency of Fs = 16 GHz (=16000MHz).

- Determine sampling interval Ts=…
- Enter the length in samples N=…
- Generate the signal x=… ($x(t) = \sin(2\pi f_c t)$)
- Plot in time domain plot(t,x)
- Compute the length N and the frequency resolution Fo of the signal
- Compute DFT Fx=fft(x);
- Generate Frequency axis f=…
- Plot the <u>two-sided</u> (i.e., frequencies -Fs/2…Fs/2) amplitude spectrum of x: plot(f,…
- Add title and labels to the figure (help title, help xlabel, help ylabel)

```
%EXAMPLE CODE is shown with this font and color
% See the example of creating the signal, and plotting it in the time domain and
% frequency domain in Sections 1 and 2:
% ...
title('Time domain Plot of x(t)')
xlabel('t [s]')
ylabel('Amplitude')
axis([0 20e-9 -1.2 1.2])
%[x-axis_left_limit x-axis_right_limit y-axis_down_limit y-axis_up_limit]
```

Notice that the time and frequency plotting commands can be mostly copy-pasted for the corresponding plots in the following tasks!

3.2 MULTIPLICATION BETWEEN TWO SIGNALS

Let’s consider a new signal as $s(t) = x(t)\cdot m(t)$, where $m(t)$ is a sine wave with frequency of 750 MHz.

- Generate the new sinusoidal signal m = sin(..)
- Multiply the x and m signals s = … (remember the element-wise multiplication .*)
- Plot in time domain: plot(t,s)
- Plot the two-sided amplitude spectrum of s(t): plot(f,…

3.3 ADDING A NOISE SIGNAL

Let’s consider a model for a noisy signal as $y(t) = 10\cdot s(t) + n(t)$, where $s(t)$ is the signal from the above multiplication task (3.2) and $n(t)$ is white Gaussian noise with zero mean and variance of 1.

- Generate the noise signal `n = randn(size(x))` (help randn)
- Add the noise signal to signal x to get the noisy signal `y = 10*s + n;`
- Plot in time domain plot(t,y)
- Plot the two-sided amplitude spectrum of y so that the frequency-axis is given in MHz
    - I.e. divide the frequency axis values by 1 MHz:

```
plot(f/1e6,fftshift(abs(F_y))/N)
```

4 LINEAR FILTERING

4.1 LOW PASS BUTTERWORTH FILTER

- Create a low pass Butterworth filter (help butter)
    - Order of 10
    - Cut-off frequency of 200MHz
        - f_cut = …
        - order= …

```
fr=f_cut/(Fs/2); % Cut-off frequency normalized to 1.
[b,a]=butter(order,fr); % Coefficients of the filter (help butter)
```

- Plot the frequency response of the filter (help freqz)

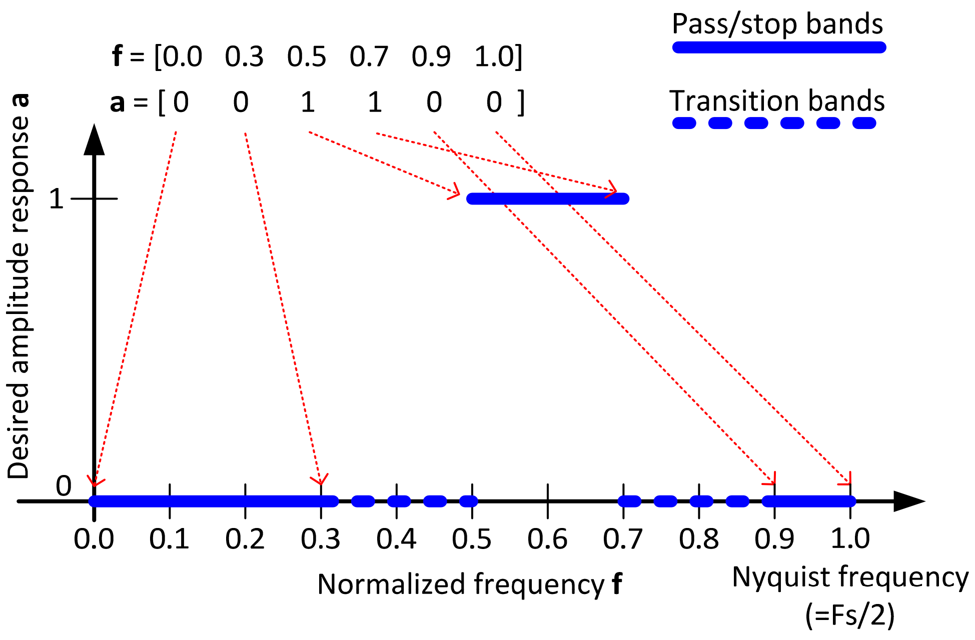
```
freqz(b,a,N,Fs)
title('Frequency response of the Butterworth filter')
```

- Filter the signal $y(t)$ (from task 3.3) with the generated Butterworth filter (help filter)
    - y_filtered_butter = filter(…);
- Plot the filtered signal in time and frequency domain
    - Notice the filter delay
- Compare the original $y(t)$ and the filtered $y(t)$ with each other in time domain and frequency domain
    - What “part” of the signal is still visible after filtering in time/frequency domain?

4.2 BAND PASS FIR (FINITE IMPULSE RESPONSE) FILTER

- Create a band pass FIR filter (help firpm, see the figure below)
    - Order of 60
    - Bass-band: 1.3 GHz…1.8 GHz
    - Transition bands: 0.8 GHz…1.3 GHz and 1.8 GHz…2.3 GHz

Illustration of arbitrary chosen **f** and **a** vectors used in the bass band filter design with firpm (b = firpm(n,f,a), where n is the filter order, and vectors f and a are illustrated below)



f = [0.0 0.3 0.5 0.7 0.9 1.0]
a = [0 0 1 1 0 0]
Pass/stop bands
Transition bands
Desired amplitude response **a**
1
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Normalized frequency **f**
Nyquist frequency (=Fs/2)

```
order=60; % filter order
f_filter = [0 0.8e9 1.3e9 1.8e9 2.3e9 Fs/2]/(Fs/2);
a_filter = [0 0 1 1 0 0];
b = firpm(order,f_filter,a_filter);
```

- Plot the impulse response of the filter by using stem-function (help stem)

```
stem(-order/2:order/2,b)
```

- Plot the amplitude response of the filter by using fft

```
F_b = fft(b,N); % We use same length with the frequency vector for the fft
```

- Filter the signal *y(t)* (from task 3.3) with the generated FIR filter (help filter)

```
y_filtered_FIR = filter(b,1,y);
```

- Plot the filtered signal in time and frequency domain
    - Again, notice the filter delay
- Compare *y(t)* and the filtered *y(t)* with each other in time domain and frequency domain
    - What “part” of the signal is still visible after filtering in time/frequency domain?