Laboratory 2C:

Fourier Series Application & Settings

In this laboratory, you experiment some applications and settings of the Fast-Fourier-Transformation (FFT) in non-ideal conditions (practical cases).

You use the FFT to decode the telephone dialling system DTMF or “Touch-Tone”, and check the effects of settings like windowing and zero-padding.

1. Decoder for DTMF Telephone Ton Dialing

The DTMF or „Touch-Tone“ (dual-tone multi-frequency signaling) is a widespread dialing technics in telephony to transmit the dialed number over the network. The corresponding symbols and frequencies are given in the table below:

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

(a) Load in Matlab (use the command ***load()***) the data file *touchtoneX.mat (x for A, B, C)*, where a sequence of audio tones representing a dialed number, and other parameter (like sampling frequency) are saved.

(b) Hear the audio sequence (use the command ***sound()***) and prepare a plot of the audio signal in the time domain.
Hint: check which variables where included in the structure you loaded.

(c) Extend your script to process the audio sequence and find out the three dialed numbers. Hints:

- Cut out nine intervals from the audio sequence representing each a digit from the dialed number, and use the ***fft()*** to analyse each interval.

- Prepare a plot of the spectrum and either manually or using the ***max()*** or ***findpeaks()*** functions, find out the two frequencies contained in each interval and the corresponding symbol (in the DTMF table).

2. Using the FFT in practical cases

The ideal condition to apply the ***fft()*** as a numerical approximation of the Fourier Series, is to take an observation window (N*Ts), which matches exactly one period or an integer number of periods from the incoming signal, and a fine resolution in the frequency domain (small value for Fs/N).

In practical cases this cannot be done, because the incoming signal is usually unknown, and you do not have the absolute control over the sampling frequency and length of the input buffer (e.g. on the oscilloscope, or on a microcontroller).

What can we expect then in such “real” cases, and how can we optimise or improve the calculated output?

(a) Try out the code below and calculate for the different N values the achievable resolution in the frequency domain. Check with the pointer the frequency values for the first harmonic, and determine which ck coefficient does it corresponds to.

```
% PARAMETERS
N = 2^9;              % number of points, try N=128, 256
aux = 0:1:N-1;        % auxiliary index vector
Fs = 80e3;            % sampling frequency

t = (1/Fs)*aux;
f = (Fs/N)*aux;

% FUNCTIONS
x_t = 2*square(2*pi*1.15e3*t);
X_f = (1/N)*fft(x_t);

% PLOTS
figure(1)
subplot(121),plot(t,x_t),grid on
         xlabel('t (s)')
subplot(122),plot(f,db(X_f),'b',f,db(X_f_w),'r'),grid on
         xlabel('f (Hz)')
```

(b) Leave N=512, and add a second version of the spectrum calculated using a time window of the type Hamming (see code lines below). Superpose this second spectrum to your first plot and comment on the effect of this windowing in the spectrum calculation.
Hint: you can better observe the effect of the windowing by zooming in the first 10 harmonics.

```
window = hamming(N)';
X_f_w = (1/N)*fft(x_t.*window);
```

(c) Open, execute and analyse the Matlab script ***sisy_fft_settings_n_effects.m***.
Identify the effects of the different settings. For example, how does the zero-padding works?