TELE303 Tutorial / Lab 1 -

Part A. Tutorial

Let's review a few important concepts and theorems we learnt in last week's lectures.

Given the bandwidth and SNR of a channel, its theoretical capacity upper limit is given by the Shannon's theorem: $C = B\log_2(1 + \text{SNR})$.

On the other hand, we know the Nyquist theorem also holds: $C \leq B\log_2 M$, where M is the level of signaling.

1. Can you tell what is the relationship between the Shannon capacity and the Nyquist capacity?

2. Suppose a Wi-Fi channel has a bandwidth of 20MHz, and $\text{SNR}_{\text{dB}} = 30\text{dB}$. What is the maximal capacity? Is this close to what we get in reality, why?

3. To achieve an intended capacity of 14 Mbps, with the SRN=127, how big a bandwidth do we need?

Part B. Lab1: Signaling & DFT

This lab involves using Python packages to conduct Fourier analysis of digital signals. This lab is assessed and is worth 2 marks. Complete all the tasks and report to the tutor.

1 Getting started with Numpy

NumPy is a fundamental package used for scientific computing and contains an efficient implementation of various utilities in linear algebra, Fourier transform, and statistics. Paired with other packages included in the larger SciPy system [1], including the plotting toolkit "Matplotlib" (pylab), it provides powerful open-source tools for mathematics, science and engineering.

Before the lab, please read through the NumPy Tutorial (ref. links at the Piazza site) to get familiar with the basic syntax of NumPy. You can use Python/IDLE to work with the following scripts.

2 Signal & Spectrum

Recall in Lecture 2 we looked at the basic form of all analog signals - sine waves. Now we can investigate, in a digital setting, the effect of three key elements for a sine wave: amplitude, frequency and phase.

2.1 A simple sine wave

Run Python2.7/IDLE. Open a New Window from IDLE, type in the following script and save it as "basic-sine.py":

```
# import necessary libraries
import numpy as np
import pylab as pl
# time t is an array, ranged from 0.0 to 1.0 second, with 0.001s increment
t = np.arange(0.0, 1.0, 0.001)
# signal definition
s = 3*np.sin(2*np.pi*2*t)
# plot the signal s against time t
l, = pl.plot(t,s, lw=2, color='red')
# Make the plot visible
pl.show()
```

Answer the following questions:

- What is the amplitude, and frequency of the sine wave?

  A= _____ , f= _____ Hz.

- What is the sampling rate? Does it satisfy Nyquist's theorem? (Tip: check the Python code.)

  _____

- **Task A**: change the code, so that it displays a composite function $s(t) = 3\sin(2\pi 2t) + 2\sin(2\pi 5t)$. Do you think it is periodic? If yes what's its frequency? _____ Hz

[1] `http://www.scipy.org`

2.2 Make it interactive

Based on 'basicsine.py', we now add some subplots and UI control components. Save the following as 'sinewave.py'. Run the script, and see how different settings change the waveform.

Answer this (trivial) question: what is the phase (in degrees) that gives a cosine wave? _____

```
import numpy as np
import pylab as pl
from matplotlib.widgets import Slider, Button, RadioButtons

# generate the first subplot - the signal
ax = pl.subplot(111)
pl.subplots_adjust(left=0.15, bottom=0.35)
t = np.arange(0.0, 1.0, 0.001)
a0 = 5
f0 = 3
p0 = 0
s = a0*np.sin(2*np.pi*f0*t)
l, = pl.plot(t,s, lw=2, color='red')
pl.axis([0, 1, -10, 10])

# add a few axes: [x, y, width, height]
axcolor = 'lightgoldenrodyellow'
axfreq = pl.axes([0.15, 0.2, 0.65, 0.03], axisbg=axcolor)
axamp  = pl.axes([0.15, 0.15, 0.65, 0.03], axisbg=axcolor)
axph = pl.axes([0.15, 0.1, 0.65, 0.03], axisbg=axcolor)
# add sliders for axes; give min, max
sfreq = Slider(axfreq, 'Freq', 0.1, 30.0, valinit=f0)
samp = Slider(axamp, 'Amp', 0.1, 10.0, valinit=a0)
sph = Slider(axph, 'Phase', -180, 180.0, valinit=p0)

# define callback for the sliders; update signal display
def update(val):
    amp = samp.val
    freq = sfreq.val
    ph = sph.val
    l.set_ydata(amp*np.sin(2*np.pi*freq*t+ph/180.0*np.pi))
    pl.draw()
sfreq.on_changed(update)
samp.on_changed(update)
sph.on_changed(update)

# get the Reset button to work
resetax = pl.axes([0.8, 0.025, 0.1, 0.04])
button = Button(resetax, 'Reset', color=axcolor, hovercolor='0.975')
def reset(event):
    sfreq.reset()
    samp.reset()
    sph.reset()
button.on_clicked(reset)

pl.show()
```

2.3 DFT

We have played a bit with waveforms, but what about the spectrum? We need to use Fourier transform, which decomposes a signal into a set of sine waves. For a digital signal the Discrete Fourier Transform extracts the frequency components on discrete intervals $f_k = 2k\pi/N$ (k=0,1,..., N-1 for a signal of length N). The series of the amplitudes over the N frequency levels makes the power spectrum of the signal. FFT is a fast algorithm for DFT.

FFT can be quite complicated, but fortunately DFT/FFT has been implemented in NumPy. Have a try with the following (save it as 'fftex.py' and run it from IDLE):

```
import numpy as np
import pylab as pl

# get the PI value
PI=np.pi
# specify sampling rate (Hz) and sampling time length (s)
samplrate=1000
nsec=1.0    # 1 second
# get the discrete sampling time sequence
#t=sp.r_[0:nsec:1.0/samplrate]
t=np.arange(0,nsec,1.0/samplrate)
# here's a signal (f=? Hz)
sig=np.sin(2*PI*10*t)
# plot out the waveform
pl.figure(1)
pl.plot(sig)
pl.title('Original sig')

# conduct FFT
fftout=np.fft.fft(sig)
# get the power spectrum
ps=np.abs(fftout)
# the rest is for visualisation
pl.figure(2)
pl.plot(ps)
pl.xlabel('n')
pl.ylabel('Amplitude')

pl.show()
```

Run the code. Answer these questions:

- What is the signal in mathematical form? What is its frequency?
- Explain the spikes observed in the spectrum plot.

2.4 Inverse Fourier transform

Let's then have a play with the **inverse** Fourier transform and see if the signal can be restored from "fftout", the outcome of the DFT. This can be done with a single line:

```
f2=np.fft.ifft(fftout)
```

Note f2 is a complex array but we are only interested in the real part, i.e. "f2.real".

Task B. Write a few more lines and have the waveform of restored signal displayed in a new figure. Have that compared with the old 'sig' shown in the 'Figure 1' window. Are the waveforms the same?

Take a break now and show your work to the tutor(s) ☺.

3 DTMF

Let us make this lab a little more interesting by looking into a real world application. Dual Tone Multi-Frequency (DTMF) is an important technique used in wireline and mobile phones. Suppose we are using a call centre service and need to key in some numbers (e.g., a menu selection, or a PIN). How does the call centre automatically make out which key/number has been pressed on your phone?

DTMF uses a simple addition of two frequency components to indicate which key on a telephone touch pad is being pressed. The frequency-digit relationship is listed below.

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

Here the column corresponds to a high frequency, and the row corresponds to the low frequency. For example, the hash key, # would generate a signal as the sum of two sine waves, one in 1477 Hz (high frequency), and another in 941 Hz (low frequency).

Task C. Modify your 'fftex.py' code and save it as 'dtmf.py'. In the code, generate the waveform for digit '3' by using two frequencies; display the power spectrum. Note you'll need to set an appropriate sampling rate.

Is it possible to tell which digit is pressed by looking at the power spectrum of the received signal?

End of Lab 1. Remember to show your work to a tutor.

STUDENT ID: _____, NAME: _____