LABORATORY MANUAL

SC3102: Signals, Systems and Transforms
Hardware Lab 1 (Location: N4-01a-03)

Experiment 1: Signals and its representations

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

LAB – 1

Signals and its representations

Objectives

The objective of this laboratory is to learn Signals and its various representations using python programming language. In this laboratory, we will

a. understand basic representation of signals using python
b. work on signals in time domain
c. understand the concepts of sampling theorem and analog to discrete-time conversion
d. learn to down-sample a real-world signal

We will be using python 3.8 using Anaconda environment [1]. You can choose to use the school's laboratory environment or your own laptop's environment during the laboratory.

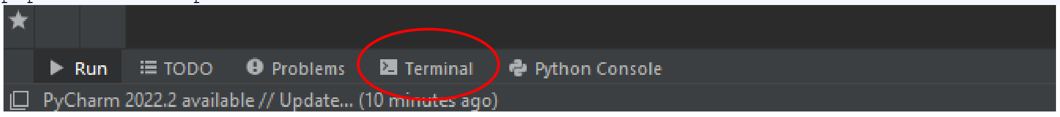
If you are using the lab pc, you may use the PyCharm IDE already installed in your pc.
Else you may install Anaconda and the following libraries using conda install
https://docs.anaconda.com/anaconda-cloud/user-guide/howto#use-packages

- Numpy: https://anaconda.org/anaconda/numpy
- Scipy: https://anaconda.org/anaconda/scipy
- Matplotlib: https://anaconda.org/anaconda/matplotlib

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal
```

If you encounter any error in importing the required packages, run the following code one by one in the PyCharm terminal.

```
pip install numpy
pip install scipy
pip install matplotlib
```



Instructions to students

The following rules and regulations apply for all the laboratories pertaining to SC3102.

Read the lab manual thoroughly before coming to the lab. You are expected to write all the python programs on their own. Refer to the sample code provided only after you have exhausted all your resources. You are allowed to open the "code" folder only after one hour of the start of the lab.

If you have questions on how to develop the code, try using the sample code and check with the TA (Teaching Assistant).

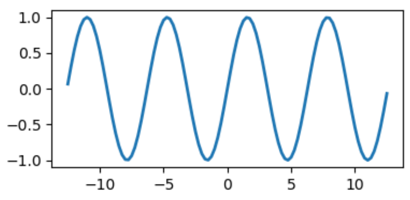
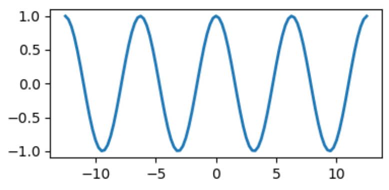
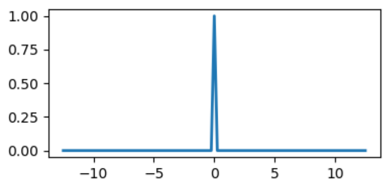
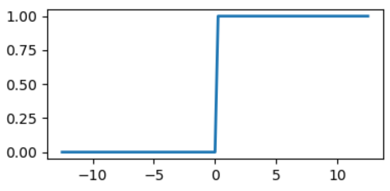
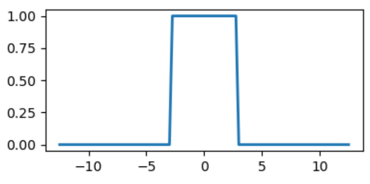
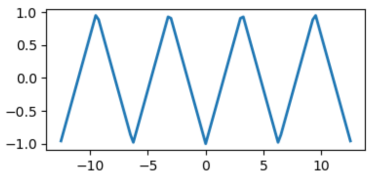
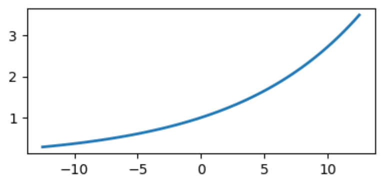
Towards the end of the lab sessions, you have to take a lab quiz to test the concepts learned in the lab and the related lectures. Marks for the quiz will be carried over to the coursework component for the evaluation of the course.

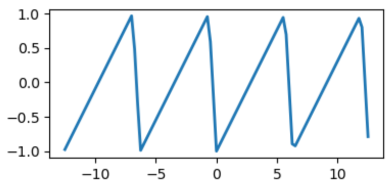
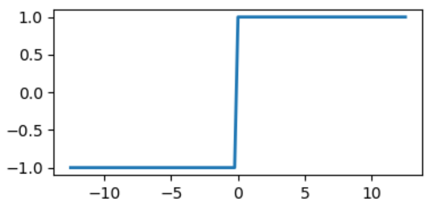
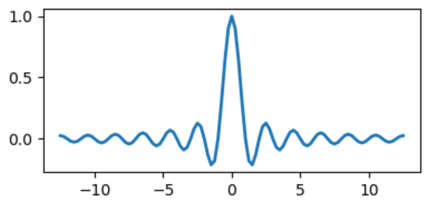
Attendance for the laboratory sessions is compulsory. Retakes for the labs are not allowed without valid reasons. If you are absent for the laboratory session and subsequent lab quiz, you will receive 0 marks for the respective lab session.

1. Basic representation of signals

In this exercise we will learn to write a python program to generate the following discrete time signals. Sample code is given below for your reference along with a sample output in figure 1.1. There are different ways to plot a signal. The sample code given is just one way. You may plot the signals using other resources as well.

Do note that some of the plots generated in this exercise will be needed for the subsequent labs.

SL No	Signal	Python sample code	Figure
1	Sine wave $\sin(2\pi f\ x)$	numpy.sin(x)	
2	Cosine wave $\cos(2\pi f\ x)$	numpy.cos(x)	
3	Unit impulse $\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$	signal.unit_impulse()	
4	Unit Step wave $u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$	x > 0	
5	Square wave $u[n] =$ $\begin{cases} 1, & -k < n < k \\ 0, & \text{otherwise} \end{cases}$	(x > -3)*(x < 3)	
6	Triangular wave $x[n] =$ $\begin{cases} n, & 0 < n < k \\ -n + 2k, & k < n < 3k \\ n - 2k, & 3k < n < 4k \end{cases}$	signal.sawtooth(x,0.5)	
7	Exponential wave e^x	np.exp()	

8	Sawtooth wave $x(n) = n - \lfloor n \rfloor$	signal.sawtooth()	
9	Signum function $x[n] = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$		
10	Sinc wave $\mathrm{sinc(x)} = \dfrac{\sin(x)}{x}$	np.sinc(x)	

Sample code

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import signal

x = np.linspace(-12.5, 12.5, 101)
plt.figure()


#_Fill_in_your_code_here
y1 =      # Sine wave
y2 =      # Cosine wave
y3 =      # Unit Impulse wave
y4 =      # Unit Step wave
y5 =      # Unit square wave
y6 =      # Triangular wave
y7 =      # Exponential wave
y8 =      # Sawtooth wave
y9 =      # Sign function
y10 =     # Sinc wave

wave_name = ['Sine wave', 'Cosine wave', 'Unit Impulse wave', 'Unit step wave','Square
wave','Triangular wave','Exponential wave','Sawtooth wave','Sign function','Sinc
fucntion']

y = [y1,y2,y3,y4,y5,y6,y7,y8,y9,y10]

plt.figure(figsize=(10,10))

for i in range(10):
  plt.subplot(5,2,i+1)
  plt.plot(x, y[i])
  plt.title(wave_name[i])
  plt.ylabel('Amplitude')
  plt.xlabel('Time')

plt.tight_layout()
plt.show()
```

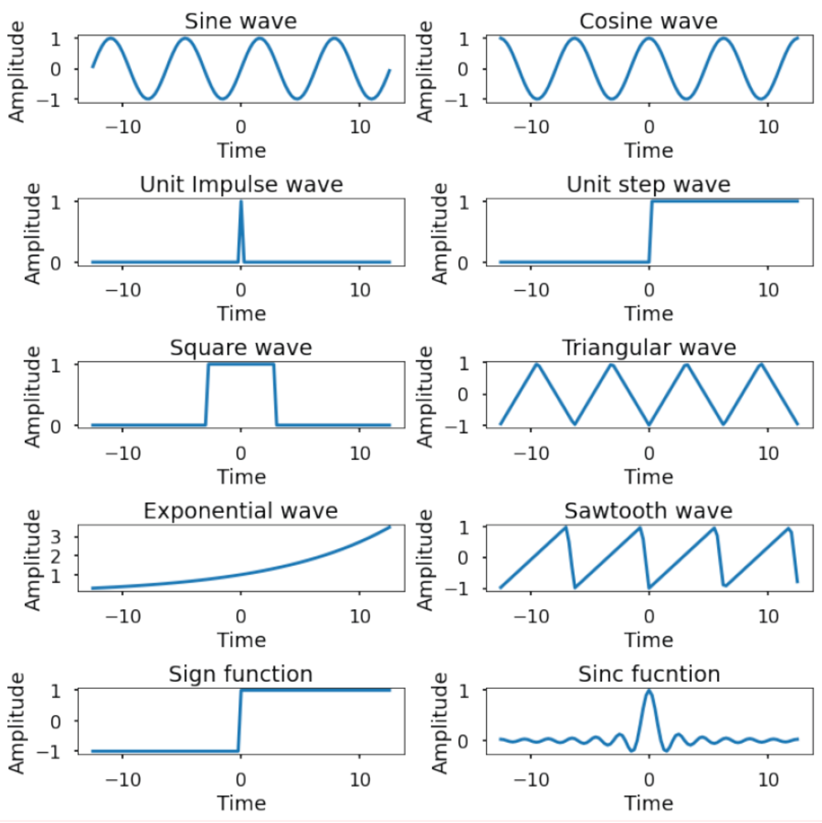


Figure 1.1: Sample output of Q1

If you are unable to complete the question on time, modify the file "Lab_1_Q1_example.py" in the Q1 folder to plot all the sequences.

2. Time domain analysis

In this exercise we will work on signals in the time domain. This requires measuring a signal at a constant interval over time. The frequency with which we measure a signal is referred to as the sampling frequency. The units of this are typically described in Hz or the number of cycles per second. It is critical that the sampling frequency is consistent over the entire measurement of the time series.

2.1. Sampling theorem

In this subsection we verify the sampling theorem. The definition of proper sampling is quite simple. Suppose we sample a continuous time signal, say a sine wave, in some manner. If we can exactly reconstruct the signal from its samples, then we have done sampling properly.

Generate a sine wave $$y(t) = A\,sin(2\pi * 100t + \pi/6)$$

Shannon sampling theorem states that for actual reconstruction of the signal, an analog signal has to be sampled at least twice the maximum signal frequency. It is also called the Nyquist criteria for sampling. The minimum sampling frequency required for the actual reconstruction of the signal is also called the Nyquist rate. Through this experiment, we are learning the relation of different sampling frequencies for the actual reconstruction of the signal. So, for the above expression, what is the minimum sampling frequency required for the actual reconstruction of the signal? What happens if we sample the signal below the minimum sampling frequency?

Write a python program to demonstrate your reasoning. The program should plot the sine wave y(t) given above sampled at different sampling frequencies. You should test the signal with the following frequency criteria and repeat the exercise with different frequency values.

a. Sampling frequency below Nyquist criteria
b. Sampling frequency matching with Nyquist rate
c. Sampling frequency above the Nyquist rate

Sample code is given below along with sample output. Complete the sequence to verify the sampling theorem.

Hint: Try plotting the sequence with *plt.plot(x,y)* as well as *plt.stem(x,y)*. Stem plots will give sampling instances more clearly.

```
Fm=100 #Frequency of sime wave
t1 = np.linspace(0,0.1,200)

y1 = np.sin(2 * np.pi * Fm * t1 + np.pi/6)  #Sine wave

fig, axs = plt.subplots(1, 2,figsize=(10, 4),constrained_layout=True)
axs[0].plot(t1,y1,label = 'continuous')
axs[0].set_title("Sine wave")
axs[0].set_xlabel('time in sec')
axs[0].set_ylabel('y(t)')

fs1=int(input("Enter the sampling frequency : ")) #sampling rate
n1 = np.arange(0,0.1*fs1)/fs1

#Fill_in_your_code_here
```

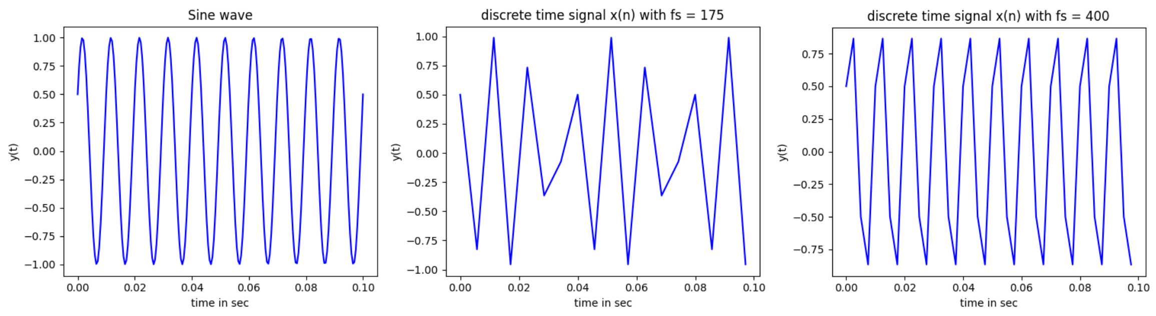


Figure 2.1: Sample output of Q2.1

If you are unable to complete the question on time, modify the file “Lab_1_Q2_sampling.py” in the Q2 folder to plot all the sequences.

2.2. Addition of two discrete time signals

In this exercise we will learn to represent signals in its discrete form. Given three signals $y1(t) = A\,cos(2\pi * 10t)$, $y2(t) = B\,cos(2\pi * 15t)$, and $y3(t) = y1(t) + y2(t)$ sampled at $Fs = 60$. Sample code is given below to generate the signals. Complete the code to plot the signals. Using the plots as a reference, answer the following questions. You may change the code as required.

a. Find the discrete time representation of y1[n], y2[n], and y3[n].
b. Find the period of y3[n].
c. How are the periods of y3[n] related to y1[1] and y2[n]?
d. Is y3[n] an energy or power signal? Find its (energy or power) and compare it against (energy or power) of y1[n] and y2[n].

```
# define gcd function
def gcd(x, y):
   """This function implements the Euclidian algorithm to find G.C.D. of two
numbers"""
   while(y):
       x, y = y, x % y
   return x

# define lcm function
```

```
def lcm(x, y):
    """This function takes two integers and returns the L.C.M."""
    lcm = (x*y)//gcd(x,y)
    return lcm

# define the main function
def main():
    A=0.5; F1=10; Phi = 0; Fs=60; sTime=0; eTime = 1;
    t1 = np.arange(sTime,eTime,1.0/Fs)
    y1 = A*np.cos(2 * np.pi * F1 * t1 + Phi)

    B=0.3; F2=15;
    t2 = np.arange(sTime,eTime,1.0/Fs)
    y2 = B*np.cos(2 * np.pi * F2 * t2 + Phi)

    y3 = #Fill_your_code_here

    #how many samples in one cycle
    nSamplesPeriod1 = int(Fs/F1)
    nSamplesPeriod2 = int(Fs/F2)
    nSamplesPeriod_LCM = lcm( int(nSamplesPeriod1),int(nSamplesPeriod2))
    s = 'Number of samples per cycle N1='+ repr(nSamplesPeriod1)+'  N2='+
repr(nSamplesPeriod2) + '  LCM = '+ repr(nSamplesPeriod_LCM)
    print(s)
    nSamplesPeriod_int = int(nSamplesPeriod_LCM)

    J = 3   # lets plot J cycles

# Fill in your code here to plot the figures


if __name__ == '__main__':
     main()
```

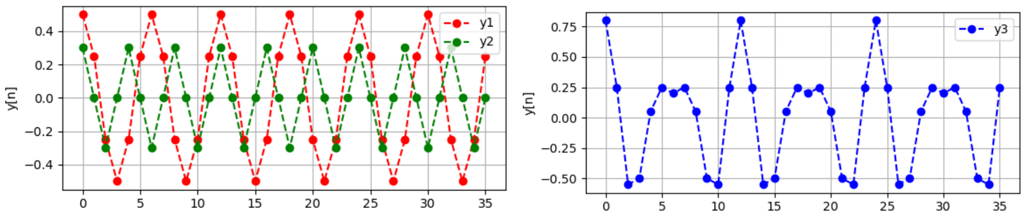


Figure 2.2: Sample output of Q2.2

The python program is given in the file “Lab_1_Q2_addition.py” file in the Q2 folder to plot the signals y1, y2 and y3.

3. Working on a Natural signal

In this subsection we learn to plot a natural real-world signal. One such signal is the electroencephalogram (EEG) signals which is the recording of brain activity. During this painless test, small sensors are attached to the scalp of a subject to pick up the electrical signals produced by the brain. A small portion of the recorded signal is provided in the file “EEG_exp.mat” in the Q3 folder. These signals were recorded by a brain computer interface (BCI) machine.

Copy the “EEG_exp.mat” from NTU learn to the project directory.

3.1. Plotting an EEG Signal

Use the sample code provided to plot the EEG signals. The sample output is given in figure 3.1. Take note to plot the figure labels correctly. You can limit the number of samples, say the first 1000 samples, to get a zoomed in clear image of the signal.

```
from scipy import io # this submodule let's us load the signal we want
EEG_data = io.loadmat('EEG_exp.mat', squeeze_me = True)
# print all the variables that exist in the dictionary
print(EEG_data.keys())
```

```
# this contains the EEG data
EEG = EEG_data['EEG']
# this contains the sampling rate, in Hz (or samples/second)
fs = EEG_data['fs']

# Plot the EEG signal, fill your code here
# Plot the first 1000 samples, we will use this in the subsequent questions
```

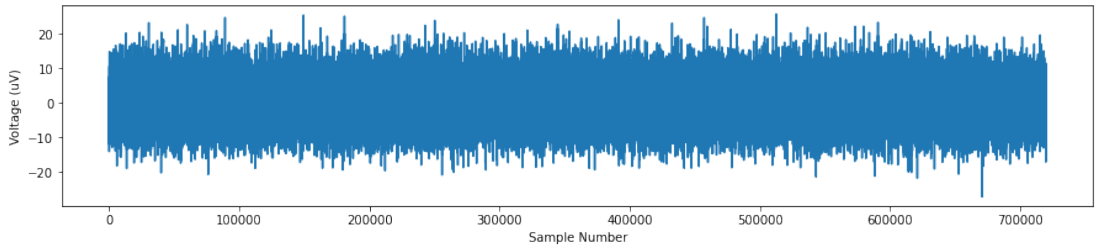


Figure 3.1: EEG signals loaded form "EEG_exp.mat" file.

To load these signals for further analysis, run the "Lab_1_Q3_loadEEG.py" file.

3.2. Analog to Discrete time conversion

People used to analyze signals using analog circuits. This is hard and requires extensive firsthand knowledge about circuitry. Once you want to analyze the signal on a "digital" computer, you must "digitize" the signal. To digitize an analog signal, you must discretely sample, both in value (voltage, brightness, etc.) and in time. It is like drawing a grid over your continuous signals and interpolating its values only at where the grid crosses.

The signal is already a digitally sampled (720,000 samples) time series represented by discrete points. This means that the signal is already stored as a digital signal since all computers are digital. But we assume this as a continuous signal and digitize it using a 4-bit ADC, means it will have 16 levels. A sample code for doing this is given below. Modify this and run it after loading the EEG files.
Hint: To find delta_v, think of the number of voltage levels and the voltage resolution. In other words, what is the finest voltage that can distinguish two samples

```
min_v, max_v = -32,32
delta_v = #voltage resolution

# create the digitization vector with new possible values that your signal can take
ADC_levels = np.arange(min_v,max_v,delta_v)+delta_v/2

# digitize the EEG signal with our ADC with the function np.digitize
# note that we have to scale the redigitized signal to its original units
EEG_quant = np.digitize(EEG,bins=ADC_levels)*delta_v+min_v

# Plot the EEG signal and the digitized EEG signal on top of it
#_Fill_in_your_code_here
```

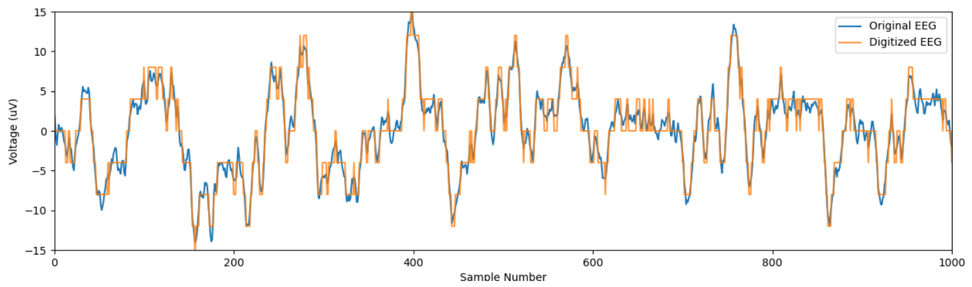


Figure 3.2: Sample output for ADC

Change the values of "delta_v" and draw conclusions. What happens when "delta_v" is increased?

What are the number of possible voltage levels of the new digitized EEG signal? How is this related to “delta_v”?
Find the minimum number of bits that can approximately represent the signal. Justify (open ended question)

A sample code for this exercise is given in Q3 folder in the file “Lab_1_Q3_ADC.py”.

3.3. Down-sampling in time

The x-axis in the section 3.1 plot of EEG signal denotes the “sample number”, which corresponds to the position of each value in the EEG data array. In real life, the x-axis usually represents the time vector which marks the clock time at which the data is recorded. Sometimes the EEG datasets include a time vector, which is not the case here. So, in this exercise, we create the corresponding time vector.

A sample code for creating the time vector for the given EEG signal is given below. What is sampling period? Identify sampling rate from the Q3.1, that is from the EEG data. Use this to find the sampling period “dt”.

What is the total time duration of the given EEG signal?

What is the total number of samples of the EEG signal?

Use the sampling rate information and total number of samples to calculate the total duration of the signal. Let this value be “T_exp” in the code signifying the duration of the experiment.

Create a time vector corresponding to the EEG data with evenly spaced samples with an interval “dt”. Let this vector be “t_EEG” in the code. Hint: check the function np.arange()

```
np.arange()
```

Plot the EEG signal as a line chart with x-axis as the time (using “t_EEG”). Assume the EEG signal start at 0. Remember to label the plot with appropriate units.

To simulate down-sampling, plot every 10th value of the EEG signal. This applies to both the time vector and the EEG data. This can be done by indexing the array properly. Read on how to index an array in python. The provided sample code has the solution on how the array can be indexed at every 10th sample. Try indexing for different values and see how down sampling changes the EEG signal. Repeat the exercise with different down-sampling rate.

```
dt =                # _FILL_IN_YOUR_CODE_HERE    (sampling period)
T_exp =             # _FILL_IN_YOUR_CODE_HERE    (total duration of EEG signal)
t_EEG =             # _FILL_IN_YOUR_CODE_HERE    (EEG time vector)
d_rate = 10         # Downsampling rate
# Plotting the signal and its downsampled version
plt.figure(figsize=(15,3))

plt.plot(#_FILL_IN_YOUR_CODE_HERE)
plt.plot(t_EEG[1:1000:d_rate], #Fill , '.-', label='Downsampled', alpha=0.8)
plt.xlim([0,1]); plt.ylim([-15, 15]);
plt.legend()

# _FILL_IN_YOUR_CODE_HERE for labelling the plot
```

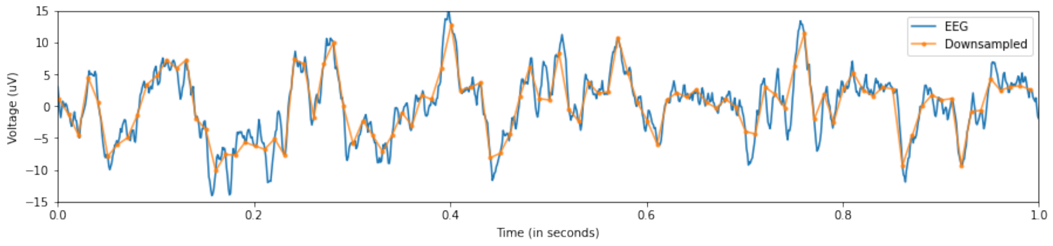


Figure 3.3: Sample output for down-sampling

The code for creating the time vector for the given EEG signal is given in “Lab_1_Q3_sampling.py”

4. Polar plot of signals (Optional)

Write a python program to generate and plot the following discrete time signal

$$y[n] = A^n e^{j(\omega n+\phi)}$$

where A is a real number, digital frequency ω (in radian/sample), phase shift ϕ (radian), for a given range $n = 0...N$, and n, $N \in Z$. Sample code is given below. You may edit the code as required. Experiment with different values of Phi, ω and A. Visualize the complex discrete sequence $y[n]$ in 3 ways, e.g., A=0.95, $\omega = 2\pi/36$.

```
# plotting 3D complex plane
import numpy as np
from scipy import signal
import matplotlib.pyplot as plt


def main():

#Fill in your code here to generate the signals

          A= #Fill the value of A
          w1=2*np.pi/18;
          Phi = #Fill
          numSamples = #Fill

    n = np.arange(0 , #Fill , 1 )
    y1 = np.multiply(np.power(#Fill), np.exp(#Fill)))

# plot the signals in 2D, Q4.1

if __name__ == '__main__':
    main()
```

2-D plot of real and imaginary values in the same figure using distinct colors.

```
#plotting in 2-d, both the real and imag values
    plt.figure(1)
    plt.plot(n, y1[0:numSamples].real,'r--o')
    plt.plot(n, y1[0:numSamples].imag,'g--o')
    plt.xlabel(#Fill); plt.ylabel(#Fill)
    plt.grid()
```

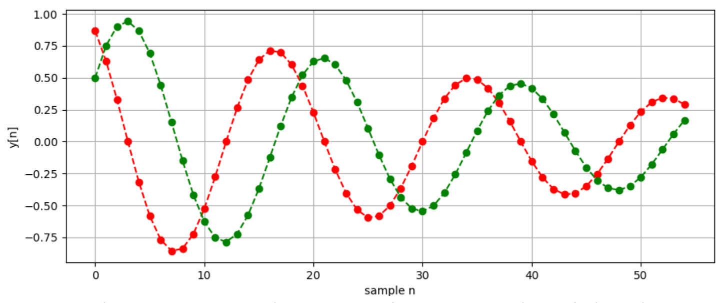


Figure 4.1: Sample output of Q4.1, 2D plot of signal

4.1. Polar plots of the sequence.

```
#plotting in polar
    plt.figure(2)
    for x in y1:
        plt.polar([0,np.angle(#Fill)],[0,np.abs(#Fill)],marker='o')
```

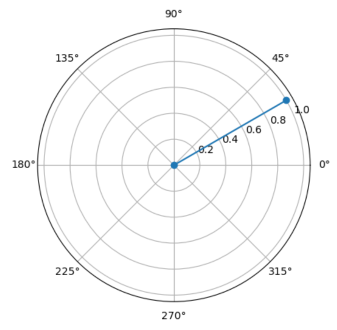


Figure 4.2: Sample output of Q4.2, polar plot of signal

4.2. 3-D plot showing trajectory with respect to sample index.

```
# plotting 3D complex plane
    plt.rcParams['legend.fontsize'] = 10
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    reVal = y1[0:numSamples].real
    imgVal = y1[0:numSamples].imag
    ax.plot(n,reVal, imgVal,  label='complex exponential phasor')
    ax.scatter(n,reVal,imgVal, c='r', marker='o')
    ax.set_xlabel(#Fill)
    ax.set_ylabel(#Fill)
    ax.set_zlabel(#Fill)
    ax.legend()
    plt.show()
```

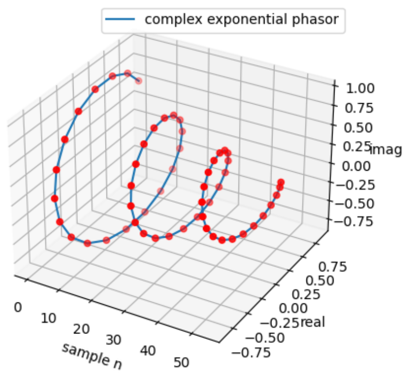


Figure 4.3: Sample output of Q4.3, 3D plot of signal

Redo Q3.4 with = $2\pi/18$, comment on the difference in the 3 plots.

Now run the file "Lab_1_Q4_3D_plot.py" for the plotting the function

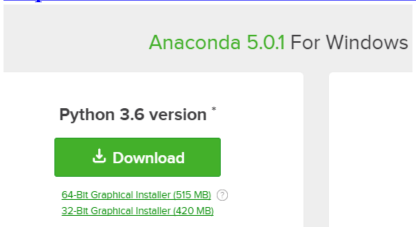
$$W^k = 1e^{j(\frac{2\pi}{N}kn)} \text{ and } k \in Z$$

Generate the 3-D plots of W^k for N = 16, n = 0...N-1, plot W^0, W^1, W^2, W^3. Comment on their relationship with respect to k and *number of cycles* and angular frequency.

The python program for exercise 4.5 can be found in "Lab_1_Q4_polar_plot.py" file in the Q4 folder.

5. References

[1] Anaconda Environment for Python (Python 3.6)
https://www.anaconda.com/download/


Anaconda 5.0.1 For Windows
Python 3.6 version *
Download
64-Bit Graphical Installer (515 MB)
32-Bit Graphical Installer (420 MB)