**Communication Assignment 2**

| Name | Sec | BN | ID |
| --- | --- | --- | --- |
| Asmaa Adel Abdelhamed | 1 | 14 | 9202285 |
| Samaa Hazem Mohamed | 1 | 32 | 9202660 |

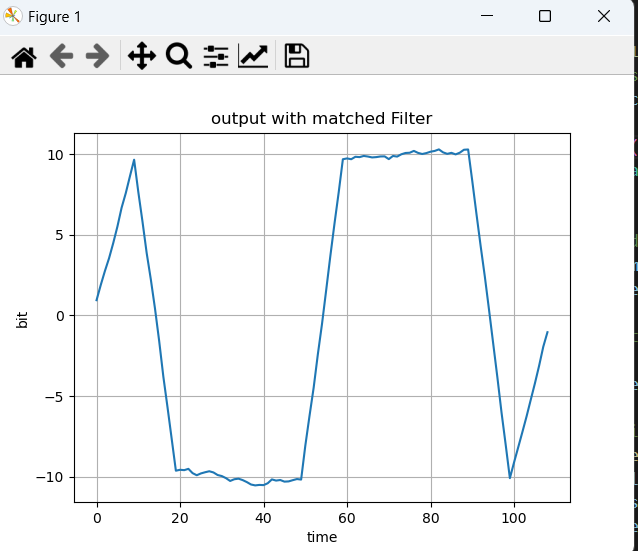
**For theoretical solutions for question 1 and question 2, you will find it at the end of the report**

**Question 2:**

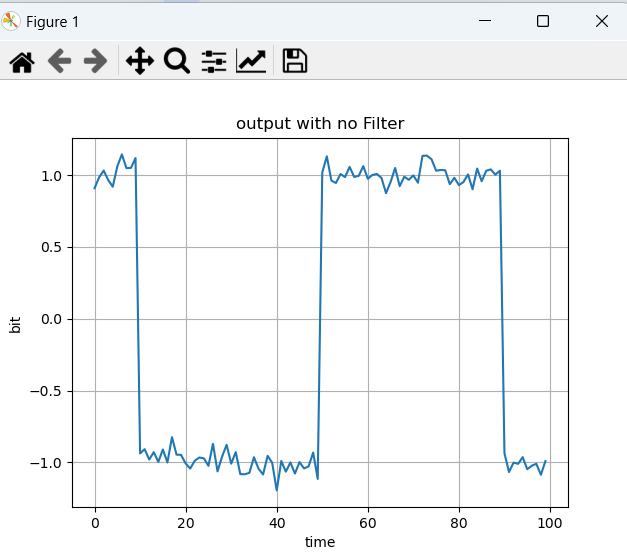
* **Plots for the outputs for each filter:**

To make it visible, we made the number of bits = 10, and the number of samples =10

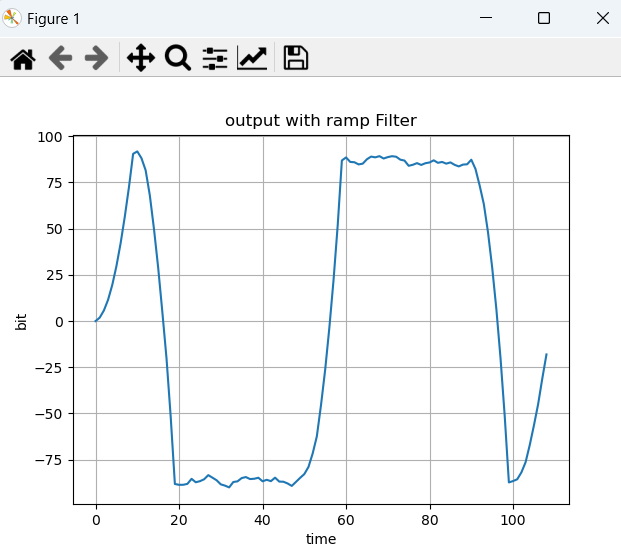
**Filter 1 output:**

****

**Filter 2 output:**

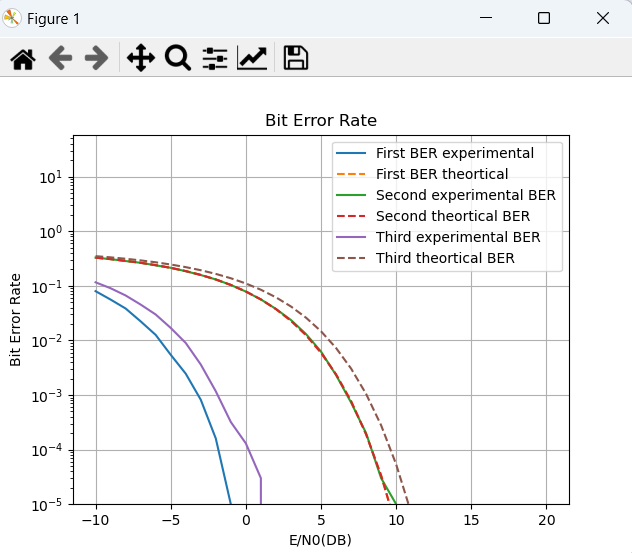
****

**Filter 3 output:**

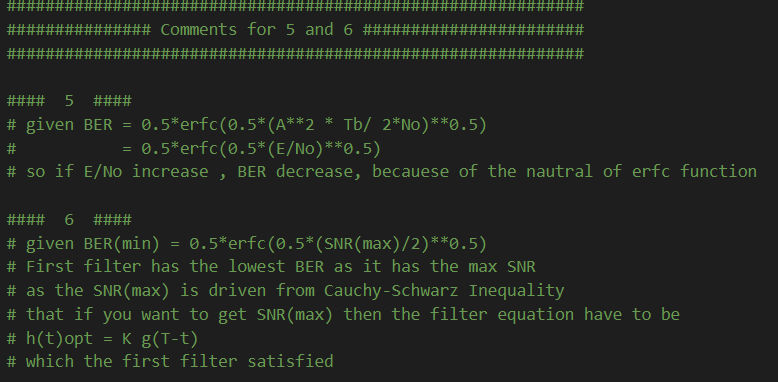
****

* **The plot for theoretical and simulated Bit Error Rate (BER) Vs E/𝑵o:**

To make it visible, we made the number of bits = 10^5, and the number of samples =10

****

**Comments for req 5 & req 6:**

****

**The code:**

**############################################################**

**############### Comments for 5 and 6 #######################**

**############################################################**

**#### 5 ####**

**# given BER = 0.5\*erfc(0.5\*(A\*\*2 \* Tb/ 2\*No)\*\*0.5)**

**# = 0.5\*erfc(0.5\*(E/No)\*\*0.5)**

**# so if E/No increase , BER decrease, becauese of the nautral of erfc function**

**#### 6 ####**

**# given BER(min) = 0.5\*erfc(0.5\*(SNR(max)/2)\*\*0.5)**

**# First filter has the lowest BER as it has the max SNR**

**# as the SNR(max) is driven from Cauchy-Schwarz Inequality**

**# that if you want to get SNR(max) then the filter equation have to be**

**# h(t)opt = K g(T-t)**

**# which the first filter satisfied**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import math**

**############################################################**

**################## Convolution Function ####################**

**############################################################**

**def applyConvolution(noisySamples, receivedFilterValues):**

**convolutionResultSampledTp = np.zeros(numOfBits)**

**if (receivedFilterValues is not None):**

**convolutionResult = np.convolve(**

**noisySamples.flatten(), receivedFilterValues)**

**else:**

**convolutionResult = noisySamples.flatten()**

**for i in range(numOfBits):**

**convolutionResultSampledTp[i] = convolutionResult[(numOfSamplesPerBit - 1) + numOfSamplesPerBit \* i]**

**return convolutionResult, convolutionResultSampledTp**

**############################################################**

**######################## BER Simulated #####################**

**############################################################**

**def calculateBERSimulated(bits, recievedBits):**

**receivedSamples = np.ones(numOfBits)**

**receivedSamples = np.sign(recievedBits)**

**# calculate probability of error**

**error\_probability = np.sum(receivedSamples != bits)**

**error\_probability /= numOfBits**

**return error\_probability**

**############################################################**

**######################## main function #####################**

**############################################################**

**def func(samples,receivedFilterMatched, type, plot):**

**experimentalBER = []**

**theorticalBER = []**

**for EOverN0\_db in range(-10, 21):**

**EOverN0\_dec = 10 \*\* (EOverN0\_db / 10)**

**# generating random noise**

**generatedNoise = np.random.normal(**

**# we want sigma of the noise t0 equal np.sqrt(No/2)**

**# 0, E/(2\*EOverN0\_dec), numOfBits\*numOfSamplesPerBit).reshape((numOfBits, numOfSamplesPerBit))**

**0, np.sqrt(1/(2\*EOverN0\_dec)), numOfBits\*numOfSamplesPerBit).reshape((numOfBits, numOfSamplesPerBit))**

**# add noise to samples**

**noisySamples = samples + generatedNoise**

**# apply convolution for the noisy samples**

**filteredSamples, recievedBits = applyConvolution(**

**noisySamples, receivedFilterMatched)**

**# append BER simulated for plotting**

**experimentalBER.append(calculateBERSimulated(bits, recievedBits))**

**# append BER theortical for plotting**

**if type == 3:**

**theorticalBER.append(0.5\*math.erfc((3\*\*0.5/2) \* (EOverN0\_dec \*\* 0.5)))**

**else:**

**theorticalBER.append(0.5\*math.erfc(EOverN0\_dec \*\* 0.5))**

**if plot == 1:**

**# ploting**

**plt.figure()**

**plt.plot(range(0, filteredSamples.flatten(**

**).shape[0]), filteredSamples.flatten(), label="bit value")**

**plt.xlabel('time')**

**plt.ylabel('bit')**

**if type == 1:**

**plt.title('output with matched Filter')**

**elif type == 2:**

**plt.title('output with no Filter')**

**else:**

**plt.title('output with ramp Filter')**

**plt.grid()**

**plt.show()**

**return experimentalBER, theorticalBER**

**############################################################**

**############### Generating bits and samples ################**

**############################################################**

**#constants**

**numOfBits = 10**

**numOfSamplesPerBit = 10**

**# generate random bits with equal probability**

**# bits = np.asarray([(random.randint(0, 1)\*2 - 1) for i in range(numOfBits)])**

**bits = np.random.choice([-1, 1], size=(numOfBits,), p=[1./2, 1./2])**

**# generate samples in range [numOfBits, numOfSamplesPerBit]**

**samples = (np.asarray([[bits[i] for i in range(numOfBits)]**

**for \_ in range(numOfSamplesPerBit)])).T**

**# receive with matched filter**

**filter\_1 = np.ones(numOfSamplesPerBit)**

**experimentalBER\_1, theorticalBER\_1 = func(samples,filter\_1, 1, 1)**

**# receive with no filter**

**filter\_2 = None**

**experimentalBER\_2, theorticalBER\_2 = func(samples,filter\_2, 2, 1)**

**# receive with ramp filter**

**filter\_3 = np.linspace(0, 10, numOfSamplesPerBit)**

**for i in range(len(filter\_3)):**

**filter\_3[i] = np.sqrt(3) \* filter\_3[i]**

**experimentalBER\_3, theorticalBER\_3 = func(samples,filter\_3, 3, 1)**

**########################################################################################################################**

**############################################################**

**############### Generating bits and samples ################**

**############################################################**

**# constants**

**numOfBits = 100000**

**numOfSamplesPerBit = 10**

**# generate random bits with equal probability**

**bits = np.random.choice([-1, 1], size=(numOfBits,), p=[1./2, 1./2])**

**# generate samples in range [numOfBits, numOfSamplesPerBit]**

**samples = (np.asarray([[bits[i] for i in range(numOfBits)]**

**for \_ in range(numOfSamplesPerBit)])).T**

**# receive with matched filter**

**filter\_1 = np.ones(numOfSamplesPerBit)**

**experimentalBER\_1, theorticalBER\_1 = func(samples,filter\_1, 1, 0)**

**# receive with no filter**

**filter\_2 = None**

**experimentalBER\_2, theorticalBER\_2 = func(samples,filter\_2, 2, 0)**

**# receive with ramp filter**

**filter\_3 = np.linspace(0, 10, numOfSamplesPerBit)**

**for i in range(len(filter\_3)):**

**filter\_3[i] = np.sqrt(3) \* filter\_3[i]**

**experimentalBER\_3, theorticalBER\_3 = func(samples,filter\_3, 3, 0)**

**#ploting**

**plt.figure()**

**plt.plot(range(-10, 21), experimentalBER\_1, label = "First BER experimental")**

**plt.plot(range(-10, 21), theorticalBER\_1, "--", label = "First BER theortical")**

**plt.plot(range(-10, 21), experimentalBER\_2, label = "Second experimental BER")**

**plt.plot(range(-10, 21), theorticalBER\_2, "--", label = "Second theortical BER")**

**plt.plot(range(-10, 21), experimentalBER\_3, label = "Third experimental BER")**

**plt.plot(range(-10, 21), theorticalBER\_3, "--", label = "Third theortical BER")**

**plt.xlabel('E/N0(DB)')**

**plt.ylabel('Bit Error Rate')**

**plt.yscale('log')**

**plt.ylim(10\*\*(-5))**

**plt.title('Bit Error Rate')**

**plt.legend()**

**plt.grid()**

**plt.show()**