Федеральное агентство связи

Сибирский Государственный Университет Телекоммуникаций и Информатики

**Кафедра ТС и ВС**

**Расчетно-графическая работа**

**По дисциплине: Основы систем мобильной связи**

**Выполнил**: Коротаев Данила Евгеньевич

**Группа**: ИА-131

Новосибирск, 2023 г

**Задание на расчетно-графическую работу**

**1. Цель работы**

Закрепить и структурировать знания, полученные в рамках изучения дисциплины «Основы систем мобильной связи».

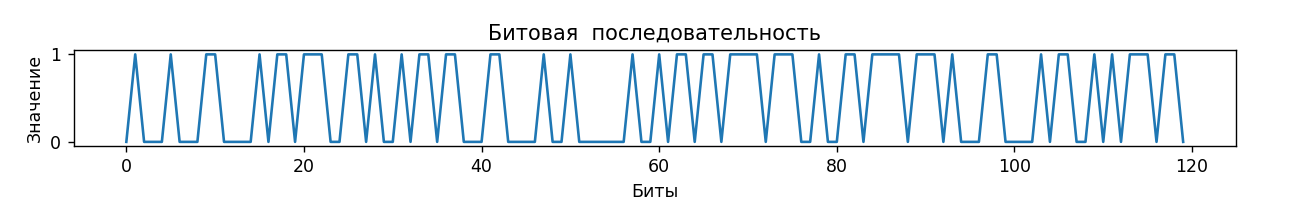
**2. Задание и порядок выполнения расчетно-графической работы**

1) Введите с клавиатуры ваши имя и фамилию латиницей.

first\_name = input("Введите имя: ")

last\_name = input("Введите фамилию: ")

2) Сформируйте битовую последовательность, состоящую из L битов, кодирующих ваши имя и фамилию латинице ASCII-символов. Результат: массив нулей и единиц с данными и разработанный ASCII-кодер. Визуализируйте последовательность на графике.



**Danila Korotaev**

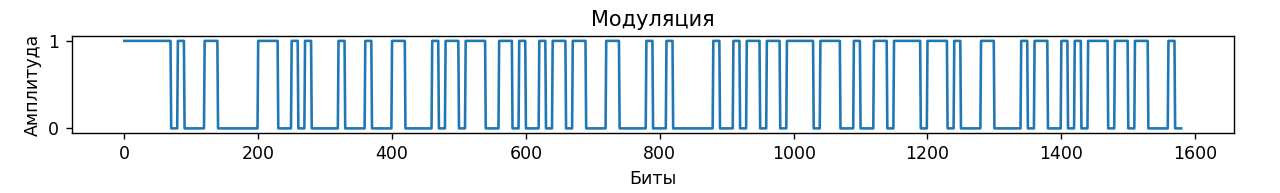
3) Вычислите CRC длиной M бит для данной последовательности, используя входные данные для своего варианта из работы №5 и добавьте к битовой последовательности. Результат: CRC-генератор и выведенный в терминал CRC.

**Битовая последовательность: [0 1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 1 0]**

**CRC: [1 1 0 0 0 1 0]**

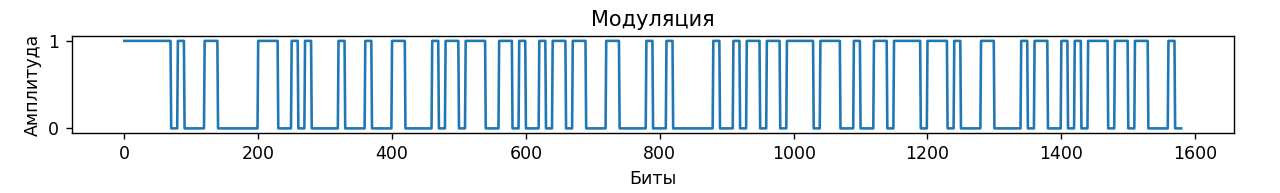
4) Для того, чтобы приемник смог корректно принимать такой сигнал и находить моменты начала, нужно реализовать синхронизацию. Для этого перед отправкой полученной последовательности добавьте последовательность Голда, которую вы реализовывали в работе №4, длиной G-бит. Результат: функция генерации последовательности Голда и массив с битами данных, CRC и синхронизации. Визуализируйте последовательность на графике.

Nx = list(posl\_Gold) + list(bit\_p) + list(crc)



5) Преобразуйте биты с данными во временные отсчеты сигналов, так чтобы на каждый бит приходилось N-отсчетов. Результат: массив длиной Nx(L+M+G) нулей и единиц – но это уже временные отсчеты сигнала (пример амплитудной модуляции). Визуализируйте последовательность на графике.

Nx = list(posl\_Gold) + list(bit\_p) + list(crc)



6) Создайте нулевой массив длиной 2хNx(L+M+G). Введите с клавиатуры число от 0 до Nx(L+M+G) и в соответствие с введенным значением вставьте в него массив значений из п.5. Результат – массив Signal – визуализируйте на графике.

Nx\_2x = [0] \* (len(Nx) \* 2)

q = input("Введите значение 0 - {}: ".format(len(Nx)))

q = int(q)

Nx\_2x[q : q + len(Nx)] = Nx

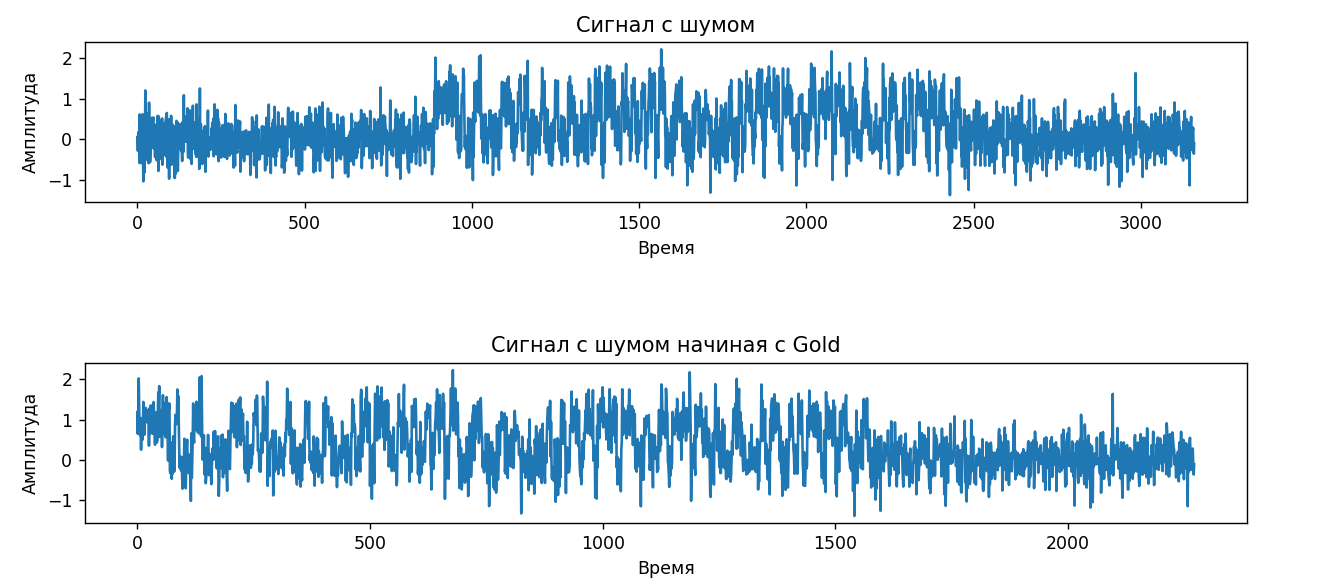


7) Предположим, что сформированная выше последовательность, промодулировала высокочастотное несущее колебание, передалась через радиоканал и на приемной стороне была оцифрована с заданной частотой дискретизации fs (число отсчетов сигнала в 1 секунде). Проходя через канал отсчеты сигнала исказились (опустим пока историю с затуханием и изменением амплитуды) – к ним добавились значения шумов, присутствовавших в канале, которые можно получить, используя нормальный закон распределения с μ=0 и σ – вводится с клавиатуры (float).

То есть нужно сформировать массив с шумом размером 2хNx(L+M+G), реализовав его с помощью нормального распределения, например,



Затем нужно поэлементно сложить информационный сигнал с полученным шумом. Визуализировать массив отсчетов зашумленного принятого сигнала.



8) Реализуйте функцию корреляционного приема и определите, начиная с какого отсчета (семпла) начинается синхросигнал в полученном массиве, удалите лишние биты до этого массива, выведите значение в терминал. Результат: функция корреляционного приемника.

def cor\_priem(arr, gold):

    gold\_x = np.repeat(gold, 10)

    q = np.correlate(arr, gold\_x)

    w = np.argmax(q)

    return arr[w:]

signal = cor\_priem(Nx\_2x\_n, posl\_Gold)

9) Зная длительность в отсчетах N каждого символа, разберите оставшиеся символы. Накапливайте по N отсчетов и сравнивайте их с пороговым значением P (подумайте, какое значение порога следует выбрать, чтобы интерпретировать полученные семплы нулями или единицами). Напишите функцию, которая будет принимать решение по каждым N отсчетам – 0 передавался или 1, на выходе которой должно быть (L+M+G) битов данных. Лишние отсчеты можно отбросить.

def decod(arr):

    d\_bit = []

    for i in range(0, len(arr), 10):

        average = int(sum(arr[i : i + 10])) / 10

        if average > 0.5:

            d\_bit.append(1)

        else:

            d\_bit.append(0)

    return d\_bit

10) Удалите из полученного массива G-бит последовательности синхронизации.

r = r[len(posl\_Gold) :]

11) Проверьте корректность приема бит, посчитав CRC. Выведите в терминал информацию о факте наличия или отсутствия ошибки.

def crc\_proverka(massiv):

    G = [1, 0, 1, 1, 1, 1, 0, 1]

    for i in range(len(massiv) - 8):

        if massiv[i] == 1:  # пропуск если 0

            for j in range(8):

                massiv[i + j] ^= G[j]  # xor

    return massiv[-7:]  # оставляем последнии 7 бит

crc = crc\_proverka(r.copy())

if all(x == 0 for x in crc):

    print("Удачно")

else:

    print("Неудачно")

12) Если ошибок в данных нет, то удалит биты CRC и оставшиеся данные подайте на ASCII-декодер, чтобы восстановить посимвольно текст. Выведите результат на экран.

**Удачно**

**[0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0]**

13) Визуализируйте спектр передаваемого и принимаемого (зашумленного) сигналов. Измените длительность символа, уменьшите ее в два раза и увеличьте тоже вдвое. Выведите на одном графике спектры всех трех сигналов (с короткими, средними и длинными символами).

fft1 = abs(np.fft.fftshift(np.fft.fft(Nx\_2x))) / len(Nx\_2x) + 0.1

fft2 = abs(np.fft.fftshift(np.fft.fft(Nx\_2x\_n))) / len(Nx\_2x\_n)

x = np.arange(-len(fft1) / 2, len(fft1) / 2)

# Разный repeat

fft05 = np.repeat(Nx13, 5)

fft1 = np.repeat(Nx13, 10)

fft2 = np.repeat(Nx13, 20)

# делаем длинну одинаковой

fft05 = list(fft05) + list(fft05)

fft2 = fft2[: len(fft1)]

x = np.arange(-len(fft1) / 2, len(fft1) / 2)

# делим для нормирования

fft05 = np.fft.fft(fft05) / len(fft05)

fft1 = np.fft.fft(fft1) / len(fft1)

fft2 = np.fft.fft(fft2) / len(fft2)

# Зануление первого элемента(для красоты)

fft05[0] = 0

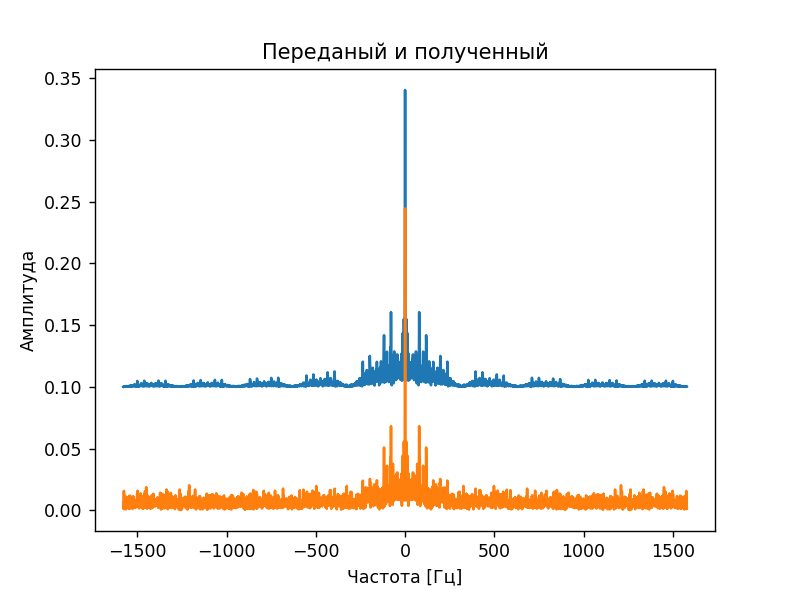
fft1[0] = 0

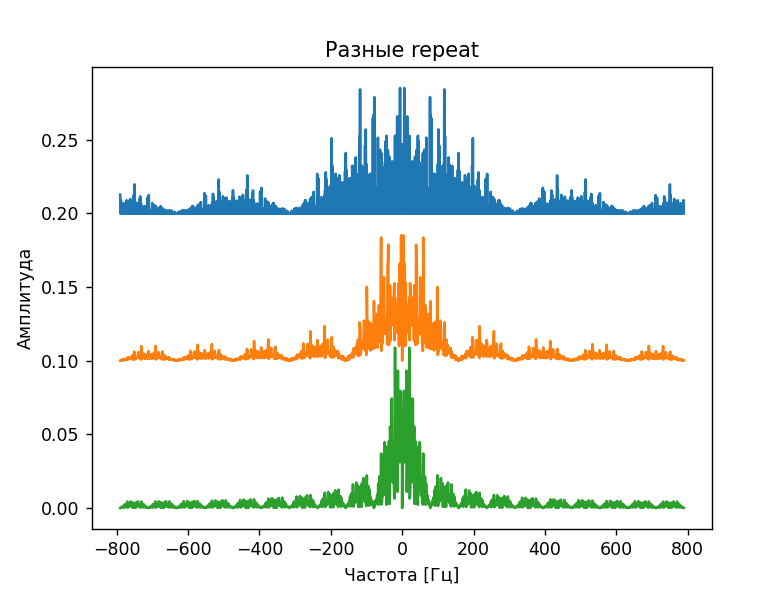
fft2[0] = 0

fft05 = abs(np.fft.fftshift(fft05)) + 0.2

fft1 = abs(np.fft.fftshift(fft1)) + 0.1

fft2 = abs(np.fft.fftshift(fft2))





**Заключение**

Я закрепил и структурировал знания, полученные в рамках изучения дисциплины «Основы систем мобильной связи». Исследовал как зависит количество ошибок распознавания сигнала от величины шума.



<https://github.com/DanielKorotaev/OSMS>