**Министерство цифрового развития, связи и массовых коммуникаций РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего**

**образования**

**«Сибирский государственный университет телекоммуникаций и информатики»**

**(СибГУТИ)**

**Расчетно-графическая работа**

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

**10 вариант**

Выполнил студент группы ИА-132:

Кацуро М.А.

Проверил преподаватель:

Дроздова В.Г.

Новосибирск 2023

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.

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.

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.

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.

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.

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.

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.

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.

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

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

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.

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).

14) Сделайте промежуточные выводы по каждому пункту работы и общее заключение.

15) Оформите работу. Отчет должен содержать титульный лист, содержание, цель и задачи работы, теоретические сведения, исходные данные, этапы выполнения работы, сопровождаемые скриншотами и графиками, демонстрирующими успешность выполнения, и промежуточными выводами, результирующими таблицами и заключение и ссылка в виде QR-кода на репозиторий с кодом (git).

**Ход выполнения работы**

Основной код 1-12 задания

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#define POLYNOMIAL "11011110" // порождающий полином

#define LENGTH 5 // длинна массива для последовательности голда

#define L 50 // длинна битов для имени и фамилии

#define M 50 // длинна бит для последовательности с CRC

#define G 50 // длинна бит для последовательности голда

#define N 6 // отчеты для 5 задания

#define RANDOM N\*(L+M+G) //длинна переменной для рандомного ввода

// Функция для кодирования ASCII-символов в битовую последовательность

**void** encodeToBinary(**char** \*input, **char** \*binaryData) {

**int** i, j, k;

**for** (i = 0; input[i] != '\0'; ++i) {

**int** asciiCode = input[i];

**for** (j = 7; j >= 0; --j) {

k = asciiCode >> j;

**if** (k & 1) {

\*binaryData = '1';

} **else** {

\*binaryData = '0';

}

++binaryData;

}

}

}

// Функция для вычисления CRC

**void** calculateCRC(**const** **char** \*data, **const** **char** \*polynomial, **char** \*result) {

**int** dataLength = strlen(data);

**int** polynomialLength = strlen(polynomial);

**char** extendedData[dataLength + polynomialLength - 1];

strcpy(extendedData, data);

strcat(extendedData, "0000000");

**for** (**int** i = 0; i < dataLength; i++) {

**if** (extendedData[i] == '1') {

**for** (**int** j = 0; j < polynomialLength; j++) {

extendedData[i + j] ^= polynomial[j] - '0';

}

}

}

strncpy(result, extendedData + dataLength, dataLength);

result[dataLength] = '\0';

}

// сдвиг по х из 4 лабы

**void** shift\_register\_x(**int** \*register\_state\_x) {

**int** feedback = (register\_state\_x[2] + register\_state\_x[3]) % 2;

**for** (**int** i = LENGTH - 1; i > 0; i--) {

register\_state\_x[i] = register\_state\_x[i - 1];

}

register\_state\_x[0] = feedback;

}

// сдвиг по у из 4 лабы

**void** shift\_register\_y(**int** \*register\_state\_y) {

**int** feedback = (register\_state\_y[1] + register\_state\_y[2]) % 2;

**for** (**int** i = LENGTH - 1; i > 0; i--) {

register\_state\_y[i] = register\_state\_y[i - 1];

}

register\_state\_y[0] = feedback;

}

// расчет последовательности голда

**void** generate\_pseudo\_random\_sequence(**char** \*result, **int** \*register\_state\_x, **int** \*register\_state\_y, **int** length) {

printf("Последовательность Голда равняется: ");

**for** (**int** i = 0; i < length; i++) {

**int** bit = (register\_state\_x[4] + register\_state\_y[4]) % 2;

printf("%d", bit);

shift\_register\_x(register\_state\_x);

shift\_register\_y(register\_state\_y);

\*result = bit + '0';

++result;

}

\*result = '\0';

printf("\n\n");

}

// поиск максимальной корреляции

**double** calculate\_correlation(**char** \*sequence1, **char** \*sequence2, **int** offset) {

**double** correlation = 0.0;

**int** length = strlen(sequence1);

**int** length2 = strlen(sequence2);

**for** (**int** i = 0; i < length; i++) {

correlation += (sequence1[i] - '0') \* (sequence2[(i + offset) % length2] - '0');

}

**return** correlation / length;

}

// функция для уменьшения массива на N

**void** decreaseSequence(**char** \*input, **char** \*output, **int** length) {

**int** inputLength = strlen(input);

**int** outputIndex = 0;

**for** (**int** i = 0; i < inputLength; i += length) {

output[outputIndex++] = input[i];

}

output[outputIndex] = '\0';

}

// функция для поиска аналогичных последовательностей в массиве и удаления ее

**void** removeDuplicates(**char** \*input, **const** **char** \*pattern, **char** \*output) {

**int** inputLength = strlen(input);

**int** patternLength = strlen(pattern);

**int** outputIndex = 0;

**for** (**int** i = 0; i < inputLength; i += patternLength) {

// Сравниваем подстроку с шаблоном

**if** (strncmp(input + i, pattern, patternLength) != 0) {

strncpy(output + outputIndex, input + i, patternLength); // Если не совпадает, копируем в выходной массив

outputIndex += patternLength;

}

}

output[outputIndex] = '\0';

}

// функция преобразования битовой последовательности в ASCII-символы

**void** binaryToASCII(**char** \*binaryData, **char** \*asciiResult, **int** nameBit) {

**int** binaryLength = strlen(binaryData);

**int** asciiIndex = 0;

**for** (**int** i = 0; i < nameBit; i += 8) {

**char** byte[9];

strncpy(byte, binaryData + i, 8);

byte[8] = '\0';

// Преобразуем бинарные данные в целое число

**int** asciiCode = strtol(byte, **NULL**, 2);

// Преобразуем целое число в ASCII-символ

asciiResult[asciiIndex++] = (**char**)asciiCode;

}

asciiResult[asciiIndex] = '\0';

}

// функция для генерации шума по формуле

**void** generateNormalNoise(**double** \*noise, **int** length, **double** mean, **double** stddev) {

**for** (**int** i = 0; i < length; i++) {

// Генерация нормально распределенного шума

**double** u1 = ((**double**) rand() / RAND\_MAX);

**double** u2 = ((**double**) rand() / RAND\_MAX);

**double** z = sqrt(-2.0 \* log(u1)) \* cos(2.0 \* M\_PI \* u2);

// Применение среднего и стандартного отклонения

noise[i] = mean + stddev \* z;

}

}

**int** main() {

**char** firstName[L], lastName[L]; // имя и фамилия

**char** binaryData[L\*8]; // битовая последовательность

**char** crcResult[M]; // результат срс

**char** randomsequence[G]; // результат последовательности голда

**char** resultfourth[2\*(L+M+G)]; // общий результат всех данных 4 заданий

**char** resultfifth[N\*(L+M+G)]; // результат моей последовательности с домнажением ее на N отчетов

**char** resultsixth[2\*N\*(L+M+G)]; // результат 6 задания

**char** resultseventh[2 \* RANDOM]; // результат 7 задания

**char** resulteight[2 \* RANDOM]; // результат 8 задания

**int** register\_state\_x[LENGTH] = {0, 1, 0, 1, 0}; // данные которые подаются на х для расчета голда

**int** register\_state\_y[LENGTH] = {1, 0, 0, 0, 1}; // данные которые подаются на у для расчета голда

**int** cons = pow(2, LENGTH) - 1;

printf("Введите ваше имя: ");

scanf("%s", firstName);

printf("Введите вашу фамилию: ");

scanf("%s", lastName);

encodeToBinary(firstName, binaryData); // преобразую в биты

**int** nameBit = strlen(binaryData); // для дальнейшей расшифровки использую

encodeToBinary(lastName, binaryData + 8 \* strlen(firstName)); // преобразую в биты

FILE \*file = fopen("secondTask.txt", "w");

**if** (file == **NULL**) {

perror("Не удалось открыть файл");

exit(EXIT\_FAILURE);

}

fprintf(file, "%s", binaryData);

fclose(file);

printf("Битовая последовательность: %s\n\n", binaryData);

generate\_pseudo\_random\_sequence(randomsequence, register\_state\_x, register\_state\_y, cons);// вызов функции для расчета последовательность голда

strcat(resultfourth, randomsequence); // добавляю в результат последовательность голда

strcat(resultfourth, binaryData); // добавляю в результат битовую последовательность

calculateCRC(binaryData, POLYNOMIAL, crcResult); // вызов функции для расчета срс

printf("CRC: %s\n\n", crcResult);

strcat(resultfourth, crcResult); // добавляю в результат срс последовательность

printf("Битовая последовательность с последовательностью голда: %s\n\n", resultfourth);

// сохряню в файл

file = fopen("thirdTask.txt", "w");

**if** (file == **NULL**) {

perror("Не удалось открыть файл");

exit(EXIT\_FAILURE);

}

printf("Битовая последовательность с последовательностью голда и CRC: %s\n\n", resultfourth);

fprintf(file, "%s", resultfourth);

fclose(file);

// Увеличени битовой последовательности на N

**int** index = 0;

**int** resultNIndex = 0;

**while** (resultfourth[index] != '\0') {

**for** (**int** i = 0; i < N; i++) {

resultfifth[resultNIndex++] = resultfourth[index];

}

++index;

}

resultfifth[resultNIndex] = '\0';

printf("Результат последовательности с отсчетами: %s\n", resultfifth);

// Сохраняем временные отсчеты в файл

file = fopen("fifthTask.txt", "w");

**if** (file == **NULL**) {

perror("Не удалось открыть файл");

exit(EXIT\_FAILURE);

}

fprintf(file, "%s", resultfifth);

fclose(file);

// 6 пункт

printf("Введите число от 0 до %d: ", RANDOM);

// символ массива на который встанет массив с отчетами

**int** element;

scanf("%d", &element);

// Проверка на корректность ввода

**if** (element < 0 || element >= 2 \* RANDOM) {

printf("Введенное число не входит в заданный диапазон\n");

**return** 1; // Возвращаем код ошибки

}

// Заполнение нулями до указанного индекса

**for** (**int** i = 0; i < element; i++) {

resultsixth[i] = '0';

}

// Вставка массива resultfifth начиная с указанного индекса

strcpy(resultsixth + element, resultfifth);

memset(resultsixth + element + strlen(resultfifth), '0', **sizeof**(resultsixth) - (element + strlen(resultfifth))); // заполняю оствашиеся элементы нулями

file = fopen("sixthTask.txt", "w");

// file = fopen("sixthTaskMinN.txt", "w");

// file = fopen("sixthTaskAverageN.txt", "w");

// file = fopen("sixthTaskHighN.txt", "w");

**if** (file == **NULL**) {

perror("Не удалось открыть файл");

exit(EXIT\_FAILURE);

}

// Вывод результата

printf("Массив после вставки:\n");

**for** (**int** i = 0; i < 2 \* RANDOM; i++) {

printf("%c", resultsixth[i]);

fprintf(file, "%c", resultsixth[i]);

}

printf("\n");

fclose(file);

// 7 пункт

file = fopen("seventhTask.txt", "w");

// file = fopen("seventhTaskMinN.txt", "w");

// file = fopen("seventhTaskAverageN.txt", "w");

// file = fopen("seventhTaskHighN.txt", "w");

**if** (file == **NULL**) {

perror("Не удалось открыть файл");

exit(EXIT\_FAILURE);

}

**double** noise[2 \* RANDOM];

**double** mean = 0.0; // Среднее значение

**double** stddev = 0.1; // Стандартное отклонение

generateNormalNoise(noise, 2 \* RANDOM, mean, stddev); // вызов функции которая создает шум

// Создаем массив resultseventh с шумом

**for** (**int** i = 0; i < (2\*RANDOM); i++) {

// Применяем шум к элементу resultsixth

**double** noisyValue = resultsixth[i] - '0' + noise[i];

fprintf(file, "%f ", noisyValue);

**if** (noisyValue < 0) {

noisyValue = 0;

} **else** **if** (noisyValue > 0.6) {

noisyValue = 1;

}

resultseventh[i] = (**char**)('0' + noisyValue);

}

printf("\n");

fclose(file);

// 8 пункт

**char** randomseq[RANDOM]; // для последовательности голда с N отчетами

**int** resultsNIndex = 0;

**int** indexs = 0;

// увеличиваю последовательность голда

**while** (randomsequence[indexs] != '\0') {

**for** (**int** i = 0; i < N; i++) {

randomseq[resultsNIndex++] = randomsequence[indexs];

}

++indexs;

}

// Находим максимальную корреляцию

**double** maxCorrelation = -1.0;

**int** syncStartSample = 0;

**for** (**int** offset = 0; offset < (2 \* RANDOM); offset++) { // ищу последовательность голда в массиве

**double** correlation = calculate\_correlation(randomseq, resultseventh, offset);

**if** (correlation > maxCorrelation) {

maxCorrelation = correlation;

syncStartSample = offset;

}

}

printf("Начальный семпл синхросигнала: %d\n", syncStartSample);

// избавляюсь от шума в начале

**int** resulteightIndex = 0;

**for** (**int** i = syncStartSample; i < (2 \* RANDOM); i++) {

resulteight[resulteightIndex++] = resultseventh[i];

}

resulteight[resulteightIndex] = '\0';

**int** resultfifthLength = strlen(resultfifth); // узнаю длинну данных из пятого пункта

// Создаем массив resultninth с размером, равным количеству элементов в resultfifth

**char** resultninth[resultfifthLength + 1];

// Узнаем, сколько элементов нужно скопировать из resulteight в resultninth

**int** elementsToCopy = resultfifthLength;

// Копируем элементы из resulteight в resultninth

strncpy(resultninth, resulteight, elementsToCopy);

resultninth[elementsToCopy] = '\0';

// 9 пункт

decreaseSequence(resultninth, resultninth, N); // вызываю функцию которая убирает отсчеты

resultninth[elementsToCopy] = '\0';

printf("Результат в массиве после удаления отсчетов:\n");

printf("%s\n", resultninth);

**char** resulttenth[2 \* (L + M + G)]; // хранятся данные без отчетов

**char** resulteleventh[L + M + G]; // данные без последователности голда

removeDuplicates(resultninth, randomsequence, resulttenth); // удаляю последовательность голда с помощью функции

printf("Результат данных после удаления последовательности голда:\n");

printf("%s\n", resulttenth);

**char** checkCRC[M]; // для повторной проверки срс

calculateCRC(resulttenth, POLYNOMIAL, checkCRC); // вызов функции для расчета срс на приеме

printf("CRC на приеме равняется: %s\n", checkCRC);

// Удаление повторяющихся элементов в resulteleventh на основе crcResult

removeDuplicates(resulttenth, crcResult, resulteleventh);

printf("Результат данных после удаления CRC:\n");

printf("%s\n", resulteleventh);

// расшифровка данных 12 пункт

**char** asciiResultName[nameBit / 8 + 1];

binaryToASCII(resulteleventh, asciiResultName, nameBit);

// Вывод расшифровки только первых nameBit бит

printf("Расшифрованное имя: %s\n", asciiResultName);

**int** allBit = strlen(resulteleventh); // узнаю все биты для расшифровки имени и фамилии по отдельности

**int** lastnameBit = allBit - nameBit; // длинна фамилии

**int** nameasciiSize = nameBit / 8 + 1;

**char** nameasciiResultRest[nameasciiSize];

binaryToASCII(resulteleventh + nameBit, nameasciiResultRest, lastnameBit);

// Вывод оставшейся части в ASCII

printf("Расшифрованная фамилия: %s\n", nameasciiResultRest);

**return** 0;

}

**Вывод этих заданий**



Еще одна проверка



**Визуализация графиков**

import numpy as np

import matplotlib.pyplot as plt

# Чтение битовой последовательности из файла

with open("secondTask.txt", "r") as file:

second\_task\_binary\_data = file.read()

# Преобразование битовой последовательности в массив нулей и единиц

second\_task\_binary\_array = np.array([int(bit) for bit in second\_task\_binary\_data]) # последовательность из второго задания

with open("thirdTask.txt", "r") as file:

fourth\_task\_binary\_data = file.read()

fourth\_task\_binary\_array = np.array([int(bit) for bit in fourth\_task\_binary\_data]) # последовательность из четвертого задания

with open("fifthTask.txt", "r") as file:

fifth\_task\_binary\_data = file.read()

fifth\_task\_binary\_array = np.array([int(bit) for bit in fifth\_task\_binary\_data]) # последовательность из пятого задания

with open("sixthTask.txt", "r") as file:

sixth\_task\_binary\_data = file.read()

sixth\_task\_binary\_array = np.array([int(bit) for bit in sixth\_task\_binary\_data if bit.strip()]) # фильтрация символов

with open("seventhTask.txt", "r") as file:

seventh\_task\_data = [float(value) for value in file.read().split()]

# Построение двух графиков в одном окне

plt.subplot(4, 1, 1)

plt.plot(second\_task\_binary\_array, drawstyle='steps-pre')

plt.title('Битовая последовательность')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.subplot(4, 1, 2)

plt.plot(fourth\_task\_binary\_array, drawstyle='steps-pre')

plt.title('Измененная битовая последовательность')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.subplot(4, 1, 3)

plt.plot(fifth\_task\_binary\_array, drawstyle='steps-pre')

plt.title('Битовая последовательность с N отчетами')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.subplot(4, 1, 4)

plt.plot(sixth\_task\_binary\_array, drawstyle='steps-pre')

plt.title('Массив Signal')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.tight\_layout()

plt.show()

# Визуализация массива отсчетов зашумленного принятого сигнала с использованием стем-графика

plt.plot(seventh\_task\_data, drawstyle='steps-pre')

plt.title('Визуализация зашумленного принятого сигнала ')

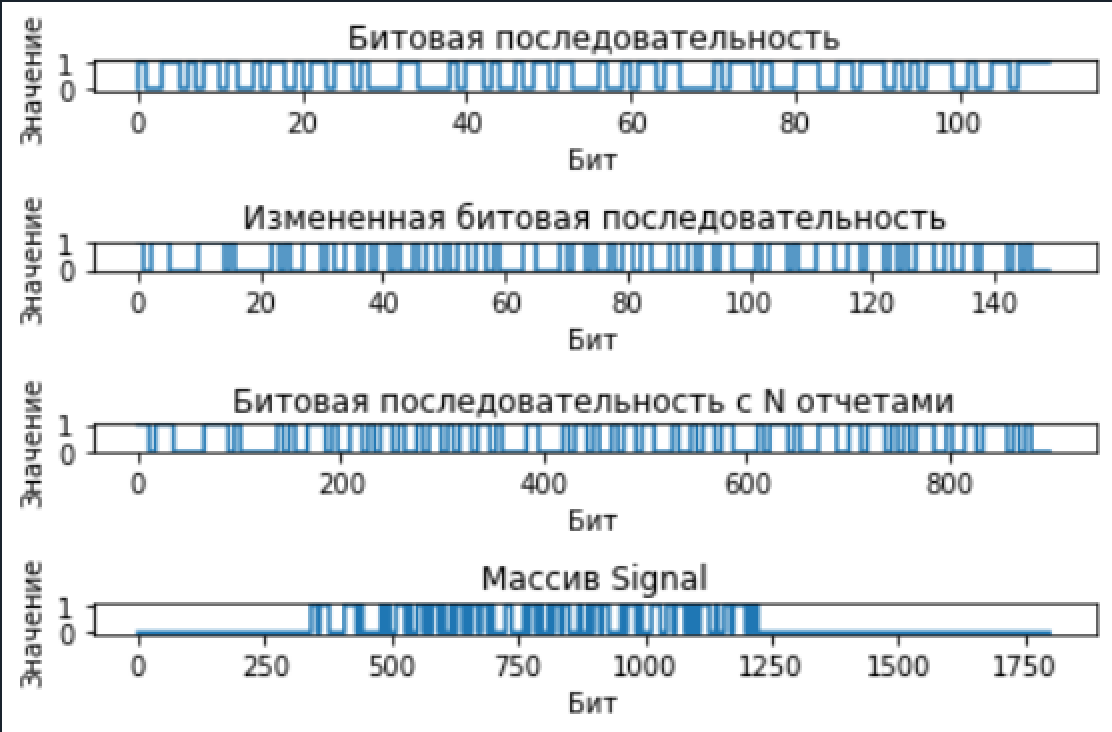
plt.xlabel('Отчеты')

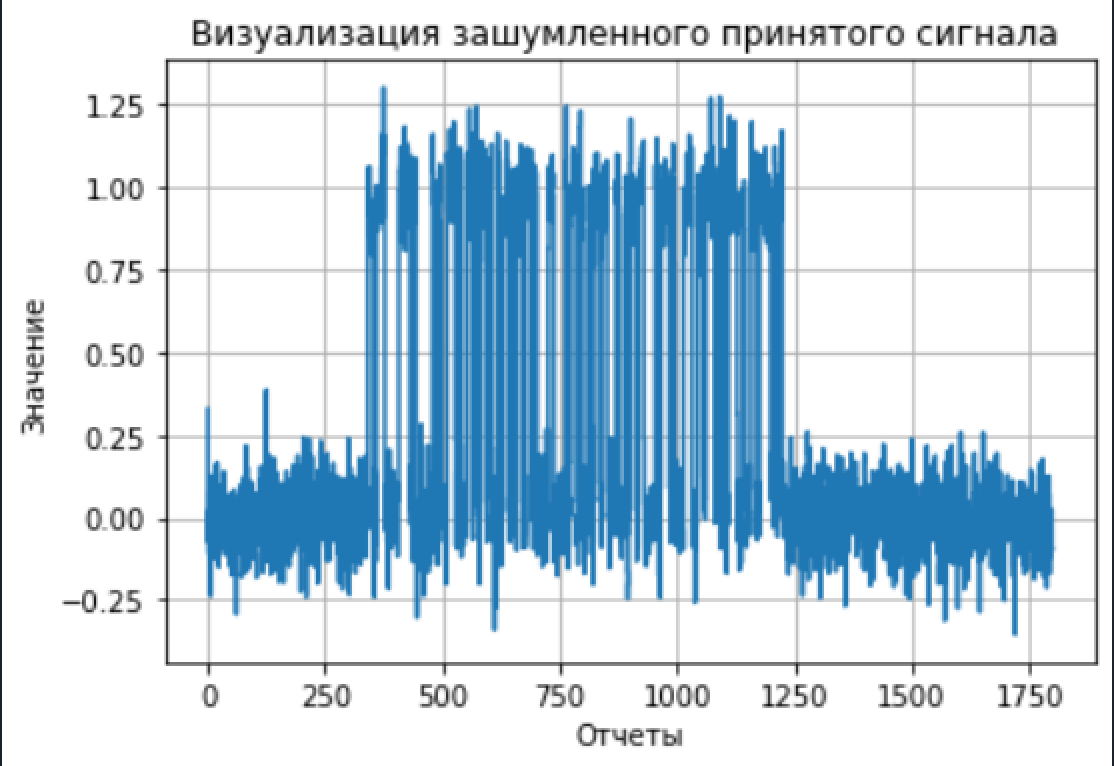
plt.ylabel('Значение')

plt.grid(True)

plt.show()

**Вывод графиков**





**Код для визуализации 13 пункта**

import numpy as np

import matplotlib.pyplot as plt

from scipy.fft import fft, fftfreq

# Чтение битовой последовательности из файла

with open("seventhTaskMinN.txt", "r") as file:

seventhMin\_binary\_data = np.array([float(value) for value in file.read().split()])

# Преобразование битовой последовательности в массив нулей и единиц

# seventhMin = np.array([float(bit) for bit in seventhMin\_binary\_data]) # последовательность которую мы принимаем с N = 2

with open("sixthTaskMinN.txt", "r") as file:

sixthMin\_binary\_data = file.read()

sixthMin = np.array([int(bit) for bit in sixthMin\_binary\_data]) # последовательность которую мы передаем на передачу с N = 2

with open("seventhTaskAverageN.txt", "r") as file:

seventhAverage\_binary\_data = np.array([float(value) for value in file.read().split()])

# seventhAverage = np.array([float(bit) for bit in seventhAverage\_binary\_data]) # последовательность которую мы принимаем с N = 6

with open("sixthTaskAverageN.txt", "r") as file:

sixthAverage\_binary\_data = file.read()

sixthAverage = np.array([int(bit) for bit in sixthAverage\_binary\_data if bit.strip()]) # последовательность которую мы передаем на передачу с N = 6

with open("seventhTaskHighN.txt", "r") as file:

seventhHigh\_binary\_data = np.array([float(value) for value in file.read().split()])

# seventhHigh = np.array([float(bit) for bit in seventhHigh\_binary\_data]) # последовательность которую мы принимаем с N = 10

with open("sixthTaskHighN.txt", "r") as file:

sixthHigh\_binary\_data = file.read()

sixthHigh = np.array([int(bit) for bit in sixthHigh\_binary\_data if bit.strip()]) # последовательность которую мы передаем на передачу с N = 10

# сигнал на передаче

sendingMinN = np.fft.fft(sixthMin)

sendingAvgN = np.fft.fft(sixthAverage)

sendingHighN = np.fft.fft(sixthHigh)

# сигнал на приеме

receptionMinN = np.fft.fft(seventhMin\_binary\_data)

receptionAvgN = np.fft.fft(seventhAverage\_binary\_data)

receptionHighN = np.fft.fft(seventhHigh\_binary\_data)

kM = np.arange(1200)

kA = np.arange(2700)

kH = np.arange(4200)

plt.plot(kH, np.abs(receptionHighN), label='Сигнал на приеме с N=10', color='red')

plt.plot(kA, np.abs(receptionAvgN), label='Сигнал на приеме с N=10', color='blue')

plt.plot(kM, np.abs(receptionMinN), label='Сигнал на приеме с N=2', color='orange')

plt.title('Передаваемый сигнал')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.legend()

plt.grid(True)

plt.show()

plt.plot(kH, np.abs(sendingHighN), label='Сигнал на передаче с N=10', color='red')

plt.plot(kA, np.abs(sendingAvgN), label='Сигнал на передаче с N=6', color='blue')

plt.plot(kM, np.abs(sendingMinN), label='Сигнал на передаче с N=2', color='orange')

plt.title('Принимаемый сигнал')

plt.xlabel('Бит')

plt.ylabel('Значение')

plt.legend()

plt.grid(True)

plt.show()

**Вывод графиков**





**QR-код на репозиторий git**

