МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ВС и ТС

Отчет по лабораторной работе №4
по дисциплине
Основы систем мобильной связи

по теме:

ИЗУЧЕНИЕ КОРРЕЛЯЦИОННЫХ СВОЙСТВ ПОСЛЕДОВАТЕЛЬНОСТЕЙ, ИСПОЛЬЗУЕМЫХ ДЛЯ СИНХРОНИЗАЦИИ В СЕТЯХ МОБИЛЬНОЙ СВЯЗИ

Студент:
Группа ИА-331 *Я.А Гмыря*

Предподаватель:
Заведующая кафедрой ТС и ВС *В.Г Дроздова*

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ЦЕЛЬ И ЗАДАЧИ

Цель:

Получить представление о том, какие существуют псевдослучайные двоичные последовательности, какими корреляционными свойствами они обладают и как используются для синхронизации приемников и передатчиков в сетях мобильной связи

Задачи:

1) Напишите программу на языке C/C++ для генерации последовательности Голда, используя схему, изображенную на рисунке 4.4, если ваша группа с четным номером и 4.5 – если с нечетным, и порождающие полиномы x и y, при этом x – это ваш порядковый номер в журнале в двоичном формате (5 бит), а y – это $x+7$ (5 бит). Например, ваш номер 22, значит:
$x = 1\ 0\ 1\ 1\ 0$, $y = 1\ 1\ 1\ 0\ 1$.

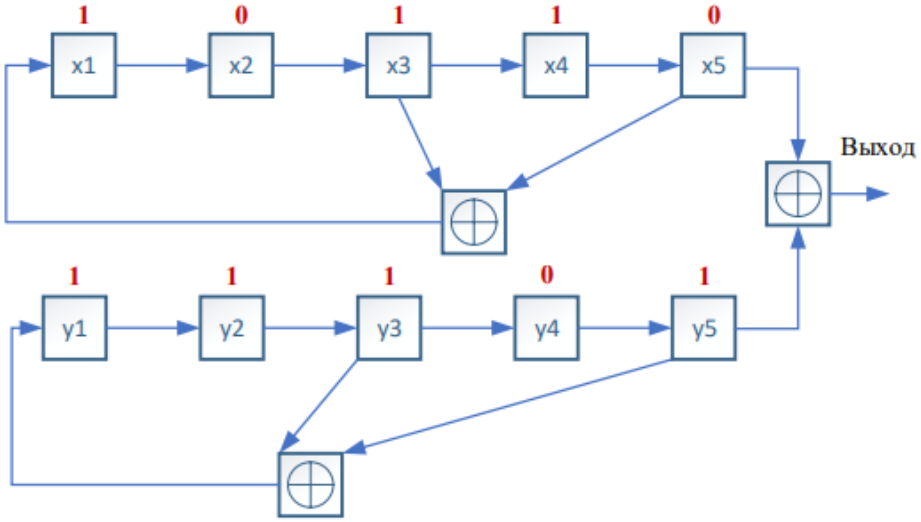

Рис. 4.4. Генерация последовательности Голда (вариант для *четного* номера группы)

Рисунок 1 — Задание к лабораторной работе

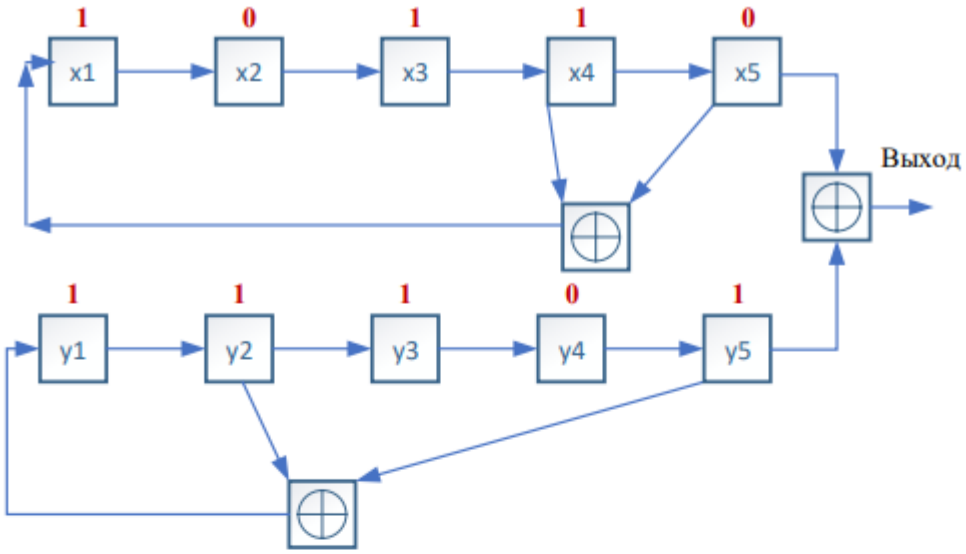

Рис. 4.5. Генерация последовательности Голда (вариант для *нечетного* номера группы)

1) Выведите получившуюся последовательность на экран.
2) Сделайте поэлементный циклический сдвиг последовательности и посчитайте автокорреляцию исходной последовательности и сдвинутой. Сформируйте таблицу с битовыми значениями последовательностей, в последнем столбце которой будет вычисленное значение автокорреляции, как показано в примере ниже.

Сдвиг	бит 1	бит 2	бит 3	Автокорреляция
0	1	1	0	1
1	0	1	1	-1/3
2	0	1	1	-1/3
3	1	1	0	1

3) Сформируйте еще одну последовательность Голда, используя свою схему (рис.4.4 или 4.5), такую что $x=x+1$, а $y= y-5$.
4) Вычислите значение взаимной корреляции исходной и новой последовательностей и выведите в терминал.
5) Проделайте шаги 1-5 в Matlab. Используйте функции xcorr() и autocorr() для вычисления соответствующих корреляций. Сравните результаты, полученные в Matlab и C/C++.
6) Выведите на график в Matlab функцию автокорреляции в зависимости от величины задержки (lag).

Рисунок 2 — Задание к лабораторной работе

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Что такое псевдослучайная двоичная последовательность?

Псевдослучайные двоичные последовательности (PN-sequences – PseudoNoise) – частный случай псевдослучайных последовательностей, элементы которой принимают значения из [-1;1].

1.2 Область применения псевдослучайных двоичных последовательностей

Такие последовательности очень часто используются в сетях мобильной связи.

1.2.1 Оценка BER

BER – Bit Error Rate. Передатчик передает приемнику заранее известную PN последовательность бит, а приемник, анализируя значения бит на конкретных позициях, вычисляет количество искаженных бит и вероятность битовой ошибки в текущих радиоусловиях, что затем может быть использовано для работы алгоритмов, обеспечивающих помехозащищенность системы

1.2.2 Синхронизация

Используя заранее известную синхронизирующую PN-последовательность (синхросигнал), приемник делает корреляционный прием данных и если фиксируется корреляционный пик, то на стороне приема можно корректно разметить буфер с отсчетами на символы, слоты, кадры и пр.

1.2.3 Расширение спектра

Используется для повышения эффективности передачи информации с помощью модулированных сигналов через канал с сильными линейными ис-

кажениями (замираниями), делая систему устойчивой к узкополосным помехам (например, в 3G WCDMA).

1.3 Свойства псевдослучайных двоичных последовательностей

1.3.1 Сбалансированность

Число единиц и число нулей на любом интервале последовательности должно отличаться не более чем на одну.

1.3.2 Цикличность

В каждом фрагменте псевдослучайной битовой последовательности примерно половину составляли циклы длиной 1, одну четверть – длиной 2, одну восьмую – длиной 3 и т.д.

1.3.3 Корреляция

Корреляция оригинальной битовой последовательности с ее сдвинутой копией должна быть минимальной.

1.4 Генерация псевдослучайных двоичных последовательностей

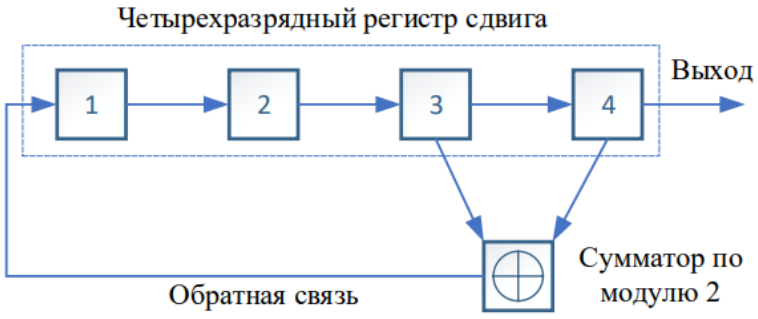

Рис. 4.1. Пример способа формирования псевдослучайной битовой последовательности.

Рисунок 3 — Генерация m-последовательности

Есть сдвиговый регистр на N бит. Младший (правый) бит идет в итоговую m-последовательность, а исходная последовательность циклически

сдвигается, причем старший (левый) бит вычисляется как xor между последним и предпоследним битом исходной последовательности. Итоговая последовательность будет иметь длину $2^m - 1$. Вычислять ее можно бесконечно, но после $2^m - 1$ элемента начнутся повторы.

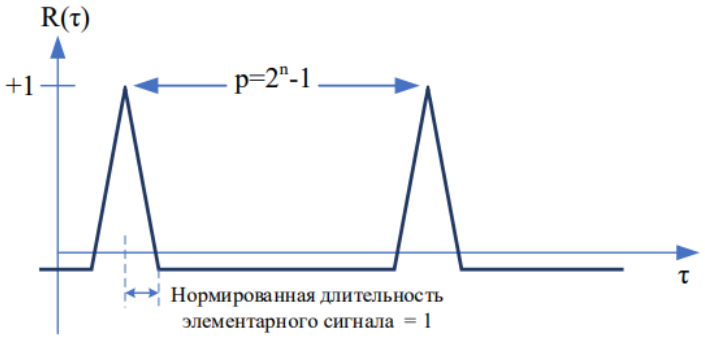

Рис. 4.2. Автокорреляционная функция псевдослучайной бинарной последовательности в зависимости от величины задержки

Рисунок 4 — Автокорреляция псевдослучайной двоичной последовательности

Чем длиннее последовательность, тем выше пик ее автокорреляционной функции. Чем острее автокорреляционный пик (то есть чем длиннее последовательность), тем удобней использовать данные последовательности для решения проблем синхронизации в сетях мобильной связи.

1.5 Разновидности псевдо-шумовых битовых последовательностей

Существуют коды Баркера, коды Голда, коды Касами, коды Уолша-Адамара. Коды Голда формируются путем суммирования по модулю 2 двух Мпоследовательностей одинаковой длины. Коды Касами также формируются из М-последовательностей путем взятия периодических выборок из этих последовательностей и суммированием их по модулю два. Данные коды обладают очень хорошими взаимокорреляционными свойствами.

1.6 Способ поиска синхросигнала

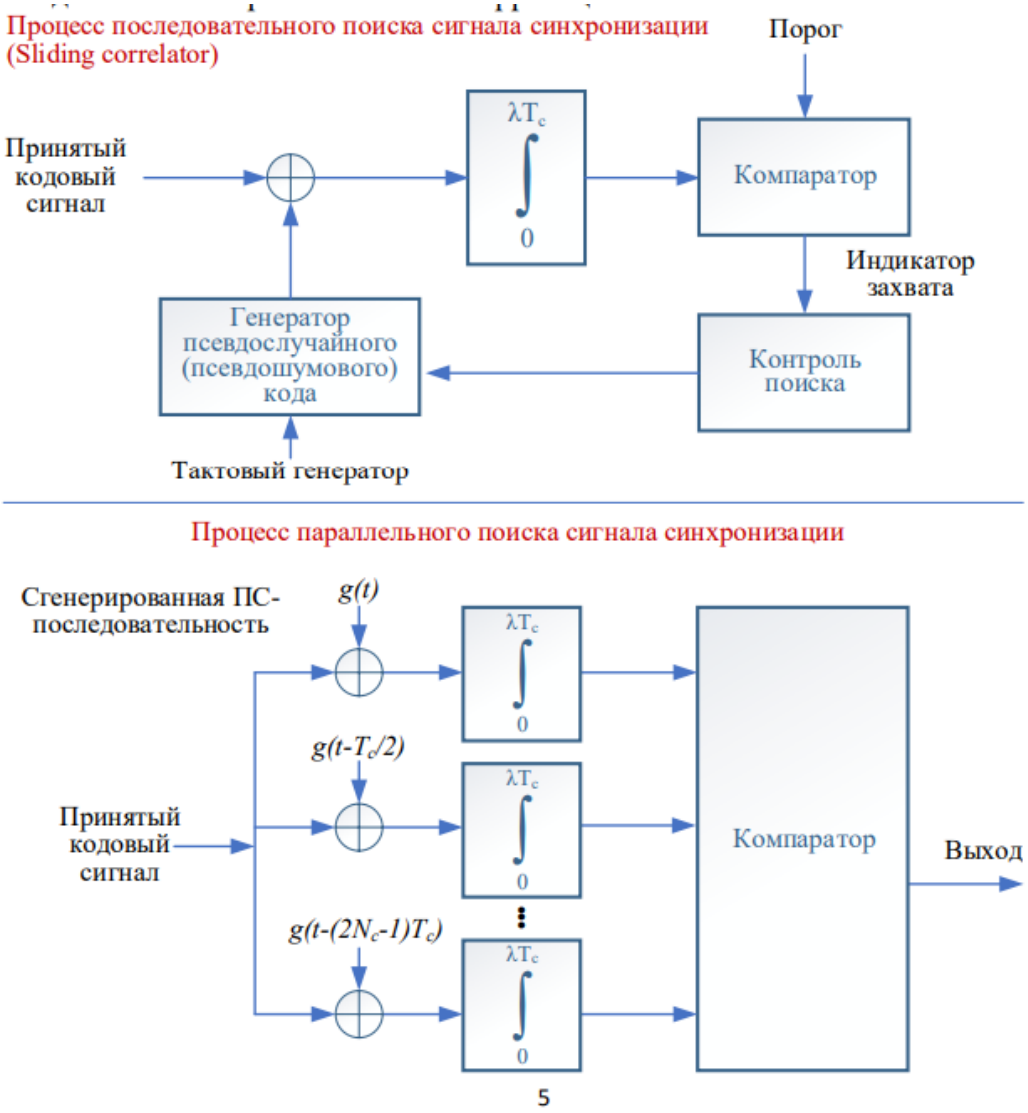

Рисунок 5 — Способы поиска синхросигнала

ГЕНЕРАЦИЯ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

2.1 C++

Для генерации псевдослучайных последовательностей буду использовать в основном битовые операции, чтобы поработать с бинарным представлением числовых типов данных. Исходные и итоговые последовательности будут храниться в одной переменной типа int. Само число не несет в себе смысла, имеет смысл только его бинарное представление.

2.1.1 Генерация m-последовательности

2.1.2 Используемые define

```
    #define NUMBER_IN_JOURNAL 6
    #define NUM_FORMAT 5
    #define MAX_BIT_SIZE pow(2, NUM_FORMAT) - 1
```

Здесь NUMBER_IN_JOURNAL необходим для формирования правильных исходных последовательностей для моего варианта, NUM_FORMAT - формат бинарного представления чисел (максимальное кол-во знаков), MAX_BIT_SIZE - размер итоговой последовательности.

2.1.3 Генерация

```
template <typename T>
int m_seq_gen(T x_seq, int xor_bit1, int xor_bit2){

    int result = 0;

    //for xor operation
    int x1, x2 = 0;

    for(int i = MAX_BIT_SIZE-1; i >= 0; --i){
        //add to result last bit
        result |= (x_seq & 1) << i;
```

```
        //get last bit
        x1 =(x_seq >> 1 * xor_bit1) & 1;

        //get pre-last bit
        x2 = (x_seq >> 1 * xor_bit2) & 1;

        //shift and add xor operation result in first bit of
           x_seq
        x_seq = ((x_seq >> 1) | ((x1 ^ x2) << NUM_FORMAT-1));
    }

    return result;
}
```

Функция принимает число x_seq, бинарное представление которого - исходная битовая последовательность. Переменные xor_bit1, xor_bit2 определяют, какие биты участвуют в формировании бита обратной связи (который после сдвига числа становится старшим битом). Отсчет битов идет справа. Выбор битов для формирования бита обратной связи важна, поскольку мы хотим сгенерировать последовательность Голда, которая состоит из двух m-последовательностей. При формировании этих двух m-последовательностей выбираются разные биты для формирования бита обратной свзязи. Функция возвращает 32-битную последовательность. Если битовая последовательность меньше 32 бит, то все будет корреткно, но при выводе числа в бинарном виде нужно будет учитывать реальный размер последовательности, чтобы правильно ”обрезать”двоичное представление числа и вывести только нужную часть.

В функции создается переменная result, которая будет содержать итоговую m-последовательность. Переменные x1,x2 нужны для хранения и работы с битами, которые формируют бит обратной связи.

В цикле итерируемся по итоговой m-последовательности от старшего бита к младшему и на каждой итерации заполняем итоговую m-последовательность. Далее в строке 11 записываем в итоговую последовательность последний бит исходной последовательности. Делается это путем операции побитового И с исходной последовательности и единицей и

сдвигом результата на i бит. После побитового И получаем последний бит исходной последовательности (следует из бинарного представления единицы и таблицы истинности для операции И). Сдвиг позволяет сместить полученный бит влево так, чтобы он стал i-ым битом итоговой последовательности. Операция побитового ИЛИ необходима, чтобы записать значение бита в result.

Далее получаем биты, участвующие в формировании бита обратной связи. Делается это путем сдвига исходной последовательности так, чтобы нужный бит стал последним в числе, с последующей операцией побитового И с единицей, чтобы выделить последний бит.

Перед началом следующей итерации осталось сдвинуть исходную последовательность. Для этого сдвигаем последовательность на 1 бит вправо и маской ((x1 x2) « NUM_FORMAT-1)) с помощью побитового ИЛИ записываем бит обратной связи, который получем операцией xor между x1 и x2.

2.1.4 Вспомогательные функции

2.1.5 Вывод бинарного числа

```
template <typename T>
void print_bin(T number){

    for(int i = MAX_BIT_SIZE-1; i >= 0; --i){
        printf("%d ", (number >> i) & 1);
    }
}
```

В языке C/C++ изначально нет возможности выводить число в бинарном виде, поэтому я написал специальную функцию. Она принимает на вход число (битовую последовательность) и ничего не возвращает. В функции происходит итерация по числу от старшего бита к младшему, чтобы выводить число слева на право, а не наоборот. На каждой итерации с помощью операции сдвига и побитового И с единицей выделяется нужный бит и выводится.

2.1.6 Циклический сдвиг

```
template <typename T>
T cycle_shift(T number, int shift) {

    for (int i = 0; i < shift; ++i) {
        T last_bit = number & 1;
        number >>= 1;
        number |= (last_bit << (static_cast<int>(MAX_BIT_SIZE) -
           1));
    }

    return number;
}
```

Функция приниммает на вход битовую последовательность и сдвиг. Возвращает сдвинутую последовательность. В цикле на каждой итерации сдвига выделяем последний бит при помощи побитового И, сдвигаем число на 1 знак вправо и в послендий бит с помощью побитового ИЛИ записываем последний бит, который предварительно сдвинем.

2.1.7 Сравнение битовых последовательностей

```
template <typename T>
void bit_seq_compare(T seq_1, T seq_2, int& coincided, int&
   non_concided){

    //reset variable
    coincided = 0;
    non_concided = 0;

    //variables for bit values
    int seq_1_bit = 0;
    int seq_2_bit = 0;

    //iterate on bits
    for(int i = 0; i < MAX_BIT_SIZE; ++i){
        //get bit values
        seq_1_bit = (seq_1 >> i) & 1;
        seq_2_bit = (seq_2 >> i) & 1;
```

```
        //compare bit
        if(seq_1_bit == seq_2_bit){
            ++coincided;
        } else{
            ++non_concided;
        }

    }
}
```

Функция принимает две последовательности, которые будут сравниваться, и 2 переменные по ссылке, в которых вернется кол-во совпадающих и несовпадающих битов. Сама функция ничего не возвращает. В функции итерируемся по битам, получаем последний бит каждой последовательности и сравниваем их.

2.1.8 Тесты

Зададим начальные последовательности, сгенерируем m-последовательности и последовательности Голда. Выведем всё на экран.

```
    int main(){
    int l = 3;
    //define base seq
    int8_t x_seq1 = NUMBER_IN_JOURNAL;
    int8_t y_seq1 = NUMBER_IN_JOURNAL + 7;

    int8_t x_seq2 = NUMBER_IN_JOURNAL + l;
    int8_t y_seq2 = y_seq1 - 5;

    //output base seq

    printf("x_seq1: ");
    print_bin(x_seq1);

    printf("\n");

    printf("y_seq1: ");
    print_bin(y_seq1);
```

```
    printf("\n");

    printf("x_seq1: ");
    print_bin(x_seq2);

    printf("\n");

    printf("y_seq2: ");
    print_bin(y_seq2);

    printf("\n\n");

    //generate m-seq
    int m_seq_1 = m_seq_gen(x_seq1, 0, 1);
    int m_seq_2 = m_seq_gen(y_seq1, 0, 3);

    int m_seq_3 = m_seq_gen(x_seq2, 0, 1);
    int m_seq_4 = m_seq_gen(y_seq2, 0, 3);

    printf("m_seq1: ");
    print_bin(m_seq_1);

    printf("\n");

    printf("m_seq2: ");
    print_bin(m_seq_2);

    printf("\n");

    printf("m_seq3: ");
    print_bin(m_seq_3);

    printf("\n");

    printf("m_seq4: ");
    print_bin(m_seq_4);

    printf("\n\n");

    //generate golden-seq
    int golden_seq1 = m_seq_1 ^ m_seq_2;
    int golden_seq2 = m_seq_3 ^ m_seq_4;
```

```
    //output godlen-seq
    printf("golden-seq1: ");
    print_bin(golden_seq1);

    printf("\n");

    printf("golden-seq2: ");
    print_bin(golden_seq2);
    printf("\n\n\n");

    return 0;
}
```

```
x_seq1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
y_seq1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1
x_seq1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
y_seq2: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0

m_seq1: 0 1 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1
m_seq2: 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 0 0 1 0 1 0 1 1
m_seq3: 1 0 0 1 0 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 1 1 1
m_seq4: 0 0 0 1 0 1 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0

golden-seq1: 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0
golden-seq2: 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 1 0 1
```

Рисунок 6 — Вывод получившихся последовательностей

Первую последовательность Голда будем циклически сдвигать и вычислять корреляцию с исходной последовательностью

```
    int coincided = 0;
    int non_coincided = 0;

    printf("lag:\t\toriginal golden_seq:\t\t\t\t\t\tshift
       golden_seq:\t\t\t  c:\tnc:\tcorr:\n");

    for(int i = 0; i < MAX_BIT_SIZE+1; ++i){
        printf("%d  ", i);

        bit_seq_compare(golden_seq1, cycle_shift(golden_seq1,
           i), coincided, non_coincided);

        print_bin(golden_seq1);
```

```
        printf("   ");

        print_bin(cycle_shift(golden_seq1, i));

        printf("   ");

        printf("%d\t%d", coincided, non_coincided);

        printf("   ");

        printf("%lf\n", (coincided - non_coincided) /
           static_cast<double>(MAX_BIT_SIZE));
    }

    return 0;
```

```
lag:          original golden_seq:                                shift golden_seq:                    c:   nc:     corr:
0  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   31   0  1.000000
1  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1   13   18  -0.161290
2  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1   17   14  0.096774
3  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1   19   12  0.225806
4  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1   13   18  -0.161290
5  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1   21   10  0.354839
6  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0   13   18  -0.161290
7  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1   15   16  -0.032258
8  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1   15   16  -0.032258
9  1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0   17   14  0.096774
10 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1   15   16  -0.032258
11 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0   15   16  -0.032258
12 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1   17   14  0.096774
13 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1   13   18  -0.161290
14 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 0   21   10  0.354839
15 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1   13   18  -0.161290
16 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1   13   18  -0.161290
17 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1 1   21   10  0.354839
18 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0 1   13   18  -0.161290
19 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0 0   17   14  0.096774
20 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0 0   15   16  -0.032258
21 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1 0   15   16  -0.032258
22 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 1   17   14  0.096774
23 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0   15   16  -0.032258
24 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1 0   15   16  -0.032258
25 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0 1   13   18  -0.161290
26 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1 0   21   10  0.354839
27 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0 1   13   18  -0.161290
28 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1 0   19   12  0.225806
29 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1 1   17   14  0.096774
30 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 0 1   13   18  -0.161290
31 1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   1 1 0 1 0 1 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 0   31   0  1.000000
```

Рисунок 7 — Автокорреляция последовательности Голда

Здесь c - число совпадающих битов, nc - несовпадающих. Можем заметить пик корреляции при shift=0, и shift=31. Таким образом приемник может точно идентифицировать синхро-последовательность даже в случае помех или сдвига фазы сигнала. Корреляция вычислялась по формуле $\frac{p}{MAX\_BIT\_SIZE}$, где $p = c - nc$

Вычислим кросс-корреляцию между двумя последовательностями Голда:

```
    //xcorr
    bit_seq_compare(golden_seq1, golden_seq2, coincided,
       non_coincided);

    printf("Cross-corr b/w golden_seq1 & golden_seq2: %f\n\n",
       (coincided - non_coincided) /
       static_cast<double>(MAX_BIT_SIZE));
```

```
Cross-corr b/w golden_seq1 & golden_seq2: -0.096774
```

Рисунок 8 — Кросс-корреляция между последовательностями Голда

Видим, что корреляция отрицательная, т.е приемнику будет трудно спутать между собой ожидаемую последовательность с искаженной или чужой.

2.2 MATLAB

Реализуем ту же логику в MATLAB с визуализацией.

2.2.1 Генерация m-последовательности

В бибилотеках MATLAB есть готовая функция для генерации m-последовательностей comm.PNSequence, но она работает по какому-то другому алгоритму, поэтому я написал свою функцию для вычисления m-последовательности. Алгоритм аналогичен коду на C++, но в MATLAB я уже не работал с битами, а работал с векторами.

```
%% function for generate m-seq
function m_seq = m_seq_gen(seq, xor_el1, xor_el2)

    m_seq_len = 2^length(seq) - 1;
    m_seq = zeros(m_seq_len, 1);

    for i=1:m_seq_len
```

```
        %get bits for feedback
        xor_bit1 = seq(end-xor_el1);
        xor_bit2 = seq(end-xor_el2);
        %write last bit default seq to result seq
        m_seq(i) = seq(end);
        %shift seq
        seq = circshift(seq, 1);
        %write in head bit from feedback
        seq(1) = xor(xor_bit1, xor_bit2);
    end

end
```

2.2.2 Тесты

Зададим исходные последовательности, m-последовательности и последовательности Голда. Каждый этап визуализируем:

```
%% define model params
NUM_IN_JOURNAL = 6;
BIT_SIZE = 5;
l = 3;

%% define base seq
seq1 = dec2bin(NUM_IN_JOURNAL, BIT_SIZE) - '0';
seq2 = dec2bin(NUM_IN_JOURNAL + 7, BIT_SIZE) - '0';

seq3 = dec2bin(NUM_IN_JOURNAL + l, BIT_SIZE) - '0';
seq4 = dec2bin(NUM_IN_JOURNAL + 2, BIT_SIZE) - '0';

%% output result
fprintf("seq1: ");
disp(seq1);

fprintf("seq2: ");
disp(seq2);

timeline = 0:1:BIT_SIZE-1;

figure;
plot(timeline, seq1);
```

```
hold on;
plot(timeline, seq2);
plot(timeline, seq3);
plot(timeline, seq4);
hold off;
grid on;
legend(sprintf("seq1: %s", dec2bin(NUM_IN_JOURNAL, BIT_SIZE)),
   ...
        sprintf("seq2: %s", dec2bin(NUM_IN_JOURNAL + 7,
           BIT_SIZE)), ...
        sprintf("seq3: %s", dec2bin(NUM_IN_JOURNAL + 1,
           BIT_SIZE)), ...
        sprintf("seq4: %s", dec2bin(NUM_IN_JOURNAL + 2,
           BIT_SIZE)));
xlabel("t,s");
ylabel("A,v");
title("base seq");

%% generate m-seq
m_seq1 = m_seq_gen(seq1, 0, 1);
m_seq2 = m_seq_gen(seq2, 0, 3);
m_seq3 = m_seq_gen(seq3, 0, 1);
m_seq4 = m_seq_gen(seq4, 0, 3);

%% output result

fprintf("m-seq1: ");
disp(m_seq1);

fprintf("m-seq2: ");
disp(m_seq2);

fprintf("m-seq3: ");
disp(m_seq3);

fprintf("m-seq4: ");
disp(m_seq4);

timeline = 0:1:2^BIT_SIZE-2;

figure;
```

```
subplot(4, 1, 1);
plot(timeline, m_seq1);
xlabel("t,s");
ylabel("A,v");
title("m-seq1");

subplot(4, 1, 2);
plot(timeline, m_seq2);
xlabel("t,s");
ylabel("A,v");
title("m-seq2")

subplot(4, 1, 3);
plot(timeline, m_seq3);
xlabel("t,s");
ylabel("A,v");
title("m-seq3")

subplot(4, 1, 4);
plot(timeline, m_seq4);
xlabel("t,s");
ylabel("A,v");
title("m-seq4")
grid on;


%% generate golden-seq
golden_seq1 = xor(m_seq1, m_seq2);
golden_seq2 = xor(m_seq3, m_seq4);

%% output result
fprintf("golden_seq1: ")
disp(golden_seq1');

fprintf("golden_seq2: ")
disp(golden_seq2');

figure;

subplot(2, 1, 1);
plot(timeline, golden_seq1);
xlabel("t,s");
```

```
ylabel("A,v");
title("golden-seq1");

subplot(2, 1, 2);
plot(timeline, golden_seq2);
xlabel("t,s");
ylabel("A,v");
title("golden-seq2");
```

Визуализация:

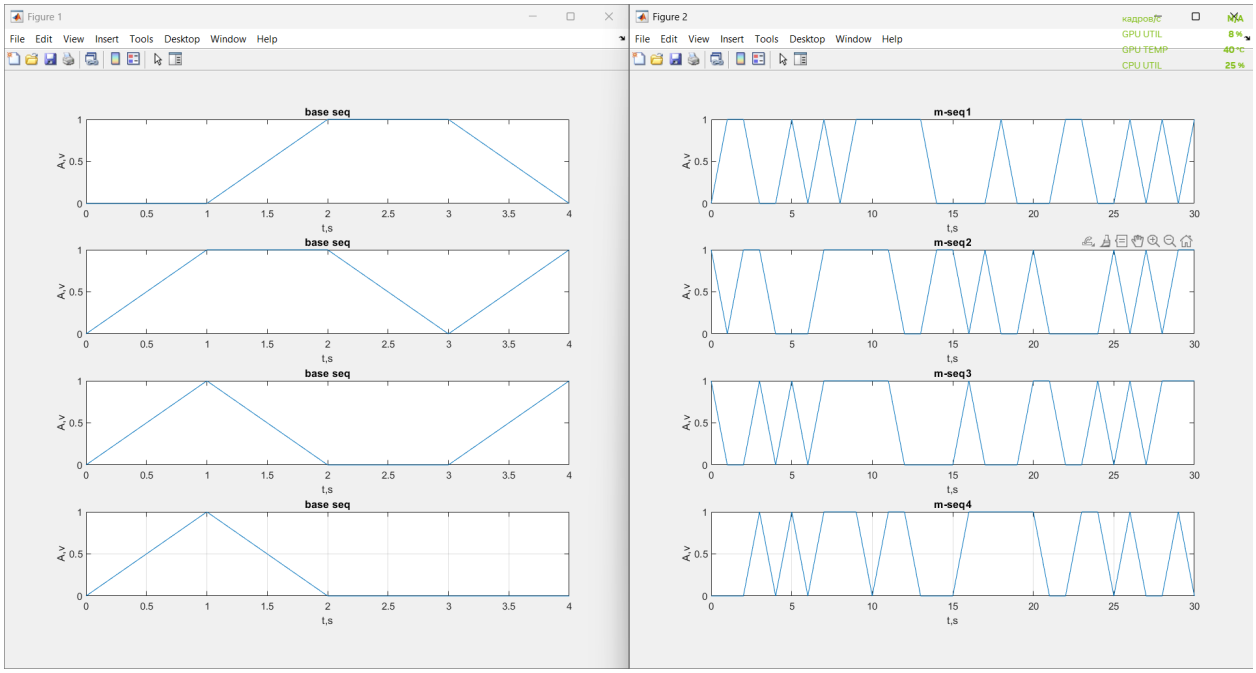

Рисунок 9 — Визуализация исходных и m последовательностей

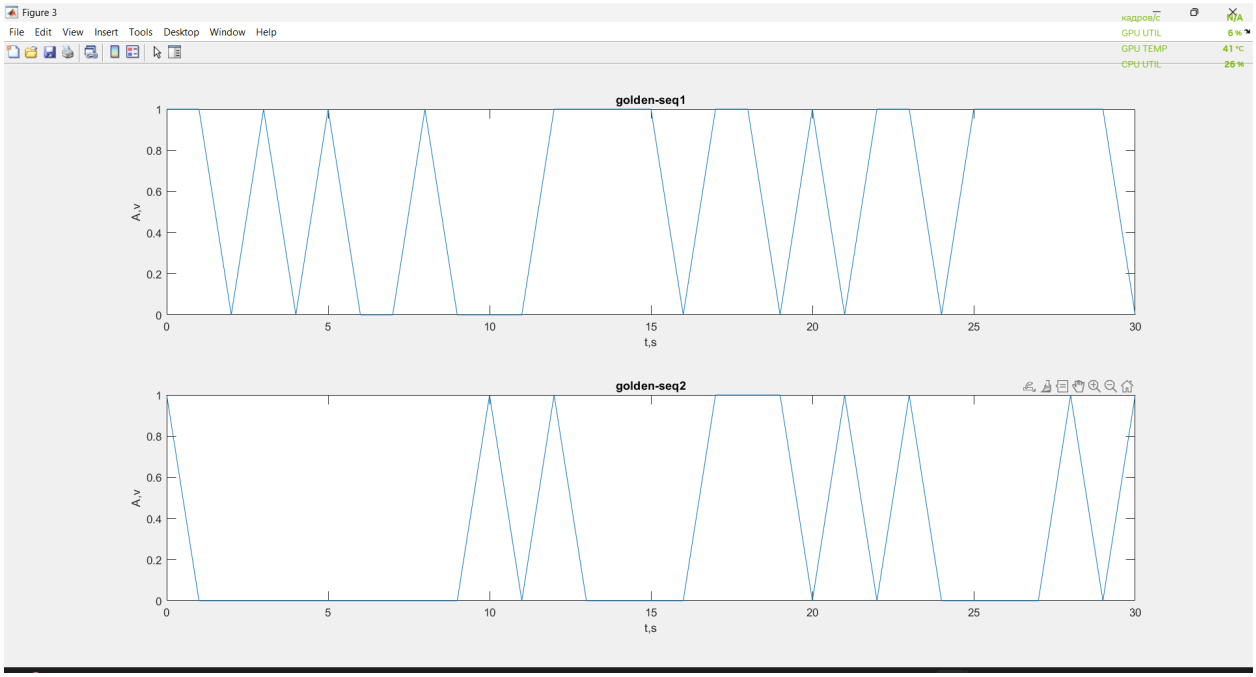

Рисунок 10 — Визуализация последовательностей Голда

Вычислим автокорреляцию для первой последовательности Голда и кросс-корреляцию между двумя последовательностями Голда. Построим графики зависимости корреляции от лага:

```
%% CrossCorr
[c,lags] = xcorr(golden_seq1, golden_seq2, 'normalized');
figure;
subplot(2, 1, 1);
plot(lags,c);
xlabel("lags");
ylabel("Corr");
title("Corr bw golden-seq1&golden-seq2");
grid on;

%%AutoCorr
[c,lags] = xcorr(golden_seq1, 'normalized');
subplot(2, 1, 2);
plot(lags,c);
xlabel("lags");
ylabel("Corr");
title("golden-seq1 autocorr");
grid on;
```

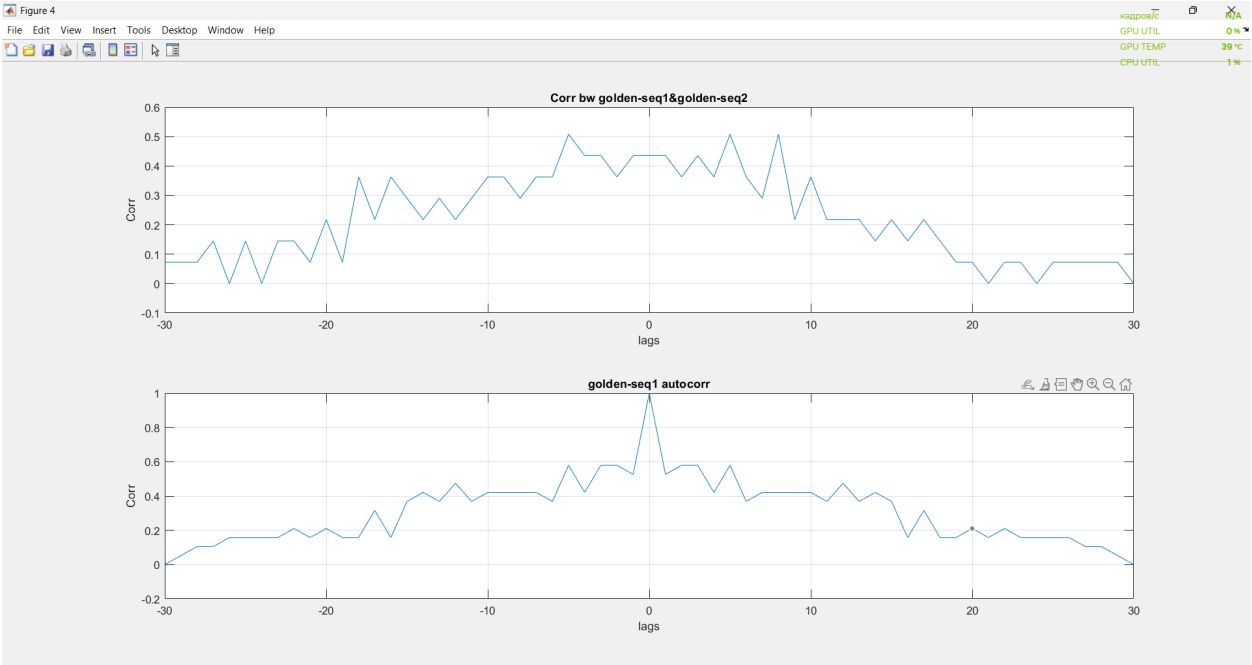

Рисунок 11 — Визуализация автокорреляционной и кросс-корреляционной функциии

Можем заметить, что на графике кросс-корреляции не возникает пика корреляции, т.е две последовательности Голда между собой перепутать не получится. На графике автокорреляции видим один единственный корреляционный пик в точке с лагом равным нулю, т.е при искажении/сдвиге последовательности приемник не допустит ошибки при определении корреляционного пика.

2.3 Контрольные вопросы

1. Для чего в мобильных сетях могут использоваться псевдослучайные последовательности?
2. Что значит положительная корреляция сигналов?
3. Что такое корреляционный прием сигналов?
4. Как вычисление корреляционных функций помогает синхронизироваться приемникам и передатчика в сетях мобильной связи?
5. Какими свойствами обладают псевдослучайные битовые последовательности?
6. Какие разновидности PN-последовательностей вам известны?

Ответы:

1. В мобильных сетях псевдослучайные последовательности могут использоваться для оценки BER: передатчик отправляет заранее известную на приемной стороне последовательность, а приемник вычисляет кол-во ошибок в принятой последовательности. Это позволяет узнать, в какой степени сигнал искажает сигнал. Помимо этого псевдослучайные последовательности используются для синхронизации UL/DL каналов между базовой станцией и абонентом. С помощью корреляционного приема абонент может правильно понять, как размечена временная ось базовой станции (в какие моменты она отправляет SIB или в какие моменты она принимает запросы на прохождение процедуры RA). Также такие последовательности используются для расширения спектра, что делает систему более устойчивой к узкополосным помехам.
2. Положительная корреляция сигналов означает, что между сигналами есть явная связь: если растет первый сигнал, то растет и другой (аналогично с убыванием).
3. Корреляционный прием сигналов - способ принятия сигнала, при котором приемная сторона вычисляет корреляцию между принятым сигналом и заранее известной последовательностью с целью найти корреляционный пик, который означает, что из канала прилетела синхропоследовательность.
4. С помощью вычисления корреляционной функции приемник может понять, в какой момент начинается передача полезной информации. Это позволяет правильно интерпретировать принятый сигнал.
5. Сбалансированность: кол-во нулей и единиц одинаковое или отлично на единицу. Цикличность: в каждом фрагменте псевдослучайной битовой последовательности примерно половину составляли циклы длиной 1, одну четверть – длиной 2, одну восьмую – длиной 3 и т.д. Корреляция - корреляция оригинальной битовой последовательности с ее сдвинутой копией должна быть минимальной. Должен иметься один единственный пик корреляции в случае, когда сдвиг равен нулю.
6. М-последовательности, последовательности Голда, коды Баркера, коды Касами, коды Уолша-Адамара.

2.4 Github

Рисунок 12 — Ссылка на github

ВЫВОД

В ходе работы я получил представление о том, какие существуют псевдослучайные двоичные последовательности, какими корреляционными свойствами они обладают и как используются для синхронизации приемников и передатчиков в сетях мобильной связи.