Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

ФГБОУ ВО «СибГУТИ»

Каф ТСиВС

**Курсовая работа**

по дисциплине: «Мультифункциональные беспроводные сети 5G»

на тему: «Имитационная модель канала связи OFDM»

**Выполнил:**

Студент группы ИА-032

\_\_\_\_\_\_\_\_\_\_\_\_\_ / Болотов Е.Д.

**Проверил:**

Старший преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_ / Ахпашев Р.В.

**Работа зачтена:**

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024г.

Новосибирск, 2024 г

## **Практика 0. Генерация исходного сообщения**

**Реализация**

function message = generateMessage(messageLength)

% Алфавит, используемый для генерации сообщения

alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .';

% Проверка корректности заданной длины сообщения

if messageLength < 1

error('Длина сообщения должна быть положительным числом');

end

% Генерация случайного сообщения заданной длины

message = '';

for i = 1:messageLength

% Случайный выбор символа из алфавита

randIndex = randi(length(alphabet));

message = [message alphabet(randIndex)];

end

end

## **Практика 1. Знаковое кодирование**

**Задание**

Реализовать знаковое кодирование и декодирование текстового со-

общения.

**Реализация**

**Функция encodeMessage**

function encodedMessage = encodeMessage(message)

% Алфавит из 64 символов

alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .';

% Проверка длины сообщения

if length(message) < 30 || length(message) > 100

error('Длина сообщения должна быть от 30 до 100 символов');

end

% Кодирование сообщения

encodedMessage = '';

for i = 1:length(message)

charIndex = find(alphabet == message(i)) - 1; % Индекс символа в алфавите

encodedChar = dec2bin(charIndex, 6); % Преобразование индекса в 6-битовое двоичное число

encodedMessage = [encodedMessage encodedChar]; % Добавление закодированного символа к сообщению

end

end

**Функция decodeMessage**

function decodedMessage = decodeMessage(encodedMessage)

% Алфавит из 64 символов

alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 .';

% Декодирование сообщения

decodedMessage = '';

for i = 1:6:length(encodedMessage)

encodedChar = encodedMessage(i:i+5); % Извлечение 6-битового блока

charIndex = bin2dec(encodedChar) + 1; % Преобразование двоичного числа в индекс символа

decodedMessage = [decodedMessage alphabet(charIndex)]; % Добавление символа к декодированному сообщению

end

end

## **Практика 2. Помехоустойчивое кодирование**

**Задание**

Реализовать операцию сверточного кодирования и витерби декоди-

рования битового сообщения.

**Реализация**

**Функция convEncode**

function encodedBits = convEncode(inputBits)

% Определение генераторных полиномов в двоичной форме

g1 = [1 1 1 1 0 0 1]; % 171 oct -> 1111001 bin

g2 = [1 0 1 1 0 1 1]; % 133 oct -> 1011011 bin

% Инициализация регистра сдвига и выходного вектора

register = [0 0 0 0 0 0 0];

encodedBits = [];

for bit = inputBits

% Сдвиг в регистре и добавление входного бита

register = [bit register(1:end-1)];

% Вычисление выходных битов с помощью свертки с генераторными полиномами

outputBit1 = mod(sum(register & g1), 2);

outputBit2 = mod(sum(register & g2), 2);

% Добавление выходных битов к результату

encodedBits = [encodedBits outputBit1 outputBit2];

end

end

**Функция viterbiDecode**

function [yzli, ves, add\_to\_metr, pyt, in\_delay\_pipe] = viterbiDecode()

k = 7;

G1\_oct = 171;

G2\_oct = 133;

% Convert the generator polynomials to binary

G1\_bin = dec2bin(oct2dec(G1\_oct));

G1 = double(G1\_bin - '0');

G2\_bin = dec2bin(oct2dec(G2\_oct));

G2 = double(G2\_bin - '0');

yzli = {};

for i = 0:(2^(k-1)-1)

yzli = [yzli {dec2bin(i, k-1)}];

end

pyt = zeros(2, 2^(k-1));

for i = 1:2^(k-1)

yzel = cell2mat(yzli(i));

yzel1 = yzel(2:k-1);

d = 0;

for j = 1:2^(k-1)

yzel = cell2mat(yzli(j));

yzel2 = yzel(1:k-2);

if strcmp(yzel1, yzel2)

d = d + 1;

pyt(d, i) = j;

end

end

end

ves = cell(2, 2^(k-1));

for i = 1:2^(k-1)

for n = 1:2

nomer\_pred = pyt(n, i);

yzel\_pred = cell2mat(yzli(nomer\_pred));

yzel = cell2mat(yzli(i));

yzel\_mat = double(yzel - '0');

yzel\_mat = [yzel\_mat(1) double(yzel\_pred - '0')];

out\_s1 = mod(sum(G1 .\* yzel\_mat), 2);

out\_s2 = mod(sum(G2 .\* yzel\_mat), 2);

ves{n, i} = [out\_s1, out\_s2];

end

end

add\_to\_metr = [0, ones(1, 2^(k-1) - 1) \* 20];

in\_delay\_pipe = zeros(2, 2^(k-1));

for i = 1:2^(k-1)

yzel = cell2mat(yzli(i));

in\_delay\_pipe(:, i) = double(yzel(1) - '0');

end

end

**Функция viterbiDecode2**

function decodedBits = viterbiDecode2(encodedBits)

% Initialization

[yzli, ves, add\_to\_metr, pyt, in\_delay\_pipe] = viterbiDecode();

numStates = size(pyt, 2);

trellisLength = length(encodedBits) / 2;

pathMetrics = inf(numStates, trellisLength + 1);

pathMetrics(1, 1) = 0; % Assume we start at state 0

survivorPaths = zeros(numStates, trellisLength);

% Iteration over received bits

for i = 1:trellisLength

receivedBits = encodedBits(2\*i-1:2\*i);

for currentState = 1:numStates

% Find previous states that transition to the current state

for prevStateIndex = 1:2

prevState = pyt(prevStateIndex, currentState);

branchMetric = computeBranchMetric(receivedBits, ves{prevStateIndex, currentState});

newPathMetric = pathMetrics(prevState, i) + branchMetric;

% Update path metrics and survivor paths

if newPathMetric < pathMetrics(currentState, i + 1)

pathMetrics(currentState, i + 1) = newPathMetric;

survivorPaths(currentState, i) = prevState;

end

end

end

end

% Traceback from the last state

[~, lastState] = min(pathMetrics(:, end));

decodedBits = zeros(1, trellisLength);

for i = trellisLength:-1:1

prevState = survivorPaths(lastState, i);

decodedBits(i) = in\_delay\_pipe(1, lastState); % Assuming the first row corresponds to the bit

lastState = prevState;

end

% Correct the order of decoded bits (if necessary)

decodedBits = fliplr(decodedBits);

return;

end

function branchMetric = computeBranchMetric(receivedBits, expectedBits)

% Преобразование из ячеек, если необходимо

if iscell(expectedBits)

expectedBits = cell2mat(expectedBits);

end

branchMetric = sum(receivedBits ~= expectedBits);

end

## 

## **Практика 3. Перемежение**

**Задание**

Реализовать операцию прямого и обратного перемежения закоди-

рованного битового сообщения.

**Реализация**

**Функция interleaveMessage**

function [interleavedMessage, permutationVector] = interleaveMessage(originalMessage)

% Получаем длину исходного сообщения

messageLength = length(originalMessage);

% Генерируем вектор перестановок

permutationVector = randperm(messageLength);

% Инициализируем выходной вектор сообщения

interleavedMessage = zeros(1, messageLength);

% Выполняем перестановку

for i = 1:messageLength

interleavedMessage(i) = originalMessage(permutationVector(i));

end

end

**Функция deinterleavedMessage**

function deinterleavedMessage = deinterleaveMessage(interleavedMessage, permutationVector)

% Получаем длину перемеженного сообщения

messageLength = length(interleavedMessage);

% Инициализируем выходной вектор сообщения

deinterleavedMessage = zeros(1, messageLength);

% Выполняем обратную перестановку

for i = 1:messageLength

deinterleavedMessage(permutationVector(i)) = interleavedMessage(i);

end

end

## 

## **Практика 4. QPSK-модуляция**

**Задание**

Реализовать операцию QPSK-модуляции битового сообщения и

QPSK-демодуляции символов модуляции.

**Реализация**

**Функция qpskModulate**

function symbols = qpskModulate(bits)

% Убедимся, что количество бит кратно двум

if mod(length(bits), 2) ~= 0

error('Количество бит должно быть кратно двум');

end

% Инициализация выходного вектора комплексных символов

numSymbols = length(bits) / 2;

symbols = zeros(1, numSymbols);

% Модуляция

for i = 1:numSymbols

bitPair = bits(2\*i-1:2\*i);

if isequal(bitPair, [0 0])

phase = pi/4;

elseif isequal(bitPair, [0 1])

phase = 3\*pi/4;

elseif isequal(bitPair, [1 1])

phase = 5\*pi/4;

else % [1 0]

phase = 7\*pi/4;

end

symbols(i) = exp(1j\*phase); % комплексный символ

end

end

**Функция qpskDemodulate**

function bits = qpskDemodulate(symbols)

numBits = length(symbols) \* 2;

bits = zeros(1, numBits);

for i = 1:length(symbols)

phase = angle(symbols(i));

if phase < 0

phase = phase + 2\*pi; % Нормализация отрицательных фаз

end

if phase < pi/2

bits(2\*i-1:2\*i) = [0 0];

elseif phase < pi

bits(2\*i-1:2\*i) = [0 1];

elseif phase < 3\*pi/2

bits(2\*i-1:2\*i) = [1 1];

else

bits(2\*i-1:2\*i) = [1 0];

end

end

end

## **Практика 5. OFDM-модуляция**

**Задание**

Реализовать операцию QPSK-модуляции битового сообщения и

QPSK-демодуляции символов модуляции.

**Реализация**

function [ofdmSignal, parameters] = ofdmModulate(complexSymbols, deltaRS, T)

% Параметры

N = length(complexSymbols); % Общее число поднесущих

parameters.N = N;

parameters.deltaRS = deltaRS; % Шаг опорных поднесущих

parameters.T = T; % Размер циклического префикса

% Генерация индексов для опорных символов и данных

parameters.refIndices = 1:deltaRS:N; % Опорные символы через каждые deltaRS поднесущих

parameters.dataIndices = setdiff(1:N, parameters.refIndices); % Данные на остальных поднесущих

% Инициализация значениями опорных символов (здесь используются простые числа в качестве примера)

Rtx = ones(1, length(parameters.refIndices)); % Реальные значения должны быть определены спецификацией

parameters.Rtx = Rtx;

% 1. Применение IFFT к комплексным символам для перехода из частотной области во временную

ifftSignal = ifft(complexSymbols, N);

% Убедимся, что ifftSignal является вектором-строкой для упрощения операций

ifftSignal = ifftSignal(:).'; % Преобразование в строку, если это не так

% 2. Добавление циклического префикса

cyclicPrefix = ifftSignal(end-T+1:end); % Получаем последние T элементов как префикс

ofdmSignal = [cyclicPrefix, ifftSignal]; % Добавляем префикс к сигналу

end

## **Практика 6. Модель канала передачи**

**Задание**

Реализовать операцию QPSK-модуляции битового сообщения и

QPSK-демодуляции символов модуляции.

**Реализация**

function receivedSignal = multipathChannel(inputSignal, N\_B, N\_0, B, f\_0)

c = 3e8; % Скорость света, м/с

T\_0 = 1/B; % Длительность дискретного отсчета, с

L = length(inputSignal); % Длина сигнала

% Инициализация результирующего сигнала

receivedSignal = zeros(1, L);

% Параметры многолучевости

D = randi([10, 500], 1, N\_B); % Расстояние для каждого луча, м

G = c ./ (4\*pi\*D\*f\_0); % Коэффициенты ослабления для каждого луча

% Расчет задержек

D\_min = min(D); % Минимальное расстояние

tau = round((D - D\_min) / (c\*T\_0)); % Задержки в отсчетах

% Генерация лучей

for i = 1:N\_B

delayedSignal = [zeros(1, tau(i)), inputSignal(1:end-tau(i))]; % Добавление задержки

receivedSignal = receivedSignal + G(i) \* delayedSignal; % Добавление ослабленного сигнала

end

% Добавление АБГШ

noisePowerLinear = 10^(N\_0/10) / 1000; % Перевод мощности шума из дБм в Вт

noise = sqrt(noisePowerLinear/2) \* (randn(1, L) + 1i\*randn(1, L)); % Генерация шума

receivedSignal = receivedSignal + noise; % Сложение сигнала с шумом

% Обрезка сигнала до исходной длины, если требуется

receivedSignal = receivedSignal(1:L);

end

## **Практика 7. Эквалайзирование, OFDM демодуляция**

**Задание**

Реализовать операцию QPSK-модуляции битового сообщения и

QPSK-демодуляции символов модуляции.

**Реализация**

function dataSymbols = ofdmDemodulate(receivedSignal, T\_CP, refIndices, Rtx, dataIndices, N)

% Удаление циклического префикса

signalWithoutCP = receivedSignal(T\_CP+1:end);

% Выполнение ДПФ для перевода сигнала в частотную область

fftSignal = fft(signalWithoutCP);

% Выделение опорных сигналов

Rrx = fftSignal(refIndices);

% Оценка АЧХ канала

H = Rrx ./ Rtx;

% Интерполяция АЧХ на весь спектр

x = find(~isnan(H)); % Индексы известных значений H

HEQ = interp1(x, H(x), 1:N, 'linear', 'extrap');

% Эквалайзирование: компенсация искажений канала

CEQ = fftSignal ./ HEQ;

% Извлечение данных с поднесущих, предназначенных для передачи данных

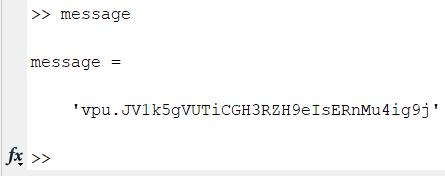
dataSymbols = CEQ(dataIndices);

end

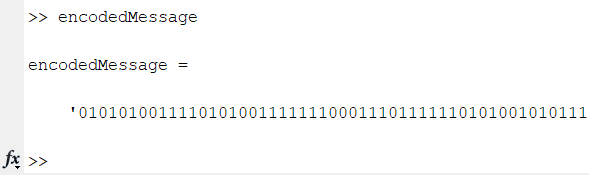
## **Практика 8. Сравнение сообщений. Результаты**

В ходе серии практических работ был выполнен полный цикл передачи данных, включающий в себя генерацию исходного сообщения, его кодирование, модуляцию, передачу через канал с искажениями и последующее декодирование и демодуляцию. Это позволило нам на практике ознакомиться с основными этапами и проблемами цифровой передачи данных, а также с методами их решения.

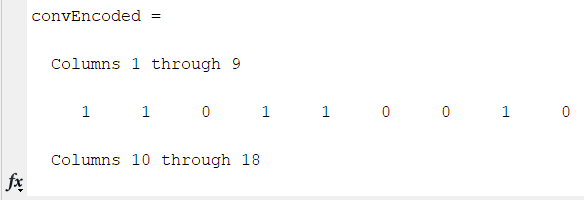
На первом этапе было сгенерировано исходное сообщение.



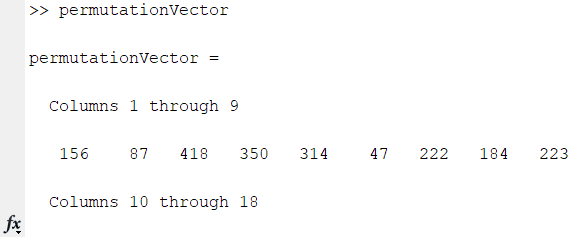
Затем это сообщение подверглось знаковому кодированию для представления в бинарной форме.



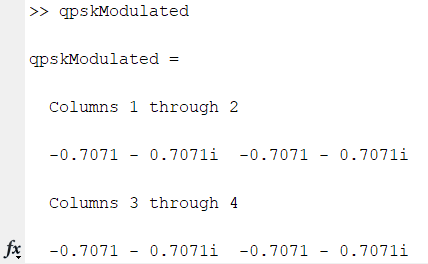
Далее, для повышения помехоустойчивости, к сообщению было применено сверточное кодирование.

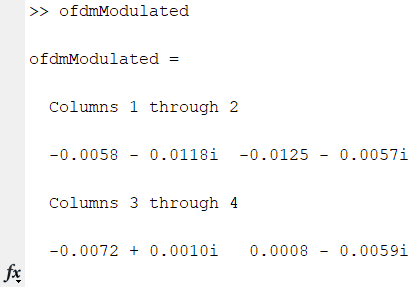


Чтобы минимизировать влияние длительных помех, использовалось перемежение бит. В процессе был создан вектор permutationVector, который нумерует биты при перемежении.

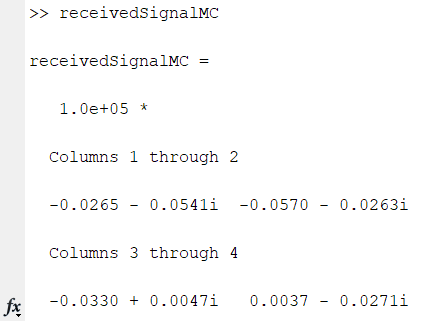


Для модуляции исходного сообщения применялись QPSK и OFDM, что позволило эффективно использовать радиоканал и снизить влияние многолучевого распространения.

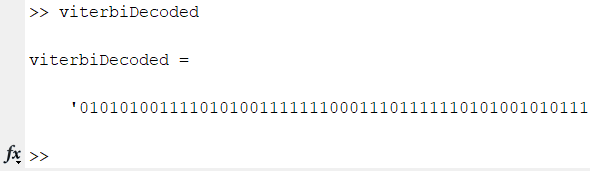


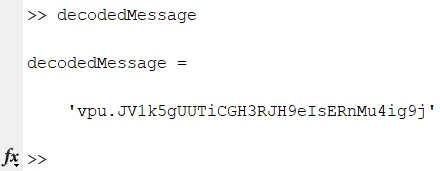


Модель канала передачи включала в себя искажения и шум, имитирующие реальные условия передачи данных.

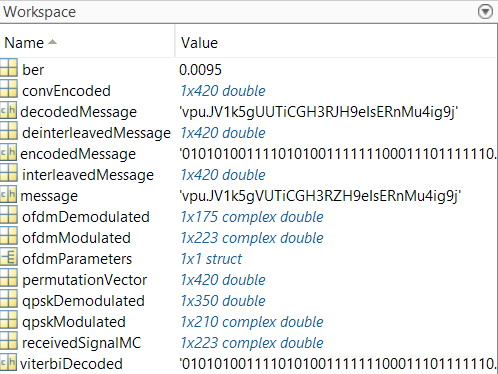


На приемной стороне проводилось эквалайзирование и демодуляция сигнала для восстановления исходного сообщения. В итоге получились декодированные биты и принятое сообщение.





Переменные, которые получились в ходе работы:



Для оценки качества передачи данных между отправленным и принятым сообщениями была рассчитана битовая ошибка (BER).

function ber = calculate\_BER(original\_bits, decoded\_bits)

% Преобразование строк в числовые массивы, если это строки

if ischar(original\_bits) || isstring(original\_bits)

original\_bits = double(original\_bits) - double('0');

end

if ischar(decoded\_bits) || isstring(decoded\_bits)

decoded\_bits = double(decoded\_bits) - double('0');

end

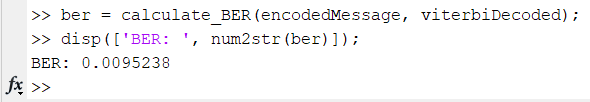
% Расчет количества несовпадающих битов

error\_count = sum(original\_bits ~= decoded\_bits);

% Расчет BER

ber = error\_count / length(original\_bits);

end



Расчет BER показал, что в процессе передачи и последующего декодирования произошло небольшое количество ошибок, что свидетельствует о высокой эффективности использованных методов коррекции и восстановления данных. Это подчеркивает важность комплексного подхода к передаче данных, включая кодирование, модуляцию и эквалайзирование, для обеспечения надежности и эффективности в условиях помех.

**Вывод**

Таким образом, проделанная работа демонстрирует, как с помощью различных технологий можно значительно повысить качество и надежность передачи данных в цифровых коммуникационных системах. Результаты подтверждают эффективность применения помехоустойчивого кодирования и современных методов модуляции для борьбы с искажениями в канале передачи. Расчет BER служит важным инструментом для оценки качества системы связи и позволяет выявить потенциальные направления для дальнейшего улучшения.