Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Телекоммуникационные технологии

Лабораторная работа №7:
Помехоустойчивое кодирование

Работу выполнил:
Сергеев А.А.
Группа: 33531/2
Преподаватель:
Богач Н.В.

Санкт-Петербург
2019

Содержание

1 Цель **2**

2 Постановка задачи **2**

3 Теоретический раздел **2**
3.1 Циклические коды . . . 2
3.2 Коды БЧХ . . . 2
3.3 Коды Хэмминга . . . 2
3.4 Коды Рида-Соломона . . . 2

4 Ход работы **3**
4.1 Код Хэмминга . . . 3
4.1.1 Decode/Encode . . . 3
4.1.2 Кодирование с помощью проверочной и генераторной матриц . . . 3
4.2 Циклический код . . . 4
4.3 Код БЧХ . . . 4

5 Вывод **5**

1 Цель

Изучение методов помехоустойчивого кодирования и сравнение их свойств.

2 Постановка задачи

1. Провести кодирование/декодирование сигнала, полученного с помощью функции *randerr* кодом Хэмминга 2-мя способами: с помощью встроенных функций *encode*/*decode*, а также через создание проверочной и генераторной матриц и вычисление синдрома. Оценить корректирующую способность кода.

2. Выполнить кодирование/декодирование циклическим кодом, кодом БЧХ, кодом Рида-Соломона. Оценить корректирующую способность кода.

3 Теоретический раздел

Функция encode/decode осуществляют кодирование и декодирование соответственно сообщения с использованием блочного кода. Тип используемого кода задаётся в числе параметров функции. Линейный блочный код в общем случае описывается порождающей матрицей (generator matrix). Кодирование блока (вектора) производится путём его умножения на порождающую матрицу. При контроле ошибок на приёмной стороне используется проверочная матрица кода (parity-check matrix). Преобразование порождающей матрицы в проверочную и обратно осуществляется функцией *gen*2*par*. Если умножение кодированного блока на проверочную матрицу не даёт нулевого вектора, то полученный результат (его называют синдром – syndrome) позволяет определить, какие именно символы были искажены в процессе передачи. Для двоичного кода это позволяет исправить ошибки. Декодирование линейного блочного кода, таким образом, можно осуществить с помощью таблицы, в которой для каждого значения синдрома указан соответствующий вектор ошибок. Создать такую таблицу на основании проверочной матрицы кода позволяет функция *syndtable*. Функция $gfweight$ позволяет определить кодовое расстояние для линейного блочного кода по его порождающей или проверочной матрице.

3.1 Циклические коды

Циклические коды – это подкласс линейных кодов, обладающие тем свойством, что циклическая перестановка символов в кодированном блоке даёт другое возможное кодовое слово того же рода. Для работы с циклическими кодами в пакете *Communications* есть две функции. Задав число символов в кодируемом и закодированном блоках, с помощью функции *cyclpoly* можно получить порождающий полином циклического кода. Далее, использовав этот полином в качестве одного из параметров функции *cyclgen*, можно получить порождающую и проверочную матрицы для данного кода.

3.2 Коды БЧХ

Коды БЧХ являются одним из подклассов циклических блочных кодов. Для работы с ними функции высокого уровня вызывают специализированные функции *bchenc* (кодирование) и *bchdec* (декодирование). Кроме того, функция *bchgenpoly* позволяет рассчитывать параметры или порождающий полином для двоичных кодов БЧХ.

3.3 Коды Хэмминга

Коды Хэмминга являются одним из подклассов циклических блочных кодов. Порождающий полином для кодов Хэмминга неприводим и примитивен, а длина кодированного блока равна $2m-1$. Порождающая и проверочная матрицы для кодов Хэмминга генерируются функцией *hammgen*.

3.4 Коды Рида-Соломона

Коды Рида-Соломона являются одним из подклассов циклических блочных кодов. Это единственные поддерживаемые пакетом *Communications* недвоичные коды. Для работы с кодами Рида-Соломона функции высокого уровня вызывают специализированные функции $rsenc$(кодирование) и $rsdec$(декодирование). Кроме того, функции $rsenc$ и $rsdec$ осуществляют кодирование и декодирование текстового файла. Функция $rsgenpoly$ генерирует порождающие полиномы для кодов Рида-Соломона.

4 Ход работы

4.1 Код Хэмминга

4.1.1 Decode/Encode

Генерируем сообщение функцией *randerr* длиной 11, проводим его кодирование и декодирование с использованием функций *encode* и *decode*:

```
function dec_enc(is_two)
    global msg;
    % using encode/decode
    code = encode(msg, 7, 4)
    code_error = code;
    code_error(3) = not(code_error(3))
    if is_two
        code_error(4) = not(code_error(4))
    end
    dec = decode(code_error, 7, 4)
    if dec == msg
        disp('SUCCESS');
    else disp('ERROR');
    end
end
```

Сгенерированное сообщение: 0010.
Закодированное сообщение: 1110010.
Закодированное сообщение с 1 ошибкой: 1100010.
Декодированное сообщение: 0010.
Как видно, изначально сгенерированное сообщение совпадает с декодированным.
Теперь допустим в закодированном сообщении две ошибки и попробуем провести декодирование:

Сгенерированное сообщение: 0010.
Закодированное сообщение: 1110010.
Закодированное сообщение с 2мя ошибками: 1101010.
Декодированное сообщение: 1000.
ERROR

4.1.2 Кодирование с помощью проверочной и генераторной матриц

Произведем кодирование/декодирование сигнала кодом Хэмминга с помощью проверочной и генераторной матриц и вычислим синдром.
При умножении исходного сообщения на генераторную матрицу в ее конечной части.
сохраняется исходная посылка, т.к. соответствующий блок генераторной матрицы представляет собой единичную матрицу. Оставшуюся часть формирует контрольные биты.
Формирование синдрома происходит с помощью домножения на проверочную матрицу.
Синдром указывает на ошибочный бит в посылке. Далее, он исправляется.

```
function use_matrix
    global msg;

    [h, g, n, k] = hammgen(3);
    m = msg * g
    m = rem (m, ones(1, n) .* 2)
    m(2) = not(m(2))
    synd = m * h';
    synd = rem(synd , ones(1, n - k) .* 2)
    stbl = syndtable(h);
    syndr_de = bi2de(synd, 'right-msb')
    z = stbl(syndr_de + 1, :)
    res = rem(z + m, 2)
end
```

msg = 0010.
m = 1110010.
Допускаем ошибку:
m =1010010.
synd = 010.

В переводе в десятиричную систему счисления получаем 2.
z = 0, 1, 0, 0, 0, 0, 0.
Исправленное сообщение без ошибки: 1110010.
Корректирующая способность кода равна 1.

4.2 Циклический код

Производим кодирование и декодирование сообщения:

```
function cycle_code
    global msg;
    % cycle code %
    n = 7; k = 4;
    pol = cyclpoly (n, k, 'max')
    [h, g] = cyclgen(7, pol);
    code = msg * g;
    code = rem(code, ones(1, n) .* 2)

    code(2) = not(code(2))

    synd = code * h'

    synd = rem(synd , ones(1, n - k) .* 2)

    stbl = syndtable(h)

    syndr_de = bi2de(synd, 'right-msb')
    z = stbl(syndr_de + 1, :)
    rez = rem(z + code, 2)
end
```

Сгенерированное сообщение: 0010.
pol = 1101
code = 1110010
code = 1010010
synd = 212
synd = 010
stbl =
0 0 0 0 0 0 0
0 0 1 0 0 0 0
0 1 0 0 0 0 0
0 0 0 0 1 0 0
1 0 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 1 0 0 0
0 0 0 0 0 1 0

syndrde2 = 2
z = 0100000
rez = 1110010
Был построен полином циклического кода $x^3 + x^2 + 1$ который использовался в качестве параметра функции cyclgen. Получены порождающая и проверочная матрицы.Корректирующая способность кода равна 1.

4.3 Код БЧХ

```
function bch
    global msg;
    % BCH codes %
    code_p = comm.BCHEncoder(7, 4);
    dec_p = comm.BCHDecoder(7, 4);
    tmp = msg';
    code = step(code_p, tmp(:))'

    code(2) = not(code(2))
    decode = step(dec_p, code')'
end
```

Сгенерированное сообщение: 0010.
Закодированное сообщение (без ошибки): 0010110.
Закодированное сообщение (с ошибкой): 0110110.
Декодированное сообщение: 0010.
Нетрудно убедиться, что исходное и полученное сообщения идентичны, что свидетельствует об успешном исправлении допущенной ошибки.
Корректирующая способность кода равна 1.

5 Вывод

В данной лабораторной работе проведены кодирования и декодирования посылок кодами Хэмминга, циклическим и БЧХ. Выбирать метод кодирования стоит в зависимости от типа посылки и зашумленности канала. Код Хэмминга достаточно простой в использовании и не требует больших мощностей. Но существенным недостаткой является то, что все рассмотренные в работе коды позволяют исправить только одну ошибку в бинарном коде.