Лабораторная работа №7

Никитин А.Е.

15 мая 2017 г.

1 Помехоустойчивое кодирование

1.1 Цель работы

Изучение методов помехоустойчивого кодирования, а также сравнение их свойств.

1.2 Постановка задачи

1. Провести кодирование/декодирование сигнала, полученного с помощью функции randerr кодом Хэмминга двумя 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, БЧХ-кодом, кодом Рида-Соломона. Оценить корректирующую способность кода.

1.3 Справочные материалы

- Солопченко Г.Н. Теория информации. СПб.: Изд-во Политехнического унив., 2010. С. 36-55;

- Темников Ф. Е., Афонин В. А., Дмитриев В. И. Теоретические основы информационной техники. М.: Энергия, 1971. С. 96-177;

- Кузьмин И.В., Кедрус В.А. Основы теории информации и кодирования. К.: Вища шк. Головное изд-во, 1986. С. 79-104.

1.4 Теоретические положения

Битовые ошибки в каналах связи нельзя исключить полностью, даже если выбранный способ кодирования дискретного сигнала код обеспечивает хорошую степень синхронизации и высокий уровень отношения сигнала к шуму. Поэтому при передаче дискретной информации применяются специальные коды, которые позволяют обнаруживать (а иногда даже исправлять) битовые ошибки.

Преимуществом цифровых методов записи, воспроизведения и передачи аналоговой информации является возможность контроля достоверности считанных с носителя или полученных по линии связи данных.

Для этого можно применять те же методы, что и в случае компьютерных данных, – вычисление контрольной суммы, повторная передача искаженных кадров, применение самокорректирующихся кодов.

Распознавание и коррекцию искаженных данных сложно осуществить средствами физического уровня, поэтому чаще всего эту работу берут на себя протоколы, лежащие выше: канальный, сетевой, транспортный или прикладной. В то же время распознавание ошибок на физическом уровне экономит время, так как приемник не ждет полного помещения кадра в буфер, а отбраковывает его сразу при распознавании ошибочных битов внутри кадра.

Есть два подхода работы протоколов. Протоколы, реализующие первый принцип, обеспечивают надежность за счет повторной передачи искаженных или потерянных пакетов. Такие протоколы основаны на том, что приемник в состоянии распознать факт искажения информации в принятом кадре. Еще одним, более эффективным подходом, чем повторная передача пакетов, является использование самокорректирующихся кодов, которые позволяют не только обнаруживать, но и исправлять ошибки в принятом кадре.

1.4.1 Классификация помехоустойчивого кодирования

Помехоустойчивость кодирования обеспечивается за счет введения *избыточности* в кодовые комбинации. Избыточность позволяет наложить на передаваемые последовательности символов дополнительные условия, проверка которых на приемной стороне дает возможность обнаружить и исправить ошибки, возникающих в результате влияния помех.

Все помехоустойчивые коды можно разделить на два основных класса: *блочные* и *непрерывные* (Рис. **??**).

В блочных кодах каждому сообщения (или его элементу) сопоставляется кодовая комбинация (блок) из определенного количества сигналов. Блоки кодируются и декодируются отдельно друг от друга.

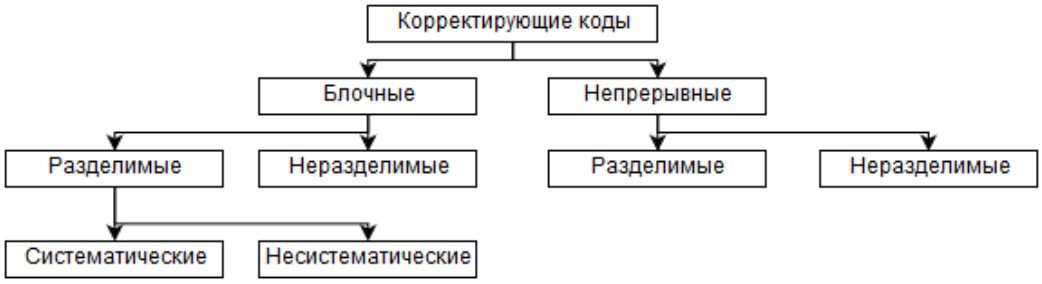

Рис. 1: Классификация помехоустойчивых кодов.

Блочные коды могут быть равномерными, когда длина n кодовых комбинаций постоянна, или неравномерными, когда n постоянно. Неравномерные помехоустойчивые коды не получили практического применения из-за сложности их технической реализации.

В непрерывных кода введение избыточности в последовательность входных символов осуществляется без разбивки ее на отдельные блоки. Процессы кодирования и декодирования в непрерывных кодах имеют также непрерывный характер.

Как блочные, так и непрерывные коды в зависимости от методов внесения избыточности подразделяются на *разделимые* и *неразделимые*. В разделимых кодах четко разграничена роль отдельных символов. Одни символы являются информационными, другие – проверочными и служат для обнаружения и исправления ошибок.

В неразделимых кодах разделение информационной и проверочной части невозможно, что затрудняет декодирование, особенно при необходимости исправления ошибок. К таким кодам относятся коды с постоянным весом и некоторые другие.

Разделимые коды делятся на *систематические* и *несистематические*. Систематические коды характеризуются тем, что сумма по модулю 2 двух разрешенных комбинаций дает комбинацию того же кода. Процессы кодирования и декодирования в систематических кодах сводятся к подсчету сумм по модулю 2 информационных и проверочных символов в различных сочетаниях.

Несистематические коды, к числу которых относятся коды с суммированием, указанным выше свойством не обладают. Метод построения таких кодов состоит в том, что проверочные символы определяются как результат суммирования символов, входящих в кодовую комбинацию или ее часть.

Разновидностью систематических кодов являются циклические коды, характеризующиеся тем, что циклическая перестановка всех символов одной комбинации дает другую комбинацию, принадлежащую этому же коду.

Принципиально все перечисленные коды могут быть использованы как для обнаружения, так и для исправления ошибок. Однако отмеченные выше удобства построения кодирующих и декодирующих устройств определили преимущественное применение лишь некоторых из них.

1.4.2 Связь корректирующей способности кода с кодовым расстоянием

Блоковые (равномерные) коды характеризуются так называемым *минимальным кодовым расстоянием*. Количеством единиц в кодовой комбинации называют *весом кодовой комбинации* и обозначают w. Например, кодовая комбинация 100101100 характеризуется длиной $n = 9$ и весом $w = 4$.

Степень отличия любых двух кодовых комбинация данного кода характеризуется так называемым *расстоянием между кодами* d. Для двоичного кода под данным термином понимается *расстояние Хэмминга*. Оно выражается числом позиций или символов, в которых комбинации отличаются одна от другой, и определяется как вес суммы по модулю два этих кодовых комбинация. Например, для определения расстояния между комбинациями 100101100 и 110110101 необходимо просуммировать их по модулю два: $1\dot{0}01\dot{0}\dot{1}10\dot{0} \oplus 1\dot{1}01\dot{1}\dot{0}10\dot{1} = 010011001$ (точками выделены отличающиеся биты).

Полученная в результате суммирования новая кодовая комбинация характеризуется весом $w = 4$. Следовательно, расстояние Хэмминга $d = 4$. При этом в качестве *минимального кодового расстояния* $d_{\text{мин}}$ выбирается наименьшее из всех расстояний по Хэммингу для любых пар различных кодовых слов, образующих код. $d_{\text{мин}}$ – очень важная характеристика кода, ибо именно она характеризует его *корректирующую способность*:

$t = \left\lfloor \dfrac{d_{\text{мин}} - 1}{2} \right\rfloor$, здесь угловые скобки обозначают округление вниз.

Код, образованный различными $N = 2^n$ кодовыми словами длиной n, имеет $d = 1$. Этот код не обладает свойством избыточности и не предоставляет возможности обнаруживать или исправлять ошибки, потому что ошибка при передаче одного символа переводит одно слово этого кода в другое кодовое слово. Это наглядно показано на Рис. **??**, а), где жирными точками отмечены кодовые слова, отстоящие друг от друга на расстоянии, равном 1. Кодовые слова, расстояние между которыми равно 2, показаны жирными точками на Рис. **??**, б). Это разрешенные кодовые слова. Остальные точки изображают неразрешенные кодовые слова, отсутствующие в коде.

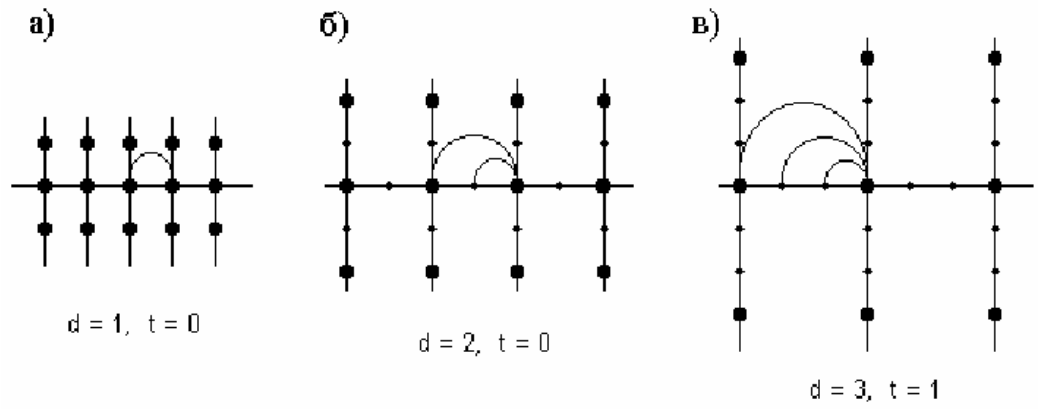

Рис. 2: Кодовое расстояние и ошибки в передаче кодовых слов.

Из Рис. ??, б) видно, что код с расстоянием 2 дает возможность обнаружить одну ошибку, поскольку одна ошибка приводит любое разрешенное кодовое слово к неразрешенному. Ошибки в двух позициях переводят кодовое слово в другое слово из этого же кода, и эти ошибки обнаружены быть не могут. На Рис. ??, в) изображены разрешенные кодовые слова с расстоянием между ними, равном 3. Из рисунка видно, что такой код допускает исправление одной ошибки и обнаружение двух ошибок.

В общем случае при необходимости *обнаружить* ошибки кратности r (т.е. количества искаженных символов в в кодовой комбинации) минимальное хэмингово расстояние между разрешенными кодовыми комбинациями должно быть по крайней мере на единцу больше r, т.е $d_{\text{мин}} \geq r+1$.

Для *исправления* ошибок кратности s минимальное хэмингово расстояние между разрешенными комбинациями должно удовлетворять соотношению $d_{\text{мин}} \geq 2s+1$. Т.е. для исправления одиночно ошибки каждой разрешенной кодовой комбинации необходимо сопоставить свое подмножество запрещенных кодовых комбинаций, состоящее из $d_{\text{мин}} = 3$ кодов. Для $n = 3$ за разрешенные комбинации можно, например, принять 000 и 111. Тогда для 000 запрещенными будут 001, 010 и 100, а для 111 – 110, 101 и 011.

В реальных каналах связи длительность импульсов помехи может превышать длительность символа. При этом одновременно искажаются несколько расположенных рядом символов комбинации. Ошибки такого рода получили название пачек ошибок или пакетов ошибок. Длиной пачки ошибок называется число следующих друг за другом символов, левее и правее которых в кодовой комбинации искаженных символов не содержится.

1.4.3 Показатели качества корректирующего кода

Любой корректирующий код характеризуется рядом показателей:

- длиной n;
- основанием m;
- количеством информационных символов i;
- количеством корректирующих символов k;
- полным числом всех возможных кодовых комбинаций $N = m^n$;
- числом разрешенных кодовых комбинаций (мощностью кода) $N_{\text{р}}$;
- весом кодовой комбинации w;
- кодовым расстоянием d;
- наименьшим расстоянием между разрешенными кодовыми комбинациями $d_{\text{мин}}$;
- и пр.

Однако одной из основных характеристик корректирующего кода является избыточность кода. Для того, чтобы код приобрел способность к обнаружению и коррекции ошибок, необходимо отказаться от его безызбыточности. Для этого и разделяют всё множество возможных комбинаций двоичных символов на два подмножества, как уже говорилось ранее: *допустимых* кодовых слов и *недопустимых*. Разбиение осуществляется таким образом, чтобы увеличить минимальное кодовое расстояние между допустимыми словами. В этом случае любая однократная ошибка превращает допустимое кодовое слово в недопустимое, что позволяет ее обнаружить.

Естественно, что введение дополнительных контрольных разрядов увеличивает затраты на хранение или передачу кодированной информации. При этом фактический объем полезной информации остается неизменным. В этом случае можно говорить об избыточности помехоустойчивого кода, которую формально можно определить как отношение числа контрольных разрядов к общему числу разрядов кодового слова:

$$R_n = k/n * 100\%,$$

где k – число контрольных разрядов, а n – общее число бит комбинации.

Пожалуй, основным показателем качества корректирующего кода является его способность обеспечить правильный прием кодовых комбинаций при наличии искажений под воздействием помех, т.е. *помехоустойчивость кода*.

Количественная оценка помехоустойчивости кода может быть осуществлена по-разному. Можно использовать вероятность правильного приема кодовых комбинаций $P_{\text{пр}} = 1 - P_{\text{ош}}$, где $P_{\text{ош}}$ – вероятность ошибочного приема кодовых комбинаций.

Если код не обладает корректирующими свойствами, то вероятность ошибочного према $P_{\text{ош}}$ будет равна вероятности искажения кодовых комбинаций P_k. Для корректирующего кода $P_{\text{ош}} < P_k$. В реальных условиях $P_{\text{ош}} \leq 1$, поэтому более удобным критерием оценки помехоустойчивости кода является логарифмическая величина

$$S_k = \lg \frac{1}{P_{\text{ош}}} = \lg \frac{1}{1 - P_{\text{пр}}}.$$

1.4.4 Циклические коды

Групповым кодом называют такой код, множество кодовых комбинаций которого образует группу относительно операции сложения по модулю 2. Любой групповой код может быть записан в виде матрицы, включающей i линейно независимых строк по n символов. Среди всего многообразия таких кодов можно выделить коды, у которых строки образующих матриц связаны дополнительным условием цикличности.

Все строк образующей матрицы такого кода могут быть получены сдвигом одной комбинации, называемой *образующей* для данного кода. Коды, удовлетворяющие этому условию, получили название *циклических кодов*.

Сдвиг осуществляется справа налево с переносом выдвигаемого символа в освобождающуюся позицию. Запишем совокупность кодовых комбинаций, получающихся циклических сдвигом одной n-разрядной комбинации, например шестиразрядной 010001:

$$010001, 100010, 000101, 001010, 010100, 101000.$$

Любые $i \leq n$ комбинаций этого множества могут составить образующую матрицу кода.

Число возможных циклических (n, i) кодов значительно меньше числа различных групповых (n, i) кодов.

Циклические коды удобно рассматривать, представляя комбинации двоичного кода в виде полинома $n-1$-й степени от фиктивной перемен-

ной x:

$$G(x) = a_{n-1}x^{n-1} + a_{n-2}^{n-2} + \cdots + a_1 x + a_0,$$

где a_j – цифры комбинации (0 и 1). Показатели степени соответствуют номерам разрядов.

Многочлен называется нормированным, если коэффициент при старшей степени равен единице.

Теперь действия над кодовыми комбинациями сводятся к действиям над многочленами.

Вышеупомянутый циклический сдвиг некоторого образующего многочлена степени $n-k$ соответствует простому умножению на x. Умножив, например, многочлен $x^3 + x^2 + 1$, соответствующий комбинации 0001101, на x, получим многочлен x^4+x^3+x, соответствующий 0011010. Если при этом образуется полином степени n, то необходимо заменить x^n на 1 – это аналогично сложению по модулю 2 с многочленом $x^n + 1$.

Можно убедиться, что кодовая комбинация, получающаяся сложением этих двух комбинаций, также будет соответствовать результату умножения многочлена x^3+x^2+1 на полином $x+1$, если приведение подобных осуществлять по модулю 2. Действительно,

$$\begin{array}{r} 0001101 \\ \oplus\ 0011010 \\ \hline 0010111, \end{array} \qquad \begin{array}{r} x^3 + x^2 + 0 + 1 \\ x + 1 \\ \hline x^3 + x^2 + 0 + 1 \\ \oplus\ x^4 + x^3 + 0 + x \\ \hline x^4 + 0 + x^2 + x + 1. \end{array}$$

Отсюда ясно, что при соответствующем выборе образующего многочлена любая разрешенная кодовая комбинация циклического кода может быть получена в результате умножения образующего полинома на некоторый другой полином. Иными словами, любой многочлен циклического кода делится на образующий полином без остатка. Ни один многочлен, соответствующий запрещенной кодовой комбинации, на образующий многочлен без остатка не делится. Это свойство позволяет обнаруживать ошибку. По виду остатка можно определить и вектор ошибки.

Умножение и деление многочленов весьма просто осуществляется на регистрах сдвига с обратными связями, что и явилось причиной широкого применения циклических кодов.

Рассмотрим простейший циклический код, обнаруживающий все одиночные ошибки. Любая принятая по каналу связи кодовая комбинация $h(x)$, возможно содержащая ошибку, может быть представлена в виде

суммы по модулю 2 неискаженной комбинации кода $f(x)$ и вектора (кода) ошибки $\xi(x)$:

$$h(x) = f(x) \oplus \xi(x).$$

При делении $h(x)$ на образующий полином $g(x)$ остаток, указывающий на наличие ошибки, будет обнаружен только в том случае, если многочлен, соответствующий вектору ошибки, не делится на $g(x)$. Вектор одиночной ошибки будет иметь единицу в искаженном разряде и нули во всех остальных. Ему будет соответствовать многочлен $\xi(x) = x^j$. Последний не должен делиться на $g(x)$. Среди неприводимых полиномов, входящих в разложение $x^n + 1$, многочленом наименьше степени, удовлетворяющий указанному условию, является $x + 1$.

Для исправления одиночных ошибок или обнаружения двойных существуют циклические коды Хэмминга с $d_{\text{мин}} = 3$, чьи некоторые основные характеристики приведены в таблице ниже.

Показатель неприводимого многочлена	Образующий многочлен	Число остатков	Длина кода
2	$x^2 + x + 1$	3	3
3	$x^3 + x + 1$	7	7
3	$x^3 + x^2 + 1$	7	7
4	$x^4 + x^3 + 1$	15	15
4	$x^4 + x + 1$	15	15
5	$x^5 + x^2 + 1$	31	31
5	$x^5 + x^3 + 1$	31	31
5	$x^5 + x^2 + x + 1$	31	31

1.4.5 Код Боуза-Чоудхури-Хоквингема

Ранее мы выяснили, что неразложимый многочлен $g(x)$ является порождающим многочленом кода длины n, если он делит двучлен $x^n + 1$ без остатка, то есть если $(x^n + 1) \mod g(x) = 0$. Для этого все корни многочлена $g(x)$ должны быть корнями двучлена $x^n + 1$. И вообще, корни всех неразложимых многочленов, произведение которых есть двучлен $x^n + 1$, также должны быть корнями этого двучлена. Общее количество его корней должно быть равно степени этого двучлена. С другой стороны, поскольку любой многочлен $a(x)$, представляющий собой кодовое слово, должен делиться на порождающий многочлен $g(x)$, то все корни порождающего многочлена должны быть корнями многочлена $a(x)$. Но порождающий многочлен неразложим на сомножители, а это должно означать, что у порождающего многочлена корней нет. С подобным обстоятельством мы сталкивались при решении, например, квадратных уравнений с вещественными коэффициентами, у которых в вещественной

области корней не было. В этих случаях приходилось расширять множество вещественных чисел до множества комплексных чисел и находить корни в этом расширенном множестве. В каком же множестве лежат все корни двучлена $x^n + 1$? Оказывается, что корнями этого двучлена являются элементы $\alpha_j = x^j \mod g(x)$.

Коды Боуза-Чоудхури-Хоквингема (коды БЧХ) являются линейными циклическими кодами и представляют собой обобщение циклических кодов. Эти коды позволяют исправлять многократные ошибки и пачки ошибок. БЧХ-коды задаются корнями порождающих многочленов. В качестве порождающих многочленов в этом случае служат приводимые многочлены g(x)=p(x)q(x), которые суть произведения неприводимых нормированных многочленов.

В БЧХ-коде построение образующего многочлена, в основном, зависит от двух параметров: от длины кодового слова $n = m + i$ и от числа исправляемых ошибок s.

Особенностью кода является, то что для исправления числа ошибок $s \geq 2$ еще недостаточно условия, что между комбинациями кода минимальное кодовое расстояние $d_{\text{мин}} = 2 * s + 1$. Необходимо также, чтобы длина кода n удовлетворяла условию

$$n = 2h - 1, \tag{1}$$

где h – любое целое число.

При этом n всегда будет нечетным числом и принимать значения: 1, 3, 7, 15, 31, 63, 127... и т.д, т.е не все i могут быть заданы пользователем.

Выбранная величина n определяет число контрольных символов k:

$$k \leq h * s \leq [\log_2(n + 1)] * s. \tag{2}$$

При решении задачи выбора допустимого числа информационных символов при заданных корректирующих свойствах удобно пользоваться таблицей, в которой приведены соотношения корректирующих и информационных разрядов для БЧХ кодов.

1.5 Ход работы

1.5.1 Циклические коды

Кодирование данных в MATLAB циклическим кодом осуществляется также функцией **encode**. Для кодирования этой функции нужно передать длину последовательности n, число информационных битов i, указать метод кодирования ($'cyclic'$ в данном случае). Можно указать

порождающий полином. Однако, было решено закодировать данные при помощи средств языка C. Ниже приведен листинг проекта.

В программном пакете MATLAB есть функция **encode**, но в связи с принятым решением производить кодирование с помощью языка C, просто опишем каким образом она работает: в том случае, если **encode** используется для циклического кодирования, сперва определяет, передан ли ей порождающий полином. Если его нет, то вызывается функция **cyclpoly**, генерирующей порождающий многочлен минимальной длины для заданных n и i. Затем с помощью **cyclgen** формируются порождающая матрица G, которая умножается на исходную битовую последовательность со взятием остатка по делению, образуя кодовую последовательность. Это выполняется так же, как в случае с кодом Хэмминга.

Выполнил циклическое кодирование кодом $(7, 4)$ с помощью **encode**:

```
>> msg=[1 0 1 1]; % пол байта
>> code=encode(msg,7,4,'cyclic')
   code =
        0     0     0     1     0     1     1
```

Отчетливо видно, что и на этот раз первые три бита являются контрольными, а оставшиеся – информационные и соответствуют том, что было передано в msg.

Для сравнения выполним циклическое кодирование при помощи следующей программы.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <locale.h>

#define N 15
#define Np 7

// Парсер строки в биты
unsigned long str_to_bits(char* str)
{
    int len = strlen(str);
    unsigned long res = 0;

    for (int i = 0; i < len; i++)
    {
        if (str[i] == '1')
```

```
            res++;
        res <<= 1;
    }
    res >>= 1;

    return res;
}
//*****************************************
//Пересер битов в строки для вывода
char* bits_to_str(unsigned long num, int k)
{
    char* str = new char[k + 1];
    unsigned long mask = (1 << (k - 1));

    for (int i = 0; i < k; i++)
    {
        if (num & mask)
            str[i]='1';
        else
            str[i]='0';
        mask >>= 1;
    }
    str[k]='\0';

    return str;
}
//*****************************************
//Деление полиномов
unsigned long DividePolynoms(unsigned long U, int k,
   unsigned long P, int p)
{
    int mask = (1 << (k + p - 2));

    P <<= (k - 1);

    for (int i = 0; i < k; i++)
    {
        if (U & mask)
            U ^= P;
        mask >>= 1;
        P >>= 1;
    }
```

```
    return U;
}
//*****************************************
//Сдвиг
unsigned long Shift(unsigned long U, int k, char
   direction)
{
    unsigned long mask, shift_bit;

    if (direction == 'l')
    {
        mask = (1 << (k - 1));
        shift_bit = (U & mask) ? 1 : 0;
        mask = ((U & mask) << 1);
        U <<= 1;
        U ^= mask;
    }
    else
    {
        mask = 1;
        shift_bit = ((U & mask) << (k - 1));
        U >>= 1;
    }

    return U + shift_bit;
}
//*****************************************
//Поиск порождающей матрицы
unsigned long* FindGenMatrix(unsigned long P, int k, int
   p)
{
    unsigned long* G = new unsigned long[k];

    for(int i = 0; i < k; i++)
    {
        G[i] = (1 << (k + p - 2 - i));
        G[i] += DividePolynoms(G[i], k, P, p);
    }

    return G;
}
//*****************************************
//Вывод матрицы
```

```
void PrintMatrix(unsigned long * matrix, int m, int n)
{
    for(int i = 0; i < m; i++)
    {
        printf("%s\n", bits_to_str(matrix[i], n));
    }
}
//*****************************************
//Поиск максимальной длинны кода
int FindCodeDistance(unsigned long * G, int k, int p)
{
    unsigned long  mask;
    int d = k + p - 1, min;

    for(int i = 0; i < k; i++)
    {
        min = 0;
        mask = 1;
        for (int j = 0; j < k + p - 1; j++)
        {
            if (G[i] & mask)
                min++;
            mask <<= 1;
        }
        if (d > min)
            d = min;
    }

    return d;
}
//*****************************************
//Поиск веса кода
int FindWeight(unsigned long num, int k)
{
    int w = 0;
    unsigned long mask = 1;

    for (int i = 0; i < k; i++)
    {
        if (num & mask)
            w++;
        mask <<= 1;
    }
```

```
    return w;
}
//*****************************************
//Главная функция
int main(int argc, char* argv[])
{
        setlocale(LC_ALL,"russian");
    int lenght_of_U = 0;
    int polynom_lenght = 0;
    int d = 0;
    char err[N+1];
    char Ustr[N-Np+1];
    char Pstr[Np+1];
    unsigned long U;
    unsigned long P;
    unsigned long R;
    unsigned long E;
    unsigned long *G;
    unsigned long *H;

        char choise = '0';

                        printf("Введите исходное сообщени
   e:\n");
                        scanf("%s", Ustr);
                        lenght_of_U = strlen(Ustr);
                        U = str_to_bits(Ustr);

                        printf("Введите пораждающий полин
   ом:\n");
                        scanf("%s", Pstr);
                        polynom_lenght = strlen(Pstr);
                        P = str_to_bits(Pstr);

                        U <<= (polynom_lenght - 1);
                        R = DividePolynoms(U, lenght_of_U
   , P, polynom_lenght);
                        U += R;
                        printf("CRC: %s\n", bits_to_str(U
   , lenght_of_U + polynom_lenght - 1));

                        G = FindGenMatrix(P, lenght_of_U,
```

```
    polynom_lenght);
                        printf("Пораждающая матрица: \n")
   ;
                        PrintMatrix(G, lenght_of_U,
   lenght_of_U + polynom_lenght - 1);

                        H = FindCheckMatrix(G,
   lenght_of_U, polynom_lenght);
                        printf("Проверочная матрица: \n")
   ;
                        PrintMatrix(H, polynom_lenght -
   1, lenght_of_U + polynom_lenght - 1);

                        d = FindCodeDistance(G,
   lenght_of_U, polynom_lenght);
                        printf("Наименьшее расстояние: %d
   \n", d);
                        printf("Кодирование может обнаруж
   ить ошибки уровня %d, правильный уровень ошибок %d\n",
    d-1, (d-1)/2);

    getch();
    return 0;
}
```

Попробовал получить другую генераторную матрицу, с помощью которой можно осуществить циклическое кодирование $(7, 4)$:

```
>> cyclpoly(7,4,'all')
   ans =
        1     0     1     1
        1     1     0     1
```

Видно, существует два порождающих полинома. Воспользуюсь вторым: $x^3 + x^2 + 1$.

```
>> code=encode(msg,7,4,'cyclic',poly)
   code =
        1     0     0     1     0     1     1
        0     0     0     1     1     0     1
```

В результате получены другие кодовые комбинации тех же сообщений,

обладающие, тем не менее, аналогичными помехоустойчивыми свойствами.

Декодирование выполняется функцией **decode**. Ее работа немногим отличается от предыдущего случая. Она использует **cyclpoly**, если порождающий полином не был передан, для генерации проверочной матрицы H и порождающей G с помощью функции *cyclgen* (теперь берутся оба возвращаемых аргумента). Далее создается таблица декодирования синдрома. Это делает функция **syndtable**, использующая матрицу H. Дальнейшие действия аналогичны тем, что были в случае кодов Хэмминга: вычисляется синдром, определяется позиция ошибки, коррекция декодируемой последовательности и извлечение информационных бит.

Осуществил декодирование двух блоков, используя **decode** и сгенерированный ранее порождающий многочлен:

```
>> rx=decode(code,7,4,'cyclic',poly)
   rx =
        1     0     1     1
        1     1     0     1
>> biterr(msg,rx) % определение числа ошибок
   ans =
        0
```

Полученные декодированные данные полностью совпадают с тем, что было до кодирования.

Попробовал внести в кодированные блоки одиночные ошибки:

```
>> err=randerr(2,7) % 1 ошибка в случайном месте строки
   err =
        0     0     0     1     0     0     0
        0     0     0     0     1     0     0
>> bitxor(err,code) % внесение помехи
   ans =
        1     0     0     0     0     1     1
        0     0     0     1     0     0     1
>> rx=decode(ans,7,4,'cyclic',poly) % декодирование
   rx =
        1     0     1     1
        1     1     0     1
>> biterr(rx,msg) % ошибки?
   ans =
        0
```

Полученные декодированные последовательности бит обработаны пра-

вильно.

Определение корректирующей способности кода возможно с помощью порождающего многочлена и функции **gfweight**:

```
>> wt = gfweight(poly,7) % 7 - длина кодовой комбинации
   wt =
        3
```

Выяснил, что $d_{\text{мин}} = 3$. Выводы, сделанные ранее по кодам Хэмминга, здесь также уместны: корректирующая способность равна одному.

1.5.2 Код Боуза-Чоудхури-Хоквингема

В MATLAB для кодирования кодом БЧХ можно использовать **encode**, однако она считается устаревшей и рекомендуют использовать **bchenc**. Для работы последней функции требуется, чтобы кодируемое сообщение было представлено в поле Галуа. **bchenc** по заданным параметрам n и i генерирует порождающий полином $g(x)$ вызовом функции **bchgenpoly**. По $g(x)$ строится порождающая матрица с помощью **cyclgen** (второй возвращаемый аргумент). Затем происходит кодирование путем умножения сообщения на G. Брать дополнительно результат умножения по модулю 2 не нужно, т.к. эта операция в полях Галуа делается как раз по модулю 2 (в случае, если размерность поля равна двум, т.е. $GF(2)$).

Начал кодирования с формирования сообщения в поле Галуа:

```
>> msg = [1 0 1 1; 0 1 1 1]; % два блока
>> msg = gf(msg) % перевод в поле Галуа
   msg = GF(2) array.
   Array elements =
         1       0       1       1
         0       1       1       1
```

Теперь можно выполнить **bchenc**:

```
>> code = bchenc(msg,7,4) % кодирование
   code = GF(2) array.
   Array elements =
         1       0       1       1       0       0       0
         0       1       1       1       0       1       0
```

Получившаяся кодовая комбинация также лежит в поле Галуа. Видно, что первые $i = 4$ символов являются информационными, а оставшиеся $k = 3$ – корректирующими.

Внес в кодовую комбинацию одиночную ошибку, чтобы затем декодировать и убедиться, что БЧХ-код $(7, 4)$ позволяет исправить такой тип ошибок:

```
>> ncode=code+randerr(2,7) % внесение помех
ncode = GF(2) array.
Array elements =
      1       0       1       1       0       0       1
      0       1       0       1       0       1       0
>> rx=bchdec(ncode,7,4) % декодирование
rx = GF(2) array.
Array elements =
      1       0       1       1
      0       1       1       1
>> isequal(msg,rx) % проверка на равенство
ans =
     1
```

Как и ожидалось, код $(7, 4)$ справился с обнаружением и исправлением одиночной ошибки.

Попробовал сгенерировать порождающий полином для получения других значений помехоустойчивости. Из (**??**) выяснил, что после длины $n = 7$ следует 15. Пусть требуется обеспечить исправление до 2 ошибок, т.е. $s = 2$. Из (**??**) нашел, что число корректирующих бит $k = 8$. Тогда информационных бит всего 7. Воспользовался функцией **bchgenpoly**, которая образует требуемый порождающий полином:

```
>> [poly, t]=bchgenpoly(15,7)
   poly = GF(2) array.
   Array elements =
         1   1   1   0   1   0   0   0   1
   t =
        2
```

Получил полином $g(x) = x^8 + x^7 + x^6 + x^4 + 1$. Также вторым аргументом функция вернула корректирующую способность: она равна двум, как и требовалось.

Далее произвел формирование десяти случайных сообщений длины семь, их кодирование, внесение шума путем сложения с матрицей, генерируемой **randerr**, декодирование и проверка с исходными сообщениями:

```
>> msg=gf(randint(10,7,[0 1])); % 10 7-битных чисел
>> code=bchenc(msg,15,7); % кодирование
```

```
>> ncode=code+randerr(10,15,2); % внесение по 2 ошибки в
   строку
>> rx=bchdec(ncode,15,7); % декодирование
>> isequal(rx,msg) % проверка с исходными кодами
   ans =
        1
```

Таким образом, сформированный полином действительно обеспечивает заявленную корректирующую способность.

1.5.3 •

1.6 Вывод