

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

**Институт искусственного интеллекта**

**Кафедра автоматических систем**

Система менеджмента качества обучения

СМКО МИРЭА 7.3/04.ЗДпр.5КУБ/О/220400.62-16

**КУРСОВАЯ РАБОТА   
по дисциплине «Сети и системы передачи информации»**

ТЕМА: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Код БЧХ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

(название темы)

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

Студент \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

Группа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ шифр \_\_\_\_\_\_\_\_\_\_

МОСКВА 2023 г.



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА-Российский технологический университет»**

**РТУ МИРЭА**

**Институт кибернетики**

**Кафедра автоматических систем**

|  |  |  |
| --- | --- | --- |
|  | УТВЕРЖДАЮ: |  |
|  | Заведующий кафедрой автоматических систем  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (Ф.И.О.) |
|  | «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_2023 г. |

**ЗАДАНИЕ**

на выполнение курсовой работы по дисциплине

«Сети и системы передачи информации»

Студент: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ф.И.О.)

Группа \_\_\_\_\_\_\_\_\_\_\_, шифр \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема: «\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»

(название темы)

Перечень вопросов, подлежащих разработке:

- анализ принципов кодирования/декодирования;

- выбор и анализ алгоритмов работы кодера/декодера;

- разработка и описание аппаратной (программной) реализации кодера;

- разработка и описание аппаратной (программной) реализации декодера;

- разработка и описание примера кодирования/декодирования;

- оценка эффективности метода;

- анализ преимуществ и недостатков метода.

Срок представления к защите курсовой работы до «\_\_\_»\_\_\_\_\_\_\_\_\_\_20\_\_ г.

Задание на курсовую работу выдал \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.

Задание на курсовую работу получил \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (Ф.И.О.)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.

**Оглавление**

[**Введение** 4](#_Toc135867236)

[**Теоретическая часть** 5](#_Toc135867237)

[**Общая часть** 5](#_Toc135867238)

[**Описание алгоритма кодирования БЧХ-кодов** 6](#_Toc135867239)

[**Описание алгоритма декодирования БЧХ-кодов** 8](#_Toc135867240)

[**Декодирование напрямую** 10](#_Toc135867241)

[**Декодер на основе алгоритма Евклида** 11](#_Toc135867242)

[**Практическая часть** 11](#_Toc135867243)

[**Арифметика Галуа** 12](#_Toc135867244)

[**Сравнение декодеров** 14](#_Toc135867245)

[**Заключение** 20](#_Toc135867246)

[**Список литературы** 21](#_Toc135867247)

# **Введение**

В современном мире объем получаемых и используемых данных и информации ежедневно увеличивается, вследствие чего возрастает требования для качества и скорости передачи этих самых данных. Они передаются через так называемые каналы связи, алгоритмов реализации которых существует огромное множество, однако помимо требований к скорости в большинстве случаев первое место занимает целостность передаваемой информации, а это возможно не всегда поскольку на каналы связи могут воздействовать внешние факторы такие как различные природные явления, повреждения аппаратной части канала или человеческий фактор, например вмешательство из вне. Для уменьшения влияния вышеприведенных факторов на качество передачи информации были разработаны и описаны алгоритмы помехоустойчивого кодирования, которые позволяют обнаруживать и исправлять определенное количество ошибок, которые были получены во время передачи сообщения через канал связи.

В данной курсовой работе мной будет рассмотрен а также реализован на языке программирования высокого уровня Python корректирующий код Боуза-Чоудхури-Хоквингема (БЧХ), с реализацией декодера с помощью Евклидова алгоритма и алгоритма ПГЦ, также будет рассмотрено время кодирования и декодирования для различных сообщений с разным количеством предполагаемых ошибок для того чтобы определить насколько данная реализация и данный тип кодов подходит для применения в реальных задачах. Также я рассмотрю данный алгоритмы со стороны алгоритмической сложности, чтобы выбрать тот из них, который является более оптимальным при выполнении декодирования БЧХ-кодов.

# **Теоретическая часть**

## **Общая часть**

В данной курсовой работе будет рассматриваться примитивные БЧХ коды, имеющие длину сообщения и построенные на основе расширенного поля Галуа , поскольку не двоичные БЧХ-коды не являются интересными в вопросе применения на практике, за исключением кодов Рида-Соломона рассмотрение которого не входит в данную работу.

Пусть,

n - длинна кодовых слов

k – длинна информационного блока

d – минимальное расстояние Хэмминга[[1]](#footnote-1),

тогда множество кодовых слов длинны n с минимальным расстоянием d – назовем *блоковым кодом*

Полученный выше блоковый код способен гарантированно обнаруживать до ошибок и исправлять до ошибок

Теперь рассмотрим развитие идеи блокового кода, которое и приведет нас непосредственно к БЧХ коду. Пусть множество с операциями суммы и произведения по модулю 2 образует линейное пространство над конечным полем , тогда блоковый код называется *линейным* если множество его кодовых слов образует линейное подпространство размерности k общего линейного пространства . Получается, что для любого линейного кода произвольная линейная комбинация кодовых слов является кодовым словом

*Циклическим кодом* называется линейный блоковый - код над полем любой циклический сдвиг кодового слова также является кодовым словом этого кода

*Порождающим полиномом* циклического кода является такой полином g(x) степени m = n – k, который

1. Является делителем полинома
2. Все кодовые слова c(x) могут быть представлены как g(x)u(x) mod (), где u(x) – некоторый полином степени, не превышающей

k – 1

Существует 2 подхода к кодированию линейных циклических блоковых кодов, с этого момента и далее под словом код будет подразумеваться именно линейный циклический блоковый код, если не обозначено иное.

* Несистематическое

При несистематическом кодирование кодовое слово c(x) получается, как результат произведения информационного полинома m(x) на порождающий полином g(x)

* Систематическое

В случае систематического кодирования кодовое слово получается из информационного подблока и проверочного. Для циклического кода с порождающим полином g(x) алгоритм систематического кодирования выглядит как:

В практической части мной будет реализовано систематическое кодирование линейных кодов

## **Описание алгоритма кодирования БЧХ-кодов**

Чтобы провести кодирование БЧХ кода для начала необходимо найти минимальный полином. *Минимальным полиномом* для элемента является полином , если он является неприводимым[[2]](#footnote-3) полином минимальной степени, для которого является корнем. Таким образом, корнями минимального полинома являются: . Полученный набор элементов из поля называется цикломатическим классом смежности для элемента , которые либо совпадают, либо не пересекаются, а количество классов равняется либо является делителем .

Для того чтобы построить минимальный полином для элемента необходимо:

1. Построить цикломатической класс, порожденный
2. Найти коэффициенты полинома путем перемножения многочленов для всех

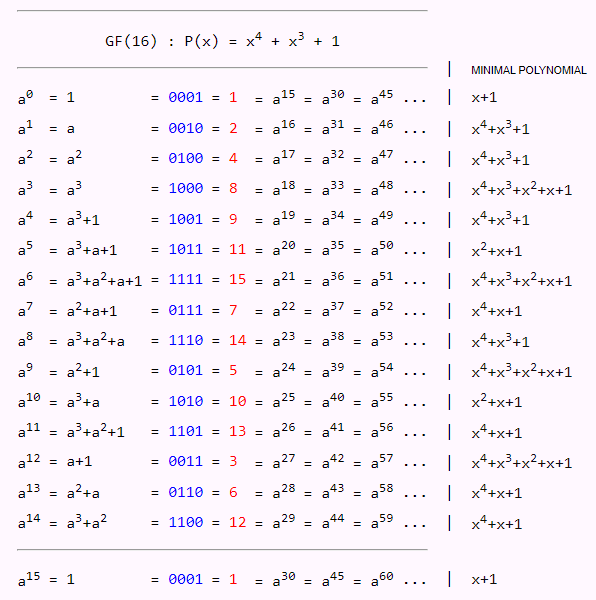
**

Рис.1 Пример для n = 15, t = 2.

Это работает поскольку полином имеет коэффициенты из и является полином для и также элементов поля которые входят в цикломатический класс с .

Таким образом при параметрах и количестве ошибок *кодом БЧХ* назовем [n, k][[3]](#footnote-4) код, в котором порождающий многочлен определятся как минимальный многочлен для элементов и – произвольный примитивный элемент. Данный набор элементов называется *нулями БЧХ-кода.* Также хочется заметить, что поскольку кодовое расстояние d не меньше, чем 2t + 1, то БЧХ-коды способны исправлять не менее t ошибок.



Рис. 2 Пример вариантов бчх-кодов.

## **Описание алгоритма декодирования БЧХ-кодов**

Перед тем как перейти к описанию существующих алгоритмов декодирования хотелось бы определить общие понятия, которые будут использоваться в нескольких алгоритмах, и которые могут помочь фабулу декодирования кодов БЧХ.

*Синдромами принятого искаженного сообщения w(x)* назовем значение полинома *w(x)* в нулях БХЧ кода, то есть как показывалось в предыдущем разделе . Очевидно, что если *w(x)* является кодовым словом, то все синдромы .

Далее поскольку БЧХ-код является помехоустойчивым кодом и как следствие может обнаруживать и исправлять определенное количество ошибок в полученном кодовом слове. Введем понятие *полинома локаторов ошибок:*

*.*

Корнями полученного полинома являются , из этого можно показать, что коэффициенты полинома локаторов ошибок будут удовлетворять СЛАУ вида:

Перед тем как перейти к разбору и анализу существующих алгоритмов декодирование БЧХ-кодов, опираясь на вышеприведённую СЛАУ приведу общую схему для декодирования.

1. При получении кодового слова *w(x)* необходимо вычислить синдромы данного сообщения, затем провести проверку синдромов на равенство нулю, и если все синдромы равны нулю, то возвращаем *w(x)* как результат декодирования.
2. Поскольку синдромы не равны нулю необходимо решить СЛАУ, приведённую выше и с помощью этого, найдем количество допущенных ошибок и коэффициенты полинома локаторов ошибок.
3. Далее находим все корни для полинома локаторов ошибок с помощью полного перебора, по полученным корням необходимо вычислить номера позиций , в которых были обнаружены ошибки.
4. В полученных позициях исправляем ошибки путем инвертирования битов в *w(x)* в соответствии с полученными позициями.

Существующие алгоритмы декодирования БЧХ-кодов зачастую отличаются 2-м шагом данного общего алгоритма, поэтому в данной курсовой работе мной будут рассмотрены основные, которые пользуются популярностью в современном мире, однако реализован будет только один из них.

Описывать алгоритмы декодирования я буду в порядке их появления, чтобы наглядно показать, как со временем изменялся подход к декодированию БЧХ-кодов:

## **Декодирование напрямую**

Также известен как алгоритм Питерсона-Горенстейна-Цирлера (ПГЦ). Данный алгоритм декодирования является самым первым, который был разработан Питерсоном для случая q = 2 (двоичного), далее спустя Горенстейн и Цирлер нашли обобщение данного алгоритма для общего случая.

Суть данного алгоритма заключается в непосредственном решении СЛАУ что я не является полностью оптимальным решением поскольку его сложность растет как куб минимального хэмингового расстояния и рационально его использовать только для малых значений *d.*

Однако основная проблема заключается даже не в сложности алгоритма, помимо этого достаточно трудоемкой задачей является определение количества фактически допущенных ошибок при передаче данных. В случае ПГЦ данная ситуация решается путем обычного перебора по всем значениям *v,* начиная с t. Таким образом первым шагом является попытка решать СЛАУ при текущем *v*.

Если получается так, что матрица СЛАУ является невырожденной[[4]](#footnote-5), то значение *v* будем считать количеством допущенных ошибок, а коэффициенты полинома локаторов ошибок находятся из решения СЛАУ.

В случае если матрица СЛАУ является вырожденной, то это есть признак того, что количество ошибок меньше t, полином локаторов ошибок , значение *v* уменьшается на единицу, а алгоритм повторяется.

Если же СЛАУ не имеет решения на хотя бы одной итерации алгоритма, то ПГЦ выдает отказ от декодирования. Такой же результат будет, если после исправления синдромы не равны нулю, что также означает, что кодовое слово не найдено.

## **Декодер на основе алгоритма Евклида**

Как видно из названия данный алгоритм базируется на стандартном алгоритме Евклида по нахождению наибольшего общего делителя двух чисел (НОД), однако в случае декодирования БЧХ-кодов необходимо найти НОД не двух чисел, а двух полиномов.

Чтобы провести декодирование для начала рассмотрим синдромный полином вида , где в качестве берем вычисленные ранее синдромы. Таким образом можно показать, что и удовлетворяют уравнению:

В полученном уравнении – это некоторый многочлен из[x], степень которого не превышает t. Таким образом значения , для заданных многочленов и могут быть найдены с помощью расширенного алгоритма Евклида. Алгоритм продолжает свою работу пока степень остатка не станет меньше или равна t. Степень найденного полинома равна количеству фактически допущенных ошибок *v* при передаче данных. Отказ от декодирования не выдается, если количество корней не совпадает с *v*.

# **Практическая часть**

В этом разделе моей курсовой работы я сравню программные реализации алгоритмов декодирования БЧХ-кодов, которые были описаны мной в теоретической части в плане скорости работы в целом и при различном количестве ошибок для достаточно большого n.

Также в данном разделе будут описаны основные функции, которые были реализованы для построения симметричного БЧХ-кода, в частности арифметики Галуа, а также кодирования циклических кодов.

## **Арифметика Галуа**

Для данных функций в моей курсовой работе представлен отдельный файл gf.py, и в данном разделе мной будут описаны основные функции, используемые при работе с полями Галуа.

X, Y – две матрицы одинакового размера из элементов поля ,

pm – матрица соответствия между десятичным и степенным представлением в поле

p – полином из [x], numpy.array-вектор коэффициентов, начиная со старшей степени

A – квадратная матрица из элементов поля

b – вектор из элементов поля

n – длина кода

t – исправляемое число ошибок

g – порождающий многочлен кода, numpy.array-вектор коэффициентов

R – нули кода, numpy.array-вектор десятичных чисел, соответствующих элементам из

U – набор исходных сообщений для кодирования, numpy.array-матрица, бинарная матрица

W – набор принятых сообщений, numpy.array-матрица размера

* **genPowMatrix** (primpoly: int) -> ndarray – функция принимает примитивный многочлен primpoly и возвращает матрицу соответствия между десятичным представлением и степенным представлением ненулевых элементов поля по стандартному примитивному элементу, где

[0] = степень примитивного элемента, при которой получается индекс массива от 1 до n-1

[1] = значение примитивного элемента при возведении его в степень индекса от 1 до n-1

* **add** (A: np.ndarray, B: np.ndarray) -> np.ndarray – функция возвращает матрицу размера X, являющуюся поэлементным суммированием матриц X и Y
* **sum** (X: np.ndarray, axis=0) -> np.ndarray - Функция возвращает результат суммирования матрицы X по размерности
* **prod** (X: np.ndarray, Y: np.ndarray, pm: np.ndarray) -> np.ndarray - функция возвращает матрицу размера X, являющуюся поэлементным произведением матриц X и Y
* **divide** (X: np.ndarray, Y: np.ndarray, pm: np.ndarray) -> np.ndarray - функция возвращает матрицу размера X, являющуюся поэлементным делением матриц X и Y
* **linsolve** (A: np.ndarray, b: np.ndarray, pm: np.ndarray) -> np.ndarray - Функция возвращает решение СЛАУ в случае невырожденности A и numpy.nan иначе
* **minpoly** (x: np.ndarray, pm: np.ndarray) - Функция осуществляет поиск минимального полинома в ] для набора корней, задаваемых x и возвращает минимальный полином, а также все корни минимального полинома
* **clean\_zeros** (p: np.ndarray) -> np.ndarray – функция принимает полином и возвращает его же, но без ведущих нулей
* **polyval** (p: np.ndarray, x: np.ndarray, pm: np.ndarray) -> np.ndarray - Функция возвращает значения полинома p для набора элементов x.
* **polyad** (p1: np.ndarray, p2: np.ndarray) -> np.ndarray - Функция возвращает результат произведения двух полиномов в виде numpy.array-вектора
* **polyprod** (p1: np.ndarray, p2: np.ndarray, pm: np.ndarray) -> np.ndarray - Функция возвращает результат произведения двух полиномов в виде numpy.array-вектора коэффициентов
* **polydiv** (p1: np.ndarray, p2: np.ndarray, pm: np.ndarray) -> tuple[np.ndarray, np.any] - Функция осуществляет деление с остатком многочлена p1 на многочлен p2 и возвращает частное (numpy-array-вектор коэффициентов) и остаток от деления (также numpy-array-вектор коэффициентов)
* **euclid** (p1: np.ndarray, p2: np.ndarray, pm: np.ndarray, max\_deg=0) - Функция реализует расширенный алгоритм Евклида для пары многочленов p1 и p2, где max\_deg – максимально допустимая степень остатка, число, если равно нулю, то алгоритм Евклида работает до конца и возвращает остаток (numpy-array-вектор коэффициентов), коэффициент при p1 (numpy-array-вектор коэффициентов), коэффициент при p2 (numpy-array-вектор коэффициентов)
* **generateValues** (self) -> None – генерирует на основе полученных значений n и t, pm,R,g а также poly – случайный примитивный полином необходимой степени
* **findPrimePoly** (self, n: int) -> list[int] – Функция возвращает список полученных примитивных полиномов заданной степени в десятичном виде
* **encode** (self, U: np.ndarray) -> np.ndarray - Функция осуществляет систематическое кодирование циклического кода и возвращает numpy.array матрицу с закодированными сообщениями
* **decode** (self, W: np.ndarray) -> np.ndarray - Функция осуществляет декодирование БЧХ кода и возвращает numpy.array-матрицу с декодированными сообщениями размера <число сообщений>×n. В случае отказа от декодирования соответствующая строка матрицы состоит из numpy.nan
* **dist** (self) - Функция возвращает кодовое расстояние (число), найденное полным перебором

## **Сравнение декодеров**

В данной части моей курсовой работы мной будут рассмотрены и сравнены описанные алгоритмы декодирования БЧХ кодов при различных размерах и количествах ошибок.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | t | Euclid | PGZ |  | n | t | Euclid | PGZ |
| 7 | 3 | 0.0010009 | 0.0010018 |  | 1023 | 3 | 0.0339985 | 0.0690005 |
| 15 | 3 | 0.0009742 | 0.0010250 |  | 1023 | 7 | 0.0359998 | 0.0700026 |
| 15 | 7 | 0.0019853 | 0.0019984 |  | 1023 | 15 | 0.0400023 | 0.0739977 |
| 31 | 3 | 0.0020003 | 0.0019987 |  | 1023 | 31 | 0.0460296 | 0.0799701 |
| 31 | 7 | 0.0040009 | 0.0029998 |  | 1023 | 63 | 0.0589998 | 0.0910265 |
| 31 | 15 | 0.0060003 | 0.0029998 |  | 1023 | 127 | 0.0850015 | 0.1199999 |
| 63 | 3 | 0.0030000 | 0.0049989 |  | 1023 | 254 | 0.1429987 | 0.2559738 |
| 63 | 7 | 0.0040004 | 0.0050011 |  | 1023 | 508 | 0.2630246 | 1.1240201 |
| 63 | 15 | 0.0059729 | 0.0070295 |  | 2047 | 3 | 0.0700004 | 0.1400003 |
| 63 | 31 | 0.0109994 | 0.0090265 |  | 2047 | 7 | 0.0750268 | 0.1450002 |
| 127 | 3 | 0.0050023 | 0.0079966 |  | 2047 | 15 | 0.0789990 | 0.1480250 |
| 127 | 7 | 0.0060027 | 0.0089726 |  | 2047 | 31 | 0.0859997 | 0.1539993 |
| 127 | 15 | 0.0079992 | 0.0109751 |  | 2047 | 63 | 0.0999706 | 0.1730015 |
| 127 | 31 | 0.0130036 | 0.0139968 |  | 2047 | 127 | 0.1339998 | 0.2120271 |
| 127 | 63 | 0.0239770 | 0.0160024 |  | 2047 | 254 | 0.2069983 | 0.3669751 |
| 255 | 3 | 0.0099993 | 0.0169978 |  | 2047 | 508 | 0.3470018 | 1.2610414 |
| 255 | 7 | 0.0109949 | 0.0179772 |  | 2047 | 1016 | 0.6609998 | 8.6554713 |
| 255 | 15 | 0.0129695 | 0.0209990 |  | 4095 | 3 | 0.1379743 | 0.2830005 |
| 255 | 31 | 0.0190012 | 0.0239990 |  | 4095 | 7 | 0.1545689 | 0.2857997 |
| 255 | 63 | 0.0310011 | 0.0339990 |  | 4095 | 15 | 0.1499741 | 0.2899985 |
| 255 | 127 | 0.0500264 | 0.0559747 |  | 4095 | 31 | 0.1869981 | 0.3070023 |
| 511 | 3 | 0.0200267 | 0.0369995 |  | 4095 | 63 | 0.1869957 | 0.3459733 |
| 511 | 7 | 0.0210011 | 0.0380249 |  | 4095 | 127 | 0.2289999 | 0.3950243 |
| 511 | 15 | 0.0240052 | 0.0419750 |  | 4095 | 254 | 0.3360050 | 0.5910006 |
| 511 | 31 | 0.0280299 | 0.0429704 |  | 4095 | 508 | 0.5240004 | 1.4380033 |
| 511 | 63 | 0.0399997 | 0.0510001 |  | 4095 | 1016 | 0.9620275 | 9.5209491 |
| 511 | 127 | 0.0629992 | 0.0750265 |  | 4095 | 2032 | 1.9243233 | 75.2072654 |
| 511 | 254 | 0.1130259 | 0.1919749 |  |  |  |  |  |

Таблица 1 – Время выполнения декодирования БЧХ-кодов

В данной таблице приведены результаты декодирования БЧХ-кодов с различными параметрами 2 разными декодерами. Из данной таблицы видно, что с ростом размера сообщения и особенно с ростом количество ошибок PGZ декодер показывает результаты намного хуже декодера основанного на алгоритме Евклида, который справляется даже с достаточно большими сообщениями за малое время, которое минимум в несколько раз меньше чем декодирование аналогичного БЧХ-кода PGZ декодером.

Ниже будут приведены графики зависимости времени выполнения декодирования от параметров БЧХ-кода

График 1 – БЧХ(127, t)

График 2 – БЧХ (255, t)

График 3 – БЧХ (511, t)

График 4 – БЧХ (1023, t)

График 5 – БЧХ (2047, t)

Согласно результатам, которые были изображены на графиках видно, что для малого размера сообщения алгоритм декодирования PGZ действительно может быть достаточно эффективным в сравнении с декодером на основе алгоритма Евклида. Однако с ростом размера сообщения и количества ошибок скорость работы PGZ значительно увеличивается, в то время как Евклидов алгоритм показывает относительно стабильное время работы.

Вторая часть изучения работоспособности и эффективности реализованных мною алгоритмов декодирования БЧХ-кодов будет заключаться в анализе того, как данные алгоритмы реагируют на получение большего количества ошибок чем ограничение сверху.

В каждом случае рассматривался код с t+1 ошибкой, в таблицу занесены средние результаты, полученные при декодировании для семидесяти пяти кодов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | t | Success % | Missed % | Error % |  | n | t | Success % | Missed % | Error % |
| 7 | 3 | 0 | 100 | 0 |  | 1023 | 3 | 0 | 16 | 84 |
| 15 | 3 | 0 | 0.33 | 0.66 |  | 1023 | 7 | 0 | 0 | 100 |
| 15 | 7 | 0 | 100 | 0 |  | 1023 | 15 | 0 | 0 | 100 |
| 31 | 3 | 0 | 14.66 | 85.33 |  | 1023 | 31 | 0 | 0 | 100 |
| 31 | 7 | 0 | 1.33 | 98.66 |  | 1023 | 63 | 0 | 0 | 100 |
| 31 | 15 | 0 | 100 | 0 |  | 1023 | 127 | 0 | 0 | 100 |
| 63 | 3 | 0 | 18.66 | 81.33 |  | 1023 | 254 | 0 | 0 | 100 |
| 63 | 7 | 0 | 0 | 100 |  | 1023 | 508 | 0 | 0 | 100 |
| 63 | 15 | 0 | 0 | 100 |  | 2047 | 3 | 0 | 21.33 | 78.66 |
| 63 | 31 | 0 | 0 | 100 |  | 2047 | 7 | 0 | 0 | 100 |
| 127 | 3 | 0 | 17.33 | 82.66 |  | 2047 | 15 | 0 | 0 | 100 |
| 127 | 7 | 0 | 0 | 100 |  | 2047 | 31 | 0 | 0 | 100 |
| 127 | 15 | 0 | 0 | 100 |  | 2047 | 63 | 0 | 0 | 100 |
| 127 | 31 | 0 | 0 | 100 |  | 2047 | 127 | 0 | 0 | 100 |
| 127 | 63 | 0 | 100 | 0 |  | 2047 | 254 | 0 | 0 | 100 |
| 255 | 3 | 0 | 13.33 | 86.66 |  | 2047 | 508 | 0 | 0 | 100 |
| 255 | 7 | 0 | 0 | 100 |  | 2047 | 1016 | 0 | 0 | 100 |
| 255 | 15 | 0 | 0 | 100 |  | 4095 | 3 | 0 | 16 | 84 |
| 255 | 31 | 0 | 0 | 100 |  | 4095 | 7 | 0 | 0 | 100 |
| 255 | 63 | 0 | 0 | 100 |  | 4095 | 15 | 0 | 0 | 100 |
| 255 | 127 | 0 | 100 | 0 |  | 4095 | 31 | 0 | 0 | 100 |
| 511 | 3 | 0 | 18.66 | 81.33 |  | 4095 | 63 | 0 | 0 | 100 |
| 511 | 7 | 0 | 0 | 100 |  | 4095 | 127 | 0 | 0 | 100 |
| 511 | 15 | 0 | 0 | 100 |  | 4095 | 254 | 0 | 0 | 100 |
| 511 | 31 | 0 | 0 | 100 |  | 4095 | 508 | 0 | 0 | 100 |
| 511 | 63 | 0 | 0 | 100 |  | 4095 | 1016 | 0 | 0 | 100 |
| 511 | 127 | 0 | 0 | 100 |  | 4095 | 2032 | 0 | 0 | 100 |
| 511 | 254 | 0 | 0 | 100 |  |  |  |  |  |  |

Таблица 2 – результаты декодирования при t+1 ошибок

Вывод: согласно результатам, которые были получены во время проверки и анализа работоспособности реализованных декодеров БЧХ-кодов можно утверждать, что наиболее эффективным алгоритмом является алгоритм на основе разложения Евклида, поскольку для больших размеров сообщения он показывает меньшее время работы, чем алгоритм PGZ для аналогичных сообщений.

Второй результат, который был получен мной говорит о том, что БЧХ-коды гарантированно исправляют до t ошибок в слове, однако если произойдет даже t+1 ошибка ни один из представленных декодеров не сможет восстановить переданную информацию. Также можно заметить, что при малых размерах сообщения или же при малом t декодер восстанавливает сообщение ошибочно, в то время как для больших значений n и t декодеры посылают отказ от декодирования.

# **Заключение**

В данной курсовой работе мной был рассмотрен и реализован на языке программирования Python двоичный симметричный БЧХ-код, а также проведено сравнения двух популярных декодеров на основе различных алгоритмов и предложен наиболее оптимальный. Помимо это было проведен анализ того, как БЧХ-код работает в условия чрезмерного количества ошибок при передаче сообщения.

# **Список литературы**

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7

1. Для линейного кода определяется как минимальный хэммингов вес (количество ненулевых бит) среди ненулевых кодовых слов [↑](#footnote-ref-1)
2. Такой полином, который нельзя разложить на множители [↑](#footnote-ref-3)
3. d опущено за не важностью [↑](#footnote-ref-4)
4. Определитель отличен от нуля [↑](#footnote-ref-5)