Московский Государственный Университет им. М.В. Ломоносова
Факультет Вычислительной Математики и Кибернетики

Отчёт по практическому заданию в рамках курса прикладной алгебры

Конечные поля и коды БЧХ

Работу выполнил
студент 323 группы
Чепурнов А. В.

Москва 2018

Постановка задачи

Реализовать в среде Python 3 основные операции в поле $\mathbb{F}_2^q$ (сложение, умножение, деление, решение СЛАУ, поиск минимального многочлена из $\mathbb{F}_2[x]$ для заданного набора корней из поля $\mathbb{F}_2^q$), основные операции для работы с многочленами из $\mathbb{F}_2^q[x]$ (произведение многочленов, деление многочленов с остатком, расширенный алгоритм Евклида для пары многочленов, вычисление значения многочлена для набора элементов из $\mathbb{F}_2^q$), процедуры построения порождающего многочлена, систематического кодирования циклического кода, вычисления истинного минимального расстояния циклического кода и декодирования БЧХ-кода с помощью метода PGZ и на основе расширенного алгоритма Евклида.

Рекомендуемые проверки на корректность:

1. Порождающий полином БЧХ-кода должен быть делителем многочлена $x^n - 1$
2. Произвольное кодовое слово БЧХ-кода $v(x)$ должно делиться без остатка на порождающий многочлен кода $g(x)$, а также обращаться в ноль на нулях кода (все синдромы кодового слова равны нулю)
3. Минимальный многочлен $m_\alpha(x)$ для элемента $\alpha \in \mathbb{F}_2^q$, вычисляемый как многочлен с корнями $\alpha, \alpha^2, \alpha^4 \dots, \alpha^{2^q}$, должен иметь коэффициенты из $\mathbb{F}_2$
4. Минимальное кодовое расстояние БЧХ-кода d, найденное полным перебором, должно быть не меньше, чем величина 2t + 1

Реализация

Помимо описанных в задании функций, в **gf.py** включена функция **clean_zeros(p)**, принимающая полином из (**numpy.array**-вектор коэффициентов, начиная со старшей степени) и возвращающая его же, но без ведущих нулей, а также **polyadd(p1, p2)**, принимающая два полинома и возвращающая их сумму (в отличии от функции **add()** входные векторы могут быть разной длинны). В **bch.py** реализован описанный в задании класс **BCH**. Метод **dist()** имеет дополнительный необязательный параметр **check**, обеспечивающий два режима работы функции: при **check=False** (значение по умолчанию) возвращает кодовое расстояние, найденное полным перебором, при **check=True** осуществляет проверки 2 и 4 кода на корректность и возвращает **True**, если они пройдены, **False –** иначе. Метод **checker()** без входных параметров проводит все проверки на корректность, возвращает **True**, если они пройдены, **False –** иначе.

Для исследования кодов в **bch.py** реализованы две функции:

plot(n, imgname)
Описание параметров:

- **n** – длина кода
- **imgname** – имя файла для графика

Строит график зависимости скорости БЧХ-кода r = k/n от количества исправляемых кодом ошибок t для заданной длины кода n.

test(code, r, s, inaccuracy=False)
Описание параметров:

- **code** – БЧХ-код, объект класса **BCH**
- **r –** число ошибок при передаче по каналу с шумом
- **s –** число передаваемых сообщений
- **inaccuracy** – неточная генерация ошибок

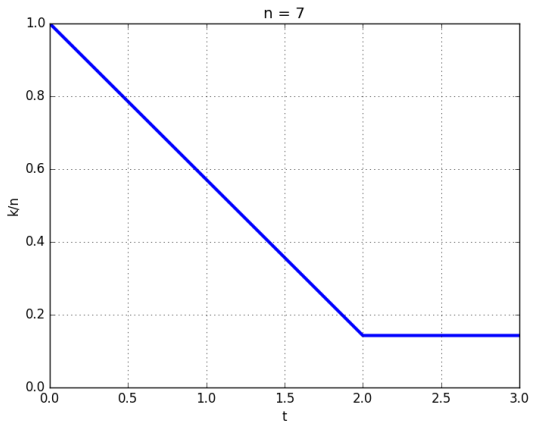
Кодирует **s** слов БЧХ-кодом **code**, генерирует в полученном сообщении ошибки, дешифрует его алгоритмами PGZ и Euclid, возвращает кортеж из переменных типа **float** со следующей статистикой:

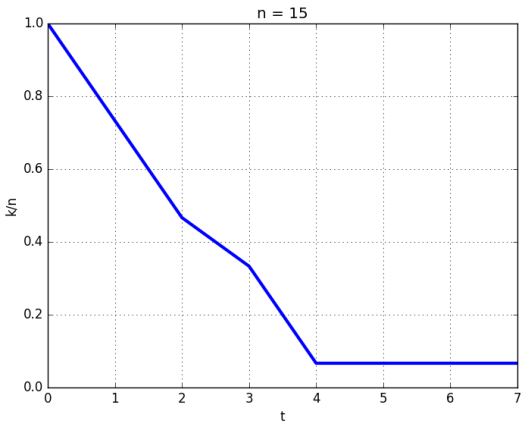
- Время работы алгоритма PGZ
- Доля исправленных алгоритмом PGZ слов
- Доля неверно дешифрованных алгоритмом PGZ слов
- Доля слов, для которых PGZ выдал отказ от декодирования
- Время работы алгоритма Euclid
- Доля исправленных алгоритмом Euclid слов
- Доля неверно дешифрованных алгоритмом Euclid слов
- Доля слов, для которых Euclid выдал отказ от декодирования

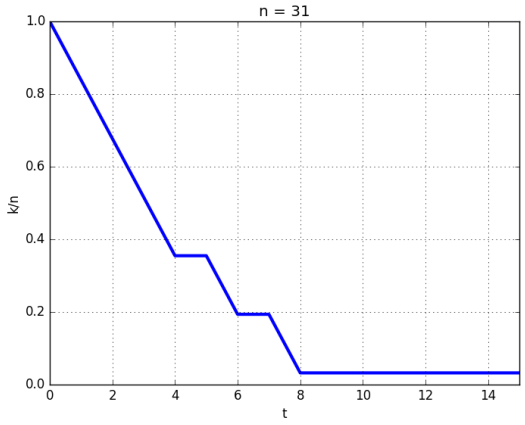
При **inaccuracy=False** в каждом кодовом слове заменяется ровно **r** случайных бит, при **inaccuracy=True** каждый бит сообщения заменяется с вероятностью **r/n**, где **n** – длина кода.

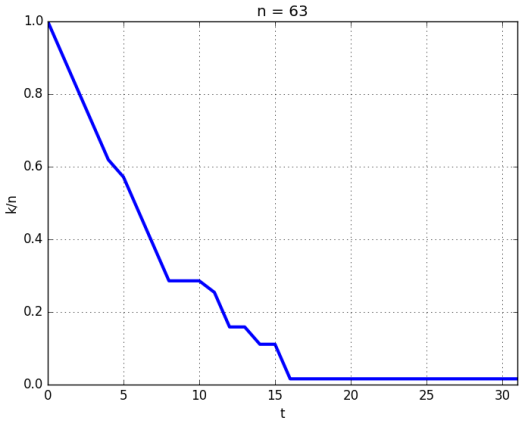
Какие t следует выбирать на практике для заданного n

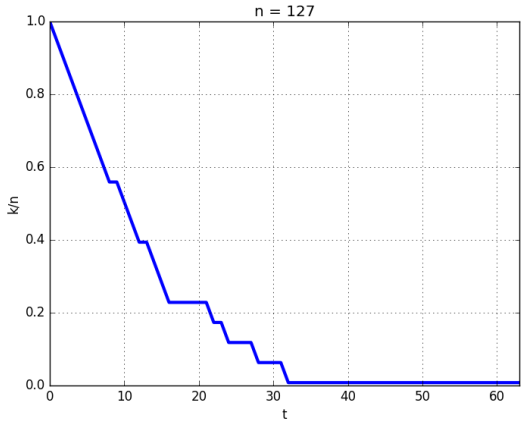
Выбирая t для заданной длины n необходимо определить, что приоритетнее – предостеречь сообщение от большего числа ошибок или уменьшить количество избыточной информации. Понятно, что нужно выбирать только такие t, что для t + 1 скорость кода будет ниже. Также можно заметить, что выбирать $t > \lfloor n/4 \rfloor$ всегда невыгодно, так как тогда степень порождающего многочлена кода $g(x)$ равна n – 1, а k = 1 (тривиальные коды). Отсюда можно сделать вывод, что в общем случае логичнее всего выбрать самое большое значение t для самого длинного горизонтального участка линии скорости (см. рис для больших n) в левой половине графика. Этому значению соответствует $t = \lfloor n/6 \rfloor$.

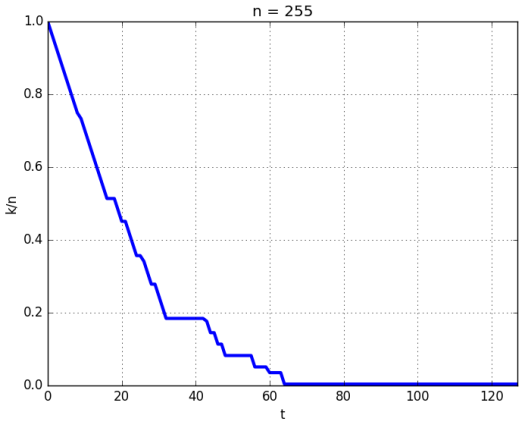


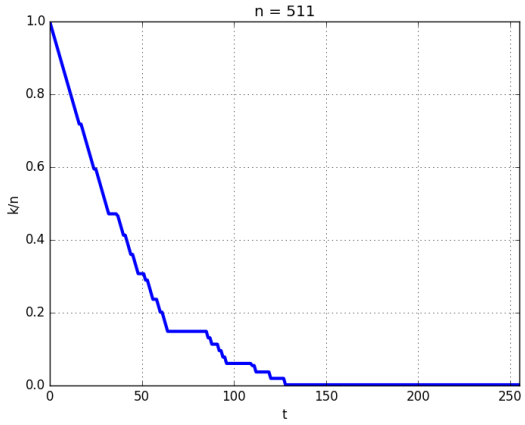


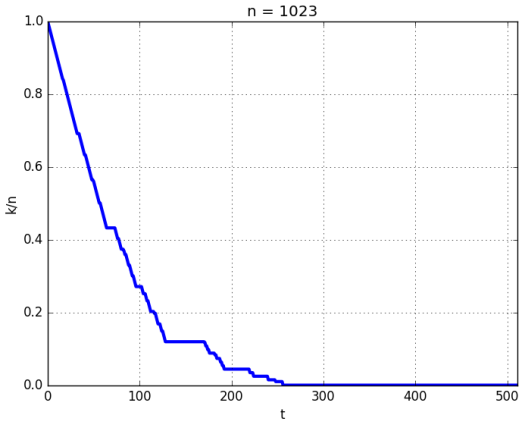












БЧХ-коды с d > 2t + 1

Для получения БЧХ-кодов с истинным минимальным расстоянием d > 2t + 1 достаточно для фиксированного n выбрать на графике на горизонтальном участке линии скорости любое t кроме самого большого. Например $t = \lceil n/8 \rceil$ для n > 15:

d(BCH(31, 4)) = 11 > 2*4 + 1 = 9

Или уже упомянутые тривиальные коды с k = 1: $\lfloor n/4 \rfloor < t < \lfloor (n-1)/2 \rfloor$

d(BCH(7, 2)) = 7 > 2*2 + 1 = 5

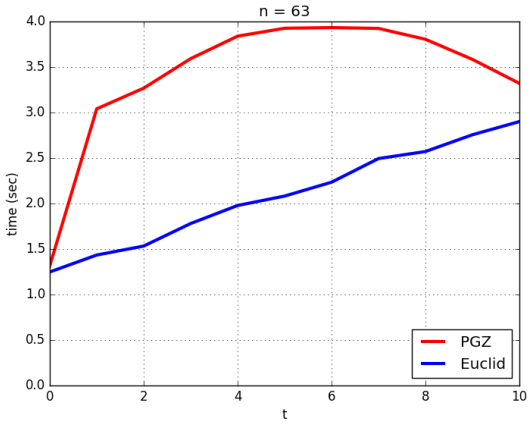
d(BCH(15, 4)) = 15 > 2*4 + 1 = 9

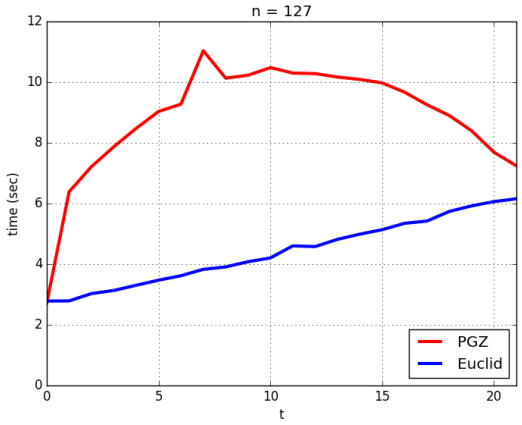
d(BCH(31, 8)) = 31 > 2*8 + 1 = 17

Сравнение двух методов декодирования по времени работы

Исследования показали, что в основном алгоритм декодера Euclid работает быстрее PGZ. Иногда PGZ срабатывает быстрее, это происходит при количестве ошибок r равном числу исправляемых ошибок БЧХ-кода t или не сильно меньше него. В этом случае алгоритму нужно решить СЛАУ один или несколько раз, что быстрее поиска полинома локаторов расширенным алгоритмом Евклида. Однако при уменьшении числа ошибок, преимущество декодера Euclid становятся очевидным.

n	t	PGZ 1 ошибка	Euclid 1 ошибка	PGZ t ошибок	Euclid t ошибок	PGZ > t ошибок	Euclid > t ошибок
3	1	0.119968	0.173810	0.119968	0.173810	0.118276	0.183223
7	1	0.291026	0.263672	0.291026	0.263672	0.282222	0.269598
15	3	0.735964	0.438629	0.801350	0.878387	0.768340	0.731264
31	5	1.644110	0.750127	1.588457	1.387943	1.602330	1.458460
63	10	3.039866	1.471508	3.382564	2.968749	3.346169	2.916843
127	21	6.359517	2.785934	6.841852	6.206841	7.068943	6.142856
255	42	13.172846	5.424193	14.461357	12.468600	14.602691	12.501773
511	85	29.164334	11.095959	31.061385	26.553629	31.186594	26.597538





Испытания кода

Тесты, где в каждом кодовом слове заменялось ровно **r** случайных бит:

c	t	r	OK	Доля ошибок	Доля отказов
3	1	0	100%	0%	0%
7	1	0	100%	0%	0%
15	3	0	100%	0%	0%
31	5	0	100%	0%	0%
63	10	0	100%	0%	0%
127	21	0	100%	0%	0%
255	42	0	100%	0%	0%
511	85	0	100%	0%	0%

n	t	r	OK	Доля ошибок	Доля отказов
3	1	1	100%	0%	0%
7	1	1	100%	0%	0%
15	3	1	100%	0%	0%
31	5	1	100%	0%	0%
63	10	1	100%	0%	0%
127	21	1	100%	0%	0%
255	42	1	100%	0%	0%
511	85	1	100%	0%	0%

n	t	r	OK	Доля ошибок	Доля отказов
3	1	1	100%	0%	0%
7	1	1	100%	0%	0%
15	3	3	100%	0%	0%
31	5	5	100%	0%	0%
63	10	10	100%	0%	0%
127	21	21	100%	0%	0%
255	42	42	100%	0%	0%
511	85	85	100%	0%	0%

n	t	r	OK	Доля ошибок	Доля отказов
3	1	2	0%	100%	0%
7	1	2	0%	100%	0%
15	3	4	0%	42%	58%
31	5	6	0%	12%	88%
63	10	11	0%	0%	100%
127	21	22	0%	0%	100%
255	42	43	0%	0%	100%
511	85	86	0%	0%	100%

n	t	r	OK	Доля ошибок	Доля отказов
3	1	2	0%	100%	0%
7	1	5	0%	100%	0%
15	3	11	0%	34%	66%
31	5	23	0%	21%	79%
63	10	47	0%	0%	100%
127	21	95	0%	0%	100%
255	42	191	0%	0%	100%
511	85	383	0%	0%	100%

Тесты, где каждый бит сообщения заменялся с вероятностью **r/n**:

n	t	r	OK
31	5	1	100%
31	5	2	99%
31	5	3	92%
31	5	4	84%
31	5	5	58%
63	10	1..5	100%
63	10	6	98%
63	10	7	86%
63	10	8	81%
63	10	9	76%
63	10	10	62%
127	21	1..11	100%
127	21	12	99%
127	21	13	99%
127	21	14	99%
127	21	15	94%
127	21	16	92%
127	21	17	88%

n	t	r	OK
127	21	18	81%
127	21	19	80%
127	21	20	67%
127	21	21	47%
255	42	1..28	100%
255	42	29	99%
255	42	30	98%
255	42	31	100%
255	42	32	98%
255	42	33	98%
255	42	34	95%
255	42	35	87%
255	42	36	87%
255	42	37	88%
255	42	38	83%
255	42	39	67%
255	42	40	71%
255	42	41	56%
255	42	42	56%

Все тесты были проведены для сообщений из 100 слов.

Основные выводы

Испытания подтверждают, что БЧХ-коды гарантированно исправляют до t ошибок в слове. Однако, если произойдёт хотя бы t + 1 ошибка, декодер уже не восстановит информацию. Даже в тривиальном коде BCH(15, 4) при 5 ошибках декодер всегда выдаёт отказ, хотя очевидно, что слова можно восстановить. Если ошибок больше t, при небольших n декодер чаще неверно исправляет слова, тогда как при больших n всегда выдаёт отказ.

Результаты работы алгоритмов PGZ и Euclid идентичны, однако на практике лучше использовать Euclid, так как он быстрее и меньше зависит от количества ошибок в слове.

Тесты, в которых каждый бит терялся с вероятностью **r/n**, показывают, что в реальных условиях для минимизации потерянной информации число исправляемых кодом ошибок **t** должно быть хотя бы в 4/3 раза больше **r**.