Отчёт о выполнении практического задания “Конечные поля и коды БЧХ”
по курсу “Прикладная алгебра”
Студента 321 группы ВМК МГУ
Гарькавого Ивана Сергеевича

Москва, 2018 г.

Исходные коды модулей gf и bch, таблицы в формате csv, а также полную информацию о проведении исследований в формате ipython notebook, можно найти в github-репозитории: https://github.com/garx0/pa

Пункты 5, 6 задания:
При переборе значений параметров n и t БЧХ-кода видно, что для определенных интервалов значений t получаются одни и те же коды (а значит, с одинаковым числом исправляемых ошибок). Поэтому была составлена таблица БЧХ-кодов для n от 7 до 2047 (т.е. n = 7, 15, 31, 63, 127, 255, 511, 1023, 2047), у которых t – максимальное среди значений t, задающих этот же код (т.е. таблица всех различные БЧХ коды для данных n).

Часть этой таблицы для n ≤ 127: (для построения поля Галуа был взят первый в лексикографическом порядке неприводимый многочлен данной степени; если взять другой, то и порождающий многочлен g получится другим)

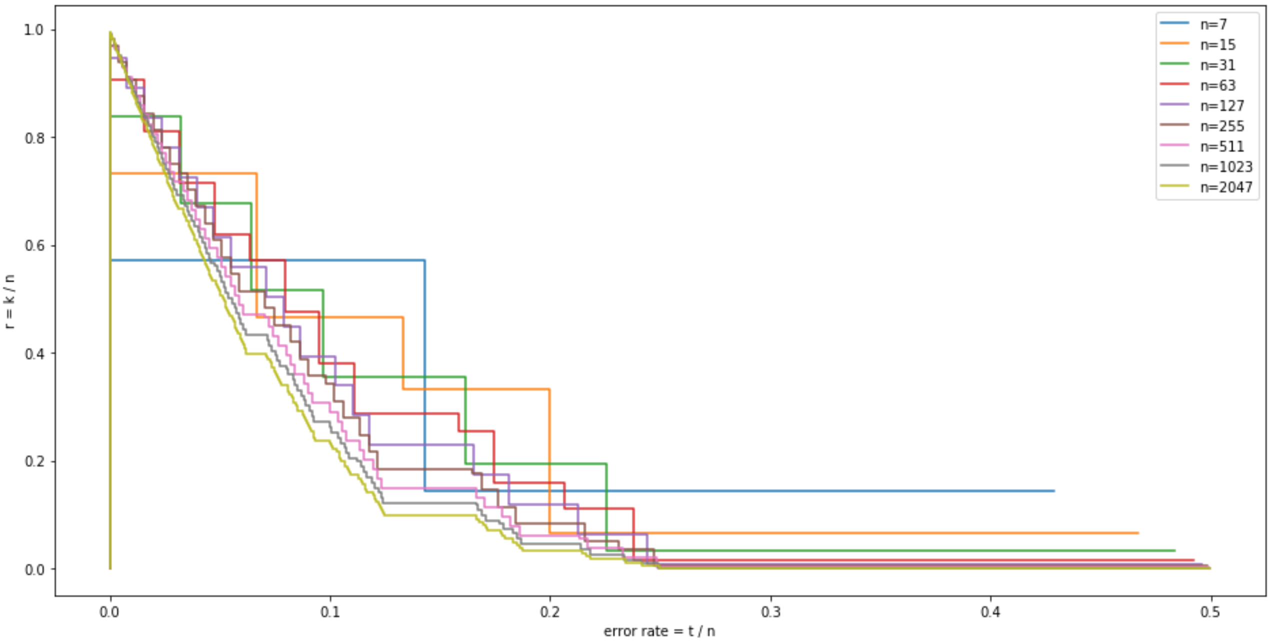
	n	k	t	r = k / n	g (hex)
0	7	4	1	0.571429	B
1	7	1	3	0.142857	7F
2	15	11	1	0.733333	13
3	15	7	2	0.466667	1D1
4	15	5	3	0.333333	537
5	15	1	7	0.066667	7FFF
6	31	26	1	0.838710	25
7	31	21	2	0.677419	769
8	31	16	3	0.516129	8FAF
9	31	11	5	0.354839	1626D5
10	31	6	7	0.193548	32DEA27
11	31	1	15	0.032258	7FFFFFFF
12	63	57	1	0.904762	43
13	63	51	2	0.809524	1539
14	63	45	3	0.714286	782CF
15	63	39	4	0.619048	1DB2777
16	63	36	5	0.571429	86E8113
17	63	30	6	0.476190	37CD0EB67
18	63	24	7	0.380952	F69AC20921
19	63	18	10	0.285714	2F30B529D3D5
20	63	16	11	0.253968	CD930BDD3B2B
21	63	10	13	0.158730	2759262D5D506D
22	63	7	15	0.111111	153225B1D0D73DF
23	63	1	31	0.015873	7FFFFFFFFFFFFFFF
24	127	120	1	0.944882	83
25	127	113	2	0.889764	547D
26	127	106	3	0.834646	29301B
27	127	99	4	0.779528	18A5793F
28	127	92	5	0.724409	E11CA9B57
29	127	85	6	0.669291	7767AD3EA6F
30	127	78	7	0.614173	292C316EEA273
31	127	71	9	0.559055	12B7F8913932C11
32	127	64	10	0.503937	F4845518B9582A1F
33	127	57	11	0.448819	41DA919D9EFB36A699
34	127	50	13	0.393701	29131F09AC7A1C06EE6F
35	127	43	14	0.338583	19A1630A2E2E0D166F0C5D
36	127	36	15	0.283465	D5306D6BFDBC8574719E70D
37	127	29	21	0.228346	5106DAE17A61E520C606A4E29
38	127	22	23	0.173228	3921BD09D78037A53FA89AEAE2B
39	127	15	27	0.118110	1657BC0A307F9382810E59A6D16BB

	n	k	t	r = k / n	g (hex)
40	127	8	31	0.062992	AB31169C84A4DB8F41A8CBB0EBCFBF
41	127	1	63	0.007874	7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

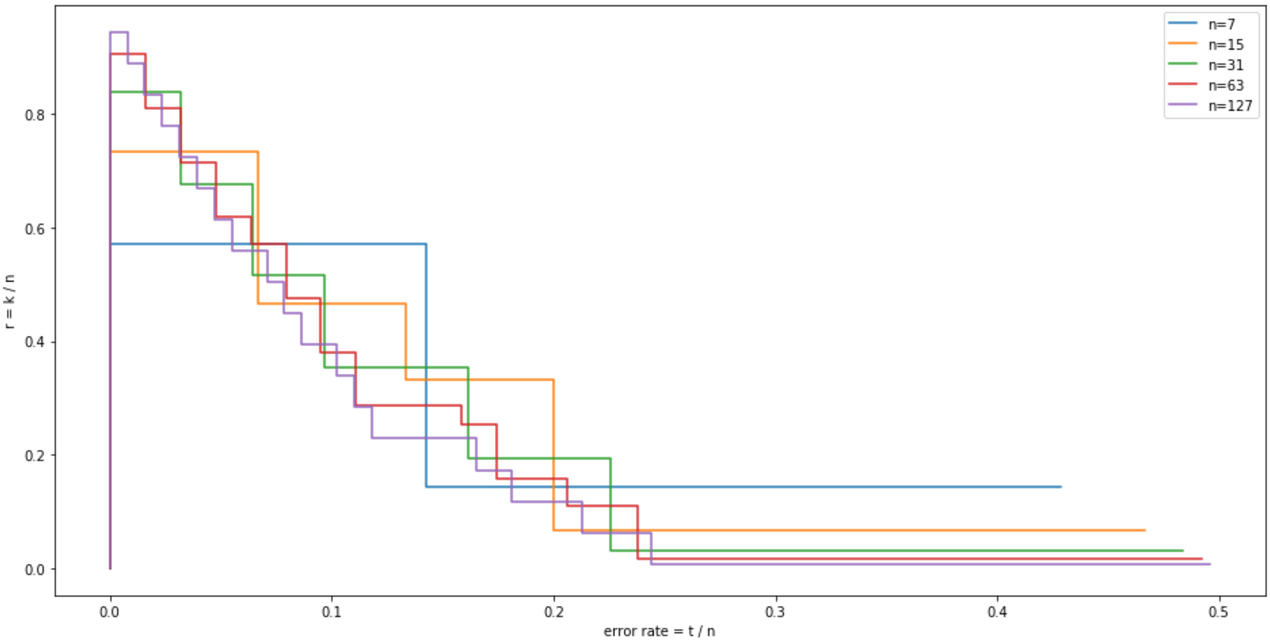
Примеры кодов, у которых реальное кодовое расстояние d > 2t+1:

1) n=7, t=2, d=7 > 5 = 2t+1 (и этот же код задается n=7, t=3, d=7 ≥ 7 = 2t+1)
2) n=15, t=4, d=15 > 9 = 2t+1 (и этот же код задается n=15, t=7, d=15 ≥ 15 = 2t+1).

Для этой таблицы нужно построить графики зависимости скорости кода (k/n) от числа исправляемых ошибок. Но чтобы более корректно сравнить такие графики для разных n, лучше изобразить зависимость скорости от отношения числа ошибок к длине сообщения в котором они могут возникнуть (т.е. от частоты ошибок), что и было сделано.



Тот же график, но без n ≥ 255:



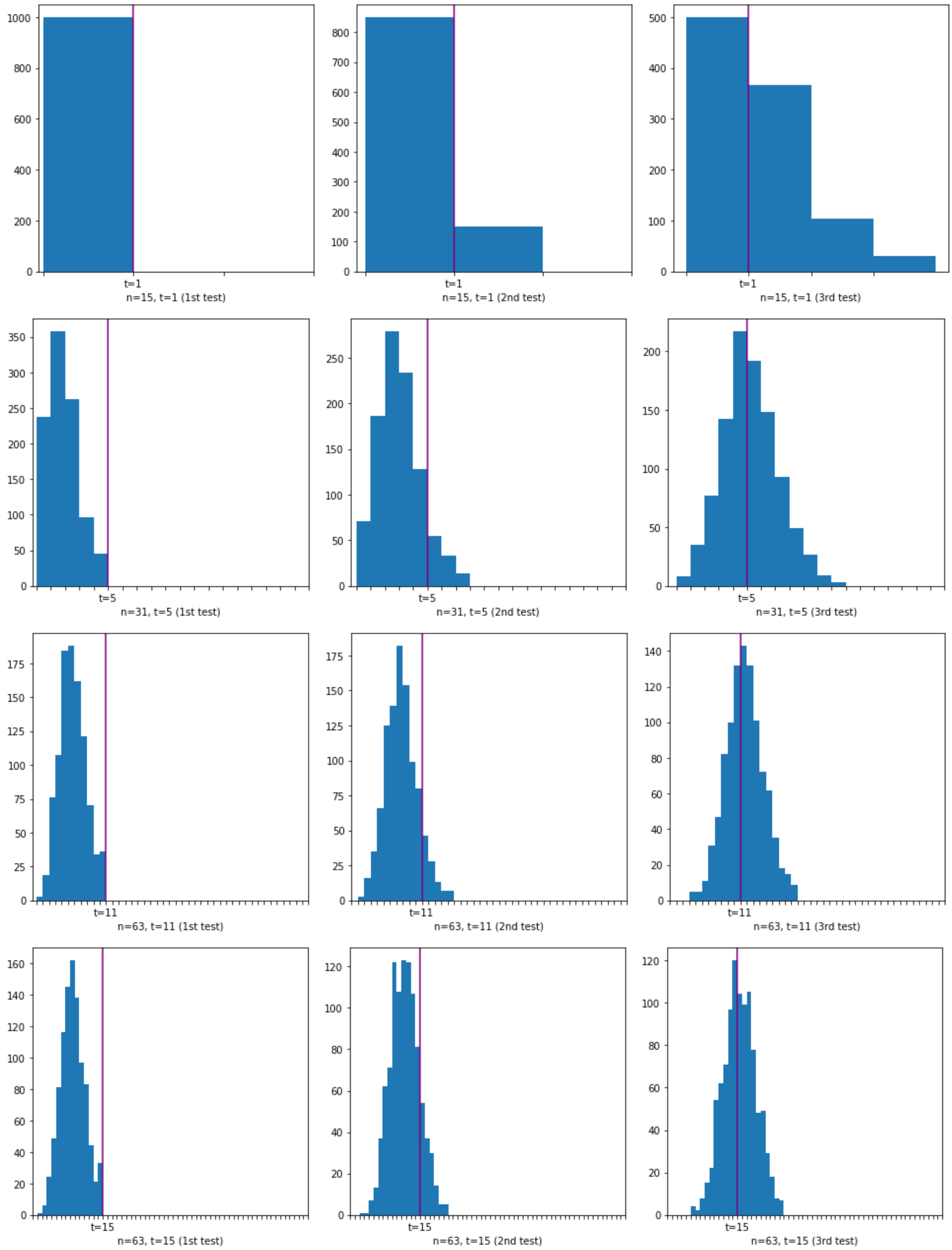
Видно, что скорость падает при росте t. Также она медленно падает при фиксированном отношении t/n (частоте ошибок) и росте n, поэтому рассматривать коды с n от 255 смысла нет.
При фиксированном n, число t надо выбирать в соответствии с зашумленностью среды передачи данных, и, если есть возможность, брать меньше, чтоб скорость не была слишком низкая. Если есть свобода в выборе n (т.е. выборе деления потока данных на сообщения), и надо строить код для определенной частоты ошибок, стоит провести вертикальную черту на этом графике, и по ней выбрать n, при которых скорость будет выше (но не любое, т.к. от выбора также зависит время кодирования и декодирования кода, а также надежность кода при числе ошибок, превышающем t (эта особенность рассмотрена в конце П.7,8)).

Пункты 7, 8 задания:
Для тестирования декодеров, а именно измерения времени и доли разных исходов декодирования, была построена модель среды передачи информации.
Под сообщением подразумевается двоичный вектор, подаваемый на вход кодеру или декодеру, а под набором сообщений – матрица, составленная из таких векторов.
Набор незакодированных сообщений U строится так, чтобы количества единиц на сообщение были распределены по набору сообщений равномерно (в итоге также получается, что общее количество нулей и единиц в большом наборе сообщений примерно одинаково). Набор U подаётся на вход кодеру, и на выходе получаем набор закодированных сообщений V.
Выбирается amt > 0 – значение уровня шума (это – единственный параметр среды передачи информации). В наборе закодированных сообщений V каждое значение бита отклоняется на случайное значение из отрезка [-amt/2, amt/2]. Затем эти значения вновь "дискретизируются", т.е. подбирается ближайшее к зашумленному значению значение бита. В итоге получается набор сообщений W, содержащий некоторое число ошибок. W подается на вход декодеру.
Очевидно, что при значениях amt < 1 ошибок в W не возникнет.

Для тестирования БЧХ-кодов с разными значениями n и t подбирается такое значение amt уровня шума, чтобы количества ошибок в сообщениях из W были распределены определенным образом относительно t, а именно – чтобы среди всех сообщений в W определенный процент (percent) содержал не более, чем t ошибок. Эта зависимость amt от t для фиксированных значений n и percent, вычислена экспериментальным путём на достаточно большом количестве испытаний (для этого сначала вычисляется зависимость t от amt).

Из описанной выше таблицы БЧХ-кодов для тестов были выбраны n от 7 до 63 и для каждого n - все t, кроме последних значений (т.к. они задают тривиальные коды повторения, у них k=1). Для каждого БЧХ-кода были проведены 3 теста:
1-й тест: для каждого t подбирается такое amt, что примерно в 99% сообщений достаточное для БЧХ-кода (т.е., не превышающее t) количество ошибок (т.е. p=99). Затем в остальных сообщениях искусственно исправляются “лишние” ошибки (в итоге в них будет t ошибок), чтобы в итоге во всех сообщениях было не более, чем t ошибок.
2-й тест: для каждого t подбирается такое amt, что примерно в 90% сообщений достаточное для БЧХ-кода количество ошибок (т.е. p=90).
3-й тест: для каждого t подбирается такое amt, что примерно в 50% сообщений достаточное для БЧХ-кода количество ошибок (т.е. p=50).
То есть, во 2 и 3 тестах декодерам на вход подаются также и сообщения, превышающие их способность декодирования.
Здесь приведены гистограммы распределения количеств ошибок для некоторых n в разных тестах (это – гистограммы того же распределения, что использовалось в тестах, но на другом числе испытаний и с другим результатом генератора случайных чисел).

Эти гистограммы более наглядно показывают отличие трёх вышеописанных тестов друг от друга.



В каждом тесте было сгенерировано, закодировано и раскодировано n_msg=10000 сообщений. Результаты тестов содержатся в таблице ниже.

Пояснения к таблице:

test – номер теста (из тестов, описанных выше)

n_overdrives – кол-во “перегрузов” (т.е. сообщений, в которых больше t ошибок) в n_msg сообщений

pgz/euclid: n_succ, n_err, n_refuse – число успехов, ошибочных раскодирований, отказов (в n_msg сообщений) соответственно при использовании того или иного декодера
pgz/euclid time – время, затраченное декодерами pgz/euclid на раскодирование n_msg сообщений (время, потраченное на операции, не относящиеся к pgz или euclid, сюда не включено)

n	t	test	n_overdrives	pgz n_succ	pgz n_err	pgz n_refuse	euclid n_succ	euclid n_err	euclid n_refuse	pgz_time	euclid_time
7	1	1	0	10000	0	0	10000	0	0	0.053358	0.203457
7	1	2	73	9927	73	0	9927	73	0	0.162503	0.619373
7	1	3	1494	8506	1494	0	8506	1494	0	0.601855	2.322008
15	1	1	0	10000	0	0	10000	0	0	0.110557	0.412387
15	1	2	117	9883	117	0	9883	117	0	0.210682	0.766882
15	1	3	1420	8580	1420	0	8580	1420	0	0.591846	2.271186
15	2	1	0	10000	0	0	10000	0	0	0.491396	0.875439
15	2	2	212	9788	85	127	9788	85	127	1.306225	2.579588
15	2	3	2457	7543	934	1523	7543	934	1523	2.399009	5.650297
15	3	1	0	10000	0	0	10000	0	0	2.051725	2.412509
15	3	2	300	9700	124	176	9700	124	176	3.639027	5.026078
15	3	3	2589	7411	1076	1513	7411	1076	1513	4.854858	8.815676
31	1	1	0	10000	0	0	10000	0	0	0.231470	0.865476
31	1	2	148	9852	148	0	9852	148	0	0.218774	0.814881
31	1	3	1701	8299	1701	0	8299	1701	0	0.665201	2.502017
31	2	1	0	10000	0	0	10000	0	0	0.535488	0.956363
31	2	2	174	9826	73	101	9826	73	101	1.261528	2.501006
31	2	3	3301	6699	1378	1923	6699	1378	1923	2.641430	6.458144
31	3	1	0	10000	0	0	10000	0	0	2.176690	2.694820
31	3	2	360	9640	63	297	9640	63	297	3.669877	5.371347
31	3	3	3777	6223	603	3174	6223	603	3174	5.090748	10.333877
31	5	1	0	10000	0	0	10000	0	0	8.752275	6.543626
31	5	2	379	9621	58	321	9621	58	321	11.367690	11.053808
31	5	3	3507	6493	461	3046	6493	461	3046	11.366011	17.918162
31	7	1	0	10000	0	0	10000	0	0	21.832654	12.029697
31	7	2	645	9355	20	625	9355	20	625	23.801091	19.197422
31	7	3	3419	6581	137	3282	6581	137	3282	20.467444	26.651836
63	1	1	0	10000	0	0	10000	0	0	0.441690	1.593049
63	1	2	566	9434	566	0	9434	566	0	0.416728	1.525151
63	1	3	1782	8218	1782	0	8218	1782	0	0.710524	2.594324
63	2	1	0	10000	0	0	10000	0	0	1.002094	1.863004
63	2	2	419	9581	191	228	9581	191	228	1.597078	3.262952
63	2	3	2710	7290	1306	1404	7290	1306	1404	2.619277	6.263365
63	3	1	0	10000	0	0	10000	0	0	2.832270	3.653784
63	3	2	268	9732	43	225	9732	43	225	3.579413	4.974113
63	3	3	3203	6797	599	2604	6797	599	2604	5.103746	10.196848
63	4	1	0	10000	0	0	10000	0	0	5.425744	5.307448
63	4	2	384	9616	11	373	9616	11	373	6.865499	7.977324
63	4	3	3974	6026	158	3816	6026	158	3816	8.689767	15.465904
63	5	1	0	10000	0	0	10000	0	0	9.130939	7.153699
63	5	2	432	9568	23	409	9568	23	409	11.159973	10.987796
63	5	3	4302	5698	186	4116	5698	186	4116	12.239836	20.515295
63	6	1	0	10000	0	0	10000	0	0	13.800295	8.879763
63	6	2	1171	8829	2	1169	8829	2	1169	16.967506	17.430661
63	6	3	4354	5646	29	4325	5646	29	4325	16.040164	24.986658

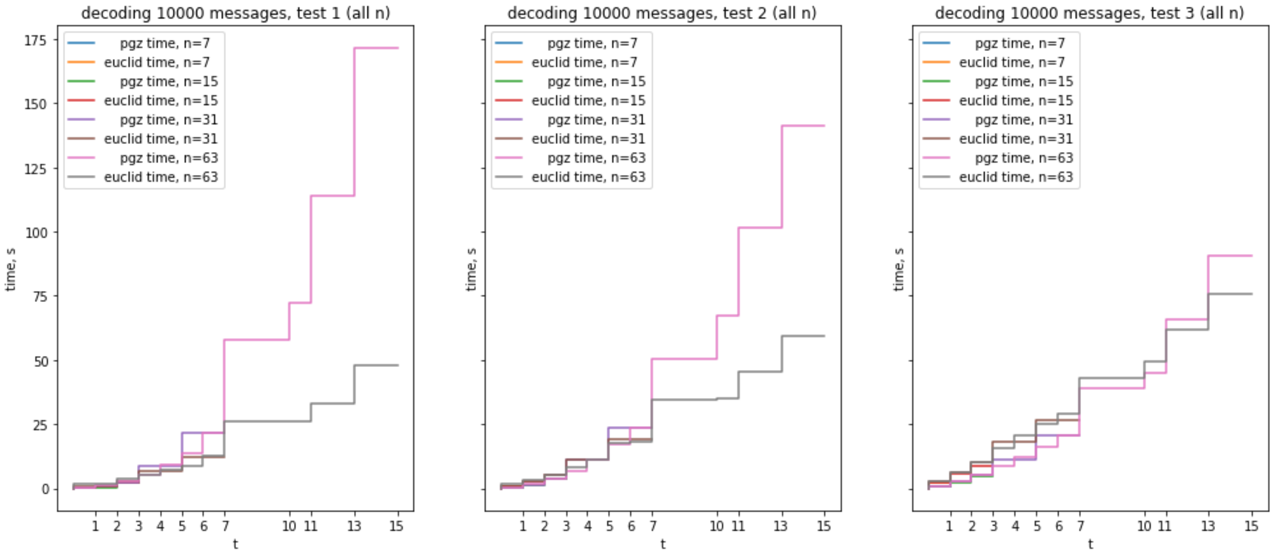
n	t	test	n_overdrives	pgz n_succ	pgz n_err	pgz n_refuse	euclid n_succ	euclid n_err	euclid n_refuse	pgz_time	euclid_time
63	7	1	0	10000	0	0	10000	0	0	21.455190	12.426105
63	7	2	505	9495	1	504	9495	1	504	23.513076	18.162182
63	7	3	4456	5544	5	4451	5544	5	4451	20.547171	29.140253
63	10	1	0	10000	0	0	10000	0	0	57.738851	25.846090
63	10	2	1017	8983	0	1017	8983	0	1017	50.550377	34.331175
63	10	3	3872	6128	6	3866	6128	6	3866	39.170765	42.857307
63	11	1	0	10000	0	0	10000	0	0	72.193035	26.040520
63	11	2	551	9449	0	551	9449	0	551	67.458173	34.993011
63	11	3	4391	5609	14	4377	5609	14	4377	45.061476	49.301864
63	13	1	0	10000	0	0	10000	0	0	114.211208	32.825032
63	13	2	565	9435	1	564	9435	1	564	101.706916	45.579454
63	13	3	4129	5871	3	4126	5871	3	4126	65.875663	61.643344
63	15	1	0	10000	0	0	10000	0	0	171.803122	47.885419
63	15	2	825	9175	0	825	9175	0	825	141.368384	59.544360
63	15	3	4183	5817	0	4183	5817	0	4183	90.551945	75.908159

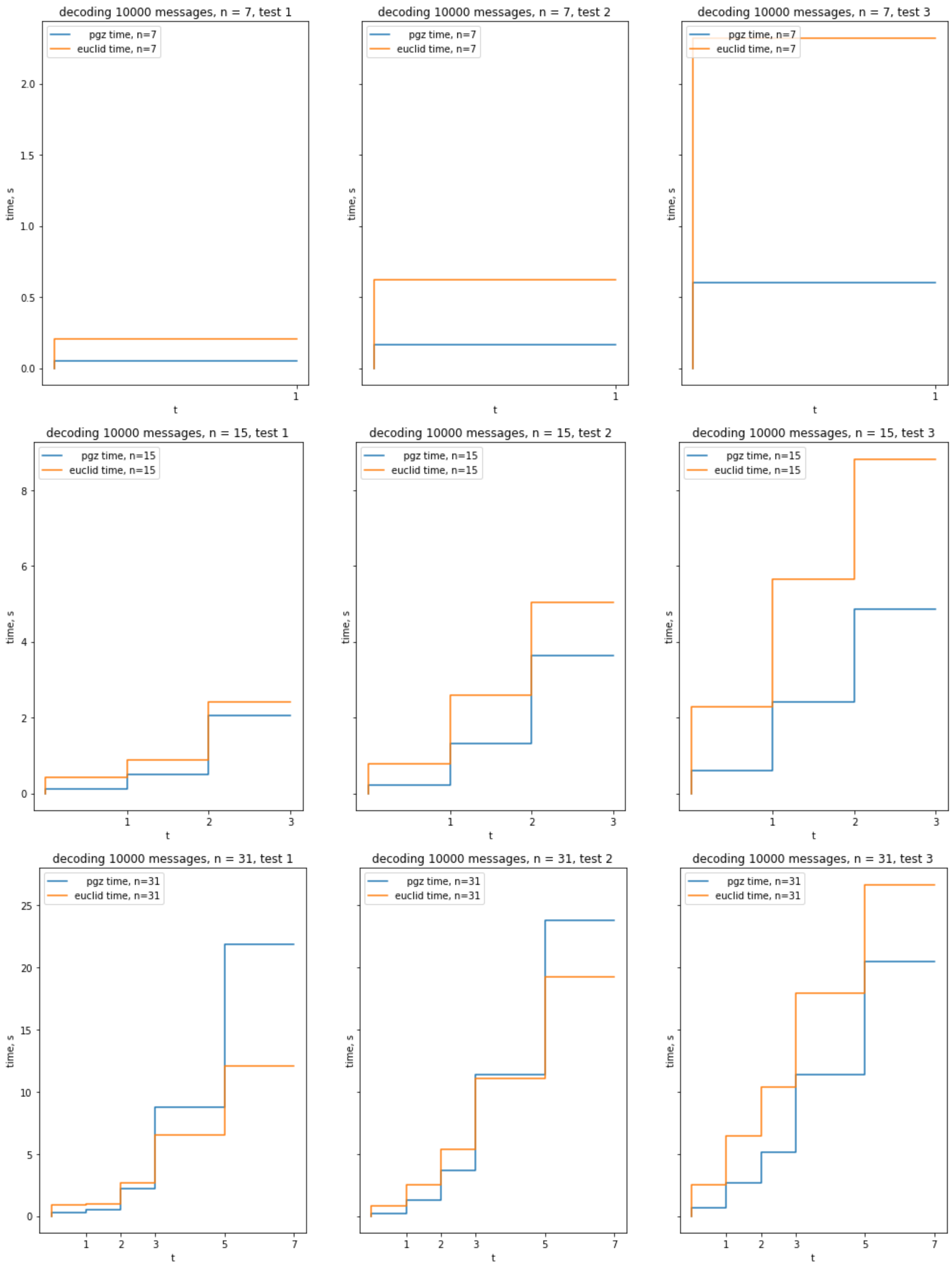
Особенности:

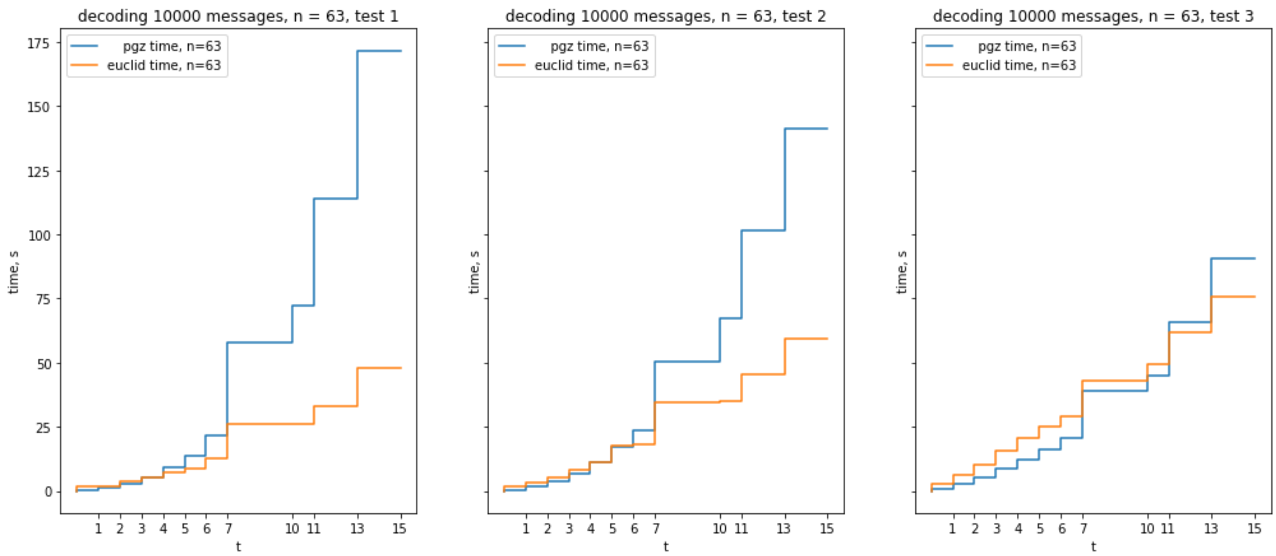
1) Когда ошибок ≤ t, оба декодера всегда правильно раскодируют сообщение

2) Когда ошибок > t, то всегда либо оба декодера ошибочно раскодируют сообщение, либо оба декодера дают отказ (всегда n_err+n_refuse = n_overdrives, причем pgz_n_err = euclid_n_err)

По данным тестов построим графики времени:

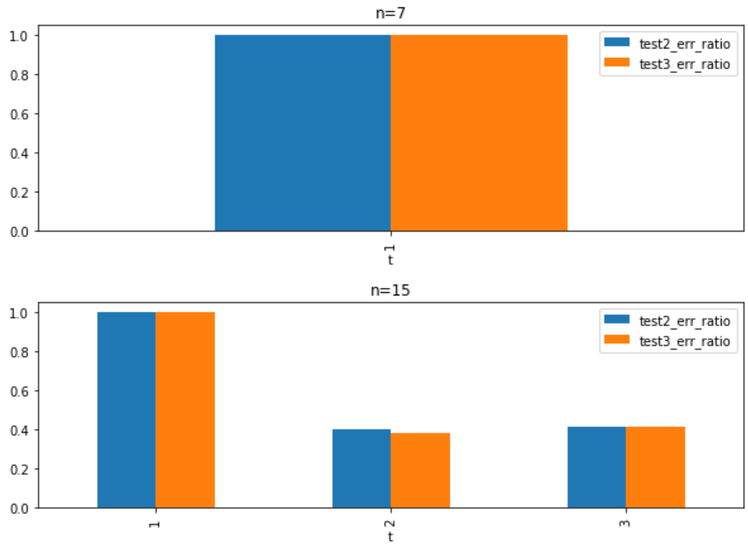


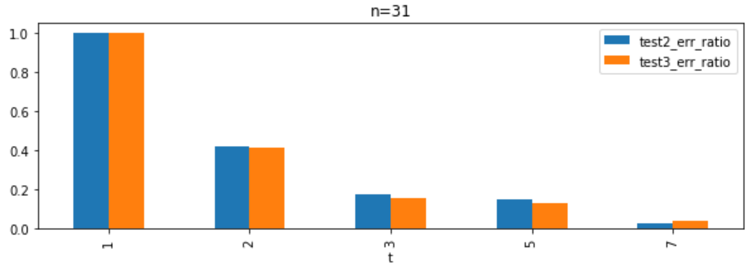

decoding 10000 messages, n = 7, test 1
decoding 10000 messages, n = 7, test 2
decoding 10000 messages, n = 7, test 3
pgz time, n=7
euclid time, n=7
decoding 10000 messages, n = 15, test 1
decoding 10000 messages, n = 15, test 2
decoding 10000 messages, n = 15, test 3
pgz time, n=15
euclid time, n=15
decoding 10000 messages, n = 31, test 1
decoding 10000 messages, n = 31, test 2
decoding 10000 messages, n = 31, test 3
pgz time, n=31
euclid time, n=31
time, s
t

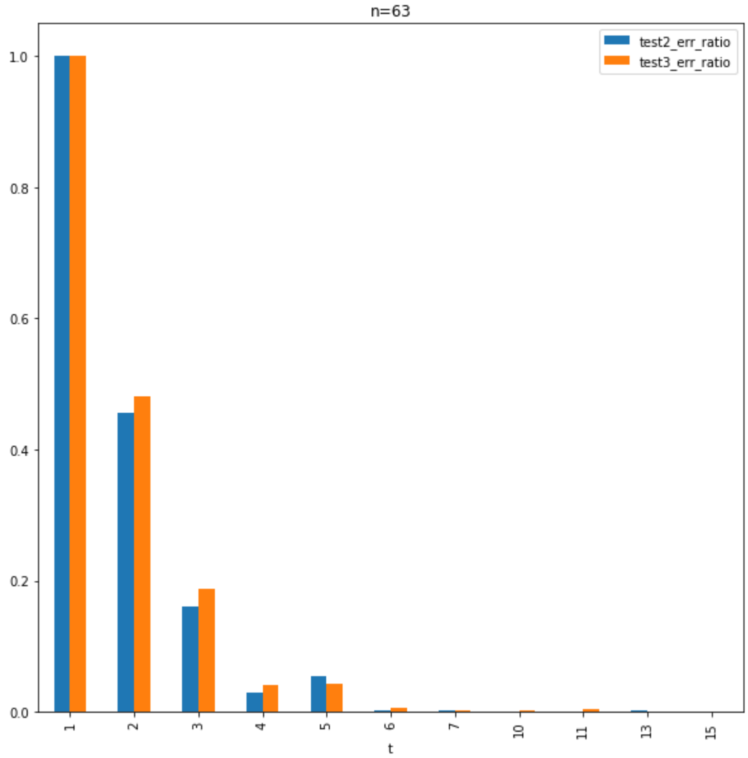


На 1 и 2 тестах (т.е. когда перегрузов нет и когда перегрузов 10%), то для $t \leq 3$ (причем для всех n) euclid отстаёт от pgz, а на $t > 3$ (такое бывает только при n от 31, т.к. тривиальные коды повторения не рассматривались в тестах) он обгоняет pgz, причем при дальнейшем росте t отставание pgz от euclid увеличивается. На 3 тесте (когда перегрузов ~50%, а следовательно, может возникать еще больше ошибок, чем во 2 тесте), euclid медленнее pgz, и только при n=63 обгоняет pgz после t = 11. При n < 63 оба декодера замедляются при росте доли перегрузов, но при n == 63, $t \geq 11$ pgz от этого ускоряется (наверное, раньше находит решение СЛАУ, но потом раскодированное сообщение не проходит проверку на синдромы и делимость на g, и код даёт отказ).
Следовательно, если t подобрано так, что искажения в системе вызывают в основном до t ошибок в сообщении, то для малых t быстрее работает pgz, а для больших t – euclid.

Построим по данным тестов также графики отношения числа ошибочных раскодирований к числу перегрузов (при перегрузах правильного раскодирования не происходит). Отсюда же видно, какое отношение числа отказов к числу перегрузов (т.к. error_ratio + refuse_ratio = 1) Эти данные полностью идентичны для обоих декодеров pgz и euclid, поэтому на графиках они не различаются.







Бросается в глаза особенность, что при любых n, если t = 1, то при возникновении более одной ошибки код всегда неверно раскодирует сообщение и никогда не даёт отказ (наверное, потому, что следующее после t число в этом случае уже в 2 раза больше t). При росте t доля ошибочных раскодирований падает, а доля отказов растет. В данных тестах оно падает до нуля (причем результаты при 10% перегрузов и 50% перегрузов особо не отличаются). Также стоит заметить, что результаты на 2 и 3 тестах слабо различаются. Следовательно, чтобы лучше защититься от случаев неправильного раскодирования (если t подобрано в соответствии с уровнем шума в среде передачи информации, но всё же неизбежно происходят перегрузы), надо брать не слишком малое t.

Вывод:

Для передачи информации в зашумленной среде передачи информации надо выбирать БЧХ-код и декодер в соответствии с характеристиками данной среды (уровень шума), а также желаемыми скоростью, надёжностью (отношением ошибок к отказам при “перегрузах”), размерами сообщений и объёмами вычислительных ресурсов. На тестах с большим количеством испытаний мы убедились, что при корректном для кода числе ошибок код всегда правильно раскодирует сообщение, а при некорректном – либо ошибается, либо отказывается.