Домашние задания по теории кодирования

Марковников Никита Михайлович
группа М4139

Декабрь, 2017

1 Глава 2

1.1 Задача 1

Нужно построить коды длины 6 наилучшие в смысле минимального расстояния. То есть коды над полем F_2^6.

1. $R = \frac{1}{6}$: верхняя граница d в этом случае по неравенству Синглтона равна 6, $G = (1, 1, 1, 1, 1, 1)$;
2. $R = \frac{2}{6}$: верхняя граница d в этом случае равна 5. Пусть $d = 5$, тогда любые 4 столбца проверочное матрицы H линейно-независимы, и существует набор из 5 линейно-зависимых столбцов. Рассмотрим линейную комбинацию $C_5 = \sum_{i=1}^{4} \alpha_i c_i$, где $\alpha_i = 1$ для любого i. В ином случае не все четыре столбца линейно-независимы. Следовательно, $d = 4$.

   Проверочная матрица $H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$. Тогда $G = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}$
3. $R = \frac{3}{6}$. Аналогично предыдущему пункту $d = 3$: $H = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$. Тогда $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$
4. $R = \frac{4}{6}$. Аналогично, $d = 2$. $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$
5. $R = \frac{5}{6}$. По неравенству Синглтона: $d \leq 2$. $G = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$
6. $R = \frac{6}{6} = 1$. По неравенству Синглтона: $d \leq 1$. $G = I_6$

1.2 Задача 2

Дана скорость $R = \frac{1}{2}$. Нужно построить зависимость ошибки от длины кодов при декодировании с исправлением ошибок кратности до $t = [\frac{d-1}{2}]$ для ДСК с переходной вероятностью $p_0 \in \{0.1, 0.01, 0.001\}$. При передаче n бит кода, верная передача происходит тогда, когда $P_C = \sum_{i=0}^{[\frac{d-1}{2}]} \binom{n}{i} p_0^i (1 - p_0)^{n-i}$. Тогда $P_\epsilon = 1 - P_C$. Энергетический выигрыш — это разность энергетических уровней ДСК при декодировании и без нужных для достижения вероятности ошибки p. В нашем случае, $p_0(\frac{E_b}{N_0}) = \Phi(-\sqrt{2}\sqrt{\frac{E_b}{N_0}})$. Подставив это в зависимость вероятности ошибки от переходной вероятности, получим уравнение $P_e(\frac{E_b}{N_0}) = p$.

Таблица 1: Зависимость ошибки от длины кодов

n	k	d	$p = 0.1$	$p = 0.01$	$p = 0.001$
2	1	2	0.190000000	0.019900000	0.001999000
4	2	2	0.343900000	0.039403990	0.003994004
6	3	3	0.114265000	0.001460448	0.000014960
8	4	4	0.186895270	0.002690078	0.000027888
10	5	4	0.263901071	0.004266200	0.000044761
12	6	4	0.340997748	0.006174538	0.000065561
14	7	4	0.415370859	0.008401244	0.000090275
16	8	5	0.210750660	0.000507942	0.000000555
18	9	6	0.266204005	0.000729160	0.000000807
20	10	6	0.323073195	0.001003576	0.000001126
22	11	7	0.171927897	0.000063330	0.000000007
24	12	8	0.214262239	0.000090538	0.000000010
26	13	7	0.259058373	0.000125364	0.000000015
28	14	8	0.305433697	0.000168979	0.000000020
30	15	8	0.352560828	0.000222598	0.000000027
32	16	8	0.399694098	0.000287474	0.000000035
34	17	8	0.446184952	0.000364891	0.000000045
36	18	8	0.491488595	0.000456162	0.000000057

Таблица 2: Энергетический выигрыш

n	k	d	E
2	1	2	3.9
4	2	2	3.3
6	3	3	3.1
8	4	4	5.1
10	5	4	4.9
12	6	4	4.8
14	7	4	4.6
16	8	5	4.5
18	9	6	5.8
20	10	6	5.7
22	11	7	5.6
24	12	8	6.6
26	13	7	5.5
28	14	8	6.4
30	15	8	6.3
32	16	8	6.2
34	17	8	6.2
36	18	8	6.1

Таблица 3: Таблица синдромного декодирования

0000	0000000000
0001	0001010000
0010	0000000100
0011	0001000010
0100	0000010000
0101	0001000000
0110	0000000010
0111	0000100001
1000	0000001000
1001	0000010001
1010	0000100000
1011	0100000000
1100	0000011000
1101	0000000001
1110	0010000000
1111	1000000000

1.3 Задача 3 + 7

$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$ Приведем матрицу к виду $H = (P^T I_r)$. Для этого поменяем 2 и 3 строки местами, затем вычтем из 2 строки 3. Затем, вычтем из 1, 2 и 3 строк 4. Получим матрицу в систематической форме: $H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$. И получим порождающую матрицу в виде $G = (I_k P)$: $G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$. Так, $k = 6$, $n = 10$, скорость кода $R = \frac{6}{10}$. Минимальное расстояние $d = 4$.

1.4 Задача 9

Дана последовательность $\boldsymbol{g} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$. Построим порождающую матрицу $(7, 4)$-кода как набор из четырех циклических сдвигов $\boldsymbol{g}$:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Матрица находится в ступенчатом виде, нулевых строк нет, следовательно ранг матрицу равен 4, а значит число линейно-независимых сток также равно 4, значит кодовые слова матрицы G являются базисом линейного подпространства размерности 4, значит код — линейный. Пусть матрица G порождает кодовое слово $\boldsymbol{c}$. Докажем, что тогда она порождает и циклический сдвиг $\boldsymbol{c'}$.

$$\boldsymbol{c} = \boldsymbol{m}G = (m_1, m_2, m_3, m_4) \cdot G = (m_1, m_1 + m_2, m_2 + m_3, m_1 + m_3 + m_4, m_2 + m_4, m_3, m_4)$$

Значит, $\boldsymbol{c'} = \boldsymbol{m'}G = (m'_1, m'_1 + m'_2, m'_2 + m'_3, m'_1 + m'_3 + m'_4, m'_2 + m'_4, m'_3, m'_4) = (m_4, m_1, m_1 + m_2, m_2 + m_3, m_1 + m_3 + m_4, m_2 + m_4, m_3)$. Так, $\boldsymbol{m'} = (m_4, m_1 - m_4, m_2 + m_4, m_3)$.

Таблица 4: Таблица параметров полученных кодов

n \ k	1	2	3	4	5	6	7	8	9	10	11	12
5	4	2	2	2	2	–	–	–	–	–	–	–
6	4	4	2	2	2	2	–	–	–	–	–	–
7	4	4	4	2	2	2	1	–	–	–	–	–
8	4	4	4	3	3	2	2	2	–	–	–	–
9	4	4	4	3	3	2	2	2	2	–	–	–
10	4	4	4	3	3	3	2	2	2	2	–	–
11	4	4	4	3	3	3	3	3	3	3	3	–
12	4	4	4	3	3	3	3	3	3	3	3	3

Найдем минимальное расстояние кода d. Приведем матрицу G к систематической форме $G = \begin{bmatrix} I_k & P \end{bmatrix}$:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Получим проверочную матрицу:

$$H = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Отсюда видно, что любые 2 столбца матрицы H линейно-независимы. И найдутся 3 линейно-зависимых столбца. Значит, $d = 3$.

1. $k \leq 4$, то, очевидно, $d = 3$;
2. при $k > 4$ и $n = 7$ матрица G не является порождающей;
3. при $k > 4$ и $n < 7$ имеем $d \leq 2$.

1.5 Задача 10

Дана последовательность $\boldsymbol{g} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 \end{pmatrix}$. Нужно, дополняя $\boldsymbol{g}$ нулями построить циклические коды с расстоянием 4. Была получена следующая таблица с помощью программы, написанной на языке Python, код которой представлен ниже:

```
import numpy as np

def next_bit_vector(length):
    import itertools
    return (np.asarray(i) for i in itertools.product([0, 1],
           repeat=length))

def hamming_distance(c1, c2):
    return sum(c1[0, i] != c2[0, i] for i in range(0, c1.shape[1]))

def minimal_distance(matrix):
    G = np.matrix(matrix)
    k, n = G.shape[0], G.shape[1]
    codewords = [m * G % 2for m in next_bit_vector(k)]
    d = n + k + 1
    for i in range(0, len(codewords)):
        for j in range(i + 1, len(codewords)):
            d = min([d, hamming_distance(codewords[i], codewords[j])])
    return d
```

```
def get_cyclic_code(g, k):
    n = len(g)
    return [[g[i - j] for i in range(n)] for j in range(min(k, n))]

def get_cyclic_codes_dists(g, min_k, max_k, min_n, max_n):
    result = [[0 for x in range(max_n - min_k + 1)]
              for y in range(max_n - min_n + 1)]
    for n in range(min_n, max_n + 1):
        current = list(g)
        while len(current) != n:
                current.append(0)
        for k in range(min_k, min(n, max_k) + 1):
            dist = minimal_distance(get_cyclic_code(current, k))
            result[n - min_n][k - min_k] = dist
    return result

if __name__ == '__main__':
    g = [1, 1, 1, 0, 1]
    result = get_cyclic_codes_dists(g,
                                    min_k=1,
                                    max_k=12,
                                    min_n=len(g),
                                    max_n=12)
    print(result)
```

Итого, имеем, что $d = 4$ при:

1. $k = 1$ и $n \geq 5$;
2. $k = 2$ и $n \geq 6$;
3. $k = 3$ и $n \geq 7$;

1.6 Задача 11 (исправление, не было учтено поле)

Из последовательности $\boldsymbol{g} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$ была получена следующая порождающая матрица с параметрами $k = 3, n = 10, d = 5$ с помощью программы, код которой представлен ниже:

$$G = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
import numpy as np

def next_bit_vector(length):
    import itertools
    return (np.asarray(i) for i in itertools.product([0, 1], repeat=length))

def hamming_distance(c1, c2):
    return sum(c1[0, i] != c2[0, i] for i in range(0, c1.shape[1]))

def minimal_distance(matrix):
    G = np.matrix(matrix)
    k, n = G.shape[0], G.shape[1]
    codewords = [m * G % 2 for m in next_bit_vector(k)]
    d = n + k + 1
    for i in range(0, len(codewords)):
        for j in range(i + 1, len(codewords)):
            d = min([d, hamming_distance(codewords[i], codewords[j])])
```

```
    return d

def get_cyclic_code(g, k):
    n = len(g)
    return [[g[i - j] for i in range(n)] for j in range(min(k, n))]

def get_cyclic_codes_dists(g, min_k, max_k, min_n, d):
    for k in range(min_k, max_k + 1):
        current = list(g)
        for n in range(min_n, k * d):
            if k > n: break
            while len(current) != n:
                current.append(0)
            matrix = get_cyclic_code(current, k)
            dist = minimal_distance(matrix)
            if dist == d:
                return matrix
    return None

if __name__ == '__main__':
    g = [0, 0, 1, 0, 0, 1, 0, 1, 1, 1]
    result = get_cyclic_codes_dists(g,
                                    min_k=3,
                                    max_k=10,
                                    min_n=len(g),
                                    d=5)
    print(result)
```

1.7 Задача 12

Даны три последовательности:

1. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$
2. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 0 & 1\end{pmatrix}$
3. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 1 & 1\end{pmatrix}$

Нужно посчитать минимальное расстояние и скорость для кодов, полученных из последовательностей, дополняя их нулями до четной длины $n > 4$ и применяя сдвиг вправо на две позиции. Воспользуемся следующей программой, написанной на языке Python для нахождения минимальное длины:

```
def get_conv_code(g, n):
    assert n % 2 == 0
    current = [(x, False) for x in g] + [(0, False)] * (n - len(g))
    for i in reversed(range(0, n)):
        if current[i][0] == 1:
            current[i] = (1, True)
            break
    temp, result = current, []
    while not temp[-1][1]:
        result.append([x for (x, _) in temp])
        temp = current[-2:] + current[:-2]
        current = temp
    return result

if __name__ == '__main__':
    print(minimal_distance(get_conv_code([1, 1, 0, 1], 12)))
```

```
    print(minimal_distance(get_conv_code([1, 1, 0, 1, 0, 1], 12)))
    print(minimal_distance(get_conv_code([1, 1, 0, 1, 1, 1], 12)))
```

Пусть полученные коды имеют параметры k и n. Затем, что $n = len(g) + 2(k-1)$. Значит, скорость кода $R = \frac{k}{len(g)+2(k-1)}$ Так, получаем следующие результаты:

1. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$, $d = 3$, $R = \frac{k}{2k+2}$;
2. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 0 & 1\end{pmatrix}$, $d = 4$, $R = \frac{k}{2k+4}$;
3. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 1 & 1\end{pmatrix}$, $d = 5$, $R = \frac{k}{2k+4}$;

1.8 Задача 13

Дана скорость кода $R = \frac{1}{2}$. Значит, нужно строить квазициклические $(k, 2k)$-линейные коды. Пусть $G = \begin{bmatrix}I & P\end{bmatrix}$, где I — единичная матрица $k \times k$, а P — матрица $k \times k$, полученная сдвигами на $c > 1$ позиций последовательности g. Так как $R = \frac{1}{2}$, не умаляя общности, возьмем $c = 2$. Воспользуемся следующей программой для нахождения минимального расстояния:

```
def get_quasicyclic_code(g, n, k, c):
    assert n % 2 == 0
    current = [(x, False) for x in g] + [(0, False)] * (n - len(g))
    for i in reversed(range(0, n)):
        if current[i][0] == 1:
            current[i] = (1, True)
            break
    temp, result = current, []
    while len(result) < k:
        result.append([x for (x, _) in temp])
        temp = current[-c:] + current[:-c]
        current = temp
    return result

if __name__ == '__main__':
    g = [1, 1, 0, 1]
    for k in range(2, 20):
        code = get_quasicyclic_code(g=g,
                                    n=2*k,
                                    k=k,
                                    c=2)
        print("k = {}, d = {}".format(k, minimal_distance(code)))
```

Получили следующие результаты:

1. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1\end{pmatrix}$:
   - $k = 2$: $d = 2$;
   - $k > 2$: $d = 3$;
2. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 0 & 1\end{pmatrix}$:
   - $k \in \{2, 3\}$: $d = 2$;
   - $k > 3$: $d = 4$;
3. $\boldsymbol{g} = \begin{pmatrix}1 & 1 & 0 & 1 & 1 & 1\end{pmatrix}$:
   - $k \in \{2, 3, 4\}$: $d = 2$;
   - $k \in \{5, 6\}$: $d = 3$;
   - $k \in \{7, 8\}$: $d = 4$;
   - $k > 8$: $d = 5$

1.9 Задача 14

Для нахождения радиуса покрытия воспользуемся формулой:

$$\rho = \max_s \rho(s)$$

где $\rho(s)$ обозначает минимальный вес вектора ошибок $\boldsymbol{e}$ такого, что ему соответствует синдром $\boldsymbol{e}H^T = s$

```
def sindrome(check_matrix):
    H = np.matrix(check_matrix)
    H = np.matrix(H.transpose())
    sindrom_dict = {}
    for error_vector in next_bit_vector(H.shape[0]):
        s = error_vector * H % 2
        s = np.asarray(s).reshape(-1)
        str_s = str(s)
        if str_s not in sindrom_dict or \
                    np.sum(sindrom_dict[str_s]) > np.sum(error_vector):
            sindrom_dict[str_s] = error_vector
    return sindrom_dict


def get_radius(matrix):
    s = sindrome(matrix)
    return max([np.sum(s[str(v)]) for v in next_bit_vector(len(matrix))])


if __name__ == '__main__':
    H = None
    print(get_radius(H))
```

1. $H = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$. $\rho = 3$.
2. $H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$. $\rho = 3$.
3. $H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$. $\rho = 2$
4. $H = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$. $\rho = 2$
5. $H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$. $\rho = 1$.
6. $G = I_6 \Rightarrow \rho = 0$

2 Глава 3

2.1 Задача 1

Нужно доказать, что коды Хэмминга и код Голея при $d = 7$ удовлетворяют границе Хэмминга с равенством. Для любого q-ичного кода, исправляющего t ошибок, с длиной n и минимальным

расстоянием $d \geq 2t+1$ число кодовых слов M удовлетворяет неравенству

$$M \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i}(q-1)^i}$$

2.1.1 Код Хэмминга.

Кодами Хэмминга называются коды, исправляющие любые однократные ошибки ($t=1$), такие что при числе проверочных символов r длина кода равна $n = 2r-1$, а число информационных символов равно $k = 2^r - r - 1$. Подставим данные параметры в формулу:

$$M = 2^k = 2^{2^r-r-1} \leq \frac{2^{2r-1}}{\sum_{i=0}^{1} \binom{2r-1}{i}} = \frac{2^{2r-1}}{1+2^r-1} = 2^{2^r-r-1}$$

2.1.2 Код Голея.

Рассмотрим код Голея с параметрами: $n=23$, $k=12$, $d=7$, $t=3$. Подставим данные параметры в формулу:

$$M = 2^k = 2^{12} = \frac{2^{23}}{\binom{23}{0}+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}} = \frac{2^{23}}{2^{11}} = 2^{12}$$

Действительно, коды Хэмминга и код Голея при $d=7$ удовлетворяют границе Хэмминга с равенством.

2.2 Задача 3

Вариант 95: $n=18$, $k=6$, $d=6$. Считаем, что $q=2$

2.2.1 Граница Хэмминга.

Воспользуемся границей Хэмминга:

$$M \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i}(q-1)^i}$$

Так как $d \geq 2t+1$, то $t=2$ удовлетворяет этому условию и даст наиболее точную границу Хэмминга. Подставим данные параметры:

$$M = 2^k \leq \frac{2^{18}}{\sum_{i=0}^{2} \binom{18}{i}} = \frac{2^{18}}{1+18+153} \approx 1524.093$$

Значит, получаем

$$k \leq \left[\log_2\left(\frac{2^{18}}{172}\right)\right] = 10$$

2.2.2 Граница Варшамова–Гилберта.

Воспользуемся границей Варшамова–Гилберта для нахождения нижней оценки:

$$q^{n-k} > \sum_{i=0}^{d-2} \binom{n-1}{i}(q-1)^i \Rightarrow$$

$$2^{18-k} > \sum_{i=0}^{4} \binom{17}{i} = 1+17+136+680+2380 = 3214 \Rightarrow$$

$$k < 6.35$$

Значит, при $k=6$ существует двоичный линейный данными параметрами, следовательно, $k \geq 6$.

2.2.3 Граница Грайсмера.

Воспользуемся границей Грайсмера:

$$N(k,d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil$$

- $k = 6 \Rightarrow n \geq 14$;
- $k = 7 \Rightarrow n \geq 15$;
- $k = 8 \Rightarrow n \geq 16$;
- $k = 9 \Rightarrow n \geq 17$;
- $k = 10 \Rightarrow n \geq 18$;

Но, уточнение границ мы не получили.

2.2.4 Граница Плоткина.

$$\frac{k}{n} \leq 1 - 2\frac{d}{n} \Rightarrow k \leq n - 2d = 18 - 2 \cdot 6 = 6$$

2.2.5 Итого.

В итоге, получили $k = 6$.

2.3 Задача 4

Вариант 95: $n = 18$, $k = 6$, $d = 6$. Считаем, что $q = 2$

2.3.1 Граница Хэмминга.

Воспользуемся границей Хэмминга:

$$M \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i}(q-1)^i} \Rightarrow$$

$$M = 2^6 \leq \frac{2^{18}}{\sum_{i=0}^{t} \binom{18}{i}} \Rightarrow$$

$$\sum_{i=0}^{t} \binom{18}{i} \leq 2^{12}$$

Максимальное $t \geq 0$, при котором неравенство выполняется, равно 4 и $d \geq 2t+1$, значит $d \leq 10$.

2.3.2 Граница Варшамова–Гилберта.

Воспользуемся границей Варшамова–Гилберта для нахождения нижней оценки:

$$q^{n-k} > \sum_{i=0}^{d-2} \binom{n-1}{i}(q-1)^i \Rightarrow$$

$$2^{12} > \sum_{i=0}^{d-2} \binom{17}{i} \Rightarrow d \geq 6$$

2.3.3 Граница Грайсмера.

Воспользуемся границей Грайсмера:

$$N(k,d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{2^i} \right\rceil \Rightarrow$$

$$18 \geq \sum_{i=0}^{5} \left\lceil \frac{d}{2^i} \right\rceil \Rightarrow d \leq 8$$

2.3.4 Граница Плоткина.

$$\frac{k}{n} \leq 1 - 2\frac{d}{n} \Rightarrow d \leq \frac{n-k}{2} = 6$$

2.3.5 Итого.

В итоге, получили $d = 6$.

3 Глава 3

3.1 Задача 5.

Для нахождения границ минимального расстояния кода d с помощью границ Хэмминга (оценка сверху) и Варшамова–Гилберта (оценка снизу) воспользуемся следующей программой, написанной на языке Python:

```
from scipy.special import binom
import math

def hamming_bound(n, k, q):
    M = math.pow(q, k)
    for t in range(0, 10000):
        bound = math.pow(q, n) / \
                (sum(binom(n, i) *
                    math.pow(q - 1, i) for i in range(0, t + 1)))
        if bound < M:
            d = (t - 1.) * 2. + 1
            return d if d % 2 == 0 else d + 1
    return None

def vg_bound(n, k, q):
    M = math.pow(q, n - k)
    for d in range(0, 10000):
        bound = sum(binom(n - 1, i) *
            math.pow(q - 1, i) for i in range(0, d - 1))
        if M <= bound:
            return d - 1
    return None

if __name__ == "__main__":
    for n, k, d in params:
        print("{}\t&\t{}\t&\t{}\t&\t{}\t&\t{}".
        format(n, k, d, hamming_bound(n, k, 2), vg_bound(n, k, 2)))
```

В итоге получили следующие результаты:

Таблица 5: Таблица границ Хэмминга (a) и Варшамова-Гилберта (b) для лучших кодов со скоростью $R = \frac{1}{2}$

n	k	d	(a)	(b)
8	4	4	4	3
10	5	4	4	3
12	6	4	4	3
14	7	4	6	4
16	8	5	6	4
18	9	6	6	4
20	10	6	6	4
22	11	7	8	5
24	12	8	8	5
26	13	7	8	5
28	14	8	8	5
30	15	8	10	6
32	16	8	10	6
34	17	8	10	6
36	18	8	10	6
38	19	$8-9$	10	7
40	20	$8-9$	12	7

3.2 Задача 7.

Для нахождения оценок длин оптимальных кодов с минимальным расстоянием $d = 8$, используя границу Грайсмера, воспользуемся следующей программой, написанной на языке Python:

```
from scipy.special import binom
import math

def graimer_bound(k, d):
    return sum(math.ceil(d / math.pow(2, i)) for i in range(0, k))

if __name__ == "__main__":
    for k, n in params:
        print("{}\t&\t{}\t&\t{}".format(k, n, graimer_bound(k, d=8)))
```

В итоге получили следующие результаты:

Таблица 6: Таблица оценок Грайсмера при $d = 8$

k	n	Оценка
3	14	14
4	15	15
5	16	16
6	18	17
7	19	18
8	20	19
9	21	20
10	22	21
11	23	22
12	24	23
13	27	24
14	28	25
15	30	26
16	31	27
17	32	28
18	34	29
19	35	30
20	36	31
21	37	32
22	38	33
23	40	34

4 Глава 4

5 Задача 2.

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Приведем матрицу к виду $H = (P^T I_r)$. Для этого поменяем 2 и 3 строки местами, затем вычтем из 2 строки 3. Затем, вычтем из 1, 2 и 3 строк 4. Получим матрицу в систематической форме:

$$H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

И получим порождающую матрицу в виде $G = (I_k P)$:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Далее представлена полученная программа для моделирования ДСК и АГБШ канала с МП декодированием, написанная на языке Python:

```
# coding=utf-8
import numpy as np
import math
from abc import ABC, abstractmethod
import random
```

```
import matplotlib.pyplot as plt
from scipy.stats import norm

def next_bit_vector(length):
    import itertools
    return (np.asarray(i) for i in
    itertools.product([0, 1], repeat=length))

def hamming_distance(c1, c2):
    return sum(c1[i] != c2[i] for i in range(0, c1.shape[0]))

class Decoder(ABC):
    def __init__(self, G):
        self.G = np.matrix(G)
        self.k, self.n = self.G.shape[0], self.G.shape[1]

    @abstractmethod
    def decode(self, y):
        pass

class MLDecoder(Decoder):
    def __init__(self, G):
        super().__init__(G)
        self.C = [np.asarray(m * self.G % 2).reshape(-1)
        for m in next_bit_vector(self.k)]
        self.ξ = [2. * cm - 1. for cm in self.C]

    @abstractmethod
    def L(self, y) -> np.ndarray:
        pass

    def decode(self, y):
        L = [np.dot(ξm, self.L(y)) for ξm in self.ξ]
        return list(next_bit_vector(self.k))[np.argmax(L)]

class BSCMLDecoder(MLDecoder):
    def __init__(self, G, p0):
        super().__init__(G)
        self.p0 = p0

    def L(self, y) -> np.ndarray:
        δ = math.log((1. - self.p0) / self.p0)
        return (2. * np.asarray(y) - 1.) * δ

class AWGNMLDecoder(MLDecoder):
    def __init__(self, G):
        super().__init__(G)

    def L(self, y) -> np.ndarray:
        return np.asarray(y)

BSCMADecoder = BSCMLDecoder
AWGNMADecoder = AWGNMLDecoder

class Channel(ABC):
    def __init__(self, G):
        self.G = np.matrix(G)
        self.k, self.n = self.G.shape[0], self.G.shape[1]
```

```
    @abstractmethod
    def send(self, m):
        pass

class BSC(Channel):
    def __init__(self, G, p0):
        super().__init__(G)
        self.p0 = p0

    def send(self, m):
        x = np.asarray(np.asarray(m) * self.G % 2).reshape(-1)
        random.seed()
        return [xi if random.random() >= self.p0 else 1 - xi for xi in x]

class AWGNC(Channel):
    def __init__(self, G, N):
        super().__init__(G)
        self.N = N

    def send(self, m):
        x = np.asarray(np.asarray(m) * self.G % 2).reshape(-1)
        random.seed()
        return [(2. * xi - 1.) + random.gauss(0, 1) *
        math.sqrt(self.n / (2. * self.k * self.N)) for xi in x]

def get_db(x):
    return 10. * math.log10(x)

def model(matrix, n, channel_creator, decoder_creator,
start=0., end=6.6, step=0.1, bound=1e-5):
    random.seed()
    xs, ys = [], []
    k = len(matrix)
    messages = list(next_bit_vector(k))

    def find_without_decoding(l, r, eps, p):
        m = (l + r) / 2.
        if r - l < eps: return m
        value = 1. - norm.cdf(math.sqrt(2. * m))
        return find_without_decoding(l, m, eps, p)
        if p > value else find_without_decoding(m, r, eps, p)

    print("Bound without decoding: {}".format(
    get_db(find_without_decoding(0., 10., 1e-7, bound))))

    for en in np.arange(start, end, step):
        xs.append(get_db(en))
        channel = channel_creator(matrix, en)
        decoder = decoder_creator(matrix, en)
        ys.append(
            sum(hamming_distance(m, decoder.decode(channel.send(m)))
                for m in random.choices(messages, k=n)) / (n * k))

        if abs(ys[-1]) <= bound:
            print("Bound with decoding: {}".format(xs[-1]))

        print("Step {} of {}".format(len(xs), (end - start) / step))
```

```
    return xs, ys

def draw_plot(xs, ys, filename):
    plt.plot(xs, ys)
    plt.xlabel('Eb / N0 (Db)')
    plt.ylabel('Bit error probability')
    plt.grid(True)
    plt.savefig(filename)
    plt.show()

if __name__ == '__main__':
    G = [[1, 0, 0, 0, 0, 0, 0, 1, 0, 1],
         [0, 1, 0, 0, 0, 0, 0, 0, 1, 1],
         [0, 0, 1, 0, 0, 0, 1, 0, 1, 0],
         [0, 0, 0, 1, 0, 0, 1, 0, 0, 1],
         [0, 0, 0, 0, 1, 0, 1, 1, 0, 0],
         [0, 0, 0, 0, 0, 1, 0, 1, 1, 0]]

    n = 100_000

    xs, ys = model(matrix=G,
                   n=n,
                   channel_creator=lambda G, en:
                       BSC(G, norm.cdf(math.sqrt(2. * en))),
                   decoder_creator=lambda G, en:
                       BSCMLDecoder(G, norm.cdf(math.sqrt(2. * en))))
    draw_plot(xs, ys, filename="bsc_ml_{}.png".format(n))

    xs, ys = model(matrix=G,
                   n=n,
                   channel_creator=lambda G, en: AWGNC(G, en),
                   decoder_creator=lambda G, en: AWGNMLDecoder(G))
    draw_plot(xs, ys, filename="awgn_ml_{}.png".format(n))
```

На Рисунке 1 представлен полученный график для ДСК, а на Рисунке 2 — для АБГШ канала.

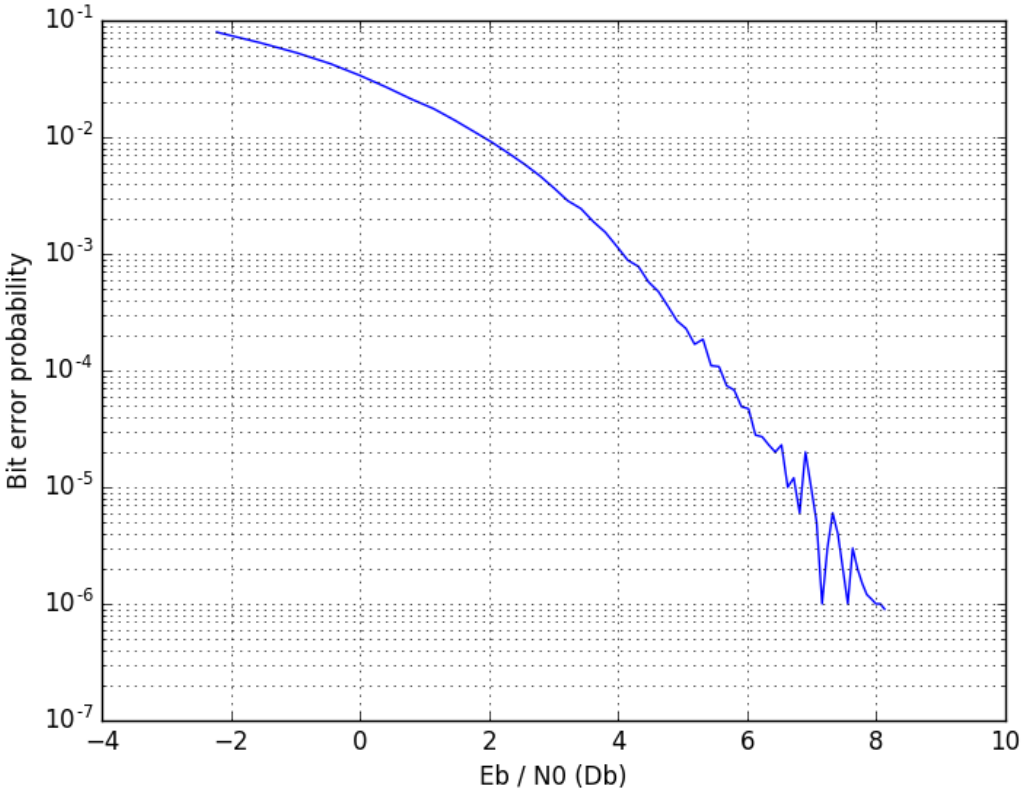

Рис. 1: График для ДСК

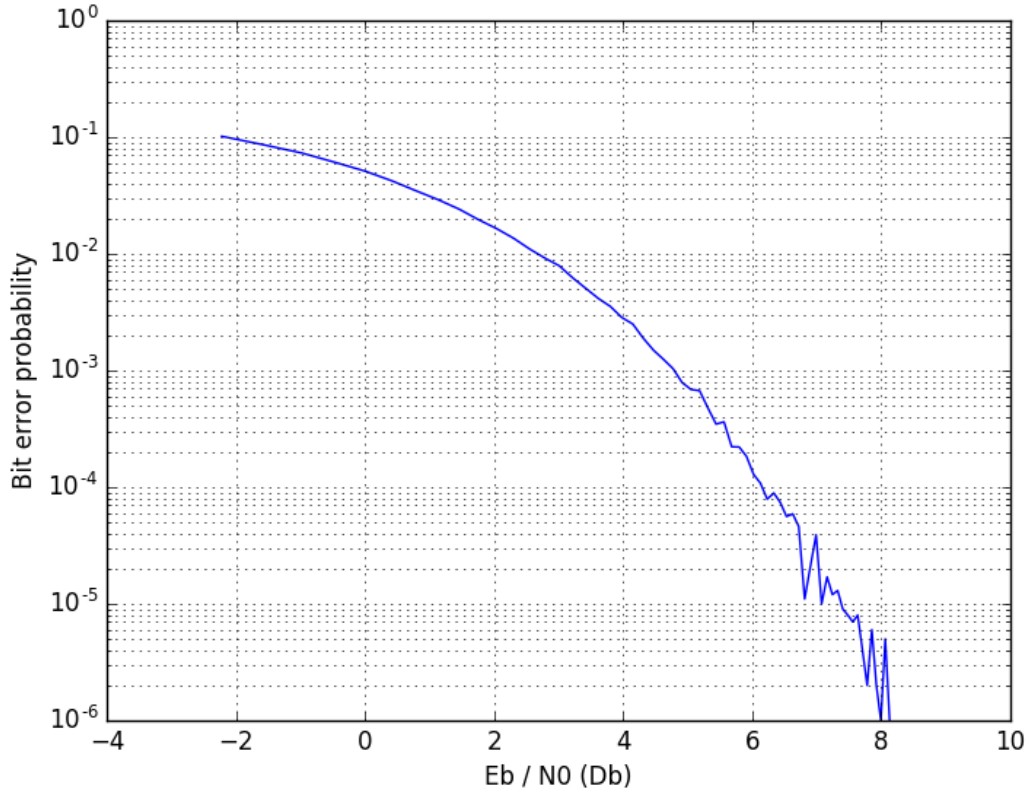

Рис. 2: График для АБГШ канала.

Из графиков и с помощью программы можно найти, что при передаче без декодирования требуется $\frac{E_b}{N_0} = 9.59$ Дб и при вероятности ошибки на бит 10^{-5}:

- для ДСК при передаче с кодированием требуется $\frac{E_b}{N_0} = 6.63$ Дб, так энергетический выигрыш равен $9.59 - 6.63 = 2.96$ Дб;
- для АБГШ канала при передаче с кодированием требуется $\frac{E_b}{N_0} = 7.08$ Дб, так энергетический выигрыш равен $9.59 - 7.08 = 2.51$ Дб.

5.1 Задача 4.

Проверочная матрица в систематической форме:

$$H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Порождающая матрица в систематической форме:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Приведем матрицу G к МСФ:

$$G = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

И построим следующую решетку:

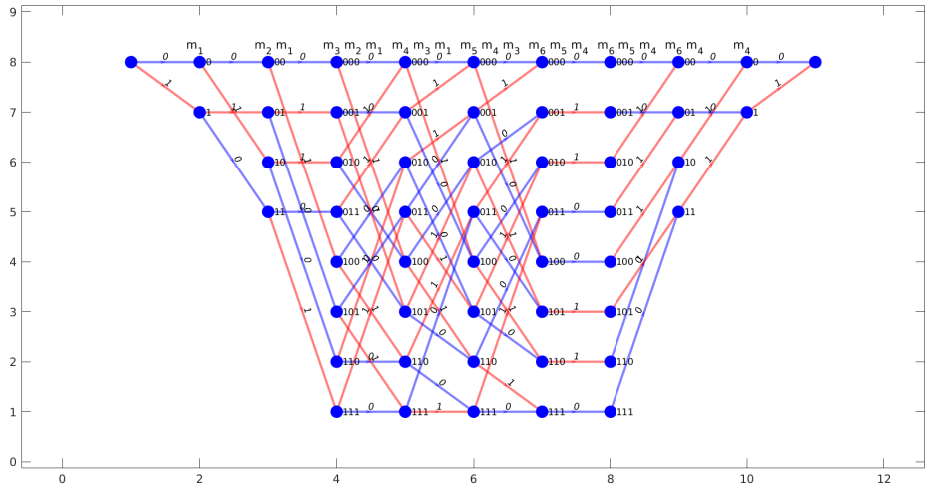

Рис. 3: Решетка по порождающей матрице в МСФ

Построим также следующую синдромную решетку:

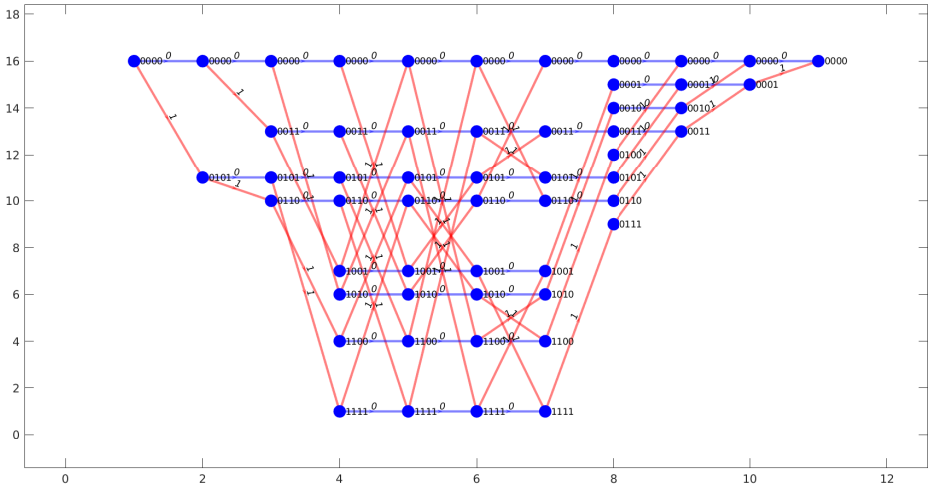

Рис. 4: Синдромная решетка

6 Глава 7

6.1 Задача 1.

Дана последовательность символов:

$$[0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1]$$

Воспользуемся следующей программой, написанной на языке Python:

```
# coding=utf-8
import numpy as np

def seq2pol(s):
    return '+'.join(["x^{}".format(i) for i in range(0, s.shape[0])
        if s[i] != 0])

def bm_algorithm(S, c):
    s = np.asarray(S)
    L = 0
    Λ = np.asarray([1.])
    B = np.asarray([1.])
    d = s.shape[0]
    for i in range(0, d):
        r = i + 1
        Δ = sum(Λ[j] * s[r - j - 1] for j in range(0, L + 1)) % 2.
        B = np.insert(B, 0, [0.])
        if Δ != 0.:
            T = (np.append(Λ, [0.] * (B.shape[0] - Λ.shape[0])) - Δ * B)
            % 2
            if 2. * L <= r - 1:
                B = Λ / Δ
                L = r - L
            Λ = T
        print("${}$ &${}$ &${}$ &${}$ &${}$ &${}$ \\\\".
        format(r, s[i], int(Δ), seq2pol(B), seq2pol(Λ), L))

    result = S
    for i in range(0, c):
        result.append(int(sum(Λ[j + 1] *
        result[len(result) - j - 1] for j in range(0, L)) % 2))
    return result


if __name__ == "__main__":
    print(bm_algorithm([0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1], 10))
```

Таблица 7: Применение алгоритма Берлекэмпа–Месси

r	s	Δ	$B(x)$	$\Lambda(x)$	L
0	—	0	1	1	0
1	0	0	x^1	x^0	0
2	1	1	x^0	$x^0 + x^2$	2
3	0	0	x^1	$x^0 + x^2$	2
4	0	1	x^2	x^0	2
5	1	1	x^0	$x^0 + x^3$	3
6	1	1	x^1	$x^0 + x^1 + x^3$	3
7	0	1	$x^0 + x^1 + x^3$	$x^0 + x^1 + x^2 + x^3$	4
8	0	0	$x^1 + x^2 + x^4$	$x^0 + x^1 + x^2 + x^3$	4
9	0	1	$x^0 + x^1 + x^2 + x^3$	$x^0 + x^1 + x^5$	5
10	1	0	$x^1 + x^2 + x^3 + x^4$	$x^0 + x^1 + x^5$	5
11	1	1	$x^0 + x^1 + x^5$	$x^0 + x^1 + x^2 + x^3 + x^4$	6
12	1	1	$x^1 + x^2 + x^6$	$x^0 + x^3 + x^4 + x^6$	6

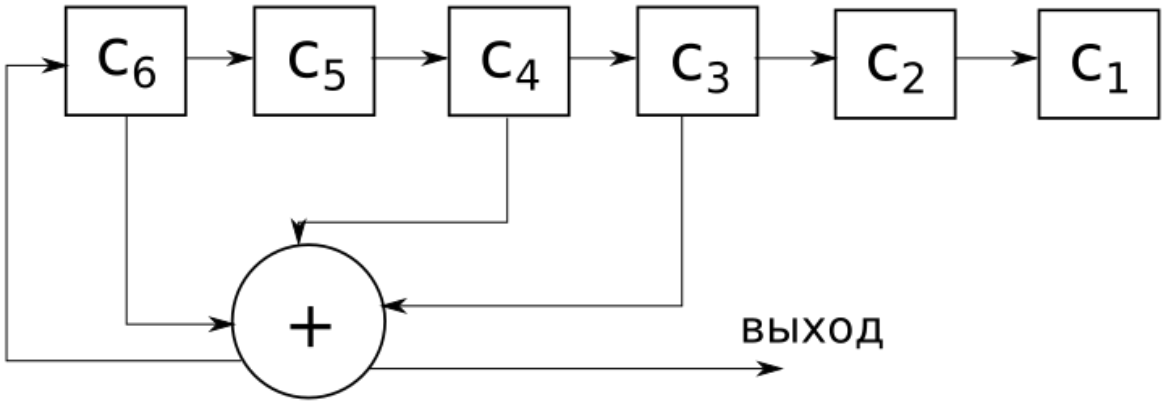

Рис. 5: Полученная схема

Продленная на 10 символов последовательность выглядит следующим образом:

$$[0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0]$$

6.2 Задача 6.

Примитивный полином $73_8 = 111011_2$: $p(x) = 1 + x + x^2 + x^4 + x^5$

Выход канала $4602671437_8 = 100110000010110111001100011111_2$: $v(x) = x + x^4 + x^5 + x^{11} + x^{13} + x^{14} + x^{16} + x^{17} + x^{18} + x^{21} + x^{22} + x^{26} + x^{27} + x^{28} + x^{29} + x^{30}$

Найдем циклическое представление мультипликативной группы поля:

Таблица 8: Мультипликативая группа поля

Степень	Многочлен	Последовательность
$-\infty$	0	00000
0	1	00001
1	x	00010
2	x^2	00100
3	x^3	01000
4	x^4	10000
5	$x^4 + x^2 + x + 1$	10111
6	$x^4 + x^3 + 1$	11001
7	$x^2 + 1$	00101
8	$x^3 + x$	01010
9	$x^4 + x^2$	10100
10	$x^4x^3x^2 + x + 1$	11111
11	$x^3 + 1$	01001
12	$x^4 + x$	10010
13	$x^3 + x + 1$	10011
14	$x^4 + 1$	10001
15	$x^4 + x^2 + 1$	10101
16	$x^4 + x^3 + x^2 + 1$	11101
17	$x^3 + x^2 + 1$	01101
18	$x^4 + x^3 + x$	11010
19	$x + 1$	00011
20	$x^2 + x$	00110
21	$x^3 + x^2$	01100
22	$x^4 + x^3$	11000
23	$x^2 + x + 1$	00111
24	$x^3 + x^2 + x$	01110
25	$x^4 + x^3 + x^2$	11100
26	$x^3 + x^2 + x + 1$	01111
27	$x^4 + x^3 + x^2 + x$	11110
28	$x^3 + x + 1$	01011
29	$x^4 + x^2 + x$	10110
30	$x^4 + x^3 + x + 1$	11011

Воспользуемся алгоритмом Питерсона-Горенстейна-Цирлера. Найдем синдромный многочлен:

$$S(x) = x(a^2 + x + 1) + x^2(a^4 + a^2 + 1) + x^3(a^2 + x + 1) + x^4(a^4 + a^3 + x + 1)$$

Запишем систему уравнений для коэффициентов многочлена локаторов ошибок:

$$\begin{bmatrix} a^2 + a + 1 & a^4 + a^2 + 1 \\ a^4 + a^2 + 1 & a^2 + a + 1 \end{bmatrix} \begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} a^2 + a + 1 \\ a^4 + a^3 + a + 1 \end{bmatrix}$$

Решив систему, получим полином локаторов ошибок

$$\Lambda(x) = (a^4 + a^2)x^2 + (a^2 + a + 1)x + 1$$

Найдем корни $\Lambda(x)$:

$$x_1 = a^3 + a^2 + a + 1$$
$$x_2 = a^4 + a^3 + a^2 + a$$

Локаторы ошибок:

- $a^4 + a^2 + a + 1$, степени 5
- a^4, степени 4

Так, позиций ошибок равны 5 и 4. Запишем систему уравнений для значений ошибок:

$$\begin{bmatrix} a^4+a^2+a+1 & a^4 \\ a^4+a^3+a^2+a+1 & a^3+a \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} a^2+a+1 \\ a^4+a^2+1 \end{bmatrix}$$

Решив систему, получим, что $Y_1 = 1$ и $Y_2 = 1$. Так, получили вектор ошибок $e(x) = x^4 + x^5$. Вычтем из $v(x)$ полученный вектор $e(x)$:

$$c(x) = x + x^{11} + x^{13} + x^{14} + x^{16} + x^{17} + x^{18} + x^{21} + x^{22} + x^{26} + x^{27} + x^{28} + x^{29} + x^{30}$$

Порождающий многочлен

$$g(x) = 1 + a + a^2 + a^3 + a^4 + ax + a^2x^2 + x^3(a^4 + a^3 + a^2 + a) + x^4$$

Теперь, для расчета полинома локаторов ошибок воспользуемся алгоритмом Берлекэмпа–Месси:

Таблица 9: Применение алгоритма Берлекэмпа–Месси

r	s	Δ	$B(x)$	$\Lambda(x)$	L
0	—	0	1	1	0
1	0	a^2+a+1	a^3+a	$x(a^2+a+1)+1$	1
2	1	0	$(a^3+a)x$	$x(a^2+a+1)+1$	1
3	0	a	$(a^4+a^3)x+a^4+a^3+a+1$	$(a^4+a^2)x^2+x(a^2+a+1)+1$	2
4	0	0	$(a^4+a^3)x^2+x(a^4+a^3+a+1)$	$(a^4+a^2)x^2+x(a^2+a+1)+1$	2

Видим, что многочлены локаторов ошибок, полученных с помощью двух алгоритмов, совпали. Далее, алгоритм повторяет действия алгоритма Питерсона-Горенстейна-Цирлера, то есть получим, что $v(x) = c(x) + e(x)$.

7 Глава 8

7.1 Задачи 5 и 6.

Вариант 95: дан свёрточный код с порождающими многочленами $(7, 5, 1)$ (в двоично-восьмеричной форме).

$$\boldsymbol{g}_1 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$
$$\boldsymbol{g}_2 = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$$
$$\boldsymbol{g}_3 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

На Рисунке 6 представлена схема данного свёрточного кодера:

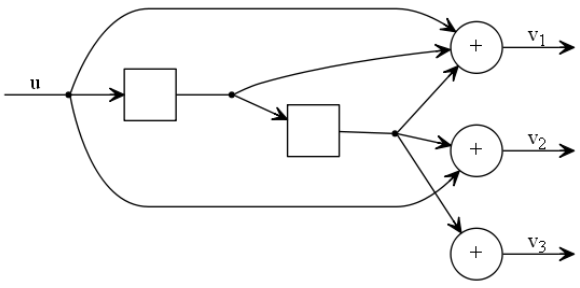

Рис. 6: Схема свёрточного кодера

На Рисунке 7 представлена схема конечного автомата соответсвующего кодеру:

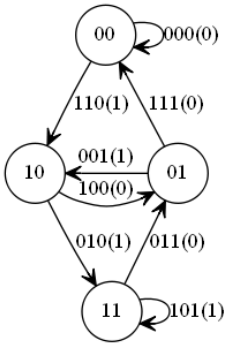

Рис. 7: Схема конечного автомата

На Рисунке 8 представлена схема конечного автомата соответсвующего кодеру с разметкой, учитывающей веса ребер и информационных последовательностей:

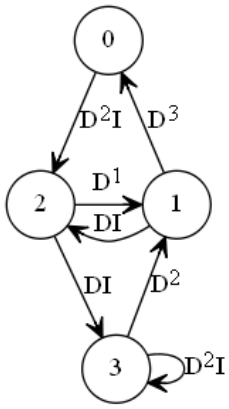

Рис. 8: Разметка, учитывающая веса ребер и информационных последовательностей

Получаем систему уравнений:

$$\begin{cases} g_0(D,I) = D^3 g_1(D,I) \\ g_1(D,I) = D g_2(D,I) + D^2 g_3(D,I) \\ g_2(D,I) = DI g_1(D,I) + D^2 I \\ g_3(D,I) = D^2 I g_3(D,I) + DI g_2(D,I) \end{cases}$$

Решив систему, получим:

$$T(D,I) = g_0(D,I) = -\frac{D^6 I(1 - D^2 I + D^3 I)}{-1 + 2D^2 I - D^4 I^2 + D^5 I^2} =$$

$$= D^6 I + D^8 I^2 + D^9 I^2 + D^{10} I^3 + 3D^{11} I^3 + D^{12} I^4 + 6D^{13} I^4 + D^{14} I^4 (I+1) + 10D^{15} I^5 + D^{16} I^5 (I+5) + \dots$$

$$T(D) = T(D,I)|_{I=1} = D^6 + D^8 + D^9 + D^{10} + 3D^{11} + D^{12} + 6D^{13} + D^{14} + 10D^{15} + D^{16} + \dots$$

$$F(D) = \frac{\partial}{\partial I} T(D,I)|_{I=1} =$$

$$= D^6 + 2D^8 + 2D^9 + 3D^{10} + 9D^{11} + 4D^{12} + 24D^{13} + 9D^{14} + 50D^{15} + 31D^{16} + \dots$$

Свободное расстояние кода $d_f = 6$. Построим графики зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК и канала с АБГШ:

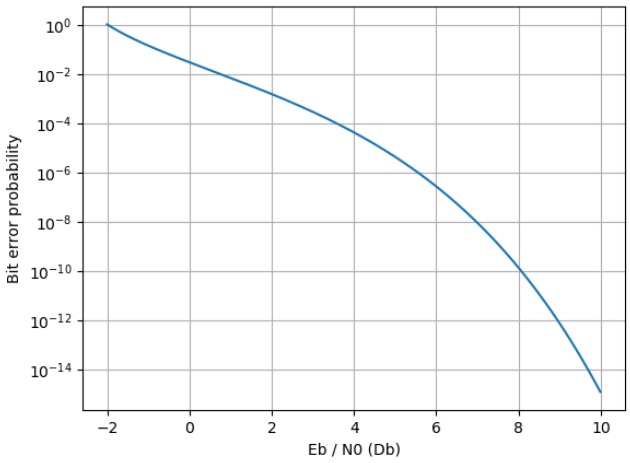

Рис. 9: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК

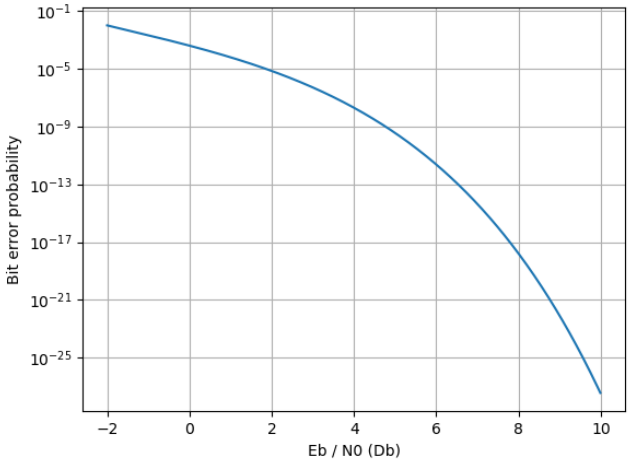

Рис. 10: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для канала с АБГШ

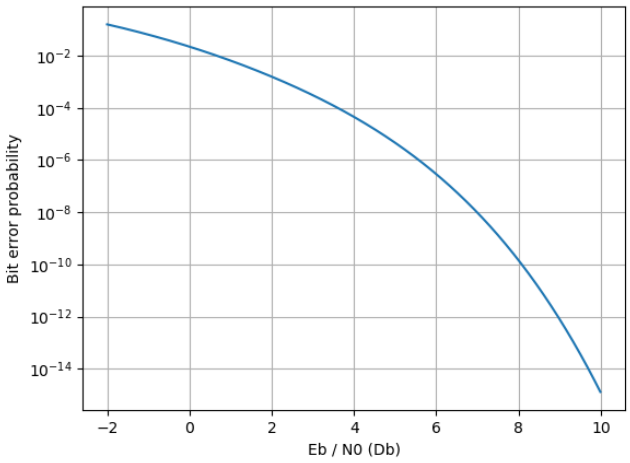

Рис. 11: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК, основываясь на усеченном до 5 членов спектре весов

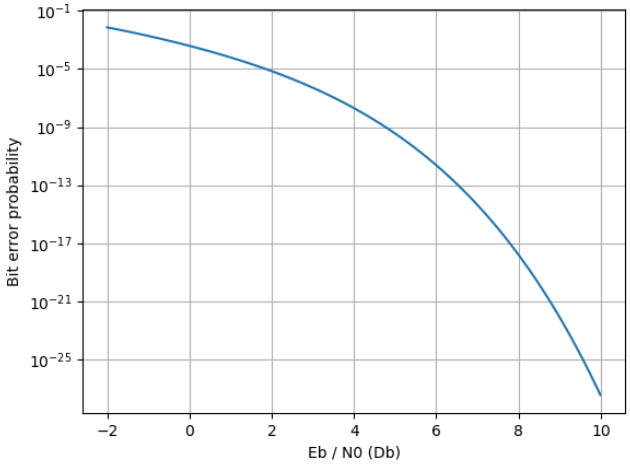

Рис. 12: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для канала с АБГШ, основываясь на усеченном до 5 членов спектре весов

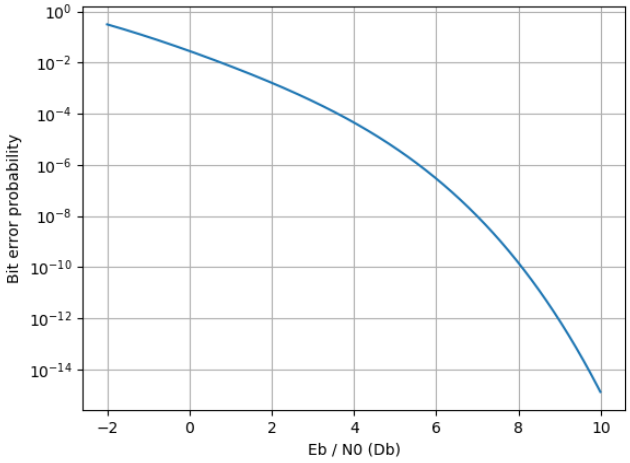

Рис. 13: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК, основываясь на усеченном до 10 членов спектре весов

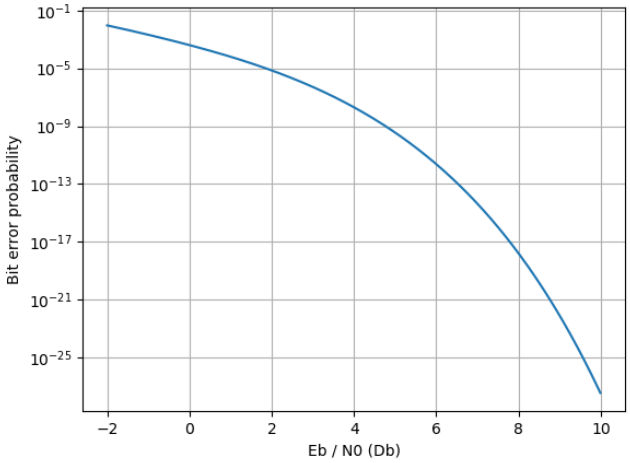

Рис. 14: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для канала с АБГШ, основываясь на усеченном до 10 членов спектре весов

Графики были получены с помощью следующей программы, написанной на языке Python:

```
import math
from scipy import special
import numpy as np
import matplotlib.pyplot as plt

DF = 6
FCoeffs =    [1, 2, 2, 3,  9,  4,  24, 9,  50, 31]
FPowers = [6, 8, 9, 10, 11, 12, 13, 14, 15, 16]

def Q(x):
    return 0.5 * special.erfc(x / math.sqrt(2))

def Peb(df, B, D0, F, p0):
    return B(df, p0) * F(D0(p0))
```

```
def bscB(w, p0):
    return (1. - p0) / (1. - 2. * p0)  math.sqrt(2. / (math.pi * w))

def bscD0(p0):
    return 2. * math.sqrt(p0 * (1. - p0))

def awgncB(w, sg):
    return Q(math.sqrt(2. * w * sg)) * math.exp(w * sg)

def awgncD0(sg):
    return math.exp(-sg)

def F(D):
    return -math.pow(D, 6.) * ((D - 1.) * math.pow(D, 4.) -
    2. * (D - 1.) * math.pow(D, 2.) - 1.) /
    math.pow(((D - 1.) * math.pow(D, 4.) + 2. * math.pow(D, 2) - 1), 2.)

def F_5(D):
    return np.dot(FCoeffs[0:5], list(map(lambda i: math.pow(D, i), FPowers[0:5])))

def F_10(D):
    return np.dot(FCoeffs, list(map(lambda i: math.pow(D, i), FPowers)))

sdb = list(np.arange(-2, 10., 0.01))
sb = list(map(lambda x: math.pow(10., x / 10.), sdb))
sig = list(map(lambda x: math.sqrt(0.5 / x), sb))
sig1 = list(map(lambda x: 1. / x, sig))
p0 = list(map(lambda x: Q(x), sig1))

bscPEB = list(map(lambda p: Peb(DF, bscB, bscD0, F, p), p0))
awgncPEB = list(map(lambda p: Peb(DF, awgncB, awgncD0, F, p), sb))

bscPEB_5 = list(map(lambda p: Peb(DF, bscB, bscD0, F_5, p), p0))
awgncPEB_5 = list(map(lambda p: Peb(DF, awgncB, awgncD0, F_5, p), sb))

bscPEB_10 = list(map(lambda p: Peb(DF, bscB, bscD0, F_10, p), p0))
awgncPEB_10 = list(map(lambda p: Peb(DF, awgncB, awgncD0, F_10, p), sb))

def draw_plot(xs, ys, filename):
    plt.semilogy(xs, ys)
    plt.xlabel('Eb␣/␣N0␣(Db)')
    plt.ylabel('Bit␣error␣probability')
    plt.grid(True, which='both')
    plt.savefig(filename)
    plt.show()

draw_plot(sdb, bscPEB, "bsc.png")
draw_plot(sdb, awgncPEB, "awgnc.png")

draw_plot(sdb, bscPEB_5, "bsc5.png")
draw_plot(sdb, awgncPEB_5, "awgnc5.png")

draw_plot(sdb, bscPEB_10, "bsc10.png")
draw_plot(sdb, awgncPEB_10, "awgnc10.png")
```

7.2 Задача 7.

Вариант 95: дан свёрточный код с порождающими многочленами $(7, 5, 1)$ (в двоично-восьмеричной форме). С помощью программы, написанной на языке Matlab и представленной ниже, были получены графики зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК и канала с АБГШ:

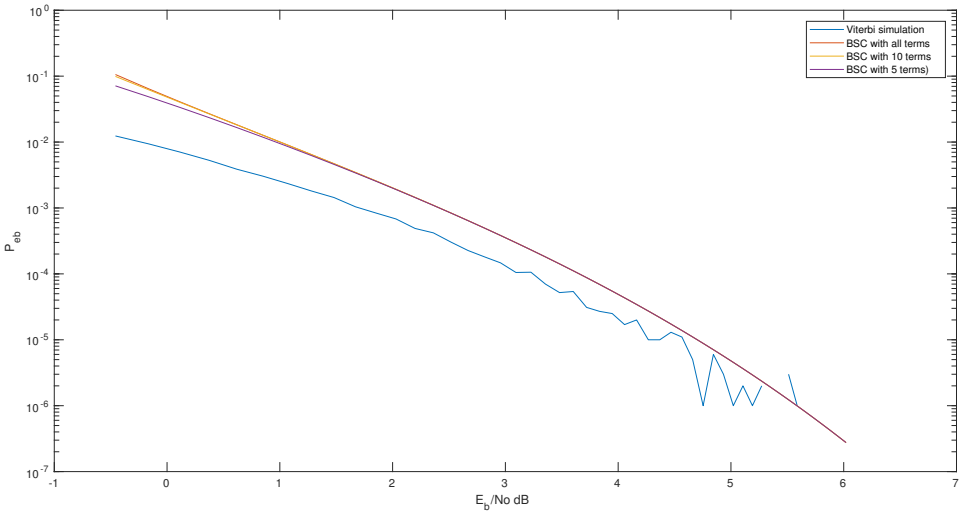

Рис. 15: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для ДСК с помощью моделирования декодирования алгоритмом Витерби

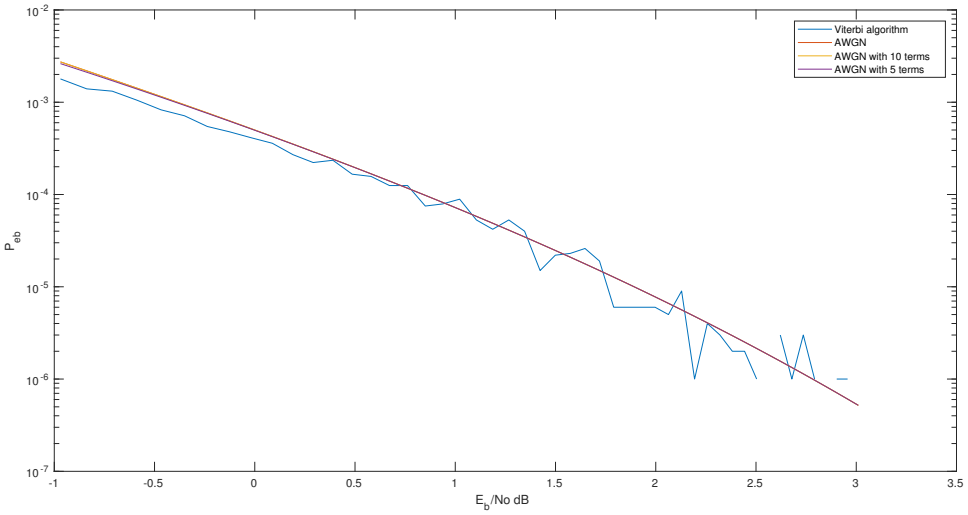

Рис. 16: График зависимости аддитивной оценки вероятности ошибки на бит от отношения сигнал/шум для канала с АБГШ с помощью моделирования декодирования алгоритмом Витерби

```
function y = pseudoF(D, kfs, pws, cnt)
res = 0;
for i=1:cnt
    res = res + kfs(i) * D^pws(i);
end
y = res;
```

```
end
```

```
gens = [7 5 1];
kfs = [1 4 12 32 80 192 448 1024 2304 5120];
pws = [6 8 10 12 14 16 18 20 22 24];
t = poly2trellis(3,gens);
spect = distspec(t, 10);
df = spect.dfree;
spect.weight
s = RandStream.create('mt19937ar', 'seed',94384);
prevStream = RandStream.setGlobalStream(s);
msg = randi([0 1],1000000,1);
hConvEnc = comm.ConvolutionalEncoder(t);
n = 50;
xs = linspace(0.9, 4, n);
xsIndB = 10 * log10(xs);
ratio = zeros(n, 1);
number = zeros(n, 1);
pebsAWGN = zeros(n, 1);
pebsBSC = zeros(n, 1);
pebsAWGN10 = zeros(n, 1);
pebsBSC10 = zeros(n, 1);
pebsAWGN5 = zeros(n, 1);
pebsBSC5 = zeros(n, 1);
for i=1:n
    x = xs(i);
    b = (1 - normcdf(sqrt(2 * df * x))) * exp(df * x);
    d0 = exp(-x);
    pebsAWGN(i) = b * F(d0);
    pebsAWGN10(i) = b * pseudoF(d0, kfs, pws, 10);
    pebsAWGN5(i) = b * pseudoF(d0, kfs, pws, 5);
    p = 1 - normcdf(sqrt(2 * x));
    b = (1 - p) / (1 - 2 * p) * sqrt(2 / (df * pi));
    d0 = 2 * sqrt(p * (1 - p));
    pebsBSC(i) = b * F(d0);
    pebsBSC10(i) = b * pseudoF(d0, kfs, pws, 10);
    pebsBSC5(i)  = b * pseudoF(d0, kfs, pws, 5);
    tblen = 48;
    hVitDec = comm.ViterbiDecoder(t, 'InputFormat', 'Hard', 'TracebackDepth', tblen,
        'TerminationMethod', 'Continuous');
    hErrorCalc = comm.ErrorRate('ReceiveDelay', tblen);
    code = step(hConvEnc,msg);
    qcode = bsc(code, p);
    delay = tblen;
    decoded = step(hVitDec,qcode); % Decode.
    ber = step(hErrorCalc, msg, decoded);
    ratio(i) = ber(1);
    number(i) = ber(2);
end;
figure;
semilogy(xsIndB, ratio, 'DisplayName', 'Viterbi simulation');
hold on;
semilogy(xsIndB, pebsBSC, 'DisplayName', 'BSC with all terms');
hold on;
semilogy(xsIndB, pebsBSC10, 'DisplayName','BSC with 10 terms');
hold on;
semilogy(xsIndB, pebsBSC5, 'DisplayName', 'BSC with 5 terms)');
legend('show')
```

```
xlabel('E_b/No dB');
ylabel('P_{eb}');
hold off;
format long;
log10(ratio)
RandStream.setGlobalStream(prevStream);
```

```
gens = [7 5 1];
kfs = [1 4 12 32 80 192 448 1024 2304 5120];
pws = [6 8 10 12 14 16 18 20 22 24];
t = poly2trellis(3,gens);
spect = distspec(t, 10);
df = spect.dfree;
spect.weight
s = RandStream.create('mt19937ar', 'seed',94384);
prevStream = RandStream.setGlobalStream(s);
msg = randi([0 1],1000000,1);
hConvEnc = comm.ConvolutionalEncoder(t);
n = 50;
xs = linspace(0.8, 2, n);
xsIndB = 10 * log10(xs);
ratio = zeros(n, 1);
number = zeros(n, 1);
pebsAWGN = zeros(n, 1);
pebsBSC = zeros(n, 1);
pebsAWGN10 = zeros(n, 1);
pebsBSC10 = zeros(n, 1);
pebsAWGN5 = zeros(n, 1);
pebsBSC5 = zeros(n, 1);
for i=1:n
    x = xs(i);
    b = (1 - normcdf(sqrt(2 * df * x))) * exp(df * x);
    d0 = exp(-x);
    pebsAWGN(i) = b * F(d0);
    pebsAWGN10(i) = b * pseudoF(d0, kfs, pws, 10);
    pebsAWGN5(i) = b * pseudoF(d0, kfs, pws, 5);
    p = 1 - normcdf(sqrt(2 * x));
    b = (1 - p) / (1 - 2 * p) * sqrt(2 / (df * pi));
    d0 = 2 * sqrt(p * (1 - p));
    pebsBSC(i) = b * F(d0);
    pebsBSC10(i) = b * pseudoF(d0, kfs, pws, 10);
    pebsBSC5(i)  = b * pseudoF(d0, kfs, pws, 5);
    hChan = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (Eb/No)',...
      'EbNo', xsIndB(i));
    tblen = 48;
    hMod = comm.QPSKModulator('BitInput',true);
    hDemodLLR = comm.QPSKDemodulator('BitOutput',true,...
        'DecisionMethod', 'Log-likelihood ratio');
    hVitDec = comm.ViterbiDecoder(t, 'TracebackDepth', tblen, 'TerminationMethod', 'C
    hErrorCalc = comm.ErrorRate('ReceiveDelay', tblen);
    code = step(hConvEnc,msg);
    hChan.SignalPower = (code'*code)/length(code);
    code = step(hMod, code);
    ncode = step(hChan,code);
    ncode = step(hDemodLLR,ncode);
    qcode = ncode;
    delay = tblen;
    decoded = step(hVitDec,qcode);
```

```
    ber = step(hErrorCalc, msg, decoded);
    ratio(i) = ber(1);
    number(i) = ber(2);
end;
figure;
semilogy(xsIndB, ratio, 'DisplayName', 'Viterbi algorithm');
hold on;
semilogy(xsIndB, pebsAWGN, 'DisplayName', 'AWGN');
hold on;
semilogy(xsIndB, pebsAWGN10, 'DisplayName', 'AWGN with 10 terms');
hold on;
semilogy(xsIndB, pebsAWGN5, 'DisplayName', 'AWGN with 5 terms');
legend('show')
xlabel('E_b/No dB');
ylabel('P_{eb}');
hold off;
format long;
log10(ratio)
RandStream.setGlobalStream(prevStream);
```