Задача 1.4

Пусть было отправлено слово x, а принято слово y. Предположим, что $\rho(x, y) = d$. Значит произошло d ошибок. вероятность этого $p = \left(\frac{p_0}{q-1}\right)^d (1-p_0)^{n-d} =$ $\left(\frac{p_0}{(q-1)(1-p_0)}\right)^d (1-p_0)^n$. Так как $1 - p_0$ и n — константы, то задача минимизции p эквивалентна задаче минимизации $\left(\frac{p_0}{(q-1)(1-p_0)}\right)^d$. Эта задача эквивалентна минимизации d (то есть расстояния Хэмминга), если $0 < \frac{p_0}{(q-1)(1-p_0)} < 1$

1. $0 < \frac{p_0}{(q-1)(1-p_0)} \rightarrow \begin{cases} p_0 > 0 \\ (q-1)(1-p_0) > 0 \end{cases} \rightarrow 0 < p_0 < 1$ — так как p_0 — вероятность, выполнено всегда (вариант с $p_0 < 0$ опущен)

2. $\frac{p_0}{(q-1)(1-p_0)} < 1 \rightarrow p_0 < (q-1)(1-p_0) \rightarrow 1 < q - q \cdot p_0 \rightarrow p_0 < \frac{q-1}{q}$

$0 < p_0 < \frac{q-1}{q}$

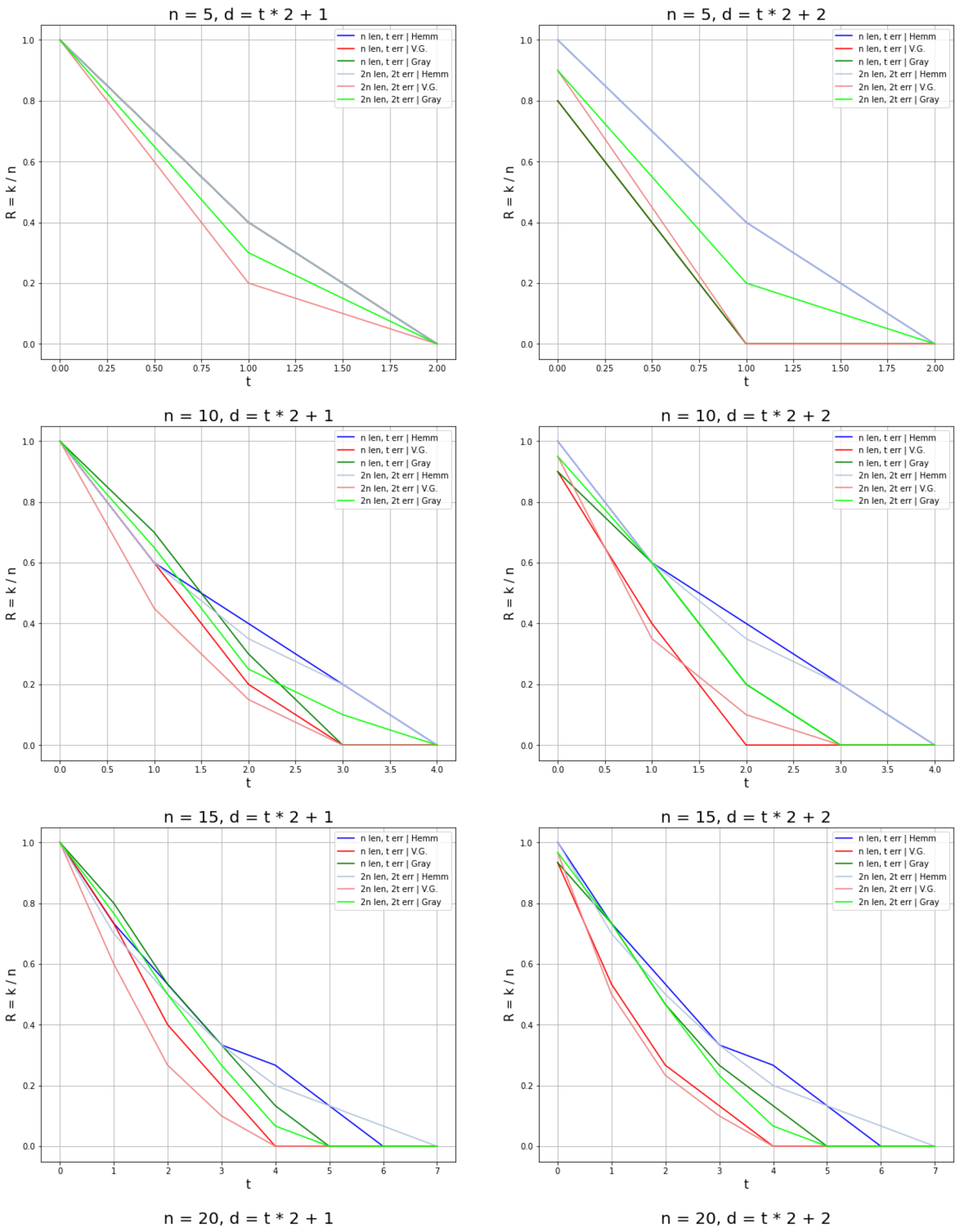
Задача 1.11

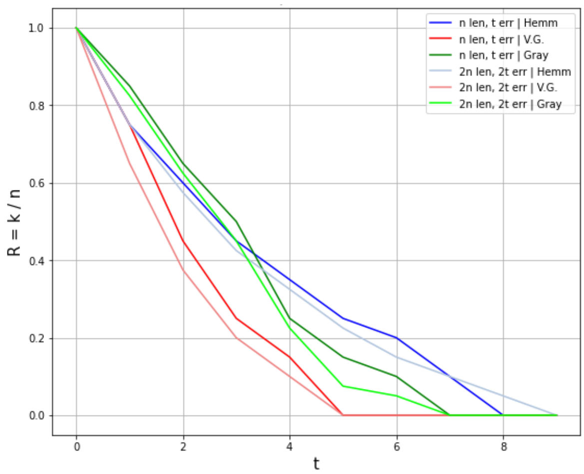
Посмотрим для различных длин кодовых слов n какова скорость потенциальная скорость кода R, которую можно оценить с помощью границ Хэмминга, Варшамова–Гилберта и Грея для разного числа исправленных ошибок t

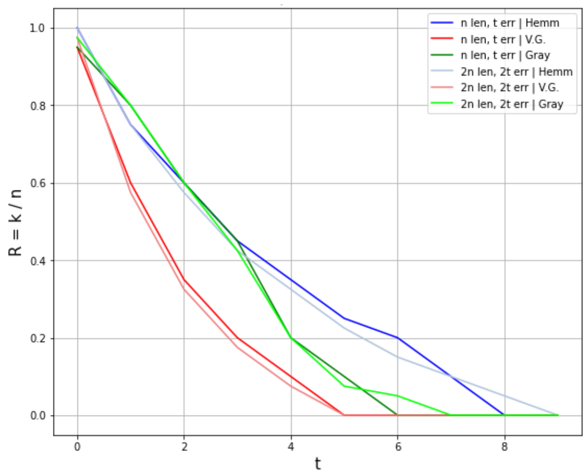
In [2]:

```
import math
from scipy.special import binom
from math import log2
import matplotlib.pyplot as plt

def bord(n, d):
    # Hemm
    t = (d - 1) // 2
    max_h_k = 0
    for k in range(2, n + 1):
        M = 2 ** k
        if M <= 2 ** n / sum([binom(n, i) for i in range(0, t + 1)]):
            max_h_k = k

    # V.G.
    max_vg_k = 0
    for k in range(2, n + 1):
        if 2 ** (n - k) > sum([binom(n - 1, i) for i in range(d - 1)]):
            max_vg_k = k

    # Gray
    max_g_k = 0
    for k in range(2, n + 1):
        if n >= sum([math.ceil(d / 2 ** i) for i in range(k)]):
            max_g_k = k
    return (max_h_k, max_vg_k, max_g_k)

glob_n = 50
def plot(fig, n, i, ch):
    if ch:
        plus = 1
    else:
        plus = 2
    ind_p = i * 2 + plus
    ax = fig.add_subplot(glob_n, 2, ind_p)
    ax.set_title("n = {}, d = t * 2 + {}".format(n, plus), fontsize=20)
    ax.set_xlabel('t', fontsize=15)
    ax.set_ylabel('R = k / n', fontsize=15)

    ks = [[], [], []]
    ks2 = [[], [], []]
    for t in range((n - 1) // 2 + 1):
        d = t * 2 + plus
        h_k, vg_k, g_k = bord(n, d)
        ks[0].append(h_k / n)
        ks[1].append(vg_k / n)
        ks[2].append(g_k / n)

        d = t * 4 + plus
        h_k, vg_k, g_k = bord(2 * n, d)
        ks2[0].append(h_k / (2 * n))
        ks2[1].append(vg_k / (2 * n))
        ks2[2].append(g_k / (2 * n))

    ts = list(range((n - 1) // 2 + 1))
```

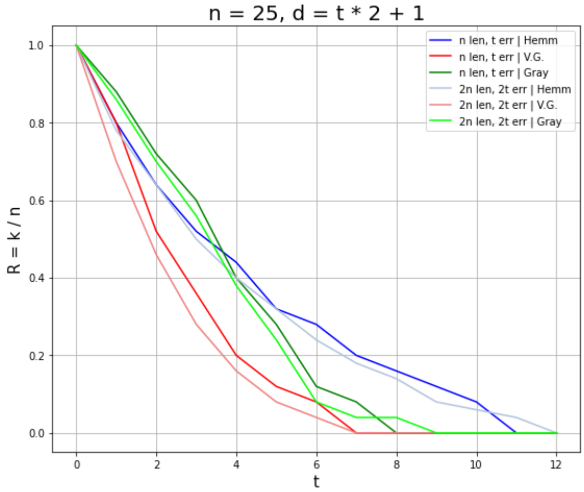
```
    ax.plot(ts, ks[0], label="n len, t err | Hemm", color="blue")
    ax.plot(ts, ks[1], label="n len, t err | V.G.", color="red")
    ax.plot(ts, ks[2], label="n len, t err | Gray", color="green")
    ax.plot(ts, ks2[0], label="2n len, 2t err | Hemm", color="lightsteelblue'
    ax.plot(ts, ks2[1], label="2n len, 2t err | V.G.", color="lightcoral")
    ax.plot(ts, ks2[2], label="2n len, 2t err | Gray", color="lime")
    ax.legend()
    ax.grid()


fig = plt.figure(figsize=(20, 450))
cur = 0
for i in range(5, glob_n + 1, 5):
    plot(fig, i, cur, True)
    plot(fig, i, cur, False)
    cur += 1
```



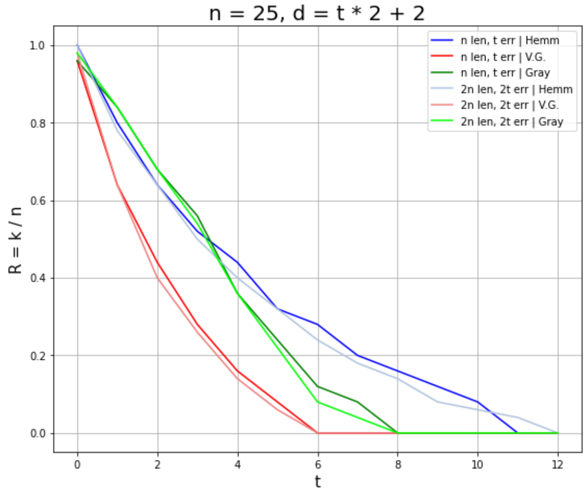




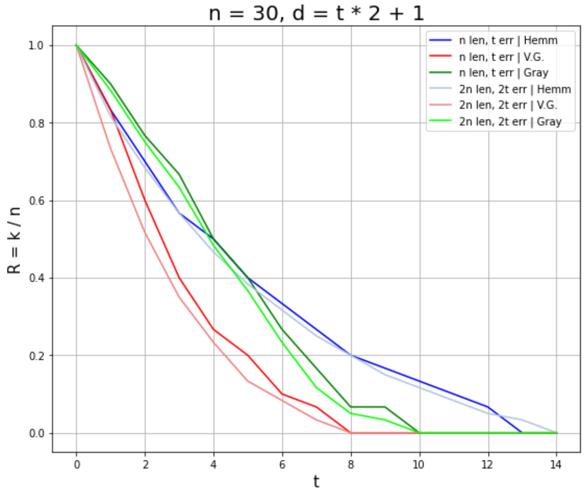
n = 25, d = t * 2 + 1



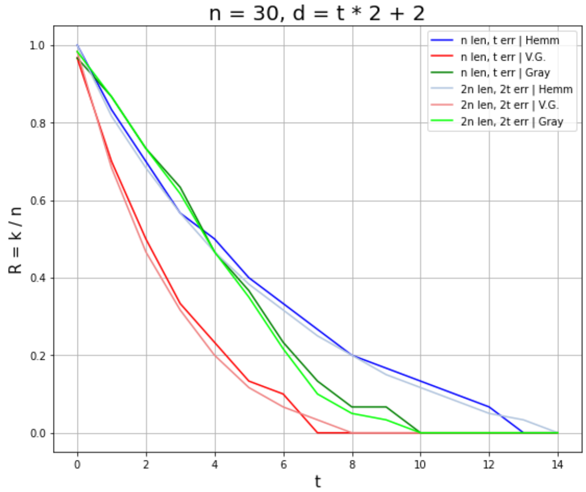
n = 25, d = t * 2 + 2



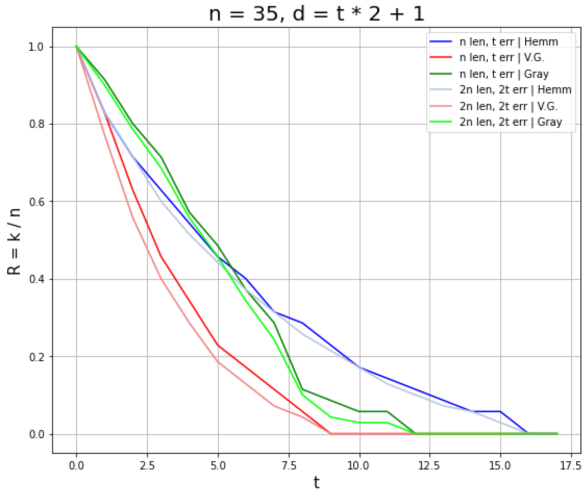
n = 30, d = t * 2 + 1



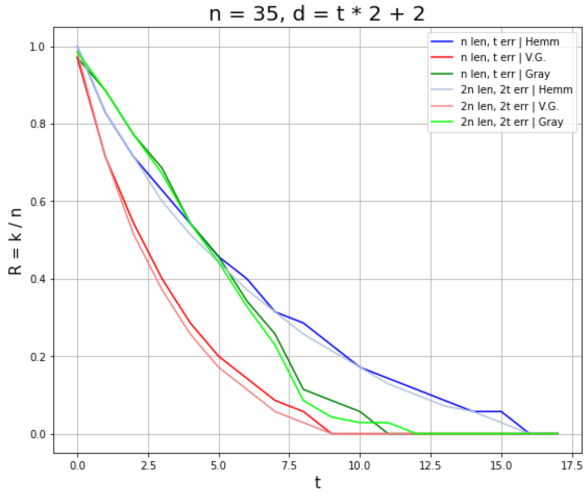
n = 30, d = t * 2 + 2



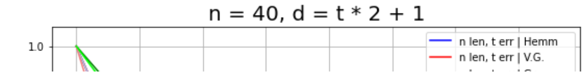
n = 35, d = t * 2 + 1



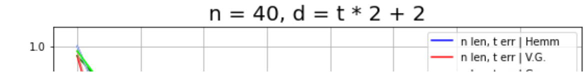
n = 35, d = t * 2 + 2

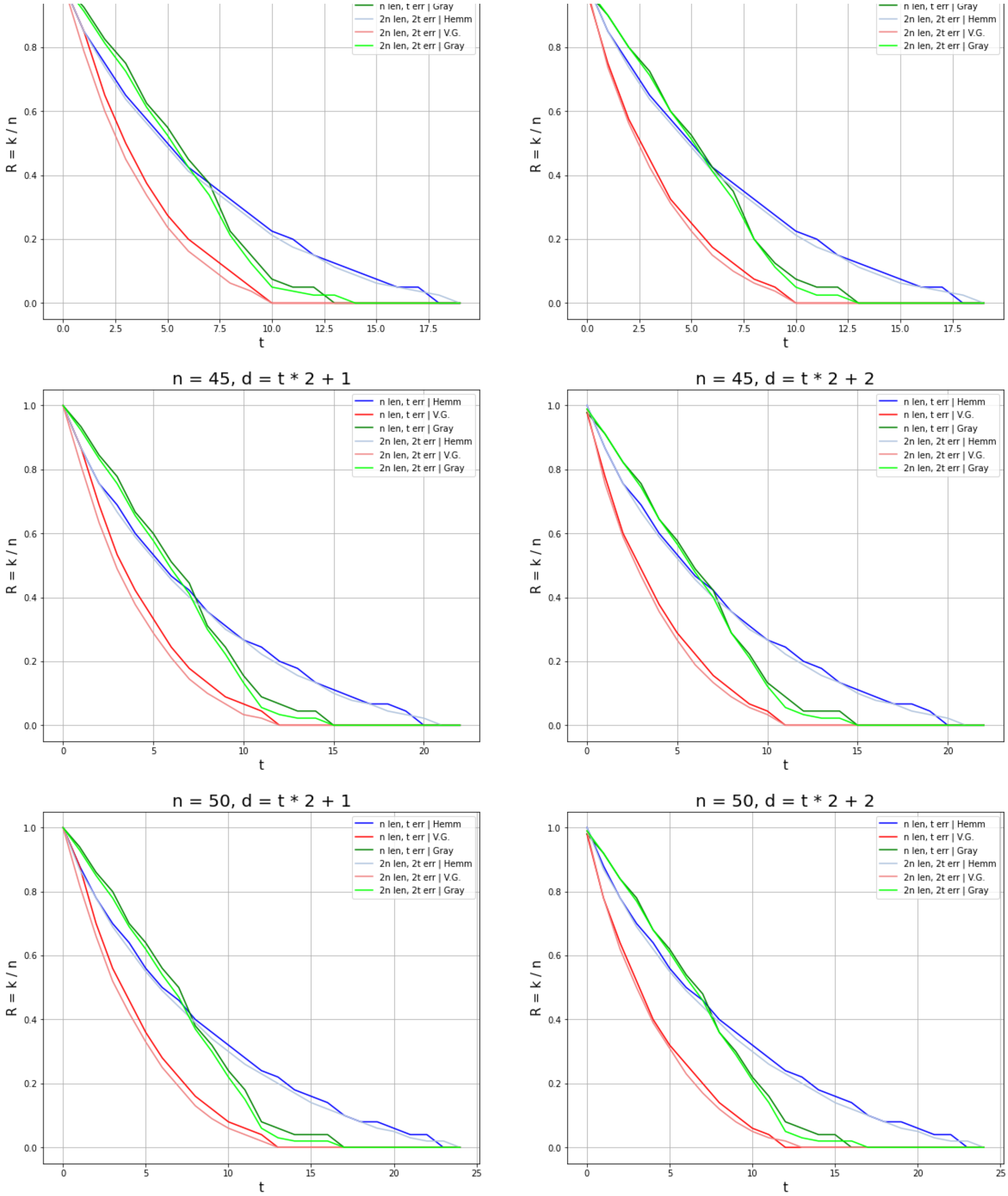


n = 40, d = t * 2 + 1



n = 40, d = t * 2 + 2





Можно заметить, что небольших n разница не сильно заметна. Однако при увеличении длины видно, что согласно всем границам, код длины n исправляющий t ошибок имеет чуть-чуть лучшую скорость, чем код длины $2n$ исправляющий $2t$ ошибок. При этом короткий код значительно проще кодировать и декодировать, поэтому по скорости обработки и передачи информации короткий код лучше.

Но, конечно, длинный код безопаснее, так как исправляет больше ошибок

Упражнение 2.12

Постройте порождающие и проверочные матрицы для кодов из примера 2.1:

1. $\{00, 01, 10, 11\}$.

   Базис — $\{01, 10\}$. $G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. $n = 2,\ k = 2$, то есть $r = 2 - 2 = 0$ и обратная матрица H неопределена

2. $\{000, 001, 010, 011, 100, 101, 110, 111\}$

   Базис — $\{001, 010, 100\}$. $G = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. $n = 3,\ k = 3$, то есть $r = 3 - 3 = 0$ и обратная матрица H неопределена

3. $\{000, 011, 101, 110\}$

   Базис — $\{011, 110\}$. $G = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$. Приведем матрицу к виду $G = \begin{pmatrix} I_k & P \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$. Тогда проверочная матрица $H = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$

4. $\{000, 111\}$.

   Базис — $\{111\}$. $G = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$. Матрицу уже находится в виде $G = \begin{pmatrix} I_k & P \end{pmatrix}$. Тогда проверочная матрица $H = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

Задача 2.1

Построить наилучшие в смысле минимального расстояния коды длины 6 со скоростями:

$R = \dfrac{1}{6}$: $n = 6,\ k = 1$

Порождающая матрица: $G = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$

Всего 2 кодовых слова — 0000000 и 111111. Расстояние $d = 6$ (а лучше и нельзя)

$R = \dfrac{2}{6}$: $n = 6,\ k = 2$

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

Расстояние $d = 4$. Пусть есть код с расстоянием $d = 5$. Тогда код может исправить 2 ошибки, и пространство делится на $2^2 = 4$ непересекающихся шаров (соответствующие входным словам). В каждом шаре находится $1 + 6 + 15$ ($<$нет ошибок$>$ + $<$1 ошибка$>$ + $<$2 ошибки$>$) кодовых слов. То есть должно быть $(1 + 6 + 15) \cdot 4 = 88$ кодовых слов, но их всего $2^6 = 64$

$R = \dfrac{3}{6}$: $n = 6,\ k = 3$

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

Расстояние $d = 3$. Пусть существует код с расстоянием $d = 4$. Для этого необходимо, чтобы любые 3 столбца проверочной матрицы были линейно независимы. Не теряя общности будем считать, что матрица имеет вид $H = \begin{pmatrix} x_{11} & x_{12} & x_{13} & 1 & 0 & 0 \\ x_{21} & x_{22} & x_{23} & 0 & 1 & 0 \\ x_{31} & x_{32} & x_{33} & 0 & 0 & 1 \end{pmatrix}$.

Рассмотрим следующие множества столбцов:

(a) $\{1, 5, 6\}$: для линейной независимости необходимо $x_{11} = 1$

(b) $\{1, 4, 6\}$: … $x_{21} = 1$

(c) $\{1, 4, 5\}$: … $x_{31} = 1$

То есть $\begin{pmatrix} x_{11} \\ x_{21} \\ x_{31} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

Аналогично, $\begin{pmatrix} x_{12} \\ x_{22} \\ x_{32} \end{pmatrix} = \begin{pmatrix} x_{13} \\ x_{23} \\ x_{33} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$. Но тогда стоблцы $1, 2, 3$ линейно зависимы

$R = \dfrac{4}{6}:$ $n = 6,\ k = 4$

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

Расстояние $d = 2$. Пусть есть код с расстоянием $d = 3$. Тогда код может исправить 1 ошибки, и пространство делится на $2^4 = 16$ непересекающихся шаров (соответствующие входным словам). В каждом шаре находится 1+6 ($<$нет ошибок$>$ + $<$1 ошибка$>$) кодовых слов. То есть должно быть $(1+6) \cdot 16 = 112$ кодовых слов, но их всего $2^6 = 64$

$R = \dfrac{5}{6}:$ $n = 6,\ k = 5$

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

Расстояние $d = 2$. Согласно границе синглтона $d \leqslant n - k + 1 = 6 - 5 + 1 = 2$. Значит, код оптимален

$R = 1:$ $n = 6,\ k = 6$

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

Расстояние $d = 1$ — единственный возможный код (иначе будут коллизии)

Задача 2.2

Если канал может исправить t ошибок, то вероятность ошибки при декодировании равна

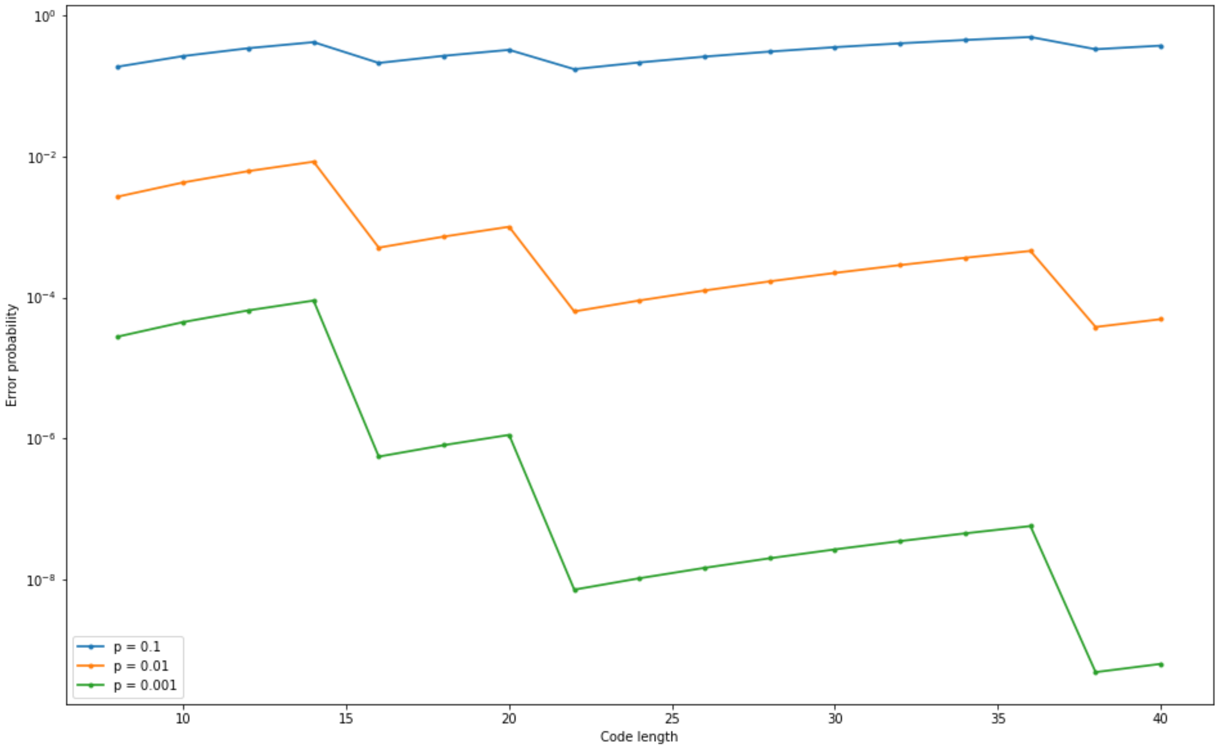
$$p_{err} = 1 - \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}$$

где p - переходная вероятность в ДСК.

In [85]:
```
from scipy.special import binom

def p_err(n, k, d, p):
    t = (d - 1) // 2
    return 1 - sum([binom(n, i) * p ** i * (1 - p) ** (n - i) for i in range(
```

In [86]:
```
import matplotlib.pyplot as plt

ns = list(range(8, 41, 2))
ks = list(range(4, 21))
ds = [4, 4, 4, 4, 5, 6, 6, 7, 8, 7, 8, 8, 8, 8, 8, 9, 10]

def add_plot_with_err(p):
    errs = [p_err(ns[i], ks[i], ds[i], p) for i in range(len(ns))]
    plt.semilogy(ns, errs, marker=".", linewidth=1.7, label="p = {}".format(p
```

In [88]:
```
plt.figure(figsize=(16, 10))
add_plot_with_err(0.1)
add_plot_with_err(0.01)
add_plot_with_err(0.001)

plt.xlabel('Code length')
plt.ylabel('Error probability')
plt.legend()
plt.show()
```



$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$$

Мы знаем, что $P_b = 10^{-5}$. Найдем $\sqrt{\frac{2E_b}{N_0}}$

In [89]:
```
from scipy.special import erfc
import numpy as np

def Q(x): return erfc(x / np.sqrt(2)) / 2

def find_x(p):
    l = 0
    r = 10
    while np.fabs(r - l) > 1e-15:
        m = (r + l) / 2
        if Q(m) > p:
            l = m
        else:
            r = m
    return (l + r) / 2
```

In [90]:
```
p = 1e-5
x = find_x(p)
print("{} = Q({})".format(p, x))
E0N0 = x ** 2 / 2
```

```
1e-05 = Q(4.2648907939228256)
```

$\sqrt{\frac{2E_b}{N_0}} = 4.265$. Значит $\frac{E_b}{N_0} = 9.095$

Тогда теоретический выигрыш кодирования равен
$\frac{E_b}{N_0} - \frac{E_s}{N_0} = 9.095 - (-1.59) = 10.685$ Дб

Задача 2.3 + 2.7

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

   Приведем матрицу к виду

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

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

   Скорость $R = \frac{k}{n} = \frac{3}{6} = \frac{1}{2}$

   Минимальное расстояние $d = 3$ (вычисленно программно, просто перебором всех входных слов)

   Для проверочной матрицы $H_1 = G$

Синдром	Вектор ошибки
000	000000
001	001000
010	010000
011	000101
100	100000
101	000001
110	000100
111	000010

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

   Приведем матрицу к виду

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

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

   Скорость $R = \frac{k}{n} = \frac{2}{6} = \frac{1}{3}$

   Минимальное расстояние $d = 3$

   Для проверочной матрицы $H_1 = G$

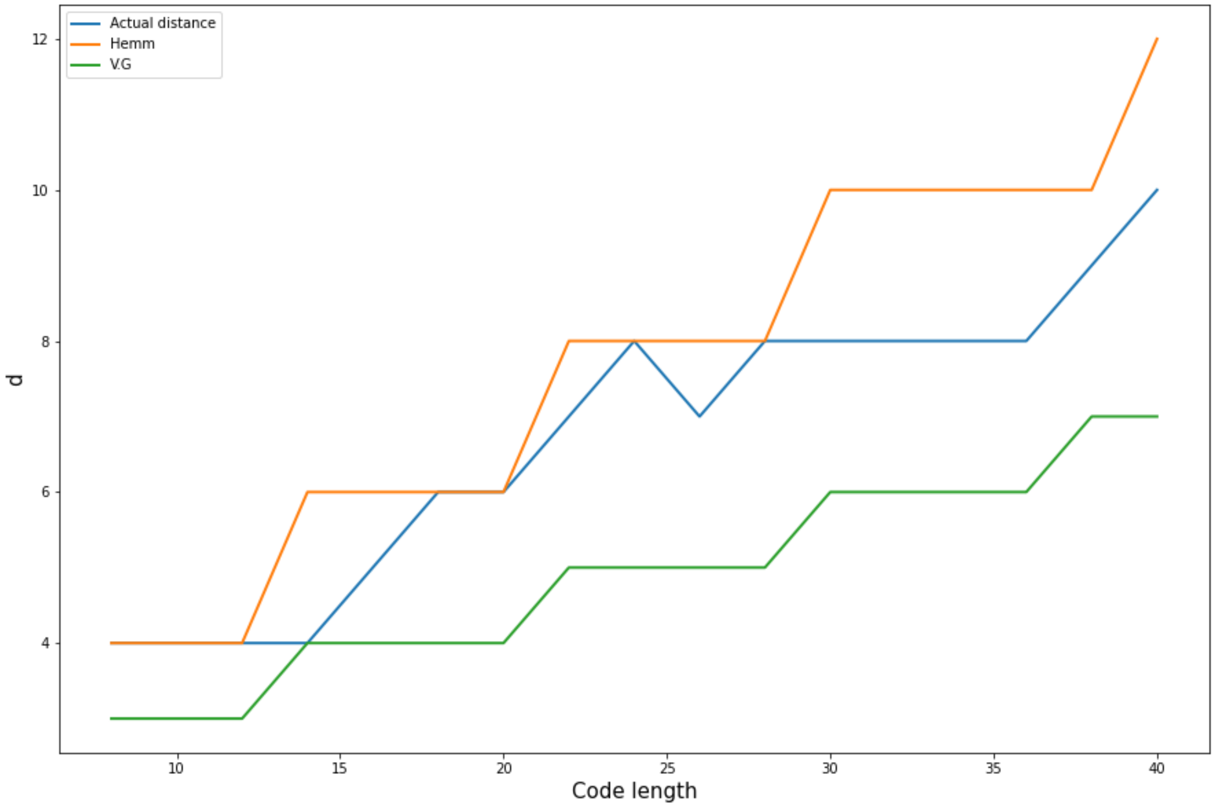
Синдром	Вектор ошибки
00	000000
01	000001
10	000010
11	000100

Задача 3.5

Почитаем для всех кодов со скоростью $R = \dfrac{1}{2}$ из таблицы границы Хэмминга и Варшамова–Гилберта. Результаты сравнения на графике

In [3]:
```
from scipy.special import binom

def bord(n, k):
    # Hemm
    M = 2 ** k
    max_h_d = 0
    for d in range(2, n + 1):
        t = (d - 1) // 2
        if M <= 2 ** n / sum([binom(n, i) for i in range(0, t + 1)]):
            max_h_d = d

    # V.G.
    max_vg_d = 0
    for d in range(2, n + 1):
        if 2 ** (n - k) > sum([binom(n - 1, i) for i in range(d - 1)]):
            max_vg_d = d

    return (max_h_d, max_vg_d)
```

In [13]:
```
import matplotlib.pyplot as plt

ns = list(range(8, 41, 2))
ks = list(range(4, 21))
ds = [4, 4, 4, 4, 5, 6, 6, 7, 8, 7, 8, 8, 8, 8, 8, 9, 10]

hds = []
vgds = []
for i in range(len(ns)):
    h_d, vg_d = bord(ns[i], ks[i])
    hds.append(h_d)
    vgds.append(vg_d)

plt.figure(figsize=(15, 10))
plt.plot(ns, ds, label='Actual distance', linewidth=2)
plt.plot(ns, hds, label='Hemm', linewidth=2)
plt.plot(ns, vgds, label='V.G', linewidth=2)

plt.xlabel('Code length', fontsize=15)
plt.ylabel('d', fontsize=15)
plt.legend()
plt.show()
```


Actual distance
Hemm
V.G
12
10
8
6
4
d
10
15
20
25
30
35
40
Code length