МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра математического моделирования и анализа данных

ШИМКО Андрей Чеславович

ОБНАРУЖЕНИЕ ВКРАПЛЕНИЙ В ДВОИЧНУЮ ЦЕПЬ МАРКОВА НА ОСНОВЕ ЭНТРОПИЙНЫХ ХАРАКТЕРИСТИК

Дипломная работа

Научный руководитель Егор Валентинович Вечерко
кандидат физико-математических наук,
старший преподаватель

Допущена к защите
«___»_____ 2017 г.
Зав. кафедрой математического
моделирования и анализа данных
_____ Ю.С. Харин
доктор физико-математических наук,
профессор,
член-корреспондент НАН Беларуси

Минск, 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Стеганография представляет собой специфическую область человеческой деятельности, связанной с разработкой и анализом методов сокрытия факта передачи информации. Подобно криптографии, стеганография известна со времен античности. Но на этом аналогии, по крайне мере в контексте теоретических исследований, заканчиваются. За последние четверть века возникла и успешно раз- вивается новая математическая дисциплина криптология, или, что то же самое, математическая криптография, изучающая математические модели криптографических схем. Попытки создания ма- тематической стеганографии (которую, быть может, следует именовать также стеганологией) пред- принимаются, но исследования здесь находятся лишь в зачаточном состоянии.

Такое положение дел обусловлено прежде всего сложностью возникающих в стеганографии задач. Всякая попытка построения математических моделей стеганографических систем сопряжена с необходимостью рассмотрения большого количества случаев и подслучаев, не допускающих простой и единообразной трактовки. Другими словами, внешняя среда, в которой должны функционировать стеганографические системы, имеет гораздо большее, по сравнению с внешней средой криптографических схем, количество степеней свободы.

Основными критериями для оценки и сравнения различных методов построения стеганографических систем являются их стойкость и емкость. В отличие от достаточно исследованных криптографических систем, оценки стойкости стегосистем более сложны и само понятие стойкости имеет большое число различных формулировок, что объясняется разнообразием задач стеганографи-ческой защиты данных. В настоящей работе исследуются методы построения стегосистем, предназначенных для скрытия факта передачи конфиденциальных сообщений. Говоря о стойкости криптографических систем, важно упомянуть о принципе Керкхоффса, который заключается в том, что система защиты информации должна обеспечивать свои функции даже при полной информированности противника о ее структуре и алгоритмах, и вся секретность системы должна заключаться в ключе. Этот принцип также можно соотнести с определением стойкости стегосистем. В данном случае, ключом может являться, например, секретная последовательность, определяющая порядок прохода, элементов контейнера при внедрении бит информа-

ции, что имеет место в алгоритмах рассеянного заполнения контейнеров. Второй критерий, емкость метода, определяет максимальное количество встраиваемой информации, и может выражаться в единицах бит на пиксель.

ГЛАВА 1
Теоретический подход

1.1 Основные понятия

Определение 1.1.1. Стеганография - наука о способах передачи (хранения) скрытой информации, при которых скрытый канал организуется на базе и внутри открытого канала с использованием особенностей восприятия информации, причем для этой цели могут использоваться такие приемы, [3]как:

1. Полное сокрытие факта существования скрытого канала связи
2. Создание трудностй для обнаружения, извлечения и модификации передаваемых скрытых сообщений внутри открытых сообщений-контейнеров
3. Маскировки скрытой информации в протоколе

Определение 1.1.2. Контейнером $b \in B$ (носителем) называют несекретные данные, которые используют для сокрытия сообщения [3].

Определение 1.1.3. Сообщение $m \in M$ - секретная информация, наличие которой в контейнере необходимо скрыть [3].

Определение 1.1.4. Ключ $k \in K$ - секретная информация, известная только законному пользователю, которая определяет конкретный вид алгоритма сокрытия [3].

Определение 1.1.5. Пустой контейнер - контейнер, не содержащий сообщения [3].

Определение 1.1.6. Заполненный контейнер - контейнер, с внедренным в него сообщением [3].

Определение 1.1.7. Стеганографический алгоритм - два преобразования: прямое стеганографическое преобразование $F : M \times B \times K \to B$ и обратное стеганографическое преобразование $F^{-1} : B \times K \to M$, сопоставляющее соответственно тройке (сообщение, пустой контейнер, ключ) контейнер-результат и паре (заполненный контейнер, ключ) - исходное сообщение, [3]причем:

$$F(m, b, k) = b_{m,k}, F^{-1}(b_{m,k}, k) = m, m \in M, b_{m,k}, b \in B, k \in K \qquad (1.1)$$

Определение 1.1.8. Под стеганографической системой будем понимать $S = (F, F^{-1}, M, B, K)$, представляющую собой совокупность сообщений, секретных ключей, контейнеров и связывающих их преобразований [3].

Определение 1.1.9. Внедрение (сокрытие) - применение прямого стеганографического преобразования к конкретным контейнеру, ключу и сообщению [3].

Определение 1.1.10. Извлечение сообщения - применение обатного стеганографического преобразованияю [3].

Определение 1.1.11. l-грамм - подпоследовательность из l подряд иущих элементов последовательности.

Определение 1.1.12. Под дискретным источником сообщений будем понимать устройство, порождающее последовательности, составленные из букв конечного алфавита A ($|A|=n<\infty$). При этом буквы последовательностей пораждаются в дискретный момент времени:$t = 0, 1, 2, ...; t = ..., -2, -1, 0, 1, 2, ...$; [2]

Определение 1.1.13. Если вероятность того, что источник порождает некоторую поседовательность $a_{i_1}...a_{i_l}$, составленную из букв алфавита A, в момент времени 1, 2, ... l, равна вероятности того, что порождается точно такая же последовательность в момент времени j+1, ..., j+l для любых j,l;$a_{j_1}...a_{j_l}$, то источник называется стационарным. [2] Стационарность означает неизменность во времени всех конечномерных распределений соответствующего случайного процесса.

Определение 1.1.14. Энтропией источника назовем величину:

$$H_\infty = \lim_{l\to\infty} \frac{H(C_l)}{l}; \tag{1.2}$$

если данный предел существует [2].

1.2 Теоретико-информационный подход к оценке стойкости систем

В настоящем разделе рассматривается теоретико-информационный подход к определению стойкости стеганосистем для стеганографических каналов без повторений в присутствии пассивного противника, обладающего неограниченными вычислительными возможностями.

В основе всех известных определений стойкости стеганосистем лежит требование неотличимости распределения вероятностей на множестве стего от распределения вероятностей на множестве пустых контейнеров. Рассматривается статистическая неотличимость, или, иначе говоря, неотличимость относительно произвольных алгоритмов.

Парадигма неотличимости распределений вероятностей заимствована из математической криптографии. Заметим, однако, что ее адекватность для стеганографии не очевидна. По крайней мере, в случае стеганографического канала без повторений не ясно, насколько оправданными будут усилия отправителя по имитации распределения вероятностей на множестве пустых контейнеров. Не следует ли вместо этого стремиться передать скрытое сообщение в одном из наиболее вероятных контейнеров?

Вполне очевидна идея создания стеганографического канала путем маскировки скрытого сообщения под шум, вносимый алгоритмом шифрования в исходный контейнер.

Проверка гипотезы о стойкости системы состоит в том, чтобы определить, какая из двух гипотез - H_0 или H_1 - является верным толкованием наблюдаемой величины Q. Есть два возможных распределения вероятностей, которые принято обозначать P_{Q_0}, P_{Q_1}, над пространством возможных наблюдений. Если верна гипотеза H_0, тогда Q была порождена согласно P_{Q_0}, если же верна гипотеза H_1, тогда Q была порождена согласно P_{Q_1}. Правило принятия решения - это двоичное отображение, заданное на пространстве возможных наблюдений, которое составляет одну из двух возможных гипотез для каждого возможного элемента q. Основной мерой проверки гипотезы является относительная энтропия или различие между двумя распределениями вероятности P_{Q_0} и P_{Q_1}, определяемое следующим выражением:

$$H(P_{Q_0}||P_{Q_1}) = \sum_q P_{Q_0}(q) log \frac{P_{Q_0}(q)}{P_{Q_1}(q)}; \tag{1.3}$$

Относительная энтропия между двумя распределениями всегда неотрицательна и равна нулю только тогда, когда распределения равномерны. Несмотря на то, что остиельная энтропия не является метрикой с точки зрения

математики (так как не симметрична и не удовлетворяет аксиоме треугольника), полезно считать ее таковой. Двоичная относительная энтропия $d(\alpha, \beta)$ определяется как:

$$d(\alpha, \beta) = \alpha \log \frac{\alpha}{1-\beta} + (1-\alpha) \log \frac{1-\alpha}{\beta}; \tag{1.4}$$

где α - вероятность ошибки первого рода, β - вероятность ошибки второго рода.

ГЛАВА 2
Математическая модель

2.1 Математическая модель вкраплений на основе схемы независимых испытаний

Контейнер представляет собой последовательность случайных величин распределенных по закону Бернулли с параметром p:

$$\mathcal{L}x_t = Bi(1,p), x_i \in V = 0,1, i = \overline{1,T}; \tag{2.1}$$

Вкрапляемое сообщение имеет вид:

$$\mathcal{L}m_t = Bi(1,\theta), m_i \in V = 0,1, i = \overline{1,\tau}; \tag{2.2}$$

Ключ γ_t определяет момент времени вкрапления i-того бита сообщения в исходный контейнер:

$$\mathcal{L}\gamma_t = Bi(1,\delta), \gamma_i \in V = 0,1, i = \overline{1,T}; \tag{2.3}$$

Вкрапление битов m_t производится по правилу, заданному следующим функциональным преобразованием:

$$y_t = (1-\gamma_t)x_t + \gamma_t m_{\tau_t}; \tag{2.4}$$

Лемма 2.1.1. Для модели (2.1)-(2.4)

$$P\{y_t = 1\} = (1-\delta)p + \delta\theta; \tag{2.5}$$

$$P\{y_t = 0\} = (1-\delta)(1-p) + \delta(1-\theta); \tag{2.6}$$

Доказательство. Воспользуемся формулой полной вероятности:
$P\{y_t = 1\} = P\{(1-\gamma_t)x_t + \gamma_t m_{\tau_t} = 1\} = \sum_{j\in V} P\{y_t = 1, \gamma_t = j\} = \sum_{j\in V} P\{\gamma_t = j\}P\{y_t = 1|\gamma_t = j\} = (1-\delta)P\{x_{=}1, \gamma_t = 0\} + \delta P\{m_{\tau_t} = 1, \gamma_t = 1\} = (1-\delta)p + \delta\theta;$
Тогда:
$P\{y_t = 0\} = 1 - P\{y_t = 1\} = (1-\delta)(1-p) + \delta(1-\theta);$ □

$$h = \frac{H(y_1, ..., y_t)}{T} = \frac{TH(y_1)}{T} = H(y_1); \tag{2.7}$$

Воспользуемся леммой 2.1.1:
$h = -P\{y_t = 1\} \log_2 P(y_t = 1) - P\{y_t = 0\} \log_2 P(y_t = 0) = -((1-\delta)p + \delta\theta) \log_2((1-\delta)p + \delta\theta) - ((1-\delta)(1-p) + \delta(1-\theta)) \log_2((1-\delta)(1-p) + \delta(1-\theta))$.

2.2 Математическая модель вкраплений в двоичную стационарную марковскую последовательность 1-го порядка и ее свойства

Рассмотрим модель (2.1)-(2.4).

Пусть контейнер (2.1) пердставляет собой цепь Маркова 1-го порядка с вектором распределения вероятностей $\pi = (\frac{1}{2}, \frac{1}{2})$ и матрицей вероятностей одношаговых переходов

$$P(\varepsilon) = \frac{1}{2}\begin{pmatrix}1+\varepsilon, 1-\varepsilon\\ 1-\varepsilon, 1+\varepsilon\end{pmatrix}, |\varepsilon| < 1, \varepsilon \neq 0. \tag{2.8}$$

Лемма 2.2.1. Для модели (2.1)-(2.4) с условием (2.8):

$$P\{y_{t-1} = 1, y_t = 1\} = \frac{1}{4}(1+\varepsilon)(1-\delta)^2 + \theta\delta(1-\delta) + \theta^2\delta^2; \tag{2.9}$$

$$P\{y_{t-1} = 1, y_t = 0\} = \frac{1}{4}(1-\varepsilon)(1-\delta)^2 + \frac{1}{2}\delta(1-\delta) + \theta(1-\theta)\delta^2; \tag{2.10}$$

$$P\{y_{t-1} = 0, y_t = 1\} = \frac{1}{4}(1-\varepsilon)(1-\delta)^2 + \frac{1}{2}\delta(1-\delta) + \theta(1-\theta)\delta^2; \tag{2.11}$$

$$P\{y_{t-1} = 0, y_t = 0\} = \frac{1}{4}(1+\varepsilon)(1-\delta)^2 + \delta(1-\theta)(1-\delta) + \delta^2(1-\theta)^2. \tag{2.12}$$

Доказательство. Рассмотрим биграмм: $\{y_{t-1}, y_t\}$
$(a_1, a_2) \in \{0,1\}, P\{y_{t-1} = a_1, y_t = a_2\} = \sum_{(b_1,b_2)\in\{0,1\}^2} P\{y_{t-1} = b_1, y_t = b_2, \gamma_{t-1} = a_1, \gamma_t = a_2\} = \sum_{(b_1,b_2)\in\{0,1\}^2} P\{y_{t-1} = b_1, y_t = b_2|\gamma_{t-1} = a_1, \gamma_t = a_2\}P\{\gamma_{t-1} = a_1, \gamma_t = a_2\}$.
Для (2.9):
$\sum_{(b_1,b_2)\in\{0,1\}^2} P\{y_{t-1} = b_1, y_t = b_2|\gamma_{t-1} = a_1, \gamma_t = a_2\}P\{\gamma_{t-1} = a_1, \gamma_t = a_2\} = \frac{1}{2}\cdot\frac{1}{2}(1+\varepsilon)(1-\delta)^2 + \theta\delta(1-\theta) + \theta^2\delta^2$.
Для случаев (2.10)-(2.12) доказывается аналогично. □

Для формул (2.9)-(2.12) справедливо условие нормировки:
$\sum_{(a_1,a_2)\in\{0,1\}^2} P\{y_{t-1} = a_1, y_t = a_2\} = 1$.
Далее полагаем, что $\theta = \frac{1}{2}$.
Тогда:

$$P\{y_{t-1} = 1, y_t = 1\} = P\{y_{t-1} = 0, y_t = 0\} = \delta^2\frac{\varepsilon}{4} - \delta\frac{\varepsilon}{2} + \frac{1+\varepsilon}{4}; \tag{2.13}$$

$$P\{y_{t-1} = 1, y_t = 0\} = P\{y_{t-1} = 0, y_t = 1\} = -\delta^2\frac{\varepsilon}{4} + \delta\frac{\varepsilon}{2} + \frac{1-\varepsilon}{4}. \tag{2.14}$$

Определение 2.2.1. [2] Дискретный стационарный источник называется марковским источником порядка m, если для любого l(l>m) и любой последовательности $c_l = (a_{i_1}, ..., a_{i_l})$ выполняется: $P\{a_{i_l}|a_{i_{l-1}}, ..., a_{i_1}\} = P\{a_{i_l}|a_{i_{l-1}}, ..., a_{i_{l-m+1}}\}$.

Определение 2.2.2. [2] Величина: $H^{(k)} = \sum_{\{c_k\}} P\{a_{i_1}, ..., a_{i_k}\} \log P\{a_{i_k}|a_{i_{k-1}}..., a_{i_1}\}$ называется шаговой энтропией марковского источника порядка k.

Введем понятие энтропии на знак для l-граммы:

$$H_l(\delta) = -\frac{1}{l} \sum_{(a_1,...,a_l)\in\{0,1\}} P\{y_{t-l} = a_1, ..., y_{t-1} = a_l\} \log P\{y_{t-l} = a_1, ..., y_{t-1} = a_l\}. \tag{2.15}$$

При $\delta = 0$ стегоконтейнер y совпадает с контейнером x, тогда:

$$H_l(0) = -\frac{1}{l}(H\{x_1\} + (l-1)H\{x_2|x_1\}); \tag{2.16}$$

$$\lim_{l\to\infty} H_l(0) = \lim_{l\to\infty} -\frac{1}{l}(H\{x_1\} + (l-1)H\{x_2|x_1\}) = H\{x_2|x_1\}. \tag{2.17}$$

Рассмотрим случайную величину $\xi \in B = \{b_1, ..., b_m\}$, заданную на вероятностном пространстве $(\Omega, \mathcal{F}, \mathcal{P})$, $P\{\xi = b_i\} = p_i$.

Определение 2.2.3. [2] Величина

$$I\{b_i\} = -\log p_i \tag{2.18}$$

называется собственной информацией, содержащейся в исходе $b_i \in B$.

Велиина $I\{b_i\}$ изменяется от нуля в случае реализации достоверного исхода до бесконечности, когда $p(b_i) = p_i \to 0$. Величину $I\{b_i\}$ можно интерпретировать как априорную неопределённость события $\{\xi = b_i\}$.

Случайная величина $I\{\xi\}$ имеет математическое ожидание

$$EI\{\xi\} = -\sum_{b_i\in B} p_i \log p_i. \tag{2.19}$$

Определение 2.2.4. Величина $EI\{\xi\}$ называется средней собственной информацией.

Средняя собственная информация равна энтропии: $EI\{\xi\} = H\{\xi\}$.

Для представления функции логарифма воспользуемся формулой Маклорена первого порядка:

$$f(\delta) = f(\delta_0) + (\delta - \delta_0)f'(\delta_0) + o((\delta - \delta_0)^2). \tag{2.20}$$

Для краткости, в обозначении функции log будем использовать основание b.

Согласно (2.20) в точке $\delta = 0$ справедливо асимптотическое разложение 1-го порядка:

$$\log_b(a_0(\varepsilon) + \delta a_1(\varepsilon) + \delta^2 a_2(\varepsilon) + o(\delta^2)) =$$
$$= \log a_0(\varepsilon) + \delta \left(\log(a_0(\varepsilon) + \delta a_1(\varepsilon) + \delta^2 a_2(\varepsilon) + o(\delta^2))\right)' |_{\delta=0} + o(\delta) =$$
$$= \log a_0(\varepsilon) + \delta \frac{1}{\ln b} \frac{a_1(\varepsilon)}{a_0(\varepsilon)} + o(\delta).$$

Учитывая (2.20), найдем асимптотические выражения при $\delta \to 0$ для собственной информации $I\{y_{t-1} = i_1, y_t = i_2\}, i_1, i_2 \in \{0, 1\}$:

$$I\{y_{t-1} = 0, y_t = 0\} = I\{y_{t-1} = 1, y_t = 1\} = -\log(\frac{1}{4}(1+\varepsilon)(1-\delta)^2 + \frac{1}{2}\delta(1-\delta) +$$
$$\frac{1}{4}\delta^2) = -\log\frac{1+\varepsilon}{4} + \delta\frac{1}{\ln b}\frac{2\varepsilon}{1+\varepsilon} + o(\delta);$$
$$I\{y_{t-1} = 0, y_t = 1\} = I\{y_{t-1} = 1, y_t = 0\} = -\log(\frac{1}{4}(1-\varepsilon)(1-\delta)^2 +$$
$$\frac{1}{2}\delta(1-\delta) + \frac{1}{4})\delta^2) = -\log\frac{1-\varepsilon}{4} - \delta\frac{1}{\ln b}\frac{2\varepsilon}{1-\varepsilon} + o(\delta).$$

Лемма 2.2.2. Если имеет место монобитная модель вкраплений (2.1)-(2.4), то для энтропии при $l = 2$ справедливо асимптотическое разложение 1-го порядка

$$H_2(\delta) = H_2(0) + 2\delta\varepsilon\log\frac{1+\varepsilon}{1-\varepsilon} + O(\delta^2). \tag{2.21}$$

Доказательство. $H_2(\delta) = -(P\{y_{t-1} = 0, y_t = 0\}\log(P\{y_{t-1} = 0, y_t = 0\}) + P\{y_{t-1} = 0, y_t = 1\}\log(P\{y_{t-1} = 0, y_t = 1\}) + P\{y_{t-1} = 1, y_t = 0\}\log(P\{y_{t-1} = 1, y_t = 0\}) + P\{y_{t-1} = 1, y_t = 1\}\log(P\{y_{t-1} = 1, y_t = 1\})) = -2(P\{y_{t-1} = 0, y_t = 0\}\log(P\{y_{t-1} = 0, y_t = 0\}) + P\{y_{t-1} = 0, y_t = 0\}\log(P\{y_{t-1} = 0, y_t = 1\})) = -2((\delta^2\frac{\varepsilon}{4} - \delta\frac{\varepsilon}{2} + \frac{1+\varepsilon}{4})(-\log(\frac{1+\varepsilon}{4}) + \delta\frac{1}{\ln b}\cdot\frac{2\varepsilon}{1+\varepsilon} + o(\delta^2)) + (-\delta^2\frac{\varepsilon}{4} + \delta\frac{\varepsilon}{2} + \frac{1-\varepsilon}{4})(-\log(\frac{1-\varepsilon}{4}) - \delta\frac{1}{\ln b}\cdot\frac{2\varepsilon}{1-\varepsilon} + o(\delta^2))) = \frac{1}{2}(-(1+\varepsilon)\log(\frac{1+\varepsilon}{4}) - (1-\varepsilon)\log(\frac{1-\varepsilon}{4}) + 2\delta\varepsilon\log(\frac{1+\varepsilon}{1-\varepsilon}))) + O(\delta^2) = H_2(0) + 2\delta\varepsilon\log(\frac{1+\varepsilon}{1-\varepsilon}))) + O(\delta^2).$ □

Определение 2.2.5. [2] Величина $\lim_{k\to\infty} H^{(k)} = \lim_{k\to\infty} H_k = H_\infty \geqslant 0$ называется энтропий марковского источника, где H_k - энтропия на знак.

Рассмотрим условные вероятности появления трехграммы $(0, 0, 0)$ при условии всевозможных стегоключей.

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0\} = (1-\delta)^3 P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0 | \gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 0\} +$
$+ \delta(1-\delta)^2(P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0 | \gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 0\} + P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0 | \gamma_{i-1} = 0, \gamma_i = 1, \gamma_{i+1} = 0\} + P\{y_{i-1} = 0, y_i = 0, y_{i+1} =$

$0|\gamma_{i-1}=0,\gamma_i=0,\gamma_{i+1}=1\}) +$
$+\delta^2(1-\delta)(P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_{i-1}=1,\gamma_i=1,\gamma_{i+1}=0\}+P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_{i-1}=0,\gamma_i=1,\gamma_{i+1}=1\}+P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_{i-1}=1,\gamma_i=0,\gamma_{i+1}=1\}) +$
$+\delta^3(P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_{i-1}=1,\gamma_i=1,\gamma_{i+1}=1\}).$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=0,\gamma_2=0,\gamma_3=0\}=P\{x_{i-1}=0,x_i=0,x_{i+1}=0\}=P\{x_{i-1}=0\}P\{x_i=0,x_{i+1}=0|x_{i-1}=0\}=P\{x_{i-1}=0\}P\{x_i=0\}P\{x_{i+1}=0|x_i=0\}=\frac{1}{2}\cdot\frac{1}{2}(1+\varepsilon)\cdot\frac{1}{2}(1+\varepsilon)=\frac{1}{8}(1+\varepsilon)^2;$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=1,\gamma_2=0,\gamma_3=0\}=P\{\xi=0,x_i=0,x_{i+1}=0\}=P\{\xi=0\}P\{x_i=0,x_{i+1}=0\}=P\{\xi=0\}P\{x_i=0\}P\{x_{i+1}=0|x_i=0\}=\frac{1}{2}\cdot\frac{1}{2}(1+\varepsilon)\cdot\frac{1}{2}=\frac{1}{8}(1+\varepsilon);$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=0,\gamma_2=1,\gamma_3=0\}=P\{x_{i-1}=0,\xi=0,x_{i+1}=0\}=P\{\xi=0\}P\{x_{i-1}=0,x_{i+1}=0\}=\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}(1+\varepsilon^2)=\frac{1}{8}(1+\varepsilon^2);$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=0,\gamma_2=0,\gamma_3=1\}=P\{x_{i-1}=0,x_i=0,\xi=0\}=P\{\xi=0\}P\{x_{i-1}=0,x_i=0\}=\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}(1+\varepsilon)=\frac{1}{8}(1+\varepsilon);$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=0,\gamma_2=1,\gamma_3=1\}=P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=1,\gamma_2=0,\gamma_3=1\}=P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=1,\gamma_2=1,\gamma_3=0\}=P\{y_{i-1}=0,y_i=0,y_{i+1}=0|\gamma_1=1,\gamma_2=1,\gamma_3=1\}=P\{\xi=0,\xi=0,\xi=0\}=P\{\xi=1\}P\{x_{i-1}=1\}P\{x_i=0\}=\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}=\frac{1}{8};$

$P\{y_{i-1}=0,y_i=0,y_{i+1}=0\}=\frac{1}{8}\big(\varepsilon(\varepsilon+2)\delta^2-2\varepsilon(\varepsilon+1)\delta+(1+\varepsilon)^2\big).$

Найдем вероятности появления всевозможных трехграмм в стегоконтейнере $\{y_t\}$:

$$P\{y_{i-1}=1,y_i=1,y_{i+1}=1\}=P\{y_{i-1}=0,y_i=0,y_{i+1}=0\}=$$
$$=\tfrac{1}{8}\big(\varepsilon(\varepsilon+2)\delta^2-2\varepsilon(\varepsilon+2)\delta+(1+\varepsilon)^2\big),$$
$$P\{y_{i-1}=1,y_i=1,y_{i+1}=0\}=$$
$$=P\{y_{i-1}=0,y_i=0,y_{i+1}=1\}=P\{y_{i-1}=0,y_i=1,y_{i+1}=1\}=$$
$$=P\{y_{i-1}=1,y_i=0,y_{i+1}=0\}=\tfrac{1}{8}\big(-\varepsilon^2\delta^2+2\varepsilon^2\delta-\varepsilon^2+1\big),$$
$$P\{y_{i-1}=1,y_i=0,y_{i+1}=1\}=P\{y_{i-1}=0,y_i=1,y_{i+1}=0\}=$$
$$=\tfrac{1}{8}\big(\varepsilon(\varepsilon-2)\delta^2-2\varepsilon(\varepsilon-2)\delta+(1-\varepsilon)^2\big).$$

Теорема 2.2.1. Если имеет место монобитная модель вкраплений (2.1)-(2.4) , то для энтропии при $l=3$ справедливо асимптотическое разложение 1-го

порядка:

$$H_3(\delta) = H_3(0) + 2\varepsilon\delta \log \frac{1+\varepsilon}{1-\varepsilon} + O(\delta^2); \tag{2.22}$$

собственная информация имеет вид:

$$I\{y_{i-1}=0, y_i=0, y_{i+1}=0\} = -\log \frac{(1+\varepsilon)^2}{8} + \delta \frac{1}{\ln b} \cdot \frac{2\varepsilon^2+4\varepsilon}{(1+\varepsilon)^2} + O(\delta^2),$$

$$I\{y_{i-1}=1, y_i=0, y_{i+1}=0\} = I\{y_{i-1}=0, y_i=0, y_{i+1}=1\} =$$

$$= -\log \frac{1-\varepsilon^2}{8} - \delta \frac{1}{\ln b} \cdot \frac{2\varepsilon^2}{1-\varepsilon^2} + O(\delta^2),$$

$$I\{y_{i-1}=0, y_i=1, y_{i+1}=0\} = -\log \frac{(1-\varepsilon)^2}{8} + \delta \frac{1}{\ln b} \cdot \frac{2\varepsilon^2-4\varepsilon}{(1-\varepsilon)^2} + O(\delta^2);$$

$$I\{y_{i-1}=j_1, y_i=j_2, y_{i+1}=j_3\} = I\{y_{i-1}=1-j_1, y_i=1-j_2, y_{i+1}=1-j_3\},$$
$$j_1, j_2, j_3 \in \{0,1\}.$$

Доказательство. Подставляя в (2.20) найденные выражения для вероятностей трехграмм, получим асимптотические выражения для собственной информации.

Используя выражения для собственной информации, получим:
$H_3(\delta) = -2(\frac{1}{8}\big(\varepsilon(\varepsilon+2)\delta^2 - 2\varepsilon(\varepsilon+2)\delta + (1+\varepsilon)^2\big)(\log(\frac{(1+\varepsilon)^2}{8}) + \delta\frac{1}{\ln b}\cdot\frac{-2\varepsilon^2-4\varepsilon}{(1+\varepsilon)^2}) + 2\frac{1}{8}\big(-\varepsilon^2\delta^2+2\varepsilon^2\delta-\varepsilon^2+1\big)(\log(\frac{1-\varepsilon^2}{8})+\delta\frac{1}{\ln b}\cdot\frac{2\varepsilon^2}{1-\varepsilon^2}) + \frac{1}{8}\big(\varepsilon(\varepsilon-2)\delta^2-2\varepsilon(2+\varepsilon)\delta+(1-\varepsilon)^2\big)(\log(\frac{(1-\varepsilon)^2}{8}) + \delta\frac{1}{\ln b}\cdot\frac{-2\varepsilon^2+4\varepsilon}{(1-\varepsilon)^2})) + O(\delta^2) = -((1-\varepsilon)\log(1-\varepsilon) + (1+\varepsilon)\log(1+\varepsilon) + \log(\frac{1}{8}) + 2\varepsilon\delta\log\frac{1-\varepsilon}{1+\varepsilon}) + O(\delta^2) = H_3(0) - 2\varepsilon\delta\log\frac{1-\varepsilon}{1+\varepsilon} + O(\delta^2).$ □

Рассмотрим условные вероятности появления четырехграммы $(0,0,0,0)$ при условии всевозможных стегоключей.

$P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0\} = (1-\delta)^4 P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=0, \gamma_{i+1}=0, \gamma_{i+2}=0\} + \delta(1-\delta)^3\Big(P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=0, \gamma_{i+1}=0, \gamma_{i+2}=1\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=0, \gamma_{i+1}=1, \gamma_{i+2}=0\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=1, \gamma_{i+1}=0, \gamma_{i+2}=0\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=1, \gamma_i=0, \gamma_{i+1}=0, \gamma_{i+2}=0\}\Big) + \delta^2(1-\delta)^2\Big(P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=0, \gamma_{i+1}=1, \gamma_{i+2}=1\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=1, \gamma_{i+1}=1, \gamma_{i+2}=0\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=1, \gamma_i=1, \gamma_{i+1}=0, \gamma_{i+2}=0\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=1, \gamma_i=0, \gamma_{i+1}=1, \gamma_{i+2}=0\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=0, \gamma_i=1, \gamma_{i+1}=0, \gamma_{i+2}=1\} + P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=1, \gamma_i=0, \gamma_{i+1}=0, \gamma_{i+2}=1\}\Big) + \delta^3(1-\delta)\Big(P\{y_{i-1}=0, y_i=0, y_{i+1}=0, y_{i+2}=0 | \gamma_{i-1}=$

$0, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = 1\} + P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 1, \gamma_{i+2} = 1\} + P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} + P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = 0\}\Big) + \delta^4(P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = 1\}).$

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 0, \gamma_{i+2} = 0\} = \frac{(1+\varepsilon)^3}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 0, \gamma_{i+2} = 0\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} = \frac{(1+\varepsilon)^2}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 1, \gamma_{i+1} = 0, \gamma_{i+2} = 0\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 1, \gamma_{i+2} = 0\} = \frac{(1+\varepsilon)(1+\varepsilon^2)}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 0, \gamma_{i+2} = 0\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 0, \gamma_{i+1} = 1, \gamma_{i+2} = 1\} = \frac{1+\varepsilon}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 1, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 1, \gamma_{i+2} = 0\} = \frac{1+\varepsilon^2}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = 0\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 0, \gamma_{i+2} = 1\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 0, \gamma_{i+1} = 1, \gamma_{i+2} = 1\} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 0, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0|\gamma_{i-1} = 1, \gamma_i = 1, \gamma_{i+1} = 1, \gamma_{i+2} = 1\} = \frac{1}{16}$;

$P\{y_{i-1} = 0, y_i = 0, y_{i+1} = 0, y_{i+2} = 0\} = P\{y_{i-1} = 1, y_i = 1, y_{i+1} = 1, y_{i+2} = 1\} = \frac{1}{16}\big(\delta^4\varepsilon^2 + \delta^3(-4\varepsilon^2) + \delta^2(\varepsilon^3 + 8\varepsilon^2 + 3\varepsilon) + \delta(-2\varepsilon^3 - 8\varepsilon^2 - 6\varepsilon) + \varepsilon^3 + 3\varepsilon^2 + 3\varepsilon + 1\big)$.
Найдем вероятности появления всевозможных четырехграмм в стегоконтей-

нере $\{y_t\}$:

$$P\{y_{i-1}=0,y_i=0,y_{i+1}=0,y_{i+2}=1\}=P\{y_{i-1}=1,y_i=1,y_{i+1}=1,y_{i+2}=0\}=$$
$$P\{y_{i-1}=1,y_i=0,y_{i+1}=0,y_{i+2}=0\}=P\{y_{i-1}=0,y_i=1,y_{i+1}=1,y_{i+2}=1\}=$$
$$\tfrac{1}{16}\big(\delta^4(-\varepsilon^2)+\delta^3\cdot 4\varepsilon^2+\delta^2(-\varepsilon^3-6\varepsilon^2+\varepsilon)+\delta(2\varepsilon^3+4\varepsilon^2-2\varepsilon)-\varepsilon^3-\varepsilon^2+\varepsilon+1\big);$$
$$P\{y_{i-1}=0,y_i=1,y_{i+1}=0,y_{i+2}=0\}=P\{y_{i-1}=1,y_i=0,y_{i+1}=1,y_{i+2}=1\}=$$
$$P\{y_{i-1}=0,y_i=0,y_{i+1}=1,y_{i+2}=0\}=P\{y_{i-1}=1,y_i=1,y_{i+1}=0,y_{i+2}=1\}=$$
$$\tfrac{1}{16}\big(\delta^4(-\varepsilon^2)+\delta^3\cdot 4\varepsilon^2+\delta^2(\varepsilon^3-6\varepsilon^2-\varepsilon)+\delta(-2\varepsilon^3+4\varepsilon^2+2\varepsilon)+\varepsilon^3-\varepsilon^2-\varepsilon+1\big);$$
$$P\{y_{i-1}=0,y_i=0,y_{i+1}=1,y_{i+2}=1\}=P\{y_{i-1}=1,y_i=1,y_{i+1}=0,y_{i+2}=0\}=$$
$$\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(-\varepsilon^3+4\varepsilon^2+\varepsilon)+\delta(2\varepsilon^3-2\varepsilon)-\varepsilon^3-\varepsilon^2+\varepsilon+1\big);$$
$$P\{y_{i-1}=0,y_i=1,y_{i+1}=1,y_{i+2}=0\}=P\{y_{i-1}=1,y_i=0,y_{i+1}=0,y_{i+2}=1\}=$$
$$\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(\varepsilon^3+4\varepsilon^2-\varepsilon)+\delta(-2\varepsilon^3+2\varepsilon)+\varepsilon^3-\varepsilon^2-\varepsilon+1\big);$$
$$P\{y_{i-1}=1,y_i=0,y_{i+1}=1,y_{i+2}=0\}=P\{y_{i-1}=0,y_i=1,y_{i+1}=0,y_{i+2}=1\}=$$
$$=\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(-\varepsilon^3+8\varepsilon^2-3\varepsilon)+\delta(2\varepsilon^3-8\varepsilon^2+6\varepsilon)$$
$$-\varepsilon^3+3\varepsilon^2-3\varepsilon+1\big).$$

Теорема 2.2.2. Если имеет место монобитная модель вкраплений (2.1)-(2.4) , то для энтропии при $l=4$ справедливо асимптотическое разложение 1-го порядка:

$$H_4(\delta)=H_4(0)+\frac{24\varepsilon\delta}{16}\log\frac{1+\varepsilon}{1-\varepsilon}+O(\delta^2); \tag{2.23}$$

собственная информация имеет вид:

$$I\{y_{i-1}=0,y_i=0,y_{i+1}=0,y_{i+2}=0\}=I\{y_{i-1}=1,y_i=1,y_{i+1}=1,y_{i+2}=1\}=$$
$$=-\left(\log\frac{(1+\varepsilon)^3}{16}+\delta\frac{-2\varepsilon^3-8\varepsilon^2-6\varepsilon}{(1+\varepsilon)^3\ln b}\right)+O(\delta^2);$$
$$I\{y_{i-1}=0,y_i=0,y_{i+1}=0,y_{i+2}=1\}=I\{y_{i-1}=1,y_i=1,y_{i+1}=1,y_{i+2}=0\}=$$
$$I\{y_{i-1}=1,y_i=0,y_{i+1}=0,y_{i+2}=0\}=I\{y_{i-1}=0,y_i=1,y_{i+1}=1,y_{i+2}=1\}=$$
$$=-\left(\log\frac{(1-\varepsilon)(1+\varepsilon)^2}{16}+\delta\frac{2\varepsilon^3+4\varepsilon^2-2\varepsilon}{(1-\varepsilon)(1+\varepsilon)^2\ln b}\right)+O(\delta^2);$$
$$I\{y_{i-1}=0,y_i=0,y_{i+1}=1,y_{i+2}=0\}=I\{y_{i-1}=1,y_i=1,y_{i+1}=0,y_{i+2}=1\}=$$
$$I\{y_{i-1}=0,y_i=1,y_{i+1}=0,y_{i+2}=0\}=I\{y_{i-1}=1,y_i=0,y_{i+1}=1,y_{i+2}=1\}=$$
$$=-\left(\log\frac{(1-\varepsilon)^2(1+\varepsilon)}{16}+\delta\frac{-2\varepsilon^3+4\varepsilon^2+2\varepsilon}{(1-\varepsilon)^2(1+\varepsilon)\ln b}\right)+O(\delta^2);$$

$$I\{y_{i-1}=0,y_i=0,y_{i+1}=1,y_{i+2}=1\}=I\{y_{i-1}=1,y_i=1,y_{i+1}=0,y_{i+2}=0\}=$$
$$=-\left(\log\frac{(1-\varepsilon)(1+\varepsilon)^2}{16}+\delta\frac{2\varepsilon^3-2\varepsilon}{(1-\varepsilon)(1+\varepsilon)^2\ln b}\right)+O(\delta^2);$$
$$I\{y_{i-1}=0,y_i=1,y_{i+1}=1,y_{i+2}=0\}=I\{y_{i-1}=1,y_i=0,y_{i+1}=0,y_{i+2}=1\}=$$
$$-\left(\log\frac{(1-\varepsilon)^2(1+\varepsilon)}{16}+\delta\frac{-2\varepsilon^3+2\varepsilon}{(1-\varepsilon)^2(1+\varepsilon)\ln b}\right)+O(\delta^2);$$
$$I\{y_{i-1}=1,y_i=0,y_{i+1}=1,y_{i+2}=0\}=I\{y_{i-1}=0,y_i=1,y_{i+1}=0,y_{i+2}=1\}=$$
$$-\left(\log\frac{(1-\varepsilon)^3}{16}+\delta\frac{2\varepsilon^3-8\varepsilon^2+6\varepsilon}{(1-\varepsilon)^3\ln b}\right)+O(\delta^2).$$

Доказательство. Подставляя в (2.20) найденные выражения для вероятностей четырехграмм, получим асимптотические выражения для собственной информации.

Используя выражения для собственной информации, получим:

$$H_4(\delta)=-2\Big(\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(\varepsilon^3+8\varepsilon^2+3\varepsilon)+\delta(-2\varepsilon^3-8\varepsilon^2-6\varepsilon)+\varepsilon^3+3\varepsilon^2+3\varepsilon+1\big)\big(\log\tfrac{(1+\varepsilon)^3}{16}+\delta\tfrac{-2\varepsilon^3-8\varepsilon^2-6\varepsilon}{(1+\varepsilon)^3\ln b}+O(\delta^2)\big)+2\tfrac{1}{16}\big(\delta^4(-\varepsilon^2)+\delta^3\cdot4\varepsilon^2+\delta^2(-\varepsilon^3-6\varepsilon^2+\varepsilon)+\delta(2\varepsilon^3+4\varepsilon^2-2\varepsilon)-\varepsilon^3-\varepsilon^2+\varepsilon+1\big)\big(\log\tfrac{(1-\varepsilon)(1+\varepsilon)^2}{16}+\delta\tfrac{2\varepsilon^3+4\varepsilon^2-2\varepsilon}{(1-\varepsilon)(1+\varepsilon)^2\ln b}+O(\delta^2)\big)+2\tfrac{1}{16}\big(\delta^4(-\varepsilon^2)+\delta^3\cdot4\varepsilon^2+\delta^2(\varepsilon^3-6\varepsilon^2-\varepsilon)+\delta(-2\varepsilon^3+4\varepsilon^2+2\varepsilon)+\varepsilon^3-\varepsilon^2-\varepsilon+1\big)\big(\log\tfrac{(1-\varepsilon)^2(1+\varepsilon)}{16}+\delta\tfrac{-2\varepsilon^3+4\varepsilon^2+2\varepsilon}{(1-\varepsilon)^2(1+\varepsilon)\ln b}+O(\delta^2)\big)+\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(-\varepsilon^3+4\varepsilon^2+\varepsilon)+\delta(2\varepsilon^3-2\varepsilon)-\varepsilon^3-\varepsilon^2+\varepsilon+1\big)\big(log\tfrac{(1-\varepsilon)(1+\varepsilon)^2}{16}+\delta\tfrac{2\varepsilon^3-2\varepsilon}{(1-\varepsilon)(1+\varepsilon)^2\ln b}+O(\delta^2)\big)+\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(\varepsilon^3+4\varepsilon^2-\varepsilon)+\delta(-2\varepsilon^3+2\varepsilon)+\varepsilon^3-\varepsilon^2-\varepsilon+1\big)\big(\log\tfrac{(1-\varepsilon)^2(1+\varepsilon)}{16}+\delta\tfrac{-2\varepsilon^3+2\varepsilon}{(1-\varepsilon)^2(1+\varepsilon)\ln b}+O(\delta^2)\big)+\tfrac{1}{16}\big(\delta^4\varepsilon^2+\delta^3(-4\varepsilon^2)+\delta^2(-\varepsilon^3+8\varepsilon^2-3\varepsilon)+\delta(2\varepsilon^3-8\varepsilon^2+6\varepsilon)-\varepsilon^3+3\varepsilon^2-3\varepsilon+1\big)\big(\log\tfrac{(1-\varepsilon)^3}{16}+\delta\tfrac{2\varepsilon^3-8\varepsilon^2+6\varepsilon}{(1-\varepsilon)^3\ln b}+O(\delta^2)\big)\Big)=\tfrac{(1+\varepsilon)^3}{16}\log\tfrac{(1+\varepsilon)^3}{16}+3\tfrac{(1+\varepsilon)^2(1-\varepsilon)}{16}\log\tfrac{(1+\varepsilon)^2(1-\varepsilon)}{16}+3\tfrac{(1+\varepsilon)(1-\varepsilon)^2}{16}\log\tfrac{(1+\varepsilon)(1-\varepsilon)^2}{16}+\tfrac{(1-\varepsilon)^3}{16}\log\tfrac{(1-\varepsilon)^3}{16}+\tfrac{24\varepsilon\delta}{16}\log\tfrac{1+\varepsilon}{1-\varepsilon}+O(\delta^2)=H_4(0)+\tfrac{24\varepsilon\delta}{16}\log\tfrac{1+\varepsilon}{1-\varepsilon}+O(\delta^2).\qquad\square$$

Оценим остаточный член для асимптотического выражения энтропии биграммы:

$$r_n(\delta)=\frac{f^{(n+1)}(\overline{\delta})}{(n+1)!}(\delta-\delta_0),\overline{\delta}\in[\delta_0,\delta].\tag{2.24}$$

Для асимптотического разложения 1-го порядка при $\delta_0=0$ остаточный член имеет вид:

$$r_n(\delta)=\frac{f''(\overline{\delta})}{2}\delta,\overline{\delta}\in[0,\delta].\tag{2.25}$$

$$(\log(P_{00}))''=\tfrac{-2\varepsilon(\delta^2\varepsilon-2\delta\varepsilon+\varepsilon-1)}{\ln b(\delta^2\varepsilon-2\delta\varepsilon+\varepsilon+1)^2};$$

$$(\log(P_{01}))'' = \frac{-2\varepsilon(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon + 1)}{\ln b(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon - 1)^2};$$

Тогда остаточный член для $\log(P_{00}) = \log(P_{11})$ равен:

$$r_{n_{00}}(\delta) = \frac{\delta}{2} \cdot \frac{-2\varepsilon(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon - 1)}{\ln b(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon + 1)^2}|_{\delta=\overline{\delta}} \tag{2.26}$$

Остаточный член для $\log(P_{10}) = \log(P_{01})$ равен:

$$r_{n_{10}}(\delta) = \frac{\delta}{2} \cdot \frac{-2\varepsilon(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon + 1)}{\ln b(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon - 1)^2}|_{\delta=\overline{\delta}} \tag{2.27}$$

Тогда на основании (2.26) и (2.27) остаточный член для энтропиии будет равен:

$$R_n(\delta) = -2(P_{00}r_{n_{00}}(\delta) + P_{10}r_{n_{01}}(\delta)) = -2\big((\delta^2\frac{\varepsilon}{4} - \delta\frac{\varepsilon}{2} + \frac{1+\varepsilon}{4})$$
$$(\frac{\delta}{2} \cdot \frac{-2\varepsilon(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon - 1)}{\ln b(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon + 1)^2}|_{\delta=\overline{\delta}}) +$$
$$(-\delta^2\frac{\varepsilon}{4} + \delta\frac{\varepsilon}{2} + \frac{1-\varepsilon}{4})(\frac{\delta}{2} \cdot \frac{-2\varepsilon(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon + 1)}{\ln b(\delta^2\varepsilon - 2\delta\varepsilon + \varepsilon - 1)^2}|_{\delta=\overline{\delta}})\big)$$

2.3 Линейный дискриминантный анализ

Линейный дискриминантный анализ (ЛДА), а также связанный с ним линейный дискриминант Фишера — методы статистики и машинного обучения, применяемые для нахождения линейных комбинаций признаков, наилучшим образом разделяющих два или более класса объектов или событий. Полученная комбинация может быть использована в качестве линейного классификатора или для сокращения размерности пространства признаков перед последующей классификацией. ЛДА тесно связан с дисперсионным анализом и регрессионным анализом, также пытающимися выразить какую-либо зависимую переменную через линейную комбинацию других признаков или измерений. В этих двух методах зависимая переменная — численная величина, а в ЛДА она является величиной номинальной (меткой класса). Помимо того, ЛДА имеет схожие черты с методом главных компонент и факторным анализом, которые ищут линейные комбинации величин, наилучшим образом описывающие данные. Для использования ЛДА признаки должны быть непрерывными величинами, иначе следует использовать анализ соответствий (англ. Discriminant Correspondece Analysis). [1]

2.3.1 Линейный дискриминантный анализ для случая двух классов

Для каждого образца объекта или события с известным классом y рассматривается набор наблюдений x (называемых ещё признаками, переменными или измерениями). Набор таких образцов называется обучающей выборкой (или набором обучения, обучением). Задачи классификации состоит в том, чтобы построить хороший прогноз класса y для всякого так же распределённого объекта (не обязательно содержащегося в обучающей выборке), имея только наблюдения x.
При ЛДА предполагается, что функции совместной плотности распределения вероятностей $p(\vec{x}|y = 1)$ и $p(\vec{x}|y = 0)$ - нормальны. В этих предположениях оптимальное байесовское решение - относить точки ко второму классу если отношение правдоподобия ниже некоторого порогового значения T:

$$(\vec{x} - \vec{\mu}_0)^T \Sigma_{y=0}^{-1} (\vec{x} - \vec{\mu}_0) + \ln |\Sigma_{y=0}| - (\vec{x} - \vec{\mu}_1)^T \Sigma_{y=1}^{-1} (\vec{x} - \vec{\mu}_1) - \ln |\Sigma_{y=0}| < T$$

Если не делается никаких дальнейших предположений, полученную задачу классификации называют квадратичным дискриминантным анализом (англ. quadratic discriminant analysis, QDA). В ЛДА делается дополнительное предположение о гомоскедастичности (т.е. предполагается, что ковариационные матрицы равны, $\Sigma_{y=0} = \Sigma_{y=1} = \Sigma$) и считается, что ковариационные матрицы имеют полный ранг. При этих предположениях задача упрощается и

сводится к сравнению скалярного произведения с пороговым значением

$$\vec{\omega} \cdot \vec{x} < c$$

для некоторой константы c, где

$$\vec{\omega} = \Sigma^{-1}(\vec{\mu_1} - \vec{\mu_0}).$$

Это означает, что вероятность принадлежности нового наблюдения x к классу y зависит исключительно от линейной комбинации известных наблюдений.

2.3.2 Результаты линейного дискриминантного анализа

Линейный дискриминантный анализ применен для классификации последовательностей с вкраплениями и без вкраплений при фиксированном параметре ε.
Пусть имеется последовательность $Y = \{y_1, ..., y_T\}$, на основании Y вычисляем $(H_3(\delta), H_4(\delta))$ при фиксированном ε, тогда:
H_0: последовательность Y имеет вкрапления
H_1: последовательность Y не имеет вкраплений
тогда для $n = n_0 + n_1$, (где n_0 - количество заведомо пустых последовательностей, n_1 - количество последовательностей с вкрапениями) последовательностей можно провести дискриминантный анализ и оценить вероятновсть правильной классификации и мощность критерия.
Тогда:

$$\hat{\alpha} = \frac{n_0 - \nu_0}{n_0} - \text{оценка вероятности ошибки первого рода;} \tag{2.28}$$

$$\hat{\beta} = \frac{n_1 - \nu_1}{n_1} - \text{оценка вероятности ошибки второго рода;} \tag{2.29}$$

ν_0 - количество верно отпределенных пустых последовательностей, ν_1 - количество верно отпределенных последоваательностей с вкраплениями.
Мощность критерия:

$$\hat{w} = \frac{\nu_1}{n_1} \tag{2.30}$$

δ	$\hat{\alpha}$	$\hat{\beta}$	$\hat{w}$
0.03	0.42	0.31	0.69
0.07	0.21	0.14	0.86
0.08	0.1	0.13	0.87
0.09	0.14	0.08	0.92
0.1	0.13	0.05	0.95
0.3	0.05	0.02	0.98

Таблица 2.1 — Результаты дискриминантного анализа при $\varepsilon = 0.55$.

δ	$\hat{\alpha}$	$\hat{\beta}$	$\hat{w}$
0.01	0.47	0.4	0.6
0.03	0.3	0.2	0.8
0.05	0.18	0.16	0.84
0.07	0.1	0.08	0.92
0.1	0.06	0.05	0.95
0.3	0.02	0.02	0.98

Таблица 2.2 — Результаты дискриминантного анализа при $\varepsilon = 0.15$.

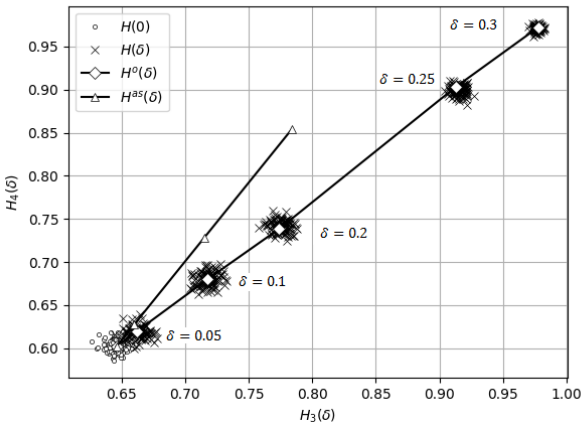

Рисунок 2.1 — График зависимости энтропии $H_4(\delta)$ от $H_3(\delta)$ при различных долях вкраплений

2.3.3 Вывод

Методом линейного дискриминантного анализа на основании энтропийных характеристик $(H_3(\delta), H_4(\delta))$ можно определить наличие вкраплений в последовательность с вероятностью ошибки второго рода 0.05 при доле вкраплений $\delta \geq 0.1$.

2.4 Исследование реальных данных на основании изображений в формате JPEG

Формат файла JPEG (Joint Photographic Experts Group - Объединенная экспертная группа по фотографии, произносится "джейпег) был разработан компанией C-Cube Microsystems как эффективный метод хранения изображений с большой глубиной цвета, например, получаемых при сканировании фотографий с многочисленными едва уловимыми (а иногда и неуловимыми) оттенками цвета. Основное отличие формата JPEG от других форматов состоит в том, что в JPEG используется алгоритм сжатия с потерями (а не алгоритм без потерь) информации. Алгоритм сжатия без потерь так сохраняет информацию об изображении, что распакованное изображение в точности соответствует оригиналу. При сжатии с потерями теряется часть информации об изображении, чтобы достичь большего сжатия. Распакованное изображение JPEG редко соответствует оригиналу абсолютно точно, но очень часто эти различия столь незначительны, что их едва можно (если вообще можно) обнаружить.

Ключевым компонентом работы алгоритма является дискретное косинусное преобразование. Дискретное косинусное преобразование (2.31) представляет собой разновидность преобразования Фурье и, так же как и оно, имеет обратное преобразование. Графическое изображение можно рассматривать как совокупность пространственных волн, причем оси X и Y совпадают с шириной и высотой картинки, а по оси Z откладывается значение цвета соответствующего пикселя изображения. Дискретное косинусное преобразование позволяет переходить от пространственного представления картинки к ее спектральному представлению и обратно. Воздействуя на спектральное представление картинки, состоящее из "гармоник", то есть, отбрасывая наименее значимые из них, можно балансировать между качеством воспроизведения и степенью сжатия [5].

$$Y[u,v] = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i,u)C(j,v)y(i,j), \tag{2.31}$$

где $C(i,u) = A(u)cos(\frac{(2i+1)u\pi}{2N})$ – гармоника сигнала,

$A(u) = \begin{cases} \frac{1}{\sqrt{2}}, \text{u=0;} \\ 1, u \neq 0; \end{cases}$ - постоянная составляющая.

Обратное дискретное косинусное преобразование:

$$y(i,j) = \frac{1}{\sqrt{2N}} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(i,u)C(j,v)Y[u,v]. \tag{2.32}$$

В получившейся матрице коэффициентов низкочастотные компоненты расположены ближе к левому верхнему углу, а высокочастотные - справа и внизу. Это важно потому, что большинство графических образов на экране компьютера состоит из низкочастотной информации. Высокочастотные компоненты не так важны для передачи изображения. Таким образом, дискретное косинусное преобразование позволяет определить, какую часть информации можно безболезненно выбросить, не внося серьезных искажений в картинку.

Далее происходит квантование коэффициентов ДКП. Здесь каждое число из матрицы делится на элемент из таблицы квантования, а результат округляется до ближайшего целого:

$$Y^q(u,v) = Round\left(\frac{Y(u,v)}{q(u,v)}\right), \tag{2.33}$$

где $q(u,v)$ - элемент таблицы квантования

Стандарт JPEG даже допускает использование собственных таблиц квантования, которые, однако, необходимо будет передавать декодеру вместе со сжатыми данными, что увеличит общий размер файла. Понятно, что пользователю сложно самостоятельно подобрать 64 коэффициента, поэтому стандарт JPEG использует два подхода для матриц квантования. Первый заключается в том, что в стандарт JPEG включены две рекомендуемые таблицы квантования: одна для яркости, вторая для цветности. Второй подход заключается в синтезе (вычислении на лету) таблицы квантовании, зависящей от одного параметра , который задается пользователем. Сама таблица строится по формуле:

$$Q(i,j) = 1 + (i+j)R, \tag{2.34}$$

где R - коэффициент сжатия.

Чем больше коэффициент квантования, тем больше данных теряется, поскольку реальное ДКП-значение представляется все менее и менее точно. Кроме того, для данных яркости и цветности применяются отдельные таблицы квантования, позволяющие квантовать данные цветности с большими коэффициентами, чем данные яркости. Таким образом, JPEG использует различную чувствительность глаза к яркости и цветности изображения.

Далее полученная матрица 8×8 переводится в 64-элементный вектор при помощи "зигзаг" -сканирования (Рис. 2.2). Таким образом, в начале вектора мы получаем коэффициенты матрицы, соответствующие низким частотам, а в конце - высоким.

Заключительна стадия алгоритма сжатия JPEG - кодирование. Полученный вектор обрабатывается с помощью алгоритмов Хаффмана или арифметического кодирования, в зависимости от реализации.

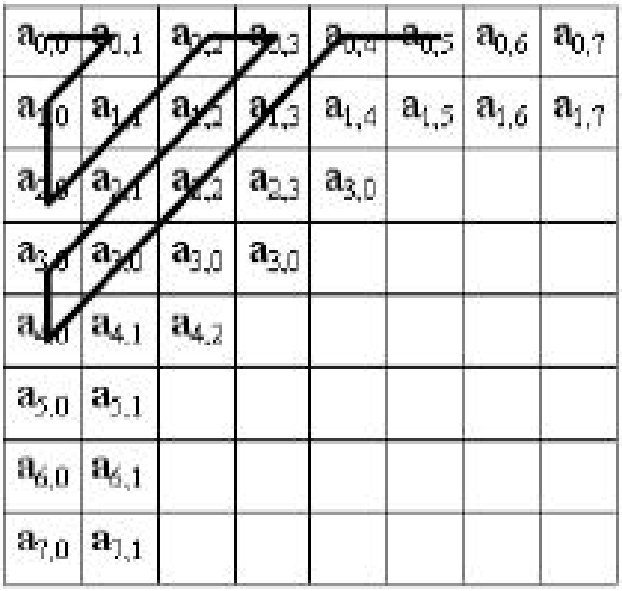

Рисунок 2.2 — "Зигзаг" -сканирование матрицы.

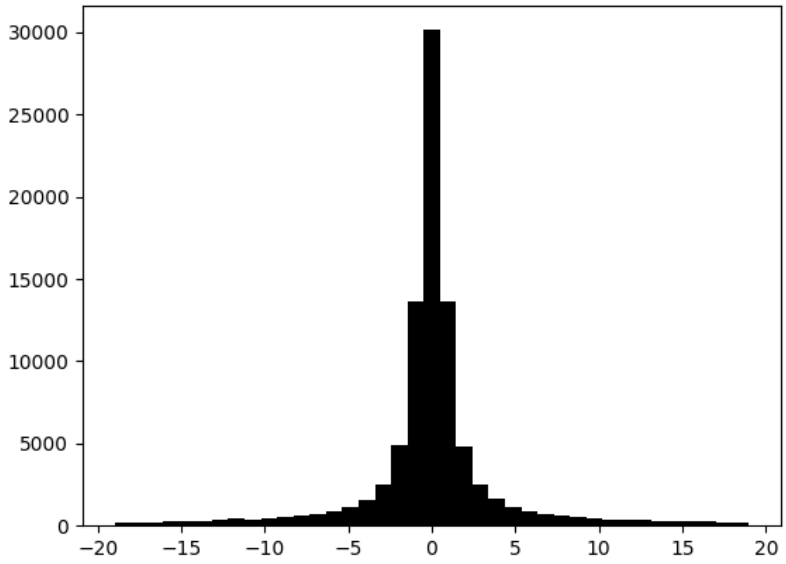

Рисунок 2.3 — Гистограмма ДКП коэффициентов jpeg изображения.

ГЛАВА 3
Компьютерные эксперименты

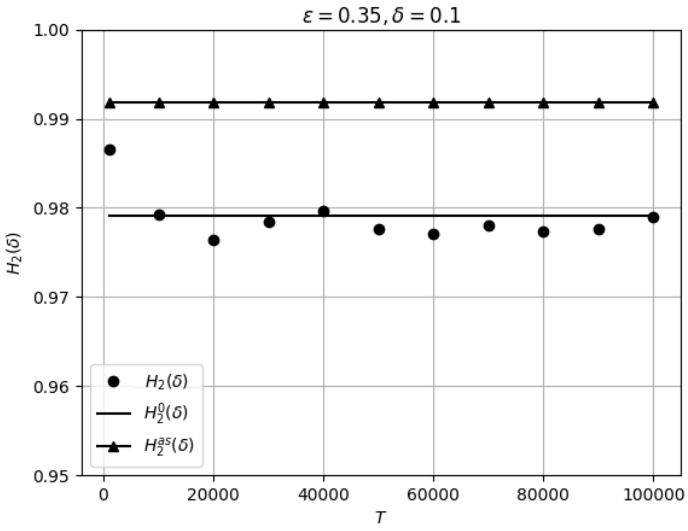

Рисунок 3.1 — График зависимости энтропии $H_2(\delta)$ от длины последовательности при $\delta = 0.1$

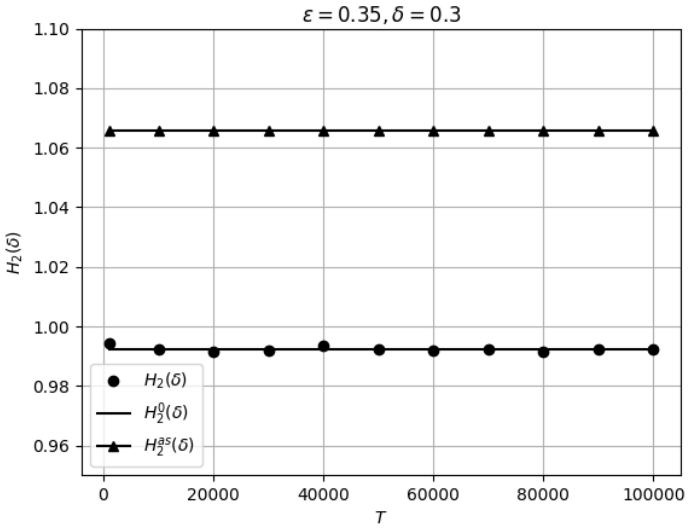

Рисунок 3.2 — График зависимости энтропии $H_2(\delta)$ от длины последовательности при $\delta = 0.3$

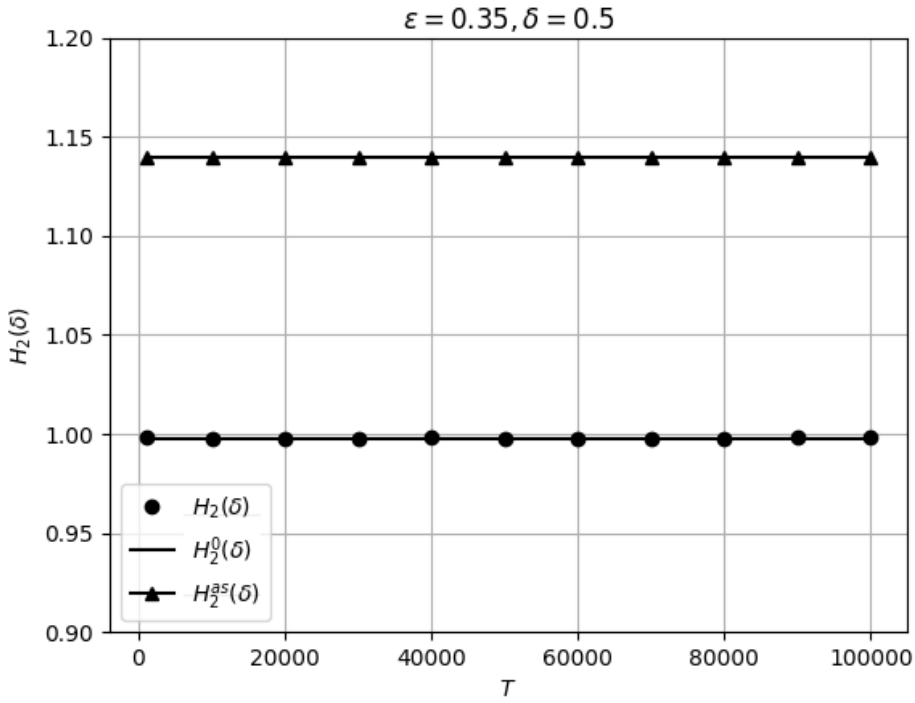

Рисунок 3.3 — График зависимости энтропии $H_2(\delta)$ от длины последовательности при $\delta = 0.5$

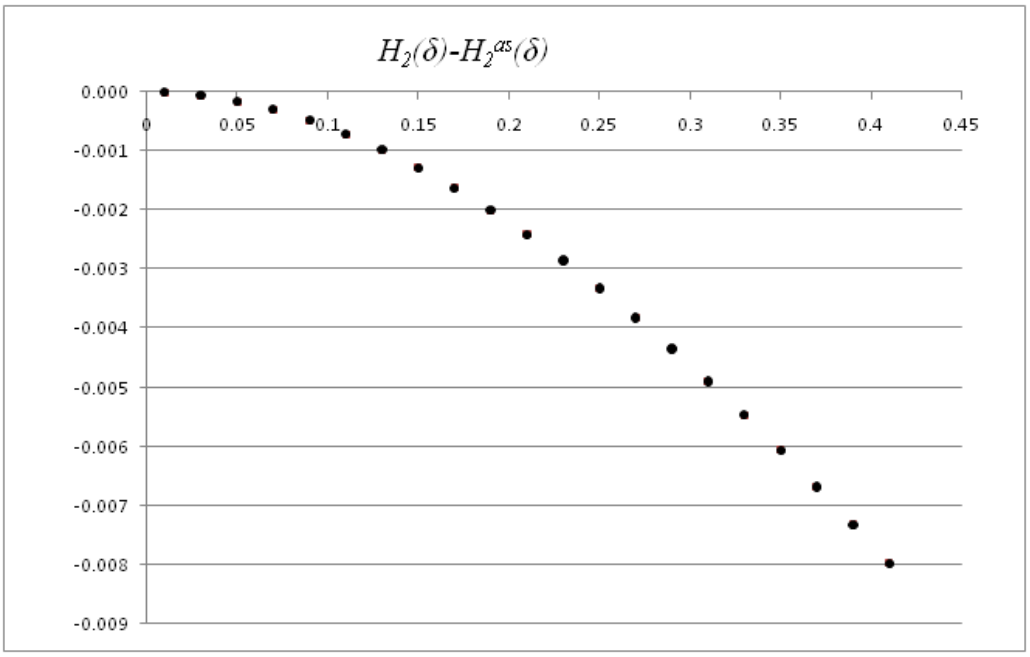

Рисунок 3.4 — График зависимости разности ассимптотического и точного значений энтропии биграммы от доли вкрапления

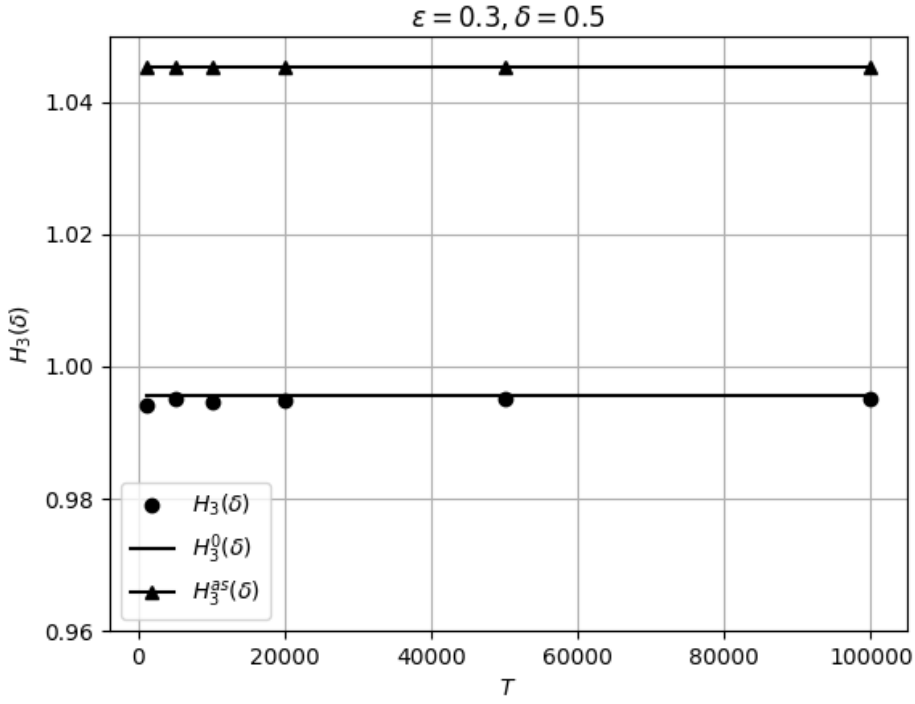

Рисунок 3.5 — График зависимости энтропии $H_3(\delta)$ от длины последовательности при $\delta = 0.5$

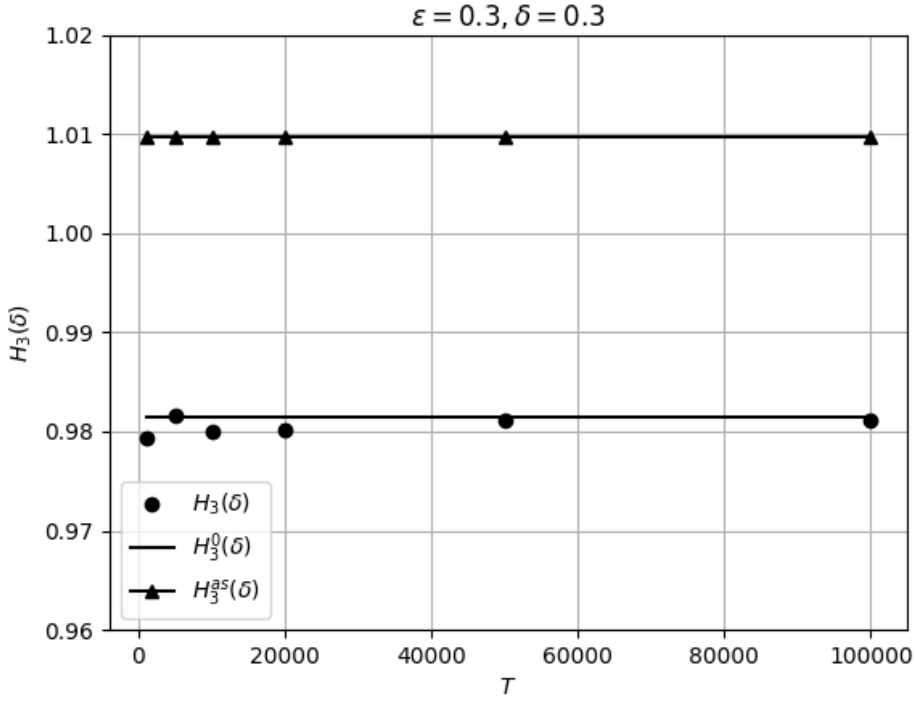

Рисунок 3.6 — График зависимости энтропии $H_3(\delta)$ от длины последовательности при $\delta = 0.3$

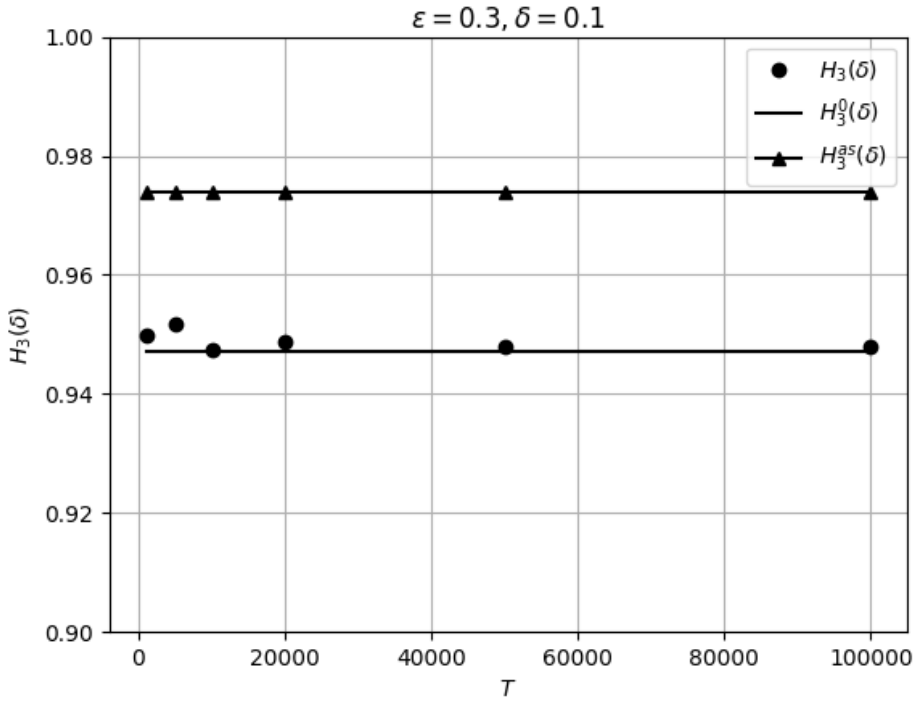

Рисунок 3.7 — График зависимости энтропии $H_3(\delta)$ от длины последовательности при $\delta = 0.1$

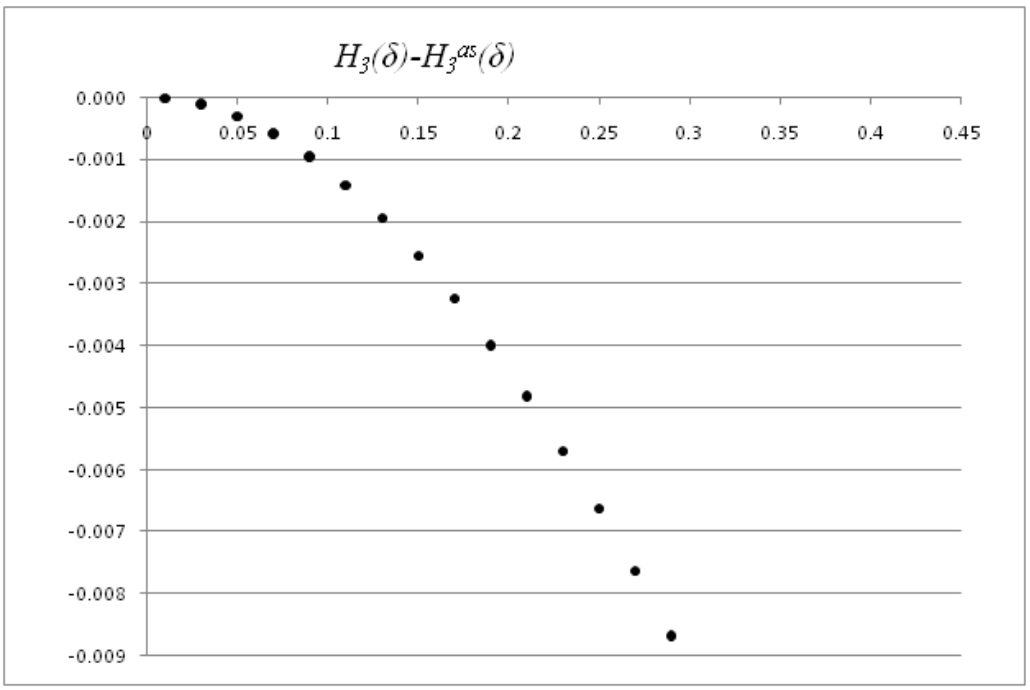

Рисунок 3.8 — График зависимости разности ассимптотического и точного значений энтропии 3-граммы от доли вкрапления

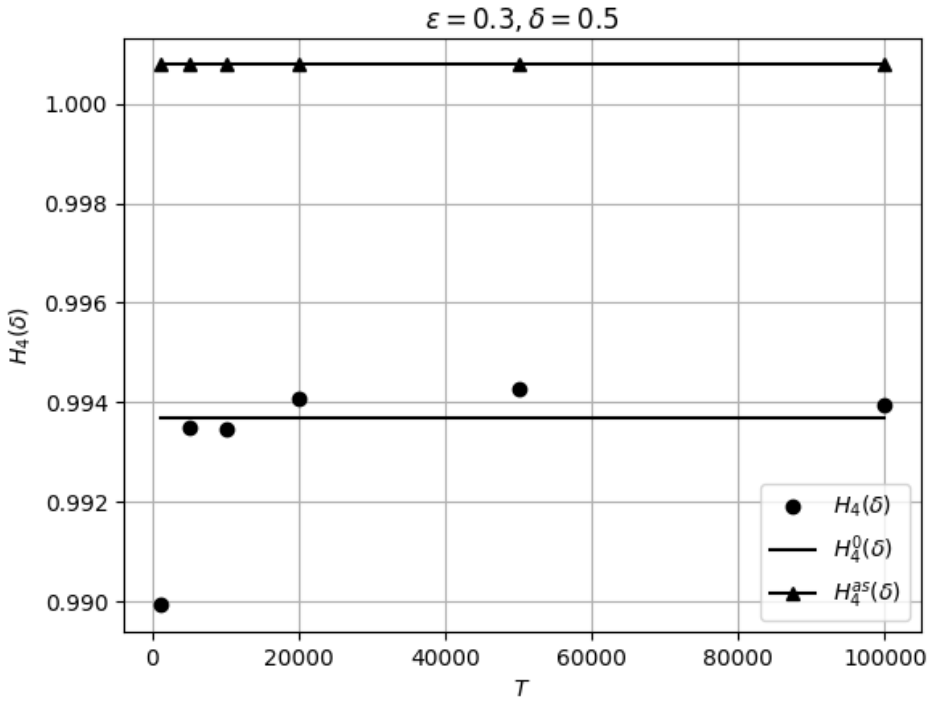

Рисунок 3.9 — График зависимости энтропии $H_4(\delta)$ от длины последовательности при $\delta = 0.5$

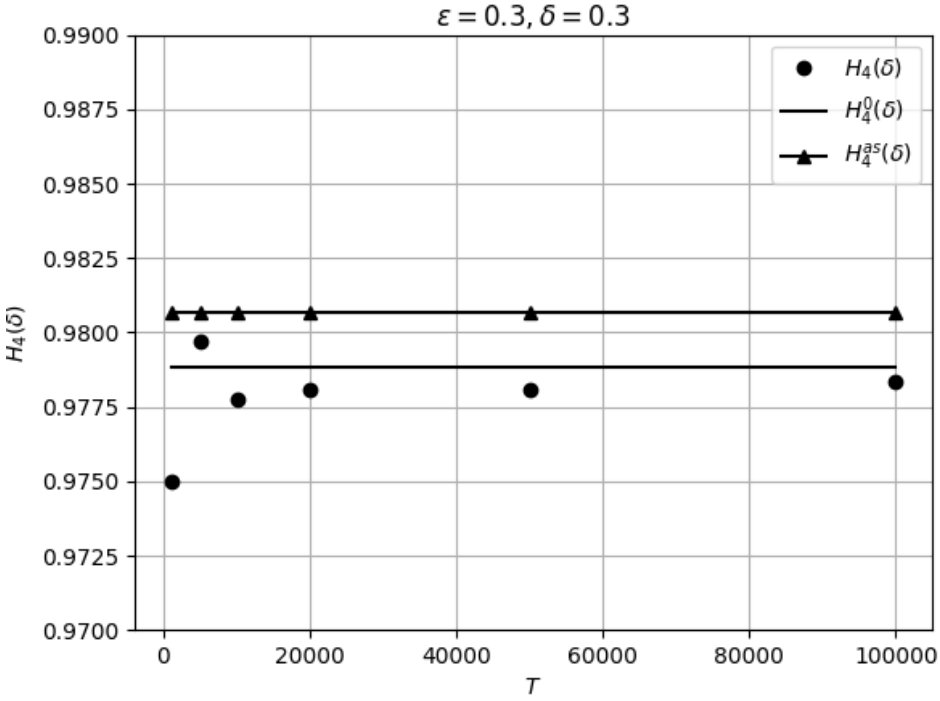

Рисунок 3.10 — График зависимости энтропии $H_4(\delta)$ от длины последовательности при $\delta = 0.3$

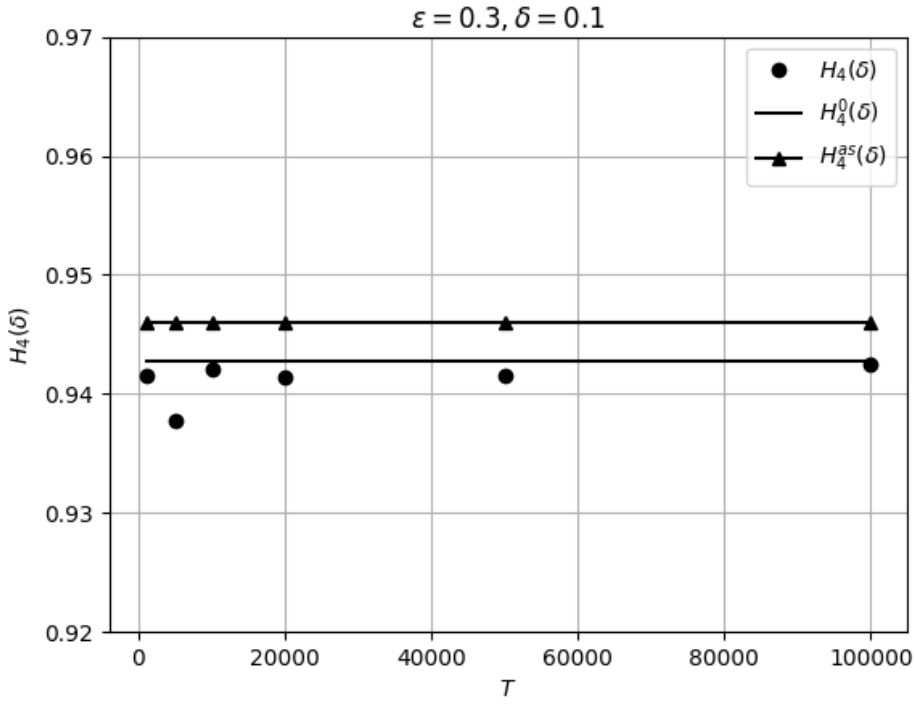

Рисунок 3.11 — График зависимости энтропии $H_4(\delta)$ от длины последовательности при $\delta = 0.1$

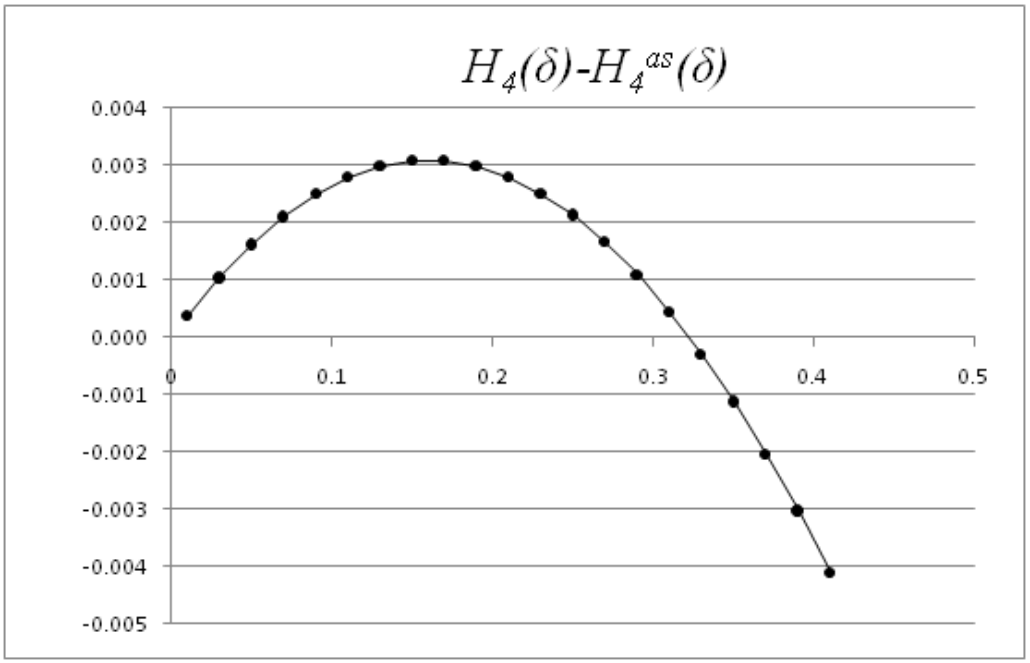

Рисунок 3.12 — График зависимости разности ассимптотического и точного значений энтропии биграммы от доли вкрапления

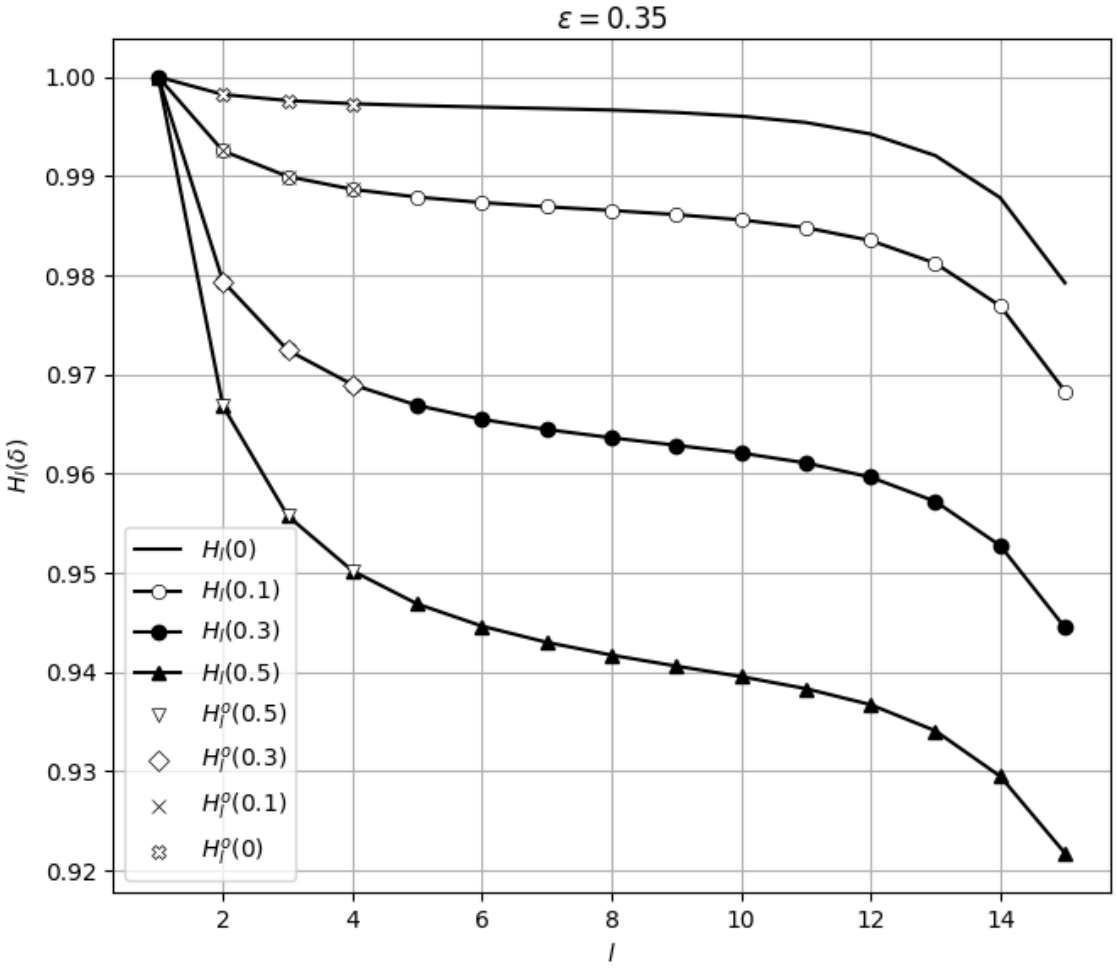

Рисунок 3.13 — Семейство графиков зависимости энтропии $H_l(\delta)$ от L при различных δ

ЗАКЛЮЧЕНИЕ

В работе получены следующие основные результаты:

1. Проведен аналитический обзор физических устройств и процессов, используемых для генерации случайных чисел.
2. Рассмотрены подходы к оценке качества выходных последовательностей источников случайности физических генераторов.
3. Разработаны алгоритмы и программное обеспечение для тестирования и оценки качества источников случайности.
4. С помощью разработанного ПО протестированы выходные последовательности реального источника.
5. Разработанное ПО может применяться для тестирования источников случайности на соответствие требованиям различных стандартов, в том числе СТБ 34.101.27-2011 «Требования безопасности к программным средствам криптографической защиты информации».

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. R.O. Duda, P.E. Hart, D.H. Stork Pattern Classification (2nd ed.) // Wiley Interscience. — 2000.
2. А. А. Духин: Теория информации - М.: "Гелиос АРВ 2007.
3. А.В. Аграновский, А. В. Балакин: Стеганография, цифровые водяные знаки о стегоанализ - М.: Вузовская книга, 2009.
4. В. Г. Грибунин, И. Н. Оков, И. В. Туринцев: Цифровая стеганография - М.: Солон-Прессб, 2002.
5. Дж. Миано: Форматы и алгоритмы сжатия изображений в действии - М.: Триумф, 2003
6. Н. П. Варновский, Е. А. Голубев, О. А. Логачев: Современные направления стеганографии. Математика и безопасность информационных технологий. Материалы конференции в МГУ 28-29 октября 2004 г., МЦМНО, М., 2005, с. 32-64.
7. Ю. С. Харин [и др.]: Криптология - Минск: БГУ, 2013.
8. Ю. С. Харин, Е. В. Вечерко "Статистическое оценивание параметров модели вкраплений в двоичную цепь Маркова Дискрет. матем., 25:2 (2013), 135-148.
9. Ю. С. Харин, Е. В. Вечерко "Распознавание вкраплений в двоичную цепь Маркова Дискрет. матем., 27:3 (2015), 123–144.

ПРИЛОЖЕНИЕ А

Исходный код основных частей разработанного приложения

A.1 Тесты перестановок

A.1.1 Основная логика теста

```
public struct ShufflingTestReport
{
    public string Name; //statistical score name
    public int[] P; //ranks
    public TestResult Result;
}

public static class ShufflingTest
{
    public const int subsetNumber = 10;
    public const int shuffleNumber = 1000;

    private struct RankRecord
    {
        public string Name;
        public int Subset;
        public int Rank;
    }

    private static Object shufLock = new Object();

    public static TestResult execute(int[] dataset, bool binary =
       false) {
        List<ShufflingTestReport> report;
        return execute(dataset, out report, binary);
    }

    public static TestResult execute(int[] dataset, out
       List<ShufflingTestReport> report, bool binary = false) {
        Random randGen = new Random();
        var ranks = new List<RankRecord>();
        int subsetLength = dataset.Length / subsetNumber;
        Console.Write("Shuffling test: Start... ");
        double percentage = .0;
        double stepPercent = 100.0 / subsetNumber;
        for(int i = 0; i < subsetNumber; ++i) {
            int[] subset = new int[subsetLength];
            //bytes are copying
            Buffer.BlockCopy(dataset, i * subsetLength << 2,
               subset, 0, subsetLength << 2);
            //var subset = dataset.Skip(subsetLength *
               i).Take(subsetLength).ToArray();
```

```
        var initScores = StatisticalScores.getScores(subset,
            binary);
        //Shuffling
        var shufScores = new List<Score>();
        Action[] shufTasks = new Action[shuffleNumber];
        for(int j = 0; j < shuffleNumber; ++j) {
            shufTasks[j] = new Action(() => {
                int[] subsetCopy = new int[subset.Length];
                lock(shufLock) {
                    Utilities.shuffle(subset, randGen);
                    Buffer.BlockCopy(subset, 0, subsetCopy, 0,
                        subset.Length << 2);
                }
                var scores =
                    StatisticalScores.getScores(subsetCopy,
                    binary);
                lock(shufScores) {
                    shufScores.AddRange(scores);
                }
            });
        }
        var parallelOptions = new ParallelOptions() {
            MaxDegreeOfParallelism = Utilities.CoreCount
        };
        Parallel.Invoke(parallelOptions, shufTasks);
        percentage += stepPercent;
        Console.Write("{0:0.}% ... ", percentage);
        //Shuffled subsets scores lists
        foreach(var initScore in initScores) {
            var scoreList = shufScores.Where(x => x.Name ==
                initScore.Name).ToList();
            int lowerRank = scoreList.Where(x => x.Value <
                initScore.Value).Count();
            int upperRank = scoreList.Where(x => x.Value <=
                initScore.Value).Count();
            int r = shuffleNumber >> 1;
            int rank = upperRank < r ? upperRank : (lowerRank
                > r ? lowerRank : r);
            ranks.Add(new RankRecord {
                Name = initScore.Name,
                Subset = i,
                Rank = rank
            });
        }
    }
    report = new List<ShufflingTestReport>();
    TestResult result = TestResult.Passed;
    foreach(var records in ranks.GroupBy(x => x.Name)) {
        int[] curRanks = (from r in records
                          orderby r.Subset
                          select r.Rank).ToArray();
        TestResult curResult = curRanks.Where(x => x < 50 || x
            > 950).Count() >= 8 ? TestResult.Failed :
```

```
                TestResult.Passed;
            if(curResult == TestResult.Failed) {
                result = TestResult.Failed;
            }
            report.Add(new ShufflingTestReport {
                Name = records.First().Name,
                P = curRanks,
                Result = curResult
            });
        }
        Console.WriteLine("\nShuffling test: Finished");
        return result;
    }
}
```

A.1.2 Вычисление статистических оценок

```
public struct Score
{
    public string Name;
    public decimal Value;
}

/// <summary>
/// introduces statistical scores used in IID tests
/// </summary>
public static class StatisticalScores
{
    public static List<Score> getScores(int[] sample, bool binary
       = false) {
        var scores = new List<Score>();
        scores.AddRange(compressionScores(sample, binary));
        scores.AddRange(runsScores(sample, binary));
        scores.AddRange(excursionScores(sample, binary));
        scores.AddRange(directionalRunsScores(sample, binary));
        scores.AddRange(covarianceScores(sample, binary));
        scores.AddRange(collisionScores(sample, binary));
        return scores;
    }
```

A.1.2.1 Оценка сжатия

```
    /// <summary>
    /// calculates compression score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>length of the compressed string in bytes</returns>
    public static Score[] compressionScores(int[] sample, bool
       binary = false) {
        byte[] input = Encoding.UTF8.GetBytes(string.Join(",",
           sample));
        //Console.WriteLine(str);
        byte[] output = BZip2.Compress(input, 4096);
        return new Score[]{
```

```
            new Score{
                Name = "compression score",
                Value = (output != null) ? output.Length : -1
            }
        };
    }
```

A.1.2.2 Оценки серий

```
    /// <summary>
    /// calculates Over/Under Runs Score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>longest run and total number of runs</returns>
    public static Score[] runsScores(int[] sample, bool binary =
       false) {
        int[] median = null;
        if(binary) {
            median = new int[] { 0, 1 };
        } else {
            median = StandardFunctions.integerMedian(sample);
        }
        //false: value is under the median
        //true: value is over the median
        //skipped: value is equal to the median
        List<bool> symbols = new List<bool>();
        for(int i = 0; i < sample.Length; ++i) {
            if(sample[i] < median[1]) {
                symbols.Add(false);
            } else if(sample[i] > median[0]) {
                symbols.Add(true);
            }
        }
        //first symbol produces first run of length 1
        int currentRunLength = 1;
        bool currentSymbol = symbols[0];
        int longestRunLength = 1;
        int runsNumber = 1;
        ////////////////////////////////////////
        for(int i = 1; i < symbols.Count; ++i) {
            if(symbols[i] != currentSymbol) {
                //the next run
                if(currentRunLength > longestRunLength) {
                    longestRunLength = currentRunLength;
                }
                currentRunLength = 1;
                ++runsNumber;
            } else {
                //the same run
                ++currentRunLength;
            }
        }
        if(currentRunLength > longestRunLength) {
            longestRunLength = currentRunLength;
```

```
        }
        return new Score[]{
            new Score{
                Name = "Longest run score",
                Value = longestRunLength
            },
            new Score{
                Name = "Runs number score",
                Value = runsNumber
            }
        };
    }
```

A.1.2.3 Оценка кумулятивного отклонения

```
    /// <summary>
    /// calculates excursion score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>maximal deviation from expected value</returns>
    public static Score[] excursionScores(int[] sample, bool
       binary = false) {
        if(sample.Length == 0) {
            return new Score[]{
            new Score{
                Name = "Excursion score",
                Value = 0
            }
        };
        }
        int initRemainder;
        int initAverage = StandardFunctions.integerAverage(sample,
           out initRemainder);
        //decimal average = initAverage + initRemainder * 100 /
           sample.Length;
        decimal average = initAverage + (decimal)initRemainder /
           sample.Length;
        decimal excursion = 0;
        decimal maxAbsValue = 0;
        for(int i = 0; i < sample.Length; ++i) {
            excursion += sample[i] - average;
            if(Math.Abs(excursion) > maxAbsValue) {
                maxAbsValue = Math.Abs(excursion);
            }
        }
        return new Score[]{
            new Score{
                Name = "Excursion score",
                Value = maxAbsValue
            }
        };
    }
```

A.1.2.4 Оценки направленных серий

```
/// <summary>
/// calculates directional runs score
/// </summary>
/// <param name="sample">sample to analyse</param>
/// <returns>longest run, total number of runs, total number
   of 1 or -1 (which is greater)</returns>
public static Score[] directionalRunsScores(int[] sample, bool
   binary = false) {
    if(binary){
        int count = sample.Length >> 3;
        int[] hammingWeights = new int[count];
        for(int j = 0; j < count; ++j) {
            for(int i = 0; i < 8; ++i) {
                hammingWeights[j] += sample[(j << 3) + i];
            }
        }
        return directionalRunsScores(hammingWeights, false);
    }
    //creation of derivatives array
    sbyte[] derivatives = new sbyte[sample.Length - 1];
    int numberOfOnes = 0;
    int numberOfMinusOnes = 0;
    //было for(int i = 0; i < derivatives.Length - 1; ++i) {
    for (int i = 0; i < derivatives.Length; ++i) {
        if(sample[i] < sample[i + 1]) {
            derivatives[i] = 1;
            ++numberOfOnes;
        } else if(sample[i] > sample[i + 1]) {
            derivatives[i] = -1;
            ++numberOfMinusOnes;
        } else {
            derivatives[i] = 0;
        }
    }
    //skipping leading 0
    int currentPosition = 0;
    while(derivatives[currentPosition] == 0 && currentPosition
       < derivatives.Length - 1) {
        ++currentPosition;
    }
    if(derivatives[currentPosition] == 0) {
        return new Score[]{
            new Score{
                Name = "Longest direct. run score",
                Value = 0
            },
            new Score{
                Name = "Direct. runs number score",
                Value = 0
            },
            new Score{
```

```
                    Name = "Direct. runs -1/1 score",
                    Value = 0
                }
            };
        }
        ////////////////////////////////////////////////
        int currentValue = derivatives[currentPosition];
        ++currentPosition;
        int runsNumber = 1;
        int currentRunLength = 1;
        int longestRunLength = 1;
        for(; currentPosition < derivatives.Length;
           ++currentPosition) {
            if(derivatives[currentPosition] == -currentValue) {
                //the next run (zeros do not break the run)
                ++runsNumber;
                if(currentRunLength > longestRunLength) {
                    longestRunLength = currentRunLength;
                }
                currentRunLength = 1;
            } else {
                //the same run
                ++currentRunLength;
            }
        }
        if(currentRunLength > longestRunLength) {
            longestRunLength = currentRunLength;
        }
        return new Score[]{
            new Score{
                Name = "Longest direct. run score",
                Value = longestRunLength
            },
            new Score{
                Name = "Direct. runs number score",
                Value = runsNumber
            },
            new Score{
                Name = "Direct. runs -1/1 score",
                Value = Math.Max(numberOfOnes, numberOfMinusOnes)
            }
        };
    }
```

A.1.2.5 Оценка ковариации

```
    /// <summary>
    /// calculates covariance score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>covariance estimate</returns>
    public static Score[] covarianceScores(int[] sample, bool
       binary = false) {
        int average = StandardFunctions.integerAverage(sample);
```

```
        int[] counts = new int[sample.Length - 1];
        for(int i = 0; i < counts.Length; ++i) {
            counts[i] = (sample[i] - average) * (sample[i + 1] -
               average);
        }
        return new Score[]{
            new Score{
                Name = "Covariance score",
                Value = StandardFunctions.integerAverage(counts)
            }
        };
    }
```

A.1.2.6 Оценки коллизий

```
    /// <summary>
    /// calculates collision score
    /// </summary>
    /// <param name="sample">sample to analyse</param>
    /// <returns>number of values to collision: minimal, maximal
       and average</returns>
    public static Score[] collisionScores(int[] sample, bool
       binary = false) {
        var collisions = StandardFunctions.allCollisions(sample);
        if(collisions.Count == 0) {
            return new Score[]{
                new Score{
                    Name = "Collision min score",
                    Value = -1
                },
                new Score{
                    Name = "Collision max score",
                    Value = -1
                },
                new Score{
                    Name = "Collision average score",
                    Value = -1
                }
            };
        } else {
            return new Score[]{
                new Score{
                    Name = "Collision min score",
                    Value = collisions.Min()
                },
                new Score{
                    Name = "Collision max score",
                    Value = collisions.Max()
                },
                new Score{
                    Name = "Collision average score",
                    Value =
                       StandardFunctions.integerAverage(collisions)
                }
```

```
            };
        }
    }
}
```

Поиск коллизий:

```
    public static int nextCollision<T>(T[] dataset, int
       startPosition) {
        var values = new HashSet<T>();
        int currentPosition = startPosition;
        while(currentPosition < dataset.Length) {
            if(values.Contains(dataset[currentPosition])){
                //the next collision is found
                return currentPosition;
            } else {
                values.Add(dataset[currentPosition]);
            }
            ++currentPosition;
        }
        //no collision is found
        return -1;
    }

    public static List<int> allCollisions<T>(T[] dataset) {
        var collisions = new List<int>();
        int currentPosition = 0;
        int nextPosition;
        while((nextPosition = nextCollision(dataset,
           currentPosition)) != -1) {
            collisions.Add(nextPosition - currentPosition + 1);
            currentPosition = nextPosition + 1;
        }
        return collisions;
    }
```

A.2 Хи-квадрат тесты

A.2.1 Хи-квадрат тест для проверки независимости для небинарных данных

```
public static ChiSquaredTestReport independenceTest(int[] dataset)
   {
        var report = new ChiSquaredTestReport{Name = "Chi-squared
           independence test"};
        var probs = dataset.GroupBy(x => x).ToDictionary(x =>
           x.Key, x => (double)x.Count() / dataset.Length);
        double maxProb = probs.Max(x => x.Value);
        double freqThreshold = 5.0 / dataset.Length;
        var freqProbs = from x in probs
                        where x.Value * maxProb >= freqThreshold
                        select x;
        int paramNumber = 1 + freqProbs.Count();
        if(paramNumber == 1) {
```

```
    report.Result = TestResult.Skipped;
    report.Comment = "Not enough data";
    return report;
}
var allPairs = from x in freqProbs
               from y in freqProbs
               select new KeyValuePair<Tuple<int, int>,
                  double>(Tuple.Create(x.Key, y.Key),
                  x.Value * y.Value * (dataset.Length -
                  1));
var expFreqPairs = (from pair in allPairs
                    where pair.Value >= 5
                    select pair)
                    .ToDictionary(pair => pair.Key, pair
                       => pair.Value);
int existRarePair = (expFreqPairs.LongCount() <
   (long)probs.Count * (long)probs.Count) ? 1 : 0;
double expRarePairsFreq = dataset.Length - 1 -
   expFreqPairs.Aggregate(0.0, (freq, pair) => freq +
   pair.Value);
int degreesOfFreedom = expFreqPairs.Count() + 1 -
   paramNumber;
if(degreesOfFreedom < 1) {
    report.Result = TestResult.Skipped;
    report.Comment = "Not enough data";
    return report;
}
var neighbors = dataset.Skip(1).Zip(dataset, (next, prev)
   => Tuple.Create(prev, next));
var observedPairs = neighbors.GroupBy(pair =>
   pair).ToDictionary(x => x.Key, x => x.Count());
var obsFreqPairs = (from pair in observedPairs
                    where expFreqPairs.ContainsKey(pair.Key)
                    select pair)
                    .ToDictionary(pair => pair.Key, pair =>
                       pair.Value);
if(existRarePair > 0) {
    int obsRarePairsFreq = (from pair in observedPairs
    where !expFreqPairs.ContainsKey(pair.Key)
                            select pair)
                            .Aggregate(0, (freq, pair) =>
                               freq + pair.Value);
    report.Stat = (expRarePairsFreq - obsRarePairsFreq) *
       (expRarePairsFreq - obsRarePairsFreq);
    report.Stat /= expRarePairsFreq;
}
foreach(var pair in expFreqPairs) {
    double obsFreq = obsFreqPairs.ContainsKey(pair.Key) ?
       obsFreqPairs[pair.Key] : 0.0;
    obsFreq -= pair.Value;
    report.Stat += obsFreq * obsFreq / pair.Value;
}
report.Df = expFreqPairs.Count - 1 + existRarePair;
```

```
            report.P = SpecMath.chisqc(report.Df, report.Stat);
            report.Result = report.P > confidenceLevel ?
               TestResult.Passed : TestResult.Failed;
            return report;
        }
```

A.2.2 Хи-квадрат тест для проверки одинаковой распределенности для небинарных данных

```
    public static List<ChiSquaredTestReport> goodnessOfFitTest(int[]
       dataset) {
            var reportList = new
               List<ChiSquaredTestReport>(subsetNumber);
            var report = new ChiSquaredTestReport{
                Name = "Chi-squared goodness-of-fit test"
            };
            int subsetLength = dataset.Length / subsetNumber;
            double multiplier = (double) subsetLength / dataset.Length;
            var probs = dataset.GroupBy(item =>
               item).ToDictionary(item => item.Key, item => multiplier
               * item.Count());
            double probThreshold = 5.0;
            var freqValues = (from item in probs
                              where item.Value >= probThreshold
                              select item).
                              ToDictionary(item => item.Key, item =>
                                 item.Value);
            if(freqValues.Count == 0) {
                report.Result = TestResult.Skipped;
                report.Comment = "Not enough data";
                reportList.Add(report);
                return reportList;
            }
            int existRareValue = (freqValues.Count < probs.Count) ? 1
               : 0;
            double rareValuesProb = dataset.Length -
               freqValues.Aggregate(0.0, (prob, item) => prob +
               item.Value);
            report.Df = freqValues.Count - 1 + existRareValue;
            for(int i = 0; i < subsetNumber; ++i) {
                report.Comment = string.Format("Subset {0}", i);
                var subset = dataset.Skip(i *
                   subsetLength).Take(subsetLength).GroupBy(item =>
                   item).ToDictionary(item => item.Key, item =>
                   item.Count());
                var subsetFreqValues = (from item in subset
                where freqValues.ContainsKey(item.Key)
                                        select item)
                                        .ToDictionary(item =>
                                           item.Key, item =>
                                           item.Value);
                report.Stat = .0;
                if(existRareValue > 0) {
```

```
                int subsetRareValuesProb = (from item in subset
                where !freqValues.ContainsKey(item.Key)
                                        select item)
                                        .Aggregate(0, (prob,
                                           item) => prob +
                                           item.Value);
                report.Stat = (rareValuesProb -
                   subsetRareValuesProb) * (rareValuesProb -
                   subsetRareValuesProb);
                report.Stat *= report.Stat / rareValuesProb;
            }
            foreach(var item in freqValues) {
                double obsProb =
                   subsetFreqValues.ContainsKey(item.Key) ?
                   subsetFreqValues[item.Key] : 0;
                obsProb -= item.Value;
                report.Stat += obsProb * obsProb / item.Value;
            }
            report.P = SpecMath.chisqc(report.Df, report.Stat);
            report.Result = report.P > confidenceLevel ?
               TestResult.Passed : TestResult.Failed;
            reportList.Add(report);
            if(report.Result == TestResult.Failed) {
                return reportList;
            } else {
                report = new ChiSquaredTestReport{
                    Name = report.Name,
                    Df = report.Df
                };
            }
        }
        return reportList;
    }
```

A.2.3 Хи-квадрат тесты для проверки независимости и одинаковой распределенности для бинарных данных

```
public static List<ChiSquaredTestReport> binaryTest(int[] dataset)
   {
        var reportList = new List<ChiSquaredTestReport>(10 +
           subsetNumber);
        int onesNumber = dataset.Count(value => value == 1);
        int zerosNumber = dataset.Length - onesNumber;
        double onesProb = (double) onesNumber / dataset.Length;
        double zerosProb = 1.0 - onesProb;
        double minProb = Math.Min(onesProb, zerosProb);
        double freq = minProb * minProb * dataset.Length;
        int subsetLength;
        for(int k = 2; (k <= 11) && (freq > 5); ++k, freq *=
           minProb) {
            var report = new ChiSquaredTestReport{
                Name = "Chi-squared independence test for binary
                   data",
```

```
                Comment = string.Format("{0}-bit chunks", k)
            };
            subsetLength = dataset.Length / k;
            var subset = new ushort[subsetLength];
            for(int i = 0; i < subsetLength; ++i) {
                for(int j = 0; j < k; ++j) {
                    subset[i] += (ushort)(dataset[i * k + j] << (k
                        - j - 1));
                }
            }
            var subsetFreq = subset.GroupBy(value =>
                value).ToDictionary(item => (ushort)item.Key, item
                => item.Count());
            report.Stat = .0;
            for(ushort i = 0; i < (1 << k); ++i) {
                byte hw = Utilities.hammingWeight(i);
                double expFreq = Math.Pow(onesProb, hw) *
                    Math.Pow(zerosProb, k - hw) * subsetLength;
                double obsFreq = subsetFreq.ContainsKey(i) ?
                    subsetFreq[i] : 0;
                report.Stat += (expFreq - obsFreq) * (expFreq -
                    obsFreq) / expFreq;
            }
            report.Df = (1 << k) - 1;
            report.P = SpecMath.chisqc(report.Df, report.Stat);
            report.Result = report.P > confidenceLevel ?
                TestResult.Passed : TestResult.Failed;
            reportList.Add(report);
            if(report.Result == TestResult.Failed) {
                break;
            }
        }
        var report2 = new ChiSquaredTestReport{
            Name = "Chi-squared goodness-of-fit test for binary
                data"
        };
        subsetLength = dataset.Length / subsetNumber;
        double expOnesNum = onesProb * subsetLength;
        for(int i = 0; i < subsetNumber; ++i) {
            int obsOnesNum = dataset.Skip(i *
                subsetLength).Take(subsetLength).Count(value =>
                value == 1);
            report2.Stat += (expOnesNum - obsOnesNum) *
                (expOnesNum - obsOnesNum) / expOnesNum;
        }
        report2.Df = subsetNumber - 1;
        report2.P = SpecMath.chisqc(report2.Df, report2.Stat);
        report2.Result = report2.P > confidenceLevel ?
            TestResult.Passed : TestResult.Failed;
        reportList.Add(report2);
        return reportList;
    }
}
```

A.3 Оценки минимальной энтропии

A.3.1 Оценка минимальной энтропии последовательности случайных величин, являющихся н.о.р.

```
/// <summary>
/// estimates minimal entropy for IID sources
/// </summary>
/// <param name="dataset"></param>
/// <param name="sampleSize">number of bits in each
   sample</param>
/// <returns>entropy estimate</returns>
public static double iidTest(int[] dataset, byte sampleSize)
{
    double maxProb = dataset.GroupBy(x => x).Max(x =>
       x.Count()) / (double)dataset.Length;
    double pmax = maxProb + 2.3 * Math.Sqrt(maxProb * (1 -
       maxProb) / dataset.Length);
    double entropy = -Math.Log(pmax);
    return Math.Min(entropy, sampleSize);
}
```

A.3.2 Тест коллизий

```
        private static double solveBinarySearch(Func<double,
           double, double, double> f, double n, double m) {
                double eps = 0.00001;
                double high = 1 - eps;
                double low = eps;
                double highF = f (high, n, m);
                double lowF = f (low, n, m);
                if (highF * lowF > 0)
                        throw new Exception("Method not
                           applicable");
                double diff;
                double diffF;
                do {
                        diff = high - low;
                        diffF = Math.Max(Math.Abs(high),
                           Math.Abs(low));
                        double middle = (high + low) / 2;
                        double middleF = f(middle,n,m);
                        if(middleF * highF > 0) {
                                high = middle;
                                highF = middleF;
                        } else {
                                low = middle;
                                lowF = middleF;
                        }
                } while ((diff>eps) || (diffF<eps));
                if (Math.Abs (high) < Math.Abs (low))
                        return high;
                return low;
```

```
            }

            private static double expectFunction(double p, double
               n, double m) {
                  double q = (1 - p) / (n - 1);
                  double f = SpecMath.igamma(n+1,1/q) * Math.Pow
                     (q, n + 1) * Math.Exp (-1/q);
                  double x = (1 / p - 1 / q) / n;
     double z = p / q * (1 / p - 1 / q) / n;
                  double y = (1 + x) * f / q;
                  return (y - x) * p / q - z - m;
            }

/// <summary>
/// estimates minimal entropy for non-IID sources using
   collision test
/// </summary>
/// <param name="dataset"></param>
/// <param name="sampleSize">number of bits in each
   sample</param>
/// <returns>entropy estimate</returns>
public static double collisionTest(int[] dataset, byte
   sampleSize) {
     //throw new Exception("not implemented");
     //you need to observe at least 1000 collisions
     //see birthday problem to calculate necessary length of
        the dataset
     //depending on the number of bits in a sample [e.g.
        Ramanujan asymptotic]
     var collisions = StandardFunctions.allCollisions(dataset);
     if(collisions.Count < 1000) {
         throw new NotEnoughDataException("Collision test:
            there are not enough collisions in the dataset");
     }
     double lowerBound =
        StandardFunctions.confidenceInterval(collisions)[0];
                  double p = solveBinarySearch (expectFunction,
                     sampleSize, lowerBound);
                  return -Math.Log (p, 2);
}

            private static double epFunction(double p, double n,
               double m) {
                  double q = (1 - p) / (n - 1);
                  double f = 1 - Math.Pow (1 - p, n);
                  double g = (n-1)*(1 - Math.Pow (1 - q, n));
                  return f+ g - m;
            }
```

A.3.3 Тест частичных коллекций

```
            private static double epFunction(double p, double n,
               double m) {
                  double q = (1 - p) / (n - 1);
```

```
                    double f = 1 - Math.Pow (1 - p, n);
                    double g = (n-1)*(1 - Math.Pow (1 - q, n));
                    return f+ g - m;
                }

        /// <summary>
        /// estimates minimal entropy for non-IID sources using
           partial collection test
        /// </summary>
        /// <param name="dataset"></param>
        /// <param name="sampleSize">number of bits in each
           sample</param>
        /// <returns>entropy estimate</returns>
        public static double partialCollectionTest(int[] dataset, byte
           sampleSize) {
            //the output space or indication of values that never
               appear in the output SHALL be
            //provided by the developer for validation testing

            //you need to observe minimum 500 events, so dataset
               length need to be at least 500*2^b,
            //where b is the number of bits in a sample
            int outputSpaceSize = 1 << sampleSize;
            int[] distinctValuesNumbers = new int[dataset.Length /
               outputSpaceSize];
            var valuesSet = new HashSet<int>();
            for(int subset = 0; subset < distinctValuesNumbers.Length;
               ++subset) {
                for(int position = 0; position < outputSpaceSize;
                   ++position) {
                    valuesSet.Add(dataset[subset * outputSpaceSize +
                       position]);
                }
                distinctValuesNumbers[subset] = valuesSet.Count;
                valuesSet.Clear();
            }
                    int m = distinctValuesNumbers.Max();
                    if (outputSpaceSize - m > 500) {
                            throw new
                               NotEnoughDataException("Partial
                               collection test: not enough data");
                    }
            double lowerBound =
StandardFunctions.confidenceInterval(distinctValuesNumbers)[0];
                    double p = solveBinarySearch (epFunction,
                       sampleSize, lowerBound);
                    return -Math.Log (p, 2);
        }
```

A.3.4 Тест марковской зависимости

```
        public static double markovTest(int[] dataset, byte
           sampleSize) {
```

```
                //the largest sample size accommodated by this
                    test is 6 bits
                int N = 1 << sampleSize;
                int power = Math.Max (N * N, 128);
        double a = - power * Math.Log(confidenceLevel, 2);
                double e = Math.Sqrt (0.5 * a /
                    dataset.Length);
                if (sampleSize > 6)
                        throw new
                            NotEnoughDataException("Markov
                            test: too big output space for the
                            test");
                SparseArray<int, double> prior = new
                    SparseArray<int, double>(0.0);
                SparseArray<int, double> priorE = new
                    SparseArray<int, double>(0.0);
                SparseArray<int, double> minP = new
                    SparseArray<int, double>(100000.0);
                SparseArray<int, int> minWay = new
                    SparseArray<int, int>(-1);
                Sparse2DMatrix<int, int, double> trans = new
                    Sparse2DMatrix<int, int, double>(1000000);
                Sparse2DMatrix<int, int, double> vertex = new
                    Sparse2DMatrix<int, int, double>(-1.0);
                prior [dataset [0]]++;
                for (int i = 1; i < dataset.Length; i++) {
                        prior [dataset [i]]++;
                        trans [dataset [i - 1], dataset [i]]++;
                }
                foreach(KeyValuePair<int, double> pair in
                    prior)
                {
            priorE[pair.Key] = Math.Sqrt (0.5 * a / pair.Value);
                        //Console.WriteLine("pair vector[{0}]
                            = {1}", pair.Key, pair.Value);
                }
                foreach (KeyValuePair<ComparableTuple2<int,
                    int>, double> pair in trans)
                {
                        int key0 = 0;
                        int key1 = 0;
                        trans.SeparateCombinedKeys(pair.Key,
                            ref key0, ref key1);
                        trans [key0,key1] = -
                            Math.Log(Math.Min(1, pair.Value /
                            prior[key0] + priorE[key0]),2);
                        //Console.WriteLine("pair matrix[{0},
                            {1}] = {2}", key0, key1,
                            pair.Value);
                }
                foreach(KeyValuePair<int, double> pair in
                    prior)
                {
```

```
                prior[pair.Key] = -
                    Math.Log(Math.Min(1, pair.Value /
                    dataset.Length + e),2);
                //Console.WriteLine("pair vector[{0}]
                    = {1}", pair.Key, pair.Value);
            }
            foreach (KeyValuePair<ComparableTuple2<int,
                int>, double> pair in trans)
            {
                int key0 = 0;
                int key1 = 0;
                trans.SeparateCombinedKeys(pair.Key,
                    ref key0, ref key1);
                if (pair.Value < minP [key0]) {
                        minP [key0] = pair.Value;
                        minWay [key0] = key1;
                }
                //Console.WriteLine("pair matrix[{0},
                    {1}] = {2}", key0, key1,
                    pair.Value);
            }
            foreach(KeyValuePair<int, double> pair in
                prior)
            {
                vertex[0,pair.Key] = pair.Value;
                //Console.WriteLine("pair vector[{0}]
                    = {1}", pair.Key, pair.Value);
            }
            for (int k = 1; k < 128; k++) {
                foreach(KeyValuePair<int, double> end
                    in prior) {
                        vertex [k, end.Key] =
                            double.MaxValue;
                        foreach(KeyValuePair<int,
                            double> begin in prior) {
                                if (vertex [k,
                                    end.Key] > vertex
                                    [k-1, begin.Key] +
                                    trans[begin.Key,
                                    end.Key]) {
                                        vertex [k,
                                            end.Key] =
                                            vertex [k -
                                            1,
                                            begin.Key]
                                            + trans
                                            [begin.Key,
                                            end.Key];
                                }
                        }
                }
            }
            double min = double.MaxValue;
```

```
                foreach(KeyValuePair<int, double> pair in
                    prior)
                {
                        if (min > vertex [127, pair.Key])
                                min = vertex [127, pair.Key];
                        //Console.WriteLine("pair vector[{0}]
                            = {1}", pair.Key, pair.Value);
                }
                return min;
        }
```

A.3.5 Тест сжатия

```
                private static double functionG(double p, double n) {
                        double firstSum = 0;
                        for (int t = dictionaryLearningSetLength+1; t
                            <= n; t++) {
                                double secondSum = 0;
                                for (int s = 1; s < t; s++)
                                        secondSum += Math.Log (s, 2) *
                                            p * p * Math.Pow (1 - p, s
                                            - 1);
                                secondSum += Math.Log (t, 2) * p *
                                    Math.Pow (1 - p, t - 1);
                                firstSum += secondSum;
                        }
                        return firstSum / (n -
                            dictionaryLearningSetLength);
                }

                private static double functionE(double p, double
                    n,double m) {
                        double q = (1 - p) / (n - 1);
                        double Ep = functionG (p, n) + (n - 1) *
                            functionG (q, n);
                        return Ep - m;
                }

        private static readonly int dictionaryLearningSetLength = 1000;

        /// <summary>
        /// estimates minimal entropy for non-IID sources using
            compression test
        /// </summary>
        /// <param name="dataset"></param>
        /// <param name="sampleSize">number of bits in each
            sample</param>
        /// <returns>entropy estimate</returns>
        public static double compressionTest(int[] dataset, byte
            sampleSize) {
             var dictionary = new Dictionary<int, int>();
             if(dataset.Length <= dictionaryLearningSetLength){
                 throw new NotEnoughDataException("Compression test:
                     not enough data");
```

```
        }
        //first part of the dataset is used to create dictionary
        int position;
        for(position = 0; position < dictionaryLearningSetLength;
           ++position) {
            if(dictionary.ContainsKey(dataset[position])) {
                dictionary[dataset[position]] = position;
            } else {
                dictionary.Add(dataset[position], position);
            }
        }
        //second part of the dataset is used for the test
        var repetitions = new List<int>(dataset.Length - position);
        for(; position < dataset.Length; ++position) {
            if(dictionary.ContainsKey(dataset[position])) {
                repetitions.Add(position -
                   dictionary[dataset[position]]);
                dictionary[dataset[position]] = position;
            } else {
                repetitions.Add(position);
                dictionary.Add(dataset[position], position);
            }
        }
        double lowerBound =
           StandardFunctions.confidenceInterval(repetitions)[0];
                    double p = solveBinarySearch (functionE,
                       sampleSize, lowerBound);
                    return -Math.Log (p, 2);
    }
```

A.3.6 Частотный тест

```
    /// <summary>
    /// estimates minimal entropy for non-IID sources using
       frequency test
    /// </summary>
    /// <param name="dataset"></param>
    /// <param name="sampleSize">number of bits in each
       sample</param>
    /// <returns>entropy estimate</returns>
    public static double frequencyTest(int[] dataset, byte
       sampleSize) {
        double maxProb = dataset.GroupBy(x => x).Max(x =>
           x.Count()) / (double)dataset.Length;
        return -Math.Log(maxProb + Math.Sqrt(0.5 * Math.Log(1 / (1
           - confidenceLevel), 2) / dataset.Length), 2);
    }
```

ПРИЛОЖЕНИЕ Б

Пример вывода результатов применения разработанного ПО

```
>EntropySourceTesting.exe data1.bin 1
Имя файла: data1.bin
Размер наблюдения в битах: 1

Хи-квадрат тест независимости для бинарных данных
2-битные строки
Статистика: 1,05940167178556
Степени свободы: 3
P-значение: 0,786882595535632
 ПРОЙДЕНО
3-битные строки
Статистика: 8,59778029152712
Степени свободы: 7
P-значение: 0,282837419348987
 ПРОЙДЕНО
4-битные строки
Статистика: 8,5445215312206
Степени свободы: 15
P-значение: 0,900104722757102
 ПРОЙДЕНО
5-битные строки
Статистика: 17,3931705350896
Степени свободы: 31
P-значение: 0,976553796184551
 ПРОЙДЕНО
6-битные строки
Статистика: 78,4500673374023
Степени свободы: 63
P-значение: 0,0907788166889782
 ПРОЙДЕНО
7-битные строки
Статистика: 123,073552236177
Степени свободы: 127
P-значение: 0,581994389766079
 ПРОЙДЕНО
8-битные строки
Статистика: 225,724166390702
Степени свободы: 255
P-значение: 0,906542467568111
 ПРОЙДЕНО
9-битные строки
Статистика: 533,320631367063
Степени свободы: 511
P-значение: 0,239141067880565
 ПРОЙДЕНО
10-битные строки
Статистика: 1026,14996542014
Степени свободы: 1023
P-значение: 0,466407672790989
```

```
 ПРОЙДЕНО
11-битные строки
Статистика: 2038,33725348191
Степени свободы: 2047
P-значение: 0,549800289599266
 ПРОЙДЕНО

Хи-квадрат тест одинаковой распределенности для бинарных данных
Статистика: 3,06259909290993
Степени свободы: 9
P-значение: 0,961761274722441
 ПРОЙДЕНО
Хи-квадрат тест ПРОЙДЕНО
Тест сжатия: Старт... 100% ...
Тест перестановок: Окончен
Оценка сжатия
313 289 144 289 197 840 787 925 557 817
 ПРОЙДЕНО
Оценка самой длинной серии
670 434 911 500 826 436 659 500 467 43
 ПРОЙДЕНО
Оценка количества серий
500 500 500 526 499 495 481 475 500 516
 ПРОЙДЕНО
Оценка кумулятивного отклонения
434 673 900 288 125 220 659 497 327 400
 ПРОЙДЕНО
Оценка самой длинной направленной серии
500 500 778 928 406 500 440 500 786 399
 ПРОЙДЕНО
Оценка количества направленных серий
61 502 348 131 734 688 337 678 634 388
 ПРОЙДЕНО
Оценка количества -1/1 в направленных сериях
652 271 130 193 546 470 703 466 862 697
 ПРОЙДЕНО
Оценка ковариации
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка минимального количества коллизий
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка максимального количества коллизий
500 500 500 500 500 500 500 500 500 500
 ПРОЙДЕНО
Оценка среднего количества коллизий
500 514 502 492 494 481 500 504 480 504
 ПРОЙДЕНО
Тест перестановок ПРОЙДЕНО
H.o.p. тест ПРОЙДЕНО

Оценка энтропии для н.о.р. данных: 0,692203546584333
Минимальная оценка энтропии: 0,692203546584333
```