МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
Балтийский федеральный университет имени Иммануила Канта
(БФУ им. И.Канта)

Институт физико-математических наук и информационных технологий

ВКР допущена к защите
Первый заместитель директора Института физико-математических наук и информационных технологий, к.ф.-м.н., доцент
_____ Шпилевой А.А.
“____” _____2017 г.

Выпускная квалификационная работа

на тему: «**Анализ и реализация алгоритмов кодирования и декодирования NXL и XNL-кодов на основе алгоритма Судана**»

ВКР защищена на оценку:
_____

Студент 6 курса
специальности «Компьютерная безопасность»
_____ И.О. Бардинов

Руководитель
К.т.н., доцент Института физико-математических наук и информационных технологий
_____ С.И. Алешников
_____

Рецензент
Технический Директор ООО «Хирокрафт»
_____ Ю.Ю. Кешишев

Калининград
2018

Оглавление

Введение

Код представляет собой схему для обнаружения и исправления ошибок во время передачи информации, которые могут возникать в зашумленных каналах связи. Код добавляет избыточную информацию в сообщения. Одной из главных целей теории кодирования является построение эффективных кодов, то есть кодов, которые достигают заданной возможности исправления ошибок с минимальным избыточным количеством информации.

Проблема поиска хороших кодов – основная проблема в теории помехоустойчивого кодирования. На протяжении многих лет теоретики, занимающиеся этим, рассматривали эту проблему, добавляя в теорию множество алгебраических, геометрических и комбинаторных элементов. В частности, коды с хорошими свойствами были получены, используя различные элементы и методы из алгебры и геометрии, отсюда получив название алгеброгеометрические коды. Изучение таких кодов является непростой задачей.

Алгеброгеометрические коды (АГ-коды, коды Гоппы) – это обширное семейство линейных кодов, которые строятся с помощью алгебраических кривых над конечными полями. Они были придуманы советским и российским математиком Валерием Денисовичем Гоппой [1]. В частных случаях эти коды могут иметь очень интересные свойства.

Как было замечено выше, вычисление минимального расстояния линейного кода является довольно большой проблемой (это NP-полная задача). Обычно нам приходится рассчитывать оценку минимального расстояния на основе некоторой доступной нижней границы, а затем оценивать наши параметры, сравнив их с несколькими верхними границами. Обычно верхние границы являются общими.

Алгеброгеометрические коды – это первая конструкция блочных кодов, которая дает последовательность кодов асимптотически лучших, чем граница Гилберта – Варшамова. Эта граница определяет предельные значения для параметров кода.

Полезность кодов Гоппы обусловлена тем, что существуют эффективные алгоритмы декодирования для этого семейства кодов. Проблема декодирования для кода над конечным полем заданной длины означает, что мы хотим найти одно (или все) кодовое слово (слова) кода в пределах заданного расстояния Хэмминга.

Естественное направление развития теории кодирования связано с исследованием методов построения новых кодов. Коды Гоппы дают отличные ли-

нейные коды над $\mathbb{F}_q$ (конечным полем) при некоторых больших значениях q. Однако они практически бесполезны при малых значениях q. Суть этой проблемы состоит в том, что для построения АГ-кодов требуются рациональные точки функционального поля над $\mathbb{F}_q$, а для небольшого q существует слишком мало рациональных точек. Возможное решение – разработать конструкцию линейных кодов, использующую рациональные точки более высокой степени. В этой работе мы рассмотрим две такие конструкции, которые были предложены Ксингом и Нидеррайтером [2, 3, 4].

Обе эти конструкции позволяют использовать точки произвольной степени и получать хорошие коды на выходе. Было найдено много превосходных примеров, улучшающих предыдущие конструкции линейных кодов, и некоторые из этих кодов являются оптимальными в том смысле, что они дают линейный $[n, k, d]$-код над $\mathbb{F}_q$, такой, что для этих значений q, n, k нет линейного $[n, k]$-кода над $\mathbb{F}_q$ с минимальным расстоянием, большим d.

В настоящее время рассматривались лишь способы приведения этих конструкций к алгеброгеометрическим кодам с дальнейшим декодированием известными алгоритмами. Однако алгоритма декодирования конкретно для этих конструкций, который бы восстанавливал исходную информацию, исправляя заданное количество ошибок, никто не разрабатывал. В данной работе мы попытаемся обобщить алгоритм декодирования Гурусвами-Судана [5] для кодов Гоппы на конструкции, предложенные Ксингом и Нидеррайтером. Разработка и построение алгоритма может привести к нахождению других конструкций, способных восстанавливать больше ошибок, а также может найти применение в программно-аппаратных комплексах передачи информации. Также можно получить алгоритм с меньшей сложностью, что в совокупности с увеличением количества исправляемых ошибок может привести к уменьшению затрат ресурсов. Из-за увеличения количества передаваемой информации в мире, уменьшение затрат ресурсов на восстановление информации и увеличение числа исправляемых ошибок является достаточно актуальным.

Целью дипломной работы является разработка алгоритма декодирования для NXL и XNL-кодов на основе алгоритма Гурусвами-Судана.

Задачами дипломной работы являются:

1. Провести обзор предварительных сведений;
2. Исследовать конструкции NXL- и XNL-кодов;
3. Рассмотреть алгоритм декодирования Гурусвами-Судана;
4. Обобщить алгоритм декодирования Гурусвами-Судана на NXL и XNL-коды;

5. Описать программный комплекс, реализующий алгоритмы кодирования и декодирования NXL и XNL-кодов.

1. Обзор предварительных результатов

1.1. Обзор теории алгебраических кривых

Определение. Пусть k – совершенное поле. Аффинным (проективным) многообразием размерности 1 над полем k называется аффинная (проективная) кривая над полем k.

Для указания поля, над которым задана кривая, мы будем писать $\mathcal{X}/k$ для кривой $\mathcal{X}$ над полем k.

Пример.

1. Проективное многообразие над полем k с $char(k) = 2$, заданное уравнением $zy^2 + yz^2 = x^3$, является проективной кривой.

2. Аффинное многообразие над полем k с $char(k) \neq 2$, заданное уравнением $y^2 = f(x)$, является аффинной кривой, где $f \in k[x]$ – не являющийся квадратом многочлен положительной степени.

Определение. Имеем $\mathcal{X} \subseteq \mathbb{A}^n$ – аффинная кривая, пусть множество

$$f_1, \dots, f_m \in \bar{k}[x_1, \dots, x_n] = \bar{k}[X]$$

будет являться множеством порождающих элементов идеала $I(\mathcal{X})$. Тогда кривая $\mathcal{X}$ называется несингулярной (гладкой) кривой в точке P, если матрица Якоби $\left(\frac{\partial f_i}{\partial x_j}(P)\right)_{1 \leq i \leq m, 1 \leq j \leq n}$ размерности $m \times n$ в точке P имеет ранг $n - 1$. В ином случае, кривая называется сингулярной в точке P. Если кривая является гладкой в каждой точке, то мы говорим, что $\mathcal{X}$ – несингулярная (гладкая) аффинная кривая.

Также мы можем говорить о сингулярности и несингулярности точек, в которых кривая является сингулярной или несингулярной соответственно.

Пример. Пусть $\mathcal{X}$ – аффинная плоская кривая, определенная уравнением $f(x, y) = 0$, где $f \in k[x, y]$. Точка P кривой $\mathcal{X}$ – гладкая тогда и только тогда, когда

$$\left(\frac{\partial f}{\partial x}(P), \frac{\partial f}{\partial y}(P)\right) \neq (0,0).$$

Другими словами, все сингулярные точки $(a, b) \in \mathcal{X}$ – решения системы уравнений

$$\begin{cases} f(a,b) = 0 \\ \left.\frac{\partial f}{\partial x}\right|_{(a,b)} = 0 \\ \left.\frac{\partial f}{\partial y}\right|_{(a,b)} = 0 \end{cases}.$$

Если $P = (a, b)$ – гладкая точка кривой $\mathcal{X}$, тогда уравнение

$$\frac{\partial f}{\partial x}(P)(x - a) + \frac{\partial f}{\partial y}(P)(y - b) = 0$$

определяет касательную к кривой $\mathcal{X}$ в точке P.

Теорема. Пусть P – точка аффинной кривой $\mathcal{X}$. Тогда $\mathcal{X}$ – гладкая в точке P тогда и только тогда, когда локальное кольцо $\mathcal{O}_P$ в точке P – кольцо дискретного нормирования.

Теорема. Множество сингулярных точек кривой $\mathcal{X}/k$ конечно.

Определение. Для несингулярной точки P кривой $\mathcal{X}$ локальный параметр кольца дискретного нормирования $\mathcal{O}_P(\mathcal{X})$ называется локальным параметром или униформизирующим параметром в точке P.

Для двух локальных колец A и B с $A \subseteq B$ мы говорим, что B доминирует над A, если максимальный идеал кольца B содержит максимальный идеал кольца A.

Лемма. Пусть $\mathcal{X}/k$ – несингулярная проективная кривая. Предположим, что R – кольцо дискретного нормирования с полем частных $\bar{k}(\mathcal{X})$. Тогда существует единственная точка P кривой $\mathcal{X}$, такая, что R доминирует над $\mathcal{O}_P(\mathcal{X})$.

Теорема. Пусть $\mathcal{X}/k$ – несингулярная проективная кривая. Тогда отображение

$$p: P \to \mathcal{O}_P = \mathcal{O}_P(\mathcal{X})$$

показывает взаимно-однозначное соответствие между точками кривой $\mathcal{X}$ и кольцами дискретного нормирования с полем частных $\bar{k}(\mathcal{X})$.

Лемма. Пусть P и Q – две точки проективной кривой $\mathcal{X}/k$. Предположим, что эти точки принадлежат одной и той же $\mathbb{F}_q$-замкнутой точке кривой $\mathcal{X}$, и пусть $\sigma \in Gal(\bar{k}/k)$ удовлетворяет $Q = \sigma(P)$. Тогда

$$\mathcal{O}_Q(\mathcal{X}) = \sigma\big(\mathcal{O}_P(\mathcal{X})\big).$$

Предложение. Пусть $\mathcal{X}/\mathbb{F}_q$ – несингулярная проективная кривая. Тогда две точки P и Q кривой $\mathcal{X}$ принадлежат одной и той же $\mathbb{F}_q$-замкнутой точке тогда и только тогда, когда

$$\mathcal{O}_P(\mathcal{X}) \cap \mathbb{F}_q(\mathcal{X}) = \mathcal{O}_Q(\mathcal{X}) \cap \mathbb{F}_q(\mathcal{X}).$$

Теорема. Пусть $\mathcal{X}/\mathbb{F}_q$ – несингулярная проективная кривая. Тогда существует естественное взаимно-однозначное соответствие между $\mathbb{F}_q$-замкнутыми точками $\mathcal{X}$ и точками $\mathbb{F}_q(\mathcal{X})$. Более того, степени этих точек совпадают.

Также мы можем говорить о локальном параметре этих точек.

Пример. Определим проективную прямую $\mathcal{X} = \mathbb{P}^1\left(\overline{\mathbb{F}_q}\right)$. Она является несингулярной проективной кривой над полем $\mathbb{F}_q$, и ее $\mathbb{F}_q$-рациональное функциональное поле является рациональным функциональным полем $\mathbb{F}_q(x)$ над полем $\mathbb{F}_q$ от переменной x. Для $\mathbb{F}_q$-замкнутой точки P кривой $\mathcal{X}$, соответствующей конечной точке, точки в P ассоциируются с корнями соответствующего нормированного неприводимого многочлена в $\overline{\mathbb{F}_q}$.

Лемма. Рациональное отображение из многообразия в кривую либо постоянно, либо доминантно.

Теорема. Существует взаимно-однозначное соответствие между классами $\mathbb{F}_q$-изоморфизмов несингулярных проективных кривых над k и классами k-изоморфизмов алгебраических функциональных полей от одной переменной с полным полем констант k, задаваемое отображением

$$X: \mathcal{X}/k \to k(\mathcal{X}).$$

Теперь приступим к дивизорам. Имеем алгебраическое функциональное поле F/k от одной переменной с полем констант k. Будем писать P_F для обозначения множества точек поля F. Группа дивизоров поля F/k, обозначаемая как $Div(F/k)$ или $Div(F)$, является свободной абелевой группой, порожденной точками поля F, то есть дивизор из $Div(F)$ – это формальная сумма

$$\sum_{P \in P_F} n_P P$$

с коэффициентами $n_P \in \mathbb{Z}$, причем $n_P = 0$ для всех $P \in P_F$, кроме конечного числа. Дивизор D называется положительным (или эффективным), записываемый как $D \geq 0$, если $n_P \geq 0 \ \ \forall P \in P_F$.

Для двух дивизоров D и G поля F, таких, что $D - G \geq 0$, будем писать $D \geq G$ или $G \leq D$.

Для дивизора $D \in Div(F)$ удобно полагать, что $v_P(D) \coloneqq n_P$ для $P \in P_F$.

Носителем дивизора D называется конечное множество

$$supp(D) \coloneqq \{P \in P_F : v_P(D) \neq 0\}.$$

Степенью дивизора D называется

$$\deg(D) = \sum_{P \in P_F} n_P \deg(P).$$

Ясно, что степень дивизора определяет групповой гомоморфизм из $Div(F)$ в $\mathbb{Z}$. Ядро этого гомоморфизма – подгруппа $D(F)$, обозначаемая как $Div^0(F)$, то есть

$$Div^0(F) \coloneqq \{D \in Div(F) : \deg(D) = 0\}.$$

Предложение. Для любого элемента $x \in F/k$ имеем

$$\sum_{\substack{P \in P_F \\ v_P(x) \geq 0}} v_P(x) \deg(P) \leq [F : k(x)].$$

Следствие. Каждый элемент F^* имеет только конечное число нулей и полюсов.

Пусть $\mathcal{N}(x), x \in F^*$ – множество нулей, $\mathcal{P}(x)$ – множество полюсов x. Они являются конечными множествами. Определим дивизор нулей как

$$(x)_0 = \sum_{P \in \mathcal{N}(x)} v_P(x) P$$

и дивизор полюсов как

$$(x)_\infty = \sum_{P \in \mathcal{P}(x)} (-v_P(x)) P.$$

Заметим, что они оба являются положительными дивизорами. Теперь определим главный дивизор как

$$div(x) = (x)_0 - (x)_\infty = \sum_{P \in P_F} v_P(x) P.$$

Предложение. Каждый элемент поля F/k имеет по крайней мере один ноль и один полюс. В частности, ядро $div(x)$ – это k^*.

Для дивизора D поля F/k определим пространство Римана-Роха

$$\mathcal{L}(D) \coloneqq \{x \in F^* : div(x) + D \geq 0\} \cup \{0\}.$$

Будем обозначать размерность этого векторного пространства как $\ell(D)$.

Предложение. Пусть D и G – дивизоры F/k. Тогда:

1. Если $D \leq G$, то $\mathcal{L}(D) \subseteq \mathcal{L}(G)$ и $\dim_k \left(\frac{\mathcal{L}(G)}{\mathcal{L}(D)}\right) \leq \deg(G) - \deg(D)$.
2. $\mathcal{L}(0) = k$.
3. $\ell(D) \geq 1$, если $D \geq 0$.
4. $\ell(D)$ – конечное для всех D.
5. Если $D = G + div(x)$ для некоторого $x \in F$, то $\ell(D) = \ell(G)$.

Теорема. Для любого $x \in F/k$ имеем

$$\deg((x)_0) = [F : k(x)].$$

Следствие. Для любого ненулевого $x \in F$ имеем

$$\deg\big(div(x)\big) = 0.$$

Следствие 2. $\ell(D) = 0$, когда $\deg(D) < 0$.

Теорема (Теорема Римана). Для любого функционального поля F существует неотрицательное целое g, зависящее только от F, такое, что

$$\ell(D) \geq \deg(D) + 1 - g$$

для любого дивизора D поля F.

Следствие. Если $\ell(D) = \deg(D) + 1 - g$ и $G \geq D$, то

$$\ell(G) = \deg(G) + 1 - g.$$

Следствие. Существует целое r, зависящее от поля F, такое, что

$$\ell(D) = \deg(D) + 1 - g$$

для всех дивизоров D поля F с $\deg(D) \geq r$.

Определение. Неотрицательное целое g из предыдущего следствия называется родом функционального поля F. Род несингулярной проективной кривой $\mathcal{X}$ над полем k определяется как род ее $\mathbb{F}_q$-рационального функционального поля $k(\mathcal{X})$.

Определение. Аделем функционального поля F/k называется отображение $\alpha: P_F \to F$, задаваемое $P \mapsto \alpha_P$, такое, что $\alpha_P \in \mathcal{O}_P$ для всех кроме конечного числа точек P поля F. Пространство $\mathcal{A}_F$ содержит все адели поля F/k и называется адель-пространством поля F.

Определение. Для любого дивизора $D \in Div(F)$ определим множество

$$\mathcal{A}_F(D) \coloneqq \{\alpha \in \mathcal{A}_F : v_P(\alpha) + v_P(D) \geq 0 \forall P \in P_F\}.$$

Определение. Дифференциалом Вейля функционального поля F/k называется $\mathbb{F}_q$-линейное отображение $\omega: \mathcal{A}_F \to k$, такое, что $\omega|_{\mathcal{A}_F(D)+F} \equiv 0$ для некоторого дивизора $D \in Div(F)$. Пусть Ω_F – множество дифференциалов Вейля поля F. Определим $\Omega_F(D)$ как множество дифференциалов Вейля, которые обнуляются на $\mathcal{A}_F(D) + F$.

Определение. Дивизор (ω) ненулевого дифференциала Вейля ω поля F – единственный дивизор W поля F, такой, что:

1. ω обнуляется на $\mathcal{A}_F(W) + F$;
2. Если ω обнуляется на $\mathcal{A}_F(D) + F$, тогда $D \leq W = (\omega)$;

Дивизор (ω) для некоторого ненулевого дифференциала Вейля ω поля F называется каноническим.

Теорема (Римана-Роха). Пусть W – канонический дивизор функционального поля F/k рода g. Тогда для любого дивизора $D \in Div(F)$ имеем

$$\ell(D) = \deg(D) + 1 - g + \ell(W - D).$$

Более того, $\ell(D) = \deg(D) + 1 - g$, когда $\deg(D) \geq 2g - 1$.

Определение. Для любого целого $n \geq 0$ пусть $A_n(F)$ – мощность множества дивизоров $D \in Div(F)$, причем $D \geq 0$ и $\deg(D) = n$.

Предложение. $A_n(F)$ является конечным множеством для $\forall n \geq 0$.

Следствие. Функциональное поле имеет конечное число рациональных точек.

Пусть $N(F)$ обозначает число рациональных точек $F/\mathbb{F}_q$. Для каждого целого $n \geq 1$ определим расширение поля констант $F_n \coloneqq F \cdot \mathbb{F}_{q^n}$. Пусть $N_n(F)$ – число его рациональных точек.

Определение. Дзета-функцией $Z(F,t)$ называется формальный степенной ряд

$$Z(F,t) \coloneqq \exp\left(\sum_{n=1}^{\infty} \frac{N_n(F)}{n} t^n\right) \in \mathbb{C}[[t]].$$

Пример. Подсчитаем дзета-функцию поля $\mathbb{F}_q(x)$ над полем $\mathbb{F}_q$. Для всех $n \geq 1$ имеем $F_n = \mathbb{F}_{q^n}(x)$ и $N_n(F) = q^n + 1$. Следовательно,

$$\log Z(F,t) = \sum_{n=1}^{\infty} \frac{q^n+1}{n} t^n = \sum_{n=1}^{\infty} \frac{(qt)^n}{n} + \sum_{n=1}^{\infty} \frac{t^n}{n} =$$
$$= -\log(1-qt) - \log(1-t) = \log\left(\frac{1}{(1-t)(1-qt)}\right),$$

то есть

$$Z(F,t) = \frac{1}{(1-t)(1-qt)}.$$

Предложение. Дзета-функция $Z(F,t)$ может быть представлена следующим образом:

1. $Z(F,t) = \prod_{P \in P_F} \left(1 - t^{\deg(P)}\right)^{-1}$;
2. $Z(F,t) = \sum_{n=1}^{\infty} A_n(F) t^n$.

Предложение. Для любого положительного целого n имеем

$$Z(F_n, t^n) = \prod_{\xi^n=1} Z(F, \xi t),$$

где произведение берется по всем комплексным корням из единицы.

Два дивизора D и G называются эквивалентными, если

$$G = D + div(x), x \in F^*.$$

Класс дивизора $[D] \coloneqq D + Princ(F)$ состоит из всех дивизоров F, эквивалентных D. Определим факторгруппу

$$Cl \coloneqq Div^0(F)/Princ(F),$$

которая называется группой классов дивизоров степени 0.

Предложение. Факторгруппа $Cl(F)$ является конечной абелевой группой.

Порядок этой группы обозначается как $h(F)$ и называется числом классов дивизоров поля F.

Лемма. Для любого дивизора $D \in Div(F)$ число положительных дивизоров в классе $[D]$ задается как

$$\frac{q^{\ell(D)}-1}{q-1}.$$

Лемма. Любое функциональное поле имеет дивизор степени 1. В частности, любое такое поле рода 0 является рациональным функциональным полем.

Теорема. Пусть $F/\mathbb{F}_q$ – функциональное поле рода g. Тогда дзета-функция $Z(F,t)$ поля F – рациональная функция вида

$$Z(F,t) = \frac{L(F,t)}{(1-t)(1-qt)},$$

где $L(F,t) \in \mathbb{Z}[t]$ – многочлен степени не более $2g$ с целочисленными коэффициентами, $L(F,0) = 1$. Более того, $L(F,1)$ равно числу классов дивизоров $h(F)$ поля F.

1.2. Основные конструкции геометрических кодов

Обширное семейство линейных кодов было введено В. Д. Гоппой, используя алгебраические кривые над конечным полем и дифференциалы. Сейчас же алгеброгеометрический код определяют также через функциональное поле и пространство Римана – Роха.

Пусть $F/\mathbb{F}_q$ – функциональное поле над полем $\mathbb{F}_q$ рода g, $N(F) > 1$ – число рациональных точек поля F. Необходимо построить линейный код над $\mathbb{F}_q$ длины $n \leq N(F)$. Выберем n различных рациональных точек $P_1, \dots, P_n$ поля F и дивизор G из поля F, причем такой, что $supp(G) \cap \{P_1, \dots, P_n\} = \emptyset$. Определим пространство Римана – Роха

$$\mathcal{L}(G) = \{f \in F^* : div(f) + G \geq 0\} \cup \{0\},$$

где $div(f)$ – главный дивизор $f \in F^*$.

Стоит сказать, что $v_{P_i}(f) \geq 0$ для $1 \leq i \leq n$, и все $f \in \mathcal{L}(G)$.

Следовательно, класс $f(P_i)$, причем $f \in \mathcal{O}_{P_i}$, определен. Так как P_i – рациональная точка, то $f(P_i)$ можно отождествить с элементами поля $\mathbb{F}_q$.

Определение. Алгеброгеометрический код $C(P_1, \ldots, P_n; G)$ определяется как образ $\mathbb{F}_q$-линейного отображения $\psi: \mathcal{L}(G) \to \mathbb{F}_q^n$, где

$$\psi(f) = \big(f(P_1), \ldots, f(P_n)\big) \forall f \in \mathcal{L}(G).$$

Теорема. Пусть $F/\mathbb{F}_q$ – функциональное поле рода g, такое, что $N(F) \geq$ $\geq g + 1$. Выберем n различных рациональных точек $P_1, \ldots, P_n$ в поле F, причем $g < n \leq N(F)$, и дивизор G в поле F, такой, что $g \leq deg(G) < n$ и $supp(G) \cap \{P_1, \ldots, P_n\} = \emptyset$. Тогда $C(P_1, \ldots, P_n; G)$ – линейный $[n, k, d]$-код над $\mathbb{F}_q$ с

$$k = \ell(G) \geq deg(G) + 1 - g, \quad d \geq n - deg(G).$$

Более того, $k = deg(G) + 1 - g$, если $deg(G) \geq 2g - 1$.

В теореме говорится, что $k + d \geq n + 1 - g$. В границе Синглтона говорится о том, что $k + d \leq n + 1$. Таким образом, род поля F контролирует отклонение $k + d$ от границы Синглтона. Если $g = 0$, то есть если F – рациональное функциональное поле над полем $\mathbb{F}_q$, то любой АГ-код по этой теореме является МДР-кодом.

Пример. Пусть $F = \mathbb{F}_q(x)$ – рациональное функциональное поле над полем $\mathbb{F}_q$. Пусть $\mathbb{F}_q = \{b_1, \ldots, b_q\}$ и для $1 \leq i \leq q$ пусть P_i – рациональная точка $x - b_i$ в поле $\mathbb{F}_q(x)$. Положим $G = (k-1)P_\infty$, где k – целое, $1 \leq k \leq q$, а P_∞ обозначает бесконечную точку. Тогда АГ-код $C(P_1, \ldots, P_q; G)$ – код Рида – Соломона.

Пример. Пусть F – эрмитово функциональное поле над $\mathbb{F}_{q^2}$, то есть

$$F = \mathbb{F}_{q^2}(x, y) \text{ и } y^q + y = x^{q+1}.$$

Тогда F имеет род $g = \frac{q^2 - q}{2}$ и $N(F) = q^3 + 1$. Пусть Q – рациональная точка поля F, лежащая над бесконечной точкой поля $\mathbb{F}_{q^2}(x)$, и пусть $P_1, \ldots, P_n$, где $n = q^3$ – оставшиеся рациональные точки поля F. Положим $G = mQ$, где m – целое, удовлетворяющим $q^2 - q - 1 \leq m < q^3$. Тогда по теореме АГ-код $C(P_1, \ldots, P_n; G)$ – линейный $[n, k, d]$-код над $\mathbb{F}_{q^2}$ с параметрами $k = m + 1 - \frac{q^2 - q}{2}$ и $d \geq q^3 - m$. Такие коды называются эрмитовыми кодами.

Пример. Возьмем предыдущий пример с параметрами $q = 2$ и $m = 4$. Получим $G = 4Q$. Тогда эрмитов код $C(P_1, \dots, P_n; G)$ из предыдущего примера – линейный $[8, 4, d]$-код над $\mathbb{F}_4$ с $d \geq 4$. Можно показать, что $d = 4$. Сначала заметим, что $f(x) = x(x + 1) \in \mathbb{F}_4[x]$ удовлетворяет условию $f \in \mathcal{L}(G)$. Далее, f имеет ровно 4 нуля в поле F, а именно рациональные точки поля F лежат над x или $x + 1$. Вследствие этого, кодовое слово $\psi(f)$ имеет вес 4, а значит $d = 4$. Код $C(P_1, \dots, P_8; G)$ – оптимальный код в том плане, что не существует линейного $[8,4]$-кода над $\mathbb{F}_4$ с минимальным расстоянием ≥ 5.

Теорема. Пусть $F/\mathbb{F}_q$ – функциональное поле, $N(F) \geq 1$, а для целого $1 \leq n \leq N(F)$ имеем дивизор G в поле F, такой, что $deg(G) < n$ и $\ell(G) \geq 1$. Тогда существует алгеброгеометрический код, который является линейным $[n, k, d]$-кодом над полем $\mathbb{F}_q$, причем

$$k = \ell(G), d \geq n - deg(G).$$

Пример. Пусть F – рациональное функциональное поле над $\mathbb{F}_q$. Теперь выберем $n = q + 1$ по теореме. Для целого k с $1 \leq k \leq q + 1$, пусть G – дивизор поля F с $\deg(g) = k - 1$. Теперь по теореме можем видеть, что существует линейный $[q + 1, k, d]$-код над $\mathbb{F}_q$ с параметрами $d \geq q - k + 2$. С другой стороны, граница Синглтона говорит о том, что $d \leq q - k + 2$. Следовательно, $d = = q - k + 2$, а этот код является МДР-кодом.

Пример. Пусть F – эрмитово функциональное поле над $\mathbb{F}_4$. Так как $N(F) = 9$, выберем $n = 9$. Пусть G – дивизор поля F с $\deg(G) = 4$. Так как $g(G) = 1$, получаем $\ell(G) = 4$. Теперь по теореме получаем линейный $[9,4, d]$-код над $\mathbb{F}_4$ с параметром $d \geq 5$. Но известно, что не существует линейного $[9,4]$-кода над $\mathbb{F}_4$ с минимальным расстоянием ≥ 6. Отсюда следует, что $d = 5$.

1.3. Анализ литературы

Для исследования XNL- и NXL-кодов мы будем использовать информацию, изложенную в книге Нидеррайтера и Ксинга [4]. В ней описаны конструкции этих кодов и представлены доказательства. Мы опишем это более подробно позже. В статье [3] и книге [4] представлены параметры для построения оптимальных XNL-кодов.

Также Штихтенот в своей книге [12] доказывает, что можно легко свести NXL-коды к кодам XNL. По этой причине мы сконцентрируемся на разработке алгоритма декодирования для XNL-кодов.

Основная цель нашего исследования – это разработка алгоритма декодирования для XNL-кодов. На текущий момент нет публикаций или исследований по разработке алгоритма декодирования в свободном доступе. Имеет смысл рассмотреть существующие алгоритмы декодирования для алгеброгеометрических кодов и выбрать наиболее подходящий для исследуемого нами кода.

В книге [16] присутствует краткое описание алгеброгеометрических кодов и общее описание известных алгоритмов декодирования.

В статье [17] изложен принцип построения алгеброгеометрических кодов на основе дивизора, построенного с помощью нескольких рациональных точек. Это можно использовать для наших XNL-кодов.

Существует несколько известных алгоритмов декодирования для АГ-кодов. Например, алгоритм Фенга – Рао. Одной из его особенностей является построение базиса поля $\mathbb{F}_q$, с чем у нас может возникнуть сложности, так как мы используем поля $\mathbb{F}_{q^i}$ для точек различных степеней i. Также это не самый эффективный алгоритм декодирования в настоящее время.

В публикации [8] Хохольд и Пеликан предлагают свой алгоритм декодирования для алгеброгеометрических кодов. Его проблема похожа на ту, что у алгоритма Фенга-Рао. Он использует поле $\mathbb{F}_q$, и адаптация для полей $\mathbb{F}_{q^i}$ может не получится. Также количество исправляемых ошибок не самое лучшее на данный момент.

Также существует алгоритм декодирования Гурусвами-Судана [5], который основан на алгоритме интерполяции (построение многочлена по определенным параметрам) и поиске корней в интерполированном многочлене (его также называют алгоритмом факторизации многочлена). Его удобство в том, что сам алгоритм не привязан к конкретному полю, а также он содержит в себе два алгоритма – интерполяцию и факторизацию. Существует множество статей и исследований по расширению этого алгоритма на различные коды – Рида-Соломона, классические коды Гоппы, эрмитовы коды.

Также стоит отметить, что алгоритмы интерполяции и факторизации для декодирования Гурусвами-Судана мы можем выбрать любые.

Большинство алгоритмов для списочных алгоритмов декодирования (каковым является и сам Гурусвами-Судан) описаны для кодов Рида-Соломона, однако один из самых последних и наиболее быстрых, который описан в статье [15], разработан для АГ-кодов. Существует его программная реализация в си-

стеме компьютерной алгебры SAGE от автора статьи. Алгоритм основан на построении систем линейных уравнений (в реализации используются матрицы) и нахождении их решения для построения интерполированного многочлена.

В статье [7] описано усовершенствование алгоритма декодирования Гурусвами-Судана. Автор оптимизировал создание матриц для нахождения искомого многочлена, что увеличило скорость работы алгоритма, однако она меньше чем в [15].

В статье [11] Хохольд и Нильсен адаптировали алгоритм декодирования Судана для Эрмитовых кодов. Мы можем использовать их наработки для нашей адаптации алгоритма для XNL-кодов.

Кроме этого, в статьях [9] и [13] также изложены алгоритмы факторизации и интерполяции. Алгоритмы достаточно старые и не обеспечивают такой скорости, как алгоритмы в статье Нильсена [15].

Также есть алгоритм нахождения корней многочлена, изложенный в статье [14]. Его скорость по сравнению с алгоритмом Нильсена оставляет желать лучшего.

2. Основные теоретические результаты

2.1. XNL и NXL-коды на алгебраических кривых

Пусть $F/\mathbb{F}_q$ – функциональное поле рода g. Пусть $G_1, \dots, G_r$ – положительные дивизоры поля F, причем их носители – попарно непересекающиеся, и

$$s_i \coloneqq \deg(G_i) \geq 1 \text{ для } 1 \leq i \leq r.$$

Положим $n = \sum_{i=1}^{r} s_i$ – длина конструируемого кода, а также пусть $n > g$. Также пусть E – положительный дивизор поля F, причем

$$supp(E) \cap supp(G_i) = \emptyset \text{ для } 1 \leq i \leq r.$$

Теперь пусть D – дивизор поля F, такой, что $\ell(D) = \deg(D) + 1 - g$. Также предположим, что $1 \leq \deg(E - D) \leq n - g$.

Заметим, что

$$\ell(D + G_i) = \ell(D) + s_i \text{ для } 1 \leq i \leq r.$$

Для каждого $i = 1, \dots, r$ выберем $\mathbb{F}_q$-базис

$$\{f_{i,j} + \mathcal{L}(D) : 1 \leq j \leq s_i\}$$

факторпространства $\mathcal{L}(D + G_i)/\mathcal{L}(D)$.

Факторпространство

$$\mathcal{L}(D + \textstyle\sum_{i=1}^{r} G_i)/\mathcal{L}(D)$$

размерности n имеет $\mathbb{F}_q$-базис

$$\{f_{i,j} + \mathcal{L}(D) : 1 \leq j \leq s_i, 1 \leq i \leq r\},$$

который мы упорядочим лексикографически. Имеем сумму $\sum_{i=1}^{r} h_i \in \mathcal{L}(D)$, где $h_i \in \mathcal{L}(D + G_i)$ для $1 \leq i \leq r$, тогда для каждого $b = 1, \dots, r$ имеем

$$h_b = \sum_{i=1}^{r} h_i - \sum_{\substack{i=1 \\ i \neq b}}^{r} h_i \in \mathcal{L}\left(D + \sum_{\substack{i=1 \\ i \neq b}}^{r} G_i\right).$$

Это показывает, что $v_P(h_b) + v_P(D) \geq 0$ для каждой точки $P \in supp(G_b)$. Но так как $h_b \in \mathcal{L}(D + G_b)$, то $h_b \in \mathcal{L}(D)$.

Теперь построим линейный код. Сперва заметим, что каждый элемент

$$f \in \mathcal{L}(D + \sum_{i=1}^{r} G_i - E) \subseteq \mathcal{L}(D + \sum_{i=1}^{r} G_i)$$

имеет единственную запись

$$f = \sum_{i=1}^{r} \sum_{j=1}^{s_j} c_{i,j} f_{i,j} + u,$$

где все $c_{i,j} \in \mathbb{F}_q, u \in \mathcal{L}(D)$.

Определение. NXL-код $C(G_1, \dots, G_r; D, E)$ определяется как образ $\mathbb{F}_q$ - линейного отображения $\eta: \mathcal{L}(D + \sum_{i=1}^{r} G_i - E) \to \mathbb{F}_q^n$, задаваемого

$$\eta(f) = (c_{1,1}, \dots, c_{1,s_1}, \dots, c_{r,1}, \dots, c_{r,s_r})$$

для всех $f \in \mathcal{L}(D + \sum_{i=1}^{r} G_i - E)$.

Следующая теорема показывает границы на размерность и минимальное расстояние этого линейного кода.

Теорема. Пусть $F/\mathbb{F}_q$ – функциональное поле рода g и пусть $G_1, \dots, G_r$ – положительные дивизоры поля F с положительными степенями $s_1, \dots, s_r$ соответственно, а также их носители – попарно непересекающиеся. Пусть D – дивизор поля F, такой, что $\ell(D) = \deg(G) + 1 - g$. Далее, пусть E – положительный дивизор поля F, такой, что $supp(E) \cap supp(G_i) = \emptyset$ для $1 \leq i \leq r$, причем $m \coloneqq \deg(E - D)$ удовлетворяет

$$1 \leq m \leq \sum_{i=1}^{r} s_i - g.$$

Тогда $C(G_1, \dots, G_r; D, E)$ – линейный $[n, k, d]$-код над полем $\mathbb{F}_q$ с

$$n = \sum_{i=1}^{r} s_i, k = \ell(D + \sum_{i=1}^{r} G_i - E) \geq n - m - g + 1, d \geq d_0,$$

где d_0 – наименьшая мощность подмножества R из $\{1, \dots, r\}$, для которого $\sum_{i \in R} s_i \geq m$. Более того, имеем $k = n - m - g + 1$, если $n - m \geq 2g - 1$.

Доказательство. Возьмем ненулевой $f \in \mathcal{L}(D + \sum_{i=1}^{r} G_i - E)$, пусть w – вес $\eta(f)$. Имеем единственное разложение для f. Определим

$$R = \{1 \leq i \leq r : \exists j, 1 \leq j \leq s_i, c_{i,j} \neq 0\}.$$

Заметим, что $|R| \leq w$. Теперь можем записать f следующим образом

$$f = \sum_{i \in R} h_i + u,$$

где $h_i \in \mathcal{L}(D + G_i)$ для $i \in R$ и $u \in \mathcal{L}(D)$. Это значит, что выполняется

$$f \in \mathcal{L}(D + \sum_{i \in R} G_i).$$

Но мы также имеем $f \in \mathcal{L}(D + \sum_{i=1}^{r} G_i - E)$, а это показывает, что

$$f \in \mathcal{L}(D + \sum_{i \in R} G_i - E).$$

Так как $f \neq 0$, то (по следствию 2, страница 10)

$$\deg(D + \sum_{i \in R} G_i - E) \geq 0.$$

Отсюда получаем следующее

$$w \geq |R| \geq d_0 > 0.$$

Это показывает не только желаемую нижнюю границу для d, но и то, что η является инъективным. Следовательно, по теореме Римана – Роха (страница 11, предварительные сведения)

$$n = \sum_{i=1}^{r} s_i, \quad k = \ell(D + \sum_{i=1}^{r} G_i - E) \geq n - m - g + 1.$$

Замечание. Один из простых способов выбора дивизоров G_i – взять различные точки P_i поля F. Эти точки не обязательно должны быть рациональными, но могут иметь различные степени. Это будет использоваться в двух следующих примерах.

Пример. Пусть $q = 2$, пусть F – рациональное функциональное поле над $\mathbb{F}_2$, также положим $r = 6$. Для точек $P_1, \ldots, P_6$ выберем три рациональных точки, точку степени 2 и две точки степени 3 поля F. Пусть D – нулевой дивизор, E –

точка степени 7 в F. Тогда по теореме $C(P_1, \dots, P_6; D, E)$ – линейный $[n, k, d]$-код над $\mathbb{F}_2$ с $n = 11, k = 5, d \geq 3$. Значит, $k + d \geq 8$. Лучшая нижняя граница для $k + d$ для АГ-кода над $\mathbb{F}_2$ длины 11 достигается при $g = 8$. Следовательно, получаем $k + d \geq 4$.

Пример. Пусть $q = 3$, пусть F – рациональное функциональное поле над $\mathbb{F}_3$, пусть $r = 13$. Выберем для 13 точек: 4 рациональные точки, 3 точки степени 2 и 6 точек степени 3 поля F. Пусть D – нулевой дивизор и E – точка степени 7 поля F. Тогда существует линейный $[n, k, d]$-код $C(P_1, \dots, P_{13}; D, E)$ над $\mathbb{F}_3$ с параметрами $n = 28, k = 22, d \geq 3$. А значит $k + d \geq 25$. Лучшая нижняя граница для $k + d$ для АГ-кода над $\mathbb{F}_3$ длины 28 достигается при $g = 15$, и тогда $k + d \geq 14$.

Пример. Пусть $q = 5$, пусть F – рациональное функциональное поле над $\mathbb{F}_5$, пусть $r = 31$. Выберем для 31 точки: 12 рациональных точек, 7 точек степени 2 и 12 точек степени 3 поля F. Пусть D – нулевой дивизор и E – точка степени 7 поля F. Тогда существует линейный $[n, k, d]$-код $C(P_1, \dots, P_{31}; D, E)$ над $\mathbb{F}_5$ с параметрами $n = 62, k = 56, d \geq 3$. А значит $k + d \geq 59$. Лучшая нижняя граница для $k + d$ для АГ-кода над $\mathbb{F}_5$ длины 62 достигается при $g = 32$, и тогда $k + d \geq 31$.

Вторая конструкция – мощный метод, комбинирующий в себе функциональное поле с (короткими) линейными кодами, для создания длинного линейного кода. Он был предложен Ксингом, Нидеррайтером и Лэмом, и возглавляет семейство XNL-кодов. Пусть $F/\mathbb{F}_q$ – функциональное поле. Пусть $P_1, \dots, P_r$ – различные точки поля F, которые могут иметь разные степени. Пусть G – дивизор поля F, такой, что

$$supp(G) \cap \{P_1, \dots, P_r) = \emptyset.$$

Для каждого $i = 1, \dots, r$ пусть C_i – линейный $[n_i, k_i, d_i]$-код над полем $\mathbb{F}_q$, такой, что $k_i \geq \deg(P_i)$. Последнее условие гарантирует, что существует $\mathbb{F}_q$-линейное отображение ϕ_i из поля классов вычетов точки P_i в C_i, которое инъективно. То есть имеем ϕ: $\prod_{i=1}^{s} F_{P_i} \to \prod_{i=1}^{s} \mathrm{C_i}$. Положим $n = \sum_{i=1}^{r} n_i$, то есть n – сумма длин кодов C_i.

Определение. XNL-код $C(P_1, \dots, P_r; G; C_1, \dots, C_r)$ определяется как образ $\mathbb{F}_q$-линейного отображения $\beta: \mathcal{L}(G) \to \mathbb{F}_q^n$, задаваемое как

$$\beta(f) = \left(\phi_1\left(f(P_1)\right), \dots, \phi_r\left(f(P_r)\right)\right) \text{ для всех } f \in \mathcal{L}(G),$$

где справа мы используем конкатенацию векторов.

Следующая теорема описывает параметры таких кодов.

Теорема. Пусть $F/\mathbb{F}_q$ – функциональное поле рода g. Пусть $P_1, \dots, P_r$ – различные точки поля F. Для каждого $i = 1, \dots, r$ пусть C_i – линейный $[n_i, k_i, d_i]$-код над $\mathbb{F}_q$ с параметром $k \geq \deg(P_i)$. Далее, пусть G – дивизор поля F, такой, что

$$supp(G) \cap \{P_1, \dots, P_r\} = \emptyset \text{ и } g \leq \deg(G) < \sum_{i=1}^{r} \deg(P_i).$$

Тогда $C(P_1, \dots, P_r; G; C_1, \dots, C_r)$ – линейный $[n, k, d]$-код над $\mathbb{F}_q$ с

$$n = \sum_{i=1}^{r} n_i \,, k = \ell(G) \geq \deg(G) + 1 - g, d \geq d_0,$$

где d_0 – минимум $\sum_{i \in M} d_i$, взятый среди всех подмножеств M из $\{1, \dots, r\}$, для которых $\sum_{i \in M} \deg(P_i) \leq \deg(G)$, где $\bar{M}$ – дополнение к M в $\{1, \dots, r\}$. Более того, имеем $k = \deg(G) + 1 - g$, если $\deg(G) \geq 2g - 1$.

Доказательство. Возьмем ненулевой $f \in \mathcal{L}(G)$, пусть w – вес $\beta(f)$. Определим

$$M = \{1 \leq i \leq r : f(P_i) = 0\}.$$

Далее

$$w = \sum_{i \in \bar{M}} w_i \geq \sum_{i \in M} d_i,$$

где w_i – вес $\phi_i\left(f(P_i)\right)$ для $1 \leq i \leq r$. Имеем $f \in \mathcal{L}(G - \sum_{i \in M} P_i)$. Так как $f \neq 0$, то (по следствию 2, страница 10)

$$\deg(G - \textstyle\sum_{i \in M} P_i) \geq 0.$$

То есть $\sum_{i \in M} \deg(P_i) \leq \deg(G)$. Следовательно, $w \geq d_0 > 0$. Это показывает не только желаемую нижнюю границу, но и то, что β является инъективным.

Стало быть, $k = \ell(G)$. Из теоремы Римана – Роха (страница 11, предварительные сведения) следует, что $\ell(G) \geq \deg(G) + 1 - g$.

Следствие. В особых случаях, когда $\deg(P_i) \geq d_i$ для $1 \leq i \leq r$, минимальное расстояние линейного кода $C(P_1, \dots, P_r; G; C_1, \dots, C_r)$ удовлетворяет неравенству

$$d \geq \sum_{i=1}^{r} d_i - \deg(G).$$

Доказательство. Поступим как в предыдущей теореме и заметим, что если $\sum_{i \in M} \deg(P_i) \leq \deg(G)$, то

$$\sum_{i \in M^-} d_i = \sum_{i=1}^{r} d_i - \sum_{i \in M} d_i \geq \sum_{i=1}^{r} d_i - \sum_{i \in M} \deg(P_i) \geq \sum_{i=1}^{r} d_i - \deg(G).$$

Это означает, что $d_0 \geq \sum_{i=1}^{r} d_i - \deg(G)$.

Замечание. Если $P_1, \dots, P_n$ – различные рациональные точки поля F, и для каждого $i = 1, \dots, r$ в качестве C_i выбран тривиальный линейный $[1,1,1]$-код над $\mathbb{F}_q$, а для ϕ_i – тождественное отображение на $\mathbb{F}_q$, тогда конструкция XNL-кодов сводится к АГ-кодам.

Пример. Пусть $q = 2$. Пусть $F = \mathbb{F}_2(x, y)$ – поле эллиптических функций, определенное следующим образом

$$y^2 + y = x + \frac{1}{x}.$$

Тогда F имеет 4 рациональные точки и 2 точки степени 2. Выберем $r = 6$, и пусть $P_1, \dots, P_4$ – 4 рациональное точки, P_5, P_6 – 2 точки степени 2. Далее, возьмем $[n_i, k_i, d_i] = [1,1,1]$ для $1 \leq i \leq 4$ и $[n_i, k_i, d_i] = [3,2,2]$ для $i = 5,6$. Тогда для $m = \deg(G) = 1, \dots, 7$ получаем линейный $[n, k, d]$-код над $\mathbb{F}_2$ с параметрами

$$n = 10, k = m, d \geq 8 - m.$$

Линейные коды с $m = 2,3,4$ и $d = 8 - m$ являются оптимальными.

Пример. Пусть $q = 2$, и пусть $F = \mathbb{F}_2(x, y)$ – поле эллиптических функций, заданное

$$y^2 + y = x^3.$$

Тогда F имеет три рациональных точки и три точки степени 2. Выберем $r = 6$, и пусть $P_1, \dots, P_3$ – три рациональное точки и $P_4, \dots, P_6$ – три точки степени 2. Далее, возьмем

$$[n_i, k_i, d_i] = [1,1,1] \text{ для } 1 \le i \le 3 \text{ и } [n_i, k_i, d_i] = [3,2,2] \text{ для } 4 \le i \le 6.$$

Тогда для $m = \deg(G) = 1, \dots,8$ получаем линейный $[n, k, d]$-код над $\mathbb{F}_2$ с параметрами

$$n = 12, k = m, d \ge 9 - m.$$

Линейные коды с $m = 3,5$ и $d = 9 - m$ являются оптимальными.

2.2. Алгоритм декодирования Судана

Алгоритм Судана был изначально представлен как алгоритм декодирования для кодов Рида-Соломона. В дальнейшем совместными усилиями Шокроллахи, Вассермана и Судана было представлено расширение алгоритма декодирования для кодов Гоппы, а также некоторых других типов кодов. В данном разделе будет описан алгоритм только для алгеброгеометрических кодов.

Здесь и далее в разделе мы будем обозначать алгебраическое функциональное поле как

$$A = (\mathbb{F}_q, \overline{X}, X, K, g, ord).$$

$\mathbb{F}_q$ – конечное поле с q элементами, $\overline{\mathbb{F}_q}$ – его алгебраическое замыкание. $\overline{X}$ – множество точек (обычно в $\overline{\mathbb{F}_q^l}$). X – подмножество $\overline{X}$, элементы которого называются рациональными точками $\overline{X}$. K – множество функций из $\overline{X}$ в $\overline{\mathbb{F}_q} \cup \{\infty\}$, где ∞ обозначает неопределенное значение. $ord: K \times \overline{X} \to Z$, $ord(f, x)$ – порядок функции f в точке x. Род A обозначается g.

Точки (x_i, y_i) мы будем описывать с помощью многочлена Q, где Q – многочлен от y с коэффициентами из K (то есть $Q[y] \in K[y]$). Для заданного значения $y_i \in \mathbb{F}_q$, $Q[y_i]$ даст элемент в K. По определению такой элемент имеет значение в $x_i \in X$. Мы также потребуем, чтобы $Q(x_i, y_i) = Q[y_i](x_i)$ обращалось в ноль. Более того, мы потребуем, чтобы (x_i, y_i) были нулями кратности r для Q. Так как $x_i \in X$ и $y_i \in \mathbb{F}_q$, то определять условия для достижения этого

следует осторожно. Будем считать, что Q имеет маленький положительный порядок l в x_o для любой подстановки y функцией из K порядка не более $\alpha = k + g - 1$ в точке x_o. Найдя такой Q, необходимо посмотреть на его корни $h \in K$.

Теперь нужно определить условия, при которых (x_i, y_i) будут считаться нулями кратности r для многочлена Q для $1 \leq i \leq n$, а также $ord(Q[f], x_0) \leq l$ для любого $f \in \mathcal{L}_{\alpha,x_0}$, где l – параметр, который будет определен позже. Для этого предположим, что имеем явно заданные функции $\phi_1, \ldots, \phi_{l-g+1}$, такие, что $ord(\phi_j, x_0) \leq j + g - 1$, и такие, что $ord(\phi_j, x_0) < ord(\phi_{j+1}, x_0)$. Пусть $s = \lfloor \frac{l-g}{\alpha} \rfloor$. Теперь посмотрим на коэффициенты q_{j_1,j_2}, такие, что

$$Q[y] = \sum_{j_2=0}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} q_{j_1,j_2} \phi_{j_1} y^{j_2}.$$

Если явно задавать Q таким образом, то второе ограничение получено. Для первого ограничения необходимо сдвинуть наш базис. Рассмотрим две леммы.

Лемма 1. Для любого $f, g \in K$ и $x \in X$ существуют $\alpha_0 \neq 0, \beta_0 \neq 0 \in F_q$, такие, что

$$ord(\alpha_o f + \beta_0 g, x) < max\{ord(f, x), ord(g, x)\}.$$

Лемма 2. Для заданных функций $\phi_1, \ldots, \phi_p$ различного порядка в $x_0 \in X$, удовлетворяющих условию $\phi_j \in \mathcal{L}_{j+g-1,x_0}$, и рациональной точки $x_i \neq x_o$, существуют функции $\psi_1, \ldots, \psi_p \in K$ с $ord(\psi_j, x_i) \leq 1 - j$, такие, что существует $\alpha_{x_i,j_1,j_3} \in \mathbb{F}_q$ для $1 \leq j_1, j_3 \leq p$, такой, что $\phi_{j_1} = \sum_{j_3=1}^{p} \alpha_{x_i,j_1,j_3} \psi_{j_3}$.

Теперь мы можем вывести необходимое нам условие для нуля кратности не меньше r. Используя лемму выше и наш предыдущий результат, мы знаем, что

$$Q(x, y) = \sum_{j_2=0}^{s} \sum_{j_3=1}^{l-g+1l-g+1-\alpha j_2} \sum_{j_1=1} q_{j_1,j_2} \alpha_{x_i,j_1,j_3} \phi_{j_3,x_i}(x) y^{j_2}.$$

Сдвиг на y_i достигается определением $Q^{(i)}(x, y) = Q(x, y + y_i)$. Члены $Q^{(i)}(x, y)$, которые делятся на y^p, имеют кратность p в $(x_i, 0)$ как нули $Q^{(i)}$,

или, что эквивалентно, кратность (x_i, y_i) как нулей Q. Имеем следующее равенство

$$Q^{(i)}(x,y) = \sum_{j_4=0}^{s} \sum_{j_3=1}^{l-g+1} q_{j_3,j_4}^{(i)} \phi_{j_3,x_i}(x) y^{j_4},$$

где

$$q_{j_3,j_4}^{(i)} = \sum_{j_2=j_4}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} \binom{j_2}{j_4} y_i^{j_2-j_4} q_{j_1,j_2} \alpha_{x_i,j_1,j_3}.$$

Так как $ord(\psi_{j_3,x_i}, x_i) \leq -(j_3 - 1)$, то мы можем достичь нашего условия, требуя $q_{j_3,j_4}^{(i)} = 0$ для всех $j_3 \geq 1, j_4 \geq 0$, таких, что $j_4 + j_3 - 1 < r$.

Теперь опишем наш алгоритм формально. В нашем алгоритме мы будет обозначать $[n]$ как множество целых числе от 1 до n.

Параметры: $n; x_0, \ldots, x_n \in X; k; g;$

Предположения: Предполагается, что мы «знаем» функции $\{\phi_{j_1} \in K | j_1 \in [l - g + 1]\}$, где $[l - g + 1]$ – множество целых чисел от 1 до $l - g + 1$, различных порядков в x_0, у которых $ord(\phi_{j_1}, x_0) \leq j_1 + g - 1$, как и функции $\{\phi_{j_3,x_i} \in K | j_3 \in [l - g + 1], i \in [n]\}$, такие, что для любого $i \in [n]$ функции $\{\psi_{j_3,x_i}\}_{j_3}$ удовлетворяют неравенству$ord(\psi_{j_3,x_i}, x_i) \leq 1 - j_3$. Наше знание в данном случае означает, что мы предполагаем, что эти функции будут доступны в нашем алгоритме.

Множество $\{\alpha_{x_i,j_1,j_3} \in \mathbb{F}_q | j_1, j_3 \in [l - g + 1], i \in [n]\}$, такое, что для каждого $i, j_1, \phi_{j_1} = \sum_{j_3} \alpha_{x_i,j_1,j_3} \psi_{j_3,x_i}$. Это предположение достаточно разумно, так как по лемме выше мы можем провести все необходимые нам вычисления в функциональном поле K.

Полиномиальный алгоритм для нахождения корней в поле K многочленов в $K[y]$, где коэффициенты обозначены как формальная сумма ϕ_{j_1}.

Алгоритм:

Вход: $n; k; y_1, \ldots, y_n \in \mathbb{F}_q;$

Выход: исходное сообщение;

Параметры алгоритма: $r; l;$

$$r = 1 + \left\lfloor \frac{2gt + an + \sqrt{(2gt + an)^2 - 4(g^2 - 1)(t^2 - an)}}{2(t^2 - an)} \right\rfloor,$$

$l = rt - 1$.

Напомним, что $a = k + g - 1$.

Шаг 1: Найти $Q[y] \in K[y]$ вида $Q[y] = \sum_{j_2=0}^{s} \sum_{j_1=1}^{l-g+1-\alpha_{j_2}} q_{j_1,j_2} \alpha_{x_i,j_1,j_3} = 0$, то есть найти значения коэффициентов $\{q_{j_1,j_2}\}$, удовлетворяющих следующим условиям:

- По крайней мере, один q_{j_1,j_2} ненулевой.
- Для каждого $i \in [n]$, $\forall j_3, j_4, j_3 > 1, j_4 > 0$, таких, что $j_3 + j_4 \leq r$, мы имеем $q^{(i)}_{j_3,j_4} = \sum_{j_2=j_4}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} \binom{j_2}{j_4} y_i^{j_2-j_4} q_{j_1,j_2} \alpha_{x_i,j_1,j_3}$.

Этот шаг называется интерполяция. Существует несколько различных алгоритмов с разной сложностью.

Шаг 2: Найти все корни $h \in \mathcal{L}_{k+g-1,x_0}$ многочлена $Q \in K[y]$. Этот шаг называется факторизация. Мы можем использовать любой известный алгоритм факторизации для выполнения этого шага.

Шаг 3: Для полученных h проверить, что условие $h(x_i) = y_i$ верно для как минимум t значений $i \in [n]$, и если это выполняется, то включаем h в выходной список.

2.3. Построение алгоритма кодирования XNL-кода

Процесс кодирования сообщения $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{F}_q^k$ для XNL-кода длины n и размерности k происходит с помощью умножения вектора на порождающую матрицу кода. Порождающую матрицу мы будем строить с помощью двух матриц: внешней и внутренней.

Внешняя порождающая матрица будет иметь вид

$$G_{out} = \begin{bmatrix} f_1(P_1) & f_1(P_2) & \ldots & f_1(P_s) \\ f_2(P_1) & f_2(P_2) & \ldots & f_2(P_s) \\ \ldots & \ldots & \ldots & \ldots \\ f_k(P_1) & f_k(P_2) & \ldots & f_k(P_s) \end{bmatrix},$$

где f_i – элемент базиса $\mathcal{L}(kP_\infty)$, s – число точек. Таким образом, мы получим матрицу размера k на s.

Элементы матрицы лежат в F_{P_i}, и нам необходимо свести их к $\mathbb{F}_q$. Для этого элементы матрицы необходимо разложить по базису F_{P_i} над $\mathbb{F}_q$. Тогда мы

получим матрицу размера k на $m = \sum_{i=1}^{s} k_i$, где k_i – размерность базиса разложения F_{P_i} над $\mathbb{F}_q$ для каждой из точек. В итоге получается матрица

$$G_{out} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \dots & \dots & \dots & \dots \\ a_{k,1} & a_{k,2} & \dots & a_{k,m} \end{bmatrix},$$

где $a_{i,j} \in \mathbb{F}_q$.

Таким образом, мы получим внешнюю матрицу, которая будет соответствовать отображению

$$\mathcal{L}(G) \to \prod_{i=1}^{s} F_{P_i}.$$

Внутреннюю матрицу мы строим как блочно-диагональную матрицу из порождающих матриц внутренних кодов C_i, имеющую вид

$$G_{in} = \begin{bmatrix} C_1 & 0 & \dots & 0 \\ 0 & C_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & C_s \end{bmatrix},$$

где C_i – выбранные нами внутренние коды. Такая матрица будет иметь размеры $m = \sum_{i=1}^{s} k_i$ на n, где n – длина XNL-кода (сумма всех длин внутренних кодов). Эта матрица будет соответствовать отображению

$$\prod_{i=1}^{s} F_{P_i} \to \prod_{i=1}^{s} \mathrm{C_i}.$$

Таким образом, перемножив эти матрицы, мы получим искомую порождающую матрицу XNL-кода размера k на n и необходимое нам отображение $\beta: \mathcal{L}(G) \to \mathbb{F}_q^n$. То есть порождающая матрица равна

$$G = G_{out} \cdot G_{in}.$$

Кодирование сообщения происходит следующим образом

$$c = \alpha \cdot G,$$

где $\alpha \in \mathbb{F}_q^k$ и $c \in \mathbb{F}_q^n$. В итоге суть кодирования заключается в умножении сообщения на порождающую матрицу. Таким образом, мы получаем отображение

$$\mathcal{L}(G) \to \prod_{i=1}^{s} \mathrm{C_i},$$

которое соответствует определению XNL-кода.

Для того, чтобы найти проверочную матрицу кода, мы находим вектора w, такие, что $G \cdot w = 0$. Из этих векторов образуем матрицу.

С помощью проверочной матрицы мы можем найти минимальное расстояние кода. Если любые $l \leq d - 1$ столбцов проверочной матрицы линейно независимы, то минимальное расстояние кода не меньше d. Если при этом найдутся d линейно зависимых столбцов, то минимальное расстояние равно d.

2.4. Построение алгоритма декодирования XNL-кода

В данном разделе алгоритм Судана будет расширен для класса алгебро-геометрических кодов, называемых XNL-коды. Для начала введем основные понятия.

Пусть $Q \in F[Z]$, где Z – трансцендентно над F, а многочлен имеет следующий вид

$$Q = \sum_{j=0}^{degQ} q_j Z^j,$$

где $q_j \in F$. Значение многочлена определяется как

$$Q(f) = \sum_{j=0}^{degQ} q_i f^i, f \in F$$

и является элементом функционального поля F.

Для каждого P_i мы можем поставить в соответствие $z_r \dots z_{r+t}$, где r зависит от того, какая степень у точки P_i. Для точки степени 1 мы получим, что $r = i$, для точки степени 2 у нас будет $r = 2i$.

Для примера, если у нас есть 2 точки степени 1 и 2 точки степени 2, а длина кода равна 6 и получено сообщение $z = (z_1, \dots, z_6) | z_i \in \mathbb{F}_q$, то для точек степени 1 у нас получится, что P_1 соответствует z_1, P_2 соответствует z_2, P_3 соответствует $z_3 z_4$, P_4 соответствует $z_5 z_6$.

Имеем $z = (z_1, \dots, z_n) | z_i \in \mathbb{F}_q$. Если $q_j(P_i)$ определено для всех j, то можно определить значение Q для пары $(P_i; z_r \dots z_{r+t})$ как

$$Q(P_i; z_r \dots z_{r+t}) = \sum_{j=0}^{degQ} q_j(P_i) z_r^j \dots z_{r+t}^j,$$

которое является элементом $\mathbb{F}_q$. Пара $(P_i; z_r \dots z_{r+t})$ является нулем кратности $s_i \in N$ (множество натуральных чисел) многочлена Q, если s_i – максимальное число, такое, что для всех $f \in F$, которые удовлетворяют равенству $f(P_i) = z_r \dots z_{r+t}$, равенство $v_{P_i}(Q(f)) = s_i$ верно.

Предположим, что слово $z = (z_1, \dots, z_n) | z_i \in \mathbb{F}_q$ принимается и его необходимо декодировать. Идея алгоритма заключается в том, чтобы найти ненулевой многочлен $Q \in F[Z]$, такой, что пары $(P_i; z_r \dots z_{r+t})$, где $t + 1$ – степень точки P_i, являются нулями Q по крайней мере заданной кратности $s_i \in N$. Если

$$f(P_i) \cdot Matrix_{C_i} = z_r \dots z_{r+t},$$

причем значение $f(P_i)$ необходимо раскладывать по базису $\mathbb{F}_{q^t}$ над $\mathbb{F}_q$, а $Matrix_{C_i}$ – порождающая матрица кода C_i, для достаточно большого числа i, то можно сделать вывод, что Q имеет достаточно много нулей для выбранной нами степени и коэффициентов Q и, следовательно, $Q(f) = 0$. Это означает, что $Z - f$ является делителем многочлена Q, и функции, соответствующие кодовым словам, ближайшим к z, найдены.

Данный алгоритм является обобщением алгоритма Судана для АГ-кодов с точками нескольких степеней, а также для XNL-кодов. Мы можем определить параметр $e \in N$ по границе исправляемых ошибок.

Алгоритм:

Вход: $n; P_i, s_i, 1 \le i \le n; z;$

Выход: исправленное кодовое слово;

Шаг 1: Интерполяция. Найти ненулевой многочлен $Q = \sum_{j=0}^{degQ} q_j Z^j \in F[Z]$, такой, что $(P_i; z_r .. z_{r+t})$ являются нулями кратности $\geq s_i \forall i = 1, \ldots, n$, $t+1$ – степень точки.

Шаг 2: Факторизация. Найти $f \in \mathcal{L}(G)$, такие, что $(Z - f)|Q$.

Шаг 3: Вернуть массив функций $f \in \mathcal{L}(G)$, полученных на шаге 2.

Шаг 4: Далее вычислить векторы $(f(P_1) \cdot Matrix_{C_1}, \ldots, f(P_k) \cdot Matrix_{C_k})$ для всех полученных $f \in \mathcal{L}(G)$.

Шаг 5: Вернуть из полученных кодовых слов те, что различаются с исходным вектором не более чем на $n - \sqrt{degG \cdot n}$ позиций.

Два основных шага алгоритма – это интерполяция и факторизация. Для них можно использовать любые существующие алгоритмы для алгеброгеометрических кодов, так как изменения достаточно небольшие. Таких алгоритмов достаточно много, и каждый имеет свою сложность.

Также можно сделать вывод о количестве исправляемых ошибок для XNL-кода. Пусть C – это $[n, k, d]$ XNL-код над функциональным полем F рода g. Тогда существует полиномиальный списочный алгоритм декодирования для XNL-кода C, который исправляет

$$e < n - \sqrt{degG \cdot n}$$

ошибок.

Сравним количество исправляемых ошибок у алгоритма Судана для алгеброгеометрических кодов и у нашей адаптации алгоритма Судана для XNL-кодов. Классический алгоритм Судана может исправить

$$e < n - \sqrt{n(k+g-1)} = n - \sqrt{n(n-d)} \text{ [5].}$$

Как можно видеть, в классическом алгоритме мы имеем в корне $k + g - 1$, а в нашей адаптации $degG$. Так как мы разрабатываем алгоритм для одноточечных кодов, то есть для случая, когда дивизор $G = kP_0$, где P_0 – выбранная нами точка, то $degG = k$. Таким образом, для кривой рода 1 мы получаем равное количество исправляемых ошибок, тогда как для кривых рода больше 1 мы получаем меньшее количество исправляемых ошибок.

2.5. Оценка сложности алгоритма декодирования

Основная сложность алгоритма декодирования Гурусвами-Судана как для АГ-кодов, так и для XNL-кода, заключена в первом и втором шаге – интерполяция и факторизация. Их сложность зависит от выбранных алгоритмов. Для расчета совокупной сложности алгоритма декодирования для XNL-кода мы выберем интерполяцию и факторизацию, изложенные в статье [15], а также будем использовать их в программном комплексе.

Сложность представленных в статье алгоритмов равна

$$\tilde{O}(n^{4/3}l^2s) \text{ [15]},$$

где s – кратность нулей, l – параметр алгоритма декодирования, $\tilde{O}$ обозначает O с опущенными логарифмами.

Шаг 3 нашего алгоритма не имеет какой-либо сложности, а вот шаг 4 требует вычислений. Вычисление функции в точке имеет сложность $O(k)$, умножение значения этой функции на порождающую матрицу кода C_i для каждой из этих точек будет равно $O(k_i n_i)$ для кода C_i. В итоге мы получаем следующую сложность для шага 3:

$$O(k^2) + \sum_{1}^{k} O(k_i n_i) = O(k^2 n).$$

Шаг 5 является обычным пробегом по каждому элементу векторов, а значит, имеет сложность $O(n)$.

В итоге мы имеем суммарную сложность для алгоритма декодирования в следующем виде:

$$\tilde{O}(n^{4/3}l^2s) + O(k^2n) + O(n) = \tilde{O}(n^{4/3}l^2s) + O(k^2n).$$

Из полученных нами результатов можно сделать вывод, что существует полиномиальный списочный алгоритм декодирования для XNL-кода C.

Для классического алгоритма Судана мы будем иметь точно такую же сложность [5], если будем использовать те же самые алгоритмы интерполяции и факторизации.

Также мы сравним полученные минимальные расстояния для построенных нами XNL-кодов с минимальными расстояниями эквивалентных кодов Рида-Соломона (под эквивалентностью мы подразумеваем одинаковые длины и размерности кодов). Также сравним полученные результаты с предусмотренными конструкцией XNL-кода ограничениями.

n	k	Минимальное расстояние XNL-кода (не менее)	Минимальное расстояние XNL-кода	Минимальное расстояние кода РС
10	7	1	2	4
13	7	3	3	7
22	14	1	4	9
33	15	7	9	19
55	30	6	7	26
80	50	2	7	31
109	66	1	8	36

Таким образом, мы выяснили экспериментальным путем, что построенная нами конструкция кодирования кода является верной, так как минимальное расстояние (четвертый столбец) превышает или равно ограничениям конструкции кода (третий столбец). Однако, достичь минимального расстояния кодов Рида-Соломона у нас не получилось.

3. Описание программного комплекса

3.1. Архитектура программного комплекса

Для разработки программного комплекса необходимо выбрать среду программирования. Существует несколько различных систем компьютерной алгебры: SINGULAR, Maple, Magma, Sage. Так как область исследования данной работы – теория кодирования, то мы можем использовать Sage и Magma. В этих системах наиболее широко представлена теория кодирования. Также мы рассмотрим другие системы, такие как Maple, Wolfram Alpha и PARI/GP.

Рассмотрим систему компьютерной алгебры Maple. В ней присутствует множество различных библиотек для работы с кривыми. Также эта система обладает очень большой скоростью вычисления и хорошей документацией. Однако она является платной, и в ней отсутствует библиотека для работы с линейными кодами.

В системе Magma, напротив, присутствует библиотека для работы с теорией кодирования. Скорость вычислений в ней тоже на достаточно высоком уровне. Но одной из целей при разработке программного комплекса является объектно-ориентированный подход для возможности расширения программы. Magma предлагает процедурный подход.

При исследовании PARI/GP было обнаружено, что, несмотря на богатый набор пакетов, данная система не обладает возможностью работы с теорией кодирования.

Wolfram Alpha является ограниченно бесплатной онлайн программой. Только на платной версии предлагается весь спектр возможностей, тогда как в бесплатной присутствует ограничение на производительность и набор библиотек. Также система не предоставляет возможность работы с объектно-ориентированной парадигмой.

Sage является свободно распространяемой системой (комплексом различных программ и библиотек) с возможностью писать алгоритмы и программы на языке программирования Python. Это позволяет нам работать с классами в стиле объектно-ориентированного программирования (под словом класс мы подразумеваем одну из ключевых возможностей объектно-ориентированного программирования), манипулировать классом с помощью его методов.

В составе Sage есть целый блок алгоритмов, относящихся к теории кодирования. Sage умеет строить множество различных кодов – коды Рида-

Соломона и обобщенные коды Рида-Соломона, БЧХ-коды, коды Хэмминга и другие.

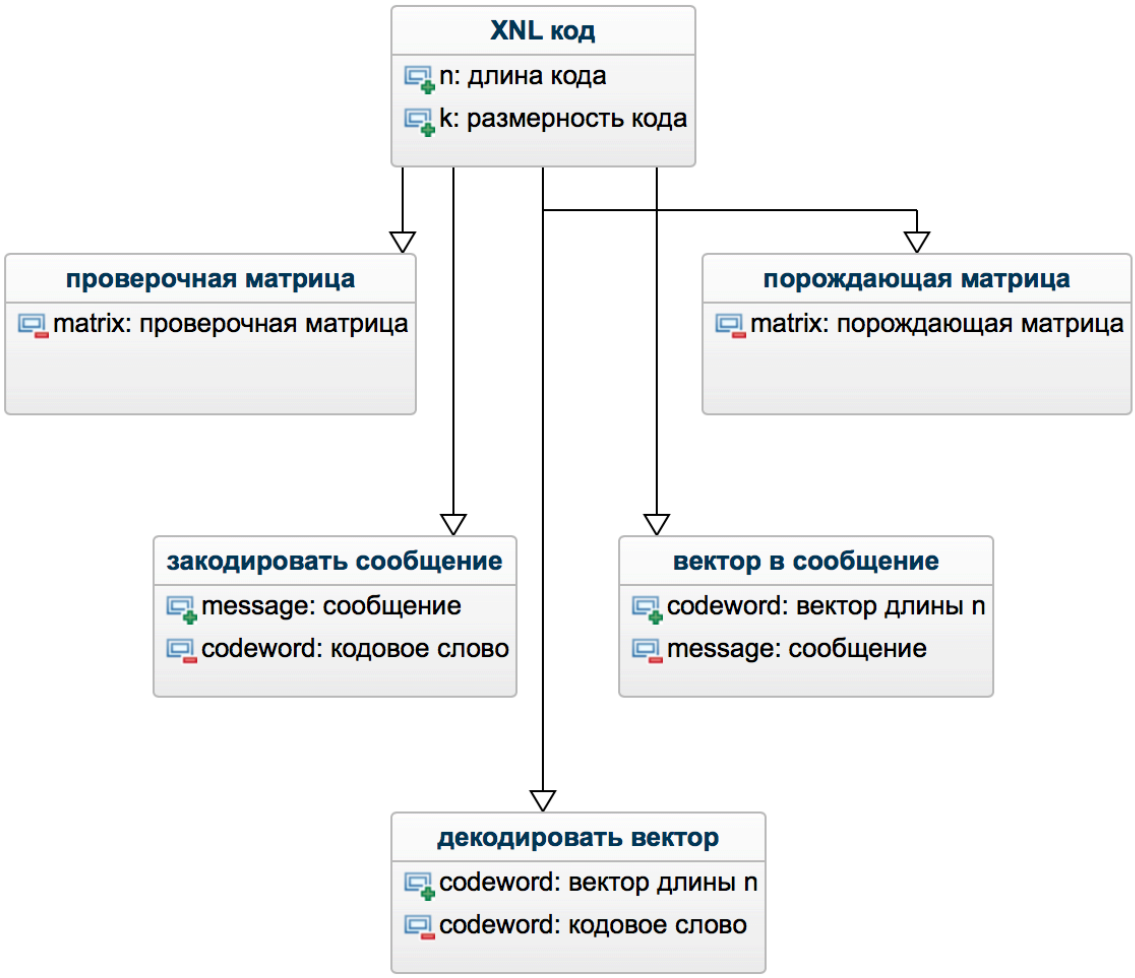
В итоге самым лучшим выбором в нашей ситуации будет система компьютерной алгебры Sage.

В данной работе будет представлен код, который может дополнить библиотеку кодов системы компьютерной алгебры Sage.

Программный комплекс разработан таким образом, чтобы его мог расширить или модифицировать любой желающий. Вся основа выполняется во время инициализации объекта класса XNL-кода. В программу можно легко добавить новые методы. Также легко можно унаследовать данный класс для расширения его работы с определенными кривыми.

3.2. Описание связей и работы программы

Так как программа написана с применением объектно-ориентированного подхода, то изобразим классовую диаграмму.



В программе используется один класс – класс XNL-кода. На схеме показаны стрелками основные методы класса. В параметрах знаком плюс обозначаются параметры на вход, знаком минус – выходное значение. Класс XNL-кода инициализируется двумя параметрами – длиной кода и его размерностью. Далее программа сама подбирает необходимые для этого внутренние параметры (выбирает точки нужных степеней и их количество для заданной длины, рассчитывает базис пространства Римана – Роха и минимальное расстояние кода). Инициализация требует некоторого времени для расчета всех параметров и подготовки к дальнейшей работе.

Методы «проверочная матрица» и «порождающая матрица» выводят соответствующие матрицы инициализированного кода. Эти матрицы генерируются на стадии инициализации кода, так что методы просто возвращают значение заранее рассчитанного параметра.

Метод «закодировать сообщение» принимает на вход вектор (сообщение), который соответствует размерности кода, и преобразует его в кодовое слово XNL-кода. Если длина этого вектора не соответствует размерности кода, программа выдаст ошибку.

Метод «вектор в сообщение» производит обратное действие – а именно переводит вектор (или кодовое слово XNL-кода) в исходное сообщение-вектор. Если входящий вектор не является кодовым словом, то есть не лежит в нашем коде, то программа выведет ошибку и попросит сначала исправить ошибки в векторе. Это делает следующий метод.

Остался последний и самый важный метод под названием «декодировать вектор». На вход подается вектор для исправление возможных ошибок. На выходе мы получаем кодовое слово. Здесь используется разработанный в данной работе алгоритм декодирования.

4. Примеры

4.1. Пример

В качестве примера возьмем функциональное поле $F = \mathbb{F}_4(x, y)$, определенное эллиптической кривой $y^2 + y = x^3 + x$. Род кривой равен $g = 1$. Необходимо построить дзета-функцию

$$Z(T) = \frac{L(T)}{(1-T)(1-qT)},$$

где $L(T)$ – L-многочлен степени $2g$ с целыми коэффициентами.

Построим поле $\mathbb{F}_4$ с помощью многочлена $x^2 + x + 1$. Мы получим элементы $\{0, a, a + 1, 1\}$, где a – корень многочлена $x^2 + x + 1$.

Для данной кривой мы имеем 5 рациональных точек: $[0:0:1]$, $[0:1:1]$, $[1:0:1]$, $[1:1:1]$, $[0:1:0]$. Количество точек степени r будем обозначать как B_r. Для построения L-многочлена нам необходимо вычислить его коэффициенты $a_{2g-i} = q^{g-i}a_i$. Мы получим $a_0 = 1, a_1 = 0, a_2 = 4$. Таким образом, $L(T) = 4t^2 + 1$ [10]. В итоге получаем дзета-функцию

$$Z(T) = \frac{4T^2+1}{(1-T)(1-4T)}.$$

Обозначим $N_i(F) = N_i$ – количество рациональных точек над $\mathbb{F}_{4^i}$. Известно что $N_n(F) = \sum_{d|n} dB_d$. Для нашей кривой имеем

$$N_1 = 5, N_2 = 25, N_3 = 65, N_4 = 225.$$

Отсюда можем получить $B_1 = 5, B_2 = 10, B_3 = 20, B_4 = 50$ – количество точек степени 1, 2, 3, 4 соответственно.

Для построения XNL-кода возьмем 5 точек степени 1, 10 точек степени 2 и 20 точек степени 3. В качестве внутренних кодов будем использовать следующие коды над $\mathbb{F}_4$: [1,1,1], [3,2,2] и [5,3,3]. Их порождающие матрицы равны

$$[1 \quad 1 \quad 0], \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \text{ и } \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & \alpha & \alpha^2 & 0 \\ 0 & 1 & \alpha^2 & \alpha & 1 \end{bmatrix}$$

соответственно. В итоге получаем XNL-код

$$[k, 3; 5,10,20; 1,2,3],$$

где $1 \leq k \leq 84$, с параметрами

$$[130, k, d \geq 85 - k].\ [10]$$

4.2. Пример

Оставим все параметры как в примере 1 и построим еще один XNL-код, для которого возьмем более простые параметры для демонстрации всех вычислений – а именно четыре точки степени 1 и две точки степени 2. В качестве внутренних кодов будем использовать коды над $\mathbb{F}_4$ из предыдущего XNL-кода. Мы получим XNL-код с параметрами

$$[10; k; d \geq 8 - k],$$

где $1 \leq k \leq 7$. Далее мы будем использовать этот XNL-код.

Пусть у нас $k = 7$. Тогда параметры будут

$$[10; 7; d \geq 1].$$

Для данной кривой множество

$$\{x^i y^j : i \geq 0, 0 \leq j \leq 1, 2i + 3j \leq k\}$$

является базисом пространства Римана-Роха $\mathcal{L}(kP_\infty)$. В качестве точек степени 1 возьмем точки

$$\{(0,0), (0,1), (1,0), (1,1)\},$$

для степени 2 возьмем

$$\{(b^2 + b, b^2), (b^2 + b + 1, b)\}.$$

Получаем базис пространства Римана-Роха

$$\{1, x, xy, x^2y, y, x^3, x^2\}$$

для $\mathcal{L}(7P_\infty)$. Для получения порождающей матрицы кода мы будем использовать произведение матрицы внешней на матрицу внутреннюю, где внешняя задается с помощью базиса пространства Римана-Роха, а внутренняя строится с помощью порождающих матриц внутренних кодов [1,1,1], [3,2,2], [5,3,3].

В нашем примере внешняя порождающая матрица равна

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & a+1 & a & a \\ 0 & 0 & 0 & 1 & a+1 & a & 0 & a \\ 0 & 1 & 0 & 1 & a & 1 & a & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Внутренняя матрица равна

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

В итоге для нашего примера мы получаем общую порождающую матрицу

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & a & a+1 & a & 0 & a \\ 0 & 0 & 0 & 1 & a+1 & 1 & a & 0 & a & a \\ 0 & 1 & 0 & 1 & a & a+1 & 1 & a & a & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Для кодирования сообщения

$$(1,0,1,0,1,0,1)$$

нам необходимо умножить его на итоговую (общую) порождающую матрицу. Получим вектор

$$(1,0,0,0,a+1,0,a+1,1,a,a+1).$$

То есть

$$(1,0,1,0,1,0,1)\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0\\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1\\ 0 & 0 & 0 & 1 & 1 & a & a+1 & a & 0 & a\\ 0 & 0 & 0 & 1 & a+1 & 1 & a & 0 & a & a\\ 0 & 1 & 0 & 1 & a & a+1 & 1 & a & a & 0\\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0\\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1\end{bmatrix} =$$

$$= (1,0,0,0,a+1,0,a+1,1,a,a+1).$$

Добавим вектор ошибки

$$(0,1,0,0,0,0,0,0,0,0)$$

к кодовому слову и получим вектор

$$(1,0,0,0,a+1,0,a+1,1,a,a+1)+(0,1,0,0,0,0,0,0,0,0) =$$

$$= (1,1,0,0,a+1,0,a+1,1,a,a+1).$$

Используя алгоритмы интерполяции и факторизации (шаг 1 и шаг 2), мы получаем два многочлена от двух переменных

$$x^2y+x^2+y+1,\ x^2y+x^2+y+x+1.$$

Далее подсчитаем первый многочлен в точке и умножим на соответствующие порождающие матрицы

$$f = x^2y+x^2+y+1$$

$$\left(f(P_1)\cdot[1\quad 1\quad 0], f(P_2)\cdot[1\quad 1\quad 0], \dots, f(P_6)\cdot\begin{bmatrix}1 & 1 & 0\\ 0 & 1 & 1\end{bmatrix}\right) =$$

$$= (1,0,0,0, a+1,0, a+1,1, a, a+1)$$

Аналогично для второго многочлена

$$f = x^2y + x^2 + y + x + 1$$

$$\left(f(P_1)\cdot [1 \quad 1 \quad 0], f(P_2)\cdot [1 \quad 1 \quad 0], \dots, f(P_6)\cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}\right) =$$

$$= (1,0,1,1,1, a, a+1,0, a, a)$$

Теперь нам необходимо подсчитать количество исправляемых ошибок алгоритмом декодирования

$$e < n - \sqrt{degG \cdot n} = 10 - \sqrt{70} = 1{,}6.$$

То есть мы можем исправить одну ошибку. Сравнив полученные кодовые слова, мы делаем вывод, что вектор

$$(1,0,0,0, a+1,0, a+1,1, a, a+1)$$

как раз отличается на 1 элемент. Следовательно, мы исправили ошибку и нашли восстановили кодовое слово.

4.3. Пример

В этом примере мы тоже оставим все параметры как в примере 1 и построим еще один XNL-код, но с более сложными параметрами. Возьмем четыре точки степени 1 и две точки степени 2. В качестве внутренних кодов будем использовать те же самые коды. Мы получим XNL-код с параметрами

$$[25; k; d \geq 18 - k].$$

Пусть у нас $k = 15$. Тогда параметры будут

$$[25; 15; d \geq 3].$$

В качестве точек степени 1 возьмем точки

$$\{(0,0),(0,1),(1,0),(1,1)\},$$

а в качестве точек степени 2 возьмем

$$\{(b^2+b,b^2),(b^2+b+1,b),(b^3,b^3+b),(b^3+1,b),$$

$$(b^3+b,b^3+b^2+b),(b^3+b+1,b^2),(b^3+b^2,b^3)\}.$$

Получаем базис пространства Римана-Роха

$$\{1,x,x^2,x^3,x^4,x^5,x^6,x^7,y,xy,x^2y,x^3y,x^4y,x^5y,x^6y\}.$$

В нашем примере внешняя порождающая, внутренняя и общая порождающая матрицы равны

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & a & 0 & a+1 & 0 & a & a+1 & a+1 & a+1 & a & a & a+1 & a & 0 & a \\
0 & 0 & 1 & 1 & a+1 & 0 & a & 0 & 0 & a & 1 & a & a & a+1 & a+1 & a+1 & 1 & a+1 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & a & a & 1 & a+1 & 1 & a+1 & 0 & a & a & a+1 \\
0 & 0 & 1 & 1 & a & 0 & a+1 & 0 & 1 & a+1 & 0 & a+1 & 0 & a & 1 & a & a & a \\
0 & 0 & 1 & 1 & a+1 & 0 & a & 0 & 1 & 0 & a+1 & 0 & 1 & 0 & a & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & a & a+1 & a & a & a & a & 1 & a+1 & 0 & a \\
0 & 0 & 1 & 1 & a & 0 & a+1 & 0 & 0 & a & a+1 & 1 & a & a+1 & 1 & 1 & 1 & a+1 \\
0 & 1 & 0 & 1 & a & 1 & 0 & 1 & a & a & 0 & 1 & 0 & a+1 & a & 1 & a & a+1 \\
0 & 0 & 0 & 1 & a+1 & a & 0 & a+1 & 1 & a+1 & 1 & 0 & a & 0 & a & a & a & a \\
0 & 0 & 0 & 1 & 1 & a+1 & 0 & a & 1 & 0 & a+1 & a+1 & a+1 & a+1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & a & 1 & 0 & 1 & a & a+1 & 1 & a & a+1 & 1 & a+1 & 1 & 0 & a \\
0 & 0 & 0 & 1 & a+1 & a & 0 & a+1 & 0 & a & 1 & a+1 & a & 1 & 1 & 0 & 1 & a+1 \\
0 & 0 & 0 & 1 & 1 & a+1 & 0 & a & a & a & 0 & a+1 & 0 & a+1 & a+1 & a & a & a+1 \\
0 & 0 & 0 & 1 & a & 1 & 0 & 1 & 1 & a+1 & a+1 & 0 & a & 0 & a+1 & a+1 & a & a
\end{bmatrix}$$

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

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & a & a & 0 & a+1 & a+1 & 0 & a & 1 & a+1 & a+1 & 0 & a+1 & a & 0 & a & a+1 & 1 & a & 0 & a & a \\
0 & 0 & 1 & 1 & a+1 & a+1 & 0 & a & a & 0 & 0 & a & a & 1 & a+1 & a & a & 1 & a+1 & a+1 & 0 & a+1 & 1 & a & a+1 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & a & 0 & a & 1 & a & a+1 & 1 & a & a+1 & 0 & a & a & a & 1 & a+1 \\
0 & 0 & 1 & 1 & a & a & 0 & a+1 & a+1 & 0 & 1 & a & a+1 & 0 & a+1 & a+1 & 0 & a & a & 1 & a+1 & a & a & 0 & a \\
0 & 0 & 1 & 1 & a+1 & a+1 & 0 & a & a & 0 & 1 & 1 & 0 & a+1 & a+1 & 0 & 1 & 1 & 0 & a & a & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & a & 1 & a+1 & a & 0 & a & a & 0 & a & 1 & a & a+1 & 0 & a & a \\
0 & 0 & 1 & 1 & a & a & 0 & a+1 & a+1 & 0 & 0 & a & a & a+1 & a & 1 & a & 1 & a+1 & 1 & 0 & 1 & 1 & a & a+1 \\
0 & 1 & 0 & 1 & a & a+1 & 1 & 0 & 1 & 1 & a & 0 & a & 0 & 1 & 1 & 0 & a+1 & a+1 & a & a+1 & 1 & a & 1 & a+1 \\
0 & 0 & 0 & 1 & a+1 & 1 & a & 0 & a+1 & a+1 & 1 & a & a+1 & 1 & 1 & 0 & a & a & 0 & a & 0 & a & a & 0 & a \\
0 & 0 & 0 & 1 & 1 & a & a+1 & 0 & a & a & 1 & 1 & 0 & a+1 & 0 & a+1 & a+1 & 0 & a+1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & a & a+1 & 1 & 0 & 1 & 1 & a & 1 & a+1 & 1 & a+1 & a & a+1 & a & 1 & a+1 & a & 1 & 0 & a & a \\
0 & 0 & 0 & 1 & a+1 & 1 & a & 0 & a+1 & a+1 & 0 & a & a & 1 & a & a+1 & a & a+1 & 1 & 1 & 1 & 0 & 1 & a & a+1 \\
0 & 0 & 0 & 1 & 1 & a & a+1 & 0 & a & a & a & 0 & a & 0 & a+1 & a+1 & 0 & a+1 & a+1 & a+1 & 1 & a & a & 1 & a+1 \\
0 & 0 & 0 & 1 & a & a+1 & 1 & 0 & 1 & 1 & 1 & a & a+1 & a+1 & a+1 & 0 & a & a & 0 & a+1 & 0 & a+1 & a & 0 & a
\end{bmatrix}$$

Для кодирования сообщения $(1,0,1,0,1,0,1,1,0,1,0,1,0,1,a)$ нам необходимо умножить его на итоговую (общую) порождающую матрицу. Получим вектор

$$(1,1,1,a,0,a,a,a,0,a,1,a,a+1,0,a+1,a+1,1,a+1,a,0,a+1,a+1,0,0,0).$$

Декодирование осуществляется как в примере 4.2.

Заключение

В данной дипломной работе мы провели обзор предварительный сведений, исследовали конструкции NXL- и XNL-кодов, рассмотрели алгоритм декодирования Гурусвами-Судана, обобщили его для XNL-кода, а также реализовали программной комплекс для данного класса кодов с возможностью кодировать и декодировать информацию.

В адаптированном для исследуемого нами кода алгоритме декодирования мы подсчитали сложность, которая оказалась такой же, как и в исходном алгоритме. В нашем алгоритме она равна

$$\tilde{O}(n^{4/3}l^2s) + O(k^2n) + O(n) = \tilde{O}(n^{4/3}l^2s) + O(k^2n).$$

Также мы исследовали свойства алгоритма и вывели количество исправляемых ошибок. Для кривой степени 1 эта граница эквивалентна границе классического алгоритма. В итоге мы можем исправить следующее количество ошибок:

$$e < n - \sqrt{degG \cdot n}.$$

Работа алгоритмов кодирования и декодирования была показана на трех различных примерах, а также предоставлен код программного комплекса, реализованного с возможностью дальнейшего расширения и возможностью его использовать в системе компьютерной алгебры SAGE.

В итоге основная цель работы была выполнена и алгоритм декодирования для XNL-кодов, к которым можно свести и NXL-коды, разработан.

В дальнейшем можно улучшить и оптимизировать работу программы, в частности можно ускорить работу алгоритма декодирования и генерацию необходимых матриц. Также можно разработать отдельный программный комплекс для NXL-кодов, которые являются подклассом XNL-кодов. Так как при одинаковых длине и размерности минимальное расстояние кода у нас получилось меньше, чем для кода Рида – Соломона, то можно провести анализ минимального расстояния XNL-кода для других кривых.

Литература

1. **В. Д. Гоппа.** Алгебраико-геометрические коды – Изв. АН СССР. Сер. матем., 46:4 (1982), 762-781
2. **H. Niederreiter, C. P. Xing, K. Y. Lam.** A new construction of algebraic-geometry codes. – Applicable Algebra Engrg. Comm. Comput. 9, 373-381 – 1999.
3. **C. P. Xing, H. Niederreiter, K. Y. Lam.** A generalization of algebraic-geometry codes. – IEEE Trans. Inform. Theory 45, 2498-2501 – 1999.
4. **H. Niederreiter, C. P. Xing.** Algebraic Geometry in Coding Theory and Cryptography. – Princeton : Princeton University Press – 2009.
5. **V. Guruswami, M. Sudan.** Improved decoding of Reed-Solomon and algebraic-geometry codes – IEEE Trans. Inform. Theory 45, 1757-1767 – 1999.
6. **Р. Хартсхорн.** Алгебраическая геометрия – М.: Мир – 1981.
7. **Vadim Olshevski, M. Amin Shokrollahi.** A Displacement Approach to Decoding Algebraic Codes. – 2000.
8. **T. Hoholdt, R. Pellicaan.** On the decoding of algebraic geometric codes. – IEEE Trans. Inform. Theory 41, 1589-1614 – 1995.
9. **R. Kotter.** Fast generalized minimum distance decoding of algebraic geometry and Reed Solomon codes. – IEEE Trans. Inform. Theory 42, 721-737 – 1996.
10. **C. Ding, H. Niederreiter, C. Xing.** Some new codes from algebraic curves. – IEEE Trans. Inform. Theory 46, 2638–2642 – 2000.
11. **T. Hoholdt, R.R. Nielsen.** Decoding Hermitian codes with Sudan's algorithm. – in: M. Fossorier, H. Imai, S. Lin, A. Poli (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. Lecture Notes in Computer Science, Springer, Berlin, 1999, pp. 260–270.
12. **H. Stichtenoth.** Algebraic Function Fields and Codes. – Springer, Berlin – 2009.
13. **R. Roth and G. Ruckenstein.** Efficient decoding of Reed-Solomon codes beyond half the minimum distance. – IEEE Trans. Inform. Theory 46, 246-257 – 2000.
14. **X. W. Wu and P. Siegel.** Efficient root-finding algorithm with application to list decoding of algebraic-geometric codes. – IEEE Trans. Inform. Theory 47, 2579-2587 – 2001.
15. **Johan S. R. Nielsen, Peter Beelen.** Sub-quadratic Decoding of One-point Hermitian Codes. – IEEE Trans. Inform. Theory 61, 3225-3240 – 2015.

16. **Anna Aarstrand Slaatsveen.** Decoding of Algebraic Geometry Codes. – Norwegian University of Science and Technology – 2011.
17. **Nathan Drake.** Decoding Of Multipoint Algebraic Geometry Codes Via Lists.– Clemson University – 2009.

Приложение

Программный комплекс для XNL-кода

```
from sage.coding.relative_finite_field_extension import *
class XNLCode():
    def __init__(self, n, k):
        self.n = n
        self.k = k
        # creating of XNL [n,k,d] linear code over F_q
        # you should choose points count, curve, field and inner codes for code here
        A.<x,y> = AffineSpace(GF(2), 2)
        self.curve = y^2 + y - x^3 - x # curve
        self.q = 4 # base F_q for our code

        Fq4.<a> = GF(self.q, 'a') # Finite Field
        self.a = a
        self.Fq4 = Fq4
        self.F = Fq4
        self.Ya = PolynomialRing(self.F,'x,y',order=TermOrder('wdeglex',(self.q,self.q+1)))
        Fq16.<b>, f = Fq4.extension(2, 'b', map=True)
        self.b = b
        self.Fq16 = Fq16
        Fq64.<c>, f = Fq4.extension(3, 'c', map=True)
        self.c = c
        self.Fq64 = Fq64

        self.func_field_points = [4, 7, 0] # x points of degree 1, y points of degree 2, ...
        self.inner_codes = [[1, 1, 1], [3, 2, 2], [5, 3, 3]] # inner codes for points of degree 1, degree 2, etc
        self.inner_codes_generator = [Matrix(Fq4,[1]), Matrix(Fq4, [[1,1,0],[0,1,1]])] # generator matrices for inner codes
        # service calculations
        self.genus = Curve([self.curve], A).genus() # curve genus
        # calculating minimum distance of XNL code
```

```
        self.d = self._get_d()
        k_upper_bound = self._get_k_upper_bound()

        assert self.k <= k_upper_bound and self.k >= 1, 'parameter k (dimension) of XNL code must be in 1 to %s but it %s' %(k_upper_bound, self.k)

        self.G = self._create_generator_matrix(self.get_points())

    def __repr__(self):
        return '[%d, %d, d >= %d] linear XNL code over GF(%d) on curve %s with genus %s' % (
                self.n,
                self.k,
                self.d,
                self.q,
                self.curve,
                self.genus
                )

    def __eq__(self, other):
        """Very superficial, but relatively cheap, check for total equivalence of two codes: their generator matrices are exactly the same."""
        return self.generator_matrix() == other.generator_matrix()

    def __contains__(self, x):
        return self.iscodeword(x)

    @cached_method
    def _get_k_upper_bound(self):
        return sum(self.func_field_points[i] * self.inner_codes[i][2] for i in [0..min(len(self.func_field_points), len(self.inner_codes)) - 1]) - self.genus

    @cached_method
    def _get_d(self):
        return sum(self.func_field_points[i] * self.inner_codes[i][2] for i in [0..min(len(self.func_field_points), len(self.inner_codes)) - 1]) - self.k - self.genus + 1
```

```
    @cached_method
    def get_points(self):
        # calculating points of degree 1
        Aff4.<x,y> = AffineSpace(self.Fq4, 2)
        C = Curve(y^2 + y - x^3 - x)
        pts1 = C.rational_points()
        # calculating points of degree 2
        Aff16.<x,y> = AffineSpace(self.Fq16, 2)
        C = Curve(y^2 + y - x^3 - x)
        pts2 = C.rational_points()
        # calculating points of degree 3
        Aff64.<x,y> = AffineSpace(self.Fq64, 2)
        C = Curve(y^2 + y - x^3 - x)
        pts3 = C.rational_points()

        points = []
        for i in range(0,self.func_field_points[0]):
            points.append(pts2[i])
        for i in range(0,self.func_field_points[1]*2, 2):
            points.append(pts2[i + len(pts1)])
        for i in range(0,self.func_field_points[2]*3, 3):
            points.append(pts3[i + len(pts2) + len(pts1)])
        return points

    def _create_inner_matrix(self):
        inner_matrix_dim = sum(self.func_field_points[i] * self.inner_codes[i][1] for i
in [0..min(len(self.func_field_points), len(self.inner_codes)) - 1])
        inner_matrix_length = sum(self.func_field_points[i] * self.inner_codes[i][0] for
i in [0..min(len(self.func_field_points), len(self.inner_codes)) - 1])
        inner_matrix = matrix(self.Fq4, inner_matrix_dim, inner_matrix_length);
        for i in range(0, self.inner_codes[0][1]*self.func_field_points[0]):
            for j in range(0, self.inner_codes[0][0]*self.func_field_points[0]):
                if(i == j):
                    inner_matrix[i,j] = self.inner_codes_generator[0][0][0]
        j = self.inner_codes[0][0]*self.func_field_points[0]
```

```
        for i in range(self.inner_codes[0][1]*self.func_field_points[0],
self.inner_codes[0][1]*self.func_field_points[0] +
self.inner_codes[1][1]*self.func_field_points[1], self.inner_codes[1][1]):
            for l in range(0, self.inner_codes[1][0]):
                for m in range(0, self.inner_codes[1][1]):
                    inner_matrix[i+m,j+l] = self.inner_codes_generator[1][m][l]
            j = j + self.inner_codes[1][0]
        return inner_matrix

    @cached_method
    def _get_riemann_roch_basis(self, field = 0):
        if(field == 0):
            field = self.Fq16
        Aff.<x,y> = AffineSpace(field, 2)
        riemann_roch_basis = []
        for j in range(0,2):
            i = 0
            while(2*i + 3*j <= self.k):
                riemann_roch_basis.append(x^i*y^j)
                i = i + 1
        return riemann_roch_basis

    def _create_outer_matrix(self, points):
        F16_to_F4 = RelativeFiniteFieldExtension(self.Fq16, self.Fq4)
        F64_to_F4 = RelativeFiniteFieldExtension(self.Fq64, self.Fq4)
        riemann_roch_basis = self._get_riemann_roch_basis()
        gen_matrix_F16 = matrix(self.Fq16, self.k, len(points));
        for i in range(0, self.k):
            for j in range(0, len(points)):
                gen_matrix_F16[i, j] = riemann_roch_basis[i](points[j][0],points[j][1])

        outer_matrix_dim = self.k
        outer_matrix_length = sum(self.func_field_points[i] * self.inner_codes[i][1] for
i in [0..min(len(self.func_field_points), len(self.inner_codes)) - 1])
        outer_matrix = matrix(self.Fq4, outer_matrix_dim, outer_matrix_length)
        for i in range(0, outer_matrix_dim):
            for j in range(0, self.func_field_points[0]):
```

```
            outer_matrix[i,j] =
F16_to_F4.relative_field_representation(gen_matrix_F16[i][j])[0]

        for i in range(0, outer_matrix_dim):
            for j in range(self.func_field_points[0],
self.func_field_points[0]+self.func_field_points[1]*2, 2):
            r,t =
F16_to_F4.relative_field_representation(gen_matrix_F16[i][self.func_field_points[0]
+ (j - self.func_field_points[0])/2])
            outer_matrix[i,j] = r
            outer_matrix[i,j+1] = t

        for i in range(0, outer_matrix_dim):
            for j in range(self.func_field_points[0]+self.func_field_points[1]*2,
self.func_field_points[0]+self.func_field_points[1]*2+self.func_field_points[2]*3,
3):
            r,t,u =
F64_to_F4.relative_field_representation(gen_matrix_F64[i][self.func_field_points[0]
+ (j - self.func_field_points[0])/2])
            outer_matrix[i,j] = r
            outer_matrix[i,j+1] = t
            outer_matrix[i,j+2] = u
        return outer_matrix

    def _create_generator_matrix(self, points):
        return self._create_outer_matrix(points) * self._create_inner_matrix()

    @cached_method
    def generator_matrix(self):
        """Return a generator matrix for the code."""
        return self.G

    @cached_method
    def parity_check_matrix(self):
        """Return a parity check matrix for the code."""
        Sp = self.generator_matrix().right_kernel()
        return Sp.basis_matrix()
```

```
def encode(self, w):
    return vector(self.Fq4, w) * self.generator_matrix()

@cached_method
def _an_information_set(self):
    """Return an information set for G. Repeated calls to this method will
    return the same information set."""
    #Randomised, optimistic algorithm: draw k random columns and ask if they
    #have full rank.
    k = self.true_dimension()
    pos = list(range(self.n))
    G = self.generator_matrix()
    while True:
        shuffle(pos)
        info_set = pos[:k]
        info_set.sort()
        Gt = G.matrix_from_columns(info_set)
        if rank(Gt) == k:
            return info_set

@cached_method
def _unencoder_matrix(self):
    """Find an information set for G, and return the inverse of those
    columns of G: this allows o unencode codewords"""
    Gt = self.generator_matrix().matrix_from_columns(self._an_information_set())
    return Gt.inverse()

def codeword_to_message(self, v):
    """Return the information corresponding to a codeword. This function
    satisfies `unencode(encode(info)) == info`.
    Since the information word is calculated from an information set of the
    code, then if `c` is not a codeword, then the returned vector might not
    be closer than `n-k` to `c`"""
    U = self._unencoder_matrix()
    info_set = self._an_information_set()
    cc = vector( v[i] for i in info_set )
```

```
    return cc * U

def iscodeword(self, r):
    """Returns whether r is a codeword"""
    # Check that the word is in the field
    if not all(ri in self.F for ri in r):
        return false
    # Check that the syndrome is zero
    syn = self.syndrome(r)
    if hasattr(syn, "is_zero"):
        return syn.is_zero()
    elif isinstance(syn, list):
        return all(s.is_zero() for s in syn)
    else:
        raise Exception("Don't know how to check if syndrome is zero.")

def syndrome(self, r):
    """Returns the syndrome of the received word.
    INPUT:
    - ``r`` -- The word to be evaluated, in the preferred form.

    OUTPUT:
    The syndrome in a preferred form. If the standard implementation of
    iscodeword is to be used, it should be a an object with the is_zero
    function, or a list of such"""
    return self.parity_check_matrix() * r

def random_codeword(self):
    """Return a random codeword from the code. Requires the calculation of
    the generator matrix"""
    k = self.true_dimension();
    return random_vector(self.Fq4, k) * self.generator_matrix();

@cached_method
def true_dimension(self):
    """The true dimension of a Hermitian code is always $m - g + 1$"""
    return self.k
```

```
    @cached_method
    def true_minimum_distance(self):
        """Compute the true minimum distance of this code by analysing the
        generator matrix. Computation time is exponential in code dimension."""
        # Calculate by calling the Sage integrated functions
        Csage = LinearCode(self.generator_matrix())
        return Csage.minimum_distance()

    @cached_method
    def list(self):
        """Tabulate and return all codewords of the code.
        NOTE: This list will have len (F.cardinality())^k"""
        G = self.generator_matrix()
        RS = G.row_space()
        return RS.list()

    def __iter__(self):
        for c in self.list():
            yield c

    def hamming_metric(self, v, w):
        val = 0
        for i in range(0, len(v)):
            if(v[i] == w[i]):
                val = val + 1
        return val

    def _information_to_function(self, mes):
        """Return the function in F[x,y] which corresponds to the information
        vector ``mes``, as returned by ``self.unencode(c)`` for a codeword
        ``c``"""
        assert len(mes) == self.k , "The message vector does not have the length of the
dimension of the code"
        riemann_roch_basis = self._get_riemann_roch_basis(self.Fq4)
        return sum(mes[i]*riemann_roch_basis[i] for i in range(0,
len(riemann_roch_basis)))
```

```
    def _function_to_information(self, f):
        """Return the information vector in F which corresponds to the function f in
F[x,y]"""
        riemann_roch_basis = self._get_riemann_roch_basis(self.Fq4)
        return vector( f.monomial_coefficient(elem) for elem in riemann_roch_basis )

    def _get_Q_polynomial(self):
        riemann_roch_basis = self._get_riemann_roch_basis(self.Fq4)
        poly = 0
        for j in range(0, len(riemann_roch_basis)):
            poly = poly + self.Fq4.random_element()*riemann_roch_basis[j]
        return poly

    def decode(self, w):
        while(1 == 1):
            poly = self._get_Q_polynomial()
            v = self._function_to_information(poly)*self.generator_matrix()
            if(self.hamming_metric(v, w) >= len(w) - 1):
                return v
Fq4.<a> = GF(4, 'a')
code = XNLCode(13,15)
code
```