Міністерство освіти і науки України
Національний Університет
«Києво-Могилянська Академія»
Кафедра математики факультету інформатики

Надлишкове кодування на алгебраїчних кривих

Текстова частина до дипломної роботи
за спеціальністю „Інтелектуальні системи прийняття рішень“

Керівник дипломної роботи:
д.ф.-м.н., проф. Боднарчук Ю. В.

Виконав студент:
Трояновський М. М.

Київ 2011

Зміст

Анотація **3**

Вступ **4**

1 Основні поняття **7**

1.1 Лінійні коди 7

1.2 Алгебраїчні криві 7

1.3 Дивізори 8

2 Алгебро-геометричні коди **10**

2.1 Означення 10

2.2 Приклад алгебро-геометричного коду 10

2.3 Кількість раціональних точок на алгебраїчній кривій . . . 13

2.4 Алгоритми декодування 14

3 Алгеброгеометричні коди на ермітових кривих **15**

3.1 Алгоритм 16

3.1.1 Попередня підготовка кодування 16

3.1.2 Кодування вхідного слова 17

3.1.3 Попередня підготовка декодування алгоритмом Скоробогатова-Вледуца 17

3.1.4 Декодування вхідного слова 18

3.2 Приклад 18

3.3 Аналіз складності 21

3.4 Програмна реалізація 24

Висновки **26**

Список використаної літератури **27**

Додаток А. Текст програми hermitian_code.py **30**

Додаток Б. Текст програми transmission.py **37**

Додаток В. Вивід сесії Maple/CASA **40**

Анотація

У даній роботі розглянуто алгебро-геометричні коди, клас лінійних кодів, побудованих на алгебраїчних кривих з використанням простору Рімана-Роха. Основну увагу присвячено кодам на ермітових кривих, для яких було досліджено їхні властивості, написано програмну реалізацію в пакеті SAGE, а також оцінено часову складність алгоритмів роботи з кодами у термінах єдиного параметра ермітової кривої.

Ключові слова: *надлишкове кодування, алгебро-геометричні коди, складність алгоритмів, SAGE.*

Вступ

Надлишкове кодування відіграє значну роль у боротьбі з помилками, які виникають при передачі даних через зашумлене телекомунікаційне середовище. Загалом для надійної передачі інформації через ненадійні канали застосовують дві стратегії: виявлення та виправлення помилок. У першому випадку до блоку даних додають контрольну інформацію, за допомогою якої одержувач має змогу зрозуміти, що під час зв'язку відбулося пошкодження відповідного блоку, тому необхідно зробити запит на повторну передачу. Найпростішим прикладом такої контрольної інформації є біт парності, який додається до блоку бінарних даних таким чином, аби загальна кількість одиниць у блоці була парною. Це дає можливість одержувачу виявити помилку в одному біті переданого блоку, проте якщо помилок було більше, вони можуть скасувати одна одну; втім подібний метод можна досить просто узагальнити для виявлення будь-якої непарної кількості помилок.

Проте існують застосування, у яких одного виявлення факту помилки при передачі інформації замало, оскільки повторна передача блоку даних може бути занадто витратною: так при зв'язку із об'єктами у космосі, що віддалені від Землі на велику відстань існує значна затримка, яка буде лише збільшуватися при ретрансмісії; при односторонній передачі даних (наприклад, у телемовленні) ретрансмісія взагалі технічно неможлива. У таких випадках необхідно намагатися виправляти помилки наперед, додаючи до блоку даних надлишкову інформацію, яка дозволятиме одержувачу відновити блок навіть за наявності помилок передачі. Простим прикладом даного підходу може бути таке бінарне кодування: кожен біт передається тричі підряд, одержувач аналізує триплети та декодує їх у той біт, якого кількісно більше у триплеті. Зрозуміло, що такий спосіб дозволяє виправити одну помилку в окремому триплеті, але ціна такої можливості досить велика: дані за такого підходу будуть передаватися

втричі повільніше.

Природнім є бажання знайти способи надлишкового кодування, які з одного боку дозволятимуть виправляти значну кількість помилок, а з іншого боку не призводять до значного сповільнення передачі. Розвиненою в цьому аспекті є теорія лінійних кодів, для яких виведені певні граничні характеристики, які дають змогу вибрати найкращі коди із усіх можливих. Зокрема, важливою є границя Сінглтона, яка стверджує, що для лінійного (n, k, d)-коду справжується нерівність: $d \leq n - k + 1$. Код який досягає відповідної рівності, називають кодом із максимально досяжною мінімальною відстанню, нетривіальним прикладом такого випадку слугують коди Ріда-Соломона. Станом на 2011 рік коди Ріда-Соломона є найбільш поширеними у практичних застосуваннях. Проте у цих кодів параметр n жорстко прив'язаний до розміру скінченного поля, над яким цей код визначений. Тому бажання створити довгі коди в порівнянні із розміром поля привело до пошуку нових методів побудови лінійних кодів, які матимуть ще кращі характеристики у порівнянні із наявними. Одним із таких прикладів є алгебро-геометричні коди, для них описані алгоритми побудови та декодування, а також доведено факт перевершення ними границі Варшамова-Гілберта, яка до цього вважалась недосяжною. Ці факти досі підживлюють інтерес до цих кодів, проте широкого практичного застосування станом на 2011 рік вони не мають, оскільки досі тривають дослідження в напрямку знаходження нових характеристик кодів, методів їх побудови та декодування. Програмна реалізація наявна для математичних пакетів Magma та Maple (CASA), проте вона все одно носить більше ознайомчий характер і непридатна для передачі інформації на великій швидкості.

Мета даної роботи — дослідити алгебро-геометричні коди: визначити необхідні теоретичні поняття, навести алгоритм побудови коду, його декодування, а також провести аналіз складності цих алгоритмів. Практична частина передбачає роботу із відповідними кодами у математичних

пакетах, а також власну реалізацію кодів за допомогою пакету Sage.

1 Основні поняття

1.1 Лінійні коди

Лінійний код — векторний підпростір C розмірності k векторного простору $\mathbb{F}_q^n$ розмірності n, де $\mathbb{F}_q$ — скінченне поле із q елементів. d — мінімальна відстань за Хемінгом між кодовими словами. Максимальна кількість помилок, які може виправити код: $r = \left\lfloor \frac{d-1}{2} \right\rfloor$. Підсумовуючи ці основні характеристики, часто кажуть про лінійний код як про $(n, k, d)_q$-код.

Нерівність Сінглтона: для (n, k, d)-коду виконується: $d \leq n-k+1$, коди для яких виконується рівність $d = n-k+1$ називають кодами із максимально досяжною мінімальною відстанню (maximum distance separable), вони є у певному сенсі найкращими. З цієї нерівності можна побачити, що, за фіксованого n, k та d не можуть бути одночасно великими.

Нерівність Варшамова-Гілберта: якщо виконується співвідношення

$$\sum_{i=1}^{d-1} C_{n-1}^{i} (q-1)^i < q^{n-k}$$

то існує лінійний $(n, k, d)_q$-код.

1.2 Алгебраїчні криві

$\mathbb{A}^n$ — n-вимірний афінний простір над полем k, точками якого будуть надори $P = (x_1, \ldots, x_n), x_i \in k$. $\mathbb{P}^n$ — n-вимірний проективний простір над k, точками якого є набори $Q = (y_0 : y_1 : \ldots : y_n), y_i \in k$, де не всі y_i дорівнюють нулю, при чому набори $(y_0 : y_1 : \ldots : y_n)$ та $(\lambda y_0 : \lambda y_1 : \ldots : \lambda y_n)$ при $\lambda \in k, \lambda \neq 0$ визначають одну й ту саму точку. При цьому простір $\mathbb{A}^n$ передбачає природнє вкладення у $\mathbb{P}^n$, яке задається формулою $P = (x_1, \ldots, x_n) \mapsto (1 : y_1 : \ldots : y_n)$.

Гладка проективна алгебраїчна крива C у N-вимірному проективному просторі P^N над полем $\mathbb{F}$ — це одновимірний многовид без особливостей, тобто сукупність розв'язків системи однорідних алгебраїчних рівнянь від $(N+1)$ змінної із коефіцієнтами із $\mathbb{F}$ таких, що матриця похідних у кожній точці задає рівняння прямої; розв'язки загалом потрібно розглядати з координатами у замиканні $\bar{\mathbb{F}}$ основного поля. Інтуїтивно гладкість означає відсутність точок повернення та самоперетину, одновимірність — єдиність дотичного напрямку у кожній точці.

З кривою C пов'язують поле $\mathbb{F}(C)$ раціональних функцій на ній, тобто відношень однорідних многочленів рівного степеню з точністю до рівнянь з C. Функція є регулярною в точці, якщо вона набуває у ній скінченного значення.

1.3 Дивізори

Дивізор алгебраїчної кривої C — це формальна сума її точок, взятих із певною кратністю:

$$D = \sum_{P \in C} n_P P, n_P \in \mathbb{Z},$$

де тільки скінченна кількість цілих чисел n_P відмінна від нуля.

Множина усіх дивізорів, позначена як Div, формує Абелеву групу із законом додавання:

$$\sum_{P \in C} n_P P + \sum_{P \in C} m_P P = \sum_{P \in C} (n_P + m_P) P$$

Степінь D — це цілочисельна сума його коефіцієнтів:

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

$\forall D_1, D_2 \in Div : deg(D_1) + deg(D_2) = deg(D_1 + D_2)$, тому відображення $deg : Div \to \mathbb{Z}$ є гомоморфізмом. Множина Div^0 дивізорів ступеню 0 є

підгрупою Div, а також ядром гомоморфізму deg.

Порядок D у точці P — це цілий коефіцієнт при P: $ord_P(D) = n_P$.

Носій дивізора — $supp(D) = \{P | n_P \neq 0\}$.

Якщо у дивізора всі n_P — невід'ємні, такий дивізор називається ефективним. Поняття ефективності дозволяє задати відношення порядку на множині дивізорів.

Довільну ненульову раціональну функцію $f \in \mathbb{K}(X)$ можна пов'язати із дивізором $(f) = \sum ord_P(f)P$. Таке визначення є коректним, бо у такої функції на X є скінченна кількість нулів та полюсів, тому $ord_P(f) \neq 0$ для скінченного числа точок P.

Нехай X — гладка проективна крива роду g та канонічного класу K. Теорема Рімана-Роха стверджує, що для будь-якого дивізора $D \in Div(X)$ виконується співвідношення

$$\ell(D) - \ell(K - D) = degD - g + 1$$

тут $\ell(D)$ — розмірність простору раціональних функцій на кривій, полюси яких у кожній точці не гірші, ніж відповідні коефіцієнти D. Зазначене співвідношення грає ключову роль для визначення конструктивних характеристик алгеброгеометричного коду.

2 Алгебро-геометричні коди

2.1 Означення

Алгебро-геометричні коди — це клас лінійних кодів, які вперше були описані Валерієм Денисовичем Гоппою, тому їх часто називають кодами Гоппи, проте це іноді може вносити неясність, оскільки такі коди не єдині, які називають на його честь. Його ідеєю було сконструювати код, за допомогою обчислення функцій із простору Рімана-Роха на раціональних точках алгебраїчної кривої.

$L(D)$ — це $\mathbb{F}$-векторний простір для будь-якого раціонального дивізора кривої, визначеної над $\mathbb{F}$. Згадавши, що лінійний код — це просто векторний підпростір $\mathbb{F}^n$, виникає природне бажання побудувати код. Проте $L(D)$ є векторним простором функцій, тому він не зобов'язаний бути кодом; однак лінійна властивість просторів Рімана-Роха дозволяє сконструювати лінійний код. Окрім того, теорема Рімана-Роха дозволяє визначити характеристики коду.

2.2 Приклад алгебро-геометричного коду

Нехай ми маємо криву C над скінченним полем $\mathbb{F}$, скінченну множину точок $\mathcal{P} = \{P_1, \ldots, P_n\}$ і дивізор D, відокремлений (disjoint) від P_i. Код Гоппи $GC = GC(\mathcal{P}, D, C)$ визначають як

$$GC = \{(\phi(P_1), \ldots, \phi(P_n)) | \phi \in L(D)\}$$

Зазначений запис є функціональним кодом (L-конструкція). Код лишків (residue code, Ω-конструкція) визначають як

$$GC' = \{c \in \mathbb{F}^n | c \cdot (\phi(P_1), \ldots, \phi(P_n)) = 0, \forall \phi \in L(D)\}$$

Конструктивні характеристики коду можна оцінити такими співвід-

ношеннями:

$$n = |\mathcal{P}|,\ k \geq n - a + g - 1,\ d \geq a - 2g + 2$$

Для прикладу нехай C — еліптична крива, задана за допомогою $f = X^3 + Y^2Z + YZ^2$ над $\mathbb{F}_4$ $(w^2 + w + 1 = 0)$. Це несингулярна крива роду 1 із дев'ятьма раціональними точками, одна з яких, $Q = (0 : 1 : 0)$, — нескінченно віддалена точка. Утворимо множину $\mathcal{P}$ усіма точками, окрім Q, тоді

$$P_1 = (0 : 0 : 1);\ P_2 = (0 : 1 : 1);\ P_3 = (1 : w : 1);\ P_4 = (1 : w^2 : 1);$$

$$P_5 = (w : w : 1);\ P_6 = (w : w^2 : 1);\ P_7 = (w^2 : w : 1);\ P_8 = (w^2 : w^2 : 1)$$

Нехай дивізор $D = 5Q$, тоді $L(D)$ — п'ятивимірний простір із базисом $L(5Q) = \langle 1, x, y, x^2xy\rangle$.

У цьому випадку конструктивна мінімальна відстань коду дорівнює 5, тому код буде здатен виправити дві помилки. Перевірочна матриця виглядатиме таким чином:

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ x(P_1) & x(P_2) & \dots & x(P_8) \\ y(P_1) & y(P_2) & \dots & y(P_8) \\ x^2(P_1) & x^2(P_2) & \dots & x^2(P_8) \\ xy(P_1) & xy(P_2) & \dots & xy(P_8) \end{pmatrix}$$

Після обчислень:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & w & w & w^2 & w^2 \\ 0 & 1 & w & w^2 & w & w^2 & w & w^2 \\ 0 & 0 & 1 & 1 & w^2 & w^2 & w & w \\ 0 & 0 & w & w^2 & w^2 & 1 & 1 & w \end{pmatrix}$$

Програмний код, який реалізує описаний вище приклад у математичному середовищі Magma:

```
> A<x,y> := AffineSpace(FiniteField(4),2);
> f:=x^3+y^2+y;
> C:=Curve(A,f);
> P1:=Places(C,1);
> Div := DivisorGroup(C);
> P1;
[
    Place at (0 : 1 : 0),
    Place at (0 : 0 : 1),
    Place at (0 : 1 : 1),
    Place at ($.1 : $.1 : 1),
    Place at ($.1 : $.1^2 : 1),
    Place at ($.1^2 : $.1 : 1),
    Place at ($.1^2 : $.1^2 : 1),
    Place at (1 : $.1 : 1),
    Place at (1 : $.1^2 : 1)
]
> D:=5*P1[1];
> P2:=[P1[2],P1[3],P1[4],P1[5],P1[6],P1[7],P1[8],P1[9]];
> D0:=Div!D;
> D0;
Divisor on Curve over GF(2^2) defined by
$.1^3 + $.2^2*$.3 + $.2*$.3^2
> RiemannRochSpace(D0);
KModule of dimension 5 over GF(2^2)
Mapping from: KModule of dimension 5 over GF(2^2) to Function Field of
over GF(2^2) defined by
$.1^3 + $.2^2*$.3 + $.2*$.3^2
```

```
> Basis(D0);
[
    1,
    x,
    x^2,
    y,
    x*y
]
> GC:=AlgebraicGeometricCode(P2,D0);
> GC;
[8, 5, 3] Linear Code over GF(2^2)
Generator matrix:
[    1     0     0     0     0 $.1^2 $.1^2     1]
[    0     1     0     0     0 $.1^2   $.1     0]
[    0     0     1     0     0   $.1     1   $.1]
[    0     0     0     1     0   $.1     0 $.1^2]
[    0     0     0     0     1     1     1     1]
```

2.3 Кількість раціональних точок на алгебраїчній кривій

Нерівність Серре дає оцінку кількості точок N на кривій роду g, визначеної над $\mathbb{F}_q$:

$$|N - (q+1)| \leq g\lfloor 2\sqrt{g} \rfloor$$

Оскільки ця величина визначає довжину відповідного алгебро-геометричного коду, доцільно застосовувати криві із найбільшою кількістю раціональних точок.

2.4 Алгоритми декодування

Основний алгоритм декодування коду $C = (X, \mathcal{P}, D)_\Omega$ за допомогою допоміжного дивізора виглядає таким чином:

1. Обчислити базис f_i простору $L(D)$, базис g_j простору $L(D')$ і базис h_k простору $L(D - D')$.

2. Для даного вектора $v \in \mathbb{F}_q^n$ обчислити синдроми $s(v, g_j, h_k)$ та $s(v, f_i)$.

3. Знайти розв'язок y системи лінійних рівнянь

$$\sum_{i=1}^{\ell} s_{ij} x_i = 0$$

4. Знайти (за допомогою перебору точок P_i ті i, для яких $g_y(P_i) = 0$.

5. Знайти розв'язок системи рівнянь

$$\sum_{i \in I_y} f_j(P_i) z_i = s(v, f_j)$$

3 Алгеброгеометричні коди на ермітових кривих

Нехай m — степінь простого числа. Тоді над полем $\mathbb{F}_{m^2}$ крива, яка визначається рівнянням $H_m = x^{m+1} + y^m z + y z^m$, називається ермітовою кривою. Ця крива має $m^3 + 1$ $\mathbb{F}_{m^2}$-раціональних точок, серед них є одна нескінченно віддалена, $Q = (0 : 1 : 0)$. Рід цієї кривої можна обчислити як $g = \frac{m(m-1)}{2}$.

Вибір даного класу кривих зумовлений такими факторами:

1. Для дивізорів зазначеного вигляду на ермітових кривих існує тривіальний алгоритм побудови простору Рімана-Роха.

2. Ермітові криві є максимальними (мають найбільшу кількість точок серед кривих даного роду над відповідним полем.

3. Зазначений вибір дивізору D спрощує вибір допоміжних дивізорів для декодування.

4. Базис простору Рімана-Роха за даної побудови складається із функцій специфічного вигляду, які можна зберігати у вигляді набору показників степенів та коефіцієнту з поля, що є більш ефективним у порівнянні із загальним випадком.

Нерівність Серра дає оцінку максимальної кількості точок N на кривій роду g над полем $\mathbb{F}_q$:

$$N \leq q + 1 + g\lfloor 2\sqrt{q} \rfloor$$

Підставивши параметри ермітової кривої у цю нерівність, отримаємо:

$$N \leq m^2 + 1 + \frac{m(m-1)}{2}\lfloor 2\sqrt{m^2} \rfloor = m^2 + 1 + m(m-1)m =$$

$$= m^2 + 1 + m^3 - m^2 = m^3 + 1$$

Звідси зрозуміло, що ермітова крива є максимальною.

Для ермітової кривої базис простору $L(aQ)$ має особливий вигляд та може бути досить тривіально обчислений:

$$L(aQ) = span\left\{ \frac{x^i y^j}{z^{i+j}} \middle| im + j(m+1) \leq a, i \leq m \right\}$$

3.1 Алгоритм

Нехай $\mathcal{P}$ — множина усіх $\mathbb{F}_{m^2}$-раціональних точок H_m, окрім Q, тоді алгеброгеометричний код $C_{H_m,a} = (H_m, \mathcal{P}, aQ)_\Omega$ називатимемо ермітовим кодом з параметрами (m, a). Параметри цього лінійного коду: $[m^3, m^3 - a + \frac{m(m-1)}{2} - 1, a - m(m-1) + 2]_{\mathbb{F}_{m^2}}$ за умови, що $a > m(m-1) - 2$.

3.1.1 Попередня підготовка кодування

Обчислимо базис $L(aQ)$, за теоремою Рімана-Роха кількість його елементів дорівнюватиме: $\ell_{aQ} = a + 1 - \frac{m(m-1)}{2}$:

$$basis(L(aQ)) = \left\{ \frac{x^i y^j}{z^{i+j}} \middle| im + j(m+1) \leq a, i \leq m \right\}$$

Обчислимо множину раціональних точок $\mathcal{P}$.

Обчислимо перевірочну матрицю H:

$$H = \begin{pmatrix} f_1(P_1) & \cdots & f_1(P_{m^3}) \\ \vdots & \ddots & \vdots \\ f_{\ell_{aQ}}(P_1) & \cdots & f_{\ell_{aQ}}(P_{m^3}) \end{pmatrix}, P_i \in \mathcal{P}, f_j \in basis(L(aQ))$$

Обчислимо породжуючу матрицю G:

$$G = \begin{pmatrix} G_1 \\ \vdots \\ G_k \end{pmatrix}, G_i \in basis(kernel(H^T))$$

Зауваження: $(H_m, \mathcal{P}, aQ)^{\perp}_{\Omega} = (H_m, \mathcal{P}, (m^3 + m^2 - m - a - 2)Q)_L$, з цих міркувань можна обчислювати матриці H і G як матриці G' та H' відповідного дуального коду.

3.1.2 Кодування вхідного слова

Для вхідного слова $v, v \in \mathbb{F}_{m^2}^k$ обчислимо кодове слово $c, c \in \mathbb{F}_{m^2}^n$:

$$c = v \cdot G$$

3.1.3 Попередня підготовка декодування алгоритмом Скоробогатова-Вледуца

Кількість помилок, які може виправити алгоритм: $t = \left\lfloor \frac{a - \frac{3m(m-1)}{2} + 1}{2} \right\rfloor$

Обчислимо базис $\{g_j\}$ простору $L((t+g)Q)$ та обчислимо базис $\{h_k\}$ простору $L((a-t-g)Q)$.

Обчислимо матрицю синдромів S:

$$S = \begin{pmatrix} g_1 \\ \vdots \\ g_{\ell((t+g)Q)} \end{pmatrix} \cdot \begin{pmatrix} h_1 & \cdots & h_{\ell((a-t-g)Q)} \end{pmatrix}$$

3.1.4 Декодування вхідного слова

Для вхідного слова $v, v = c+e, v \in \mathbb{F}_{m^2}^n$ обчислимо матрицю синдромів слова:

$$S_v = \begin{pmatrix} g_1 f_1 \cdot v & g_2 f_1 \cdot v & \dots \\ g_1 f_2 \cdot v & g_2 f_2 \cdot v & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

Тут $\phi \cdot v = (\phi(P_1), \dots, \phi(P_{m^3})) \cdot v$

Якщо отримана матриця складається із нулів, тоді вектор e є нульовим, а слово v — кодовим.

Знайдемо ядро S_v, якщо базис відповідного простору складається не з одного вектора, алгоритм не може декодувати слово v. Нехай $(s_1, \dots, s_{\ell((t+g)Q)})$ — вектор відповідного базису. Обчислимо скалярний добуток:

$$\theta = (s_1, \dots, s_{\ell((t+g)Q)}) \cdot (g_1, \dots, g_{\ell((t+g)Q)})$$

Знайдемо координати i_e, у яких відбулися помилки: перебором $P_{i_e} \in \mathcal{P}$ знаходимо такі точки, для яких $\theta(P_{i_e}) = 0$.

Вирішуємо систему рівнянь:

$$\begin{pmatrix} f_1(P_{i_{e_1}}) & \dots & f_1(P_{i_{e_t}}) \\ \vdots & \ddots & \vdots \\ f_{\ell(aQ)}(P_{i_{e_1}}) & \dots & f_{\ell(aQ)}(P_{i_{e_t}}) \end{pmatrix} \cdot \begin{pmatrix} e_1 \\ \vdots \\ e_t \end{pmatrix} = \begin{pmatrix} f_1 \cdot v \\ \vdots \\ f_{\ell(aQ)} \cdot v \end{pmatrix}, f_i \in L(aQ)$$

Отримавши значення e_{i_e}, відновимо вхідний вектор: $c = v - e$.

Декодоване слово w отримаємо із співвідношення $w \cdot G = c$.

3.2 Приклад

Для прикладу візьмемо ермітовий код з параметрами $(2, 6)$. Відповідна крива H_2 задається рівнянням $x^3 + y^2 z + y z^2$ над полем $\mathbb{F}_4$. Окрім нескінченно віддаленої точки $Q = (0 : 1 : 0)$ у неї 8 раціональних точок:

$\mathcal{P} = \{(0:0:1), (0:1:1), (1:w:1), (1:w+1:1), (w:w:1),$
$(w:w+1:1), (w+1:w:1), (w+1:w+1:1)\}$

Ця крива має рід 1 (більше того, це еліптична крива), отже параметри лінійного коду $[8, 2, 6]_{\mathbb{F}_4}$, кількість помилок, які може виправити алгоритм Скоробогатова-Вледуца $t = 2$.

$$L(6Q) = span\left\{1, y/z, y^2/z^2, x/z, xy/z^2, x^2/z^2\right\}$$

Знаходимо перевірочну матрицю обчисленням елементів базису $L(6Q)$ у точках $\mathcal{P}$, а породжуючу — як ядро перевірочної.

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

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

Нехай вхідне повідомлення $(0, 1)$, тоді кодове слово:

$$c = (0, 0, 1, 1, w, w, w+1, w+1)$$

Нехай при передачі відбулись помилки у другій та восьмій позиціях і приймальна сторона отримала слово:

$$v = (0, 1, 1, 1, w, w, w+1, 0)$$

Матриця синдромів для коду має вигляд:

$$S = \begin{pmatrix} 1 & y/z & x/z \\ y/z & y^2/z^2 & xy/z^2 \\ x/z & xy/z^2 & x^2/z^2 \end{pmatrix}$$

Для отриманого вектора v вона набуде вигляду:

$$S_v = \begin{pmatrix} w & w+1 & w \\ w+1 & 0 & 1 \\ w & 1 & 1 \end{pmatrix}$$

Локатор помилок:

$$\theta = \frac{((w+1) * x + y + z)}{z}$$

Він набуває значення 0 у точках P_2, P_3, P_8. Для знаходження значення помилок необхідно розв'язати таку систему:

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & w & w+1 \\ 1 & w+1 & w \\ 0 & 1 & w+1 \\ 0 & w & w \\ 0 & 1 & w \end{pmatrix} \times \begin{pmatrix} e_{i_2} \\ e_{i_3} \\ e_{i_8} \end{pmatrix} = \begin{pmatrix} w \\ w+1 \\ 0 \\ w \\ 1 \\ 1 \end{pmatrix}$$

Після розв'язку системи одержуємо значення помилок у відповідних позиціях:

$$e_{i_2} = 1, e_{i_3} = 0, e_{i_8} = w + 1$$

Відновлюємо кодове слово:

$$c = v - e = (0, 0, 1, 1, w, w, w + 1, w + 1)$$

Декодуємо початкове повідомлення:

$$(m_1, m_2) \times \begin{pmatrix} 1 & 1 & 0 & 0 & w+1 & w+1 & w & w \\ 0 & 0 & 1 & 1 & w & w & w+1 & w+1 \end{pmatrix}$$

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

3.3 Аналіз складності

Через те, що параметр m накладає певні обмеження на параметр a ермітового коду, складність алгоритмту можна виразити лише через параметр m.

При побудові коду, а також при декодуванні необхідно обчислювати базиси трьох просторів Рімана-Роха: $L(aQ), L((t+g)Q), L((a-t-g)Q)$. Для того, аби в теоремі Рімана-Роха досягалась рівність, необхідно накласти такі обмеження на параметри (для спрощення обчислень у виразі параметру t буде прибрано округлення вниз, це може вплинути на точність оцінки при малих m):

$$\begin{cases} a > 2g-2 \\ t+g > 2g-2 \\ a-t-g > 2g-2 \end{cases}, \begin{cases} t > g-2 \\ a > 3g+t-2 \end{cases}, \begin{cases} \frac{a-3g+1}{2} > g-2 \\ a > 3g + \frac{a-3g+1}{2} - 2 \end{cases},$$

$$\begin{cases} a-3g+1 > 2g-4 \\ 2a > 6g+a-3g+1-4 \end{cases}, \begin{cases} a > 5g-5 \\ a > 3g-3 \end{cases}, a > 5\frac{m(m-1)}{2} - 5$$

З іншого боку має виконуватись нерівність $0 < k < n$, тобто $0 < m^3 - a - \frac{m(m-1)}{2} - 1 < m^3$, яка накладає обмеження:

$$\begin{cases} a > \frac{m(m-1)}{2} - 1 \\ a < m^3 + \frac{m(m-1)}{2} - 1 \end{cases}$$

Підсумовуючи наведені нерівності, можна сказати, що a має лежати у такому проміжку:

$$5\frac{m(m-1)}{2} - 5 < a < m^3 + \frac{m(m-1)}{2} - 1,$$

тобто асимптотично a має зростати швидше, ніж m^2, але повільніше, ніж m^3, тому далі можна вважати, що $a = O(m^3)$ і $a = \Omega(m^2)$.

Більшість етапів алгоритму використовує операції із матрицями над скінченним полем, тому відповідні оцінки складності будуть прив'язані до складності операцій в полі. При використанні представлення елементів $\mathbb{F}_{m^2}$ у вигляді лишків незвідного полінома операції множення та ділення потребуватимуть $O(\ln^2 m^2) = O(\ln^2 m)$ операцій, піднесення елемента поля до додатнього степеню N потребуватиме $O(\ln N \ln^2 m)$ операцій. Додавання двох елементів поля потребуватиме $O(\ln m)$ операцій. Використання представлення елементів поля у вигляді степенів породжуючого елемента зведе операції множення та ділення до додавання та віднімання показників степенів за модулем $m^2 - 1$, складність такої операції дорівнюватиме $O(\ln(m))$. Проте така форма представлення незручна для додавання та віднімання елементів, тому необхідно попередньо створити таблицю відповідностей двох форм запису, це потребуватиме $O(m^2 \ln^2 m)$ операцій та $O(m^2)$ пам'яті.

Для ермітової кривої процедуру знаходження базису $L(aQ)$ можна зводиться до знаходження пар (i, j), для яких виконується співвідношення $im + j(m+1) \leq n$ у циклах, пробігаючи $0 \leq j \leq \left\lfloor \frac{a}{m+1} \right\rfloor$ та $0 \leq i \leq m$. Тобто усього $O(a)$ операцій вартістю порядку $O(\ln a)$ кожна, загалом $O(a \ln a)$.

Для побудови перевірочної матриці H необхідно знайти базис $L(aQ)$, це займе $O(a \ln a) = O(m^3 \ln m)$ операцій. Перебірна реалізація знаходження раціональних точок (підстановка усіх $x, y \in \mathbb{F}_{m^2}$ у рівняння кривої, найдорожча арифметична операція — піднесення елемента поля до m-го степеня) займе $O(m^2 m^2 m \ln^2 m) = O(m^5 \ln^2 m)$. Отже, у матриці

H $\ell_{aQ} = a + 1 - \frac{m(m-1)}{2}$ рядків та m^3 стовпчиків, тому загальну кількість її елементів можна оцінити як $O(m^3 m^3) = O(m^6)$. Для обчислення кожного з її елементів найдорожчою арифметичною операцією буде піднесення до степеню (в найгіршому випадку $\frac{a}{m+1} = O(m^2)$), складність операції $O(\ln m^2 \ln^2 m) = O(\ln^3 m)$. Тому загальна складність обчислення всіх елементів H буде $O(m^6 \ln^3 m)$, як бачимо це і буде найдорожчою операцією у побудові H. Із властивості самодуальності ермітового коду вважатимемо, що складність побудови матриці G не перевищуватиме складності побудови H.

Кодування полягає у множенні вектора довжини k на матрицю $k \times n$: для цього необхідно обчислити kn результатів множення елементів поля: складність $kn \ln^2 m = O(m^3 m^3 \ln^2 m) = O(m^6 \ln^2 m)$.

При декодуванні найдорожчою операцією буде обчислення матриці S_v, кількість її елементів можна оцінити як $O(m^6)$, обчислення кожного її елементу полягає в обчисленні скалярного добутку двох векторів довжиною m^3. Отже, загальна складність знаходження S_v дорівнюватиме $O(m^9 \ln^2 m)$.

Вартість розв'язку систем лінійних рівнянь, яка виникає при декодуванні має аналогічну складність (матриця розміром $m^3 \times m^3$): метод Гаусса виглядатиме так: i-тий рядок ділимо на елемент $a_{i,i}$, для кожного іншого k-го рядка віднімаємо даний, поділений на $a_{k,i}$: це буде три вкладені цикли по m^3 кроків у кожному із вартістю внутрішньої операції $O(\ln^2 m)$. Загальна складність методу: $O(m^9 \ln^2 m)$.

Наведена складність алгоритму Скоробогатова-Вледуца збігається із оцінкою, яка дається у літературі: $O(n^3)$ операцій в скінченному полі. Якщо врахувати, що для ермітового коду $n = m^3$, а вартість множення $O(\ln^2 m)$, ми отримуємо оцінку $O(m^9 \ln^2 m)$.

3.4 Програмна реалізація

За основу програмної реалізації ермітових кодів, створеної в рамках даної роботи, було взято бібліотеку CASA для пакету Maple. Ця бібліотека має досить широкі можливості та не обмежується лише роботою із кодами, проте вона мала активний розвиток орієнтовно до 2000 року, на що натякає той факт, що Maple підтримується у версії 6. На щастя, бібліотека коректно працює також із новішими версіями математичного пакету. Для власної реалізації алгоритмів було обрано пакет Sage, у якому наявні необхідні класи: скінченні поля, кільця поліномів, матриці та криві. Підтримуються операції зведення матриці до рядкової канонічної форми (потрібно для розв'язку систем лінійних рівнянь), знаходження ядра матриці, а також знаходження раціональних точок кривої. Система також надає можливість обчислювати базис простору Рімана-Роха для довільних дивізорів довільних кривих через інтерфейс Singular за алгоритмом Бріла-Ньотера, проте існують сумніви, підтверджені документацією SAGE, щодо коректності реалізації алгоритму, через це було вирішено не покладатися на наведену можливість, а використати власну реалізацію алгоритму для ермітових кривих.

Основний програмний модуль містить клас HermitianCode, який будує ермітовий код із параметрами (m, a). Якщо при побудові параметр a виявляється неузгодженим із m під час конструкції коду буде підняте виключення CodeConstructionError.

При ініціалізації будується відповідна ермітова крива, знаходяться її раціональні точки за допомогою алгоритмів із SAGE, обчислюються базиси просторів Рімана-Роха дивізорів $L(aQ), L((t+g)Q), L((a-t-g)Q)$, також обчислюються матриця синдромів коду, перевірочна та породжуюча матриці.

Процедура кодування тривіальна і полягає у множенні вектора на матрицю. Для декодування виділено допоміжну процедуру множення вектора на функцію (скалярний добуток вхідного вектора на вектор, утво-

рений значеннями функції у точках множини $\mathcal{P}$). Із практичних міркувань було застосовано оптимізацію, яка полягає в тому, що після обчислення вектора значень функції алгоритм його запам'ятовує в кеші, й надалі за необхідності використовує збережене значення.

При декодуванні обчислюється матриця синдромів для даного вхідного вектора, знаходиться її ядро, якщо відповідний простір має розмірність 0, це означає, що кількість помилок, які відбулися при передачі, перевищують кількість помилок, які може виправити алгоритм Скоробогатова-Вледуца. Далі обчислюється локатор помилок, перебираються точки із $\mathcal{P}$, знаходяться ті, у яких локатор помилок набуває значення 0. Після цього розв'язується система лінійних рівнянь значень помилок шляхом зведення матриці до рядкової канонічної форми за допомогою алгоритму, наявному в SAGE. Далі відновлюється вхідний вектор та декодується повідомлення за допомогою розв'язку системи лінійних рівнянь.

Для демонстрації було написано додатковий модуль, який імітує передавальну та приймальну сторони, а також канал зв'язку із шумом. Технічно це реалізовано за допомогою сервера, який передає клієнтові кодові слова із доданих до них шумом, при цьому можна задавати імовірність помилки у кожній позиції вектора. Відповідно клієнт отримує слова, відновлює їх за можливості, декодує та виводить декодований текст на екран, за неможливості видає на екран фрагмент із нижніх підкреслювань. Для роботи із текстом було написано додаткові процедури, які переводять літери у вектори над скінченним полем і навпаки. Відповідні процедури досить прості, тому вони накладають досить жорсткі обмеження на параметри коду для збереження узгодженості, перевірялися коди із параметрами $(4, 53)$ та $(4, 37)$.

Висновки

Хоча для алгебро-геометричних кодів доведено наявність гарних асимптотичних характеристик, а також описано алгоритми побудови та декодування, в загальному випадку практична реалізація відповідних програм може виявитись досить нетривіальною. Через це необхідно розглядати певні підкласи кодів, для яких виконуються певні спеціальні властивості, які полегшують написання програмного коду, а також враховують особливості архітектури обчислювальної техніки.

У даній роботі основний акцент було зроблено на розгляді одноточкових кодів, побудованих на ермітових кривих, для яких дивізор $D = aQ$, а множина $\mathcal{P}$ утворена рештою раціональних точок.

Для зазначеного підкласу кодів було написано програмну реалізацію із використанням математичного пакету SAGE, яка є придатною для ермітових кривих із значенням параметру $m \leq 8$. Для даної реалізації кодів також було написано приклад клієнтського застосування, яке імітує передачу тексту по зашумленому каналу, ця програма унаочнює роботу надлишкового кодування, а також надає гарне враження про швидкість даної реалізації декодування (зокрема про неможливість використання цього варіанту в промислових застосуваннях). Втім наведена програмна реалізація може бути наочним прикладом для студентів у курсах із кодування інформації.

Для ермітових кодів було також обчислено часову складність алгоритмів побудови коду, кодування та декодування основним алгоритмом (Скорбогатова-Вледуца). Всі оцінки були виражені у термінах єдиного параметру m, що однозначно задає ермітову криву. Для цього було оцінено параметр a як $O(m^3)$ і отримано такі результати: побудова коду — $O(m^6 \ln^3 m)$, кодування — $O(m^6 \ln^2 m)$, декодування — $O(m^9 \ln^2 m)$. Одержана оцінка для декодування збігаєтсья із тією, що наводиться в літературі після врахування того, що $n = m^3$.

Література

[1] С. Г. Влэдуц Д. Ю. Ногин, М. А. Цфасман. Алгеброгеометрические коды. Основные понятия / М. А. Цфасман С. Г. Влэдуц, Д. Ю. Ногин. — Издательство Московского центра непрерывного образования, 2003.

[2] Гоппа, В. Д. Алгебраико-геометрические коды / В. Д. Гоппа // Изв. АН СССР. Сер. матем. — 1982. — Vol. 46, № 4. — P. 762–781.

[3] Гоппа, В. Д. Коды, ассоциированные с дивизорами / В. Д. Гоппа // Пробл. передачи информ. — 1977. — Vol. 13, № 1. — P. 33–39.

[4] Hao, Chen. Algebraic geometric codes with applications / Chen Hao // Frontiers of Mathematics in China. — 2007. — Vol. 2, № 1. — P. 1–11.

[5] Цфасман, М. А. Коды Гоппы, лежащие выше границы Варшамова–Гилберта / М. А. Цфасман // Пробл. передачи информ. — 1982. — Vol. 18, № 3. — P. 3–6.

[6] А. М. Барг Г. Л. Кацман, М. А. Цфасман. Алгеброгеометрические коды по кривым малых родов / М. А. Цфасман А. М. Барг, Г. Л. Кацман // Пробл. передачи информ. — 1987. — Vol. 23, № 1. — P. 42–46.

[7] Cheng, Qi. Hard problems of algebraic geometry codes.

[8] Shokrollahi, M. Amin. Decoding algebraic-geometric codes beyond the error-correction bound / M. Amin Shokrollahi, Hal Wasserman // STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing. — New York, NY, USA: ACM, 1998. — P. 241–248.

[9] Dai, Zhuo Jia. — Algebraic Geometric Coding Theory. — Master's thesis, School of Mathematics and Statistics, University of Sydney, Australia, 2006.

[10] Goppa, V. D. Geometry and Codes / V. D. Goppa; Ed. by M. Hazewinkel. — New York: Springer-Verlag, 2002.

[11] Goldschmidt, D. M. Algebraic functions and projective curves / D. M. Goldschmidt. — Springer, 2002.

[12] А. Г. Ростовцев, Е. Б. Маковенко. Теоретическая криптография / Е. Б. Маковенко А. Г. Ростовцев. — АНО НПО «Професионал», 2004.

[13] Coblitz, N. Algebraic Aspects of Cryptography / N. Coblitz. — Springer, 2002.

[14] Pellikaan, Ruud. On a decoding algorithm for codes on maximal curves / Ruud Pellikaan // IEEE Transactions on Information Theory. — 1989. — Vol. 35, № 6. — P. 1228–1232.

[15] Porter, S. C. Decoding geometric goppa codes using an extra place / S. C. Porter, Ba-Zhong Shen, Ruud Pellikaan // IEEE Transactions on Information Theory. — 1992. — Vol. 38, № 6. — P. 1663–1676.

[16] Høholdt, Tom. On the decoding of algebraic-geometric codes / Tom Høholdt, Ruud Pellikaan // IEEE Transactions on Information Theory. — 1995. — Vol. 41, № 6. — P. 1589–1614.

[17] Fast decoding of algebraic-geometric codes up to the designed minimum distance / Shojiro Sakata, Jørn Justesen, Y. Madelung et al. // IEEE Transactions on Information Theory. — 1995. — Vol. 41, № 6. — P. 1672–1677.

[18] Duursma, Iwan M. Majority coset decoding / Iwan M. Duursma // IEEE Transactions on Information Theory. — 1993. — Vol. 39, № 3. — P. 1067–.

[19] Feng, Gui Liang. Decoding algebraic-geometric codes up to the designed minimum distance / Gui Liang Feng, Thammavarapu R. N. Rao // IEEE Transactions on Information Theory. — 1993. — Vol. 39, № 1. — P. 37–45.

[20] Wocjan, P. — The Brill–Noether Algorithm: Construction of Geometric Goppa Codes and Absolute Factorization. — Master's thesis, University of Kalsruhe, 1999. http://www.cs.caltech.edu/ wocjan/.

[21] Hess, Florian. Computing riemann-roch spaces in algebraic function fields and related topics / Florian Hess // J. Symb. Comput. — 2002. — Vol. 33, № 4. — P. 425–445.

[22] Deischinger, Harald. — Algebraic Geometric Codes. — Master's thesis, Johannes Kepler Universität, 1999.

[23] D. Le Brigand, J.J. Risler. Algorithme de brill-noether et codes de goppa / J.J. Risler D. Le Brigand // Bulletin de la Société Mathématique de France. — 1988. — Vol. 116, № 2. — P. 231–253.

[24] Tom Høholdt, Jacobus H. van Lint. Handbook of Coding Theory / Jacobus H. van Lint Tom Høholdt, Ruud Pellikaan. — Amsterdam: Elsevier, 1998. — P. 871–961.

Додаток А. Текст програми hermitian_code.py

```
1  import unittest
2  import logging
3
4  logger = logging.getLogger('hermitian_code')
5  logger.setLevel(logging.CRITICAL)
6  while len(logger.parent.handlers) > 1:
7      logger.parent.removeHandler(logger.parent.handlers[-1])
8
9
10
11 class DecodingError(Exception):
12     pass
13
14 class CodeConstructionError(Exception):
15     pass
16
17 class HermitianCode():
18     def __init__(self, m, a=None):
19         '''One-point AG code on Hermitian curve H_m with D=a*Q'''
20         if not is_prime_power(m):
21             raise Exception('m must be a prime power')
22         self.m = m
23         self.g = int(m*(m-1)/2)
24         field_size = m**2
25         self.F = GF(field_size,'w')
26         x, y, z = PolynomialRing(self.F, 3, 'xyz').gens()
27         self.x, self.y, self.z = x, y, z
28         f = x**(m+1) + y**m * z + y * z**m
29         self.C = Curve(f)
30         if a is None:
31             self.a = m**3 - m**2 + m + 1
32         else:
33             self.a = a
34         # t - number of errors that can be corrected with SV algorithm
```

```
35          self.t = int((self.a-3*self.g+1)/2)
36          self.L_D = self.L(self.a)
37          self.L_A = self.L(self.t + self.g)
38          self.L_C = self.L(self.a - self.t - self.g)
39          self.S = Matrix(self.L_A).transpose() * Matrix(self.L_C)
40          logger.info('Syndrom matrix:\n%s' % self.S)
41
42          # init points: P set and Q
43          points = self.C.rational_points()
44          self.Q = points[1]
45          self.P = [points[0]] + points[2:]
46          logger.info('Points in P-set\n%s' % self.P)
47
48          self.H()
49          logger.info('Parity-check matrix:\n%s' % self.H())
50          self.G()
51          logger.info('Generator matrix:\n%s' % self.G())
52          self.n = len(self.P)
53          self.k = self.G().nrows()
54          self.d = self.a - 2*self.g + 2
55          #cache for applying function to the P set
56          self.f_point_cache = {}
57          logger.info('Finished constructing code %s' % self)
58
59      def __repr__(self):
60          return 'AG [%d, %d, %d] code on Hermitian curve H_%d <%s> with D=%d*Q' % (
61                  self.n,
62                  self.k,
63                  self.d,
64                  self.m,
65                  self.C,
66                  self.a
67                  )
68
69      def L(self, a):
70          '''L(a*Q)'''
71          if a <= 2*self.g - 2:
```

```
72              raise CodeConstructionError('''The degree of Q is not greater than 2*g - 2
73                  %d <= 2*%d - 2''' % (a, self.g))
74          L_D_basis = []
75          m = self.m
76          x, y, z = self.x, self.y, self.z
77          for i in range(0, m + 1):
78              for j in range(0, a/(m+1) + 1):
79                  if i*m + j*(m+1) <= a:
80                      #print 'x^%d y^%d / z^%d' % (i, j, i+j)
81                      L_D_basis.append((x**i * y**j)/z**(i+j))
82          if len(L_D_basis) != int(a + 1 - self.g):
83              raise CodeConstructionError('''The number of functions found does not
84              satisfy the Riemann-Roch theorem
85              Found: %d, needed %d''' % (len(L_D_basis), a + 1 - self.g))
86          return L_D_basis
87
88      def _map_functions_points(self, functions, points):
89          return Matrix(
90                  self.F,
91                  [[self._apply(f, point) for point in points] for f in functions])
92
93      def H(self):
94          try:
95              return self._H
96          except AttributeError:
97              #a_dual = len(self.P) - 2 + 2*self.g - self.a
98              #if a_dual < self.a and a_dual > 2*self.g - 2:
99              # print 'using dual'
100             # self._G = self._map_functions_points(self.L(a_dual), self.P)
101             # self._H = Matrix(self.F, self._G.transpose().kernel().basis())
102             #else:
103             self._H = self._map_functions_points(self.L_D, self.P)
104             return self._H
105
106     def G(self):
107         try:
108             return self._G
```

```
109             except AttributeError:
110                 self._G = Matrix(self.F, self.H().transpose().kernel().basis())
111                 return self._G
112 
113         def encode(self, w):
114             return vector(self.C.base_ring(), w) * self.G()
115 
116         def _apply(self, f, p):
117             return f(*list(p))
118 
119         def multiply(self, v, f):
120             '''vector-function multiplication'''
121             try:
122                 f_vector = self.f_point_cache[f]
123             except KeyError:
124                 f_vector = vector(self.C.base_ring(), [self._apply(f, p) for p in self.P])
125                 self.f_point_cache[f] = f_vector
126             return v*f_vector
127 
128         def decode(self, v):
129             new_rows = []
130             for row in self.S:
131                 new_row = []
132                 for f in row:
133                     new_row.append(self.multiply(v, f))
134                 new_rows.append(new_row)
135             S = Matrix(self.C.base_ring(), new_rows)
136             logger.info('Syndrom matrix for vector %s:\n%s' % (v, S))
137             try:
138                 theta = vector(self.L_A)*vector(S.kernel().basis()[0])
139                 logger.info('Error locator: %s' % theta)
140             except IndexError:
141                 raise DecodingError
142             error_positions = []
143             logger.info('Error locator equals to zero at following points')
144             for i, p in enumerate(self.P):
145                 if self._apply(theta, p) == 0:
```

```
146                 error_positions.append(i)
147                 logger.info('P_%d' % i)
148         error_value_system = []
149         for f in self.L_D:
150             row = []
151             for i in error_positions:
152                 row.append(self._apply(f, self.P[i]))
153             row.append(self.multiply(v, f))
154             error_value_system.append(row)
155         error_value_system = Matrix(
156                 self.F,
157                 error_value_system
158                 )
159         logger.info('Error value system:\n%s' % error_value_system)
160         error_value_system = error_value_system.echelon_form()
161         logger.info('Error position and value:')
162         for i, pos in enumerate(error_positions):
163             logger.info('P_%d: %s' % (pos, error_value_system[i][-1]))
164             v[pos] -= error_value_system[i][-1]
165         logger.info('Recovered vector: %s' % v)
166         decode_g = Matrix(
167                 self.C.base_ring(),
168                 self.G().rows() + [v]
169                 ).transpose().echelon_form()
170         decoded_message = [decode_g[i][-1] for i in range(0, decode_g.ncols()-1)]
171         return decoded_message
172 
173 def test(AG):
174     print AG
175     print AG.H()
176     print '===='
177     print AG.G()
178     print 'We may add up to %d errors for SV algorithm' % AG.t
179     for j in range(0, 10):
180         w = list(AG.C.base_ring())[1]
181         message = [w**(i+j) for i in range(0, AG.k)]
182         print 'Original message:\n%s' % message
```

```
183             v=AG.encode(message)
184             print 'Encoded message:\n%s' % v
185             for i in range(0, AG.t):
186                 v[i*2] += w**(i+j)
187             print 'Message with errors:\n%s' % v
188             print 'Decoding...'
189             decoded = AG.decode(v)
190             print 'Decoded message:\n%s' % decoded
191             if message == decoded:
192                 print 'Decoded correctly'
193             else:
194                 raise DecodingError('Failed to decode correctly')
195             #if raw_input() == 'q':
196             # break
197         print 'Cached %d results for applying function to P-set' % len(AG.f_point_cache)
198 
199 class TestHermitianCode(unittest.TestCase):
200     def setUp(self):
201         self.ag_2_6 = HermitianCode(2, 6)
202         self.ag_4_40 = HermitianCode(4, 40)
203 
204     def test_2_6_H_G(self):
205         AG = self.ag_2_6
206         w = list(AG.F)[1]
207         self.assertEqual(AG.G().echelon_form(), Matrix([
208             (1, 1, 0, 0, w + 1, w + 1, w, w),
209             (0, 0, 1, 1, w, w, w + 1, w + 1)]
210             ).echelon_form()
211             )
212         self.assertEqual(AG.H().echelon_form(), Matrix([
213             (1, 1, 1, 1, 1, 1, 1, 1),
214             (0, 1, w, w + 1, w, w + 1, w, w + 1),
215             (0, 1, w + 1, w, w + 1, w, w + 1, w),
216             (0, 0, 1, 1, w, w, w + 1, w + 1),
217             (0, 0, w, w + 1, w + 1, 1, 1, w),
218             (0, 0, 1, 1, w + 1, w + 1, w, w)]
219             ).echelon_form()
```

```
220             )
221
222     def decoding_helper(self, AG):
223         w = list(AG.F)[1]
224         for j in range(0, 10):
225             message = [w**(i+j) for i in range(0, AG.k)]
226             v=AG.encode(message)
227             for i in range(0, AG.t):
228                 v[i*2] += w**(i+j)
229             decoded = AG.decode(v)
230             self.assertEqual(message, decoded)
231
232     def test_2_6_decoding(self):
233         self.decoding_helper(self.ag_2_6)
234
235     def test_4_40_decoding(self):
236         self.decoding_helper(self.ag_4_40)
237
238 if __name__ == '__main__':
239     suite = unittest.TestLoader().loadTestsFromTestCase(TestHermitianCode)
240     unittest.TextTestRunner(verbosity=2).run(suite)
241     #test(HermitianCode(2, 6))
242     #test(HermitianCode(4, 40))
```

Додаток Б. Текст програми transmission.py

```
def int_to_gf(n, F, bin_length):
    F2 = GF(2)
    def chr_to_gf2(c):
        if c == '0':
            return F2.zero()
        else:
            return F2.one()
    bin_str = bin(n)[2:].rjust(bin_length, '0')
    if F.base_ring() != F2:
        raise Exception('Field must be an extention of GF(2)')
    bits_per_gf_element = F.vector_space().dimension()
    if bin_length % bits_per_gf_element != 0:
        raise Exception('Binary length is not adjusted to F size')
    number_of_gf_elements = bin_length / bits_per_gf_element
    result = []
    for i in range(number_of_gf_elements):
        result.append(
                F(vector(F2, [chr_to_gf2(c) for c in
                    bin_str[i*bits_per_gf_element:(i+1)*bits_per_gf_element]]
                    ))
                    )
    return result

def gf_to_int(gf_list):
    bin_list = []
    for element in gf_list:
        bin_list += [str(c) for c in element._vector_()]
    return int(''.join(bin_list), 2)

def server(HC, PORT=50007, text='Hello world', p_err=0.01):
    # HC_a = HermitianCode(4, 53)
    # HC = HermitianCode(4, 37)
    # server(HC_a,text=open('/tmp/lorem.txt').read(), PORT=10001, p_err=0.3)
    import socket
```

```
35
36      HOST = ''
37      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
38      s.settimeout(5)
39      s.bind((HOST, PORT))
40      s.listen(1)
41      conn, addr = s.accept()
42      print 'Connected by', addr
43      i = 0
44      while i < len(text):
45          data = conn.recv(1024)
46          if not data:
47              break
48          message = []
49          while len(message) != HC.k:
50              try:
51                  message += int_to_gf(ord(text[i]), HC.F, 8)
52              except IndexError:
53                  message += int_to_gf(ord('~'), HC.F, 8)
54              i += 1
55          codeword = HC.encode(message)
56          for j in range(len(codeword)):
57              if random() < p_err:
58                  codeword[j] += HC.F.random_element()
59          conn.send(str(gf_to_int(codeword)))
60      conn.close()
61
62
63  def client(HC, PORT=50007):
64      # client(HC_a, PORT=10001)
65      import socket, sys
66
67      HOST = 'localhost'
68      s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
69      s.settimeout(5)
70      s.connect((HOST, PORT))
71      while 1:
```

```
72          s.send('Ok')
73          data = s.recv(1024)
74          if not data:
75              break
76          else:
77              received_word = int_to_gf(
78                      int(data),
79                      HC.F,
80                      HC.F.vector_space().dimension()*HC.n)
81              #print received_word
82              try:
83                  decoded_word = list(HC.decode(vector(HC.F, received_word)))
84                  n_gf_per_c = 8/HC.F.vector_space().dimension()
85                  for i in range(len(decoded_word)/n_gf_per_c):
86                      sys.stdout.write(chr(gf_to_int(
87                          decoded_word[i*n_gf_per_c:(i+1)*n_gf_per_c]
88                          )))
89                      sys.stdout.flush()
90              except DecodingError:
91                  for i in range(HC.k*HC.F.vector_space().dimension()/8):
92                      sys.stdout.write('_')
93                  sys.stdout.flush()
94      print
95      s.close()
```

Додаток B. Вивід сесії Maple/CASA

```
> with(casa):
‘          |__| ‘
‘          |  |          Welcome to CASA 2.5 for Maple V.5‘
‘     |   /\\|  |/\\ ‘
‘    /=\\__|  []  | ‘
‘   /           \\_       Copyright (C) 1990-2000 by Research
Institute ‘
‘   |             \\      for Symbolic Computation (RISC-Linz), the‘
‘   \\   CASA 2.5   |      University of Linz, A-4040 Linz, Austria.‘
‘    |            |‘
‘   _|     |||     |      For help type '?casa' or '?casa,<topic>'.‘
‘__/       |||     |_ ‘
Error, (in with) symbol or symbol::type expected in local list
> infolevel[‘casa/finite‘] := 19:
> C1 := finiteCurve(x^3 + y^2 + y, finiteField(4));
```

$$C1 := x^3 + y\,z^2 + y^2\,z$$

```
> H1 := GoppaPrimary(C1, "affine", 6):
```

$$G = +6\ (0 : 1 : 0)$$

$$\mathrm{D} = [0,\ 0,\ 1] + [0,\ 1,\ 1] + [1,\ \alpha,\ 1] + [1,\ \alpha^2,\ 1] + [\alpha,\ \alpha,\ 1] + [\alpha,\ \alpha^2,\ 1] + [\alpha^2,\ \alpha,\ 1] + [\alpha^2,\ \alpha^2,\ 1]$$

$$L(G) = \mathrm{span}(1,\ \frac{y}{z},\ \frac{y^2}{z^2},\ \frac{x}{z},\ \frac{x\,y}{z^2},\ \frac{x^2}{z^2})$$

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & \alpha & \alpha^2 & \alpha^2 & 1 & 1 & \alpha \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & \alpha^2 & \alpha^2 & \alpha & \alpha \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \end{bmatrix}$$

$$[n,\ k,\ d_\Gamma] = 8,\ 2,\ 6$$

```
> c :=GoppaEncode([0, 1], H1);
```

$$c := [0,\ 0,\ 1,\ 1,\ \alpha,\ \alpha,\ \alpha^2,\ \alpha^2]$$

```
> c[2] := 1; c[8]:= 0;
```

$$c_2 := 1$$

$$c_8 := 0$$

```
> SV := GoppaPrepareSV(H1):
```

$$A = +3\ (0 : 1 : 0)$$

$$C = +3\ (0 : 1 : 0)$$

$$\psi = [1, \frac{y}{z}, \frac{x}{z}]$$

$$\chi = [1, \frac{y}{z}, \frac{x}{z}]$$

$$\psi\ *\ \chi = \begin{bmatrix} 1 & \frac{y}{z} & \frac{x}{z} \\ \frac{y}{z} & \frac{y^2}{z^2} & \frac{x\,y}{z^2} \\ \frac{x}{z} & \frac{x\,y}{z^2} & \frac{x^2}{z^2} \end{bmatrix}$$

```
> GoppaDecode(c, H1, SV);
```

$$S = \begin{bmatrix} \alpha & \alpha^2 & \alpha \\ \alpha^2 & 0 & 1 \\ \alpha & 1 & 1 \end{bmatrix}$$

$$Nullspace(S) = ([\alpha,\ \alpha,\ 1])$$

$$error\ locations = \{2,\ 3,\ 8\}$$

$$Error\ Value\ System: \begin{bmatrix} 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 \\ 1 & \alpha^2 & \alpha \\ 0 & 1 & \alpha^2 \\ 0 & \alpha & \alpha \\ 0 & 1 & \alpha \end{bmatrix} * e = \begin{bmatrix} \alpha \\ \alpha^2 \\ 0 \\ \alpha \\ 1 \\ 1 \end{bmatrix}$$

$$e = \left[1,\ 0,\ \alpha^2\right]$$

$$c = \left[0,\ 0,\ 1,\ 1,\ \alpha,\ \alpha,\ \alpha^2,\ \alpha^2\right]$$

[0, 1]