RESEARCH REPORT

Université de Limoges

Faculté des Sciences et Techniques
M1 CRYPTIS – Information Security

ERROR CORRECTING CODES BASED ON CHINESE REMAINDER THEOREM

By : Imane ELACERI

Supervised by :
Dr. Ruatta Olivier & Camille Garnier

2024 /2025

1 Introduction:

The Chinese Remainder Theorem (CRT) is a fundamental result in number theory, tracing its origins back to the work of the Chinese mathematician Sun Zi*.
The theorem, which provides conditions under which a system of congruences has a unique solution, was later formalized and generalized by European mathematicians such as Euler and Gauss. It plays a crucial role in various mathematical disciplines, including algebra, cryptography, and coding theory.

In this research project, I began by studying the classical statement of the Chinese Remainder Theorem, exploring its proof and applications in different Euclidean rings. The CRT ensures that under appropriate conditions, elements of a ring can be uniquely reconstructed from their residues in a system of quotient rings. This property has significant implications in coding theory, particularly in the construction of error-correcting codes.

In fact, error-correcting codes based on the Chinese Remainder Theorem (CRT codes) leverage their structure to design efficient encoding and decoding algorithms. These codes exploit the decomposition of certain rings into direct products, allowing for redundancy and error detection mechanisms. A key part of this study involved examining their properties and their potential for practical applications.

My contribution to this project was to extend the study of these codes to a new algebraic setting, namely the euclidean ring of Gaussian integers Z[i]. By analyzing how the structural properties of this Euclidean domain influence the construction and performance of CRT-based codes, this research aims to provide new insights into their potential advantages and limitations in this extended framework.

2 Chinese Remainder Theorem and Applications:[2]

The Chinese Remainder Theorem (CRT) states that if $n_1, n_2, \ldots, n_p$ are integers that are pairwise coprime, then the system of p congruences:

$$X \equiv a_i \pmod{n_i}, \quad \text{for } i = 1, \ldots, p.$$

has a unique solution satisfying the given system of congruences, namely:

$$X = \sum_{i=1}^{p} a_i\, m_i\, y_i \pmod{M},$$

where $M = n_1\, n_2 \cdots n_p$, and m_i, y_i are chosen so that $m_i = \frac{M}{n_i}$ and $y_i \equiv m_i^{-1} \pmod{n_i}$.

Definition

Let $a, b \in A$ where A is a principal ring. We say that a and b are coprime (or relatively coprime) if their greatest common divisor $\gcd(a, b)$ is a unit in A.

There is a general statement of the Chinese Remainder Theorem over principal rings which is:

[0]* The theorem was first formulated in Sun Zi's Mathematical Classic around the 3rd century AD. .

Theorem

Let A be a principal ring, and let $\{a_j\}_{1\leq j\leq r}$ with $r\geq 2$ be nonzero and non-invertible elements of A. We denote $a=\prod_{j=1}^r a_j$, and for each j, let π_j be the canonical surjection $A\to A/\langle a_j\rangle$. If the elements a_j are pairwise coprime, the map $\varphi : x\in A\longmapsto (\pi_j(x))_{1\leq j\leq r}\in\prod_{j=1}^r A/\langle a_j\rangle$ induces a ring isomorphism:

$$\overline{\varphi} : \frac{A}{\left\langle \prod_{j=1}^r a_j\right\rangle} \longrightarrow \prod_{j=1}^r \frac{A}{\langle a_j\rangle}.$$

The inverse of this isomorphism is given by : $\varphi^{-1}\big((\pi_j(x))_{1\leq j\leq r}\big)=\sum_{i=1}^r x_i\,u_i\,b_i(\text{mod}\prod_{j=1}^r a_j)$ where $b_j=\frac{a}{a_j}$ and $u_j\in A$ satisfy $u_jb_j\equiv 1\pmod{a_j}$ with $\sum_{j=1}^r u_jb_j=1$.

Proof:

The application $\varphi : x\in A\longmapsto (\pi_j(x))_{1\leq j\leq r}\in\prod_{j=1}^r A/\langle a_j\rangle$ is a ring morphism.

Indeed, since each π_j is a ring morphism, it preserves addition, multiplication, and the identity element. By definition, $\ker(\varphi)$ consists of all elements in A that are mapped to 0 in each quotient $A/\langle a_j\rangle$, therefore: $\ker(\varphi)=\{x\in A\mid x\equiv 0\pmod{a_j}\text{ for all }j\}$.

This means that $x\in ker(\varphi)\iff x$ is a multiple of a, so $\ker(\varphi)=\langle a\rangle$.

By applying the First Isomorphism Theorem we obtain the following isomorphism:

$$\overline{\varphi} :\quad A/\ker(\varphi)=A/\langle a\rangle\cong \text{Im}(\varphi)\subseteq\prod_{j=1}^r A/\langle a_j\rangle.$$

- Let's prove the inverse isomorphism, and that φ is surjective now:

We have $b_i=\frac{a}{a_i}=\prod_{\substack{j=1\\ j\neq i}}^r a_j$, therefore $\gcd(b_1,b_2,\dots,b_r)=1$.

Indeed, if this was not the case, the b_j would share a common divisor, and by that, at least two of the a_j would also share that divisor, contradicting their pairwise coprimality.

We can then use Bézout's theorem as follows: $\exists(u_i)_{1\leq i\leq r}$ such that $\sum u_ib_i=1$.

We know that for all $j\in\{1,\dots,r\}$, b_i is a multiple of a_j when $j\neq i$, so:

$$1=\pi_j(1)=\pi_j(\sum_{i=1}^r u_ib_i)=\sum_{i=1}^r\pi_j(u_ib_i)=\sum_{i=1}^r\pi_j(u_i)\pi_j(b_i)=\pi_j(u_j)\pi_j(b_j)$$

Therefore, $\pi_j(b_j)$ is invertible in $\frac{A}{\langle a_j\rangle}$, and its inverse is $\pi_j(u_j)$.

Now, for $(\pi_j(x_j))_{1\leq j\leq r}$ given in $\prod_{j=1}^r\frac{A}{\langle a_j\rangle}$, by setting $x=\sum_{i=1}^r x_iu_ib_i$.

We have, $\forall j\in\{1,\dots,r\}:\pi_j(x)=\pi_j(x_j)\pi_j(u_j)\pi_j(b_j)=\pi_j(x_j)$, since $\pi_j(u_j)\cdot\pi_j(b_j)=1$.

Thus, we have explicitly constructed a preimage for every element of $\prod_{j=1}^r A/\langle a_j\rangle$, ensuring that φ is surjective. This proves that: $\text{Im}(\varphi)=\prod_{j=1}^r A/\langle a_j\rangle$. Therefore the inverse of the isomorphism is indeed given by:

$$\begin{aligned}\overline{\varphi}^{-1}:\prod_{j=1}^r\frac{A}{\langle a_j\rangle}&\to \underline{A/\langle\prod_{j=1}^r a_j\rangle}\\ (\pi_j(x_j))_{1\leq j\leq r}&\mapsto\sum_{i=1}^r x_iu_ib_i.\end{aligned}$$

□

In the sections below, we will explore how this theorem is applied to different euclidian rings and how it forms the basis for constructing what we call Chinese Remainder Codes.

Example in the polynomial ring $\mathbb{F}_5[X]$:

Consider the finite field $\mathbb{F}_5$ and the polynomials: $f(X) = X^2 + 2X + 3$, $g(X) = X^3 + 3X + 4$. Since $f(X)$ and $g(X)$ are coprime in $\mathbb{F}_5[X]$, the CRT applies.
We solve the system:
$$\begin{cases} F(X) & \equiv X + 2 \pmod{f(x)} \\ F(X) & \equiv X^2 + 4X + 3 \pmod{g(x)} \end{cases}$$
Since $\gcd(f(X), g(X)) = 1$, by Bézout's theorem, there exist polynomials $U(X), V(X)$ such that:
$$U(X)f(X) + V(X)g(X) = 1.$$
Using the Extended Euclidean Algorithm in $\mathbb{F}_5[X]$, we find Bézout's coefficients:
$$U(X) = 3X^2, V(X) = 2X + 4.$$
We can now construct the solution using the CRT formula:
$$F(X) = A(X)V(X)g(X) + B(X)U(X)f(X) \pmod{f(X)g(X)},$$
where: $A(X) = X + 2$ and $B(X) = X^2 + 4X + 3$,
After substituting values, we find:
$$F(X) = (X + 2)(2X + 4)(X^3 + 3X + 4) + (X^2 + 4X + 3)(3X^2)(X^2 + X + 1).$$
Expanding and simplifying modulo $f(X)g(X)$, we obtain:
$$\boxed{F(X) = 4X^4 + 4X^3 + 3X^2 + 2X + 4 \pmod{(X^2 + 2X + 1)(X^3 + 3X + 4)}.}$$
The Chinese Remainder Theorem guarantees that this solution is unique.
This solution has been verified using SageMath, and the corresponding code snippet can be found in the appendix (see figure 1 in appendix).

Example in the Gaussian ring $\mathbb{Z}[i]$:

Definition

An integral domain A is said to be Euclidean if there exists a valuation function $N : A \setminus \{0\} \to \mathbb{N}$ that allows the execution of the Euclidean algorithm, meaning that for all $a, b \in A$ with $b \neq 0$, there exist $q, r \in A$ such that: $a = qb + r$, with $N(r) < N(b)$ or $r = 0$.

In $\mathbb{Z}[i]$, the valuation function is defined as the quadratic norm: $N(a + bi) = a^2 + b^2$.
This norm satisfies the properties of an euclidian function: positivity and compatibility with euclidian division, therefore the Gaussian integers ring is indeed euclidian.

Property

Any Euclidean domain is a principal ideal domain (PID).

Proof: Let A be a Euclidean domain, meaning there exists a *valuation function* $v : A \setminus \{0\} \to \mathbb{N}$ such that for any $a, b \in A$ with $b \neq 0$, there exist $q, r \in A$ s.t: $a = qb + r$, where $r = 0$ or $v(r) < v(b)$.

Now, let I be a nonzero ideal of A. Since $I \neq \{0\}$, there exists a nonzero element $a_0 \in I$. Let's chose $d \in I$ such that $v(d)$ is minimal among all nonzero elements of I.
Applying the Euclidean division to a and d, we can write: $a = qd + r, \quad$ where $r = 0$ or $v(r) < v(d)$.
Since I is an ideal, both $a \in I$ and $qd \in I$, so their difference $r = a - qd$ is also in I. By minimality of d, we must have $r = 0$, so $a = qd$. Thus, $I = \langle d \rangle$, proving that every ideal in A is principal.
Therefore, every Euclidean domain is a principal ideal domain. $\square$

With that being said, we can apply the CRT in this ring.
Concretely, to find the gcd of two Gaussian integers $z_1, z_2 \in \mathbb{Z}[i]$, we use a process similar to the Euclidean algorithm in $\mathbb{Z}$, but adapted for the structure of $\mathbb{Z}[i]$.
Let's first define division in the ring of Gaussian integers.
Given two Gaussian integers $z_1 = a + ib$ and $z_2 = c + id$, we divide z_1 by z_2 as follows:

$$\frac{a+ib}{c+id} = \frac{(a+ib)(c-id)}{(c+id)(c-id)} = \frac{ac+bd+i(bc-ad)}{c^2+d^2}.$$

The result is generally a complex number with rational real and imaginary parts. We then round both parts to the nearest integers:

$$q = \text{round}\left(\frac{ac+bd}{c^2+d^2}\right) + i \cdot \text{round}\left(\frac{bc-ad}{c^2+d^2}\right)$$

This ensures that $q \in \mathbb{Z}[i]$. The remainder is then computed as:

$$r = z_1 - qz_2 = (a+ib) - q(c+id). \text{ (We indeed have } N(r) < N(z_2))$$[1]

We repeat this process, replacing z_1 with z_2 and z_2 with r, until the remainder is zero.
Once we obtain $\gcd(z_1, z_2)$, we check whether it is a unit $(\pm 1, \pm i)$. If so, then z_1 and z_2 are coprime; otherwise, they are not.
Now let's apply the CRT in this ring. For that let's consider the Gaussian primes:

$$\pi_1 = 3 + 2i, \qquad \pi_2 = 4 + i \qquad \text{and} \qquad \pi_3 = 5 + 3i.$$

Since these elements are pairwise coprime in $\mathbb{Z}[i]$, (See figure 2 in appendix) the CRT applies and guarantees that the following system has a unique solution modulo $\prod_{i=1}^{3} \pi_i$:

$$\begin{cases} X & \equiv 1 + 2i \pmod{3 + 2i} \\ X & \equiv -2 + 5i \pmod{4 + i} \\ X & \equiv 7 - 3i \pmod{5 + 3i} \end{cases}$$

The modulus is: $\qquad M = \prod_{i=1}^{3} \pi_i = 17 + 85i$

We then calculate m_i for i between 1 and 3 such that: $m_i = \frac{M}{\pi_i}$ and we find the following:

$$m_1 = 17 + 17i, \qquad m_2 = 9 + 19i, \qquad m_3 = 10 + 11i$$

Now we need to calculate the inverse of each m_i modulo π_i. We find the following inverses:

$$u_1 = -1 - i, \qquad u_2 = 1, \qquad \textit{and} \qquad u_3 = -2 - i$$ (See figure 3 in appendix)

Now we can construct the final solution by using the CRT formula:

$$X = x_1 u_1 m_1 + x_2 u_2 m_2 + x_3 u_3 m_3 \pmod{\pi_1 \pi_2 \pi_3},$$

[1]During the first semester, I studied the norm $\mathcal{N}(z) = |z|^2$ and verified that it defines an Euclidian function on $\mathbb{Z}[i]$. One of the questions asked by Dr. Ruatta Olivier was whether an Euclidean function could fail to satisfy the triangle inequality.I proposed the norm itself as a counterexample, but Dr. Ruatta, explained that composing it with the square root yields a positive real-valued function which indeed satisfies the triangle inequality. This question therefore remains open for me, and I hope to explore it further.

$$\text{where: } x_1 = 1+2i, \qquad x_2 = -2+5i \qquad \textit{and} \qquad x_3 = 7-3i$$

By substituting values we find:

$$X = (1+2i)(-1-i)(17+17i)+(-2+5i)(9+19i)+(7-3i)(-2-i)(10+11i) \pmod{(3+2i)(4+i)(5+3i)}.$$

After expanding and simplifying modulo $(3+2i)(4+i)(5+3i)$, we obtain:

$$X = -204-224i \equiv 17-3i \pmod{(17+85i)}.$$

The solution is verified using Sagemath, (See figure 4 in appendix)

3 Chinese Remainder Codes: Construction and Properties:

An error-detecting code or an error-correcting code aims at detect or correct errors that might occur when a message is transmitted through a noisy channel. The key point in coding theory is to add an extra information to a message, called redundancy.

3.1 Definition of an Error-Correcting Code and Its Parameters:[1]

Definition

The Hamming weight of a vector $x \in \mathbb{F}_q^n$, denoted by $w_H(x)$, is the number of nonzero symbols in x. Formally, it is defined as: $w_H(x) = |\{i, 1 \leq i \leq n \mid x_i \neq 0\}|$.

Definition

The Hamming distance between two vectors $x, y \in \mathbb{F}_q^n$, denoted by $d_H(x,y)$, is the number of positions where they differ. It is formally defined as: $d_H(x,y) = |\{i, 1 \leq i \leq n \mid x_i \neq y_i\}|$.
It's a distance in the mathematical sense of it as it verifies non-negativity, symmetry and triangle-inequality.

Definition

An error-correcting code $\mathcal{C}$ is a subset of $\mathbb{F}_q^n$ equipped with a distance function, typically the Hamming distance. A code is usually characterized by its parameters (n,k,d):

- n: the length of each codeword (number of symbols in each transmitted message).
- k: the logarithm of the code cardinality, i.e., $k = \log_q |C|$ (number of information symbols needed to distinguish codewords).
- d: the minimum Hamming distance between distinct codewords.

- The minimal distance d plays a crucial role, as a code can detect up to $d-1$ errors and correct up to $\lfloor (d-1)/2 \rfloor$ errors.
- A Maximum Distance Separable (MDS) code is a linear error-correcting code that achieves the Singleton bound: $\qquad d_{\min} = n-k+1$.

3.2 Linear Codes and Their Importance:

A code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is linear if it forms a subspace of $\mathbb{F}_q^n$. Linear codes are particularly useful because they allow efficient encoding and decoding using algebraic techniques.
In this case, the parameter k represents the *dimension* of the code, as it corresponds to the number of basis vectors needed to span the subspace $\mathcal{C}$. It also represents the length of the message m before

encoding.

3.2.1 Generator Matrix in Linear Codes:

Linear codes have the advantage of being defined by a fundamental matrix: the generator matrix G . This matrix allows for efficient encoding .
For linear codes, the minimal distance simplifies to: $d_{\min} = \min_{x \in \mathcal{C}, x \neq 0} w_H(x),$

Definition

Let C be a linear (n,k)-code over a finite field $\mathbb{F}_q$. A generator matrix G is a $k \times n$ matrix whose rows form a basis of the code C. Any codeword c can be obtained from a message $m \in \mathbb{F}_q^k$ by the linear transformation: $c = m \cdot G$.

3.3 Chinese Remainder Codes:

3.3.1 Construction of Chinese Remainder Codes:

Chinese Remainder Codes leverage the Chinese Remainder Theorem (CRT) to construct error-correcting codes based on modular arithmetic. Let $M_0, \ldots, M_{n-1} \in \mathbb{F}_q[x]$ be pairwise coprime polynomials.
We define the quotient ring: $R = \frac{\mathbb{F}_q[x]}{\langle M_0 M_1 \ldots M_{n-1} \rangle}$.
Using the CRT, this ring decomposes into: $R \cong \frac{\mathbb{F}_q[x]}{\langle M_0 \rangle} \times \cdots \times \frac{\mathbb{F}_q[x]}{\langle M_{n-1} \rangle}$.
We can define the code $\mathcal{C}$ as: $\mathcal{C} = \varphi(\mathbb{F}_q[x]_{<k})$, where $\mathbb{F}_q[x]_{<k}$ is the space of polynomials of degree less than k and φ maps a polynomial P to its residues modulo M_i: $\varphi(P) = (\pi_0(P), \pi_1(P), \ldots, \pi_{n-1}(P))$.

3.3.2 Proof of Linearity of Chinese Remainder Codes:

$\mathbb{F}_q[x]_{<k}$ is a vector space and φ is a linear transformation. By the fundamental property of vector spaces, the image of a vector space under a linear map is also a vector space. Therefore, $\varphi(\mathbb{F}_q[x]_{<k})$ is a subspace, proving that $\mathcal{C}$ is a linear code.

3.4 Minimal Distance Bound of CRT codes:

Since CRT code is defined by mapping polynomials to their modular residues, the Hamming weight of a codeword depends on how many of the residue components are nonzero.
Let $P(x) \neq 0$ be a nonzero polynomial in $\mathbb{F}_q[x]_{<k}$. Its residues $\pi_i(P) = P \mod M_i$ determines the weight of the codeword. Since P is of degree at most k and each M_i is a polynomial of degree at least 1, P can have at most $k-1$ zero residues (modulo different M_i). Thus, at least $n-k+1$ of its residues must be nonzero. Therefore, the minimal distance satisfies: $d_{\min} \geq n-k+1$.
By Singleton's bound, we know that: $d_{\min} \leq n-k+1$. From this, we deduce that: $d_{\min} = n-k+1$. Thus, the CRT codes is an MDS. MDS codes are useful since they offer the best possible trade-off between code length, message length, and error correction thanks to the fact that they meet the Singleton's bound with equality

3.5 Encoding and Decoding for Chinese Remainder Codes:

Since Chinese remainder Codes are linear, we can compute a Generator Matrix.

Let's describe the general procedure for constructing G over the polynomial ring $\mathbb{F}_q[X]$ given a set of pairwise coprime moduli.
Let the moduli be: $M_1(X), M_2(X), \dots, M_n(X)$ where each $M_i(X)$ is a monic polynomial in $\mathbb{F}_q[X]$ of degree $d_i \geq 1$.
The code length n corresponds to the number of moduli. The message length is $k < n$. The generator matrix G will therefore have size: $k \times n$.

The general form of the generator matrix is:

$$G = \begin{pmatrix} & 1 & \cdots & \cdots & & 1 \\ & X & \cdots & \cdots & & X \\ X^2 & \bmod M_1 & \cdots & \cdots & X^2 & \bmod M_n \\ & \vdots & \vdots & \ddots & & \vdots \\ X^{k-1} & \bmod M_1 & \cdots & \cdots & X^{k-1} & \bmod M_n \end{pmatrix}$$

Let's apply on an example.Let's consider the polynomial ring $\mathbb{F}_{11}[X]$ and the modulis:
$f_1 = X+1, f_2 = X+2, f_3 = X^2+7X+1, f_4 = X^2+8, f_5 = X^3+2X^2+4X+9, f_6 = X^2+5X+2$
Here $n = 6$ and using SageMath (See figure 5 in appendix), I checked that the modulis are pairwise coprime , then I computed the generator matrix with a message length $k = 3$. The obtained matrix is:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 9 & X & X & X & X \\ 1 & 4 & 4X+10 & 3 & X^2 & 6X+9 \end{pmatrix}$$

The encoding process of a message m, is done by using the standard encoding formula: $c = mG$
To illustrate the encoding process, consider the message vector: $m = \begin{bmatrix} 3 & 2 & 7 \end{bmatrix}$ that corresponds to: $3 + 2X + 7X^2$.
Using the generator matrix, We compute the encoded codeword $\mathbf{c}$ (See figure 6 in appendix) as follows:

$$c = mG = \begin{bmatrix} 3 & 2 & 7 \end{bmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 9 & X & X & X & X \\ 1 & 4 & 4X+10 & 3 & X^2 & 6X+9 \end{pmatrix}$$

Thus, the encoded codeword is: $\mathbf{c} = \begin{bmatrix} 8 & 5 & 8X+7 & 2X+2 & 7X^2+2X+3 & 0 \end{bmatrix}$
This demonstrates how to use the generator matrix to encode a message in $\mathbb{F}_5[X]$.

The decoding process of a received message y, could be done by using the Majority Voting Decoding technique: Given a set of pairwise coprime moduli $M_1(X), M_2(X), \dots, M_n(X)$ over the finite field $\mathbb{F}_q$, a message polynomial $m(X) = m_0 + m_1X + \cdots + m_{k-1}X^{k-1}$ is encoded as a codeword:
$\mathbf{c} = (m(X) \bmod M_1(X), \ \dots, \ m(X) \bmod M_n(X))$. Suppose that during transmission, the received word is $\mathbf{y} = (y_1, y_2, \dots, y_n)$, where some of the y_i values may differ from the original codeword components due to noise or errors.
The Majority voting decoding proceeds by first selecting all possible subsets of k out of the n received components. Each such subset corresponds to k moduli and k received values. Then, for each subset, we use the Chinese Remainder Theorem (CRT) to reconstruct a candidate polynomial $\tilde{m}(X)$ that satisfies: $\tilde{m}(X) \equiv y_i \pmod{M_i(X)}$ for the selected M_i.
After that we collect all such candidate polynomials obtained from the $\binom{n}{k}$ subsets and then we apply a majority vote: the polynomial that appears most frequently among the candidates is declared as the decoded message.This technique assumes that the number of errors is small enough so that the correct

polynomial appears more frequently than any incorrect candidate. This is typically guaranteed when: $n \geq 2t + k$, where t is the number of errors that can be corrected.[2]
In our previous example, for $n = 6$ and $k = 3$, we can correct up to one error ($(n - k)/2 \geq t$)
Let's simulate an error and try to use the majority vote process to see if we can recover the correct message which was $m = \begin{bmatrix} 3 & 2 & 7 \end{bmatrix}$ that corresponds to: $3 + 2X + 7X^2$. I coded the process using Sagemath, and the decoded message obtained is indeed correct. (See figure 7 in appendix)

4 My Contribution to the Research Project:

When discussing error-correcting codes based on the Chinese Remainder Theorem , we usually think of them in $\mathbb{Z}$ or in $\mathbb{K}[\mathbb{X}]$. However, the Chinese Remainder Theorem is not restricted to the integers or polynomials, it applies to any ring where we can define a notion of coprimality of ideals and where the ring is principal.

The good news is that $\mathbb{Z}[i]$ (the ring of Gaussian integers) is an Euclidean ring and therefore a principal ring as showed earlier. As a result, we also have a Chinese Remainder Theorem for suitably chosen ideals in $\mathbb{Z}[i]$ as we saw in a previous example. This opens the possibility of adapting the idea of CRT-based error-correcting codes to the setting of Gaussian integers especially that I thought that this offers some sort of 'two-dimensional representation'.

Therefore as a contribution into this research project, apart from coding CRT, generator matrix, encoding and decoding process with Majority vote technique using SageMath, I tried to dive into CRT codes in this ring and explore it's properties, potential advantages and limitations:

4.1 Linearity of Chinese Remainder Codes in $\mathbb{Z}[i]$:

It is easy to check linearity via the usual CRT homomorphism as done before:

$$\varphi\colon \mathbb{Z}[i] \longrightarrow \mathbb{Z}[i]/\langle \alpha_1 \rangle \times \cdots \times \mathbb{Z}[i]/\langle \alpha_n \rangle, \quad z \longmapsto \big(z \bmod \alpha_1, \ \ldots, \ z \bmod \alpha_n\big).$$

We have: $\varphi(z_1 + z_2) = \varphi(z_1) + \varphi(z_2)$ and for $\lambda \in \mathbb{Z}[i]$: $\varphi(\lambda z) = \lambda \cdot \varphi(z)$.

4.2 Definition of a Module and a Submodule

Before addressing the structure of Chinese Remainder Codes over $\mathbb{Z}[i]$, it is essential to clarify the fundamental concepts of modules and submodules, as they differ from vector spaces.

Definition

A module over a ring R is a generalization of a vector space, where scalars come from R instead of a field. More formally, an R-module M is an abelian group equipped with a scalar multiplication $R \times M \to M$ satisfying the following properties for all $r, s \in R$ and $x, y \in M$:

- $r(x + y) = rx + ry$ (distributivity over addition in M).
- $(r + s)x = rx + sx$ (distributivity over addition in R).
- $r(sx) = (rs)x$ (associativity of scalar multiplication).
- $1x = x$ (multiplication by the identity of R).

When R is a field, an R-module is simply a vector space.

[2] The inequality $n \geq 2t + k$ is a *sufficient* condition to ensure correct decoding by majority vote. It guarantees that the number of subsets of size k containing only correct residues exceeds the number of subsets that include at least one error. Hence, the correct message polynomial will appear more frequently than any incorrect candidate in the majority voting process.

Definition

A submodule N of an R-module M is a subset of M that is closed under addition and scalar multiplication. That is, N satisfies:

- $x, y \in N \Rightarrow x + y \in N$ (closure under addition).
- $r \in R, x \in N \Rightarrow rx \in N$ (closure under scalar multiplication).

Thus, a submodule is analogous to a subspace in vector space theory but defined in a more general algebraic setting.

My observation Before Studying $\mathbb{Z}[i]$

Before delving into the properties of CRT codes over $\mathbb{Z}[i]$, I observed a key difference compared to the case of polynomials over a field $\mathbb{F}_q[X]$:
In $\mathbb{F}_q[X]$, the quotient spaces $\mathbb{F}_q[X]/\langle M_i(X)\rangle$ naturally form vector spaces over $\mathbb{F}_q$, making the codes subspaces. This allows us to define generator matrices and use classical linear algebra techniques. Indeed when applying the Chinese Remainder Theorem, each residue class corresponds to a polynomial of degree $< \deg(M_i)$. So, when lifting back from the residue system to $\mathbb{F}_q[X]/\langle M_1 M_2 \dots M_k\rangle$, we end up with polynomials of degree $< \operatorname{lcm}(\deg(M_i))$. We therefore restrict to polynomials of degree $\leq k$ (where k is the number of moduli), which forms a subspace.

However, in $\mathbb{Z}[i]$, the quotient structures $\mathbb{Z}[i]/\langle \pi_j\rangle$ are not vector spaces because $\mathbb{Z}$ is a ring, not a field. Instead, these quotient spaces form $\mathbb{Z}$-modules, meaning that CRT codes in this setting should be considered as submodules rather than subspaces. In fact, the CRT still gives a decomposition of the ring as: $\mathbb{Z}[i]/\langle \pi_1\rangle \times \mathbb{Z}[i]/\langle \pi_2\rangle \times \cdots \times \mathbb{Z}[i]/\langle \pi_k\rangle$. But when lifting back, we do not get a vector space structure (since $\mathbb{Z}$ is not a field). Instead, we need to choose a good submodule so that we land inside a well-structured submodule of $\mathbb{Z}[i]/\langle \pi_1 \pi_2 \dots \pi_k\rangle$.

The key question is then: What submodule should we pick so that we land inside a structured space inside $\mathbb{Z}[i]/\langle \pi_1 \pi_2 \dots \pi_k\rangle$?

4.3 Refining the Choice of a Submodule: From Ideals to Structured Modules

First Attempt: Using a Proper Ideal

My first idea was to restrict valid messages to a proper ideal in $\mathbb{Z}[i]$, such as: $I = (\beta) = \{\beta w \mid w \in \mathbb{Z}[i]\}$, where $\beta \in \mathbb{Z}[i]$ is neither 0 nor a unit. This ideal forms a strict subset of $\mathbb{Z}[i]$, and the hope was that its image under the CRT map would provide a naturally redundant code.
However, this approach turned out to be flawed. In fact, it may happen that β shares nontrivial divisors with one of the π_j. In such cases, all elements of I will reduce to zero modulo that modulus, collapsing one entire component of the CRT product. Therefore, the image of I under CRT may lose entire dimensions and become structurally trivial. This makes the code ineffective for encoding meaningful data.

A Second Insight: Valuation-Based Constraints

Reflecting on the case of polynomials over $\mathbb{F}_q[X]$, I noticed that redundancy is introduced by restricting the degree: $\deg(f) < k \Rightarrow f \in \mathbb{F}_q[X]_{<k}$. This is a constraint on the valuation in $\mathbb{F}_q[X]$ which is the degree. Inspired by this, I considered an analogue in $\mathbb{Z}[i]$, where the natural valuation is the norm:

$$\mathrm{N}(z) = z\bar{z} = a^2 + b^2, \quad \text{for } z = a + bi.$$

Thus, I proposed to restrict messages to those lying inside a "ball" s.t : $\mathcal{M}_r := \{z \in \mathbb{Z}[i] \mid \mathrm{N}(z) \leq r\}$. This set is geometrically intuitive and finite, but algebraically unsound: Indeed, it is not closed under addition: $z_1, z_2 \in \mathcal{M}_r$ does not imply $z_1 + z_2 \in \mathcal{M}_r$. It is not closed under scalar multiplication either: αz may fall outside the ball. Therefore, $\mathcal{M}_r$ is not a submodule, and its image under the CRT map lacks algebraic structure.

Third Attempt: Analogy with Classical Linear Codes

In 'classical' coding theory, a linear code $\mathcal{C} \subset \mathbb{F}_q^n$ can be defined as the kernel of a matrix: $\mathcal{C} = \{x \in \mathbb{F}_q^n \mid Hx^\top = 0\}$[3] This definition introduces linear constraints that relate the components of each codeword. For example, certain positions in the codeword may be linear combinations of the others, ensuring redundancy and enabling error correction.

Since the CRT space $\mathbb{Z}[i]/\langle\pi_1\rangle \times \cdots \times \mathbb{Z}[i]/\langle\pi_n\rangle$ resembles a product of finite modules, one might ask whether similar constraints could be imposed in this context.

An approach taht I considered is to let one component (x_1 for example) to be free, and define the others via congruences: $x_j \equiv \lambda_j x_1 \mod \pi_j$ for $j = 2, \ldots, n$with $\lambda_j \in \mathbb{Z}[i]$.
This imposes a linear-like dependency across coordinates, analogous to parity-check constraints in classical codes. While this construction appears promising and well-structured to me, **I am not yet fully certain** whether it provides a robust and effective framework for encoding, decoding, and defining a well-behaved submodule. This remains a possible direction for further theoretical exploration.

5 Conclusion

This project has led me through a rich mathematical journey, bridging the Chinese Remainder Theorem with the theory of error-correcting codes in both classical and novel algebraic settings. Starting from the well-understood case over $\mathbb{F}_q[X]$, I explored a deeper structural challenge that arises when moving into the realm of Gaussian integers $\mathbb{Z}[i]$.

I investigated how CRT-based codes can be constructed and structured in this new setting, paying particular attention to the algebraic structure of submodules. While I did not yet study encoding and decoding algorithms over $\mathbb{Z}[i]$ itself, I was deeply inspired by the question of how decoding could be extended beyond majority vote. This led me to discover the existence of the key equation decoding technique [3], a path I would like to explore further in future work.

Throughout this exploration, I have also questioned whether Hamming distance is the only meaningful metric for codes in such settings. This led me to learn about the Manhattan or Manheim distance, a geometry-inspired metric more suited to Gaussian integers, and which I am eager to investigate further. More broadly, this project helped me understand that research is a long, yet passionate and rewarding process, often a series of failures, or imperfect ideas, but also a real delight in formulating hypotheses and attempting to prove (or disprove) them. This process of mathematical investigation, of trying, erring, refining, and discovering, has become for me not just a method of work, but a source of joy.

This adventure has been all the more enriching thanks to the presence of Camille Garnier who guided me and supported me along the way. I am especially grateful to her for her insightful feedback and encouragement. I would also like to express my sincere gratitude to Dr.Ruatta Olivier for proposing this fascinating topic and for all his support.

[3]The matrix H is called a parity-check matrix. It defines the set of linear relations that codewords must satisfy. Formally, for a code of dimension k, H is a matrix in $\mathbb{F}_q^{(n-k)\times n}$.

References

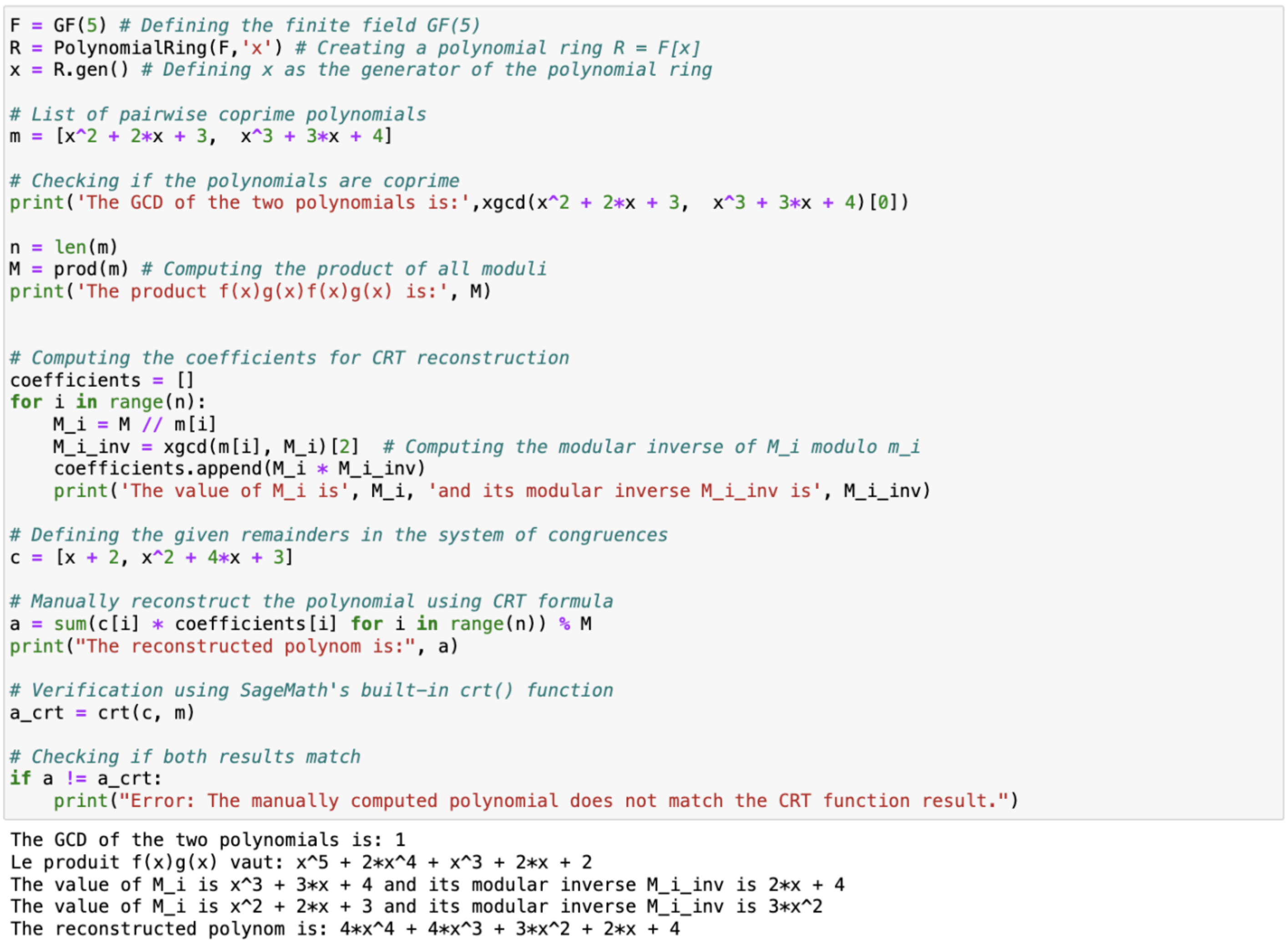
[1] Alain Couvreur. Introduction to Coding Theory.

[2] Jean-Étienne Rombaldi. Algèbre et géométrie pour l'Agrégation de Mathématiques Nouvelle version corrigée avec compléments.

[3] Jiun-Hung Yu and Hans-Andrea Loeliger. On Polynomial Remainder Codes, January 2012. arXiv:1201.1812 [cs].

Appendix:

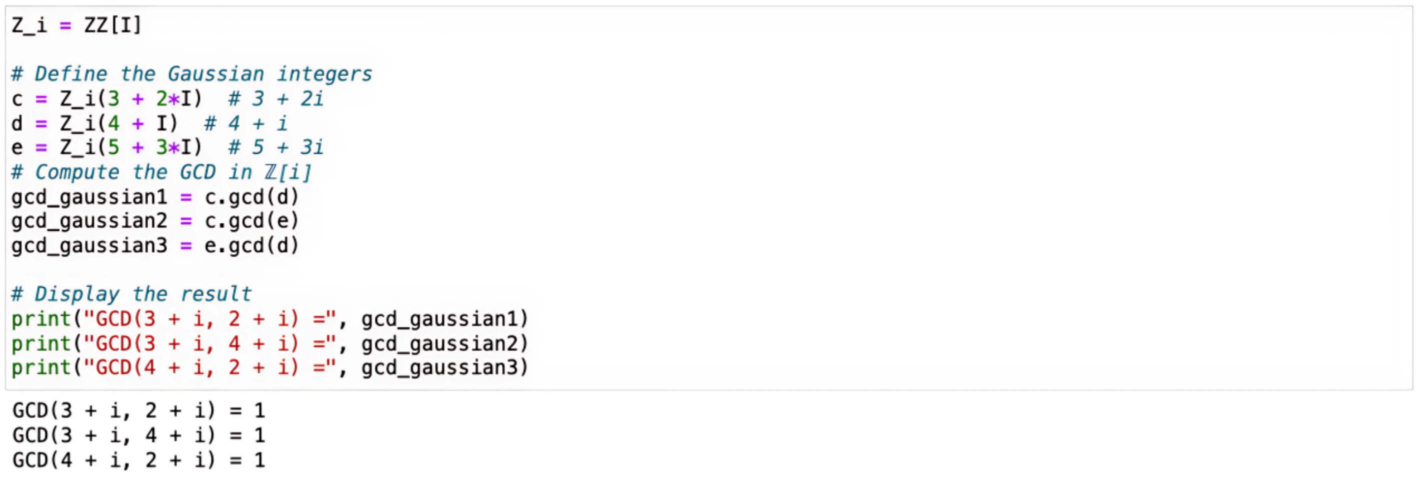
Figure 1 : CRT in F_5[X]



```
F = GF(5) # Defining the finite field GF(5)
R = PolynomialRing(F,'x') # Creating a polynomial ring R = F[x]
x = R.gen() # Defining x as the generator of the polynomial ring

# List of pairwise coprime polynomials
m = [x^2 + 2*x + 3,  x^3 + 3*x + 4]

# Checking if the polynomials are coprime
print('The GCD of the two polynomials is:',xgcd(x^2 + 2*x + 3,  x^3 + 3*x + 4)[0])

n = len(m)
M = prod(m) # Computing the product of all moduli
print('The product f(x)g(x)f(x)g(x) is:', M)


# Computing the coefficients for CRT reconstruction
coefficients = []
for i in range(n):
    M_i = M // m[i]
    M_i_inv = xgcd(m[i], M_i)[2]  # Computing the modular inverse of M_i modulo m_i
    coefficients.append(M_i * M_i_inv)
    print('The value of M_i is', M_i, 'and its modular inverse M_i_inv is', M_i_inv)

# Defining the given remainders in the system of congruences
c = [x + 2, x^2 + 4*x + 3]

# Manually reconstruct the polynomial using CRT formula
a = sum(c[i] * coefficients[i] for i in range(n)) % M
print("The reconstructed polynom is:", a)

# Verification using SageMath's built-in crt() function
a_crt = crt(c, m)

# Checking if both results match
if a != a_crt:
    print("Error: The manually computed polynomial does not match the CRT function result.")
```

```
The GCD of the two polynomials is: 1
Le produit f(x)g(x) vaut: x^5 + 2*x^4 + x^3 + 2*x + 2
The value of M_i is x^3 + 3*x + 4 and its modular inverse M_i_inv is 2*x + 4
The value of M_i is x^2 + 2*x + 3 and its modular inverse M_i_inv is 3*x^2
The reconstructed polynom is: 4*x^4 + 4*x^3 + 3*x^2 + 2*x + 4
```

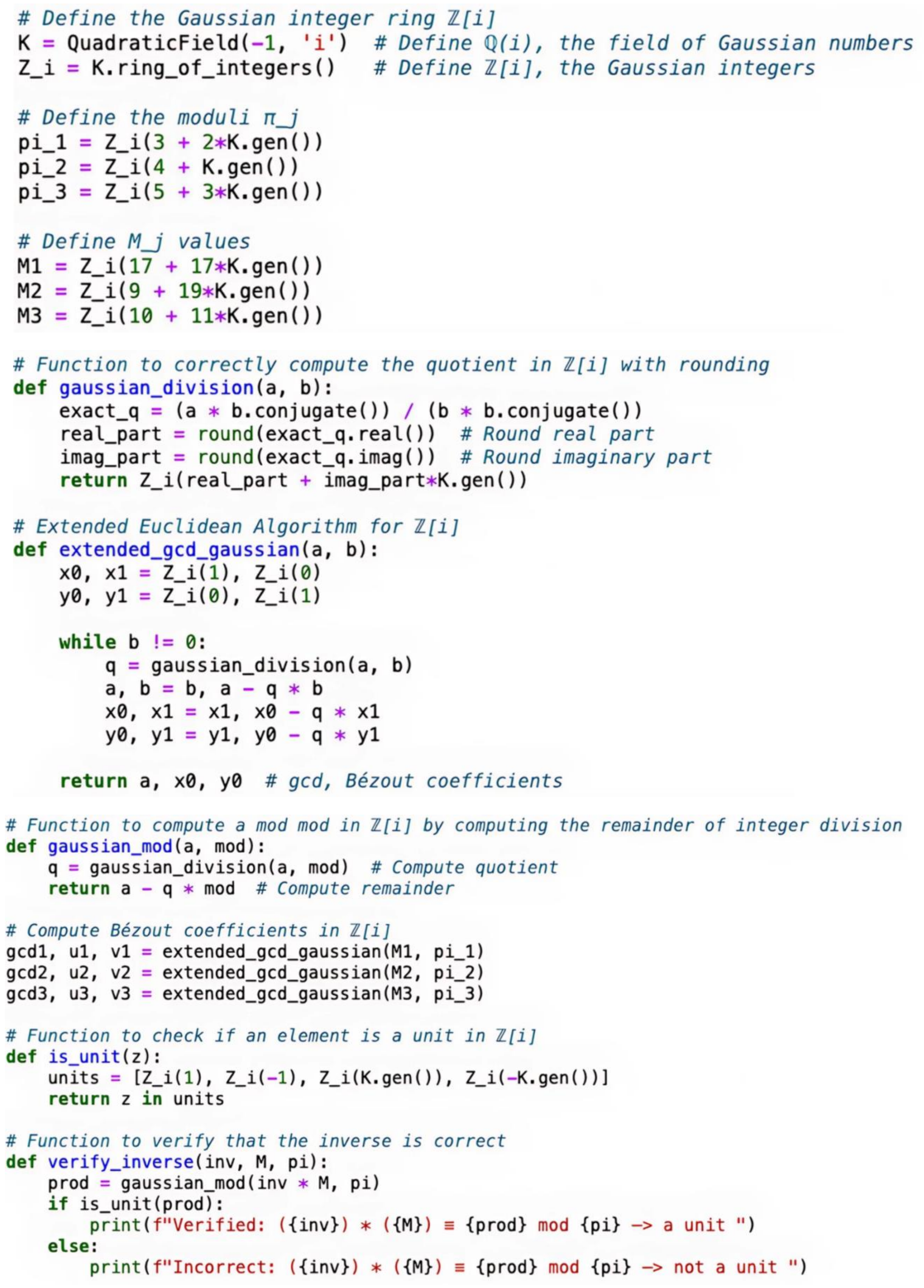
Figure 2 : Checking that Gaussian Moduli are pairwise coprime



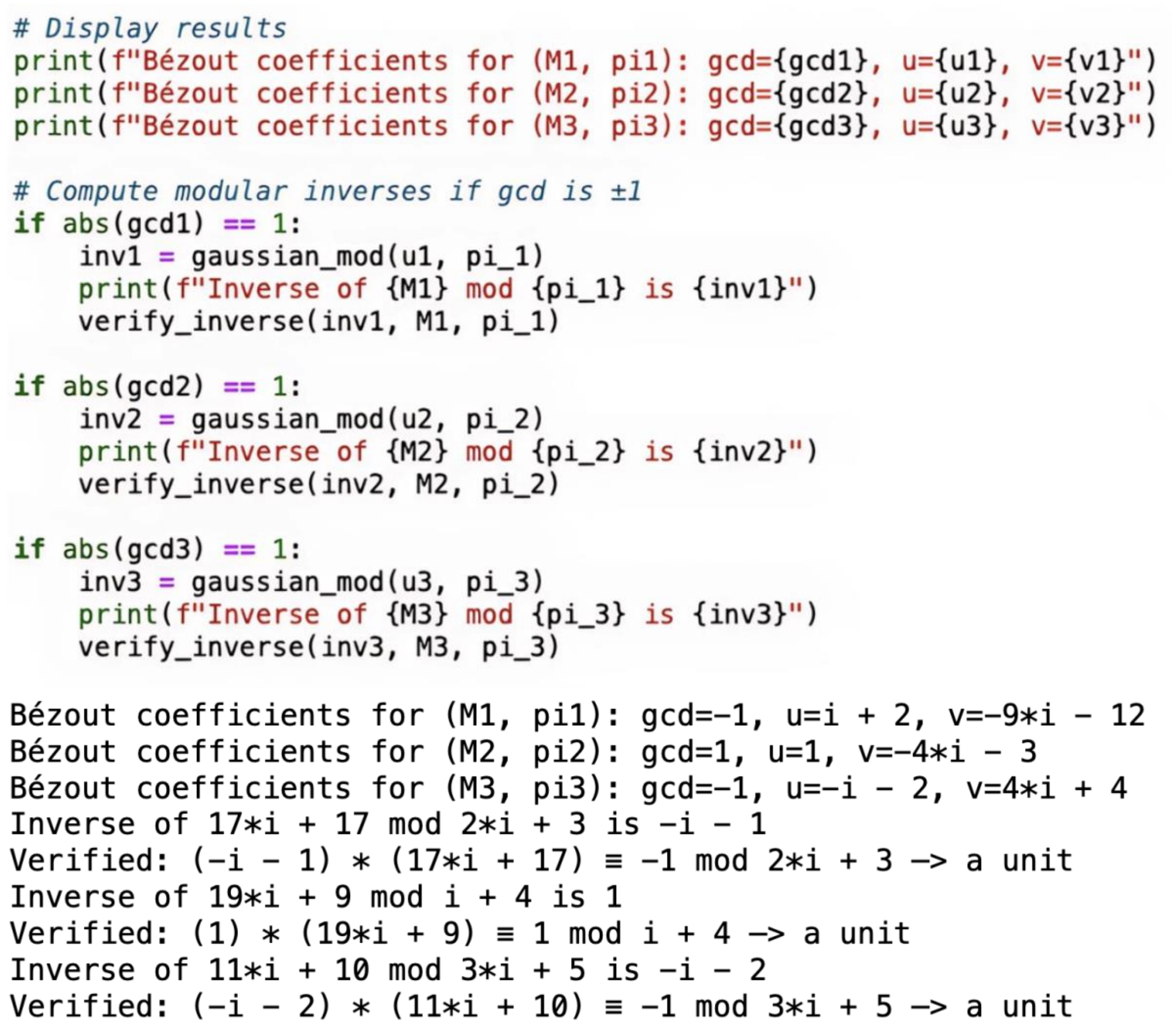
```
Z_i = ZZ[I]

# Define the Gaussian integers
c = Z_i(3 + 2*I)   # 3 + 2i
d = Z_i(4 + I)   # 4 + i
e = Z_i(5 + 3*I)   # 5 + 3i
# Compute the GCD in ℤ[i]
gcd_gaussian1 = c.gcd(d)
gcd_gaussian2 = c.gcd(e)
gcd_gaussian3 = e.gcd(d)

# Display the result
print("GCD(3 + i, 2 + i) =", gcd_gaussian1)
print("GCD(3 + i, 4 + i) =", gcd_gaussian2)
print("GCD(4 + i, 2 + i) =", gcd_gaussian3)
```

```
GCD(3 + i, 2 + i) = 1
GCD(3 + i, 4 + i) = 1
GCD(4 + i, 2 + i) = 1
```

Figure 3 : Modular Inverse in the Gaussian integers ring:



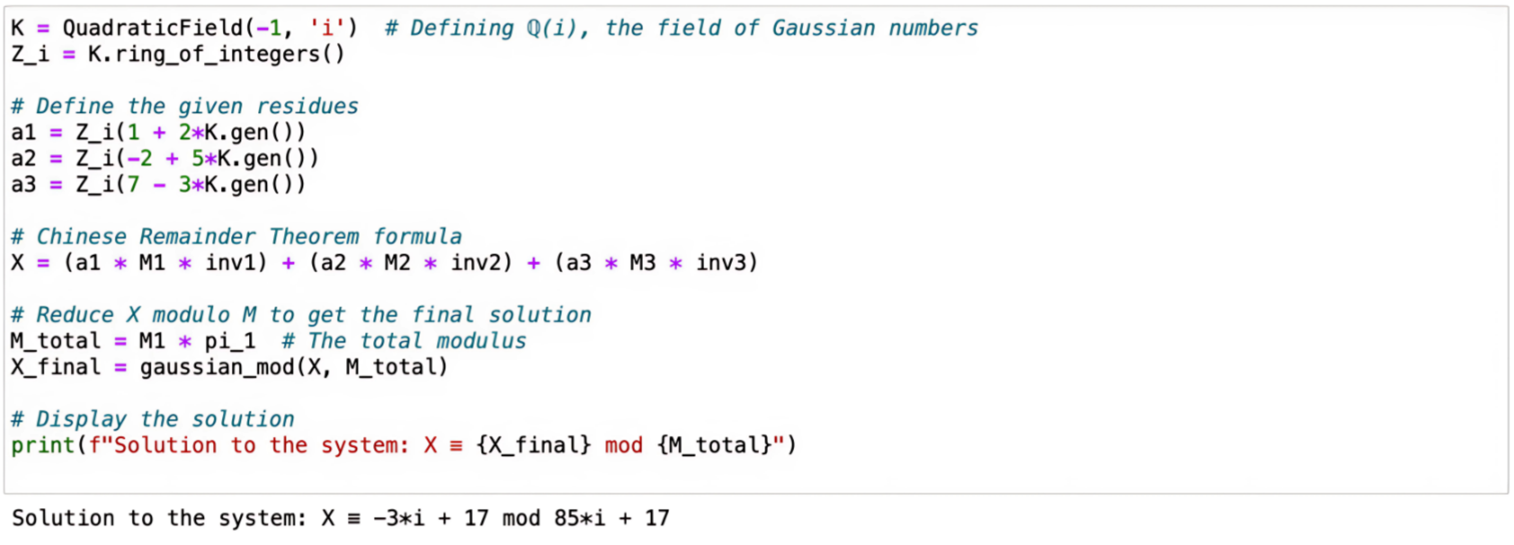
```
# Define the Gaussian integer ring ℤ[i]
K = QuadraticField(-1, 'i')  # Define ℚ(i), the field of Gaussian numbers
Z_i = K.ring_of_integers()   # Define ℤ[i], the Gaussian integers

# Define the moduli π_j
pi_1 = Z_i(3 + 2*K.gen())
pi_2 = Z_i(4 + K.gen())
pi_3 = Z_i(5 + 3*K.gen())

# Define M_j values
M1 = Z_i(17 + 17*K.gen())
M2 = Z_i(9 + 19*K.gen())
M3 = Z_i(10 + 11*K.gen())

# Function to correctly compute the quotient in ℤ[i] with rounding
def gaussian_division(a, b):
    exact_q = (a * b.conjugate()) / (b * b.conjugate())
    real_part = round(exact_q.real())  # Round real part
    imag_part = round(exact_q.imag())  # Round imaginary part
    return Z_i(real_part + imag_part*K.gen())

# Extended Euclidean Algorithm for ℤ[i]
def extended_gcd_gaussian(a, b):
    x0, x1 = Z_i(1), Z_i(0)
    y0, y1 = Z_i(0), Z_i(1)

    while b != 0:
        q = gaussian_division(a, b)
        a, b = b, a - q * b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1

    return a, x0, y0  # gcd, Bézout coefficients

# Function to compute a mod mod in ℤ[i] by computing the remainder of integer division
def gaussian_mod(a, mod):
    q = gaussian_division(a, mod)  # Compute quotient
    return a - q * mod  # Compute remainder

# Compute Bézout coefficients in ℤ[i]
gcd1, u1, v1 = extended_gcd_gaussian(M1, pi_1)
gcd2, u2, v2 = extended_gcd_gaussian(M2, pi_2)
gcd3, u3, v3 = extended_gcd_gaussian(M3, pi_3)

# Function to check if an element is a unit in ℤ[i]
def is_unit(z):
    units = [Z_i(1), Z_i(-1), Z_i(K.gen()), Z_i(-K.gen())]
    return z in units

# Function to verify that the inverse is correct
def verify_inverse(inv, M, pi):
    prod = gaussian_mod(inv * M, pi)
    if is_unit(prod):
        print(f"Verified: ({inv}) * ({M}) ≡ {prod} mod {pi} -> a unit ")
    else:
        print(f"Incorrect: ({inv}) * ({M}) ≡ {prod} mod {pi} -> not a unit ")
```



```
# Display results
print(f"Bézout coefficients for (M1, pi1): gcd={gcd1}, u={u1}, v={v1}")
print(f"Bézout coefficients for (M2, pi2): gcd={gcd2}, u={u2}, v={v2}")
print(f"Bézout coefficients for (M3, pi3): gcd={gcd3}, u={u3}, v={v3}")

# Compute modular inverses if gcd is ±1
if abs(gcd1) == 1:
    inv1 = gaussian_mod(u1, pi_1)
    print(f"Inverse of {M1} mod {pi_1} is {inv1}")
    verify_inverse(inv1, M1, pi_1)

if abs(gcd2) == 1:
    inv2 = gaussian_mod(u2, pi_2)
    print(f"Inverse of {M2} mod {pi_2} is {inv2}")
    verify_inverse(inv2, M2, pi_2)

if abs(gcd3) == 1:
    inv3 = gaussian_mod(u3, pi_3)
    print(f"Inverse of {M3} mod {pi_3} is {inv3}")
    verify_inverse(inv3, M3, pi_3)
```

```
Bézout coefficients for (M1, pi1): gcd=-1, u=i + 2, v=-9*i - 12
Bézout coefficients for (M2, pi2): gcd=1, u=1, v=-4*i - 3
Bézout coefficients for (M3, pi3): gcd=-1, u=-i - 2, v=4*i + 4
Inverse of 17*i + 17 mod 2*i + 3 is -i - 1
Verified: (-i - 1) * (17*i + 17) ≡ -1 mod 2*i + 3 -> a unit
Inverse of 19*i + 9 mod i + 4 is 1
Verified: (1) * (19*i + 9) ≡ 1 mod i + 4 -> a unit
Inverse of 11*i + 10 mod 3*i + 5 is -i - 2
Verified: (-i - 2) * (11*i + 10) ≡ -1 mod 3*i + 5 -> a unit
```

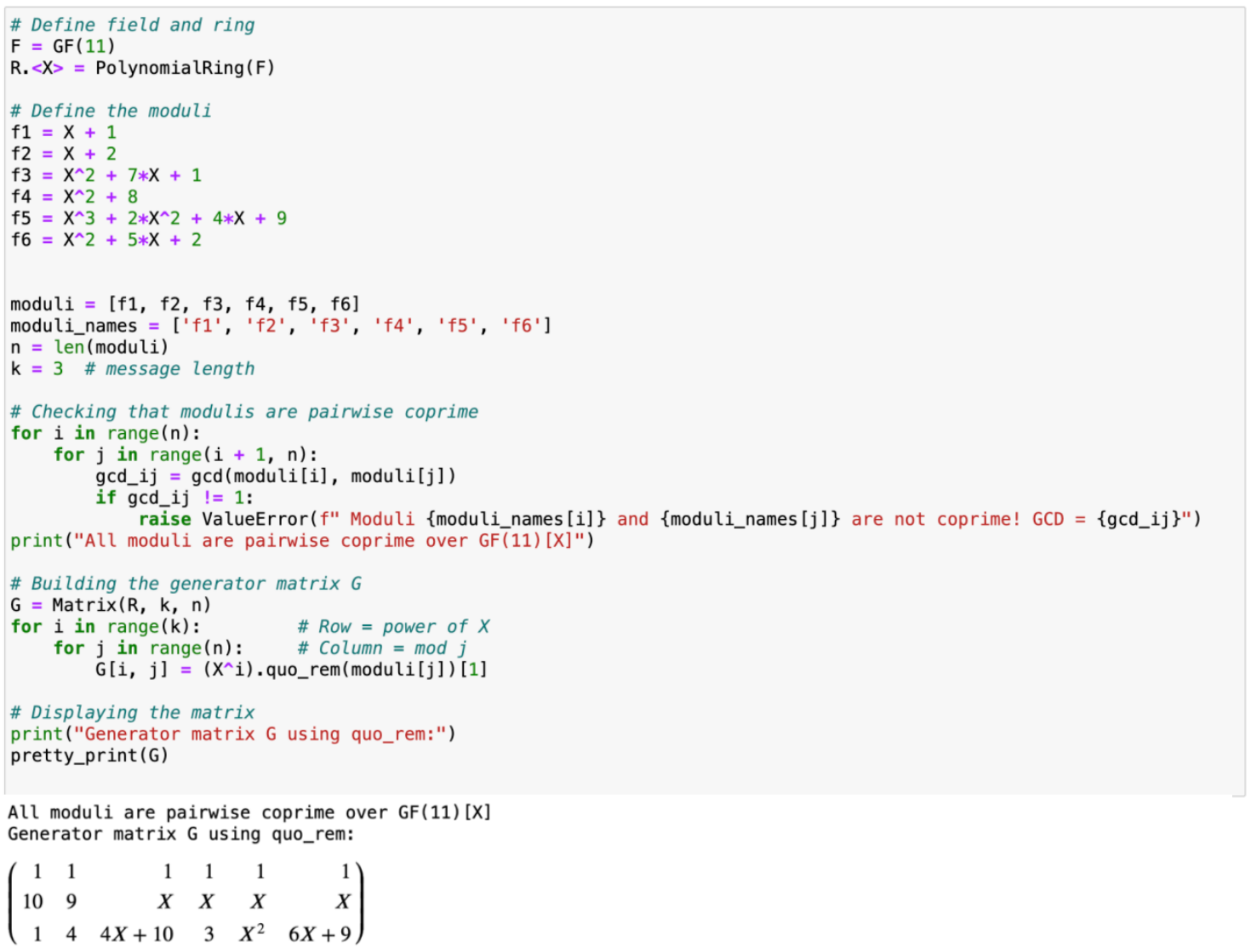
Figure 4 : Reconstructing the solution in the Gaussian integers ring:



```
K = QuadraticField(-1, 'i')  # Defining ℚ(i), the field of Gaussian numbers
Z_i = K.ring_of_integers()

# Define the given residues
a1 = Z_i(1 + 2*K.gen())
a2 = Z_i(-2 + 5*K.gen())
a3 = Z_i(7 - 3*K.gen())

# Chinese Remainder Theorem formula
X = (a1 * M1 * inv1) + (a2 * M2 * inv2) + (a3 * M3 * inv3)

# Reduce X modulo M to get the final solution
M_total = M1 * pi_1  # The total modulus
X_final = gaussian_mod(X, M_total)

# Display the solution
print(f"Solution to the system: X ≡ {X_final} mod {M_total}")
```

```
Solution to the system: X ≡ -3*i + 17 mod 85*i + 17
```

Figure 5 : Generator Matrix

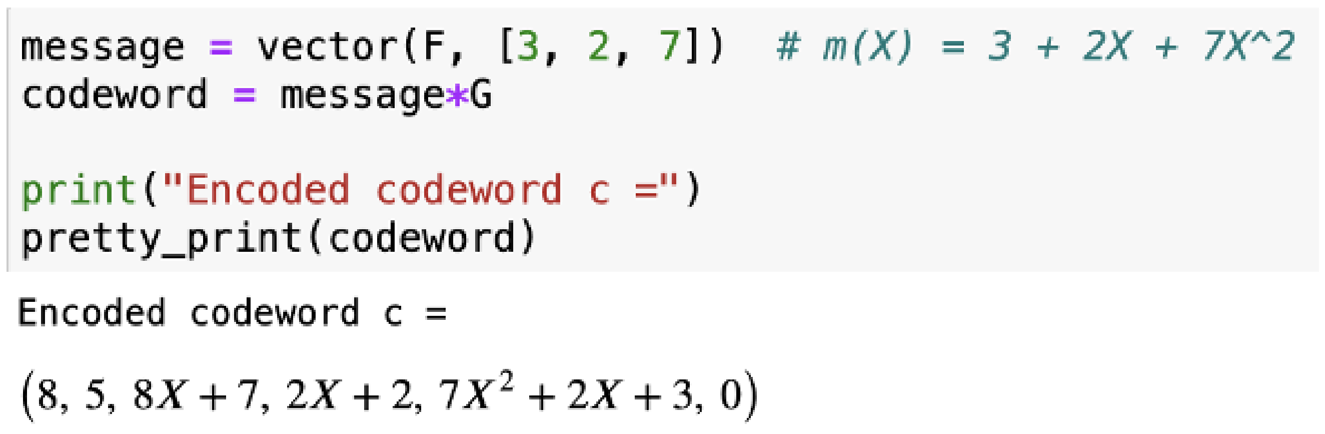


```
# Define field and ring
F = GF(11)
R.<X> = PolynomialRing(F)

# Define the moduli
f1 = X + 1
f2 = X + 2
f3 = X^2 + 7*X + 1
f4 = X^2 + 8
f5 = X^3 + 2*X^2 + 4*X + 9
f6 = X^2 + 5*X + 2


moduli = [f1, f2, f3, f4, f5, f6]
moduli_names = ['f1', 'f2', 'f3', 'f4', 'f5', 'f6']
n = len(moduli)
k = 3  # message length

# Checking that modulis are pairwise coprime
for i in range(n):
    for j in range(i + 1, n):
        gcd_ij = gcd(moduli[i], moduli[j])
        if gcd_ij != 1:
            raise ValueError(f" Moduli {moduli_names[i]} and {moduli_names[j]} are not coprime! GCD = {gcd_ij}")
print("All moduli are pairwise coprime over GF(11)[X]")

# Building the generator matrix G
G = Matrix(R, k, n)
for i in range(k):          # Row = power of X
    for j in range(n):      # Column = mod j
        G[i, j] = (X^i).quo_rem(moduli[j])[1]

# Displaying the matrix
print("Generator matrix G using quo_rem:")
pretty_print(G)
```

```
All moduli are pairwise coprime over GF(11)[X]
Generator matrix G using quo_rem:
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 10 & 9 & X & X & X & X \\ 1 & 4 & 4X+10 & 3 & X^2 & 6X+9 \end{pmatrix}$$

Figure 6 : Encoding a message

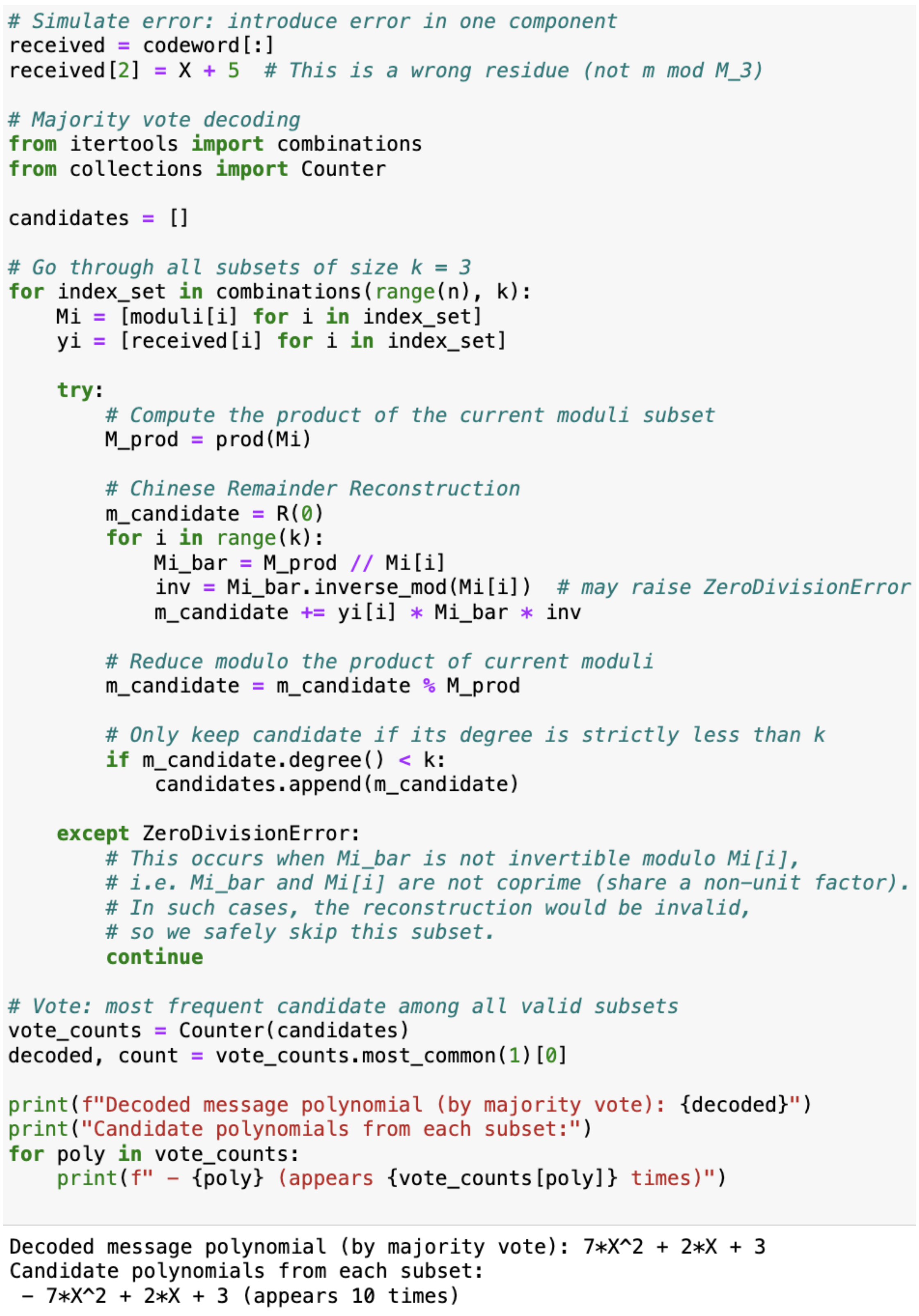


```
message = vector(F, [3, 2, 7])  # m(X) = 3 + 2X + 7X^2
codeword = message*G

print("Encoded codeword c =")
pretty_print(codeword)
```

```
Encoded codeword c =
```

$$(8,\ 5,\ 8X+7,\ 2X+2,\ 7X^2+2X+3,\ 0)$$

Figure 7 : Decoding using Majority Vote for 1 error



```
# Simulate error: introduce error in one component
received = codeword[:]
received[2] = X + 5  # This is a wrong residue (not m mod M_3)

# Majority vote decoding
from itertools import combinations
from collections import Counter

candidates = []

# Go through all subsets of size k = 3
for index_set in combinations(range(n), k):
    Mi = [moduli[i] for i in index_set]
    yi = [received[i] for i in index_set]

    try:
        # Compute the product of the current moduli subset
        M_prod = prod(Mi)

        # Chinese Remainder Reconstruction
        m_candidate = R(0)
        for i in range(k):
            Mi_bar = M_prod // Mi[i]
            inv = Mi_bar.inverse_mod(Mi[i])  # may raise ZeroDivisionError
            m_candidate += yi[i] * Mi_bar * inv

        # Reduce modulo the product of current moduli
        m_candidate = m_candidate % M_prod

        # Only keep candidate if its degree is strictly less than k
        if m_candidate.degree() < k:
            candidates.append(m_candidate)

    except ZeroDivisionError:
        # This occurs when Mi_bar is not invertible modulo Mi[i],
        # i.e. Mi_bar and Mi[i] are not coprime (share a non-unit factor).
        # In such cases, the reconstruction would be invalid,
        # so we safely skip this subset.
        continue

# Vote: most frequent candidate among all valid subsets
vote_counts = Counter(candidates)
decoded, count = vote_counts.most_common(1)[0]

print(f"Decoded message polynomial (by majority vote): {decoded}")
print("Candidate polynomials from each subset:")
for poly in vote_counts:
    print(f" - {poly} (appears {vote_counts[poly]} times)")
```

```
Decoded message polynomial (by majority vote): 7*X^2 + 2*X + 3
Candidate polynomials from each subset:
 - 7*X^2 + 2*X + 3 (appears 10 times)
```