METODOLOGÍA DE IMPLEMENTACIÓN DE MÓDULOS Y LIBRERÍAS EN HERRAMIENTAS MATEMÁTICAS DE SOFTWARE LIBRE: SOPORTE DE CAMPOS FINITOS PARA SCILAB

DANIEL ERNESTO RODRIGUEZ ORTEGA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

2009

METODOLOGÍA DE IMPLEMENTACIÓN DE MÓDULOS Y LIBRERÍAS EN HERRAMIENTAS MATEMÁTICAS DE SOFTWARE LIBRE: SOPORTE DE CAMPOS FINITOS PARA SCILAB

AUTOR:

DANIEL ERNESTO RODRIGUEZ ORTEGA

Trabajo de grado para optar por el título de Ingeniero de Sistemas

DIRECTOR:

RAFAEL FERNANDO ISAACS GIRALDO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

2009

TABLA DE CONTENIDOS

INDICE DE TABLAS

INDICE DE FIGURAS

RESUMEN

Titulo: METODOLOGÍA DE IMPLEMENTACIÓN DE MÓDULOS Y LIBRERÍAS EN HERRAMIENTAS MATEMÁTICAS DE SOFTWARE LIBRE: SOPORTE DE CAMPOS FINITOS PARA SCILAB[1].

Autor: Daniel Ernesto Rodríguez Ortega[2].

Palabras clave: campos finitos, campos de Galois, Scilab, estructuras matemáticas, grupos, anillos, campos, software libre.

Descripcion: Los campos finitos o campos de Galois son una clase de estructura matemática (como los grupos o los anillos) que tienen aplicaciones claves en la criptografía, como el algoritmo AES, usado por el gobierno de los Estados Unidos para cifrar información, y cuya matemática se basa en las operaciones entre elementos de campos finitos. También tiene aplicaciones en los algoritmos de control de errores, como el algoritmo Reed-Solomon, usado especialmente en la verificación de errores en transmisiones televisivas digitales terrestres como DVB-T y ATSC.

En el mercado actual existen extensiones para trabajar sobre campos finitos en paquetes matemáticos comerciales como Matlab o Mathematica. En contraste, no existen herramientas para trabajar sobre campos finitos en paquetes de software matemático libres como Scilab u Octave.

En este proyecto se sortea la anterior limitación al implementar una serie de funciones para trabajar con campos finitos en el paquete Scilab, que permitirá a diversos usuarios (científicos, profesores de matemáticas, ingenieros) definir campos primos y de extensión, realizar operaciones entre elementos de campos, determinar si un

[1]Trabajo de grado

[2]Facultad de Ingenierías Físico - Mecánica, Ingeniería de Sistemas e Informática, Rafael Isaacs Giraldo

polinomio es primitivo o no, determinar si un elemento del campo es primitivo o no, expresar elementos de un campo de diferentes formas (polinomial, vectorial y exponencial), realizar operaciones aritméticas sobre polinomios definidos sobre campos primos y ejecutar funciones auxiliares para implementar algoritmos de criptografía y corrección de errores en Scilab.

ABSTRACT

Title: METHODOLOGY OF IMPLEMENTATION OF MODULES AND LIBRARIES IN FREE SOFTWARE MATHEMATICAL TOOLS: FINITE FIELD SUPPORT FOR SCILAB[3].

Author: Daniel Ernesto Rodríguez Ortega[4].

keywords: finite fields, galois fields, Scilab, mathematical structures, groups, rings, fields, free software.

Description: Finite fields or Galois fields are a class of mathematical structures (like groups and rings) which have key applications in cryptography, such as the AES Algorithm, where much of its mathematic foundations are based on arithmetic operations between elements of finite fields, and is used by the USA government to encrypt classified information. Finite fields also have applications in error control algorithms, such as the Reed-Solomon algorithm, used to check if there are any errors in digital terrestrial television transmissions such as the DVB-T standard.

Currently there are many software extensions that allow working with finite fields in commercial mathematical software like Matlab or Mathematica. On the other side, there is a lack of tools for working with finite fields in open source mathematical software like Scilab or Octave.

The previous limitation is sorted out in this project, in which a set of functions are implemented for working with finite fields in the Scilab package, that will allow different class of users (scientists, math professors, engineers) to create prime fields and extension fields, perform arithmetic operations with elements of finite fields,

[3]Graduate thesis

[4]Faculty of Physics Mechanics - Engineering, Systems and Informatics Engineering, Rafael Isaacs Giraldo

to check if a polynomial is primitive or not, to check if an element of a finite field is primitive or not, list the elements of a field in different ways (polynomial, vector, exponential), perform arithmetic operations between polynomials defined over a prime field and to run helper routines for implementation of cryptography and error control algorithms in Scilab.

1. INTRODUCCION

Durante el proceso de aprendizaje en la carrera de Ingeniería de Sistemas, se han examinado diversos temas de matemática discreta que son de vital importancia en las ciencias de la computación, tales como los conjuntos, relaciones, funciones, grafos, teoría de conteo, árboles, álgebra booleana, entre muchos otros conceptos. También se da una introducción sobre algunas estructuras algebraicas como los espacios vectoriales, grupos, anillos y campos, los cuales conforman bloques básicos de información en áreas como la criptografía, bases de datos, redes de comunicaciones, almacenamiento masivo de datos y la teoría de código.

Los Campos Finitos, o Campos de Galois, ocupan un espacio importante dentro del conjunto de las estructuras algebraicas. Un campo finito, denotado como $GF(p^n)$ tiene un orden p^n finito de elementos, donde p es un número primo y $n \geq 1$, además cumplen ciertos axiomas con respecto a las operaciones definidas en el campo.

La teoría general fue desarrollada desde la mitad del siglo XIX, principalmente por el joven matemático francés Évariste Galois, y encontró su aplicación a la informática a mediados del siglo XX al aprovechar sus propiedades con operaciones a nivel de bits y con los números primos, que los hacían ideales para ser incorporados en protocolos criptográficos como el Diffie-Hellman[5], y algoritmos de corrección de errores como el Reed-Solomon.[6]

Hoy en día existen en el mercado paquetes comerciales de computación como Matlab[7] o Mathematica[8] que disponen de librerías para trabajar con Campos de Galois. En el otro espectro del mercado, herramientas libres de computación como Scilab o

[5]New Directions in Cryptography W. Diffie and M. E. Hellman, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644-654.

[6]Irving S. Reed and Xuemin Chen: Error-Control Coding for Data Networks, Boston, Kluwer Academic Publishers, 1999.

[7]Matlab. Copyright MathWorks, 1970-2009.

[8]Mathematica. Copyright Wolfram Research, 1988-2009.

GNU Octave[9] carecen de librerías para trabajar con campos (y en general de otras estructuras matemáticas avanzadas) obligando al usuario a pagar costosas licencias para acceder a esas librerías y extensiones.

Este trabajo sortea la anterior limitación, al desarrollar un "toolbox"[10] que provee un soporte básico de Campos de Galois en Scilab, colocando a disposición del usuario funciones necesarias para crear campos, hacer operaciones con elementos de campos, trabajar con polinomios sobre campos y otras funciones relacionadas que serán útiles para cualquier aplicación relacionada con esta estructura matemática.

[9]GNU Octave. Copyright John W. Eaton, 1988-2009.

[10]toolbox es el nombre que se le dá al conjunto de librerías y herramientas que extienden la funcionalidad de un paquete matemático.

2. PRESENTACION DEL PROYECTO

2.1 TITULO

METODOLOGÍA DE IMPLEMENTACIÓN DE MÓDULOS Y LIBRERÍAS EN HERRAMIENTAS MATEMÁTICAS DE SOFTWARE LIBRE: SOPORTE DE CAMPOS FINITOS PARA SCILAB.

2.2 OBJETIVO GENERAL

Diseñar e implementar módulos y librerías para incorporar definición y construcción de Campos Finitos (Campos de Galois) e implementar operaciones para trabajar con estas estructuras algebraicas en el paquete matemático de software libre Scilab.

2.3 OBJETIVOS ESPECIFICOS

- Implementar funciones primarias que permitan construir campos finitos con base a los parámetros para su definición: número p primo característico del campo, número entero $n \geq 1$ y polinomio primitivo.
- Habilitar funciones para representar los elementos de $GF(p^n)$ en diferentes formas: entera, exponencial, polinomial y en forma de tablas.
- Soportar operaciones aritméticas (adición, sustracción, multiplicación, división y potencia) entre elementos de $GF(p^n)$.
- Soportar operaciones aritméticas (adición, sustracción, multiplicación, división y potencia) entre polinomios sobre campos finitos, además de poder caracterizar polinomios como primitivos o no primitivos y poder expresar polinomios de forma simbólica o vectorial.

- Implementar funciones adicionales para asistir en tareas de control de errores y criptografía.

2.4 JUSTIFICACION

Los Campos Finitos, como otras estructuras algebraicas, componen bloques básicos de información en muchos de los algoritmos de criptografía y control de corrección de errores, infaltables en cualquier transacción electrónica susceptible a cambios durante el proceso de transmisión de los datos, y para proteger información clasificada (secreta) almacenada en medios de almacenamiento secundario.

El algoritmo AES[11], aprobado por la NSA[12] y usado por el gobierno de los Estados Unidos para cifrar información clasificada, se basa en la operación de multiplicación entre elementos de un campo finito para poder cifrar un bloque fijo de datos. El código de corrección Reed-Solomon, usado en una gran variedad de aplicaciones comerciales como las transmisiones digitales televisivas (DVB-T, ATSC)[13] y verificación de errores en discos de almacenamiento óptico (CD, DVD) emplea campos finitos del orden de 2^n para determinar un polinomio de grado $n-1$ y generar puntos para saber si los datos han llegado erróneos o no.

La elaboración de este trabajo tiene dos propósitos, ante todo, permitir trabajar sobre campos finitos en el paquete computacional Scilab, un software de computación numérica en auge entre la comunidad universitaria por ser una aplicación libre, con un gran rango de operaciones matemáticas para diversas áreas, y similar en funcionalidades a Matlab.

Este proyecto podrá ser usado tanto por fines meramente académicos, para facilitar el aprendizaje a los estudiantes de la Licenciatura de Matemáticas y de Ingeniería de esta importante estructura algebraica, o como parte de otro proyecto de mayor

[11] Joan Daemen and Vincent Rijmen: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[12] Nacional Security Agency: Agencia de Seguridad Nacional, Agencia de los Estados Unidos responsable del desarrollo de tecnologías para comunicaciones seguras en el gobierno.

[13] DVB-T y ATSC son dos tipos de formatos para transmisiones digitales televisivas terrestres.

calibre y que integre Scilab para realizar cálculos matemáticos avanzados. El usuario también podrá usar las funciones y operaciones que provee este proyecto para implementar algoritmos en el lenguaje de programación de Scilab para códigos de control de errores que involucran Campos de Galois.

De igual forma, el proyecto podría usarse para implementar algoritmos de criptografía, aunque para sistemas criptográficos avanzados que manejan información crítica, se recomienda implementar sus propias funciones para el manejo de campos finitos. En esos sistemas de cifrado es clave la optimización y la validación de cada uno de los algoritmos que involucra el sistema frente a los diferentes ataques que aprovechan vulnerabilidades en la implementación de los procedimientos.

El otro propósito que se espera cumplir con este proyecto es lograr que la Universidad Industrial de Santander participe activamente en la investigación, desarrollo y promoción de Software Libre, sin las usuales restricciones de acceso, modificación y distribución a la que es sometida la propiedad intelectual generada en la UIS.
Se espera establecer un precedente con este proyecto, al permitir liberar la herramienta bajo una licencia libre, y demostrar que la UIS sigue manteniendo derechos de autor sobre la obra en cuestión, pero bajo un nuevo paradigma que permite a otros usuarios y entidades externas examinar, modificar y distribuir copias siempre y cuando se mantengan los créditos de autoría original, en este caso, la Universidad Industrial de Santander.

3. MARCO TEORICO

3.1 INTRODUCCION AL ALGEBRA ABSTRACTA

Muchos de nosotros como estudiantes de Ingeniería, tenemos la errónea idea de que las operaciones elementales (adición, sustracción, producto y división), las variables, los sistemas de ecuaciones lineales y los polinomios representan la esencia del álgebra abstracta. En realidad el álgebra es mucho más ámplio que el álgebra elemental y se generaliza en el álgebra abstracta.

En el álgebra abstracta operaciones como la adición y la multiplicación son tratadas como operaciones generales que pueden operar sobre cualquier tipo de elementos siempre y cuando cumpla los axiomas, y sus definiciones precisas conducen a estructuras como grupos, anillos y campos.

3.2 ESTRUCTURAS ALGEBRAICAS

Una Estructura Algebraica está conformada por uno o más conjuntos cerrados sobre una o más operaciones interrelacionadas de alguna forma que satisfacen ciertos axiomas. Que un conjunto sea cerrado indica que cualesquiera dos elementos de G integrados bajo un operador general $\circ$ (que puede ser la operación adición o producto para el caso de $\mathbb{Z}$) el resultado será siempre otro elemento del grupo G, es decir:

$$\forall x, y \in G, x \circ y \in G$$

A partir de la anterior definición, se pueden construir una serie de posibles modelos con un número variado de conjuntos y operaciones sobre los conjuntos, además de cumplir algunos axiomas. Es importante aclarar que cualquier conjunto que satisface los axiomas que define una estructura es una instancia de dicha estructura,

sin importar cuantos axiomas cumple. Por ejemplo, todos los grupos son a la vez semigrupos y magmas.

3.2.1 GRUPOS

Definición 1: Un grupo, definido como $(G, \bullet)$, está compuesto por un conjunto G no vacío y una operación binaria $\bullet$ sobre G tal que se cumple las siguientes propiedades:

1. **Cerradura**: Para todo $a, b \in G$ el elemento $a \bullet b$ está únicamente definido en G.

2. **Asociatividad**: Para todo $a, b, c \in G$, se cumple:

$$a \bullet (b \bullet c) = (a \bullet b) \bullet c$$

3. **Existencia de un elemento identidad**: Existe un elemento identidad $e \in G$ tal que:

$$a \bullet e = a \ ; \ e \bullet a = a$$

   para todo $a \in G$. En el caso de la adición, el elemento identidad es 0.

4. **Existencia de un elemento inverso**: Para todo $a \in G$ existe un elemento inverso $a^{-1} \in G$ tal que:

$$a \bullet a^{-1} = e \ ; \ a^{-1} \bullet a = e$$

   En el caso de la adición es $-a$

Se puede ver que las anteriores propiedades son familiares a aquellas aplicadas en los números reales o enteros respecto a las operaciones de adición o multiplicación. Pero en la definición del grupo, la formulación de los axiomas está separada de la naturaleza de la operación, es decir, el símbolo $\bullet$ es un indicador de cualquier operación general, con tal de que los axiomas se cumplan.

Uno ejemplo es el grupo $(\mathbb{Z}, +)$, se puede verificar que los axiomas se cumplen para la operación $+$. Por otro lado, $(\mathbb{Z}, *)$ no puede conformar un grupo, se observa que no se cumple la existencia de un elemento inverso.

Existen varios grupos especiales que valen la pena mencionar:

Grupo Abeliano: Es aquel grupo en el que se cumple la propiedad conmutativa, es decir:

$$\forall a, b \in G, a \circ b = b \circ a$$

Grupos como $(\mathbb{Z}, +)$, $(\mathbb{Q}, +)$ y $(\mathbb{R}, *)$ son grupos abelianos.

Grupo Finito: Es un grupo con un número finito de elementos.

Monoide: Es una estructura matemática similar a un grupo, excepto que sólo cumple tres axiomas de grupo: cerradura, asociatividad y existencia de un elemento identidad.

Grupo Cíclico: Es un grupo conmutativo, finito o infinito, que puede ser generado por multiplicación reiterada de un sólo elemento. Es decir, existe un elemento $a \in G$ tal que para cualquier $b \in G$ hay un entero positivo j donde $b = a^j$. El elemento a es llamado el generador del grupo cíclico.

3.2.2 ANILLOS

Definición 2: Sea R un conjunto equipado con dos operaciones binarias, $+ : R \times R \longrightarrow R$ y $\bullet : R \times R \longrightarrow R$, entonces R es un anillo, denotado como $(R, +, \bullet)$ tal que:

1. R es un grupo abeliano con respecto a $+$.
2. $\bullet$ es asociativa.

3. Las leyes distributivas se cumplen, es decir, para todo $a, b, c \in G$ se tiene:

$$a \bullet (b + c) = a \bullet b + a \bullet c \; ; \; (b + c) \bullet a = b \bullet a + c \bullet a$$

Un ejemplo de anillos es el conjunto $\mathbb{Z}$. Se puede ver que $\mathbb{Z}$ también forma un grupo respecto a la operación + y un monoide respecto a la operación $\bullet$.

Anillos especiales a mencionar:

Anillo conmutativo: Un anillo es conmutativo si $\bullet$ es conmutativo.

Anillo de división: Anillo donde los elementos diferentes de 0 forman un grupo sobre $\bullet$.

Anillo de polinomios: Es un anillo formado por los polinomios definidos sobre R, y se denota como $R[x]$. Es decir si $R = \{1, 2, 3, 4, 5\}$, entonces un polinomio $f(x) \in R[x]$ si los coeficientes de $f(x)$ pertenecen a R. De modo que $R[x] = \{1,\ x,\ x + 2,\ x^3 + 4,\ ...\}$.

El concepto de divisibilidad, especializado al anillo $F[x]$, conduce a lo siguiente: El polinomio $g \in F[x]$ divide el polinomio $f \in F[x]$ si existe un polinomio $h \in F[x]$ tal que $f = gh$. Esto se generaliza en el algoritmo de división:

Teorema 1: Sea $g \neq 0$ un polinomio en $F[x]$. Entonces para cualquier $f \in F[x]$ existe los polinomios $q, r \in F[x]$, q cociente y r residuo, tal que:

$f = qg + r$, donde grado$(r) <$ grado(g), o grado$(r) = 0$.

Ejemplo: Considere $f(x) = x^3 - 12x^2 - 42 \in \mathbb{R}[x]$ y $g(x) = x - 3 \in \mathbb{R}[x]$, hallando los polinomios r y q mediante la división polinomial larga:

$$
\begin{array}{r}
x^2 - 9x - 27 \\
\hline
x - 3 \,|\, x^3 - 12x^2 + 0x - 42 \\
\underline{x^3 - 3x^2} \\
-9x^2 + 0x \\
\underline{-9x^2 + 27x} \\
-27x - 42 \\
\underline{-27x + 81} \\
-123
\end{array}
$$

Figura 1. División polinomial larga.

De modo que:

$x^3 - 12x^2 - 42 = (x^2 - 9x - 27)(x - 3) - 123$

3.2.3 CAMPOS

Definición 3: Un campo es un conjunto F en el cual están definidas las operaciones $+$ y $\bullet$, y contiene los elementos 0 y e, con la condición que $e \neq 0$. De tal forma, F es un grupo abeliano con respecto a la adición, donde 0 es el elemento identidad, y los elementos de F que son diferentes de 0 forman un grupo abeliano respecto a la multiplicación, teniendo a e como elemento identidad, que usualmente es 1.

Ejemplos de campos tenemos a los conjuntos $\mathbb{R}$, $\mathbb{Q}$ y los campos finitos, que se explicarán detalladamente en la sección 3.3.

3.3 CAMPOS FINITOS

3.3.1 GENERALIDADES

Definición 4: Un Campo Finito, también llamado Campo de Galois en honor a Évariste Galois[14] es un campo F denotado como $GF(p^n)$ o también como F_{p^n}, y tiene un número p^n finito de elementos donde se cumple que:

[14]Évariste Galois (1811 - 1832), matemático francés que introdujo el concepto de campo finito.

1. p es un número primo, llamado el característico de $GF(p^n)$.
2. n es el grado de $GF(p^n)$, donde $n \geq 1$.
3. Por cada primo p y entero positivo n existe un único campo finito con p^n elementos.

Definición 5: El característico de un campo F es definido como el menor número de veces que se debe sumar el elemento identidad multiplicativo (1) de un campo para obtener el elemento identidad aditivo (0).

Corolario: Un campo finito tiene un primo característico.

Se puede probar que el p característico de un campo debe ser primo para generar $GF(p^n)$, por razones de simplicidad, omitimos la demostración en este libro.

Un campo $GF(p^n)$ puede contener otros campos (subcampos) definidos sobre las mismas operaciones de $GF(p^n)$. Por ejemplo el campo $GF(2^{10})$ contiene otros subcampos, que son $GF(2)$, $GF(2^2)$ y $GF(2^5)$.

Cuando $n = 1$ el campo $GF(p)$ es llamado Campo Primo, cuando $n > 1$ el campo es llamado Campo de Extensión.

3.3.2 CAMPOS PRIMOS

Como se mencionó en la sección anterior, un campo $GF(p^n)$ es campo primo si $n = 1$, y está definido como:

$$GF(p) = \{0,\ 1,\ 2,\ ...,\ p-1\}$$

Definición 6: Un campo que no contiene subcampos propios es llamado un campo primo.

Por la definición anterior, cualquier campo de orden p, p primo, es un campo primo. Otro ejemplo de un campo primo es el campo $\mathbb{Q}$ de los números racionales.

Se puede ver que $GF(p)$ es el conjunto de los residuos módulo p (denotado en algunos libros como $\mathbb{Z}_p$), y de hecho toda la aritmética en $GF(p)$ se trabaja en base a módulo p. En las siguientes figuras se muestran las tablas de adición y multiplicación (también llamadas tablas de Cayley[15]) para el campo $GF(5)$:

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Tabla 1. Tabla de adición para el campo $GF(5)$.

•	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Tabla 2. Tabla de producto para el campo $GF(5)$.

Definición 7: Para los enteros arbitrarios a, b y un entero positivo n, decimos que a es congruente con b módulo n, y se escribe como $a \equiv b \,(mod\, n)$ si la diferencia $a - b$ es múltiplo de n.

[15]Arthur Cayley (1821 - 1895): Matemático inglés, fue el primer matemático que empleó tablas para ilustrar la estructura finita de un grupo.

Verificando algunas operaciones de la tabla 1 y la tabla 2 mediante las siguientes congruencias:

$4 + 0 \equiv 4 \ (mod\, 5)$
$4 + 1 \equiv 0 \ (mod\, 5)$
$3 + 3 \equiv 1 \ (mod\, 5)$
$4 \bullet 4 \equiv \ 1 \ (mod\, 5)$

Como se puede ver, el manejo de las operaciones adición y producto es bastante sencilla para campos primos, basta con realizar la operación entera normal y después reducir el resultado con la operación módulo p, el anterior proceso es llamado reducción módulo p. Para la división y sustracción en los campos primos la forma de operar cambia un poco, como se verá en la sección 3.3.6.

3.3.3 CAMPOS DE EXTENSION

Definición 8: Sea F un campo y K un subcampo de F. En este contexto, F es llamado un campo de extensión de K si $F \neq K$.

Cuando $n > 1$, los elementos de $GF(q = p^n)$ no se pueden expresar de la forma $0, 1, ...p^n - 1$ directamente, ya que las operaciones aritméticas no darían los resultados esperados, y los axiomas no se cumplirían, entonces hay que buscar otra forma de expresar los elementos de los campos de extensión. Para lograr esto vamos a enunciar algunos teoremas y definiciones.

Definición 9: Un polinomio $f(x)$ en $F[x]$ es irreducible sobre F si no es constante y no puede ser representado como el producto de dos o más polinomios no constantes de $F[x]$.

Que un polinomio sea reducible o irreducible depende de cómo está definido F. Por ejemplo el polinomio $f(x) = x^2 + 1$ es irreducible sobre $\mathbb{R}$, pero se puede factorizar sobre $\mathbb{C}$:

$$f(x) = (x + i)(x - i)$$

Que $f(x)$ sea irreducible también implica que el polinomio no tiene raíz en F, es decir que para todo $a \in F$, $f(a) \neq 0$.

Teorema 2: Un polinomio $f(x)$ es mónico si el coeficiente del término de mayor orden es 1. Por ejemplo:

$$f(x) = x^2 + x + 1$$

$$f(x) = x^5 - 1$$

Son polinomios mónicos, mientras que:

$$f(x) = 2x^2 - 1$$

$$f(x) = 3x^{10} - 2$$

No son polinomios mónicos.

Teorema 3: Sea $f \in F[x]$ un polinomio mónico irreducible sobre el campo F. Entonces existe una simple extensión algebraica de F con una raíz de f como elemento.

El anterior teorema implica que si $GF(q)$ es una extensión algebraica de $GF(p)$, y que los elementos de dicha extensión están formados mediante combinaciones algebraicas que involucran la raíz α de $f(x)$, podemos decir que los elementos de $GF(q)$ están expresados como polinomios en α sobre $GF(p)$.

Por ejemplo, dado un campo $GF(2) = \{0, 1\}$ y un polinomio $f(x) = x^2 + x + 1$ que no tiene raíz en $GF(2)$. Sea α una "raíz" de $f(x)$ tal $\alpha^2 + \alpha + 1 = 0$, donde α cumple el papel de una raíz "imaginaria"(si hacemos un paralelo con la forma como se genera el conjunto $\mathbb{C}$). Para hallar los elementos de $GF(q)$ podemos hallar las clases de residuo:

$$L = \frac{F_2[\alpha]}{f(\alpha)}$$

Para hallar las clases de residuo tomamos algunos polinomios arbitrarios de $F_2[\alpha]$, aplicamos el algoritmo de división a cada uno con $f(\alpha)$ como cociente y tomamos los residuos. El anterior proceso es llamado reducción módulo $f(\alpha)$.

Las clases de residuo L son precisamente los elementos de $GF(q = 2^2)$, que son:

$$L = \{0,\ 1,\ \alpha,\ \alpha + 1\}$$

Del conjunto L y del campo $GF(q)$ podemos ver que:

1. Tiene un orden $4 = 2^2$.
2. El grado de los polinomios es menor que $n = 2$.
3. Los polinomios están formados por combinaciones de la raíz α, como se mencionó anteriormente.
4. Los coeficientes de los polinomios pertenecen al campo primo $GF(2) = \{0,\ 1\}$.
5. El grado de $f(x)$ es igual a n y sus coeficientes también pertenecen al campo $GF(2)$.

Se puede generalizar que dado un campo $GF(q = p^n)$, los elementos pueden ser representados por el conjunto:

$$\{ax^{n-1} + bx^{n-2} +,\ \ldots,\ zx^0 |\ a,\ b, \ldots,\ z \in GF(p)\}$$

La demostración de la anterior expresión está fuera del alcance de este libro, por su complejidad matemática.

La existencia de un polinomio irreducible dado p y $n \geq 1$ se garantiza en el siguiente teorema:

Definición 10: Para cualquier potencia prima q y cualquier entero $n \geq 1$ existe un polinomio irreducible de grado n sobre $GF(q)$.

Es importante que $f(x)$ sea irreducible para poder generar el campo, opcionalmente $f(x)$ puede ser primitivo para generar los elementos del campo mediante simples potencias de x.

Definición 11: Un polinomio $f(x)$ con coeficientes en $GF(p)$ es un polinomio primitivo si tiene una raiz x en $GF(p^n)$ tal que $\{0,\ 1,\ x,\ x^2,\ x^3,\ ...,\ x^{p^n-2}\}$ es el campo completo $GF(p^n)$. En la sección 3.3.4.3 se verá la forma de probar si un polinomio es primitivo o no.

3.3.4 CONSTRUCCION DE CAMPOS FINITOS

Existen varias formas de crear un campo finito dependiendo si el campo a crear es primo o de extensión. Para un campo primo $GF(p)$ la construcción es trivial, basta con crear un conjunto con los elementos de 0 a $p-1$, comprobando previamente que p sea primo. Otra forma de generar un campo primo es buscar un elemento primitivo a de $GF(p)$ y generar potencias del elemento. Mas información en la sección 3.3.4.1.

Para un campo de extensión, la forma más usual es usar el conjunto general que se mencionó en la sección 3.3.3, comprobando previamente que p sea primo, $n \geq 1$ y que $f(x)$ tenga un grado igual a n, sus coeficientes estén en el campo primo $GF(p)$ y que sea un polinomio irreducible o primitivo. El resultado es una lista de polinomios con grado menor que n.

Ejemplo: Para $p = 3$, $n = 2$ y $f(x) = x^2 + x + 2$ se comprueba si cumple los requisitos iniciales:

1. p es primo? Sí, 3 es primo.
2. $n \geq 1$? Sí, $3 \geq 1$.
3. El grado de $f(x)$ es igual a n? Sí, el grado es 2.
4. Los coeficientes de $f(x)$ pertenecen al campo $GF(p)$? Sí, $GF(3) = \{0,\ 1,\ 2\}$
5. $f(x)$ es irreducible?

Comprobando si es irreducible al probar si $f(x)$ tiene raíz en $GF(3)$:

$f(0) = 0^2 + 0 + 2 = 2$, 0 no es raíz de $f(x)$
$f(1) = 1^2 + 1 + 2 = 1$, 1 no es raíz de $f(x)$

$f(2) = 2^2 + 2 + 2 = 2$, 2 no es raíz de $f(x)$

De modo que $f(x)$ es irreducible, aplicando el conjunto $\{ax + b \mid a, b \in GF(3)\}$ se obtiene el listado de los elementos: $\{0,\ 1,\ 2,\ x,\ x+1,\ x+2,\ 2x,\ 2x+1,\ 2x+2\}$.

3.3.4.1 Elementos primitivos

Definición 12: Denotando como $GF(q)^*$ el grupo multiplicativo de los elementos diferentes de 0 de $GF(q)$, por cada campo finito $GF(q)$ el grupo $GF(q)^*$ es cíclico.

El elemento generador del grupo cíclico $GF(q)^*$ es llamado el elemento primitivo de $GF(q)$. En los campos primos el elemento primitivo recibe el nombre de raíz primitiva y está definida de la siguiente forma:

Definición 13: Un entero a es una raíz primitiva de $GF(p)$ si $a^k \not\equiv 1\ (mod\ p)$ para $0 < k < p-1$ y $a^k \equiv 1\ (mod\ p)$ para $k = p-1$

Ejemplo: Para el campo $GF(5) = \{0,\ 1,\ 2,\ 3,\ 4\}$ el elemento 3 es elemento primitivo ya que:

$3^1 \not\equiv 1\ (mod\ 5)$
$3^2 \not\equiv 1\ (mod\ 5)$
$3^3 \not\equiv 1\ (mod\ 5)$
$3^4 \equiv 1\ (mod\ 5)$

Para los campos de extensión, hallar los elementos primitivos resulta mas sencillo, sabiendo que:

1. El grupo cíclico de $GF(p^n)$ con $p^n - 1$ elementos está definido como:

$$GF(p^n)^* = \{1,\ a^1,\ a^2,\ ..,\ a^{p^n-2}\}$$

donde a es un elemento primitivo del campo. De modo que podemos generar los elementos de un campo con sólo saber un elemento primitivo conocido y aplicar potencias sucesivas sobre el elemento (asumiendo que $f(x)$ es polinomio

primitivo).

2. El menor elemento primitivo de un campo $GF(p^n)$ es precisamente la raíz del polinomio primitivo: x.

3. Un elemento a^t del campo es un elemento primitivo si $mcd(t, p^n - 1) = 1$.

De acuerdo con lo anterior, para hallar la lista de los elementos primitivos basta con buscar los valores de t entre 1 y $p^n - 2$ tal que el $mcd(t, p^n - 1) = 1$ y aplicar las potencias a^t.

Ejemplo: Para el campo $GF(3^2)$ sabemos que x es elemento primitivo, hallando los elementos primitivos restantes:

$mcd(1, 8) = 1 \Rightarrow SI$
$mcd(2, 8) = 2 \Rightarrow NO$
$mcd(3, 8) = 1 \Rightarrow SI$
$mcd(4, 8) = 4 \Rightarrow NO$
$mcd(5, 8) = 1 \Rightarrow SI$
$mcd(6, 8) = 2 \Rightarrow NO$
$mcd(7, 8) = 1 \Rightarrow SI$

Luego los elementos primitivos son: $\{x,\ x^3,\ x^5,\ x^7\} = \{x,\ 2x + 2,\ 2x,\ x + 1\}$

Se puede determinar el número de elementos primitivos para un p y un n dado, usando la función Phi de Euler[16]:

$$\phi(n) = n \bullet \prod_{p|n} \bullet \left(1 - \frac{1}{p}\right)$$

$$\phi(5 - 1) = \phi(4) = 4 \bullet \prod_{p|4} \bullet \left(1 - \frac{1}{p}\right) = 2$$

Hay que notar que la productoria se efectúa sobre los primos divisores de n.

[16]Leonhard Paul Euler (1707 - 1783) Matemático suizo quien introdujo gran parte de la terminología matemática moderna, además de sus múltiples aportes en diferentes areas.

Ejemplo: Para $GF(3^2)$, $\phi(3^2-1) = 4$

3.3.4.2 Polinomios mínimos

El concepto del polinomio mínimo está definido en la siguiente definición:

Definición 14: Un polinomio mínimo $f(x)$ de un elemento de campo b en $GF(p^n)$ es el polinomio de menor grado posible el cual tiene a b como raíz. $f(x)$ puede tener un grado mayor o igual a n.

Por ejemplo, el polinomio mínimo del elemento $x \in GF(2^2)$ es el mismo polinomio irreducible $f(x) = x^2 + x + 1$. Para el elemento $x+1 \in GF(2^2)$ El polinomio mínimo es también $f(x) = x^2 + x + 1$ ya que:

$$(x+1)^2 + (x+1) + 1 = x^2 + 1 + x + 1 + 1 = x^2 + x + 3 \ (mod\ 2) = x^2 + x + 1 = 0$$

3.3.4.3 Polinomios primitivos

Como se ha mencionado en varias secciones anteriores, una de las condiciones para generar un campo de extensión es que $f(x)$ sea irreducible, mejor aún, que sea primitivo para facilitar la creación de los elementos del campo.

Teorema 4: $f(x)$ es un polinomio primitivo si el elemento x genera el grupo cíclico de los elementos diferentes de cero del campo $GF(p^n)$. $f(x)$ debe satisfacer las ecuaciones:

$$x^m \not\equiv 1 \ (mod\ (f(x),\ p));\ 1 \leq m \leq p^n - 2$$

$$x^{p^n-1} \equiv 1 \ (mod\ (f(x),\ p))$$

Al aplicar la reducción $mod(f(x),\ p)$ para comprobar cada congruencia, se hace primero una reducción módulo $f(x)$ y al resultado intermedio (el residuo) se aplica una reducción módulo p.

Teorema 5: Un polinomio mínimo de un elemento primitivo de un campo $GF(p^n)$ es un polinomio primitivo.

Teorema 6: Un polinomio primitivo $f(x)$ en $GF(p^n)$ es también irreducible en $GF(p)$, lo inverso no necesariamente se cumple para todos los $f(x)$.

Ejemplo: Comprobando que $f(x) = x^2 + x + 2$ es polinomio primitivo para $GF(3^2)$ mediante el teorema 4:

$$x^1 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^2 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^3 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^4 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^5 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^6 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^7 \not\equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$$x^8 \equiv 1 \ (mod \ (x^2 + x + 2, \ 3))$$

$f(x) = x^2 + x + 2$ satiface las congruencias, por lo tanto es primitivo y puede generar todos los elementos diferentes de 0 del campo $GF(3^2)$ mediante potencias de x de 0 a 7 y aplicando reducción módulo $x^2 + x + 2$:

$$x^0 = 1 \ (mod \ x^2 + x + 2) = 1$$

$$x^1 = x \ (mod \ x^2 + x + 2) = x$$

$$x^2 = x^2 \ (mod \ x^2 + x + 2) = 2x + 1$$

$$x^3 = x^3 \ (mod \ x^2 + x + 2) = 2x + 2$$

$$x^4 = x^4 \ (mod \ x^2 + x + 2) = 2$$

$$x^5 = x^5 \ (mod\, x^2 + x + 2) = 2x$$

$$x^6 = x^6 \ (mod\, x^2 + x + 2) = x + 2$$

$$x^7 = x^7 \ (mod\, x^2 + x + 2) = x + 1$$

Se generó la totalidad de los elementos diferentes de 0 de $GF(3^2)$. Tomando otro polinomio con grado 2 y con coeficientes en $GF(3)$, $f(x) = x^2 + 1$, se comprueba que es irreducible:

$$f(0) = 0^2 + 1 = 1 \ (mod\, 3) = 1$$

$$f(1) = 1^2 + 1 = 2 \ (mod\, 3) = 2$$

$$f(2) = 2^2 + 1 = 5 \ (mod\, 3) = 2$$

Sin embargo $f(x)$ no es primitivo, lo podemos comprobar aplicando las congruencias del teorema 4:

$$x^1 \not\equiv 1 \ (mod\ (x^2 + 1,\ 3))$$

$$x^2 \not\equiv 1 \ (mod\ (x^2 + x + 2,\ 3))$$

$$x^3 \not\equiv 1 \ (mod\ (x^2 + x + 2,\ 3))$$

$$x^4 \equiv 1 \ (mod\ (x^2 + x + 2,\ 3))$$

Aplicando potencias de x para tratar de hallar los elementos diferentes de 0 de $GF(3^2)$, obtenemos:

$$x^0 = 1 \ (mod\, x^2 + 1) = 1$$

$$x^1 = 1 \ (mod\, x^2 + 1) = x$$

$$x^2 = 1 \ (mod\, x^2 + 1) = 2$$

$$x^3 = 1 \ (mod\, x^2 + 1) = 2x$$

$$x^4 = 1 \ (mod\, x^2 + 1) = 1$$

$$x^5 = 1 \ (mod\, x^2 + 1) = x$$

$$x^6 = 1\ (mod\ x^2 + 1) = 2$$

$$x^7 = 1\ (mod\ x^2 + 1) = 2x$$

No es el listado completo de los elementos de $GF(3^2)$, de modo que no se pueden generar los elementos de un campo mediante potencias sucesivas de x si $f(x)$ no es primitivo, habría que buscar otro método para hallar los elementos usando este polinomio.

De ahora en adelante en el libro se hará solamente referencia al carácter primitivo de un polinomio como requisito para generar un campo, debido a la facilidad de hallar los elementos. Del mismo modo en la herramienta se trabajará solamente con polinomios primitivos para la función de creación de campos de extensión.

3.3.5 REPRESENTACION DE CAMPOS FINITOS

Existen varias formas de representar los elementos de $GF(p^n)$, dependiendo si es un campo primo o un campo de extensión. Para un campo primo la representación entera es suficiente por la facilidad al estudiar las operaciones y los axiomas de campo. En algunos libros suelen usar símbolos en vez de enteros para representar los elementos del campo primo, por ejemplo el campo $GF(3)$ puede representarse de la siguiente forma:

$+$	0	1	A
0	0	1	A
1	1	A	0
A	A	0	1

Tabla 3. Tabla de adición para el campo $GF(3)$.

Para los campos de extensión existen tres formas usuales para representar los elementos: formato polinomial, formato vectorial (o formato entero) y formato exponencial, a continuación se va a describir cada uno de ellos.

3.3.5.1 Formato polinomial

En el formato polinomial, como su nombre lo indica, los elementos de $GF(p^n)$ se representan como polinomios de grado menor que n y con coeficientes que pertenecen a $GF(p)$, tal como se ha estado haciendo desde el comienzo de este libro, otro ejemplo:

$$GF(2^3) = \{0,\ 1,\ x,\ x+1,\ x^2,\ x^2+1,\ x^2+x,\ x^2+x+1\}$$

$$GF(5^2) = \{0,\ 1,\ 2,\ 3,\ 4,\ x+1,\ x+2,\ ...\}$$

3.3.5.2 Formato vectorial

El formato vectorial es similar a trabajar con el formato polinomial, solo que en vez de representar los elementos como polinomios, se representan como vectores o tuplas, que son los coeficientes de los polinomios correspondientes. En la siguiente tabla se ilustra la correspondencia entre ambos formatos para el campo $GF(2^3)$:

Polinomial	Vectorial
0	000
1	001
x	010
$x+1$	011
x^2	100
x^2+1	101
x^2+x	110
x^2+x+1	111

Tabla 4. Elementos de $GF(2^3)$ en formato polinomial y vectorial.

Notar que la posición de los enteros en el arreglo tiene significado, en este caso el entero de la izquierda representa el término de mayor grado, 2, mientras que el entero de la derecha representa el término de menor grado, 0.

3.3.5.3 Formato exponencial

Dado un campo $GF(q = p^n)$ y un polinomio primitivo $f(x)$ con raíz "x" en $GF(q)$, se puede listar los elementos del campo mediante potencias de x de -1 a $p^n - 2$:

$$GF(q) = \{x^{-1},\ x^0,\ x^1,\ x^2,\ ...,\ x^{p^n-2}\}$$

En algunos libros de campos finitos y en paquetes matemáticos como Matlab o Mathematica usan el exponente -1 para representar el exponente $-\infty$, ya que $x^{-\infty}$ es la representación del elemento 0, entonces se podría afirmar que $x^{-1} = 0$.
Si a cada potencia x^t se aplica el algoritmo de división polinomial larga entre x^t y el polinomio primitivo $f(x)$ y se extraen los residuos, se obtiene la totalidad de los elementos de $GF(q)$ en formato polinomial. Esa es otra forma de crear un campo de extensión.

Ejemplo: Listando los elementos de $GF(2^2)$ en formato exponencial:

$$GF(2^2) = \{x^{-1},\ x^0,\ x^1,\ x^2\} = \{0,\ 1,\ x^1,\ x^2\}$$

Aplicando el algoritmo de división polinomial larga para hallar los elementos correspondientes en el formato polinomial:

$$mod\left(\frac{0}{x^2+x+1}\right) = 0$$

$$mod\left(\frac{1}{x^2+x+1}\right) = 1$$

$$mod\left(\frac{x}{x^2+x+1}\right) = x$$

$$mod\left(\frac{x^2}{x^2+x+1}\right) = x+1$$

Resumiendo la correspondencia entre los diferentes formatos en una tabla:

Exponencial	Polinomial
x^{-1}	0
x^0	1
x	x
x^2	$x+1$

Tabla 5. Elementos de $GF(2^2)$ en formato exponencial y polinomial.

Notar en este caso que a medida que el exponente aumenta, el grado de un polinomio aumenta, esto no siempre es así y se puede ver en la tabla siguiente donde se muestra los elementos de $GF(2^3)$ en los tres formatos:

Exponencial	Polinomial	Vectorial
x^{-1}	0	000
x^0	1	001
x^1	x	010
x^2	x^2	100
x^3	$x+1$	011
x^4	x^2+x	110
x^5	x^2+x+1	111
x^6	x^2+1	101

Tabla 6. Elementos de $GF(2^3)$ en los tres formatos.

3.3.6 OPERACIONES ENTRE ELEMENTOS

En la sección 3.3.1 se hizo una introducción a las operaciones adición y producto para los campos primos. A continuación se va a generalizar las operaciones para los campos de extensión y se define las operaciones sustracción y división.

3.3.6.1 Adición

Para los campos primos $GF(p)$ basta con hacer la suma entera y hacer la reducción módulo p. En los campos de extensión se hace la suma polinomial y al resultado se aplica la reducción módulo p.

Ejemplo: Dado $2x+1$ y $2x+2 \in GF(3^2)$, la suma está dada como

$$(2x+1) \ + \ (2x+2) = 4x+3 \ (mod\ 3) = x$$

3.3.6.2 Sustracción

Teniendo en cuenta que en los campos finitos no se trabajan números negativos, hay que buscar un entero positivo equivalente cuando sea necesario. Sea a y b elementos del campo primo $GF(p)$, si $a \geq b$, se hace la resta entera normal y se hace la reducción módulo p, si $a < b$, se busca un valor c positivo equivalente a $-b$ mediante la congruencia $-b \equiv c \ (mod\ p)$, finalmente se hace la suma $a+c$ y luego la reducción módulo p.
Ejemplo:

$$2 \in GF(3),\ 1 \in GF(3),\ 2-1 = 1 \ (mod\ 3) = 1$$

$$1-2 = ? \ \Rightarrow \ -2 \equiv 1 \ (mod\ 3),\ 1+1 = 2 \ (mod\ 3) = 2$$

Para los elementos de campos de extensión se aplica el mismo concepto para los coeficientes de los polinomios, haciendo la reducción módulo p al resultado final.
Ejemplo:

$$2x+1 \in GF(3^2),\ 2x+2 \in GF(3^2),\ (2x+1)-(2x+2) = ?$$

$$2-2 = 0 \ (mod\ 3) = 0$$

$$1-2 = ? \ \Rightarrow \ -2 \equiv 1 \ (mod\ 3),\ 1+1 = 2$$

$$(2x+1)-(2x+2) = 0x+(1+1) = 2$$

3.3.6.3 Producto

La operación producto en los campos finitos es igual de sencilla a la operación adición. Para $GF(p)$ se hace el producto entero común y luego se aplica la reducción módulo p al resultado. Ejemplos:

$$2 \bullet 1 = 2 \ (mod\ 3) = 2$$

$$2 \bullet 2 = 4 \ (mod\ 3) = 1$$

$$2 \bullet 0 = 0 \ (mod\ 3) = 0$$

En los campos de extensión se hace un producto entre polinomios y posteriormente se hacen dos reducciones, módulo p para reducir los coeficientes mayores o iguales a p, y reducción módulo $f(x)$ si el resultado de la multiplicación tiene un grado mayor o igual a n.

Ejemplo: El producto entre $2x + 1$ y $2x + 2$ es:

$$(2x + 1)(2x + 2) = 4x^2 + 6x + 2 \ (mod\ 3) = x^2 + 2$$

$$x^2 + 2 \ (mod\ x^2 + x + 2) = 2x$$

3.3.6.4 División

La forma de hacer la división entre elementos de campos primos es algo diferente a la división que todos conocemos para los enteros y los reales. Si tenemos el campo $GF(p)$ y queremos hacer la división a/b habría que buscar un valor c tal que multiplicado por b sea igual a a, teniendo en cuenta la reducción módulo p. Ejemplo:

Sea $4 \in GF(5)$ y $3 \in GF(5)$ hallar $4/3$. Si la operación estuviera definida para $\mathbb{R}$, se puede decir que $4/3 = 1.3333$, pero la operación está definida únicamente para $GF(5)$ por lo tanto no se pueden usar los reales, sólo se debe usar los valores del campo como respuesta. Ahora, sabemos de memoria un número multiplicado por 3 tal que al reducir el resultado aplicando módulo 5 sea igual a 4? Al menos que

tengamos a la mano una tabla de producto para $GF(5)$ será un poco dificil conocer el valor inmediatamente. Podemos aplicar un algoritmo iterativo para hallar el valor c mediante fuerza bruta:

$$3 \bullet 1 = 3 \ (mod\ 5) = 3$$

$$3 \bullet 2 = 6 \ (mod\ 5) = 1$$

$$3 \bullet 3 = 9 \ (mod\ 5) = 4$$

De modo que el valor buscado era 3. Ahora este puede parecer un algoritmo rudimentario para hallar el valor de la división, pero funciona bastante bien incluso para valores grandes de p. Si se desea optimizar el proceso de la división podemos recurrir al algoritmo extendido de Euclides que también es un algoritmo iterativo pero requiere menos pasos, o se puede usar tablas de logaritmos discretos para el campo finito $GF(5)$ y hacer restas entre logaritmos, ya que la sustracción de logaritmos es igual a la división, pero hacer esto implica construir tablas para cada p. Para efectos de simplicidad, en este proyecto se va a trabajar con el algoritmo de fuerza bruta.

En el caso de campos de extensión el algoritmo para hallar la división es el mismo, solo que esta vez se está trabajando con polinomios en vez de enteros. Ejemplo:

Sea $2x + 1 \in GF(3^2)$ y $2x + 2 \in GF(3^2)$, hallar $(2x + 1)/(2x + 2)$.
Aplicando el algoritmo iterativo:

$$(2x + 2) \bullet 1 = 2x + 2 \ (mod\ 3) = 2x + 2 \ (mod\ x^2 + x + 2) = 2x + 2$$

$$(2x + 2) \bullet x = 2x^2 + 2x \ (mod\ 3) = 2x^2 + 2x \ (mod\ x^2 + x + 2) = 2$$

$$(2x + 2) \bullet (x + 1) = 2x^2 + 4x + 2 \ (mod\ 3) = 2x^2 + x + 2 \ (mod\ x^2 + x + 2) = 2x + 1$$

El valor buscado es $x + 1$.

3.3.6.5 Potenciación

La potenciación es una multiplicación sucesiva, seguido de una reducción módulo p o módulo $(p, f(x))$ dependiendo si se está trabajando con campos primos o campos

de extensión. Ejemplos:

Dado $2x+4 \in GF(5^2)$ definido por el polinomio primitivo $f(x) = x^2+x+2$, hallar $(2x+4)^3$:

$$(2x+4)^3 = 8x^3+48x^2+96x+64 \ (mod\ 5) = 3x^3+3x^2+x+4 \ (mod\ x^2+x+2) = 4$$

3.3.7 ARITMETICA DE POLINOMIOS SOBRE CAMPOS PRIMOS

La aritmética de los polinomios sobre $GF(p)$ es similar a la aritmética entre elementos de campos de extensión, con algunas diferencias:

1. Los polinomios a operar, $f(x)$ y $g(x)$, están definidos sobre un campo $GF(p)$, es decir sus coeficientes deben pertenecer al campo primo pero $f(x)$ y $g(x)$ puede tener cualquier grado.

2. La operación aritmética entre $f(x)$ y $g(x)$ se hace como una operación normal entre polinomios, sea adición, sustracción, producto o división, pero al resultado final se le aplica la reducción módulo p solamente.

3. Si la operación es adición o sustracción los polinomios deben tener el mismo grado, para producto y división pueden tener grados diferentes.

Ejemplo: Sea $f(x) = x^2+x+4$, $g(x) = x^2+1$ y $h(x) = x^5+1$ polinomios definidos sobre $GF(5)$ tenemos:

$$f(x)+g(x) = 2x^2+x+5 \ (mod\ 5) = 2x^2+1$$

$$f(x)-g(x) = x+3 \ (mod\ 5) = x+3$$

$$f(x) \bullet g(x) = x^4+x^3+5x^2+x+4 \ (mod\ 5) = x^4+x^3+x+4$$

Aplicando el algoritmo de división polinómica larga módulo 5 entre $h(x)$ y $f(x)$, se

tiene como resultado un cociente $q(x)$ y un residuo $r(x)$ ya reducido:

$$\frac{h(x)}{f(x)} = q(x)f(x) + r(x)$$

$$\frac{h(x)}{f(x)} = (x^3 + 4x^2 + 2x + 2)(x^2 + x + 4) + 3$$

$$q(x) = x^3 + 4x^2 + 2x + 2 \; ; \; r(x) = 3$$

3.3.8 APLICACIONES DE CAMPOS FINITOS

Los campos finitos son usados principalmente en el área de la criptografía y en los algoritmos de control de errores. Los campos forman la columna vertebral del algoritmo criptográfico AES, ya que la mayoria de las operaciones matemáticas son realizadas sobre campos de galois. En el área de control de errores los campos juegan un papel central en los códigos de Hamming, usados para corregir errores en la RAM. A continuación se va a describir brevemente los dos algoritmos.

3.3.8.1 El algoritmo AES

El algoritmo AES (Advanced Encryption Standard) es un esquema de cifrado por bloques cuya unidad de cifrado es de 128 bits y cuya llave secreta puede ser de 128, 192 o 256 bits. Una unidad de cifrado por bloques se podría ver como un algoritmo que toma un arreglo de bytes determinados de texto plano y lo transforma en texto encriptado. Asumiendo que un byte es igual a 8 bits, el tamaño fijo del bloque es de 16 bytes. El algoritmo AES usa un campo $GF(2^8)$ donde realiza la mayoria de las transformaciones necesarias para convertir el texto plano en texto encriptado. Los detalles precisos de como se realiza este proceso están fuera del alcance de este libro, habría que dedicar un proyecto de grado completo para el tratamiento de este algoritmo. Lo importante era ilustrar la importancia del campo en el proceso.

3.3.8.2 Códigos de Hamming

El código de Hamming[17] es un código linear utilizado para determinar si una serie de bits transmitidos por una red llegaron con errores. Es un código detector y corrector de errores. En los datos codificados en Hamming se pueden detectar errores en un bit y corregirlos, sin embargo no se distingue entre errores de dos bits (para lo que se usa Hamming extendido). Esto representa una mejora respecto a los códigos con bit de paridad, que pueden detectar errores en sólo un bit y no pueden corregirlos.

El código de Hamming que conocemos actualmente es el código denominado $(7, 4)$ que codifica tres bits de información por cada cuatro bits de datos del mensaje, produciendo una palabra de 7 bits. Para obtener esta palabra codificada, necesitamos una matriz de chequeo de paridad H de 3×7 y una matriz generadora G de 4×7, que se pueden obtener mediante cualquier otro paquete que maneje librerias de código de Hamming, como Matlab:

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

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

Los detalles de cómo son generadas las matrices está fuera del alcance del libro, vale decir que ambas matrices son estándar para los códigos de tipo $(7, 4)$.

Luego de generar las matrices, se toma un elemento del campo $GF(2^4)$ en formato vectorial, por ejemplo $a = (1001)$ que será el dato a transmitir, y a continuación se

[17]Richard Hamming(1915 - 1988): Matemático estadounidense que inventó los códigos que llevan su nombre.

hace la multiplicacion matricial:

$$cod = a * G = (0111001)$$

obteniendo la información codificada (cod) que se va a transmitir.
Simulando un error de transmisión, se cambia el bit 3 de 1 a 0, obteniendo $rec = (0101001)$ que será la información recibida.
Para examinar en qué bit ocurrió el error, el receptor debe hacer la multiplicación de la información recibida (rec) con la matriz chequeo de paridad:

$$rec * H = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} * H = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

El 1 en la última fila indica que ocurrió un error (el vector resultado debe estar en cero para concluir que no ocurrieron errores de transmisión), comparando el vector columna con todos las columas de H para hallar alguna equivalencia, se concluye que el error ocurrió en el tercer bit, y posteriormente se corrige.

3.4 SOFTWARE LIBRE EN LOS PROYECTOS DE GRADO

La filosofía del software libre fue una de las motivaciones fundamentales a la hora de trabajar en este proyecto de grado. En esta sección se hará una síntesis del movimiento Open Source[18]: antecedentes, protagonistas, licencias, y casos de éxito. Se hará especial énfasis en la importancia que ha adquirido el software libre como producción intelectual en las universidades, ya sea en grupos de investigación o como trabajos de grado, y cómo la Universidad puede hacer cambios en el reglamento de la producción intelectual de la universidad para que tanto el creador como la UIS

[18]Open Source: Software libre en inglés.

puedan beneficiarse de esta nueva cultura de hacer software.

3.4.1 INTRODUCCION AL SOFTWARE LIBRE

El software libre consiste en un conjunto de herramientas informáticas liberadas bajo una serie de licencias que permiten además de su uso, su estudio, su modificación, su reutilización y su redistribución con ninguna o pocas restricciones. Que una aplicación libre pueda ser estudiada y modificada implica que el código fuente está disponible para ser examinado en busca de errores o para extender su funcionalidad sin ninguna restricción por parte de los diseñadores originales de la aplicación. De igual forma, la aplicación resultante puede ser distribuida nuevamente sin ninguna restricción, con la condición de que se mantenga la licencia libre original y las referencias a los autores originales.

La idea original nace en el seno de la comunidad de científicos, programadores y entusiastas de la computación en el MIT[19], alrededor de la década de 1970, donde existía una cultura de compartir software sin restricciones, e incluso las grandes empresas de la época como IBM y DEC[20] alentaban esa tendencia con el fin de que existiera un ámplio catálogo de aplicaciones para sus mainframes. Todo esto terminó con la creciente industria del software y los crecientes costos al crear software, hicieron que se comenzara a imponer derechos de autor y restricciones sobre los programas.
Richard Stallman, quien era miembro de la comunidad de investigadores del laboratorio de inteligencia artificial del MIT, observó con preocupación estos cambios en la industria de la computación, y anunció el proyecto GNU en 1983 como plataforma legal y colaborativa para crear software libre. El proyecto GNU es responsable de las licencias GPL[21], usadas en la mayoría de proyectos de código abierto, y de aplicaciones clave en las distribuciones Linux como el compilador GCC[22] y el entorno gráfico GNOME.

[19]MIT: Massachusetts Institute of Technology, Universidad pionera en ciencias aplicadas de Estados Unidos

[20]DEC: Digital Equipment Corporation: Fabricante de mainframes y minicomputadoras en las primeras décadas del siglo XX.

[21]GPL: General Public License, Licencia pública general, versiones 1, 2 y 3.

[22]GCC: GNU C Compiler. Compilador C de GNU

Hay que hacer una acotación al término "libre". Lo anterior no significa que el software deba distribuirse a ningún costo, implica más bien las libertades de uso que llevan consigo las licencias de código abierto. Los desarrolladores de software libre pueden cobrar una tarifa por la distribución de sus productos a los usuarios finales, aunque estos tienen la libertad de redistribuir el software sin ningún costo. Es por esta razón que el modelo de negocios en el software abierto está orientado al soporte de las aplicaciones: acceso a actualizaciones y correcciones de seguridad, módulos que extienden la funcionalidad del sistema, soporte personalizado, entre muchos otros beneficios extras.

Resumiendo, el tipo de licencias utilizadas en el Software Libre tratan de garantizar cuatro libertades básicas:

1. Libertad para usar el producto con cualquier propósito, incluidos el académico y el comercial.

2. Libertad de distribuir el producto a otras personas.

3. Libertad de estudiar el código fuente del producto, es decir, la libertad de saber cómo fue construido y cómo funciona, y poder modificarlo.

4. Libertad de distribuir los trabajos derivados de las modificaciones hechas al producto original.

En la actualidad hay una amplia gama de licencias de código abierto disponibles para todo tipo de proyectos, desde herramientas informáticas, librerías y documentos de ayuda, hasta libros, imágenes y obras musicales. La licencia mas difundida actualmente es la GPL en sus versiones 1, 2 y 3. El kernel de Linux, el entorno gráfico KDE, El proyecto MONO entre muchos otros están bajo esa licencia. Otro tipo de licencias ampliamente utilizadas son el conjunto de licencias BSD[23], usadas en los sistemas operativos FreeBSD, OpenBSD y NetBSD. Scilab usa la licencia libre

[23]BSD: Berkeley Software Distribution. Definida por la Universidad de Berkeley. Recibe el mismo nombre del sistema Unix

CeCILL[24], adaptada a las leyes francesas y las de la Unión Europea, pero totalmente compatible con la licencia GPL.

3.4.2 EL SOFTWARE LIBRE Y LA UNIVERSIDAD

Muchas universidades del mundo han sido capaces de desprenderse de las retribuciones económicas por la explotación de su producción intelectual, pasando a preocuparse por el cumplimiento de sus propósitos más altos, como la apropiación, creación, estudio y difusión del conocimiento para que éste contribuya efectivamente al mejoramiento del bienestar de la sociedad.

Uno de los ejemplos más impresionantes es el de la Universidad de Berkeley, en los Estados Unidos, donde se han desarrollado innumerables programas, protocolos, sistemas operativos y algoritmos,una gran mayoria disponibles a la comunidad científica bajo las licencias BSD. Las primeras implementaciones de los protocolos TCP/IP fueron desarrollados por esa Universidad, quienes luego hicieron público el código que ahora es usado en los sistemas operativos Windows y en los sistemas Unix. El servidor DNS[25] por defecto en la Internet, BIND[26], fue igualmente desarrollado por la universidad y liberado mediante la licencia BSD.

Actualmente la mayor dificultad que se deben sortear en la UIS para hacer realidad la producción intelectual bajo licencias abiertas está en la normativa de los derechos patrimoniales de los proyectos de grado. Cuando un estudiante entrega el proyecto de grado para su posterior calificación, debe firmar un documento en el que cede todos los derechos de reproducción, comunicación pública, transformación y distribución (alquiler, préstamo público o importación) de los trabajos de grado. El autor del proyecto no puede modificar, publicar, transmitir, participar en la transferencia o venta; crear trabajos derivados o de ninguna forma o explotar el contenido en parte o completamente.

[24]CeCILL: CEA CNRS INRIA Logiciel Libre. Definido por las agencias francesas CEA, CNRS y INRIA
[25]DNS: Domain Name System.
[26]BIND: Berkeley Internet Name Domain.

Esto se podría interpretar que sería ilegal por parte del autor del proyecto liberar cualquier herramienta desarrollada en los trabajos de grado como software libre, sin el consentimiendo explícito de la UIS como dueña de los derechos del trabajo. La UIS debería modificar el acta de propiedad intelectual para permitir explícitamente la protección los proyectos de grado con licencias libres, tal como ya se ha hecho en algunas universidades colombianas como la Universidad del Cauca.

4. DESARROLLO DEL PROYECTO

4.1 METODOLOGIA Y HERRAMIENTAS PROPUESTAS PARA EL DESARROLLO DE LA HERRAMIENTA

4.1.1 METODOLOGIA DE SOFTWARE

Después de haber examinado diferentes modelos de Ingeniería de Software, se decidió que el Modelo Incremental es el mejor que se ajusta a la naturaleza de este proyecto. El modelo combina elementos del modelo en cascada con la filosofía iterativa del modelo de prototipos. La puesta en marcha de esta técnica de desarrollo de software es sencilla:

a. Como primer paso, se aplica el modelo secuencial (proceso lineal de análisis, diseño, desarrollo y puesta en marcha) a un grupo de requerimientos iniciales del sistema, el resultado es un incremento del software, una aplicación con funcionalidades básicas implementadas, listas a ser ejecutadas. Esta es la primera iteración del modelo.

b. En las iteraciones subsiguientes, se aplica el mismo modelo secuencial lineal a otro grupo de requerimientos, produciendo incrementos adicionales, con más funcionalidades implementadas en el software. El proceso es repetido continuamente, hasta haber agotado todos los requerimientos disponibles, obteniendo como resultado final el sistema completo.

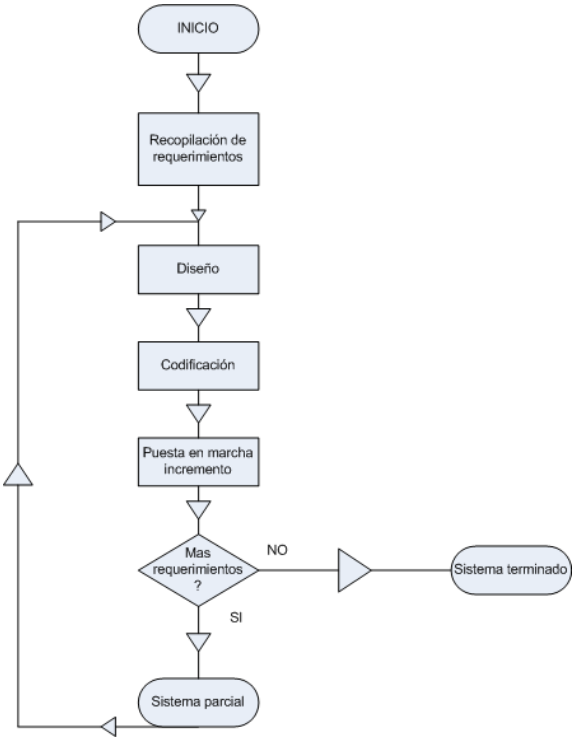

Figura 2. Diagrama de flujo del proceso del modelo secuencial.

El proceso detallado de la aplicación de esta metodología en el proyecto se describe como sigue:

1. Crear el primer incremento: Hacer un proceso de análisis, diseño, generación de código y prueba sobre el primer objetivo: la implementación de funciones para la definición de campos primos y campos de extensión. Este será el núcleo del proyecto, a partir de aquí se pueden crear procedimientos que operen sobre los campos y extraigan información de los campos. Se espera que esta primera iteración ocupe un porcentaje grande dentro del tiempo total del proyecto. Se incluye en este incremento la definición de funciones para hallar elementos primitivos, polinomios mínimos para campos primos, polinomios primitivos, determinar si un polinomio es primitivo o no y hallar listas de polinomios primitivos.

2. Crear el segundo incremento: Realizar la misma secuencia lineal para cumplir el segundo objetivo; representar los elementos del campo en diferentes for-

mas: vectorial (arreglos de enteros), exponencial, polinomio y en forma de tablas (para campos primos). Esta segunda iteración no representa un porcentaje grande dentro del tiempo total del proyecto, ya que los algoritmos para mostrar los diferentes modos de expresar los campos no representan complejidad matemática alguna.

3. Crear el tercer incremento: Soportar las operaciones de adición, sustracción, multiplicación, división y potenciación entre los elementos de los campos. El sistema implementa dos modos para cada operación, una para operar campos primos y otra para operar campos de extensión. Esta iteración supone también un buen porcentaje de tiempo sobre el proyecto.

4. Crear el cuarto incremento: Habilitar operaciones aritméticas entre polinomios sobre campos primos. No supone gran complejidad debido a que ya se tiene las funciones disponibles en el tercer incremento.

5. Crear el quinto incremento: Con el objetivo de corroborar las aplicaciones clave de los campos finitos, se va a implementar funciones para asistir en tareas de criptografía y control de errores. En este quinto incremento se implementa una pequeña rutina que involucra códigos de Hamming y campos finitos para simular errores en una trasmisión de datos y posterior chequeo de los errores. Se implementará la función Phi de Euler, que es elemento clave en los algoritmos criptográficos.

Despues de la quinta iteración, se tiene como resultado un toolbox de módulos y librerías listas para ser cargadas en memoria y utilizadas.

4.1.2 ENTORNO DE SCILAB

Como se estableció en el tema y en los objetivos del proyecto, se va a trabajar con Scilab para desarrollar el módulo de campos finitos. Scilab es un paquete matemático para cálculo científico, interactivo de libre uso y disponible para múltiples sistemas operativos (Unix, GNU/Linux, Windows, Solaris) desarrollado por INRIA y la ENPC desde 1990. Scilab fue creado para hacer cálculos numéricos aunque también

ofrece la posibilidad de hacer algunos cálculos simbólicos como derivadas de funciones polinomiales y racionales. Posee cientos de funciones matemáticas y la posibilidad de integrar programas en los lenguajes más usados (FORTRAN, Java y C). Scilab fue hecho para ser un sistema abierto donde el usuario puede definir nuevos tipos de datos y operaciones entre los mismos.

Scilab viene con numerosos módulos de gráficos 2-D y 3-D, animación, álgebra lineal, matrices dispersas, polinomios y funciones racionales, Simulación: programas de resolución de sistemas de ecuaciones diferenciales (explícitas e implícitas), Scicos: simulador por diagramas en bloque de sistemas dinámicos híbridos, Control clásico, robusto, optimización LMI, optimización diferenciable y no diferenciable, tratamiento de señales, grafos y redes, además de un módulo de estadística y funciones para hacer pequeños calculos simbólicos. También tiene incorporado un lenguaje de programación propio similar en sintaxis al lenguaje de Matlab.

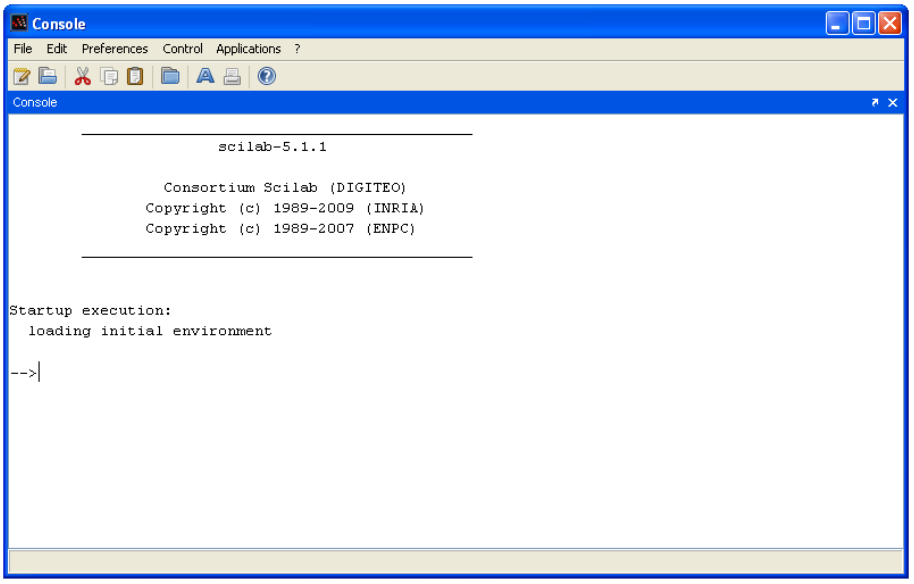

Figura 3. Interfaz de Scilab.

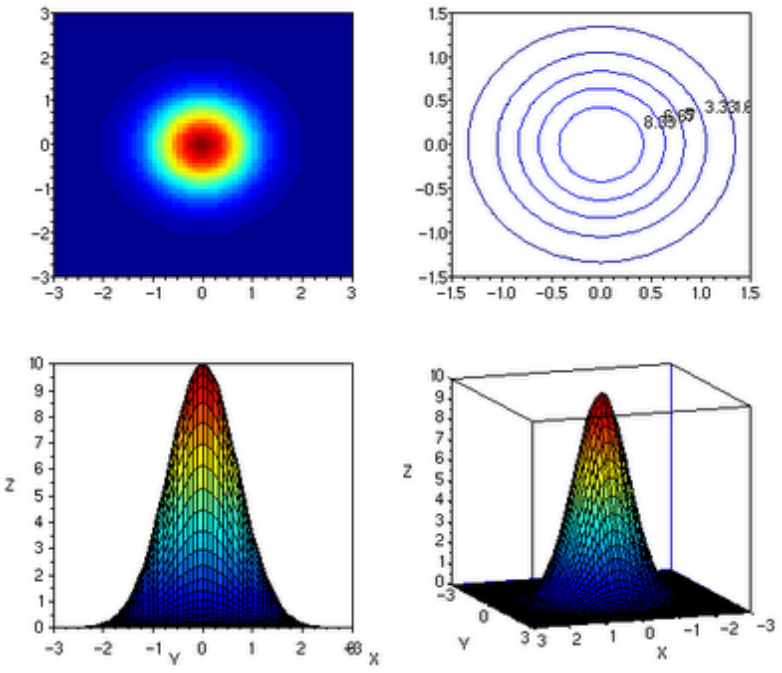

Figura 4. Otro ejemplo de la interfaz de Scilab.

Scilab dispone de una serie de utilidades para desarrolladores que deseen crear toolboxes. Con estas herramientas se puede crear el esqueleto de directorios y archivos donde se almacenará el proyecto, crear archivos de ayuda y de documentación del proyecto, configurar Scilab para usar compiladores externos de C y Fortran, crear scripts de compilación y enlazado, compilar y enlazar el proyecto completo e incluso habilitar opciones de depuración.

Para este proyecto se va a trabajar con la versión **5.1.1** de Scilab. La instalación del paquete es simple, el asistente de instalación solicita la información necesaria y hace la copia de archivos en el directorio correspondiente.

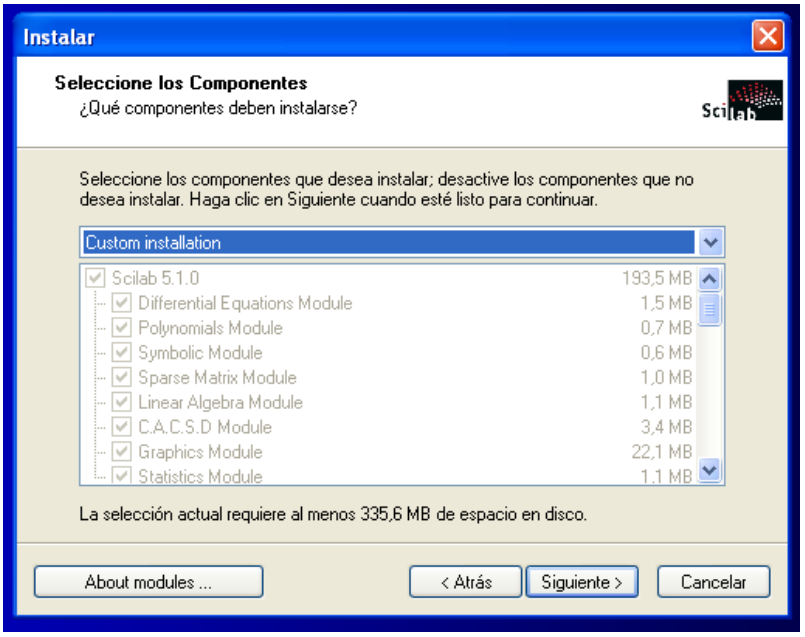

Figura 5. Instalación de Scilab.

Después de la instalación hay que dirigirse al subdirectorio "modules" en el directorio de instalación, que en este caso es "C:\Archivos de programa\scilab-5.1.1\", posteriormente se crea el directorio "campos_finitos" y allí se crean los siguientes subdirectorios y archivos necesarios para iniciar el proyecto:

- **demos:** Contiene archivos *.sce en los que están definidos programas de ejemplo que ilustran los usos de las funciones definidas por el toolbox.
- **etc:** Están los archivos campos_finitos.start y campos_finitos.quit que ejecutan instrucciones de Scilab cuando se carga y descarga de memoria el toolbox respectivamente.
- **help:** Contiene los archivos de ayuda.
- **jar:** Archivados ejecutables de java para cargar la ayuda.
- **macros:** Contiene archivos *.sci en los que están definidos funciones del toolbox en el lenguaje de Scilab.

- **src:** Almacena archivos *.c en los que están definidos las rutinas principales en C del toolbox.
- **sci_gateway:** Almacena archivos en *.c donde están definidos funciones interfaces entre Scilab y las rutinas C almacenadas en el el directorio src.
- **builder.sce:** Contiene instrucciones en Scilab para compilar y enlazar las archivos C en librerías.
- **loader.sce:** Contiene instrucciones para cargar en memoria las librerías *.dll y los macros.
- **changelog.txt:** Información de los cambios y nuevas funcionalidades.
- **license.txt:** Información de la licencia.
- **readme.txt:** Información general de la herramienta.

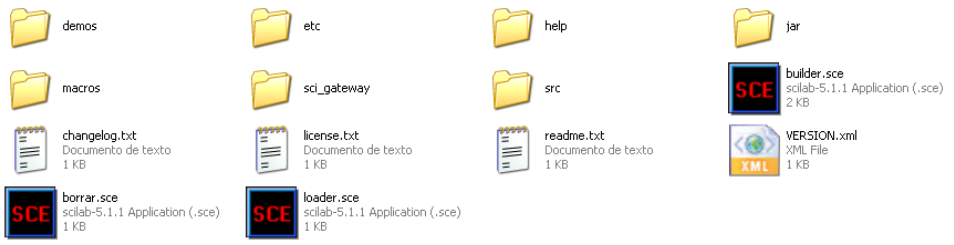

Figura 6. Estructura del toolbox en el sistema de archivos.

4.1.3 HERRAMIENTAS DE DESARROLLO

En el plan de proyecto se había planteado trabajar con el compilador LCC-Win32, pero después de ver sus carencias a la hora de depurar código enlazado con programas externos, se decidió por trabajar con el IDE[27] Dev-C++, que además de ser libre, tiene todas las herramientas que se necesitan para desarrollar el proyecto. A continuación se muestran algunas capturas:

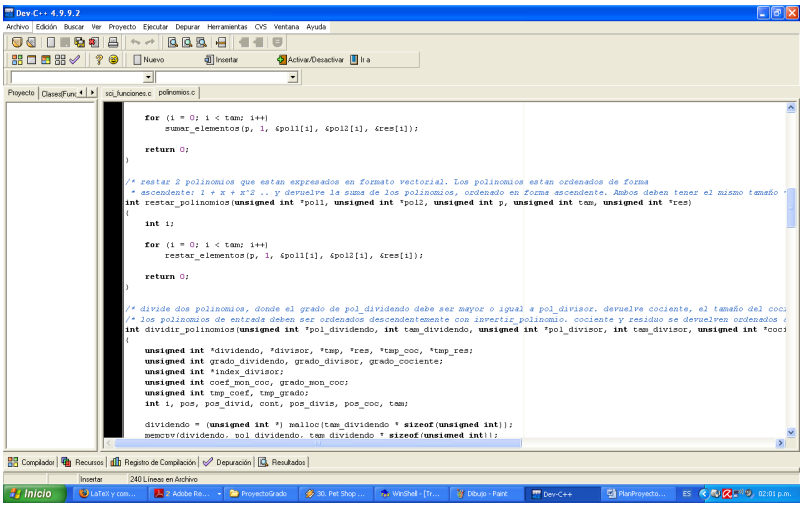

Figura 7. Aspecto del IDE Dev-C++.

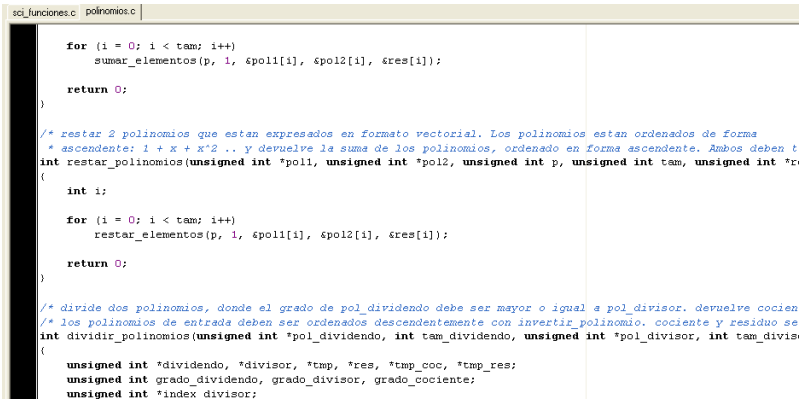

Figura 8. Aspecto del IDE Dev-C++.

[27]IDE: Integrated Development Environment, Entorno Integrado de Desarrollo.

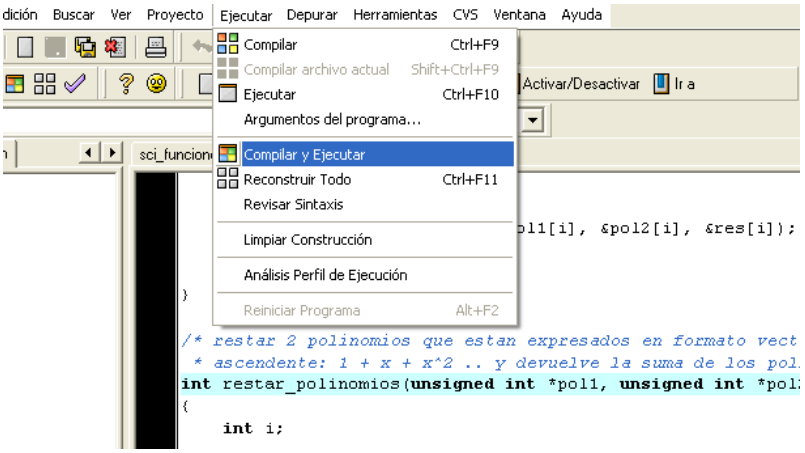

Figura 9. Aspecto del IDE Dev-C++.

4.1.4 PARADIGMA DE PROGRAMACION

Un paradigma de programación es un estilo particular de programación que determina la forma de almacenar datos en memoria, representar elementos (objetos, variables, constantes) en un programa y cómo debe ser el flujo de ejecución de las instrucciones en la aplicación. En la actualidad los paradigmas de programación más extendidos son:

- Programación orientada a objetos: Los datos y los métodos para manipular los datos son empaquetas en una unidad llamada objeto. La única forma en que un usuario puede acceder a los datos es mediante los métodos implementados en los objetos. Así pues, el funcionamiento de un objeto puede alterarse sin afectar el código que usa el objeto. La mayoría de los proyectos hoy en día usan este paradigma, y lenguajes como C++ y Java soportan programación orientada a objetos.

- Programación funcional: Los datos y los métodos son empaquetados en funciones que implementan tareas determinadas. Las funciones pueden intercambiar datos mediante los argumentos y las salidas de las funciones, semejante al comportamiento de las funciones matemáticas, de tal forma que el flujo de un programa consiste en una secuencia de evaluación de funciones. Proyectos

complejos como el Kernel de Linux (implementado en C) son desarrollados bajo este paradigma.

- Programación orientada a componentes: En este paradigma el nivel de abstracción es mayor, ya que se encapsula un módulo completo (que puede contener objetos y funciones) para realizar una tarea determinada dentro del sistema, de tal forma que la aplicación será un conjunto de módulos que intercambian datos entre sí.

El API[28] de Scilab obliga a los desarrolladores a usar rutinas en C o en Fortran para intercambiar datos entre Scilab y las aplicaciones implementadas por el programador, de modo que la decisión más razonable es usar la metodología de programación funcional para el proyecto, empleando el lenguaje C para codificar las rutinas. Otras razones más para justificar la elección de esta metodología es la abundante cantidad de ejemplos disponibles en la página de Scilab para crear módulos y librerías en C, y la dificultad de compilar y enlazar código en C++ en este entorno. Para algunas rutinas complejas (operaciones entre polinomios e impresión de varios formatos de salida para la información de los campos) se empleará el lenguaje de programación de Scilab, que permite una programación sencilla y una ejecución rápida, por ser un lenguaje de alto nivel interpretado directamente por el entorno matemático.

[28] API: Application Program Interface, Interfaz de programación de aplicaciones.

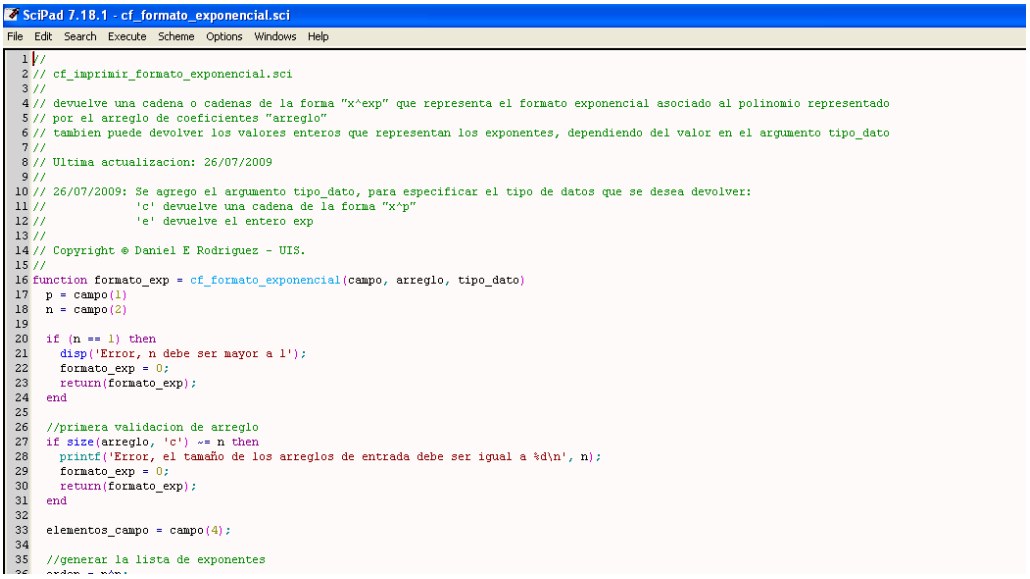

Figura 10. Ejemplo de una rutina codificada en el lenguaje de Scilab.

4.2 DISEÑO E IMPLEMENTACION DEL MODULO

4.2.1 ESTRUCTURAS DE DATOS PRINCIPALES

El proyecto en sí es relativamente sencillo de modelar, ya que no hay intercambio de información entre bases de datos y el módulo, la información se guarda en estructuras de datos que son la entrada de las diferentes funciones que componen la herramienta. La información no se necesita almacenar para un posterior uso ya que las funciones siempre devolverán los mismos resultados para un p y un n dado. La fuerza de la herramienta radica en qué tan óptimos son los algoritmos para implementar las divesas rutinas y operaciones matemáticas planteadas en el marco téorico. Básicamente se trabajó con dos estructuras:

Estructura Campo Primo:

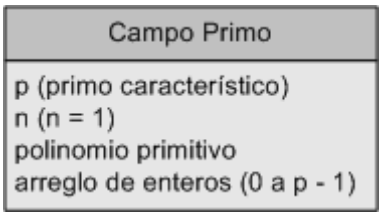

Figura 11. Estructura Campo Primo.

Esta es la estructura mas sencilla, representa un Campo Primo, está compuesta de los siguientes elementos:

1. **P:** Es el primo característico del campo. Se almacena como un entero sin signo de 32 bits.

2. **N:** Grado del campo, para un campo primo $n = 1$. Se almacena como un entero sin signo de 32 bits.

3. **Polinomio primitivo:** Aparentemente para un campo primo el polinomio primitivo no tendría sentido, ya que este se usa para efectuar una extensión algebraica sobre un campo dado, y $GF(p)$ no es una extensión algebraica (todos sus elementos son enteros positivos). Pero para efectos de compatibilidad con otros paquetes matemáticos (mas específicamente Matlab y Mathematica) expresamos el polinomio primitivo de un campo primo como un polinomio mínimo de grado 1 cuya raíz es un elemento primitivo de $GF(p)$. Por ejemplo, si $t = 2$ es elemento primitivo de $GF(5)$, el polinomio primitivo será:

   $$f(x) = x + (5 - 2) = x + 3$$

   Este elemento se almacena como un arreglo de dos enteros sin signo de 32 bits, que representan los coeficientes del polinomio ordenados **ascendentemente** de acuerdo al grado del monomio.

3	1

Figura 12. Representación en la memoria del polinomio $x + 3$.

Note que a lo largo de todo el marco teórico se ha representado los polinomios ordenados de forma descendentemente (de grado mayor a menor) mientras que en la figura 11 se muestra que los coeficientes de los polinomios se están ordenando en la memoria de menor a mayor de acuerdo a su grado. Es decir, si se representa en formato polinomial el vector que almacena el polinomio primitivo, se vería que:

$$f(x) = 3 + x$$

Se usa esta ordenación ascendente debido a que en Scilab los polinomios se trabajan de forma ascendente, por lo que en las estructuras de datos como en las funciones que manipulan polinomios en el toolbox se trabajará bajo esta ordenación.

4. **Arreglo de enteros:** Los elementos del campo primo. Se almacenan como un arreglo de p elementos (de 0 a $p - 1$) sin signo de 32 bits.

0	1	2	3	4

Figura 13. Representación en la memoria de los elementos de $GF(5)$.

Estructura Campo de Extensión:

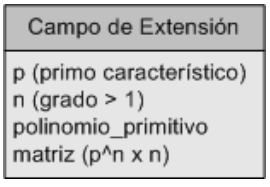

Figura 14. Estructura Campo de Extensión.

Esta es la estructura principal del programa, representa a un Campo de Extensión, a continuación se describe cada elemento:

1. **P:** Es el primo característico del campo. Se almacena como un entero sin signo de 32 bits.

2. **N:** Grado del campo. Se almacena como un entero sin signo de 32 bits.

3. **Polinomio primitivo:** Arreglo de $n + 1$ enteros sin signo de 32 bits que representan los coeficientes del polinomio primitivo $f(x)$ ordenados ascendentemente.

2	1	1

Figura 15. Representación en la memoria del polinomio $x^2 + x + 2$.

4. **Matriz:** Esta matriz de $p^n \times n$ enteros sin signo de 32 bits almacenan los vectores coeficientes ordenados ascendentemente que representan los polinomios elementos del campo $GF(p^n)$. Por ejemplo, para el campo $GF(2^2) = \{0,\ 1,\ x,\ x+1\}$, la matriz tendría la siguiente estructura:

x^0	x^1
0	0
1	0
0	1
1	1

Tabla 7. Almacenamiento en memoria de los elementos de $GF(2^2)$.

Este par de estructuras son en conjunto el núcleo de de la herramienta ya que representan una instancia de un campo finito, además de ser la entrada de la mayoría de las funciones definidas en el proyecto.

4.2.2 FUNCIONES A IMPLEMENTAR

Las funciones a implementar en el proyecto, junto con la descripción de las entradas y salidas, se muestra a continuación. Note que cada nombre de la función comienza con el prefijo "cf", para diferenciar a este grupo de rutinas frente a otras funciones y módulos en Scilab.

1. **cf_crear_campo:** Crea un campo de acuerdo con los valores de p, n y $f(x)$ dados.
   - **Entrada:** p, n y $f(x)$.
   - **Salida:** Una estructura que representa un campo primo o un campo de extensión.
2. **cf_orden:** Devuelve el orden de un campo.
   - **Entrada:** Un campo $GF(p^n)$.
   - **Salida:** El orden (número de elementos) del campo de entrada.
3. **cf_primo_caracteristico:** Devuelve el primo caracteristico de un campo.
   - **Entrada:** Un campo $GF(p^n)$.
   - **Salida:** Primo característico del campo de entrada, p.
4. **cf_grado:** Devuelve el grado de un campo.
   - **Entrada:** Un campo $GF(p^n)$.
   - **Salida:** El grado del campo de entrada, n.
5. **cf_polinomio_primitivo:** Devuelve el polinomio primitivo de un campo.
   - **Entrada:** Un campo $GF(p^n)$.
   - **Salida:** El polinomio primitivo del campo de entrada, $f(x)$.
6. **cf_elementos_campos:** Devuelve los elementos de un campo.
   - **Entrada:** Un campo $GF(p^n)$.

- **Salida:** El arreglo de elementos almacenado dentro del campo de entradas.

7. **cf_es_polinomio_primitivo:** Determina si un polinomio es primitivo o no.
   - **Entrada:** Un polinomio $f(x)$, p y n.
   - **Salida:** 1 si es polinomio primitivo, 0 si no es polinomio primitivo.
8. **cf_elementos_primitivos:** Devuelve los elementos primitivos de un campo.
   - **Entrada:** Un campo $GF(p^n)$.
   - **Salida:** el arreglo de elementos primitivos del campo.
9. **cf_devolver_pol_prim_def:** Devuelve el polinomio primitivo por defecto asociado a p y n.
   - **Entrada:** p y n.
   - **Salida:** Polinomio primitivo.
10. **cf_devolver_pol_prim:** Devuelve una lista de polinomios primitivos dado p y n.
    - **Entrada:** p y n.
    - **Salida:** una lista de polinomios primitivos.
11. **cf_polinomio_minimo:** Devuelve el polinomio mínimo asociado a un elemento del campo primo $GF(p)$.
    - **Entrada:** campo, elemento.
    - **Salida:** polinomio mínimo.
12. **cf_tabla_adicion:** Devuelve la tabla de adición para un campo primo.
    - **Entrada:** Un campo $GF(p)$.
    - **Salida:** La tabla de adición para el campo primo.
13. **cf_tabla_sustraccion:** Devuelve la tabla de sustracción para un campo primo.

- **Entrada:** Un campo $GF(p)$.
- **Salida:** La tabla de sustracción para el campo primo.

14. **cf_tabla_producto:** Devuelve la tabla de producto para un campo primo.
    - **Entrada:** Un campo $GF(p)$.
    - **Salida:** La tabla de producto para el campo primo.
15. **cf_tabla_division:** Devuelve la tabla de división para un campo primo.
    - **Entrada:** Un campo $GF(p)$.
    - **Salida:** La tabla de división para el campo primo.
16. **cf_sumar_elementos:** Suma dos elementos de un campo $GF(p^n)$.
    - **Entrada:** Un campo $GF(p^n)$ y dos elementos del campo.
    - **Salida:** Resultado de la suma.
17. **cf_restar_elementos:** Resta dos elementos de un campo $GF(p^n)$.
    - **Entrada:** Un campo $GF(p^n)$ y dos elementos del campo.
    - **Salida:** Resultado de la resta.
18. **cf_producto_elementos:** Multiplica dos elementos de un campo $GF(p^n)$.
    - **Entrada:** Un campo $GF(p^n)$ y dos elementos del campo.
    - **Salida:** Resultado del producto.
19. **cf_dividir_elementos:** Divide dos elementos de un campo $GF(p^n)$.
    - **Entrada:** Un campo $GF(p^n)$ y dos elementos del campo.
    - **Salida:** Resultado de la división.
20. **cf_potencia_elementos:** Realiza la operación potencia a un elemento del campo $GF(p^n)$.

- **Entrada:** Un campo $GF(p^n)$, la base que debe ser un elemento del campo, y el exponente que es mayor o igual a 0.
- **Salida:** Resultado de la operación potencia.

21. **cf_sumar_polinomios:** Suma dos polinomios definidos sobre un campo $GF(p)$.
    - **Entrada:** dos polinomios $f(x)$ y $g(x)$, p.
    - **Salida:** polinomio resultado de la suma.
22. **cf_restar_polinomios:** Restar dos polinomios definidos sobre un campo $GF(p)$.
    - **Entrada:** dos polinomios $f(x)$ y $g(x)$, p.
    - **Salida:** polinomio resultado de la resta.
23. **cf_producto_polinomios:** multiplica dos polinomios definidos sobre un campo $GF(p)$.
    - **Entrada:** dos polinomios $f(x)$ y $g(x)$, p.
    - **Salida:** polinomio resultado del producto.
24. **cf_dividir_polinomios:** dividir dos polinomios definidos sobre un campo $GF(p)$.
    - **Entrada:** dos polinomios $f(x)$ y $g(x)$, p.
    - **Salida:** dos polinomios, cociente y residuo .
25. **cf_potencia_polinomios:** Implementa la operación potencia sobre un polinomio $f(x)$ definido sobre el campo $GF(p)$.
    - **Entrada:** un polinomio $f(x)$, p, exponente mayor o igual a 0.
    - **Salida:** Resultado de la operación potencia.
26. **cf_formato_polinomial:** Imprime en formato polinomial los elementos del campo $GF(p^n)$.
    - **Entrada:** Un vector que almacena los elementos del campo a convertir en formato polinomial.

- **Salida:** El arreglo de polinomios.

27. **cf_formato_exponencial:** Imprime en formato exponencial los elementos del campo $GF(p^n)$.

    - **Entrada:** Un campo $GF(p^n)$, un arreglo que representa los elementos de campo a convertir en formato exponencial, el caracter 'c' o 'e' que indica que se desea imprimir cadenas polinomicas o el exponente entero.
    - **Salida:** Un arreglo de polinomios o un arreglo de enteros.

28. **cf_funcion_phi_euler:** Halla la funcion phi euler a un entero n.

    - **Entrada:** Un entero n.
    - **Salida:** El valor de la funcion phi euler aplicada a n.

4.2.3 DIAGRAMAS DE CASOS DE USO

Los siguientes diagramas de casos de uso muestran los principales requisitos funcionales del módulo, y su relación con el entorno de Scilab (usuarios y otras funciones):

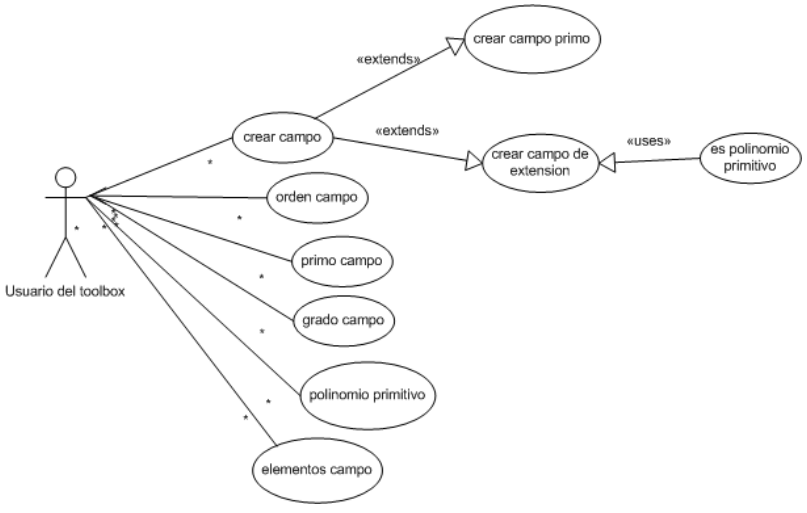

Figura 16. Diagrama de Casos de Uso 1.

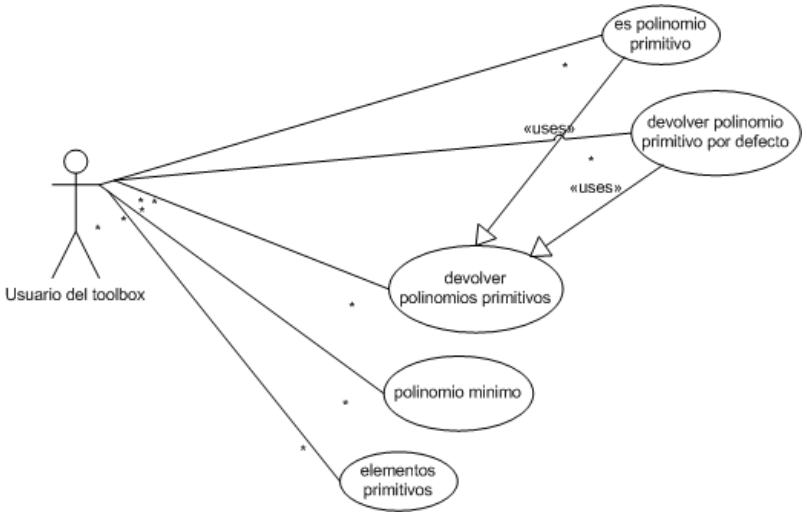

Figura 17. Diagrama de Casos de Uso 2.

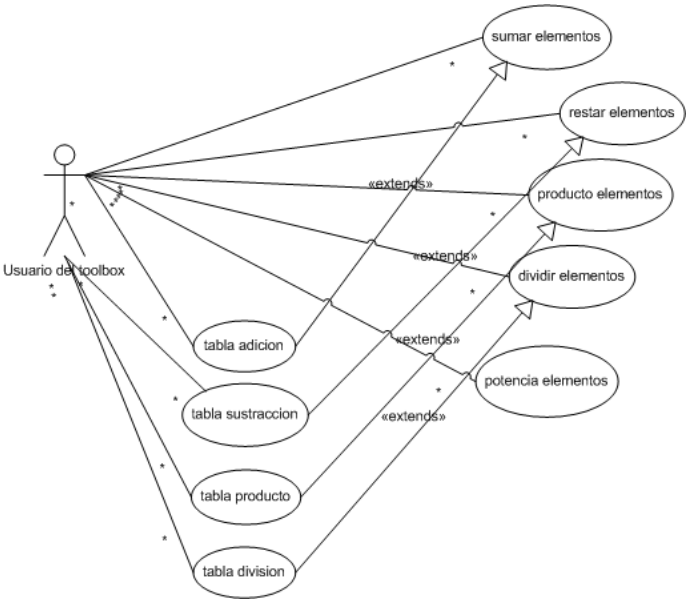

Figura 18. Diagrama de Casos de Uso 3.

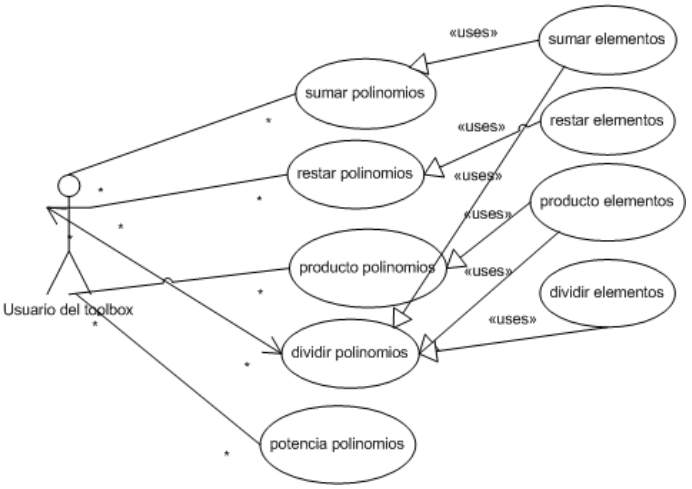

Figura 19. Diagrama de Casos de Uso 4.

4.2.4 DIAGRAMAS DE FLUJO

En los siguientes diagramas de flujo se muestra el funcionamiento interno de algunos algoritmos claves del proyecto:

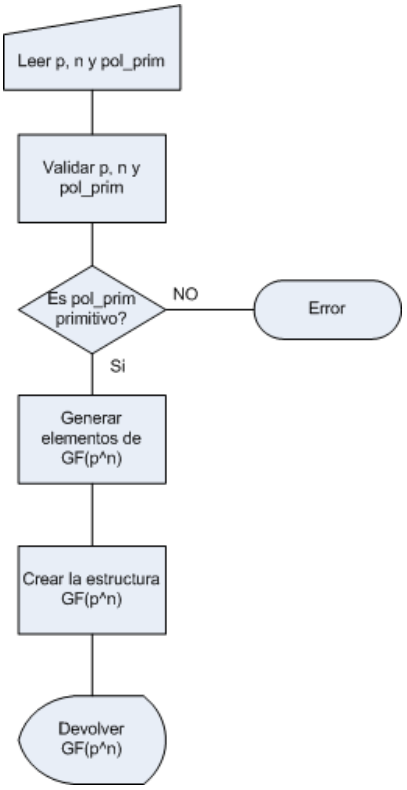

Figura 20. Diagrama de flujo de la función cf_crear_campo.

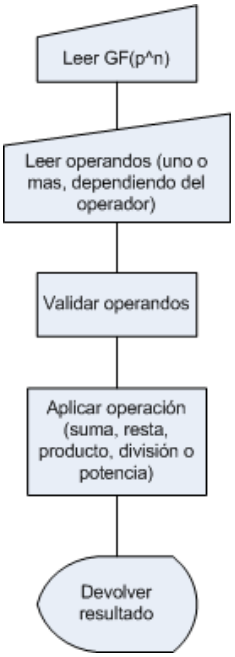

Figura 21. Diagrama de flujo de una operación entre elementos de un campo primo.

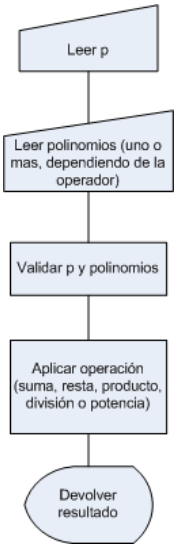

Figura 22. Diagrama de flujo de una operación entre polinomios sobre campos primos.

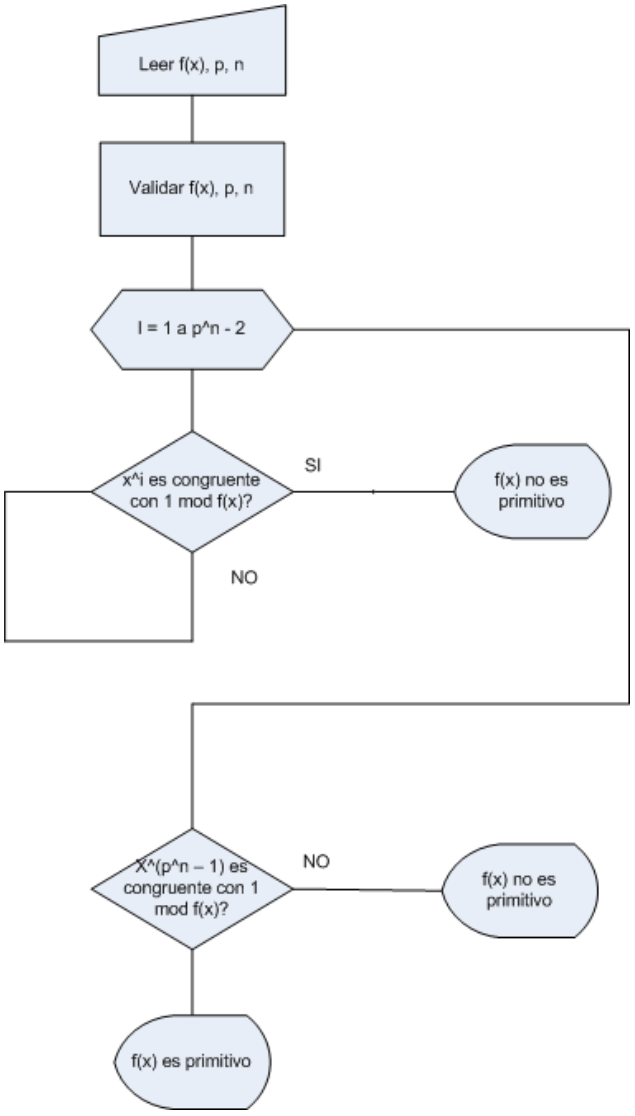

Figura 23. Diagrama de flujo de la funcion cf_es_polinomio_primitivo.

4.2.5 INTERFAZ GRAFICA DEL MODULO

Como en la mayoría de módulos que se encuentran instalados en Scilab, la entrada y salida de mensajes y datos será a través de la terminal de caracteres gráfica que despliega Scilab para insertar comandos:

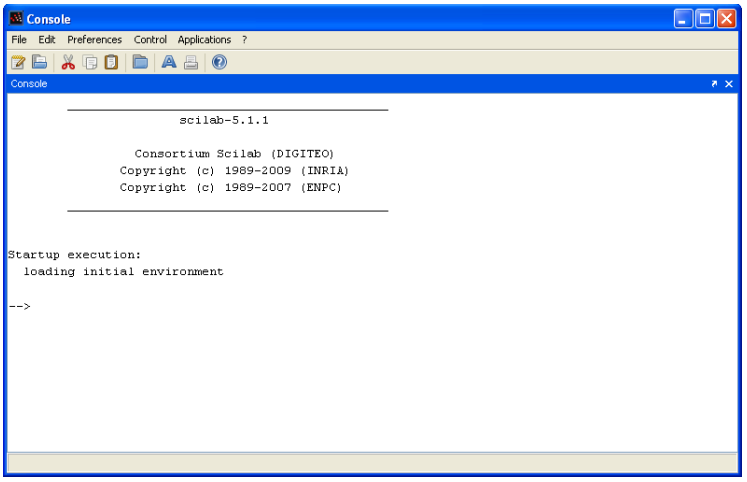

Figura 24. Interfaz gráfica de Scilab.

El usuario ejecuta las funciones del proyecto dentro en la terminal de Scilab y el terminal muestra el resultado de la evaluación de las funciones, junto con posibles mensajes de error.

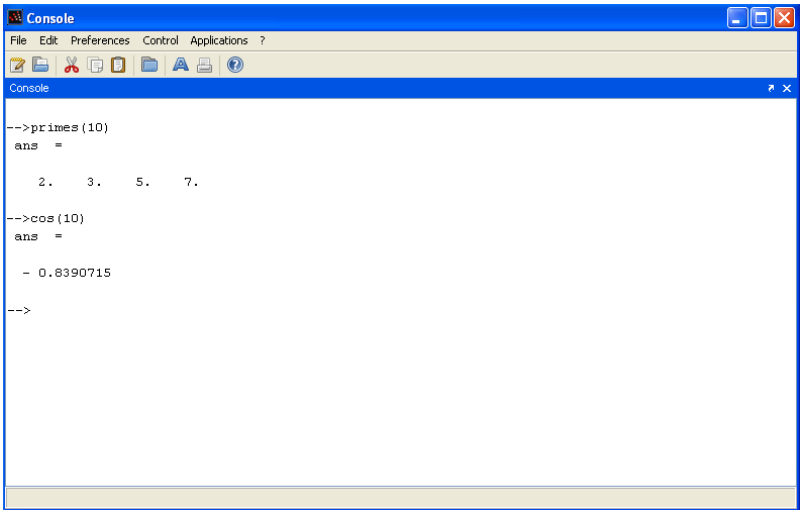

Figura 25. Interfaz gráfica de Scilab.

4.2.6 LIMITACIONES

4.2.6.1 Tamaño de p y n al usar aritmética de precisión fija

Cuando realizamos operaciones matemáticas en un computador, estamos sujetos a dos limitantes: El tamaño de las celdas de la RAM y el tamaño de los registros de memoria del procesador. En un sistema de 32 bits (4 bytes) podemos almacenar un número entero sin signo hasta un valor máximo de $4,294,967,295$, a partir de allí ocurre un desbordamiento y obtenemos resultados indefinidos. Para aplicaciones que requieren almacenar enteros con cifras del orden de 20 cifras o más, se utiliza librerías de aritmética de precisión arbitraria, que les permite sobrepasar el límite de los 4 bytes y almacenar enteros con un número grande de cifras en palabras de memoria de 5, 10, 15 bytes o mas, el único límite es la cantidad de RAM disponible.

Desafortunadamente Scilab no cuenta con librerías de precisión arbitraria, por lo que estamos limitados a valores hasta $4,294,967,295$. A medida que p y n crecen, crece también la magnitud de las variables enteras que se utilizan en los algoritmos, aumentando la probabilidad de que ocurra un desbordamiento inesperado en un punto del trabajo, produciendo resultados inesperados.

Para superar esta dificultad, se ha puesto un límite a los valores de entrada p y n, que son 101 y 32 respectivamente. De esta forma se evita un posible desbordamiento inicial.

4.2.6.2 Complejidad al portar la herramienta a Octave

En la evaluación del plan de proyecto se hizo la surgerencia de portar la herramienta desarrollada a Octave. Debido a las diferencias sobre cómo Scilab y Octave intercambian información entre su kernel matemático correspondiente y las herramientas externas, implementar el módulo en Octave implicaría crear nuevas interfaces para intercambiar datos entre las rutinas de campos finitos y Octave, lo cual significa que habría que reescribir casi la mitad del código del proyecto, por lo que se descarta portar la herramienta a Octave.

5. PRUEBAS DE LA HERRAMIENTA

Después de haber finalizado la etapa de desarrollo de la herramienta, se procede a probar si las funciones implementadas se comportan de acuerdo con los teoremas enunciados en el capítulo 3. Utilizando los mismos ejemplos mencionados en el marco teórico, se comienza a hacer pruebas con el programa.

5.1 CREACION DE CAMPOS PRIMOS

Se inicia creando el campo primo $GF(5) = \{0, 1, 2, 3, 4\}$ mediante la función **cf_crear_campo**.

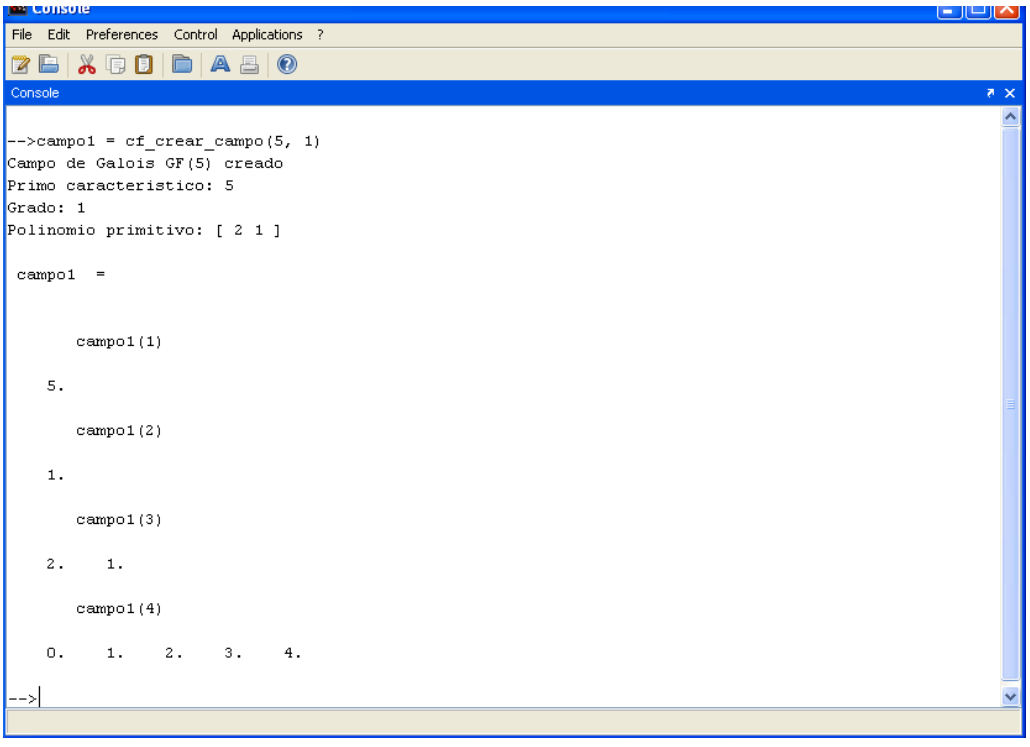

Figura 26. Creación del campo $GF(5)$.

La función imprime un párrafo inicial que muestra la información detallada del campo creado y devuelve una lista que contiene la estructura campo primo, que en este caso se almacena en la variable **campo1**. A partir de aquí el usuario puede consultar el contenido de la variable o la puede usar como entrada de otras funciones.

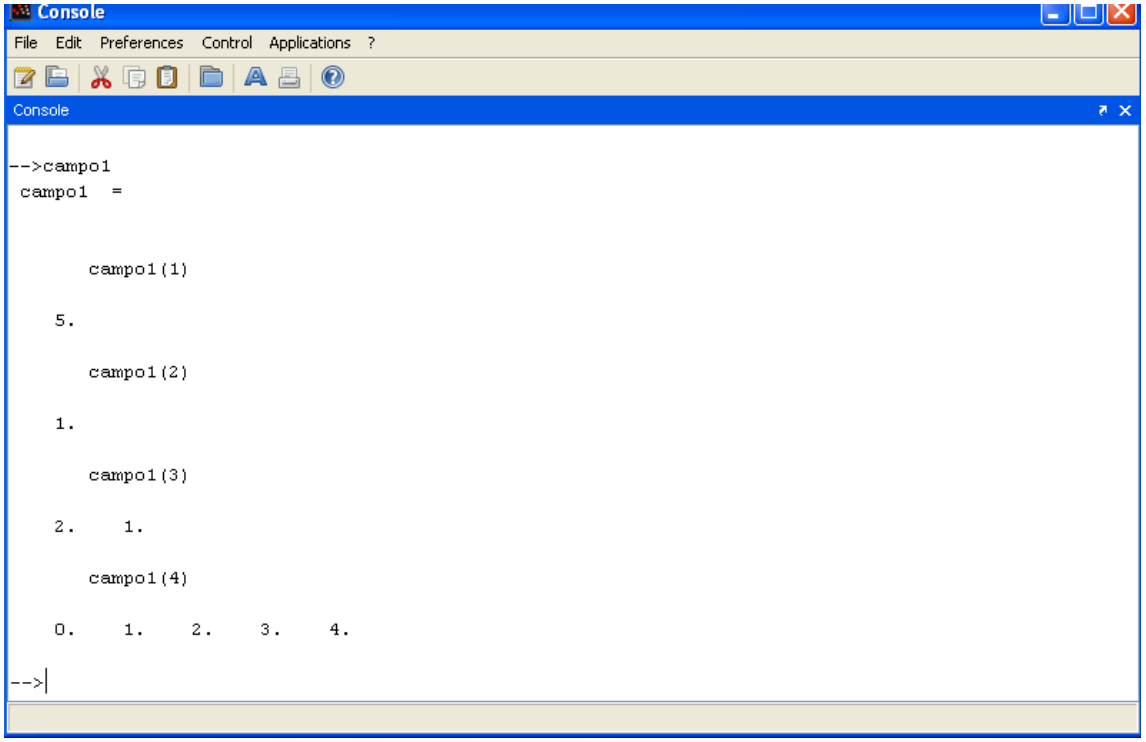

Figura 27. Consultando el contenido de la variable campo1.

Por ejemplo, podemos extraer información de **campo1** mediante las funciones **cf_primo_caracteristico**, **cf_orden**, **cf_grado** y **cf_elementos_campos**.

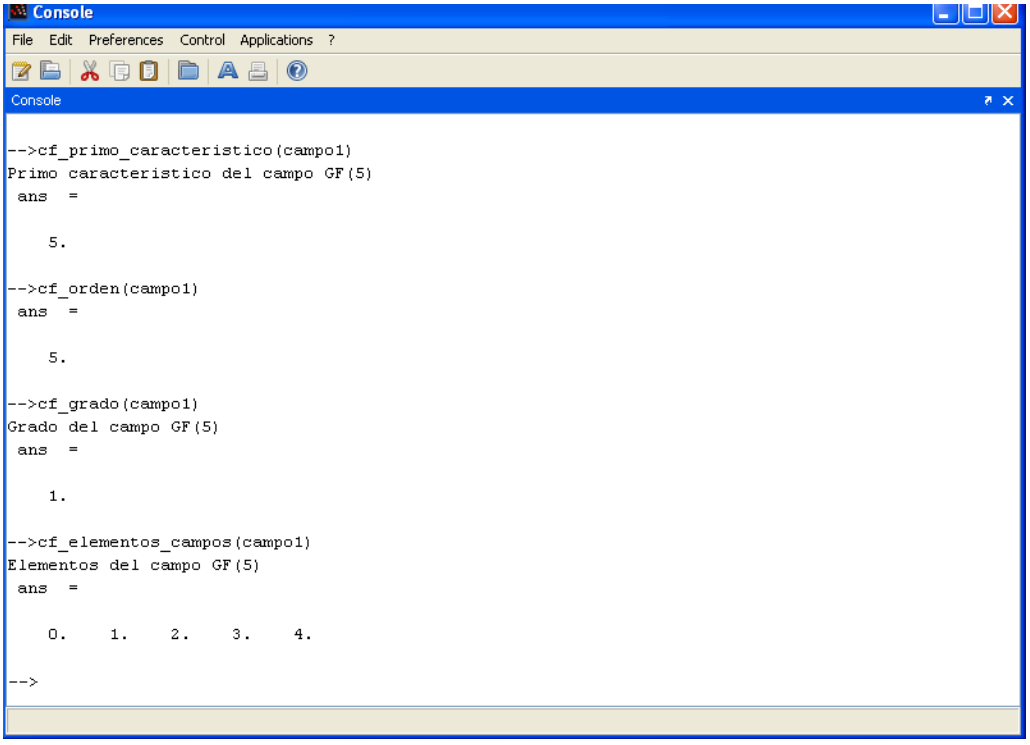

Figura 28. Extrayendo información de la variable campo1.

5.2 OPERACIONES ARITMETICAS ENTRE ELEMENTOS DE CAMPOS PRIMOS

Después de haber definido el campo primo en la sección anterior, usamos la variable **campo1** como argumento de **cf_sumar_elementos** para hacer la suma entre los elementos 3 y 2 del campo $GF(5)$.

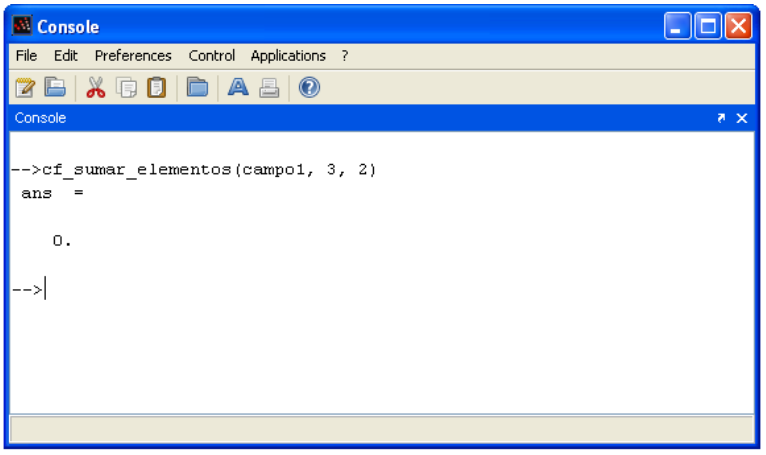

Figura 29. Operación $3+2$ en $GF(5)$.

Lo cual es el resultado esperado, ya que $3+2 = 5\,(mod\,5) = 0$. Del mismo modo llamamos a las funciones **cf_restar_elementos**, **cf_producto_elementos**, **cf_dividir_elementos** y **cf_potencia_elementos** usando los mismos valores.

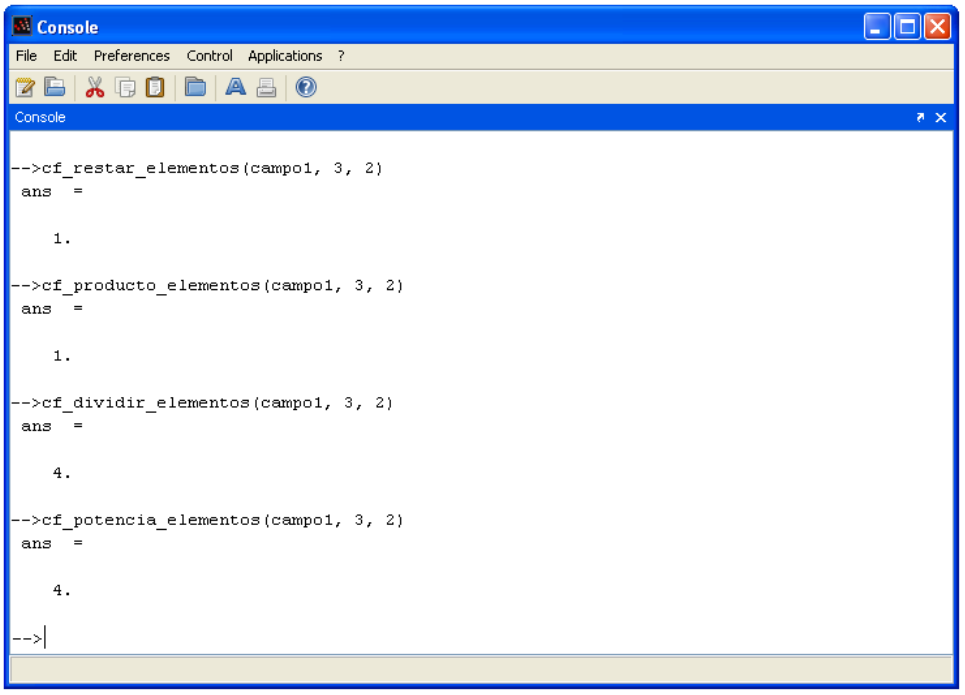

Figura 30. Operación resta, producto, división y potencia en $GF(5)$.

Se puede ver que son los valores esperados, ya que:

$3 - 2 = 1\ (mod\ 5) = 1$
$3 * 2 = 6\ (mod\ 5) = 1$
$3/2 = 4\ \Rightarrow 2 * 4 = 8\ (mod\ 5) = 3$
$3^2 = 9\ (mod\ 5) = 4$

Se pueden construir tablas de operaciones para los campos primos donde se vean todas las operaciones posibles entre los elementos del campo y sus resultados, mediante las funciones **cf_tabla_adicion**, **cf_tabla_sustraccion**, **cf_tabla_producto** y **cf_tabla_division**.

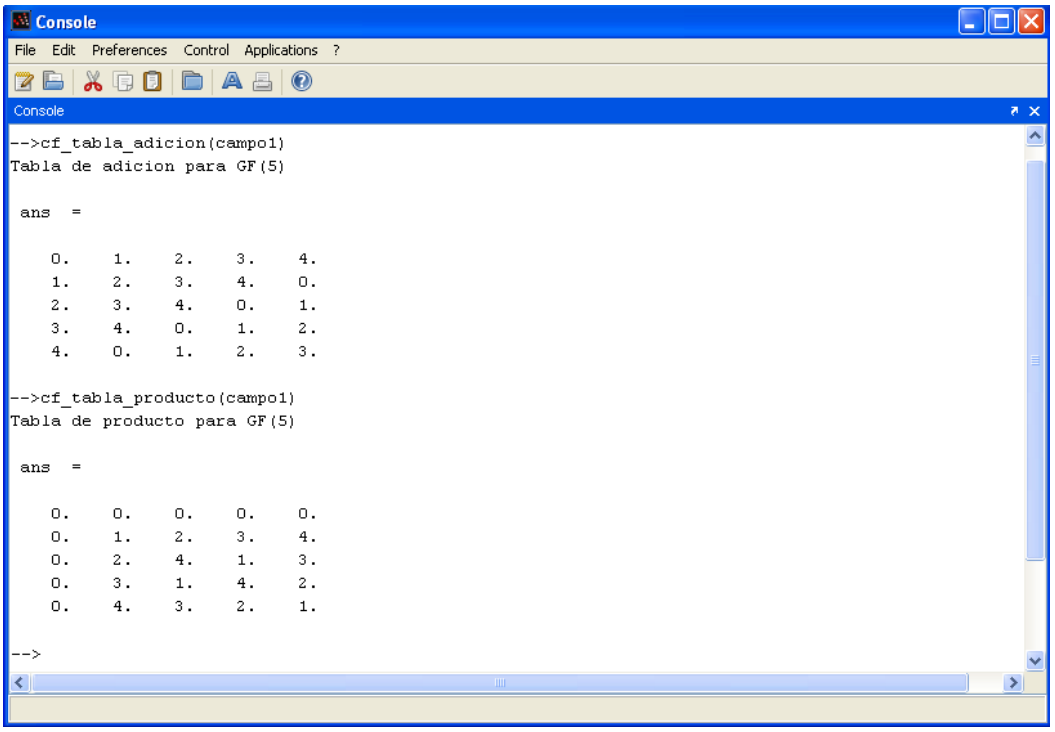

Figura 31. Tabla de adición y producto para $GF(5)$.

5.3 HALLAR ELEMENTOS PRIMITIVOS Y POLINOMIOS MINIMOS SOBRE CAMPOS PRIMOS

Trabajando de nuevo con el campo $GF(5)$, procedemos a llamar a la funcion **cf_elementos_primitivos**, pasando la variable **campo1** como argumento.

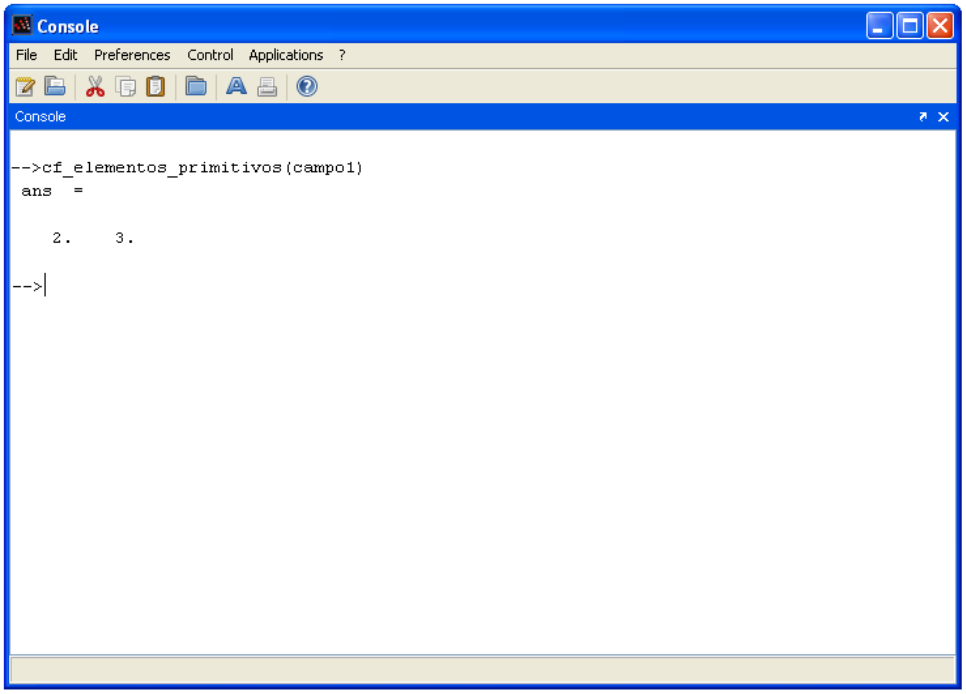

Figura 32. Salida de cf_elementos_primitivos.

Verificando que 2 y 3 son elementos primitivos del campo $GF(5)$:

$2^1 = 2 \ (mod\ 5) = 2$
$2^2 = 4 \ (mod\ 5) = 4$
$2^3 = 8 \ (mod\ 5) = 3$
$2^4 = 16 \ (mod\ 5) = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(5)$, 2 es elemento primitivo.

$3^1 = 3\ (mod\ 5) = 3$
$3^2 = 9\ (mod\ 5) = 4$
$3^3 = 27\ (mod\ 5) = 2$
$3^4 = 81\ (mod\ 5) = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(5)$, 3 es elemento primitivo.

Se puede saber de antemano cuantos elementos primitivos hay en un campo dado, mediante la función **cf_funcion_phi_euler** pasando como argumento el orden del campo -1.

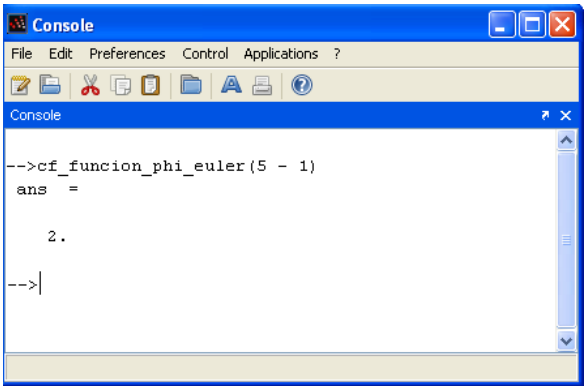

Figura 33. Salida de cf_funcion_phi_euler.

Para hallar el polinomio mínimo de un elemento $x \in GF(5)$ llamamos a la función **cf_polinomio_minimo**, pasando como argumento la variable **campo1**, y el elemento al que deseamos hallar el polinomio mínimo, por ejemplo 2.

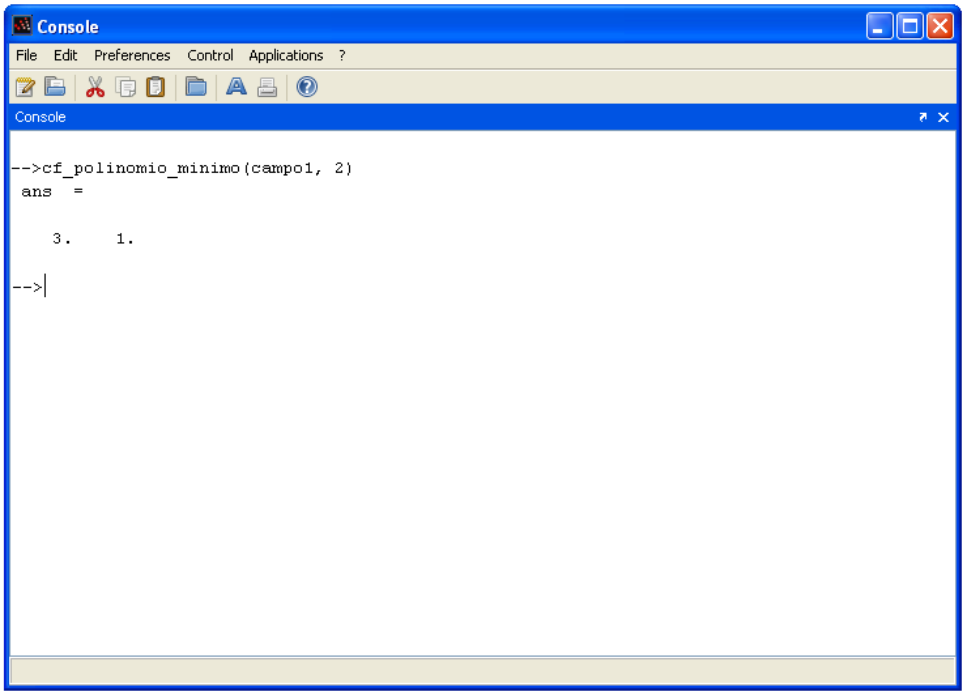

Figura 34. Salida de cf_polinomio_minimo.

La salida de la función es el polinomio $f(x) = x+3$, lo cual concuerda con el concepto de polinomio mínimo, ya que $f(2) = 2 + 3 = 5\ (mod\ 5) = 0$.

5.4 ARITMETICA DE POLINOMIOS SOBRE $GF(p)$

Tomando como ejemplo los polinomios $f(x) = x^2+1$, $g(x) = x^4+x+1$, $r(x) = x^2+x+2$ y $h(x) = x+2$, representados en la herramienta como los arreglos [1 0 1], [1 1 0 0 1], [2 1 1] y [2 1]; se va a realizar las operaciones suma, resta, producto, división y potencia bajo el campo $GF(3)$ mediante las funciones **cf_sumar_polinomios**, **cf_restar_polinomios**, **cf_producto_polinomios**, **cf_dividir_polinomios** y **cf_potencia_polinomios**. También usamos la función **cf_formato_polinomial** para convertir los arreglos de salida de las funciones en polinomios.

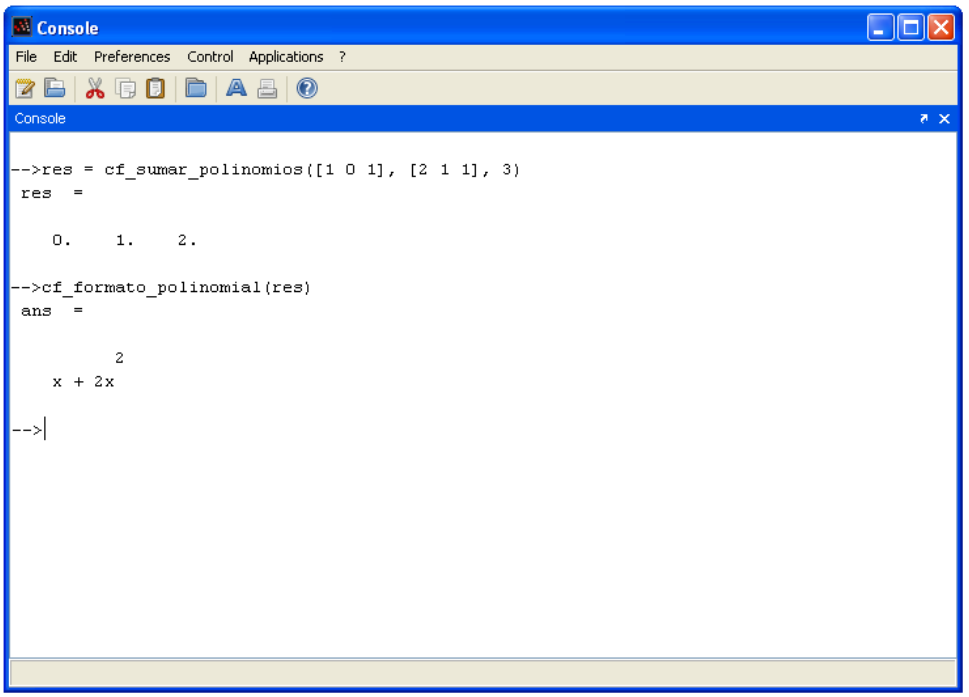

Figura 35. Salida de cf_sumar_polinomios.

Verificando la suma de los polinomios $f(x)$ y $r(x)$:

$$(x^2 + 1) + (x^2 + x + 2) = 2x^2 + x + 3 \ (mod\ 3) = 2x^2 + x$$

$$3 \equiv 0 \ (mod\ 3)$$

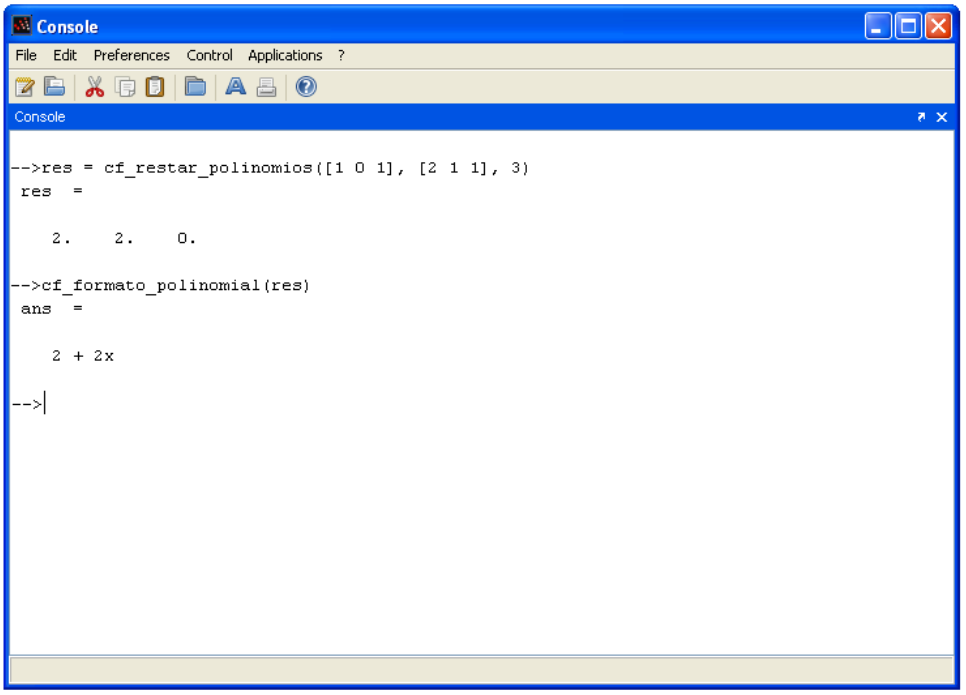

Figura 36. Salida de cf_restar_polinomios.

Verificando la resta entre los polinomios $f(x)$ y $r(x)$:

$(x^2 + 1) - (x^2 + x + 2) = 0x^2 - x - 1 = -x - 1 \ (mod\ 3) = 2x + 2$

$-1 \equiv 2 \ (mod\ 3)$

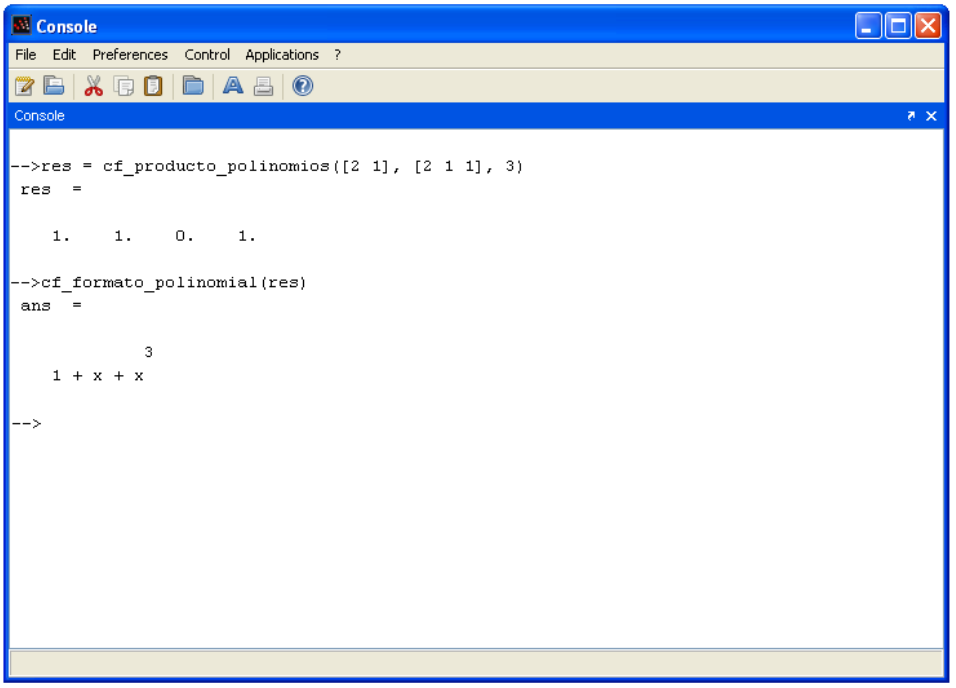

Figura 37. Salida de cf_producto_polinomios.

Verificando el producto entre los polinomios $h(x)$ y $f(x)$:

$(x+2) * (x^2+x+1) = x^3+3x^2+4x+4$

$x^3+3x^2+4x+4\ (mod\ 3) = x^3+x+1$

$3 \equiv 0\ (mod\ 3)$

$4 \equiv 1\ (mod\ 3)$

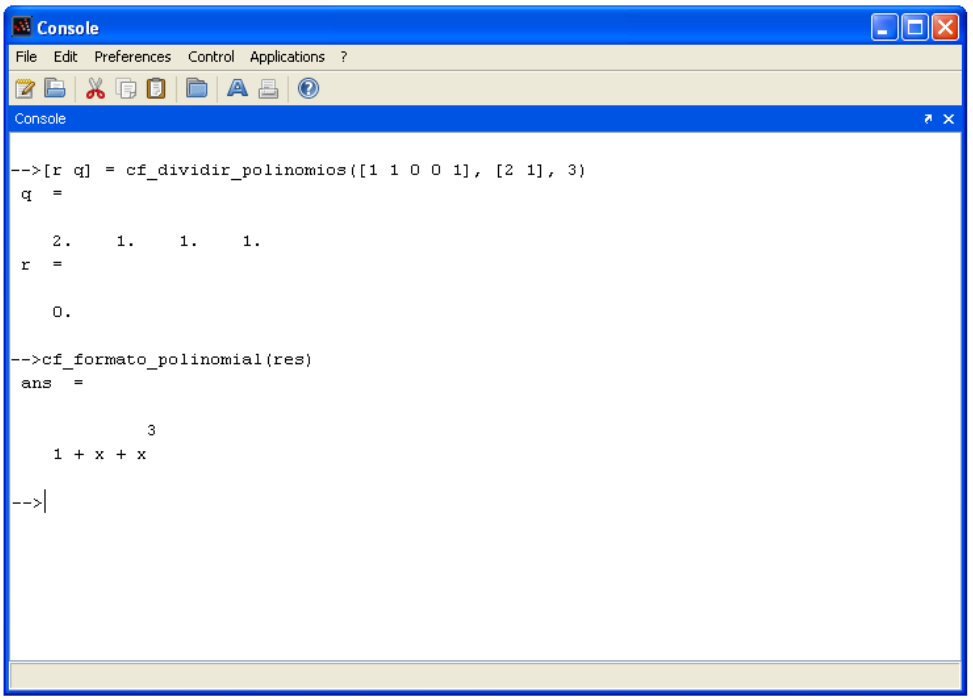

Figura 38. Salida de cf_dividir_polinomios.

Verificando la división de $g(x)$ entre $h(x)$ mediante el algoritmo de división polinomial larga:

$$\frac{g(x)}{h(x)} = q(x)h(x) + r(x)$$

$$\frac{g(x)}{h(x)} = (2x^3 + x^2 + x + 1)(x + 2) + 0$$

$$q(x) = 2x^3 + x^2 + x + 1 \; ; \; r(x) = 0$$

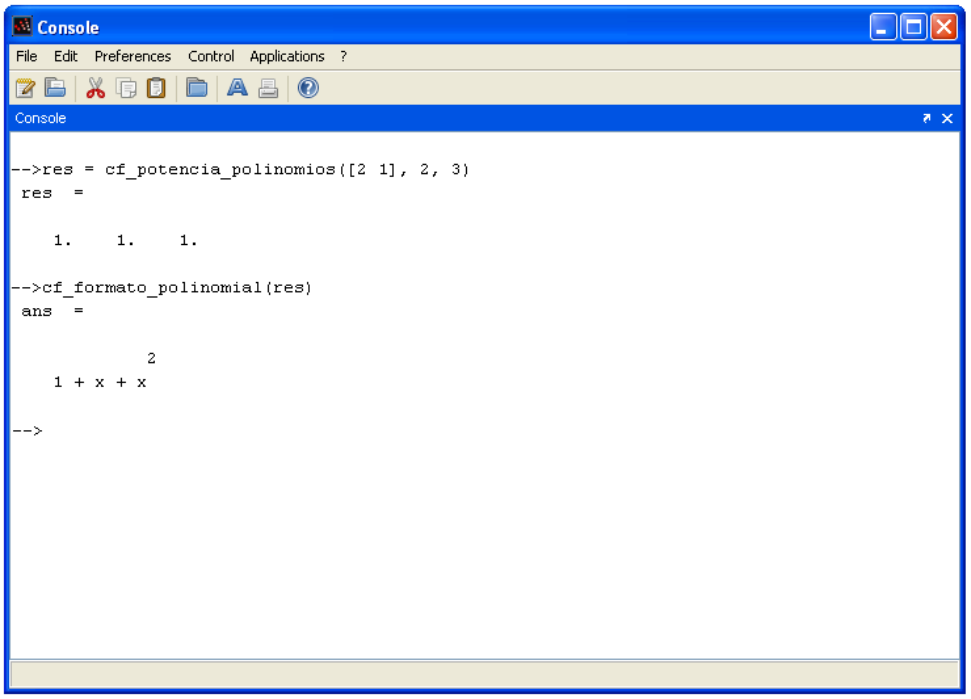

Figura 39. Salida de cf_potencia_polinomios.

Verificando el resultado de $(x+2)^2$:

$(x+2)^2 = (x+2)(x+2) = x^2 + 2x + 2x + 4 = x^2 + 4x + 4$

$x^2 + 4x + 4 \ (mod\ 3) = x^2 + x + 1$

$4 \equiv 1 \ (mod\ 3)$

5.5 CREACION DE CAMPOS DE EXTENSION

Usando la misma función utilizada en la sección 5.1, se va a crear el campo $GF(3^2)$. Para un campo de extensión, la función **cf_crear_campo** requiere de tres argumentos: p, n, y polinomio primitivo. Sabemos que $p = 3$ y $n = 2$, pero no conocemos ningún polinomio primitivo para esta combinación de p y n. Utilizando la función **cf_devolver_pol_prim** y pasando como argumento los dos valores anteriores, se obtiene una lista de polinomios primitivos:

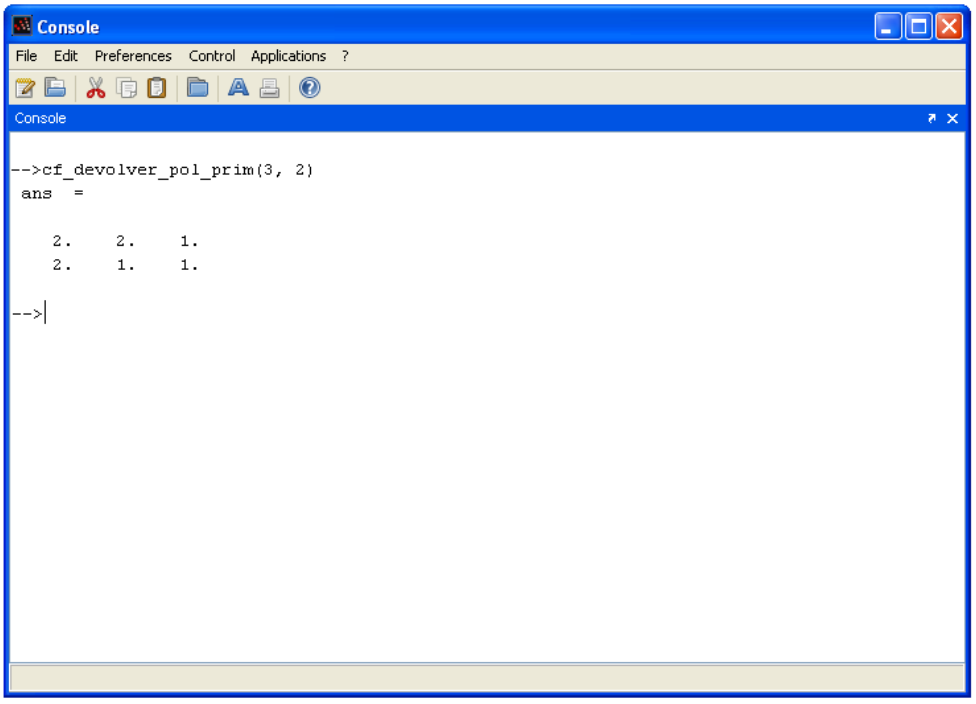

Figura 40. Salida de cf_devolver_pol_prim.

Se puede ver que los polinomios primitivos disponibles para la combinación $p = 3$ y $n = 2$ son $x^2 + x + 2$ y $x^2 + 2x + 2$. El usuario puede comprobar que los polinomios son realmente primitivos mediante la función **cf_es_pol_primitivo**.

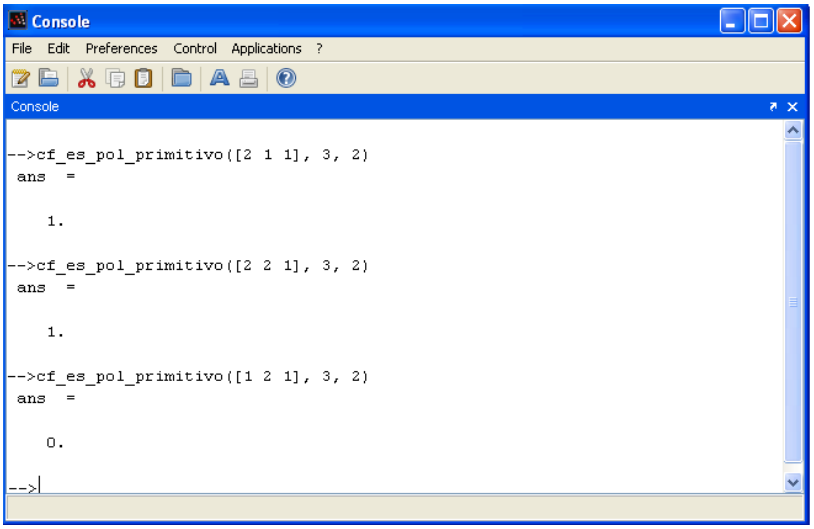

Figura 41. Salida de cf_es_pol_primitivo.

Los polinomios $x^2 + 2x + 2$ y $x^2 + x + 2$ son primitivos, mientras que $x^2 + 2x + 1$ no es primitivo.

Utilizando el polinomio $x^2 + x + 2$ como tercer argumento y llamando a la función de creación de campos:

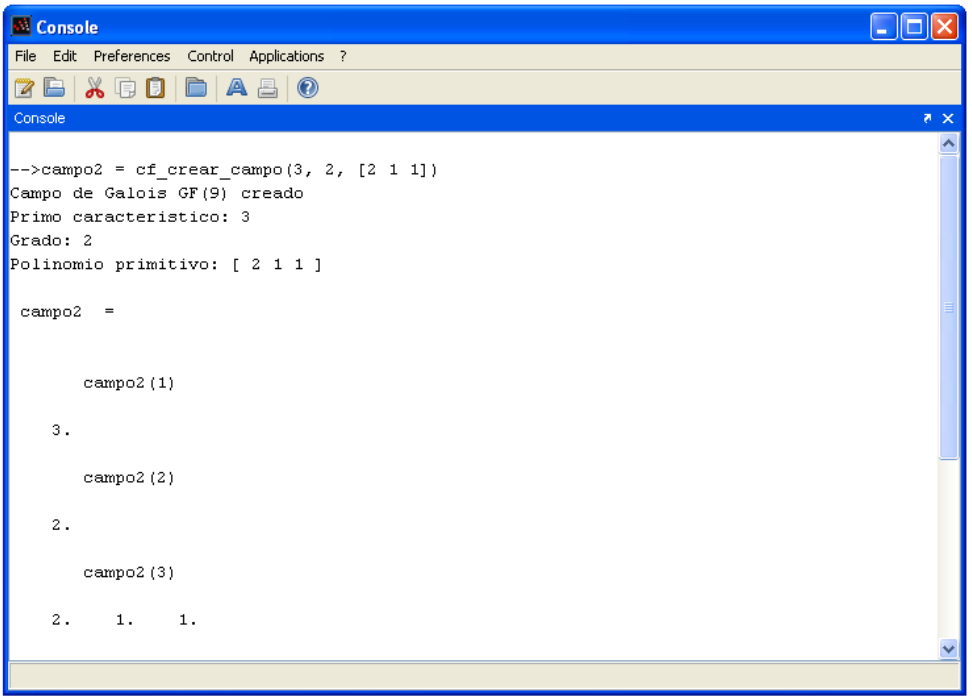

Figura 42. Salida de cf_crear_campo.

La función devuelve una estructura campo de extensión (estudiada en la sección 4.2.1), que se almacena en la variable **campo2**.

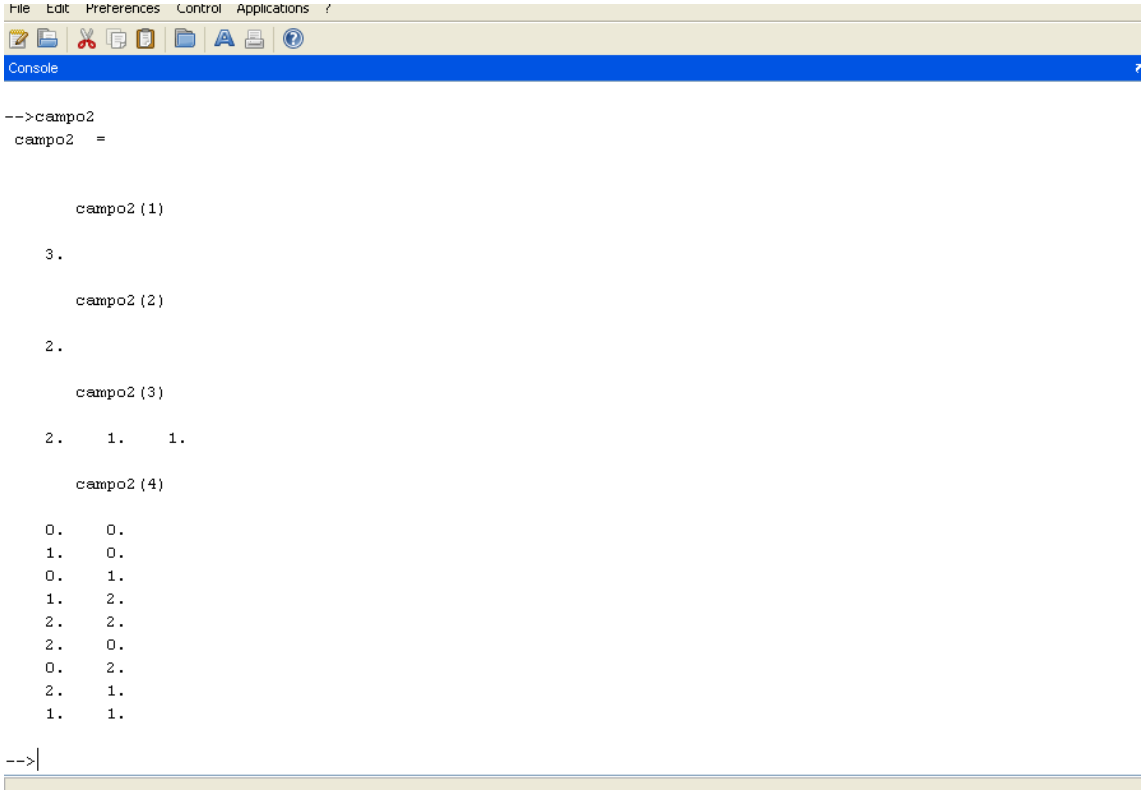

Figura 43. Contenido de la variable campo2.

También se puede extraer de **campo2** información del primo característico, grado, orden o el listado de los elementos:

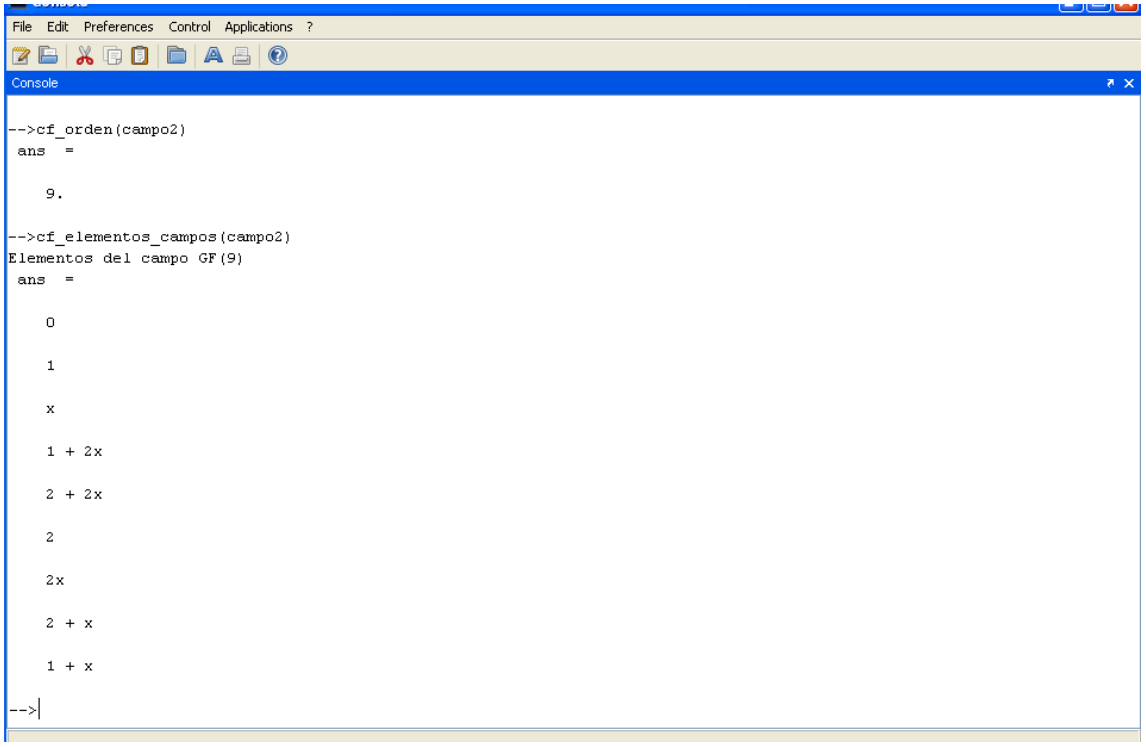

Figura 44. Extrayendo información de campo2.

5.6 OPERACIONES ARITMETICAS ENTRE ELEMENTOS DE CAMPOS DE EXTENSION

Se usan las mismas funciones que se usaron en la sección 5.2 para el campo primo $GF(5)$, en este caso se van a usar para un campo de extensión. Trabajando con los elementos $x+1$ y $2x$ que pertenecen al campo $GF(3^2)$, se ejecutan las operaciones aritméticas:

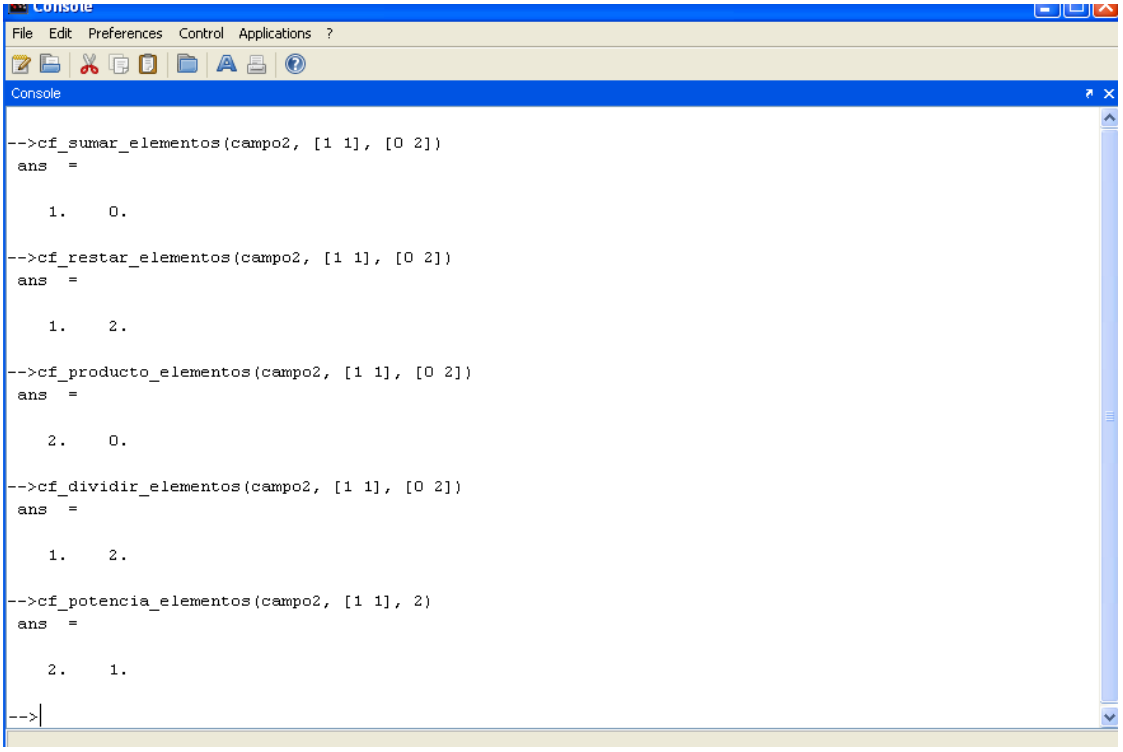

Figura 45. Operaciones aritméticas en $GF(3^2)$.

Comprobando la operación suma:

$(x+1)+2x = 3x+1 \ (mod\ 3) = 1$

$3 \equiv 0 \ (mod\ 3)$

Comprobando la operación resta:

$(x+1)-2x = -x+1 \ (mod\ 3) = 2x+1$

$-1 \equiv 2 \ (mod\ 3)$

Comprobando la operación producto:

$(x+1)*2x = 2x^2+2x \ (mod\ 3) = 2x^2+2x$

$2x^2 + 2x \ (mod\ x^2 + x + 2) = 2$

Comprobando la operación división:

$(x + 1)/2x = x + 2 \ \Rightarrow \ (x + 2) * 2x = (x + 1)$

Comprobando la operación potencia:

$(x + 1)^2 = (x + 1)(x + 1) = x^2 + x + x + 1 = x^2 + 2x + 1$

$x^2 + 2x + 1 \ (mod\ 3) = x^2 + 2x + 1$

$x^2 + 2x + 1 \ (mod\ x^2 + x + 2) = x + 2$

5.7 HALLAR ELEMENTOS PRIMITIVOS SOBRE CAMPOS DE EXTENSION

Llamando a la función **cf_elementos_primitivos** y pasando como argumento la variable **campo2**, obtenemos los elementos primitivos de $GF(3^2)$:

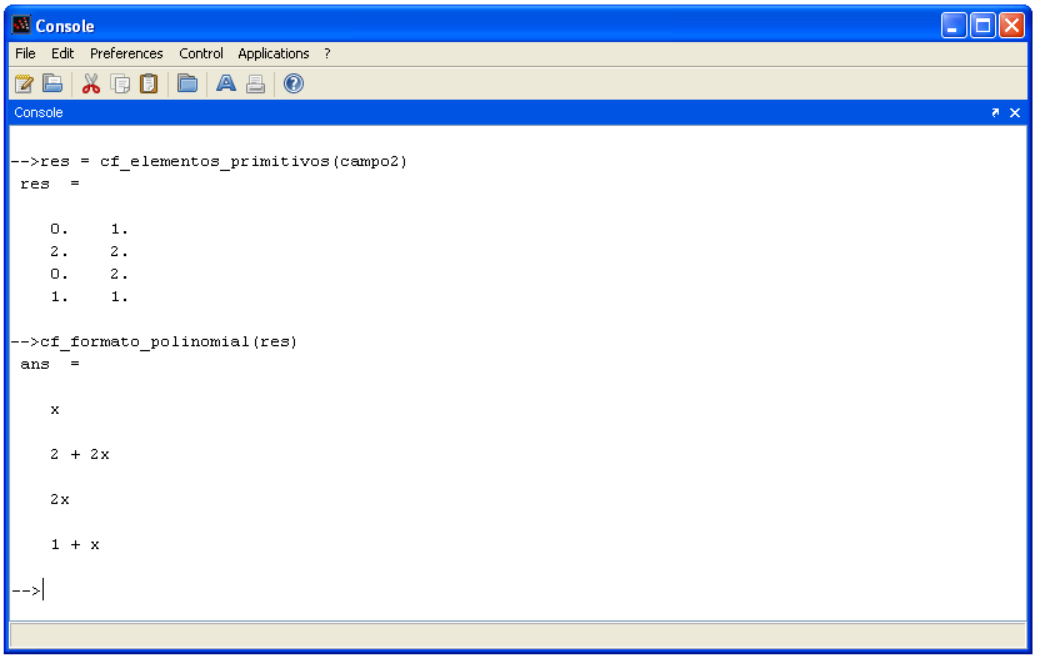

Figura 46. Elementos primitivos del campo $GF(3^2)$.

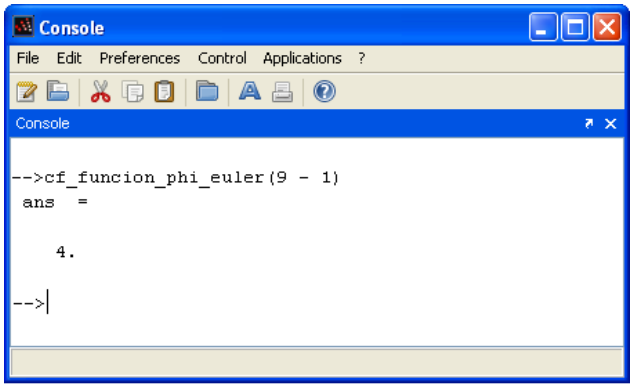

Figura 47. Número de elementos primitivos en $GF(3^2)$.

Comprobando que x es elemento primitivo:

$x^1 = x$

$x^2 = 2x + 1$

$x^3 = 2x + 2$
$x^4 = 2$
$x^5 = 2x$
$x^6 = x + 2$
$x^7 = x + 1$
$x^8 = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(3^2)$, x es elemento primitivo.

Comprobando que $2x$ es elemento primitivo:

$(2x)^1 = 2x$
$(2x)^2 = 2x + 1$
$(2x)^3 = x + 1$
$(2x)^4 = 2$
$(2x)^5 = x$
$(2x)^6 = x + 2$
$(2x)^7 = 2x + 2$
$(2x)^8 = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(3^2)$, $2x$ es elemento primitivo.

Comprobando que $2x + 2$ es elemento primitivo:

$(2x + 2)^1 = 2x + 2$
$(2x + 2)^2 = x + 2$
$(2x + 2)^3 = x$
$(2x + 2)^4 = 2$
$(2x + 2)^5 = x + 1$
$(2x + 2)^6 = 2x + 1$
$(2x + 2)^7 = 2x$
$(2x + 2)^8 = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(3^2)$, $2x + 2$ es elemento primitivo.

Finalmente, comprobando que $x + 1$ es elemento primitivo:

$(x + 1)^1 = x + 1$
$(x + 1)^2 = x + 2$
$(x + 1)^3 = 2x$
$(x + 1)^4 = 2$
$(x + 1)^5 = 2x + 2$
$(x + 1)^6 = 2x + 1$
$(x + 1)^7 = x$
$(x + 1)^8 = 1$

Se obtienen todos los elementos diferentes de 0 de $GF(3^2)$, $x + 1$ es elemento primitivo.

5.8 REPRESENTAR LOS ELEMENTOS DE UN CAMPO EN FORMATO POLINOMIAL Y EXPONENCIAL

Se ha visto anteriormente en los ejemplos de este capítulo que la función **cf_formato_polinomial** convierte los arreglos de coeficientes en polinomios. Por ejemplo, si extraemos los elementos de **campo2** en una variable y la pasamos como argumento en la función, obtenemos la totalidad de los elementos del campo expresados en formato polinomial:

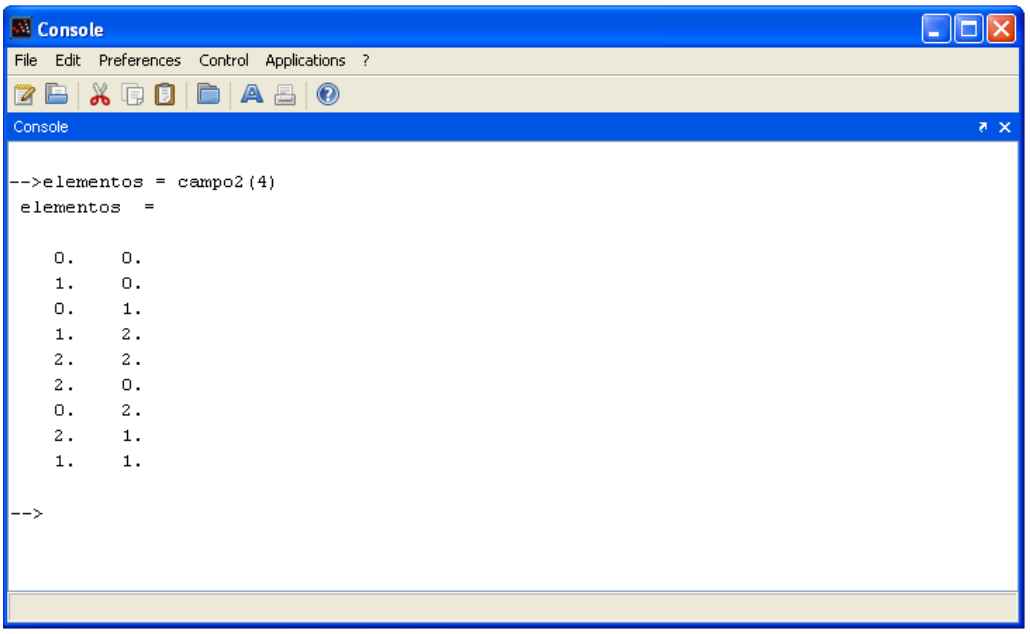

Figura 48. Extraer los elementos de $GF(3^2)$.

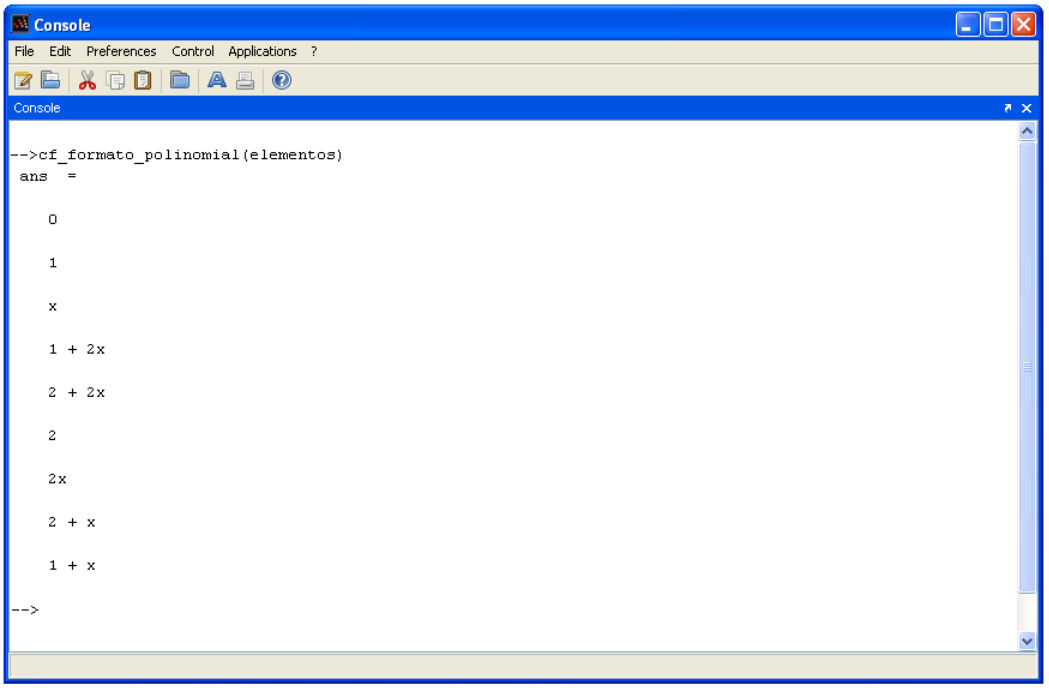

Figura 49. Elementos de $GF(3^2)$ en formato polinomial.

También podemos expresar los elementos de $GF(3^2)$ en formato exponencial, es decir de la forma x^{-1}, x^0, x^1, x^2, x^3, ..., x^7, donde cada monomio corresponde a un elemento del campo. Llamando a la función **cf_formato_exponencial**, y pasando como argumento la variable **campo2** que representa el campo, la **variable** elementos que almacena una copia de los elementos del campo, y el carácter 'c' que indica a la función que devuelva los monomios como cadenas de carácteres, obtenemos el siguiente resultado:

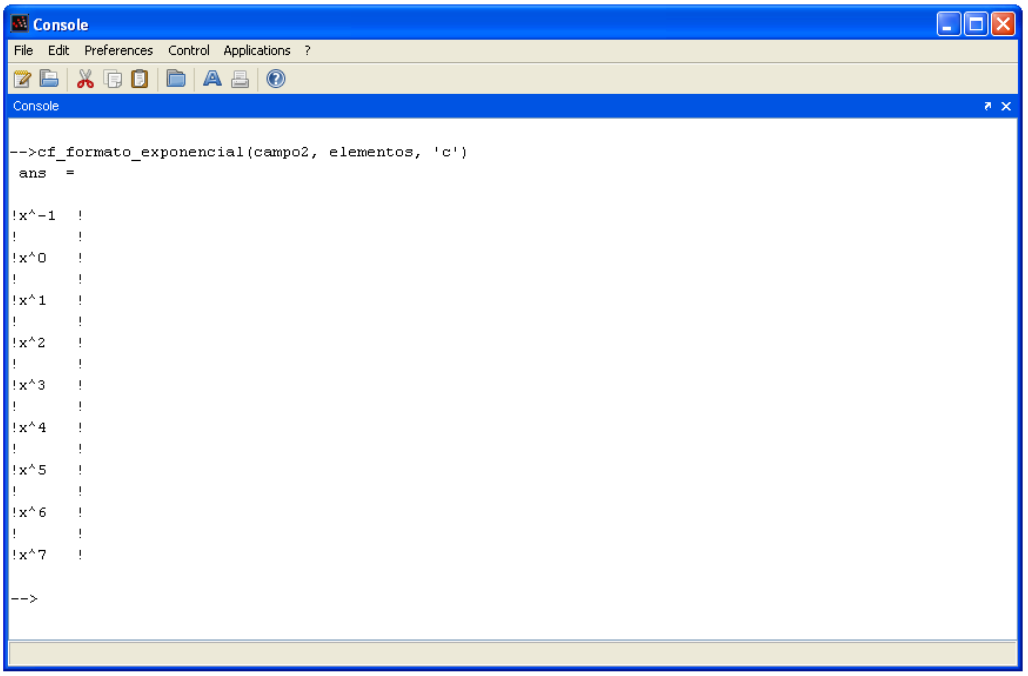

Figura 50. Elementos de $GF(3^2)$ en formato exponencial.

Se puede ver que los elementos de **campo2** estan ordenados exponencialmente, de acuerdo con la siguiente tabla:

Exponencial	Polinomial
x^{-1}	0
x^0	1
x	x
x^2	$2x+1$
x^3	$2x+2$
x^4	2
x^5	$2x$
x^6	$x+2$
x^7	$x+1$

Tabla 8. Elementos de $GF(3^2)$ en formato exponencial y polinomial.

5.9 USO DE LAS FUNCIONES IMPLEMENTADAS PARA UNA PRUEBA DE CONTROL DE ERRORES

Con el ánimo de mostrar una aplicación de los Campos de Galois, se implementó una pequeña rutina de control de errores mediante Códigos de Hamming, almacenada en el archivo EjemploControlErrores.sce.

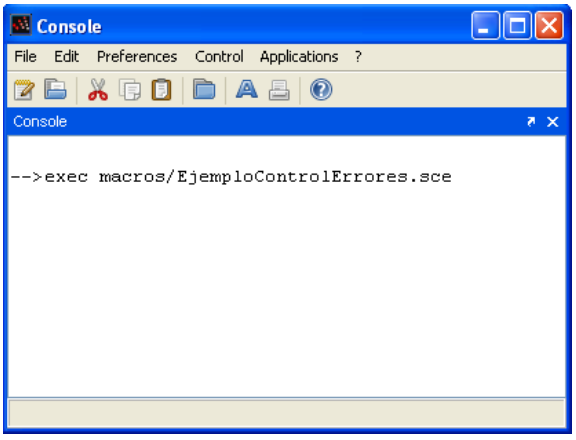

Figura 51. Ejecutar la rutina de prueba de control de errores.

La rutina crea la matriz generadora y la matriz chequeo de paridad (enunciadas en la sección 3.3.8.2) y a continuación crea un campo $GF(2^4)$, de donde extraemos el elemento de 4 bits [1 0 0 1], posteriormente el programa codifica el elemento multiplicandolo por G, obteniendo una palabra de 7 bits:

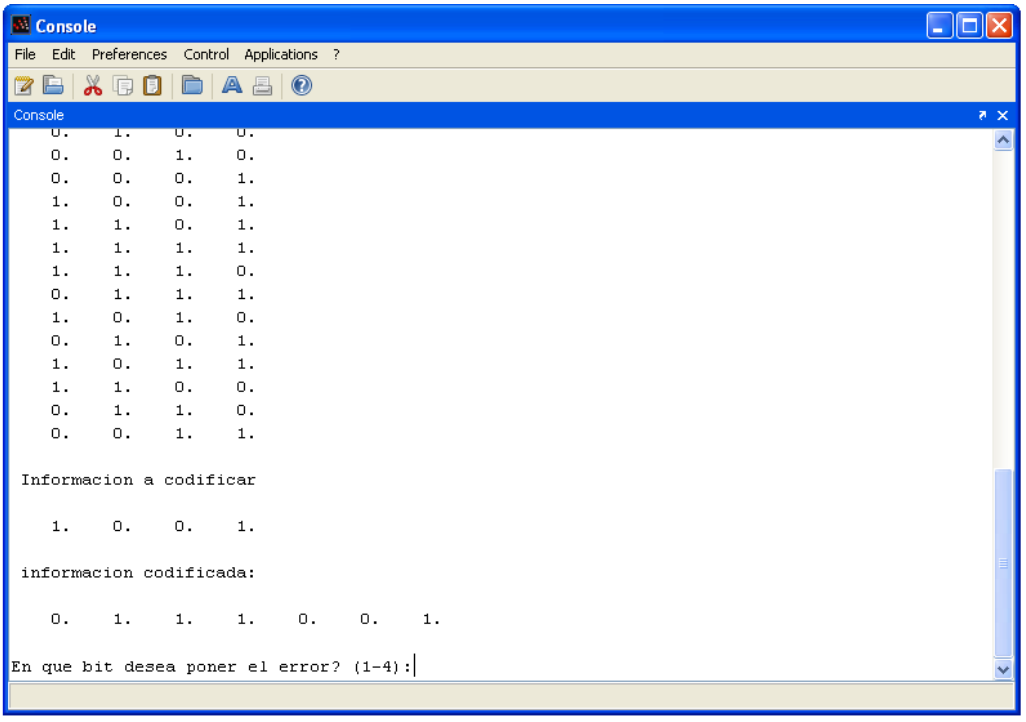

Figura 52. Ejecución de EjemploControlErrores.sce.

El programa le pide al usuario elegir en que bit desea poner el error (bit 1, 2, 3 o 4). A modo de ejemplo, seleccionamos el bit 2 como bit de error.

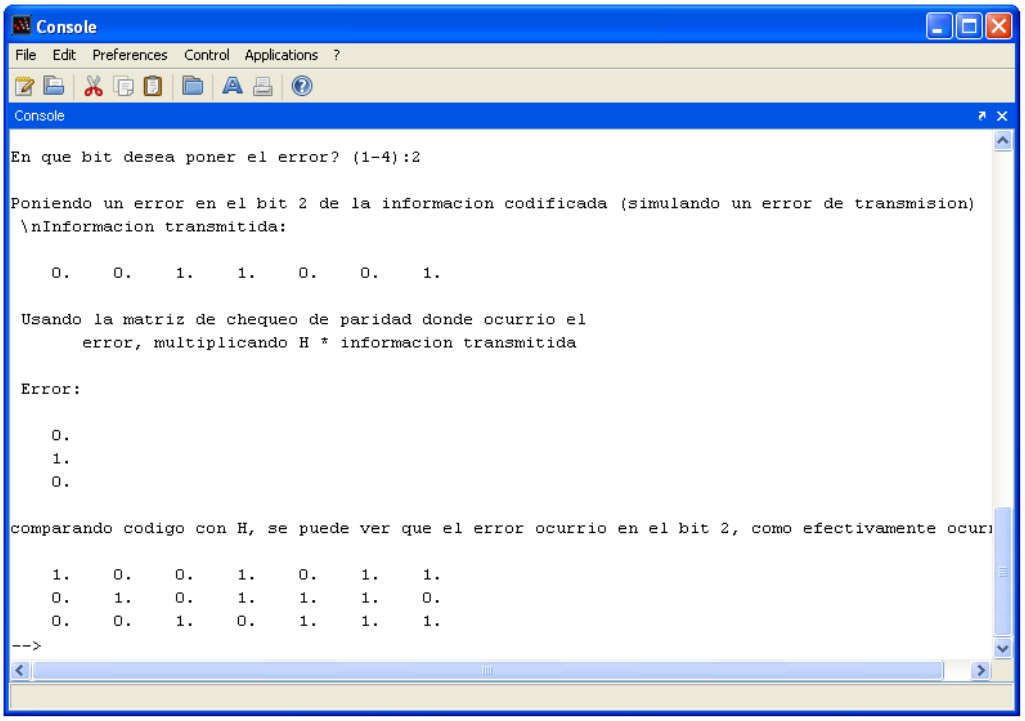

Figura 53. Salida de EjemploControlErrores.sce.

El programa cambia el bit 2 de 1 a 0, simulando el error de transmisión. De igual forma, simulando la recepción (asumiendo en el programa que el receptor no conoce dónde ocurrió el error), el programa multiplica la información recibida por la matriz H, obteniendo un vector columna. Para determinar en qué bit ocurrió el error, se compara el vector columna con los 4 primeros vectores columna de H, en la posición donde hay una coincidencia entre los dos vectores columnas, es el bit donde ocurrió el error, en este caso es 2.

6. CONCLUSIONES

- Se ha cumplido con el objetivo principal propuesto desde el plan de proyecto, se implementó una herramienta básica en Scilab que permite crear Campos Finitos o Campos de Galois, además de realizar operaciones aritméticas sobre los elementos de los campos, hallar elementos primitivos, hallar polinomios primitivos, representar los elementos de un campo en diferentes formas y realizar operaciones aritméticas entre polinomios sobre campos primos.

- Se mostró una aplicación real de los campos finitos en una rutina de control de errores, simulando los procesos de transmisión y recepción de datos a través de una red, y la posterior verificación de bits erróneos mediante operaciones entre elementos de campos finitos y matrices auxiliares.

- Se crearon rutinas que podrán ser útiles para implementar algoritmos criptográficos en Scilab.

- Con este proyecto se logró una contribución por parte de la UIS y de la Escuela de Ingeniería de Sistemas en un paquete matemático de Software Libre de ámplio uso en entornos académicos e industriales de la Unión Europea y Estados Unidos.

7. RECOMENDACIONES

- Implementar otros algoritmos para determinar si un polinomio es primitivo y para obtener la lista de polinomios primitivos dado p y n de tal forma que el tiempo de cómputo sea mucho menor al empleado en los algoritmos manejados en ete proyecto.

- Implementar el algoritmo extendido de euclides para dividir dos elementos de un campo en lugar de usar el algoritmo de fuerza bruta.

- Portar la herramienta en un paquete matemático que maneje librerías de precisión aritmética arbitraria, de tal forma que se puedan trabajar con valores de p y n de gran magnitud.

- Extender la funcionalidad de la herramienta al crear rutinas adicionales que permitan hallar polinomios mínimos para un campo de extensión, hallar los logaritmos discretos de los elementos de un campo y resolver sistemas de ecuaciones donde las incógnitas sean elementos de campos.

- Surgerir a la UIS de liberar esta herramienta bajo una licencia abierta, libre de las restricciones de propiedad intelectual impuestos a los proyectos de grado, de tal forma que la comunidad mundial de software libre pueda contribuir al mejoramiento de este módulo.

8. BIBLIOGRAFIA

LIDL, Rudolf; Niederreiter, Harald (1997), Finite Fields (2nd ed.), Cambridge University Press, ISBN 0-521-39231-4.

SETHURAMAN, B. A. (1996). Rings, Fields, Vector Spaces, and Group Theory: An Introduction to Abstract Algebra via Geometric Constructibility. Springer. ISBN 0-387-94848-1.

R.B.J.T. Allenby (1991). Rings, Fields and Groups. Butterworth-Heinemann. ISBN 0-340-54440-6.

MACLANE, Saunders; Birkhoff, Garrett (1999), Algebra (2nd ed.), AMS Chelsea, ISBN 978-0-8218-1646-2.

HERSTEIN, Israel Nathan (1996), Abstract algebra (3rd ed.), Upper Saddle River, NJ: Prentice Hall Inc., MR1375019, ISBN 978-0-13-374562-7.

MULLEN Gary; Mummert Carl, Finite Fields and Applications, American Mathematical Society, 2007. Volumen 41, ISBN 978-0-8218-4418-2.

PRESSMAN, Roger, Software Engineering, A Practitioners Approach. McGraw-Hill Science, 2004. ISBN 978-0073019338.

GOTTFRIED, Byron, Programación en C, McGraw-Hill/Interamericana de España, 1997. ISBN 84-841-1068-4.

DAEMEN Joan; Vincent Rijmen, The Design of Rijndael: AES - The Advanced Encryption Standard, Springer-Verlag, 2002. ISBN 3-540-42580-2.

O'CONNOR, S. E. Computing Primitive Polynomials.

http://seanerikoconnor.freeservers.com/Mathematics/AbstractAlgebra/

WEISSTEIN, Eric W. "Primitive Polynomial."From MathWorld, A Wolfram Web Resource.
http://mathworld.wolfram.com/PrimitivePolynomial.html

WEISSTEIN, Eric W. "Finite Field."From MathWorld, A Wolfram Web Resource.
http://mathworld.wolfram.com/FiniteField.html

BARILE, Margherita; Rowland, Todd; and Weisstein, Eric W. Algebraic Number Minimal Polynomial. From MathWorld, A Wolfram Web Resource.
http://mathworld.wolfram.com/AlgebraicNumberMinimalPolynomial.html