Universidad de Valladolid

Escuela de Ingeniería Informática

Grado en Ingeniería Informática de Servicios y Aplicaciones

Semigrupos numéricos en GAP:

Aplicaciones a códigos y criptografía

Autor: Jorge Angulo Rodriguez

Director: Jose Ignacio Farrán

Índice general

Objetivos

En el trabajo de fin de grado de matemáticas he trabajado con semigrupos numéricos, ideales de semigrupos numéricos y múltiples conceptos relacionados con estos objetos. En el presente trabajo, he colaborado con Pedro A. García Sánchez, de la universidad de Granada, que gestiona la librería "NumericalSpgs" [5], una librería que implementa funcionalidad relacionada con los semigrupos numéricos en el lenguaje GAP. Así, voy a completar y extender la funcionalidad de la librería en los siguientes aspectos:

- Implementar la descomposición en componentes $\mathbb{Z}$-irreducibles para ideales relativos de semigrupos numéricos, y descomposición en componentes irreducibles para ideales propios de semigrupos numéricos.
- Implementar a la operación $\oplus$, y caracterizar el semigrupo a partir del conjunto de ternas $(i, j, i \oplus j)$. Finalmente, calcular la sucesión ν a partir del conjunto de ternas anterior.
- Calcular los elementos de la sucesión ν, es decir, para cada i, calcular ν_i. Así mismo, calcular la sucesión completa a partir del un semigrupo numérico.
- Calcular el semigrupo asociado a una sucesión ν, y determinar si una determinada sucesión es una sucesión ν.
- Similarmente, para la sucesión τ, calcular los elementos, τ_i, y calcular la sucesión completa a partir del un semigrupo numérico. Calcular el semigrupo a partir de la sucesión τ.
- Dada la operación $\oplus$, calcular el semigrupo asociado.
- Mejorar el algoritmo calcular distancias de Feng-Rao para tipos concretos de semigrupo, como los semigrupos de tipo Arf, simétricos, generados por dos elementos, etc.

Estos son los objetivos relacionados con la implementación. A nivel conceptual, este trabajo tiene los siguientes objetivos:

- Estudiar bibliografía relacionada con los semigrupos e ideales de semigrupos, principalmente [11] y [1].
- Estudiar la bibliografía relacionada con las sucesiones ν, τ y la operación $\oplus$. La mayor parte del esto puede encontrarse en [2], y más información en el Trabajo de Matemáticas [15] y el capítulo 3 de este trabajo.
- Estudiar la bibliografía asociada e la distancia de Feng-Rao [4] [8] [12] [3] [7], y buscar resultados relacionados con el cálculo de las distancias de Feng-Rao que puedan mejorar el cálculo.

- Familiarizarme con el lenguaje GAP y la funcionalidad existente en la librería “NumericalSpgs”.

Capítulo 1

Metodología de trabajo

En esta sección vamos a describir y seleccionar la metodología de trabajo que utilizaremos y las etapas en las cuales dividiremos el proyecto.

Como hemos dicho, la primera consideración es seleccionar una metodología de trabajo. En este trabajo, vamos a usar una metodología clásica, en particular un modelo incremental. El principal motivo para la selección de esta metodología es que el desarrollo del proyecto consiste en crear múltiples funciones, independientes entre si y cuyo requisito principal el computo de un cierto objeto matemático. Cada una de estas funciones puede verse como un pequeño proyecto de desarrollo en si mismo, por lo que el proyecto, de forma global consiste en múltiples desarrollos en cascada. Así mismo, la memoria se hará por capítulos (incrementos) cada uno de los cuales el tutor validada por separado y a medida que son terminados.

Además, el proyecto tiene un "Cliente", Pedro, quien gestiona la librería y debe aprobar las contribuciones. Durante el desarrollo, se validarán el software periódicamente, tras terminar la codificación de un conjunto de funciones. Se considera entonces posibles correcciones o mejoras y en función del tiempo restante, ampliar o reducir el alcance. Notemos que si bien el alcance puede cambiar, los requisitos de cada una de las funciones son rígidos. Esto hace que sea particularmente viable el uso de una metodología de desarrollo

1.1. Ciclo de vida del proyecto

Dividiremos el proyecto en las siguiente etapas, teniendo en cuenta que al final del proceso de investigación y codificación de cada etapa, documento el proceso en la memoria del trabajo:

1. La primera fase de este trabajo, podríamos considerar, que comienza con el trabajo de matemáticas, (ya terminado) pues algunos de los conceptos explorados en dicho trabajo sirven como fundación para este trabajo [15]. Propiamente, la primera fase consiste en el estudio bibliográfico (o en este caso, repaso de los conceptos básicos sobre semigrupos

numéricos, principalmente a través del libro [11] para las nociones de semigrupos numéricos, [1] para las de ideales de semigrupos numéricos y [2] para los conceptos de sucesión ν, τ y $\oplus$.

2. En la segunda fase continuo el estudio bibliográfico, centrado en ideales y en particular, en lo que respecta a la descomposición en componentes irreducibles, que puede encontrarse en 3.3 de [1]. Realizaré la codificación de las funciones de descomposición en componentes irreducibles para ideales relativos e ideales propios.

3. En la tercera fase realizaré el estudio bibliográfico sobre las sucesiones ν y τ, así como la operación $\oplus$. En [2], se dan demostraciones constructivas de varias proposiciones que nos dan información de como se puede usar estos conceptos para caracterizar el semigrupo. Así, basándome en en estas demostraciones podré dar algoritmos que permitan realizar estas tareas.

4. Estudio las distancias de Feng y Rao clásicas y las distancias de Feng y Rao generalizadas [3]. Debo estudiar los resultados presentados en [4], [8], [12], [12] y [3] para mejorar la función "*FengRaoDistance(s,r,m)*" actualmente implementada en la librería "NumericalSpgs". Implementar las mejoras en GAP de dicha función.

5. En la quinta etapa realizo una extensa batería de pruebas para verificar que la implementación es correcta.

6. En la sexta y última etapa, realizo un estudio de optimalidad de la distancia de Feng y Rao. En esta fase estudiamos las posibles mejoras de eficiencia para la distancia de Feng y Rao, comparando la eficiencia de la implementación frente a las mejoras implementadas. En esta fase es posible realizar algunos ajustes a esta función.

1.2. Herramientas Utilizadas

En el desarrollo de este proyecto he usado las siguientes herramientas, empezando por las herramientas de desarrollo:

- GAP, en la versión 4.11.0, así como la versión 1.2.2 de la librería "NumericalSpgs". Es necesario el interprete para usar el lenguaje GAP, así como la consola para realizar pruebas.

- El editor *Atom*, como entorno de desarrollo, usando el *plug-in* de GAP. *Atom* es un editor de texto moderno y multiplataforma. Tiene la ventaja de que existe un plugin con la sintaxis de GAP.

- *Github*, para control de versiones, tanto para almacenar el código programado para este trabajo [14], como para consultar el código fuente de la librería "NumericalSpgs"[5].

- Los manuales de usuario y documentación disponible para GAP [10], así como la documentación disponible en la página web oficial [9]. También he usado el manual de la librería "NumericalSpgs"[6].

Otras herramientas que usado para el desarrollo del proyecto:

- *Overleaf*, un editor colaborativo para el lenguaje LaTeX que proporciona almacenamiento en la nube. Es el editor que he usado para escribir este documento. La característica principal que me ha llevado a elegirlo frente a otros editores LaTeX es la facilidad de sincronización entre ordenadores y sistemas operativos, pudiendo trabajar desde la instalación de *ubuntu* o desde *windows* a través del navegador.

1.3. Planificación y distribución del trabajo

Este trabajo es un proyecto a desarrollar, principalmente, por una única persona, que debe asumir los diferentes roles comúnmente asociados a un proyecto de desarrollo de software. Sin embargo, hay otros actores, que participan en el proyecto.

1.3.1. Roles en el desarrollo del proyecto

Por un lado está Pedro A. Gracia Sánchez, quien, como hemos dicho, gestiona la librería "NumericalSpgs" y es el cliente para quien se desarrolla el proyecto. Como cliente, tendrá un papel activo en el proyecto de desarrollo, ya que deberá aprobar las funciones que desarrolle en cada incremento y establer los requisitos del siguiente incremento. José Ignacio Farrán es el segundo cliente del proyecto, estableciendo los requisitos del TFG como proyecto, es decir, de la memoria y su contendido a mayores del código. Su papel también activo, validando el progreso de los incrementos realizados.

El resto de los roles son tarea del alumno, empezando con el trabajo como jefe de proyecto. La responsabilidades asociadas al jefe de proyecto incluyen la planificación temporal de las tarea, así como la selección y priorizaron de los requisitos, ya que el tiempo será limitado, y es necesario limitar el alcance del proyecto. Así, el jefe de proyecto deberá, dentro del plazo establecido, destinar tiempo a cada tarea para realizar el mejor proyecto posible. Deberá gestionar riesgos, principalmente asociados a plazos, pero otros riesgos incluyen los lugares de trabajo, puesto que bibliotecas y salas de estudio son susceptibles a cierres debido a que el proyecto se desarrollara durante una pandemia.

El segundo rol a desempeñar es de analista, siendo necesario elicitar los requisitos del proyecto a través de reuniones con los clientes y dar una estimación temporal de cada una de las

tareas. Dichas tareas incluyen las de lectura bibliográfica, desarrollo del producto software y redacción y edición de la memoria. En esta tarea, la experiencia con otro trabajo de fin de grado ayuda a realizar una mejor estimación temporal.

El tercer y último trabajo es el desarrollador, desempeñando no solo las tareas de que de programación, pruebas y documentación, si no también la investigación bibliográfica. Esta última tarea es importante en todos los roles, aunque en este caso solo hay un persona en el proyecto la incluyo dentro del rol de programador. En cualquier caso, esta tarea deberá realizarse antes que las demás.

1.3.2. Estimación temporal

Es común que en un proyecto de desarrollo software, la estimación temporal se determine en base al trabajo a realizado (los objetivos y alcance), de modo que estos se puedan cumplir en el menor tiempo posible (Técnicas de estimación, por ejemplo, puntos de función). En el caso del presente proyecto, me veo fuertemente limitado temporalmente (por motivos externos) a presentar los resultados del proyecto durante el mes de febrero de 2021.

Por tanto, establezco como fecha para terminar el proyecto el fin del mes de enero, el día 31 de enero de 2021. Así, será necesario ajustar el trabajo al tiempo disponible. Por supuesto, el trabajo debe ser aprobado por el tutor, José Ignacio Farrán, para su entrega. Gracias a la experiencia con el trabajo de matemáticas, sé que el tiempo disponible debería permitirme cumplir los objetivos establecidos.

La fecha de inicio del trabajo es el 16 de Noviembre de 2020. Teniendo en cuenta la fecha de entrega establecida, hay 48 días de trabajo completos y 15 días de trabajo con media jornada, debido a la necesidad de terminar el trabajo de matemáticas durante esos días. Trabajaré de lunes a sábado, y teniendo en cuenta 3 días festivos y 2 días dedicados a elaborar la presentación (y presentar) el TFG de matemáticas. En base a esto, el proyecto dispone de 444 horas y 63 días.

Examinando esta estimación al final del proyecto, vemos que los objetivos se han cumplido (sin necesidad de limitar el alcance), si bien se ha excedido la fecha límite establecida en 2 semanas.

1.3.3. Presupuesto

Como hemos dicho, este proyecto tiene como cliente a Pedro A. Garcia, quien gestiona la librería “NumericalSpgs”. El código generado como parte de este trabajo pasará a formar parte

de dicha librería. Si bien, en términos monetarios, el proyecto no se va a vender, incluimos en el presupuesto la mano de obra, en horas, y lo combinaremos con el coste en euros, así como un coste total en euros dado un salario hipotético.

- La mano de obra es, en este caso, gratuita, pero en cierto modo es parte del presupuesto, como hemos dicho, estimamos el tiempo de trabajo en 444 horas. Si considerásemos un salario de 10€ por cada hora, el coste asociado la mano de obra sería de 4440€.

- Como costes variables, tengo el los costes de hardware, asociados a mi portátil, cuya vida útil es de 6 años y su coste es de 900€, es decir, 0′41088€/día. así mismo, los costes de los periféricos son de 0′013704€/día. Teniendo en cuenta que estimamos que la duración del proyecto será de 63 días, el presupuesto hardware total es de 26′75€.

- Todo el software que voy a usar es software gratuito, por lo que no hay costes asociados. Así mismo, no es necesario imprimir la memoria, ni adquirir copias físicas de los libros y artículos, por lo que no hay coste asociado.

Así el presupuesto total de este trabajo de fin de grado es de **26′75€** y **444** horas. Con el salario ficticio que hemos establecido, el presupuesto sería de 4466′75€.

1.4. Seguimiento

Examinamos, al final del proceso de desarrollo, las horas de trabajo seguidas, y comparamos con la estimación inicial. La distribución final de las horas ha variado en función de las semanas y los meses, pero aproximadamente, puedo decir lo siguiente:

- Desde finales de agosto a diciembre he trabajado en el trabajo de fin de grado de matemáticas, que ha servido de base para el presente trabajo. Empecé el trabajo a mediados de noviembre, de forma paralela al de matemáticas, trabajando 13 días (Aproximadamente 60 horas)

- En diciembre he dedicado 16 días, de media algo más de 8 horas a este trabajo (teniendo que una semana la dedique a preparar la presentación del TFG de matemáticas) total de 138 horas).

- En enero he trabajado a tiempo completo en el trabajo, por un tiempo total de 200 horas y 28 días.

- La primera semana de Febrero, y parte he trabajado a tiempo completo, un total de 10 días, y 80 horas

El tiempo total es de 478 horas y 67 días trabajados (aproximadamente, si contar la parte de matemáticas). Contrastamos esto con la estimación inicial de 444 horas y 63 días, y observamos una sobrecarga de 34 horas, y un retraso de la entrega en 10 días.

La principal discrepancia con la estimación inicial es que el proyecto se ha extendido dos semanas más que la estimación inicial. Esto se debe, en parte, a que tuve que dedicar una semana de Noviembre a terminar el TFG de matemáticas. El resto se debe otros factores.

En los calendarios y el diagrama de Gantt podemos ver la distribución por días y semanas, así como las tareas realizadas, incluyendo entregar parciales de la memoria y el código.

Figura 1.1: Diagrama de Gantt

	Previo	Nov.	Nov.	Dic.	Dic.	Dic.	Dic.	Dic.	Dic.	Enero	Enero	Enero	Enero	Feb.	Feb.
Tareas\Semana		Sem. 1	Sem. 2	Sem. 3	Sem. 4	Sem. 5	Sem. 6	Sem. 7	Sem. 8	Sem. 9	Sem. 10	Sem. 11	Sem. 12	Sem. 15	Sem. 14
Etapa 1, investigación previa															
Investigación Ideales															
Programación ideales															
Caracterización: investigación															
Caracterización: programación															
Memoria capítulo 2															
Investiga dist. de Feng-Rao															
Memoria capítulos 1 y 3															
Programación dist. Feng-Rao															
Memoria capítulo 4															
Revisión y pruebas de código															
Capítulo 5 y conclusión															
Revisión de la memoria															

Figura 1.2: Trabajo mes de Noviembre

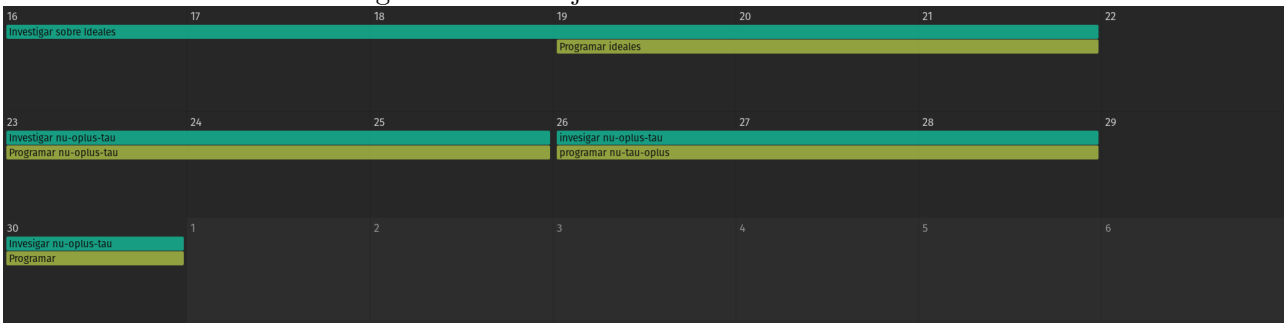


Figura 1.3: Trabajo mes de Diciembre

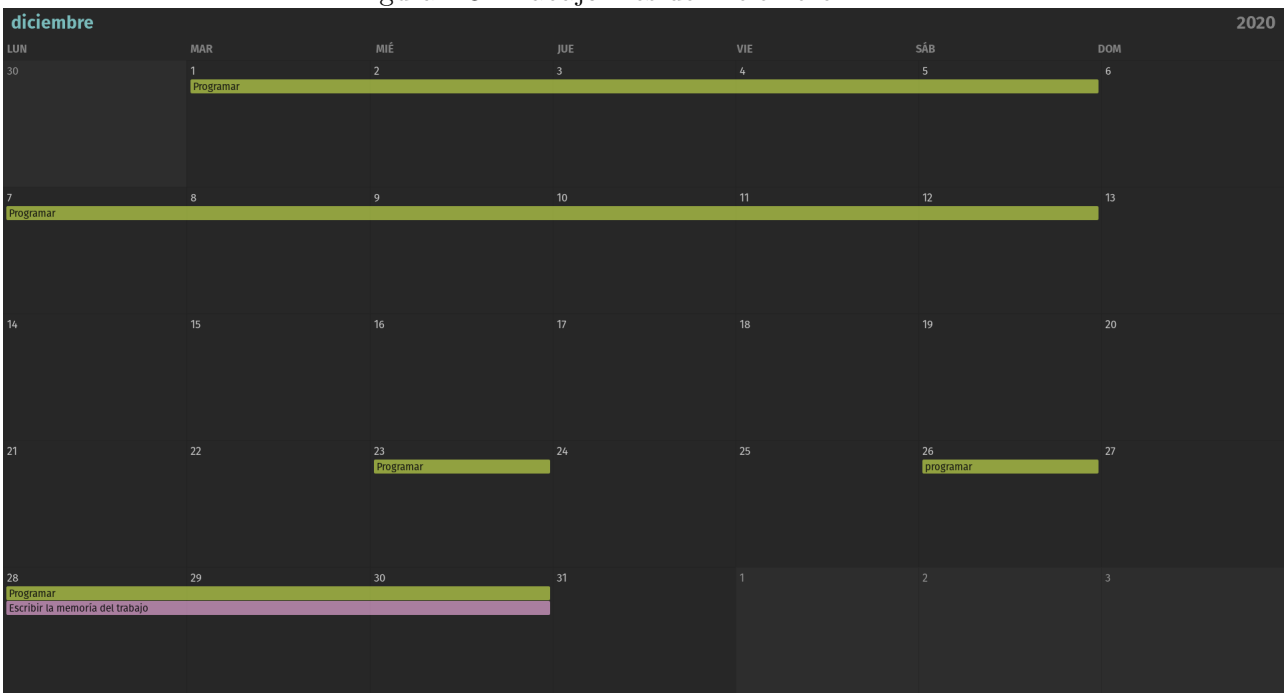


Figura 1.4: Trabajo mes de enero

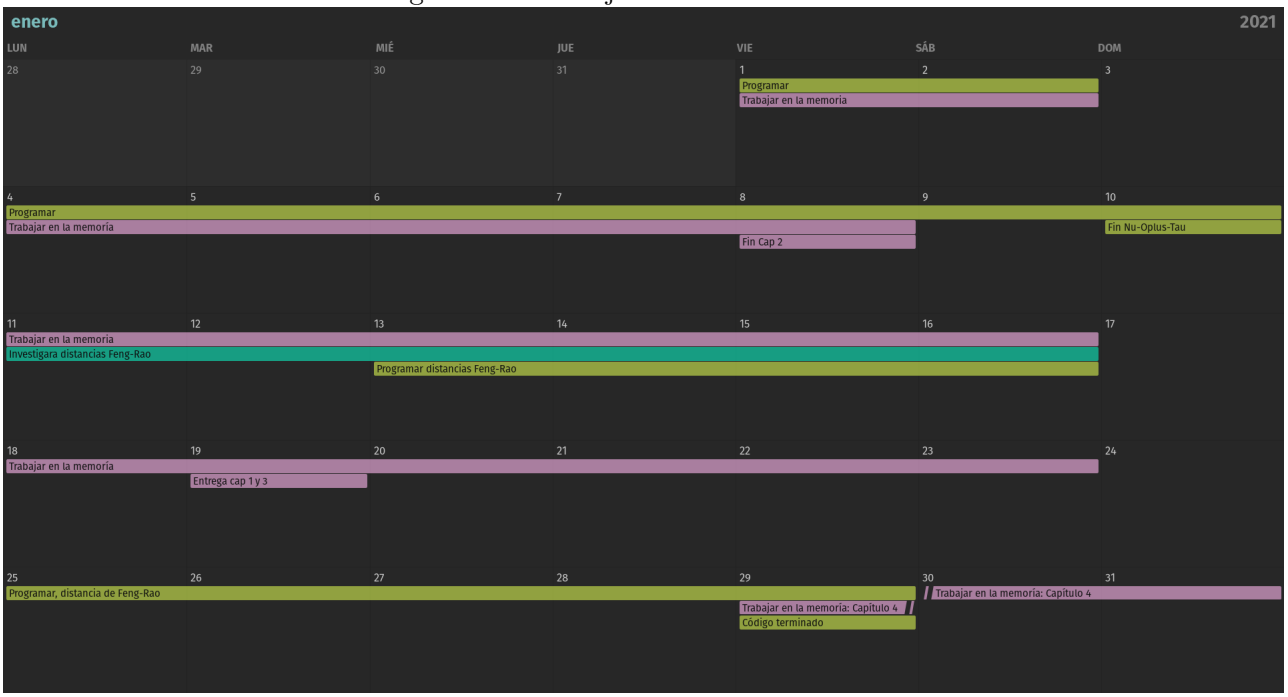
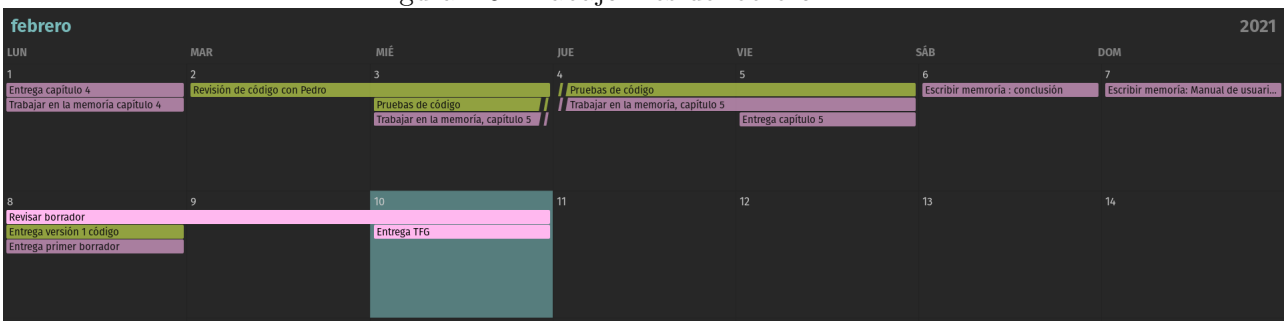


Figura 1.5: Trabajo mes de febrero



Capítulo 2

El lenguaje GAP y semigrupos numéricos

Vamos a introducir el lenguaje GAP y la librería "NumericalSpgs", en la que nos apoyaremos para dar las nociones teóricas de semigrupos numéricos, necesarias para este trabajo.

2.1. GAP: Sistema de álgebra computacional

GAP es un acrónimo de *Groups, Algorithms, Programming*, y se trata de un lenguaje y sistema para álgebra discreta computacional [9], con un énfasis particular en teoría de grupos.

Originalmente, GAP fue desarrollado en la cátedra *"Lehrstuhl D für Mathematik"* de la Universidad Técnica de Aquisgrán, Alemania, entre 1986 y 1997. A partir de 1997, el desarrollo de GAP y su mantenimiento pasó a ser coordinado por la *"School of Mathematical and Computational Sciences"* de la Universidad de *"Saint Andrews en Escocia"*. Posteriormente, en 2005, la coordinación pasó a depender de manera conjunta de una asociación de cuatro centros académicos (centros GAP), ubicados en la Universidad de *"Saint Andrews"*, la Universidad Técnica de Aquisgrán, la Universidad Técnica de *"Brunswick"* y la Universidad Estatal de Colorado en *"Fort Collins"*, Colorado. Recientemente, en 2020, un quinto centro fue añadido, la *"T.U Kaiserslautern"*

GAP es un sistema, formado por múltiples componentes, pero principalmente dos:

- El sistema central, que consta de:
  - Un núcleo (*Kernel*) escrito en *C*, que incluye un interprete del lenguaje GAP, así como las funciones y algoritmos básicos.
  - Un gran abanico de librerías funciones que implementan algoritmos algebraicos en el lenguaje GAP.

  - Un conjunto de librerías de datos que implementan objetos algebraicos, como los grupos de orden bajo. GAP proporciona un entorno o consola interactiva, que permite usar la funcionalidad de GAP y sus librerías para realizar cálculos.
  - El manual [9] [10].

- El conjunto de paquetes, implementados por los usuario, constituyen una característica muy importante del sistema y que extienden la funcionalidad del lenguaje. GAP ofrece a los autores de paquetes la oportunidad de someterlos a revisión, proceso que contribuye a mejorar la calidad final de los paquetes y que proporciona al autor un reconocimiento similar al de las publicaciones académicas. Algunos de estos paquetes son posteriormente incluidos como parte de la instalación de GAP. En la versión 4.11.0 de GAP, en Enero de 2020, hay en torno a 80 paquetes distribuidos con GAP. Otros pueden deben ser descargados e instalados. Aquellos paquetes instalados que no se carguen por defecto al iniciar GAP deben ser cargados con: "*LoadPakage("«NombrePaquete»");*".

Existen múltiples interfaces para usar GAP. Está disponible una interfaz para usar el CAS SINGULAR dentro de GAP. Asimismo, ambos pueden usarse dentro de la interfaz proporcionada por SageMath. También puede ejecutarse a traves de la SHELL de un sistema UNIX:

Figura 2.1: Consola GAP, ejecutada desde la SHELL

```
jorge@pop-os:~$ cd /opt/gap-4.11.0/bin
jorge@pop-os:/opt/gap-4.11.0/bin$ ./gap.sh
 ┌─────┐    GAP 4.11.0 of 29-Feb-2020
 │ GAP │    https://www.gap-system.org
 └─────┘    Architecture: x86_64-pc-linux-gnu-default64-kv7
 Configuration:  gmp 6.2.0, GASMAN, readline
 Loading the library and packages ...
 Packages:   AClib 1.3.2, Alnuth 3.1.2, AtlasRep 2.1.0, AutoDoc 2019.09.04,
             AutPGrp 1.10.2, Browse 1.8.8, CaratInterface 2.3.3, CRISP 1.4.5,
             Cryst 4.1.23, CrystCat 1.1.9, CTblLib 1.2.2, FactInt 1.6.3,
             FGA 1.4.0, Forms 1.2.5, GAPDoc 1.6.3, genss 1.6.6, IO 4.7.0,
             IRREDSOL 1.4, LAGUNA 3.9.3, orb 4.8.3, Polenta 1.3.9,
             Polycyclic 2.15.1, PrimGrp 3.4.0, RadiRoot 2.8, recog 1.3.2,
             ResClasses 4.7.2, SmallGrp 1.4.1, Sophus 1.24, SpinSym 1.5.2,
             TomLib 1.2.9, TransGrp 2.0.5, utils 0.69
 Try '??help' for help. See also '?copyright', '?cite' and '?authors'
gap> LoadPackage("NumericalSgps");
-----------------------------------------------------------------
Loading  NumericalSgps 1.2.1
For help, type: ?NumericalSgps:
To gain profit from other packages, please refer to chapter
'External Packages' in the manual, or type: ?NumSgpsUse
-----------------------------------------------------------------
true
gap> s1 := NumericalSemigroup(7,11,15);
<Numerical semigroup with 3 generators>
gap> SmallElements(s1);
[ 0, 7, 11, 14, 15, 18, 21, 22, 25, 26, 28, 29, 30, 32, 33, 35, 36, 37, 39 ]
gap> Gaps(s1);
[ 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 16, 17, 19, 20, 23, 24, 27, 31, 34, 38 ]
gap> Genus(s1);
21
gap> Conductor(s1);
39
```

GAP está pensado para uso académico, tanto en investigación como didáctico, en el estudio de anillos, espacios vectoriales, álgebras, estructuras combinatorias y, de interés para nuestro trabajo, semigrupos numéricos (entre otros muchos). Se trata de un sistema de código abierto, distribuido bajo la licencia GPL (o GNU).

2.1.1. Librería "NumericalSpgs"

Es una librería GAP, para realizar cálculos con semigrupos numéricos [5]. Según introduzcamos los semigrupos numéricos, hablaremos de las funciones de esta librería para tratar los conceptos teóricos en GAP. Información detallada se puede encontrar en el manual. [6]. Todo, o casi todo el código desarrollado en este trabajo tiene como objetivo extender la funcionalidad de esta librería.

2.2. Semigrupos numéricos

Los semigrupos numéricos son uno de los conceptos fundamentales de este trabajo, son el objeto base para muchas de las definiciones y algoritmos de los cuales hablaremos, y vienen dados por la siguiente definición. La mayor parte de las definiciones básicas sobre semigrupos se pueden encontrar en el libro de García Sánchez [11]. En el trabajo de fin de grado de matemáticas [15] entro en detalles sobre las demostraciones y la base teórica. En esta sección nos centramos en como manejar semigrupos como objetos en GAP.

Definición 2.3.1: Sea $\mathbb{N}_0$ el conjunto delos números naturales con el 0. Un semigrupo numérico es un par $(S, +)$, donde S es un subconjunto de $\mathbb{N}_0$, y la operación "+" es la suma habitual en los naturales. Dicha operación tiene elemento unitario 0. Pediremos como requisito que el cero forme parte del semigrupo. Consideramos además los semigrupos con la propiedad adicional de tener complemento finito en $\mathbb{N}_0$. Es decir, $S \subseteq \mathbb{N}_0$ es **un semigrupo numérico** si:

1. $0 \in S$
2. Si $\forall s, s' \in S$, entonces $(s + s') \in S$
3. $|\mathbb{N}_0 \setminus S| < \infty$ (En general asumimos que se cumple esta condición, pero es posible definir semigrupos numéricos que no verifican esta condución).

Cuando hablemos de semigrupos en este trabajo, nos referiremos a semigrupos numéricos.
Ejemplo 2.3.2:

- Los números naturales con la suma habitual $(\mathbb{N}, +)$ son un semigrupo numérico.
- El conjunto $\{0, 4, 5, 8, 9, 10, 12, 13, 14, \ldots\} \subseteq \mathbb{N}_0$ es uns emigrupo numérico.

Es posible caracterizar los semigrupos a partir un conjunto finito de generadores como:

Definición 2.3.3: Dado un semigrupo numérico S y $A \subset S$ decimos que A es un sistema de generadores de S si,

$$\langle A\rangle := \{\lambda_1 a_1 + \ldots + \lambda_n a_n \quad | \quad n \in \mathbb{N}_0,\ \lambda_1, \ldots, \lambda_n \in \mathbb{N}_0 \quad y \quad a_1, \ldots, a_n \in A\}.$$

Decimos que es un sistema minimal, si ningún subconjunto propio del mismo genera el semigrupo completo.
Así mismo, dado un conjunto finito de números naturales, A, definimos $S' := \langle A\rangle$, el semigrupo generado por A. Es un semigrupo numérico si, y solo si, el máximo común divisor de los elementos de A es 1.

Ejemplo 2.3.4: Dado el conjunto $A = \{3, 7\}$ obtenemos el semigrupo

$$S = \langle A \rangle = \{0, 3, 6, 7, 9, 10, 12, 13, 14, 15, 16, 17, \ldots\}.$$

Usamos en este caso la notación $S = \langle A \rangle = \{0, 3, 6, 7, 9, 10, 12, \rightarrow\}$, para denotar que todos los naturales mayores que 12 están en el semigrupo.

Semigrupos numéricos con "NumericalSpgs"
Vamos a ver como podemos hacer esto usando GAP, y la librería "NumericalSpgs". Así, una forma de caracterizar el semigrupo es a partir de los generadores.

```
gap> s1 := NumericalSemigroup(7,11,15);
<Numerical semigroup with 3 generators>
gap> SmallElements(s1);
[0,7,11,14,15,18,21,22,25,26,28,29,30,32,33,35,36,37,39]
gap> Generators(s1);
[ 7, 11, 15 ]
```

La función "NumericalSemigroup(7, 11, 15)" define un objeto semigrupo numérico generado por 7, 11 y 15.
SmallElements muestra los elementos del semigrupo menores iguales al conductor (definimos conductor más adelante). Así, el semigrupo $s1$ que hemos definido es $s1 := \langle 7, 11, 15 \rangle = \{0, 7, 11, 14, 15, 18, 21, 22, 25, 26, 28, 29, 30, 32, 33, 35, 36, 37, 39, \rightarrow\}$
"Gnerators" Nos da los generadores del semigrupo.

Definición 2.3.5: Dado un semigrupo numérico S, llamamos al mayor entero que no está en S **Número de Fröbenius** de S y se denota por $F(S)$. También se usa el concepto del *conductor*, que es el menor entero $C(S)$ tal que $C(S) + n \in S,\ \forall n \in \mathbb{N}$. El conducto es el número de Fröbenius más uno.

Conductor y número de Fröbenius con GAP:

```
gap> Conductor(s1);
39
gap> FrobeniusNumber(s1);
38
```

Definición 2.3.6: Dado un semigrupo numérico S, denominamos **lagunas** o **lagunas** de S (gaps en ingles) al conjunto $G(S) = \mathbb{N} \setminus S$. La cardinalidad de dicho conjunto se llama **género** o **grado de singularidad** de S y se denota por $g(S)$.

Lagunas y género con GAP:
Podemos calcular las lagunas y el género usando "NumericalSpgs"con las funciones "Gaps(s)" y "Genus(s)" respectivamente (siendo estos los nombres en inglés). Vemos un ejemplo para el semigrupo $s1 := \langle 7, 11, 15\rangle$.

```
gap> Gaps(s1);
[ 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 16,
17, 19, 20, 23, 24, 27, 31, 34, 38 ]
gap> Genus(s1);
21
```

También es posible caracterizar el semigrupo por las lagunas, pues el complemento de las lagunas es el propio semigrupo.

```
gap> g := Gaps(s1);;
gap> s := NumericalSemigroupByGaps(g);;
gap> s = s1;
true
```

No hay una fórmula calcular el género y género a partir de los generadores en general, pero si que se conoce para el caso con dos generadores:

Proposición 2.3.7: Sea S, el semigrupo numérico generado por los enteros a, b, co-primos entre si $(m.c.d(a, b) = 1)$

- $F(\langle a, b\rangle) = ab - a - b$
- $g(\langle a, b\rangle) = \frac{(ab-a-b+1)}{2}$

Lema 2.3.8: Sea S un semigrupo numérico con conductor $C(S)$ y grado de singularidad $g(S)$, entonces:

$$2g(S) \geq C(S)$$

Cuando se da la igualdad en la desigualdad anterior, hablamos de semigrupos simétricos:

Definición 2.3.9: Decimos que un semigrupo numérico S con grado de singularidad $g(S)$ y conductor c es simétrico si: $C(S) = 2g(S)$

Definición 2.3.10: Decimos que un semigrupo numérico es irreducible si no puede ser expresado como intersección de dos semigrupos que lo contienen de forma propia.

Irredicibilidad y simetría con GAP Los objetos de tipo semigrupo, en "NumericalSpgs", tienen las propiedades "IsSymmetric" e "IsIrredicible":

```
gap> s:=NumericalSemigroup(4,5);;
gap> IsIrreducible(s);
true
gap> IsSymmetric(s);
true
```

2.3. Ideales de Semigrupos Numéricos

Vamos a hablar de ideales de semigrupos numéricos. El concepto de ideal, propio de la teoría de anillos, se puede generalizar al contexto de los semigrupos numéricos. Sobre ideales de semigrupos numéricos, podemos encontrar más en [1].

Definición 2.4.1: Sea S un semigrupo numérico, decimos que $E \subset \mathbb{Z}$ es un *ideal relativo* de S si:

- $S + E = \{s + e \mid s \in S,\ e \in E\} \subset E$
- Existe $s \in S$ tal que $s + E = \{s + e \mid e \in E\} \subset S$

La segunda condición asegura que E tiene mínimo que denotaremos por $m(E)$ y llamaremos *multiplicidad* de E.

Ejemplos 1.4.2:

- Un ejemplo de ideal propio es S, que claramente es ideal de si mismo.

- Dado un semigrupo numérico S, podemos definir ideales relativos de S tomando un conjunto finito $A \subset \mathbb{Z}$ y definiendo $E := A+S$. Por ejemplo, dado $S = \{0, 3, 6, 7, 9, 10, 12, \rightarrow\}$ y $A = \{-1\}$; $E := \{-1\} + \{0, 3, 6, 7, 9, 10, 12\} = \{-1, 2, 5, 6, 8, 9, 11, \rightarrow\}$. La segunda condición se verifica, pues si $s = 12$, $s + E \subseteq S$. La primera condición se verifica, pues $\forall e \in E,\ \exists s_1 \in S,\ tal\ que\ e = -1 + s_1$. Entonces para cualquier $s_2 \in S$ tenemos $e + s_2 = -1 + s_1 + s_2 = -1 + s \in E,\ s \in S$.

- Veamos un ejemplo de ideal propio. Dado $\{0, 3, 6, 7, 9, 10, 12, \rightarrow\}S$ semigrupo numérico, podemos definir $D(12) := \{y \in S \mid 12 - y \in S\} = \{0, 3, 6, 9, 12\}$ (veremos más sobre este tipo de conjuntos más adelante). Entonces $S \setminus D(12) = \{7, 10, 13, \rightarrow\} \subseteq S$. Vemos que verifica ambas condiciones.

Definición 2.4.3: Dado un ideal relativo E del semigrupo numérico S decimos que un conjunto $\{e_1, \ldots, e_n\} \subset E$ es un sistema de generadores de E si podemos expresar un elemento cualquiera $e \in E$ como $e = e_i + s,\ s \in S,\ i \in \{1, 2, \ldots, n\}$.

Definición de ideales con GAP:
Podemos definir un ideal a partir de un conjunto de genradores, usando "IdealOfNumericalSemigroup" o como la suma de los generadores entre corchetes más el semigrupo. Así mismo, podemos usar la función "SmallElements" para obtener los elementos hasta el conductor. (A partir de el último elemento que se obtiene como resultado, todos los naturales pertenecen al ideal). Así mismo, podemos obtner los generadores del ideal usando la función MinimalGenerators(I), para obtener un sistema minimal de generadores.

```
s:=NumericalSemigroup(3,11,17);
<Numerical semigroup with 3 generators>
gap> I:=[-5,5]+s;
<Ideal of numerical semigroup>
gap> MinimalGenerators(I);
[ -5, 5 ]
gap> SmallElements(I);
[ -5, -2, 1, 4 ]
gap> I1:=IdealOfNumericalSemigroup([-5,5],s);
<Ideal of numerical semigroup>
gap> IsIntegral(I);
false
gap> I1=I;
true
```

Podemos verificar si un ideal es propio usando "IsIntegral".

Los siguientes ideales son particularmente relevantes:

Definición 2.4.4: Sea S un semigrupo numérico, definimos:

1. $M = S^* = S \setminus \{0\}$, ideal propio de S al que denominamos *ideal maximal*.
2. El *ideal conductor* es: $S - \mathbb{N} = \{z \in \mathbb{Z} \mid z + \mathbb{N} \subset S\} = \{C(S), \rightarrow\} = C(S) + \mathbb{N}$. Este es el mayor ideal común a $\mathbb{N}$ y S.
3. El *ideal canónico estándar* se define como. $K(S) = \{x \in \mathbb{Z} \mid F(S) - x \notin S\}$.

Ideal Canónico y maximal con GAP Podemos usar las funciones "MaximalIdeal" y "CanonicalIdeal" para calcular estos ideales.

```
gap> s1:=NumericalSemigroup(7,23,29);;
gap> SmallElements(CanonicalIdeal(s1));
[ 0, 6, 7, 13, 14, 20, 21, 23, 27, 28, 29, 30, 34, 35, 36, 37,
```

```
41, 42, 43, 44, 46, 48, 49, 50, 51, 52, 53, 55, 56, 57, 58, 59, 60, 62,
  63, 64, 65, 66, 67, 69 ]
gap> SmallElements(s1);
[ 0, 7, 14, 21, 23, 28, 29, 30, 35, 36, 37, 42, 43, 44,
46, 49, 50, 51, 52, 53, 56, 57, 58, 59, 60, 63, 64, 65, 66, 67, 69 ]
gap> SmallElements(MaximalIdeal(s1));
[ 7, 14, 21, 23, 28, 29, 30, 35, 36, 37, 42, 43, 44,
46, 49, 50, 51, 52, 53, 56, 57, 58, 59, 60, 63, 64, 65, 66, 67, 69 ]
```

Vemos qué es posible definir operaciones de suma y resta de ideales:

Definición 2.4.5: Dados dos ideales relativos E y H de un semigrupo numérico S, definimos:

- $E + H = \{e + h \mid h \in H,\ e \in E\}$
- $H - E = \{z \in \mathbb{Z} \mid z + E \subset H\}$

Ejemplo 2.4.6: Tomemos el semigrupo numérico $S = \langle 10, 13, 21, 22\rangle$:

Consideremos los ideales $E = \{10, 11\} + S$ y $H = \{-1, 2, 3\} + S$. Operando (con ayuda de GAP):

$$E + H = \{9, 10, 12, 13, 14, 19, 20, 22, 23, 24, 25, 26, 27, 29, \rightarrow\}$$
$$E - H = \{21, 31, 34, 38, 41, 42, 43, 44, 47, 48, 50, \rightarrow\}$$

Suma y resta de ideales con GAP:
Podemos usar los operadores "+" y "-" para operar con ideales:

```
gap> s1:=NumericalSemigroup(7,23,29);;
I:=[-5,5]+s;;
gap> I1:=[-1]+s;;
gap> SmallElements(I+I1);
[ -6, -3, 0, 3 ]
gap> SmallElements(I-I1);
[ -4, -1, 2, 5 ]
```

Veamos que las operaciones entre dos ideales de S resultan en un ideal de S.

Proposición 2.4.7: Sean E y H ideales relativos cualquiera de un semigrupo numérico S. Entonces $E + H$ y $H - E$ son también ideales relativos de S.

A continuación, vemos que algunos conceptos de semigrupos pueden ser extendidos a ideales, como el de número Fröbenius:

Definición 2.4.8: Dado un ideal relativo E de un semigrupo numérico S, llamaremos número de Fröbenius de E a $Max(\mathbb{Z} \setminus E)$ y lo denotaremos por $F(E)$.

El hecho que E este acotado inferiormente asegura que el máximo existe.

Ejemplo 2.4.9:

- Si consideramos a S como ideal de sí mismo, vemos que la definición de número de Fröbenius coincide con la definición para semigrupos:
  Si $S = \{3, 6, 7, 9, 10, 12, \rightarrow\}$ entonces $F(S) = max(\mathbb{Z} \setminus S) = 11$
- Para ideal $E = S \setminus D(12)$ n (ver el primer ejemplo de este capitulo), $F(E) = 12$.

Proposición 2.4.10: Dado un ideal relativo E de un semigrupo numérico, $E \setminus (M + E)$ es un sistema minimal de generadores de E.

Ejemplo 2.4.11: Veamos un ejemplo, dado $S = \{3, 6, 7, 9, 10, 12, \rightarrow\}$, computemos un sistema de generadores de $E = S \setminus D(12) = \{7, 10, 13, \rightarrow\}$. Como acabamos de ver, $E \setminus (E + M)$ es un sistema de generadores de S. $E + M = \{10, 13, 14, 16, \rightarrow$
$E \setminus (E + M) = \{7, 15\}$. Podemos comprobar que todos los elementos de E se pueden escribir como suma de 7 ó 15 más un elemento de S:

$$7 = 7 + 0,\ 10 = 7 + 3,\ 13 = 7 + 6,\ 14 = 7 + 7\ 15 = 15 + 0,$$
$$16 = 7 + 9,\ 17 = 7 + 10,\ 18 = 15 + 3,\ 19 = 7 + 12,\ \ldots$$

Definición 2.4.11: Decimos que dos ideales relativos H y E de S son equivalentes si existe $x \in \mathbb{Z}$ tal que $E = x + H = \{x + h \mid h \in H\}$

Esto significa que todo ideal relativo de S es equivalente a un ideal propio, pues basta con trasladar E por $m(E)$ para obtener un ideal principal: $m(E) + E \subset S$. Esto define una relación de equivalencia entre ideales relativos de un determinado semigrupo.

En particular, dado un ideal E de S, podemos tomar $\tilde{E} = E - F(E) + F(S)$. Dicho ideal relativo es un ideal propio de S equivalente a E y con el mismo número de Fröbenius que S. Usando esta notación, podemos introducir el siguiente resultado:

Proposición 2.4.12: Sea S un semigrupo numérico y K su ideal canónico estándar. Todo ideal relativo de S es equivalente a un ideal relativo $\tilde{E}$ de modo que $S - \mathbb{N} = \{C(S), \rightarrow\} \subseteq$

$\tilde{E} \subseteq K$

2.4. Irreducibilidad de ideales de semigrupos numéricos

Esta sección vamos a tratar descomposición de ideales, vamos a dar la base teórica, pues la descomposición en irreducibles no está implementado en la librería "NumericalSpgs", y la cual voy a implementar en el capítulo 2.

2.4.1. Irreducibilidad de ideales relativos

Tratamos primero los ideales relativos, discutimos primero la base teórica, y después incluiremos la implementación.

Definición 2.5.1.1: Dado un semigrupo numérico S, y E un semigrupo numérico del mismo, decimos que E es irreducible ($\mathbb{Z}$-irreducible) en S, si no puede ser expresado como intersección finita de otros ideales relativos (distintos de E) de S que lo contienen.

Un ejemplo de un ideal irreducible es K, el ideal canónico. Como hemos visto, no hay ningún ideal relativo de S que lo contenga de forma de propia. A continuación veremos que se trata del único ideal $\mathbb{Z}$-irreducible

Teorema 2.5.1.2: Sea S un semigrupo numérico y E un ideal relativo del mismo. Sea $\{x_1, \ldots, x_h\}$ un conjunto minimal de generadores del ideal $K - E$. Entonces:

$$E = (-x_1 + K) \cap \ldots \cap (-x_h + K)$$

Además esta descomposición es no redundante y única. En particular, el ideal E es irreducible si y solo si es canónico.

Definición 2.5.1.3: Dado semigrupo numérico S y un ideal relativo del mismo E, a cada uno de los ideales de la forma $(-x_i + K)$ en los que según el teorema anterior podemos descomponer E los llamaremos **componentes $\mathbb{Z}$-irreducibles** de E (la unicidad de la descomposición asegura que existe un único conjunto de componentes irreducibles para un ideal dado)

Ejemplo 2.5.1.4: Veamos un ejemplo en GAP de la descomposición descrita en el teorema. Sea $S = \{3, 4, 5\}$, y consideremos el ideal $I = \{4, 5\} + S$

```
gap> S:=NumericalSemigroup(3,5,7);;
```

```
gap> I:=[4,5]+S;;
gap> K:=CanonicalIdeal(S);;
gap> MinimalGenerators(K-I);
[ -2, 2 ]
gap> MinimalGenerators(Intersection(-2+K,2+K));
[ 4, 5 ]
```

Por lo que $\{4,5\} + S = (-2 + K) \cap (2 + K)$. Como vemos, este teorema nos da un algoritmo para saber si un ideal es $\mathbb{Z}$-irreducible y nos da una descomposición en componentes irreducibles del ideal. Como parte de este trabajo de fin de grado he implementado este algoritmo en GAP.

Implementación GAP:

```
IdealDecomposition := function(I,S)
    local K, MG, output, g;
    K:=CanonicalIdeal(S);
    MG:=K-I;
    MG:=MinimalGenerators(MG);
    output:=[];
    for g in MG do
      Add(output, g+K);
    od;
    return output;
end;
```

2.4.2. Irreducibilidad de ideales propios

Tratamos ahora la irreducibilidad de ideales propios, empezando por introducir la nueva noción de irreducibilidad.

Definición 2.5.2.5: Sea S un semigrupo numérico y E un ideal propio de S. Decimos que E es **irreducible** si no puede ser expresado como intersección finita de ideales propios de S que contengan a E propiamente.

S es ideal irreducible de si mismo, por lo que asumiremos en adelante que $0 \notin E$
Queremos un teorema que nos permita obtener una descomposición en elementos irreducibles con el caso de $\mathbb{Z}$-irreducibilidad, por lo que vamos a tener que introducir nuevos conceptos y demostrar proposiciones básicas sobre los mismos:

Definición 2.5.2.6: Sea S un semigrupo numérico:

- Dados dos enteros a,$b \in S$, decimos que $a \leq_S b$ si $b - a \in S$.
- Usando la notación anterior, definimos para $x \in S$, $D(x) = \{s \in S \mid s \leq_S x\} = \{s \in S \mid x - s \in S\}$. Al conjunto $D(x)$ lo denominaremos divisores de x en S.
- Diremos que un conjunto $X \subseteq S$ es cerrado por **divisores** (en S) si tiene la siguiente propiedad: $\forall x \in X$ y para todo $y \in S$ que verifique que $y \leq_S x$ entonces $y \in X$.

Conjuntos de divisores con "NumericalSpgs":

Usamos la función "DivisorsOfElementInNumericalSemigroup" para calcular el conjunto de divisores de un elemento asociado a un elemento del semigrupo.

```
gap> s:=NumericalSemigroup(3,7,11);;
gap> DivisorsOfElementInNumericalSemigroup(12,s) ;
[ 0, 3, 6, 9, 12 ]
```

Lema 2.5.2.7: Sea S un semigrupo numérico, y $x \in S$. Entonces, para todo ideal propio E de S, son equivalentes:

1. $x \notin E$.
2. $E \subseteq (S \setminus D(x))$

Presentamos ahora el teorema de descomposición en irreducibles para ideales propios:

Teorema 2.5.2.8: Sea S un semigrupo numérico, M ($M = S^* = S \setminus \{0\}$) su ideal maximal y sea E un ideal propio de S. Si $(E -_S M) \setminus E = \{x_1, \ldots, x_h\}$. Entonces:

$$E = (S \setminus D(x_1)) \cap \ldots \cap (S \setminus D(x_h))$$

Y esta descomposición de E en ideales de S propios e irreducibles es única y no redundante.

Ejemplo 2.5.2.9: Sean $S = \langle 3, 5, 7\rangle$ e $I = 10 + S$. Vamos a usar el teorema anterior para obtener una descomposición en irreducibles:

```
gap> S:=NumericalSemigroup(3,5,7);;
gap> I:=10+S;;
gap> Difference(Intersection(0+S,I-M),I);
[ 12, 14 ]
```

La penúltima línea calcula primero "Intersection(0+S,I-M)", que nos da $I -_S M$ (la intersección reduce la resta a elementos de S). Después con la función "Diffence()" nos da que $(I -_S M) \setminus I = \langle 12, 14\rangle$. Por tanto:

$$I = (S \setminus D(12)) \cap (S \setminus D(14))$$

Podemos obtener una expresión más explicita para la descomposición:

Vamos usar "d:=x->DivisorsOfElementInNumericalSemigroup(x,S)" para definir una función "$d(x)$" que devuelve los divisores de x en S.

"IdealByDivisorClosedSet(d(x),S)" nos devuelve el ideal $S \setminus D(x)$. Usando "MinimalGenerators()" podemos obtener un sistema de generadores de este ideal:

```
gap> d:=x->DivisorsOfElementInNumericalSemigroup(x,S);;
gap> MinimalGenerators(IdealByDivisorClosedSet(d(12),S));
[ 8, 10 ]
gap> MinimalGenerators(IdealByDivisorClosedSet(d(14),S));
[ 10, 12 ]
```

Por tanto: $I = (\{8, 10\} + S) \cap (\{10; 12\} + S)$.
Comprobamos que efectivamente, es cierto:

```
gap> Intersection(IdealByDivisorClosedSet(d(12),S),
IdealByDivisorClosedSet(d(14),S)) = I;
true
```

Podemos usar las ideal detrás del teorema, y del ejemplo anterior e implementar una función que calcule la descomposición de un ideal propio en componentes irreducibles:

Implementación en GAP

```
#Requieres IdealByDivisorClosedSet.
ProperIdealDecomposition := function(I)
  local M,Dif,S, XI,output,x,d;
  if not(IsIntegral(I)) then
    Error("The argument must be an integral ideal (proper ideal)");
  fi;
  S:=AmbientNumericalSemigroupOfIdeal(I);
  M:=MaximalIdeal(S);
  Dif:=I-M;
  XI:=Difference(Intersection(0+S,Dif),I);
  output:=[];
```

```
  for x in XI do
    d:=DivisorsOfElementInNumericalSemigroup(x,S);
    Add(output,IdealByDivisorClosedSet(d,S));
  od;
  return output;
end;
```

Capítulo 3

Caracterización de semigrupos a partir de $\oplus$, ν y τ

En este capítulo vamos a explorar ciertos conceptos que nos van a permitir caracterizar los semigrupos numéricos de formas alternativas. Introduciremos estos conceptos de forma teórica, y discutiremos la implementación de estas caracterizaciones. Algunos de estos, como es la sucesión ν, guardan relación con la distancia de Feng-Rao, que veremos en el siguiente capítulo.

3.1. Operación $\oplus$

Podemos dar una aplicación biyectiva que nos permita indexar o enumerar los elementos del semigrupo. Dicha aplicación se denomina enumeración:

Definición 3.1.1: Sea S un semigrupo numérico, $S = \{0 = s_0, s_1, \ldots, s_j, \ldots\}$, tal que $\forall i \in \mathbb{N}_0,\ s_i < s_{i+1}$. La aplicación $\lambda : \mathbb{N}_0 \to S$, $\lambda(i) = s_i$ es la **enumeración** del semigrupo S. Usaremos la notación $\lambda_i = \lambda(i)$

La aplicación anterior es claramente una biyección, y es creciente ($s_{i+1} = \lambda(i+1) > \lambda(i) = s_i = s$). Es por tanto la única aplicación con estas dos propiedades.

Definición 3.1.2: Definimos la operación $\oplus_S : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$, asociada al semigrupo S, como:

$$i \oplus_S j = \lambda^{-1}(\lambda_i + \lambda_j)$$

Es decir, en la posición del elemento suma. Para cualquier $i, j \in \mathbb{N}_0$. λ^{-1} es la inversa de la enumeración de S. Si $\lambda_k - \lambda_i = \lambda_j \in \Lambda$, entonces decimos que $k \ominus_S i = \lambda^{-1}(\lambda_k - \lambda_j) = j$

La operación $\oplus$ no estaba implementada en la librería "NumericalSpgs", por lo que esta es una de las primeras tareas a realizar. Notemos que, la función "*NumberElement_NumericalSemigroup*", nos permite calcular λ^{-1}.

```
Oplus := function(i,j,S)
  local s, k;
  s:=S[i]+S[j];
  k:=NumberElement_NumericalSemigroup(S,s);
  return k;
end;
```

Veamos un ejemplo de su uso.

Ejemplo 3.1.3: Consideremos el semigrupo $\langle 3,5\rangle = \{0,3,5,6,8,\rightarrow\}$ y su correspondiente enumeración, λ. Entonces $1\oplus 2 = \lambda^{-1}(3+5) = \lambda^{-1}(8) = 5$ y $3\oplus 2 = \lambda^{-1}(6+5) = \lambda^{-1}(11) = 8$. Veamos un ejemplo usando GAP (donde los índices comienzan en 1), aquí uso la función que he programado:

```
gap> S:=NumericalSemigroup(3,5);;
gap> Oplus(2,3,S);
5
gap> S[2]; S[3]; S[5];
3
5
8
```

La operación $\oplus$ es conmutativa y asociativa, pues $i\oplus j = \lambda^{-1}(\lambda_i+\lambda_j) = \lambda^{-1}(\lambda_j+\lambda_i) = j\oplus i$ (y similarmente para la asociatividad). El cero es su elemento unidad, $0\oplus i = \lambda^{-1}(\lambda_0+\lambda_i) = \lambda^{-1}(0+\lambda_j) = j$. Sin embargo, por lo general no hay inverso (si $\lambda_k - \lambda_i \notin S$ entonces $k \ominus_S i$ no está definida).

La operación también es compatible con la relación de orden del semigrupo. Es decir, si $a < b,\ a,b \in \mathbb{N}_0$, entonces $a\oplus c < b\oplus c$. Efectivamente $b\oplus c - a\oplus c = \lambda^{-1}(\lambda_c+\lambda_b) - \lambda^{-1}(\lambda_c+\lambda_a) > 0$, pues la enumeración es monótona creciente, luego también lo es la inversa, y $\lambda_b > \lambda_a$.

Proposición 3.1.4: La $\oplus$ operación determina el semigrupo de forma unívoca.

Efectivamente, basándonos en el teorema anterior, podemos calcular el semigrupo a partir da la función $\oplus$. Conocer la operación supone, o bien tener una regla, o conocer los valores de las ternas $(i,j,i\oplus j)$. El segundo caso es en el cual nos hemos basado para implementar en GAP esta caracterización.

```
SemigroupFromOplus := function(O)
```

```
  local S, nu;
  nu:=NuFromOplus(O);
  S:=SemigroupFromNu(nu);
  return S;
end;
```

Nos basamos para realizar el calculo en otra caracterización alternativa, que introduciremos un poco más adelante, basada en la sucesión ν. La entrada en este caso, O, es un conjunto de ternas, que se asume suficiente como para poder recrear el semigrupo completo. De hecho, en [2] (lema 50), se demuestra que es posible construir dos semigrupos numéricos para los cuales, $\oplus$, coincide para un cierto conjunto de valores. Afortunadamente, en la práctica, basta con una cantidad finita de ternas para determinar unívocamente el semigrupo.

Ejemplo 3.1.5 Vamos a ver un ejemplo, aplicando la función anterior. Usamos un pequeño programa, "*createO*", para generar las ternas de la operación $\oplus$, asociada a al semigrupo s_1. Vemos que el semigrupo obtenido por

```
gap> s1:=NumericalSemigroup(2,7,11);;
O:=createO(s1,11,11);
[ [ 1, 1, 1 ], [ 1, 2, 2 ], [ 1, 3, 3 ], [ 1, 4, 4 ], [ 1, 5, 5 ],
  [ 1, 6, 6 ],...,[ 20, 17, 39 ], [ 20, 18, 40 ], [ 20, 19, 41 ],
  [ 20, 20, 42 ] ]
gap> s2:=SemigroupFromOplus(O);
<Numerical semigroup>
gap> s2=s1;
true
```

3.2. La sucesión ν

Discutimos ahora la secuencias ν, una sucesión estrechamente relacionada con las distancias de Feng-Rao que discutiremos en el siguiente capítulo.

Definición 3.2.6: Sea S un semigrupo numérico y $\oplus_S$ la suma de índices asociada a S. Entonces definimos:

- El orden parcial en los naturales asociado a S, $(\mathbb{N}_0, \succeq_S)$ de forma que:

$$j \succeq_S i \Leftrightarrow \lambda_j - \lambda_i \in S$$

  O, equivalentemente, existe $k \in \mathbb{N}_0$, tal que $i \oplus_S k = j$.

- Dado el conjunto $D(\lambda_i) = \{j \in \mathbb{N}_0 \mid i \succeq j\} = \{j \in \mathbb{N}_0 \mid \lambda_i - \lambda_j \in S\}$, definido en la sección anterior, al cardinal de dicho conjunto, lo denotaremos por $\nu_i = |D(\lambda_i)|$.
- Denotaremos ν a la sucesión $\{\nu_i\}_{i=0}^{\infty}$

Ejemplo 3.2.7: Para el caso trivial $S = \mathbb{N}_0$, tenemos que $D(\lambda_i) = \{j \in \mathbb{N}_0 \mid i - j \in \mathbb{N}_0\} = \{j \in \mathbb{N}_0 \mid j \leq i\} \Rightarrow \nu_i = 1 + i$. Luego $\nu = 1, 2, 3, \ldots$

Podemos, fácilmente, implementar una función que calcule ν_i, así como toda la secuencia. Puesto que, a partir de cierto punto dicha secuencia es trivial, basta con una conjunto finito inicial.

```
Nu := function(i,S)
  local d;
  d:=DivisorsOfElementInNumericalSemigroup(S[i],S);
  if d=[] then
     return 1;
  else
    return Length(d);
  fi;
end;
```

Y para la la secesión, nos restringimos a $\{\nu_i\}_{i=1}^{2c-g}$. Para $i \geq 2c - g$, la sucesión verifica que $\nu_{i+1} = \nu_i + 1$.

```
NuSequence:=S->List([1..2*Conductor(S)-Genus(S)],
    i->Nu(i,S));
```

Usando la función que he implementado en GAP, podemos calcular la sucesión ν de un semigrupo numérico. Esta función nos da la primera parte de la sucesión, pues a partir de un número crece en incrementos de 1 [2].

```
gap> S:=NumericalSemigroup(3,5);
<Numerical semigroup with 2 generators>
gap> NuSequence(S);
[ 1, 2, 2, 3, 4, 4, 3, 6, 5, 6, 8, 8 ]
```

Proposición 3.2.8: Sea S un semigrupo numérico, y ν su correspondiente sucesión ν. Tenemos que para todo $i \in \mathbb{N}_0$:

$$\nu_i = |\{(j,k) \in \mathbb{N}_0^2 \mid j \oplus k = i\}|$$

Proposición 3.2.9 La ν secuencia determina unívocamente el semigrupo numérico.

La demostración del teorema anterior es constructiva, y puede encontrarse en [2] (Teorema 54). En dicha demostración nos hemos basado para la siguiente caracterización, que que nos permite construir el semigrupo a partir de la sucesión ν:

```
#Given a Nu sequence, compute the semigroup asociated to it.
SemigroupFromNu := function(NuSeq)
  local isNu, i, l, S, g, c, k, G, DTilde;
  #Some checks on sequence.
  #This are only necesary Conditions for a Nu sequence
  isNu:=true;
  l:=Size(NuSeq);
  if l>0 then
    if not(NuSeq[1]=1) then isNu:=false; fi;
    if l>1 then
      if not(NuSeq[2]=2) then isNu:=false; fi;
    else #If nu sequece is [1], then the semigroup is N
      return NumericalSemigroup(1);
    fi;
    for i in [1..l] do
      if not(NuSeq[i]<=i) then isNu:=false; fi;
    od;
  fi;
  if not isNu then
    Error("Not a valid Nu sequence.");
    return fail;
  else
  #We compute numerical semigroup from NuSeq.
  #This Semigroup may not unique if nu sequence is not complete.
    S:=[];
    #We first need to compute k, greatest i such that nu_i=nu_{i+1}
    k:=1;
    for i in [1..(l-1)] do
      if NuSeq[i]=NuSeq[i+1] then
        k:=i;
      fi;
    od;
    #With k, we can calculate both Genus and Conductor
    g:=k+1-NuSeq[k];
```

```
    c:=(k+g+1)/2;

    #We keep track of gaps.
    #Note that 1 is gap,
    #since the trivial semigroup was already conseidered.
    G:=[1,c-1];

    #This auxiliar function computes the number of gaps, l,
    #i<l<c-1 such that c-1+i-l is also a Gap
    DTilde:=function(i,c,G)
      local l, D;
      D:=0;
      for l in [(i+1)..(c-3)] do
        if l in G then
          if (c+i-l-1) in G then D:=D+1; fi;
        fi;
      od;
      return D;
    end;

    for i in Reversed([2..(c-1)]) do
      if NuSeq[c+i-g]=(c+i-2*g+DTilde(i,c,G)) then
        Add(S,i,1);
      else
        Add(G,i);
      fi;
    od;
    #Now, we determine small elements of the semigroup.
    Add(S,0,1); # O is always n the semigroup.
    Add(S,c); # Conductor is always n the semigroup.
    return NumericalSemigroupBySmallElements(S);
  fi;
end;
```

Por tanto, es posible, conocida la operación $\oplus$, determinar la sucesión ν,

Ejemplo 3.2.10:

```
s1:=NumericalSemigroup(2,7);
nu:=NuSequence(s1);
```

```
[ 1, 2, 3, 4, 2, 5, 4, 6, 6 ]
gap> s2:=SemigroupFromNu(nu);
<Numerical semigroup>
gap> s2=s1;
true
```

Proposición 3.2.11: La operación $\oplus$ determina de forma única el semigrupo.

La proposición anterior puede parecer obvia, puesto que tanto ν como $\oplus$ determinan el semigrupo, pero podemos dar un algoritmo, pasando de $\oplus$ a ν directamente.

```
NuFromOplus := function(O)
  local i, Nu, r, nu, j, maxj, maxi;
  Nu:=[];
  r:=Length(O);
  #We want the maximun i, and j
  SortBy(O, x -> x[1]);
  maxi:=O[r][1];
  SortBy(O, x -> x[2]);
  maxj:=O[r][2];
  #First, sort by k
  SortBy(O, x -> x[3]);
  j:=1;
  nu:=1;
  for i in [2..r] do
    #If this first condition is met,
    #more information is needed to compute nu[j]
    if (maxi < O[i][3]) or (maxj < O[i][3]) then break; fi;
    if O[i-1][3]=O[i][3] then
      nu:=nu+1;
    else
      Nu[j]:=nu;
      j:=j+1;
      nu:=1;
    fi;
  od;

  return Nu;
end;}
```

Es importante resaltar que, al igual que sucedía con la operación $\oplus$, si la sucesión está incompleta, el semigrupo puede no quedar unívocamente determinado. Sin embargo, como hemos dicho, es suficiente con conocer la sucesión hasta $2c - g$.

Ejemplo 3.2.12 Veamos un ejemplo de esta última función

```
s:=NumericalSemigroup(2,7);;
O:=createO(s,11,11);;
gap> NuFromOplus(O);
[ 1, 2, 3, 4, 2, 5, 4, 6, 6, 7 ]
NuSequence(s);
[ 1, 2, 3, 4, 2, 5, 4, 6, 6 ]
```

3.3. Sucesión τ

Finalmente, introducimos la secuencias τ.

Definición 3.3.13: Sea S un semigrupo numérico. Definimos su secuencia τ como:

$$\tau_i = max\{j \in \mathbb{N}_0 \mid \text{existe } k, \text{ con } j \leq k \leq i \text{ y } j \oplus_S k = i\}.$$

La primera labor de implementación, es el cálculo de los elementos de la sucesión,

```
TauSemigroup := function(i,S)
  local d, D, j, tau;
  D:=DivisorsOfElementInNumericalSemigroup(S[i],S);
  if D=[] then return 0;
  else
    for d in D do
      if d<=(S[i]/2) then
        tau:=NumberElement_NumericalSemigroup(S,d)-1;
      fi;
    od;
    return tau;
  fi;
end;
```

Y con lo anterior, podemos calcular la sucesión τ. Al igual que con la sucesión ν, a partir de cierto punto la sucesión es predecible, por lo que podemos restringirnos a valores. En concreto, si $i >= 2c - g + 1$, tenemos, para i para, que $\tau_i = tau_{i-1} + 1$; y para i impar, que $\tau_i = \tau_{i-1}$.

```
NuSequence:=S->List([1..2*Conductor(S)-Genus(S)],
```

```
    i->Nu(i,S));
```

Notemos que si τ_i es el mayor elemento, j, en N_i, con $\lambda_j \leq \lambda_i$. En concreto, si $\lambda_i/2 \in S$, entonces $\tau_i = \lambda^{-1}(\lambda_i/2)$. Notemos que, $\tau_i = 0$ si y solo si $\lambda_i = 0$ o un generador de S.

Ejemplo 3.3.14 Veamos un ejemplo para calcular la sucesión τ, y el cálculo del semigrupo a partir de dicha sucesión:

```
tau:=TauSequence(s1);
[ 0, 0, 1, 1, 0, 2, 1, 2, 2, 3 ]
gap> s2:=SemigroupFromTau(tau);
<Numerical semigroup>
gap> s2=s1;
true
```

Proposición 3.3.15: La secuencia τ determina de forma univoca el semigrupo.

Obtenermos así una forma adicional de determinar un semigrupo numérico. Podemos encontrar una demostración, constructiva del teorema en [2], teorema 60. En dicha demostración nos basaremos para implementar la siguiente función:

```
#Given a Tau sequence,
#it gives outputs the coresponding numerical semigroup.
#In orther for this to be correct,
#the Tau sequece must contain at least 2c-g+1 items
SemigroupFromTau := function(TauSeq)
  local i, j, k, l, FoundNewMin, g, c, S, aux, min;
  #The first step is to compute the minumun integer,
  # k, such that for all i,
  #Tau_{k+2i}=Tau_{k+2i+1} and Tau_{k+2i+2}=Tau_{k+2i+1}+1.
  l:=Length(TauSeq);
  k:=l-1;
  for j in Reversed([1..l]) do
    FoundNewMin:=true;
    for i in [0..Int((l-j)/2-2)] do

      FoundNewMin:=(TauSeq[j+2*i]=TauSeq[j+2*i+1]) and
        (TauSeq[j+2*i+1]+1=TauSeq[j+2*i+2]) ;

      if not FoundNewMin then break; fi; #Break out of the inner loop
    od;
```

```
    if FoundNewMin then k:=j; fi;
  od;
  #With this, we have the conductor and genus
  c:=k-TauSeq[k];
  g:=k-2*TauSeq[k]-1;

  #Now, we compute the semigrup.
  #We need initialize it to a list of lenght l
  S:=[];
  #The first values don't matter now, but will matter later
  for i in [1..(c-g)] do Add(S,0); od;
  #The values after the conductor are tivial
  for i in [(c-g)..l] do Add(S,i+g); od;

  for i in Reversed([2..c-g]) do
    #Follow the procedure, as outlined in the proof by Maria Brass
    aux:=Positions(TauSeq,i-1);
    min:=1;
    for j in [1..Length(aux)] do
        if S[aux[min]]>(S[aux[j]]) then
          min:=j;
        fi;
    od;
    S[i]:=S[aux[min]]/2;
  od;
  return NumericalSemigroupBySmallElements(S{[1..(c-g+1)]});
end;
```

Reiterando lo que hemos comentado en las secciones anteriores, las sucesiones τ de dos semigrupo numéricos pueden coincidir hasta un cierto número para dos semigrupos numéricos. Pero nos basta con conocer los primeros $2c - g$ elementos para poderlo determinar de forma única.

Capítulo 4

Distancias de Feng-Rao

En este capítulo nos centramos en un objeto particular, las distancias de Feng-Rao, relacionado con los semigrupos numéricos. El cálculo de estás distancias es una de las partes más importantes del trabajo. Recordemos que, anteriormente habíamos definido el conjunto de divisores, $D(x)$. Dado $S = \{\rho_1 = 0, \rho_2, \ldots\}$ un semigrupo numérico, y $x \in S$, definimos $D(x) = \{\lambda \in S \mid x - \lambda \in S\}$. Así mismo, definimos la secuencia $\nu = \{\nu_2 = |D(\rho_1)|,\ |\nu_2 = D(\rho_2)|,\ \ldots\}$ (en este caso, para trabajar en GAP, empezaremos los índices en 1). Usando esta noción, podemos definir la distancia de Feng-Rao clásica como:

Definición 4.0.1: Dado S, un semigrupo numérico, y m un elemento de dicho semigrupo, definimos la distancia de **Feng-Rao** clásica como:

$$\delta_{FR}(m) := min\{\nu_i \in \nu \mid \lambda_i \geq m\}$$

Es posible definirlo con una desigualdad, en cuyo caso, se obtienen resultados un poco diferentes.

El concepto es generalizable, pues se pueden definir conjuntos de divisores de varios elementos:

Definición 4.0.2: Sea S un semigrupo numérico y $s_1, s_2, \ldots, s_r$ elementos de dicho semigrupo. Entonces definimos:

- El conjunto de divisores de estos elementos, $D(s_1, s_2, \ldots, s_n) = D(s_1) \cup D(s_2) \ldots \cup D(s_r)$.
- Denotamos $\nu_{s_1,s_2\ldots,s_r} = |D(s_1, s_2, \ldots, s_n)|$
- Con lo anterior, definimos la **distancia Feng-Rao generalizada:**

$$\delta^r_{FR}(m) = min\{\nu_{s_1,s_2\ldots,s_r} \mid m \leq s_1 < s_2 < \ldots < s_r\}$$

Notamos que, para $r = 1$, $\delta^{r=1}_{FR}(m) = \delta_{FR}(m)$.

4.1. Aplicaciones y motivación

En este capítulo vamos a describir como calcular la distancia de Feng-Rao, centrándonos en mejorar la función "`FengRaoDistance(s,r,m)`", implementada en "NumericalSpgs". Esta función calcula la distancia generalizada, calculando el mínimo de la definición 4.0.2. Calcular dicho mínimo se realiza de forma recursiva, y es un calculo lento, impracticable para $r >> 0$. Consideremos un ejemplo con GAP. En este caso, la segunda llamada no termina en minutos, y me veo forzado a interrumpirla:

```
s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);;
gap> FengRaoDistance(s,5,155);
106
gap> s:=NumericalSemigroupBySmallElements([0,7]);;
FengRaoDistance(s,20,200000);
^CError, user interrupt in
```

Afortunadamente, se conocen múltiples resultados, que permiten optimizar el cálculo considerablemente para ciertos rangos de valores del semigrupo y/o para ciertos tipos de semigrupos concretos.

Vamos a hablar brevemente de las aplicaciones de las distancias de Feng-Rao. En el trabajo de matemáticas, hablo en más detalle sobre estas aplicaciones [15].

4.1.1. Aplicaciones de la distancia clásica de Feng-Rao: Códigos Correctores

La primera distancia de Feng-Rao tiene aplicaciones en el contexto de los códigos correctores lineales, pues nos permite dar una cota inferior para la distancia mínima del código, y por tanto, acotar su capacidad correctora.
Los códigos correctores están relacionados con la teoría de información. Consideremos una comunicación entre dos actores (emisor y receptor), en la cual un mensaje es transmitido entre emisor y receptor a través de un canal con ruido (es decir, pueden producirse errores, que de forma aleatoria cambien un carácter o bit). Los códigos correctores permiten codificar el mensaje de tal forma que, el receptor pueda recuperar el mensaje original, incluso si se han producido errores. Esto se consigue introduciendo redundancia en mensaje, haciéndolo más largo, y permitiendo al receptor corregir el error por el contexto. Quizá el ejemplo más sencillo de un código corrector es el uso de un *bit de paridad*.

Ejemplo 4.1.1.1:

Este es un código sencillo, que permite detectar errores (si bien no corregirlos). Dado un

mensaje binario de longitud n, el emisor puede sumar en $\mathbb{F}_2$ todos los bits del mensaje y el resultado será 1 si hay un número impar de bits con valor "uno" y 0 si hay un número impar.

El emisor quiere mandar el mensaje binario $m \in \mathbb{F}_2^{n=8},\ m = \{1,0,1,1,1,0,0,0\}$, luego computa la suma de los dígitos (módulo 2):

$$\overline{1+0+1+1+1+0+0+0} = \overline{0}\ \ (mod\ \ 2)$$

Para el mensaje m decimos que el bit de paridad es 0. Al receptor le llegará un mensaje codificado con nueve bits: $c = \{1,0,1,1,1,0,0,0,0\}$ donde los ocho primeros corresponden al mensaje y el último es el bit de paridad. Si la suma módulo 2 de los 8 primeros bits no coincide con el bit de paridad sabrá que se ha producido al menos un error.

Los códigos lineales son un tipo de códdigo corrector, que nos interesa en este caso:

Definición 4.1.1.2: Un **código lineal**, $\mathcal{C}$, sobre $\mathbb{F}_q$ es un subespacio vectorial de $\mathbb{F}_q^n$.
Consideramos los siguientes parámetros, asociados a $\mathcal{C}$:

- **Longitud**: si el código es subespacio de $\mathbb{F}_q^n$, entonces llamaremos a n longitud de $\mathcal{C}$.
- k, la **dimensión** del código como espacio vectorial.

Diremos que $\mathcal{C}$ es de tipo $[n,k$. En base a lo anterior, damos las siguientes definiciones:

Definición 4.1.1.3: Dados $\mathbf{x},\ \mathbf{y} \in \mathbb{F}_q^n$, $\mathbf{x} = (x_1,\ldots,x_n)$ e $\mathbf{y} = (y_1,\ldots,y_n)$, la **distancia de Hamming** entre $\mathbf{x},\mathbf{y}$ es:

$$d(\mathbf{x},\ \mathbf{y}) = \left|\left\{i \mid x_i \neq y_i,\ i \in \{1,2,\ldots,n\}\right\}\right|$$

Definición 4.1.1.4: Dado un código lineal $\mathcal{C}$, llamaremos a d la **distancia mínima** de $\mathcal{C}$ a:

$$d = min\{d(\mathbf{x},\mathbf{y}) \mid \mathbf{x},\ \mathbf{y} \in \mathcal{C},\ \mathbf{x} \neq \mathbf{y}\}$$

La distancia mínima, es una de las ideas centrales de los códigos correctores lineales. Si d es la distancia mínima de un código $\mathcal{C}$, y conocemos el conjunto de todos los mensajes codificados, entonces tenemos, automáticamente, un simple "**algoritmo**" para decodificar cualquier código lineal (si bien uno no muy eficiente):

1. El receptor, al recibir un mensaje codificado por un código lineal, puede comparar el mensaje recibido con el conjunto de mensajes posibles (todo el espacio vectorial, $\mathcal{C}$). Si el mensaje recibido coincide con un elemento de $\mathcal{C}$, asumirá que no se ha producido ningún error.

2. Si no coincide, entonces el receptor computará la distancia entre el mensaje-código recibido y cada uno de los códigos de $\mathcal{C}$. Asumirá que el que tenga la menor distancia al código recibido será el mensaje original.

Este algoritmo no se puede usar en la práctica salvo para códigos con pocos elementos. Sin embargo, nos da una idea de cuál es la capacidad correctora de un código, pues está basado en la idea de se producen, en general, pocos errores. Entonces, si recibimos una palabra que no está en el código, asumimos que la palabra del código que más se le asemeje es la palabra original. Si el número de errores es pequeño, esto es cierto. Está idea intuitiva se formaliza por medio del concepto de **capacidad correctora**:

Usando el algoritmo anterior, podemos corregir $t = \lfloor \frac{d-1}{2} \rfloor$ errores:

Proposición 4.1.1.5: Un código lineal $\mathcal{C}$ cuya distancia mínima es d puede corregir $t = \lfloor \frac{d-1}{2} \rfloor$ errores. A este número lo llamaremos **capacidad correctora** del código y lo denotamos por t.

Por tanto, conocer la distancia mínima es muy útil cuando trabajamos con cualquier código corrector lineal, pues nos permite saber el máximo número de errores que se pueden corregir (aunque no todos los algoritmos pueden corregir t errores).

Así mismo, calcular la distancia mínima es en muchos casos, demasiado costoso. Si el código lineal no tiene alguna propiedad conocida que permita realizar el cálculo de forma eficiente, es necesario computar las distancias dos a dos de los elementos del código (de un espacio vectorial, que igual no es conocido por completo), y después el mínimo, un cálculo muy caro, computacionalmente.

Para el tipo de códigos correctores con los cuales he trabajado en el TFG de matemáticas; los códigos algebraico geométricos en un punto, en general no se conoce la distancia mínima. Estos códigos se construyen evaluando funciones racionales que presentan polos en un único punto en un conjunto de puntos de una curva algebraica, de forma que dichas evaluaciones constituyan un espacio vectorial. Uno de los temas centrales de ese trabajo es que es posible, a partir de una curva con esta propiedad, definir un semigrupo numérico asociado a la curva, el semigrupo de Weierstrass.

Denotamos por $\mathcal{C}_m$ a un código AG (clásico) en un punto P de la curva algebraica $\mathcal{X}$, con las propiedades especificadas (además de ser suave). Al semigrupo de Weierstrass asociado lo denotamos por S. Entonces, tenemos el siguiente resultado, demostrado en [16]:

Teorema 4.1.1.6: Sea $m \in \mathbb{N}_0$, y $\mathcal{C}_m$ un código AG en un punto. Sea g el género del

semigrupo de Weiestrass del semigrupo y $d(\mathcal{C}_m)$ la distancia mínima del código. Entonces se verifica que:

$$d(\mathcal{C}_m) \geq \delta_{FR}(m+1) \geq m - 2g + 2$$

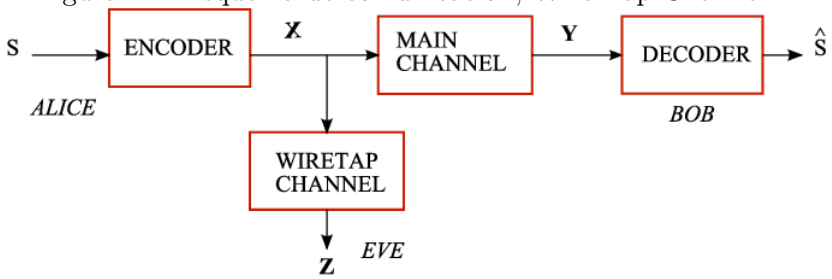
Por tanto, la distancia mínima puede ser acotada usando la distancia de Feng-Rao. Esta cota es mejor que la llamada cota de Goppa (la segunda desigualdad del teorema).

4.1.2. Aplicaciones de la distancia generalizada a la criptografía: Wire Tap Channel II

Vamos a introducir un nuevo problema de transmisión de información, introducido originalmente en [13], para el cual las distancias de Feng-Rao generalizadas son una herramienta útil. En este caso, se trata de un problema de criptografía, que pretende proteger un mensaje que se transmite a través de un canal al cual un espía tiene acceso. Se pretende resolver este problema sin el uso de una clave criptográfica. Este problema se puede resolver usando códigos correctores, y por tanto, con el beneficio añadido de corrección de errores por ruido. Este problema fue planteado en un artículo publicado por la empresa de telefonía americana *AT&T*. El nombre en inglés, *wire-tap channel* hace referencia a un canal de comunicación con múltiples cables (cable telefónico, o de fibra), en el cual un espía a "pinchado" μ cables (el nombre podría traducirse como canal pinchado).

Al mandar un mensaje, un espía puede obtener información parcial del mismo, pudiendo leer cualesquiera μ bits del mensaje. Como hemos dicho, se plantea codificar el mensaje sin usar una clave. Queremos elegir una decodificación de tal forma que, el espía tenga que tener acceso al mayor número posible de bits, μ, para poder deducir información útil sobre el mensaje original (forzando a este a pinchar más cables, lo cual es costoso y hace que sea más fácil descubrir la intrusión) Se puede usar un código lineal para este propósito, en cuyo caso, a partir del concepto de peso de Hamming generalizado, podemos dar cotas sobre la cantidad de información que, del mensaje original, el espía es capaz de deducir. El uso de un código corrector para este propósito permite proteger el mensaje frente a ruido y el espía a la vez.

Figura 4.1: Esquema de comunicación, Wire-Tap Channel II



Como hemos dicho, usamos códigos correctores lineales para la codificación. Vamos introducir un nuevo concepto asociado a los códigos lineales; el de pesos de Hamming generalizados [17], que nos permitirá realizar un análisis de la eficacia de los códigos lineales para evitar el acceso de información al espía. Primero, necesitamos definir el concepto de matriz de control.

Definición 4.1.2.8: Sea $\mathcal{C}$ un código lineal de tipo $[n, k]$, definido sobre el cuerpo $\mathbb{F}_q$.

- Diremos que $\mathcal{C}$ tiene **matriz generatriz** G si, dada f, la aplicación lineal asociada $f : \mathbb{F}_q^k \to \mathbb{F}_q^n$ se verifica que $\mathcal{C} = Img(f)$.
- Diremos que la matriz H es **matriz de control** del código $\mathcal{C}$ si para todo $\mathbf{x} \in \mathbb{F}_q^n$ se verifica que: $\mathbf{x} \in \mathcal{C} \iff H\mathbf{x}^t = \mathbf{0}$.

Definimos el peso de Hamming generalizado.

Definición 4.1.2.8: Sea $\mathcal{C}$ un código lineal de tipo $[n, k]$ definido sobre el cuerpo $\mathbb{F}_q$. Definimos:

- El **soporte del código** $\mathcal{C}$, denotado por $sop(C)$, como

$$sop(\mathcal{C}) = \{i \mid c_i = 0, \text{ para cierto } (c_1, \cdots, c_k) = \mathbf{c} \in \mathcal{C}\}.$$

- El **peso de Hamming generalizado** r-ésimo, del código lineal $\mathcal{C}$, se define como:

$$d_r(\mathcal{C}) = min\{|sop(D)| \;\big|\; \mathcal{D} \text{ es un subcódigo lineal de } \mathcal{C}, dim(\mathcal{D}) = r\}$$

Usando esta definición, y ciertas propiedades de los códigos de Hamming generalizados, que pueden encontrarse en el artículo de Wei [17], y en el trabajo de fin grado de matemáticas, podemos dar el siguiente resultado:

Teorema 4.1.2.9 Denotemos por Δ_μ a la información que puede obtener el espía (las posiciones del mensaje original que deduce), tenemos que, si la información se codifica usando un código lineal $\mathcal{C}$, con matriz de control H, y denotemos, para cierto μ, $\Delta = \Delta_\mu$. Entonces, tenemos la siguientes cotas:

$$d_{n-\mu-\Delta}(\mathcal{C}) \leq n - \mu < d_{n-\mu-\Delta+1}(\mathcal{C})$$

En vistas de este teorema, podemos, dado un nivel de información, Δ, que consideremos aceptable que el espía pueda deducir (potencialmente cero), y puesto que n es un valor fijo, nos permite acotar μ. Es decir, que podemos, usando cotas inferiores asegurar para un cierto nivel de redundancia, que el espía, aún teniendo acceso a μ bits del mensaje, no será capaz de deducir más información que Δ_μ.

Por tanto, hemos establecido la utilidad de poder calcular los pesos de Hammming generalizados. No obstante, el cálculo de estos es, en general, complicado. Podemos usar las distancias Feng-Rao (clásicas y generalizadas) para obtener cotas para los pesos de Hamming Generalizados.

Teorema 4.1.2.10: Sea $\mathcal{C}_s$ el código AG clásico en un punto, P, tenemos que $d_r(\mathcal{C}_s) \geq \delta_{FR}(r+s-1)$, para todo $r, s \in \mathbb{N}_{>0}$.

Teorema 4.1.2.11: Sea $\mathcal{C}_m$ un código AG (clásico) en un punto. Entonces:

$$d_r(\mathcal{C}_m) \geq \delta_{FR}^r(m)$$

Por tanto, las distancias de Feng-Rao nos permiten dar cotas inferiores a los pesos de Hamming generalizados, y podemos usarlas en el problema de wire-tap channel II para acotar μ, tal y como hemos explicado con el teorema 4.1.2.9.

4.2. Cálculo de las distancias de Feng-Rao

En esta sección vamos a discutir el cálculo de las distancias en el contexto del trabajo. Como hemos dicho, librería "NumericalSpgs" implementa la función "`FengRaoDistance(s,r,m)`", que calcula $\delta_{FR}^m(m)$ para el semigrupo s, pero podemos refinar esta función.

Existen numerosos resultados en la bibliografía que permiten calcular la distancia de Feng-Rao para ciertas familias de semigrupos de forma mucho más eficiente, así como ciertas mejoras generales que se podemos implementar, en esta sección del trabajo nos centramos en la implementación de dichas mejoras. En el siguiente capítulo, haremos pruebas, comparando los tiempos de ejecución de la función original con los de la versión implementada en este trabajo.

4.2.1. Mejoras para elementos $m \geq 2c-1$

Vamos a introducir un par de resultados que permiten calcular las distancias de Feng-Rao para valores del semigrupo para valores del semigrupo a partir de $2c-1$ (donde c es el conductor del semigrupo).

Teorema 4.2.1.12 Sea S un semigrupo numérico, y δ_{FR} la distancia de Feng-Rao clásica. Sean c y g el conductor y el género (respectivamente) del semigrupo. Entonces, para $m \in \mathbb{N}$, con $m \geq 2c-1$, tenemos que:

$$\delta_{FR}(m) = m + 1 - 2g$$

En el teorema 5.4.6 del TFG de matemáticas se puede encontrarse una demostración de este teorema.

Este teorema tiene un equivalente para la distancia generalizada, cuyo enunciado y demostración puede encontrarse en [3].

Teorema 4.2.1.13: Sea S un semigrupo numérico, con género g y conductor c. Sea también $r \geq 2$, entonces existe una constante $E_r = E(S, r)$, tal que para todo $m \geq 2c-1$ tenemos que:

$$\delta^r_{FR}(m) = m + 1 - 2g + E_r$$

La constante anterior, E_r, se denomina número de Feng-Rao:

Definición 4.2.1.14: Para $r \geq 2$, denominaremos número de Feng-Rao r-ésimo del semigrupo S a la constante $E_r(S, r)$, que hemos tratado en el teorema anterior.

El cálculo del número de Feng-Rao está implementado en "NumericalSpgs" por medio de la función "`FengRaoNumber(r,s)`". Así, podemos añadir al código de la función, para el calculo de la función que calcula la distancia de Feng-Rao el siguiente fragmento de código:

```
conductor := ConductorOfNumericalSemigroup(s);
if (m>=2*conductor-1) then
  genus:=GenusOfNumericalSemigroup(s);
  if r=1 then return m+1-2*genus;
  else return m+1-2*genus+FengRaoNumber(r,s);
  fi;
fi;
```

Aunque mediremos los tiempos de ejecucióne en el siguiente capítulo, este simple cambio hace que el ejemplo que vimos al principio del capítulo, donde había un error de memoría no de lugar a error:

```
gap> s:=NumericalSemigroupBySmallElements([0,7]);;
gap> FengRaoDistance2(s,20,200000);
200014
```

4.2.2. Mejoras para semigrupos de tipo Arf

Existe un tipo de semigrupo, denominado Arf, para el cual el calculo de las distancias de Feng-Rao se puede simplificar.

Definición 4.2.2.15: Sea $S = \{0 = \rho_1,\ \rho_2, \ldots\}$ un semigrupo numérico. Decimos que S es de tipo **Arf** si para todo $i, j, k \in \mathbb{N}$ tenemos que:

$$\rho_i + \rho_j - \rho_k \in S$$

En GAP, con la librería "NumericalSpgs", podemos usar "`IsArf`" para verificar si el semigrupo es de este tipo.

Ejemplo 4.2.2.16:

```
gap> s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);;
gap> IsArf(s);
true
gap> s:=NumericalSemigroup(3,5,7);;
gap> IsArf(s);
true
gap> s:=NumericalSemigroup(17,19,21);;
gap> IsArf(s);
false
```

Para las distancias clásicas, tenemos el siguiente resultado, dado en [4].

Teorema 4.2.2.17 Sea $S = \{0 = \rho_1,\ \rho_2, \ldots\}$ un semigrupo Arf, de género g y conductor $c = \rho_r$. Definimos:

$$\ell_i = \begin{cases} r + \rho_{i+1} - 2, \text{ para } i \in \{1, \ldots, r-1\} \\ 0, \text{ si } i = 0. \end{cases} .$$

Entonces, para $m \in S \subseteq \mathbb{N}$, la distancia de Feng-Rao de m se puede calcular del siguiente modo.

- Si $\ell_{i-1} < m \leq \ell_i \leq \ell_{r-1}$, entonces $\delta_{FR}(m) = 2i$.
- Si $c + r - 2 = \ell_{r-1} \geq m$, entonces $\delta_{FR}(m) = m + 1 - g$.

Este resultado no permite implementar una función para calcular la distancia clásica para semigrupos de tipo Arf:

```
FengRaoDistanceArf := function(S,m)
local c, r, i, l1, l2, n;
n:=NumberElement_NumericalSemigroup(S,m)-1;
c:=Conductor(S);
r:=NumberElement_NumericalSemigroup(S,c);
#if m>2*c-1 then return m+1-2*Genus(S); fi;
if n=0 then return 1; fi;
if ( (c+r-2) <= n) then
  return n-Genus(S)+1;
else
  #i=1
```

```
    l1:=0;
    l2:=r+S[2]-2;
    if (l1<n) and (n<=l2) then
      return 2;
    else
      for i in [2..(r-1)] do
        l1:=l2;
        l2:=r+S[i+1]-2;
        if (l1<n) and (n<=l2) then
          return 2*i;
        fi;
      od;
    fi;
  fi;
end;
```

Veamos un ejemplo del cálculo de la distancia clásica con la función anterior: Veamos un pequeño ejemplo de ejecución:

```
gap> s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);;
gap> IsArf(s);
true
gap> FengRaoDistanceArf(s,67);
6
gap> FengRaoDistance(s,1,67);
6
```

El resultado anterior se puede extender para distancias generalizadas, para el caso $r = 2$. Puede encontrarse información más detallada en los teoremas 29 y 32 de [8]. Veamos dos resultados, el primero de los cuales se aplica al cálculo de la segunda distancia de Feng-Rao para semigrupos Arf para valores pequeños.

Definición 4.2.2.19: Dado un semigrupo numérico S con el siguiente sistema minimal de generadores: $\{n_1 < n_2 < \ldots < n_p\}$, definimos n_1 como la **multiplicidad de S**, que denotaremos por $\mathbf{e}(\mathbf{S})$.

Teorema 4.2.2.19: Sea $S = \{0 = \rho_1,\ \rho_2, \ldots\}$ un semigrupo Arf, con multiplicidad $e = \rho_2$ y conductor $c = \rho_r$. Entonces,

1. Si $e = 2$, entonces, para $m \in S$ con $2 \leq m \leq c + 1$, tenemos que

$$\delta_{FR}^2(m) = \begin{cases} 3, \text{ si } m = 2, \\ 4, \text{ si } m > 2. \end{cases}$$

2. Si $e > 2$, entonces tenemos las siguientes opciones:

   a) Para $m \in S$ con $e \leq m \leq c + e - 3$, tenemos $\delta^2_{FR}(m) = 3$.

   b) Si $\rho_3 = 2\rho_2$, entonces,

$$\delta^2_{FR}(c+e-2) = \begin{cases} 3, \text{ si } \rho_{r-1} < c-2, \\ 4, \text{ si } \rho_{r-1} = c-2. \end{cases}$$

y

$$\delta^2_{FR}(c+e-1) = 4.$$

   c) Si $\rho_3 < 2\rho_2$, entonces,

$$\delta^2_{FR}(c+e-2) = \begin{cases} 3, \text{ si } \rho_{r-1} < c-2, \\ 4, \text{ si } \rho_{r-1} = c-2 \text{ y } r = 3, \\ 5, \text{ si } \rho_{r-1} = c-2 \text{ y } r < 3, \end{cases}$$

y

$$\delta^2_{FR}(c+e-1) = \begin{cases} 4, \text{ si } r = 2, \\ 5, \text{ si } r > 2. \end{cases}$$

Este teorema nos permite calcular la segunda distancia de Feng-Rao para $m < c + e$. Para valores de $m \geq m + c$, tenemos el resultado siguiente:

Teorema 4.2.2.20 Sea $S = \{0 = \rho_1, \ \rho_2, \ldots\}$ un semigrupo Arf. Para $\overline{e} \in S \setminus \{0\}$, definimos el semigrupo $S_{\overline{e}} = \{0\} \cup (\overline{e} + S)$. Sean e y c la multiplicidad y el conductor (respectivamente) de S. Sea $\overline{c} = e + c$. Entonces, para todo $k \in \mathbb{N}$, tenemos que:

$$\delta^2_{FR}(S_{\overline{e}}, \overline{c} + \overline{e} + k) = \begin{cases} \delta^2_{FR}(S, c+k) + 2, \text{ si } e = \overline{e}, \text{ y } \delta^1_{FR}(S_e, c+e+k) = \delta^2_{FR}(S, c+k), \\ \delta^2_{FR}(S, c+k) + 3, \text{ en cualquier otro caso.} \end{cases}$$

Juntando los anteriores resultados, podemos crear una función que calcule la segunda distancia de Feng-Rao de semigrupos Arf:

```
#We compute the second Feng–Rao Distance for Arf semigroups
#S is a numerical semigroup (Arf) and n is an index (positive integer)
FengRaoDistanceArf2 := function(S,m)
  local l, mAux, mSeq, fr1, fr2, fr2Seq, r, e, eSeq, c, cSeq,
  k, kAux, kSeq, i,a, sSeq, sAux, rr, multSeq, L, FRD2Small;


  #Computes FengRaoDistance for small elements (m<c+e)
```

```
FRD2Small := function(S,m)

  local fr2, e, c, r;
  fr2:=0;
  e := Multiplicity(S);
  c := Conductor(S);
  r:=NumberElement_NumericalSemigroup(S,c);
  if m>=c+e then
    Error(``m must be less or equal to Conductor+Multiplicity'');
  fi;
  if e=S[2] and e>=2 then
    #First case in the therorem e=2 and m in [2,c+1]
    if e=2 then
      if (2<=m) and (m<=c+1) then
        if m=2 then
           fr2:=3;
        else
          if m>2 then
           fr2:=4;
          fi;
        fi;
      fi;
    else #e>2
      #Case 2, when e>2 and m in [e,c+e-3]
      if (e<=m) and (m<=c+e-3) then
        fr2:=3;
      else
      #Case 3 e>2 and m = c+e-2 or m=c+e-1
        if m=c+e-1 then    # Case m=c+e-1
          if S[3]=2*S[2] then #Case S[3]=2S[2]
             fr2:=4;
          else
            if S[3]<2*S[2] then #Case S[3]<2S[2]
              if r=2 then
                 fr2:=4;
              else # r>2
                 fr2:=5;
              fi;
```

```
            fi;
          fi;
        else    # Case m=c+e-2
          if S[3]=2*S[2] then #Case S[3]=2S[2]
            if S[r-1]<c-2 then
              fr2:=3;
            else
              if S[r-1]=c-2 then
                fr2:=4;
              fi;
            fi;
          else #Case S[3]<S[2]
            if S[r-1]<c-2 then
              fr2:=3;
            else
              if (S[r-1]=c-2) and (r=3) then
                fr2:=4;
              else
                if (S[r-1]=c-2) and (r>3) then
                  fr2:=5;
                fi;
              fi;
            fi;
          fi;
        fi;
      fi;
    fi;
    return fr2;
end;
#General Case computation, for 2c>m>=c+e
mSeq:=[];
e := Multiplicity(S);
c := Conductor(S);
k:=m-c-e;
r:=NumberElement_NumericalSemigroup(S,c);
#Check theorem hypothesis e=S[2].
#Case  m>c+e-1,given by theorem 32
```

```
if m>c+e-1 then
  multSeq := MultiplicitySequence(S);
  #Remove(multSeq,1);
  #Compute S1 sequence of semigroups:
  sAux:=NumericalSemigroup(1);
  sSeq:=[sAux];
  eSeq:=[1];
  cSeq:=[0];
  kSeq:=[m-c-e];
  mSeq:=[m];

  i:=1;
  #STEP 1: Compute semigrup sequence,
  #as well as multiplicity and conductor sequence
  for a in Reversed(multSeq) do
    L:=[0];
    sAux:=a+sAux;
    Append(L,SmallElements(sAux));
    sAux:=NumericalSemigroupBySmallElements(L);

    Add(sSeq,sAux);
    Add(eSeq,Multiplicity(sAux));
    Add(cSeq,Conductor(sAux));
    i:=i+1;
  od;

  #STEP 2: Computing k and m sequences:
  l:=Length(eSeq);
  i:=l;

  while i>1 do
    kAux:=mSeq[1]-cSeq[i]-eSeq[i];
    Add(kSeq,kAux,1);
    mAux:=cSeq[i-1]+kAux;
    Add(mSeq,mAux,1);

    i:=i-1;
  od;
```

```
  #STEP 3: Computin second Feng-Rao number
      # a is the position of the first element of mSeq
      # and kSeq with regard to
      # eSeq and cSeq
    l:=Length(sSeq);
    fr2:=mSeq[1]+2;
    for i in [1..l] do

      if (cSeq[i] <= mSeq[i]) and (mSeq[i] < cSeq[i]+eSeq[i]) then
        #Theorem 29 computation
        fr2:=FRD2Small(sSeq[i],mSeq[i]);

      else #Theorem 32 computation
          if i>1 and kSeq[i]>=0 then
            fr1:=FengRaoDistanceArf(sSeq[i-1],mSeq[i-1]+eSeq[i-1]);
            if eSeq[i]=eSeq[i-1] and fr2=fr1 then
                fr2:=fr2+2;
            else
                fr2:=fr2+3;
            fi;
          fi;

      fi;
    od;
  else
    return FRD2Small(S,m);
  fi;

  return fr2;
end;
```

Veamos un pequeño ejemplo de ejecución:

```
gap> s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);;
gap> IsArf(s);
true
gap> FengRaoDistance(s,2,75);
13
gap> FengRaoDistanceArf2(s,75);
```

13

4.2.3. Mejoras para semigrupos ordinarios

Los semigrupos ordinario son un tipo de semigrupos Arf, y podemos ver si pertenecen a esta familia con `IsOrdinary(s)`.

Definición 4.2.3.21 Un semigrupo numérico S, con multiplicidad e y conductor c es **ordinarios** si verifica que $e = c$.

Ejemplo 4.2.3.22: Vemos que el semigrupo $\{0, 4, \rightarrow\}$ es ordinario:

```
gap> s:=NumericalSemigroupBySmallElements([0,4]);;
gap> IsOrdinary(s);
true
```

Para este tipo de semigrupo, el siguiente resultado nos permite calcular la distancia segunda de Feng-Rao.

Proposición 4.2.3.23: Sea S un semigrupo ordinario con conductor c y multiplicidad $e = c$. Entonces, para $m \in S$, tenemos que:

$$\delta^2_{FR}(m) = \begin{cases} 3, & \text{si } c \leq m < 2e-1, \\ m + (2e-1) + 4, & \text{si } 2e-1 \leq m. \end{cases}$$

La anterior formula permite un simple cáclulo en GAP:

```
FengRaoDistanceOrdinary2 := function(s,m)
 local e, c;
 e:=Multiplicity(s);
 c:=Conductor(s);
 if c<=m and m<2*e-1 then return 3; fi;
 return m - (2*e - 1) + 4;
end;
```

Veamos un ejemplo ejemplo para el semigrupo $S := \{0, 7, \rightarrow\}$

```
gap> s:=NumericalSemigroupBySmallElements([0,7]);;
gap> IsOrdinary(s);
true
gap> FengRaoDistance(s,2,12);
3
gap> FengRaoDistanceOrdinary2(s,12);
```

```
3
gap> FengRaoDistanceOrdinary2(s,17);
8
gap> FengRaoDistance(s,2,17);
8
```

4.2.4. Mejoras para semigrupos con dos generadores

En este caso, presentamos una fórmula para el cálculo de la distancia clásica para semigrupos generados por dos elementos, $S = \langle a, b\rangle$ con $m.c.d(a, b) = 1$. Puede parece que es una situación muy concreta, pero para es un caso muy común para semigrupos de Weiestrass, asociados a una curva algebraica.

Tenemos el siguiente resultado, basado en el artículo [12] (página 13) y [16] (teorema 5.5).

Proposición 4.2.4.24: Sea S un semigrupo generado por dos elementos, primos entre si. Si c es el conductor del semigrupo, y $m \in \mathbb{N}$, con $m > c$, tenemos que:

$$\delta_{FR}(m) = min\{\rho_k \mid \rho_k \geq m + 2 - 2g\}$$

En base este resultado, damos la siguiente función, para calcular la distancia clásica de Feng-Rao para este tipo de semigrupos:

```
FengRaoDistanceTwoGenerators := function(S,m)
  local conductor, genus, ro, p, min, L;
  conductor:=Conductor(S);
  genus:=Genus(S);
  L:=ElementsUpTo(S,2*conductor);
  min:=4*conductor;
  for ro in L do
    if ro >= m+1-2*genus and min>(ro+m+1-2*genus) then
      min:=ro;
    fi;
  od;
  return min;
end;
```

Veamos un ejemplo para un semigrupo con dos generadores:

```
gap> s1:=NumericalSemigroup(17,19);
<Numerical semigroup with 2 generators>
gap> FengRaoDistanceTwoGenerators(s1,289);
```

```
17
gap> FengRaoDistance(s1,1,289);
17
```

4.2.5. Mejoras para semigrupos de tipo simétrico

La segunda familia de semigrupos para la cual realizamos mejoras, es para los llamados semigrupos simétricos. Como dijimos en la definición 2,9, los semigrupos simétricos son aquellos que verifican que $2g = c$. La propiedad de simetría se puede verificar en "NumericalSpgs" usando "`IsSymmetric(s)`"

Estudiamos primero el caso de las distancias clásicas. Los siguientes resultados estan basados en la sección 4 del artículo [7].

Teorema 4.2.5.25: Sea S un semigrupo simétrico, con conductor c y multiplicidad e. Entonces, si $m = 2g - 1 + e$, con $e \in S \setminus \{0\}$, tenemos que:

$$\delta_{FR}(m) = m + 1 - 2g$$

El teorema anterior es aplicable a aproximadamente la mitad de los elementos del intervalo $[c-1, 2c-2]$, pero se puede extender para el resto de elementos del intervalo:
Teorema 4.2.5.26: Sea S un semigrupo simétrico con conductor c, y sea $m \in [c, 2c-2]$. Sea $n = m + c - 1$, y $\nu(x)$ el elemento $\lambda^{-1}(x)$-ésimo de la secuencia ν (definición 3.2.6). Si n' es el menor entero de S tal que $n' > n$ y $\delta = n' - n$, tenemos que:

$$\delta_{FR}(m) = min\{\nu(m),\ \nu(m+1), \ldots,\ \nu(m+\delta-3), n'\}$$

Combinamos los teoremas anteriores en una única función para calcular la distancia clásica de semigrupos simétricos:

```
FengRaoDistanceSymmetric := function(S,m)
  local g, c, n, nprima, delta, L, i;
  g:=Genus(S);
  c:=Conductor(S);
  #This algorithm is meant for m in [c,2c-1]
  #Simple case, given by theorem 4.6:
  if (m-2*g+1) in S then
    return m+1-2*g;
  fi;
  #From theorem 4.7:
  if (c<=m)  then
```

```
    n:=m-c+1;
    nprima:=n;
    while not(nprima in S) do
      nprima:=nprima+1;
    od;
    L:=[nprima];
    delta:=nprima-n;
    for i in [(m)..(m+delta-3)] do
      Add(L,Nu(NumberElement_NumericalSemigroup(S,i),S));
    od;
    return MinimumList(L);
  else
    return FengRaoDistanceBruteForce(S,m);
  fi;
  Error("Invalid Imput arguments");
end;
```

Vamos un ejemplo

```
gap> s1:=NumericalSemigroup(17,19);;
gap> IsSymmetric(s1);
true
gap> FengRaoDistanceSymmetric(s1,300);
17
gap> FengRaoDistance(s1,1,300);
17
```

Para las distancias generalizadas también es posible dar algunos resultados, sin embargo, no tan completos como en el caso de los teoremas anteriores (ver "Remark 10" de [3]), que permiten calcular la distancia de todos los elementos del intervalo $[c-1, 2c-2]$. No obstante, tenemos un resultado similar al del teorema 4.2.5.26, cuya demostración se encuentra en el teorema 9 de [3].

Teorema 4.2.5.27 Sea S un semigrupo simétrico de género g y $r \geq 2$. Sea $m = 2g - 1 + e$, para $e \in S$ y $e > 0$. Entonces, si E_r es el r-ésimo número de Feng-Rao (definición 4.2.1.14) tenemos que:

$$\delta_{FR}^{r}(m) = m + 1 - 2g + E_r$$

Esta condición nos permite calcular las distancias generalizadas de semigrupos simétricos para ciertos elementos del intervalo $[c-1, c-2]$.

```
FengRaoDistanceSymmetricGeneralized := function(s,r,m)
  if (m-2*g+1) in s then
```

```
    return m+1-2*g+FengRaoNumber(r,s);
  fi;
end
```

4.2.6. Integrando las mejoras en el código existente

Hemos implementado funciones que, en casos concretos permiten calcular la distancia de Feng-Rao (de forma más eficiente al caso general, como veremos en el siguiente capítulo). Ahora vamos a juntar todos los algoritmos en una única función, que en función del semigrupo que se pase como argumento, use un método u otro. Esta función, es una modificación de la función `FengRaoDistance` ya implementada.

Usamos una característica de GAP, que nos permite, de forma simple, saber si un cierto atributo ha sido establecido o no. Por ejemplo, para semigrupos de tipo Arf, podemos ejecutar `IsArf(S)` para verificar si el semirgupo S es de tipo Arf. Sin embargo, este calculo puede ser costoso para ciertas propiedades. En vez de esto, podemos usar `HasIsArf(S) and IsArf(S)`, que será cierto si se conoce que el semigrupo es Arf, o si de alguna propiedad que si que se conoce, se deduce que el semigrupo es Arf.

Ejemplo 4.2.6.28: Ejecutemos secuencialmente los siguientes comandos:

```
gap> s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);;
gap> HasIsArf(s);
false
gap> IsArf(s);
true
gap> HasIsArf(s);
true
```

Así, todos los algoritmos que hemos implementado serán funciones locales. El primer paso es comprobar si para el argumento m, se verifica que $m \geq 2c-1$, en cuyo caso, podemos usar los resultados de la sección 4.2.1.

```
FengRaoDistance2 := function(s,r,m)
  local conductor, genus, multiplicity, final, elementsUpToFinal,
  divisorsOfMany2, addOne2, posiblesOfLen2,
  FengRaoDistanceArf, FengRaoDistanceSymmetric,
        FengRaoDistanceBruteForce, FengRaoDistanceSymmetricGeneralized,
        FengRaoDistanceOrdinary2, FengRaoDistanceArf2, FRD2Small,
         FengRaoDistanceTwoGenerators;
```

```
  conductor := ConductorOfNumericalSemigroup(s);
  #Trivial case, when m>=2c-1
  if (m>=2*conductor-1) then
    genus:=GenusOfNumericalSemigroup(s);
    if r=1 then return m+1-2*genus;
    else return m+1-2*genus+FengRaoNumber(r,s);
    fi;
  fi;
...
```

Seguido del código anterior, están las funciones locales ya existentes, que calculan de forma recursiva la distancia de Feng-Rao, en base a la definición 4.0.2. A continuación, añadimos todos los algoritmos que hemos dado en las secciones 4.2-4.5 como funciones locales. Tras lo que realizamos las comprobaciones sobre los argumentos, para saber que función debemos llamar:

```
...
   # end of local functions
  # And here it is the Feng-Rao distance

  #Special cases

  #Classical (r=1)
  if r=1 then
    if HasIsArf(s) and IsArf(s) then
      return FengRaoDistanceArf(s,m);
    fi;

    if HasIsSymmetric(s) and IsSymmetric and m>=conductor then
      return FengRaoDistanceSymmetric(s,m);
    fi;

    if m>=conductor and Size(Generators(s))=2 then
      return FengRaoDistanceTwoGenerators(s,m);
    fi;

    return FengRaoDistanceBruteForce(s,m);

  fi;
```

```
#Generalized, r=2
if r=2 then

  if HasIsOrdinary(s) and IsOrdinary(s) and r=2 and m>=conductor then
    return FengRaoDistanceOrdinary2(s,m);
  fi;

  if HasIsArf(s) and IsArf(s) then
    return FengRaoDistanceArf2(s,m);
  fi;

fi;

#Generalized, symmetric:

if HasIsSymmetric(s) and ((m-2*Genus(s)+1) in s) then
  FengRaoDistanceSymmetricGeneralized(s,r,m);
fi;
```

Si ninguna de las funciones anteriores es aplicable a los argumentos de entrada, se ejecutará el código general, ya implementado.

En el capítulo siguiente haremos pruebas de forma extensiva, pero veamos un pequeño ejemplo, para ver que funciona:

```
gap> s:=NumericalSemigroup(3,17,51);;
gap> FengRaoDistance(s,1,300);
269
gap> FengRaoDistance2(s,1,300);
269
gap> FengRaoDistance2(s,2,171);
143
gap> FengRaoDistance(s,2,171);
143
```

Capítulo 5

Pruebas y eficiencia

En este capítulo vamos a realizar pruebas de forma extensiva. En la primera sección, nos centraremos en realizar un estudio de eficiencia para la nueva versión de la función `FengRaoDistance`, comparando con la versión existente en la versión 1.2.1 de "NumericalSpgs". En la segunda sección, haremos pruebas para comprobar que los resultados todas las funciones del proyecto son correctos, incluyendo el capítulo 2 (ideales) y el capítulo 3 (caracterizaciones de los semigrupos). El código para realizar los tests realizados pueden encontrarse en el repositorio del código [14], en el directorio tests.

5.1. Eficiencia en el cálculo de la distancia de Feng-Rao

El capítulo anterior está dedicado al cálculo de la distancias de Feng-Rao para ciertos casos especiales. En esta sección vamos a realizar un análisis, de forma experimental, en que comparemos los tiempos de ejecución de la función `FengRaoDistance2`, en la cual se incorporan todos las mejoras que hemos mencionado, frente a la función `FengRaoDistance`, la función que calcula la distancia de Feng-Rao tal y como está implementada en la versión 1.2.1 de "NumericalSpgs". La medición de los tiempo se ha realizado con la función `SpeedTestFengRao`, en el directorio de tests del repositorio de código.

```
SpeedTestFengRao := function(s,r,limInf,limSup,verbose,iter)
  local j, i, a, b, t1, t2, averageT, averageA, averageB;
  averageT:=[];
  averageA:=[];
  averageB:=[];
  for j in [1..iter] do
    Add(averageT,0);
    Add(averageA,0);
    Add(averageB,0);
    for i in [limInf..limSup] do
```

```
      if i in S then
        a:=NanosecondsSinceEpoch();
        FengRaoDistance(s,r,i);
        b:=NanosecondsSinceEpoch();
        t1:=b-a;
        a:=NanosecondsSinceEpoch();
        FengRaoDistance2(s,r,i);
        b:=NanosecondsSinceEpoch();
        t2:=b-a;
        if verbose then
          Print("\t");Print(i);Print("\t");
          Print(t1);Print("\t");Print(t2);
          Print("\t");Print("\t");Print(t1-t2);
          Print("\n");
        fi;
        averageA[j]:=averageA[j]+t1;
        averageB[j]:=averageB[j]+t2;
        averageT[j]:=averageT[j]+(t1-t2);
      fi;
    od;
    averageT[j]:=Float(averageT[j]/(limSup-limInf));
  od;
  averageA:=Float(Sum(averageA)/iter);
  averageB:=Float(Sum(averageB)/iter);
  averageT:=(Sum(averageT)/iter)/1000000;
  return [averageA/averageB,averageT];
end;
```

La función anterior calcula los valores de la distancia de Feng-Rao r-ésima para los elementos de $[limInf, limSup] \cap s$ usando `FengRaoDistance` y `FengRaoDistance2`. Mide los tiempos de ejecución y calcula la media aritmética para cada una de las funciones y devuelve la diferencia y el cociente de ambas. Se pueden realizar varias iteraciones, en cuyo caso devuelve la media de las iteraciones. En las pruebas, vamos usar valores de r pequeños. Esto es debido a las limitaciones de tiempo, pues los tests para valores grandes de r pueden tardar varias horas en ejecutarse.

5.1.1. Elementos $m \geq 2c - 1$

Vamos a comparar ambas funciones para elementos del semigrupo con $m \geq 2c - 1$. En este caso esperamos obtener mejoras substanciales, en particular para valores de $r = 1$ y otros

valores pequeños de r y para valores elevados de m, donde la diferencia entra las dos funciones será más notable. Generamos 10 semigrupos aleatoriamente, y para cada uno de ellos hacemos 3-5 iteraciones. Hagamos esto para $r = 1$, $r = 2$ y $r = 3$.

```
D:=[];;
L:=List([1..10],_->RandomNumericalSemigroup(5,10,100));;
gap> i:=1;;
gap> Add(D,
SpeedTestFengRao(L[i],1,2*Conductor(L[i]),3*Conductor(L[i]),false,5));
...
gap> i:=10;;
gap> Add(D,
SpeedTestFengRao(L[i],1,2*Conductor(L[i]),3*Conductor(L[i]),false,5));
```

Los resultados están en milisegundos y obtenemos la siguiente tabla (Valores mayores que 1 en la fila "T.Med.Co." y valores positivos en la fila "T. Dif. Med" corresponden a una mejora en el tiempo de ejecución). El siguiente caso es $\mathbf{r = 1}$.:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_9
Conductor	191	410	144	425	549	138	1944	258	1056	156
T. Med. Co.	125.38	517.7	140.2	645.2	661.7	119.1	1164.4	226.2	448.1	111.7
T. Dif. Med.	0.195	0.220	0.220	1.197	1.308	0.190	4.149	0.360	0.789	0.168

Vemos que, la mejora es considerable, entre 2 y 3 órdenes de magnitud, y cuanto mayor es el conductor, mas sustancial es la mejora. Vemos el caso para $\mathbf{r = 2}$..

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
Conductor	191	410	144	425	549	138	1944	258	1056	156
T. Med. Co.	5.34	18.23	7.551	21.02	18.24	6.52	17.93	9.027	6.42	5.61
T. Dif. Med.	3.18	5.1	5.1	107.0	102.8	3.84	342.1	11.52	20.56	3.21

Finalmente, vemos el caso con $\mathbf{r = 3}$..

	S_1	S_2	S_3	S_4	S_5	S_6
Conductor	191	410	144	425	549	138
T. Med. Co.	4.043	12.13	5.74	13.96	12.11	5.07
T. Dif. Med.	19.45	1393	43.69	2245.69	1866	29.09

Como vemos, para la distancia generalizada, la mejora no es tan marcada (pues la función `FengRaoNumber` es recursiva). Aún así, la mejora es considerable, al menos 5 veces más rápido que el método existente.

5.1.2. Semigrupos de tipo Arf

Usamos la función `ArfNumericalSemigroupsWithFrobeniusNumberUpTo(f)`, que genera todos los semigrupos Arf con número de Fröbenius menor o igual que 40, y seleccionamos 10 de ellos de forma aleatoria. Calculamos las distncias de Feng Rao en el intervalo $[2, 2c-1] \cap S$

```
gap> L1:=ArfNumericalSemigroupsWithFrobeniusNumberUpTo(40);;
gap> L1:=[L1[Random(1,n)],L1[Random(1,n)],L1[Random(1,n)],
L1[Random(1,n)],L1[Random(1,n)],L1[Random(1,n)],
L1[Random(1,n)],L1[Random(1,n)],L1[Random(1,n)],L1[Random(1,n)]];;
gap> for s in L1 do Print(IsArf(s)); od;
truetruetruetruetruetruetruetruetruetrue
for i in [1..10] do
  Add(D,SpeedTestFengRao(L1[i],1,2,2*Conductor(L1[i])-1,false,3));
od;
```

Ahora, analicemos, para el caso $\mathbf{r = 1}$, la velocidad de ejecución de ambas funciones.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	19	38	28	37	28	35	19	38	35	34
T. Med. Co.	5.79	6.28	11.68	1.419	11.24	14.40	4.37	20.49	29.72	9.84
T. Dif. Med.	0.048	0.036	0.036	0.015	0.032	0.042	0.012	0.063	0.093	0.0288

Y realizamos las mismas operaciones para $\mathbf{r = 2}$.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	19	38	28	37	28	35	19	38	35	34
T. Med. Co.	3.602	1.774	15.513	23.102	14.235	20.30	1.003	46.89	76.63	8.780
T. Dif. Med.	0.168	0.103	0.822	1.229	0.706	1.097	0.0004	2.349	2.594	0.534

Ambos métodos, que hemos implementado en el capítulo anterior para calcular las distancias de Feng Rao en semigrupo de tipo Arf son más rápidos que el algoritmo original. Si bien hay bastante variabilidad, en función del semigrupo, los resultados son claramente positivos.

5.1.3. Semigrupos de tipo ordinario

El siguiente caso a analizar es el de semigrupos ordinarios, con $c = e$. Consideramos 10 semigrupos de tipo $\{0, c, \rightarrow\}$. Ejecutamos el test de velocidad para elementos del semigrupo en el intervalo $[c, 2c-1]$. El método es para $\mathbf{r = 2}$

```
gap> for i in [1..10] do
  Add(L,NumericalSemigroupBySmallElements([0,Random(2,50)]));
```

```
od;
gap> for i in [1..10] do
  Print(IsOrdinary(L[i])); Print("\t");
od;
true true true true      true true true true true true
for i in [1..10] do
  Add(D, SpeedTestFengRao(L[i],2, Conductor(L[i]),
  2*Conductor(L[i])-1, false ,3));
od;
```

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	33	7	14	4	3	10	37	19	42	17
T. Med. Co.	231.82	13.09	50.38	5.76	4.39	31.77	286.6	89.47	369.43	70.49
T. Dif. Med.	6.019	0.28	1.03	0.119	0.084	0.638	7.751	1.869	10.48	1.492

Para este tipo de semigrupos, con una estructura clara para los cuales hay una formula sencilla, obtenemos resultados muy buenos.

5.1.4. Semigrupos de tipo simétrico

Vamos a analizar los resultados para semigrupos simétricos, con los dos métodos que hemos implementados. Para eso, generamos un conjunto de semigrupos simétricos de forma aleatoria.

```
gap> for i in [1..100] do
  Add(L, RandomNumericalSemigroup(5,10,50));
od;;
gap> for i in [1..100] do
  if(IsSymmetric(L[i])) then Add(L1,L[i]); fi;
od;;
L1:=L1{[1..10]};;
gap> for s in L1 do Print(IsSymmetric(s)); od;
truetruetruetruetruetruetruetruetruetrue
for i in [1..10] do
  Add(D, SpeedTestFengRao(L1[i],1, Conductor(L1[i]),
  2*Conductor(L1[i])-1, false ,3));
od;
```

Para estos semigrupos calculamos las distancias de Feng-Rao **clásicas** de los elementos del semigrupo en $[c, 2c-1]$, y observamos que este método es una mejora considerable frente al algoritmo existente.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	209	1871	1597	1121	935	2253	65	161	359	1409
T. Med. Co.	3.692	9.596	7.866	4.432	6.432	8.604	2.709	1.38	4.216	6.952
T. Dif. Med.	0.128	2.445	1.785	1.099	0.782	3.269	0.04	0.023	0.229	1.355

El método que hemos implementado para las distancias generalizadas en semigrupos simétricos no es válido para todos los elementos del intervalo $[c, 2c-1]$. Aún así, ejecutemos el test de velocidad para este intervalo con $\mathbf{r = 3}$. Puesto que los resultados del test de velocidad es la media de los tiempos de todos los elementos del intervalo, podríamos observar una cierta mejora en la media. En este caso nos restringimos a una iteración y semigrupos cuyo género es como mucho 200, pues el cálculo es demasiado lento.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	65	161	65	161	161	161	171	99	95	115
T. Med. Co.	0.73	0.69	0.68	0.69	0.65	0.69	0.77	0.79	0.74	0.73
T. Dif. Med.	-3.15	-4.00	-3.12	-4.15	-3.78	-11.76	-15.55	-6.64	-3.98	-6.96

Vemos que el método implementado se comporta peor que que original en este caso (Aproximadamente un 50 % más lento)

5.1.5. Semigrupos con dos generadores

El último de los casos especiales que consideramos es el caso de semigrupos generados por dos elementos y el cálculo de la distancias clásicas. Seguimos el siguiente procedimiento para generarlos:

```
gap> L:=List([1..10],_->RandomNumericalSemigroup(2,10,100));;
gap> for s in L do
  if(2=Length(Generators(s))) then
    Add(L1,s);
  fi;
od;
gap> L1:=L1{[1..10]};;
gap> Length(L1);
10
```

Veamos los resultados, con $r = 1$, primero para semigrupos con $\mathbf{c} \leq 150$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	137	17	125	131	103	125	131	149	89	139
T. Med. Co.	0.263	0.525	0.419	0.159	0.623	0.359	0.125	0.745	0.331	0.16
T. Dif. Med.	-0.16	-0.02	-0.10	-0.22	-0.05	-0.12	-0.25	-0.04	-0.11	-0.23

Y vemos que los resultados son peores que los de la función existente. Por tanto, esta función puede ser eliminada de la implementación final de la función en la librería "NumericalSpgs".

5.1.6. Comparando con semigrupos aleatorios

En los apartados anteriores hemos comparado los algoritmos con semigrupos e intervalos para los cuales están pensados. Ahora, vemos que sucede si seleccionamos semigrupos de forma aleatoria. En el peor de los casos, la función hace todas las comprobaciones añadidas sin que ninguna se cumpla. En dicho caso, se ejecuta el algoritmo general existente (Luego la nueva función será algo más lenta que la original).

```
L:=List([1..10],_->RandomNumericalSemigroup(5,2,50));;
gap> for i in [1..10] do Print(IsArf(L[i])); od;
falsefalsefalsefalsefalsefalsefalsefalsefalsefalse
gap> for i in [1..10] do Print(IsSymmetric(L[i])); od;
truefalsefalsefalsefalsefalsefalsetruefalse
gap> for i in [1..10] do
  Add(D,SpeedTestFengRao(L[i],1,Conductor(L[i]),
  3*Conductor(L[i]),false,1));
od;
```

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	72	76	180	28	123	128	118	90	102	74
T. Med. Co.	2.047	0.483	0.12	1.432	0.52	0.176	0.234	0.41	3.5	0.34
T. Dif. Med.	0.023	-0.123	-0.78	0.024	-0.215	-0.426	-0.36	-0.17	0.033	-0.15

Caso $\mathbf{r = 2}$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	72	76	180	28	123	128	118	90	102	74
T. Med. Co.	1.112	1.928	1.561	1.358	2.034	1.562	1.658	1.734	1.715	1.394
T. Dif. Med.	0.028	0.862	0.594	0.045	3.334	0.474	0.846	0.953	0.295	0.162

Caso $\mathbf{r = 3}$

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
N. Fröbenius	72	76	180	28	123	128	118	90	102	74
T. Med. Co.	1.008	1.704	1.384	0.77	1.863	1.364	1.429	1.608	1.129	1.188
T. Dif. Med.	0.005	5.116	2.345	-0.1	32.231	1.81	4.125	6.676	0.232	0.366

Los resultados varian. En el caso $r = 1$ son peores (salvo en los casos que el semigrupo es Arf o simétrico), pero las diferencias de tiempo no son pronunciadas (El caso $r = 1$ no es particularmente problemático). En los casos $r = 2$ y $r = 3$ observamos mejores tiempos, probablemente debido al cálculo en el intervalo $[2c - 1, 3c]$, en donde ya hemos visto que la función es más rápida.

5.2. Verificación de los resultados

Tal y como dijimos al principio del capítulo, verificamos que las pruebas realizadas son correctas.

5.2.1. Ideales

Vamos a verificar que los métodos para calcular las descomposiciones de Ideales dan lugar a descomposiciones correctas.

IdealDecomposition

Generamos un conjunto de ideales de forma aleatoria, y verificamos que la intersección de sus componentes irreducibles coincide con el ideal original.

```
TestIdealDecomposition := function()
 local l,aux,L1, I1,I2, k, i, j, a;
 l:=List([1..20],_->RandomNumericalSemigroup(5,10,50));;
 for i in [1..20] do
   L1:=[];
   k:=Random(1,5);
   for j in [1..k] do
     a:=Random(-15,15);
     while (a in L1) do
       a:=Random(-15,15);
     od;
     Add(L1,a);
   od;
   Sort(L1);
```

```
    I1:=L1+l[i];
    L1:=MinimalGenerators(I1);
    I1:=L1+l[i];
    aux:=IdealDecomposition(I1);
    I2:=aux[1];
    for j in [2..Length(aux)] do
      I2:=IntersectionIdealsOfNumericalSemigroup(I2,aux[j]);
    od;
    if not (I1=I2) then return fail; fi;
 od;
 return "success";
end;
```

Ejecutandolo, vemos que tiene éxito:

```
gap> TestIdealDecomposition();
"success"
```

Que cada componente es irreducible, es algo que es inmediato por el teorema de descomposición, siendo de la forma $-x_k + K$.

ProperIdealDecomposition

Seguimos un proceso parecido al caso anterior, en este caso asegurándonos de que los ideales generados son propios, usando `IsIntegral(I)` para verificar dicha propiedad.

```
TestProperIdealDecomposition := function()
 local l,aux,L1, I1,I2, k, i, j, a;
 l:=List([1..20],_->RandomNumericalSemigroup(5,10,50));;
 for i in [1..20] do
    L1:=[];
    I1:=-1+l[i];
    while not(IsIntegral(I1)) do
      k:=Random(1,5);
      for j in [1..k] do
        a:=Random(Conductor(l[i]),10*Conductor(l[i]));
        while (a in L1) do
          a:=Random(Conductor(l[i]),10*Conductor(l[i]));
        od;
        Add(L1,a);
      od;
      Sort(L1);
```

```
      I1:=L1+l[i];
    od;
    aux:=ProperIdealDecomposition(I1);
    I2:=Intersection(aux);
    if not (I1=I2) then return fail; fi;
 od;
 return "success";
end;
```

Las descomposiciones son correctas.

```
gap> TestProperIdealDecomposition();
"success"
```

Las componentes son de la forma $S \setminus D(x)$, que como hemos visto, son ideales propios e irredicibles.

5.2.2. Caracterización de semigrupos numéricos

Vamos a verificar que las funciones que están relacionadas con la caracterización de semigrupos, es decir, las funciones que discutimos en el capítulo 3, generan resultados correctos.

NuSequence y Nu

Veamos que la secuencia ν es correcta. En este caso, comparamos los resultados generados por la función con los ejemplos proporcionados en el articulo [2].

```
gap> TestNuSequence();

 nu1=[ 1, 2, 2, 3, 4, 3, 4, 6, 6, 4, 5, 8, 9, 8, 9, 10, 12, 12 ]

 NuSequence(s1)=[ 1, 2, 2, 3, 4, 3, 4, 6, 6,
 4, 5, 8, 9, 8, 9, 10, 12, 12 ]

 NuSequence(s1)=nu1? true

 nu2=[ 1, 2, 2, 2, 3, 4, 5, 4, 3, 2, 4, 6, 8, 8, 8, 8 ]

 NuSequence(s2)=[ 1, 2, 2, 2, 3, 4, 5, 4,
 3, 2, 4, 6, 8, 8, 8, 8 ]

 NuSequence(s2)=nu2? true
```

SemigroupFromNu

Veamos que el semigrupo obtenido usando esta función coincide con el original. Para ello, generamos semigrupos de forma aleatoria, calculamos su secuencia ν y a partir de esta el semigrupo. Comparamos el semigrupo obtenido con el original, y si encontramos alguna discrepancia, lo devolvemos. Ese es el código del test:

```
TestSemigroupFromNu := function()
 local l, s, last;
 l:=List([1..20],_->RandomNumericalSemigroup(5,10,100));;
 s:=First(l, s->not(s=SemigroupFromNu(NuSequence(s))));
 Print(s);
end;
```

Vemos que funciona:

```
gap> TestSemigroupFromNu();
fail
```

Devuelve, `fail`, luego todos los semigrupos son correctos.

TauSemigroup y TauSequence

Vamos que la secuencia τ es generada correctamente, comparando con el ejemplo del artículo [2].

```
gap> TestTauSequence();
 tau1=[ 0, 0, 0, 1, 1, 2, 1, 2, 2, 2, 3, 3, 4, 4, 5, 4, 5, 5, 6 ]

 TauSequence(s1)=[ 0, 0, 0, 1, 1, 2, 1, 2, 2,
 2, 3, 3, 4, 4, 5, 4, 5, 5, 6 ]

 TauSequence(s1)=tau1? true

 tau2=[ 0, 0, 0, 0, 1, 1, 2, 2, 3, 0, 1, 2, 3, 3, 3, 3, 4 ]

 TauSequence(s2)=[ 0, 0, 0, 0, 1, 1, 2, 2,
 3, 0, 1, 2, 3, 3, 3, 3, 4 ]

 TauSequence(s2)=tau2? true
```

SemigroupFromTau

Verificamos que, los semigrupos obtenidos a partir de secuencias τ coinciden con los semigrupo con los cuales generamos dichas secuencias. Usamos un enfoque similar al de `SemigroupFromNu`.

```
TestSemigroupFromTau := function()
 local l, s, last;
 l:=List([1..20],_->RandomNumericalSemigroup(5,10,100));;
 s:=First(l, s->not(s=SemigroupFromTau(TauSequence(s))));
 Print(s);
end;
```

Vemos, que los 20 pares de semigrupos coinciden, pues el test devuelve `fail`,

```
gap> TestSemigroupFromTau();
fail
```

SemigroupFromOplus

Finalmente, para esta sección, verificamos que el semigrupo generado a partir de $\oplus$ es correcto. Para ello usamos el ejemplo de la tabla de valores en [2] y la función auxiliar en `createO`. Dada la semigroup, como una lista de ternas, vemos que genera el semigrupo completo.

```
O:=[ [ 1, 1, 1 ], [ 1, 2, 2 ], [ 1, 3, 3 ], [ 1, 4, 4 ],...
...
..., [ 20, 18, 43 ], [ 20, 19, 44 ], [ 20, 20, 45 ] ];

gap> TestSemigroupFromOplus();

SemigroupFromOplus(O)=s? true
NuFromOplus(O) is subset of NuSequence(s)? true
```

5.2.3. Distancias de Feng-Rao

Vamos a probar, para cada uno de los algoritmos, los valores de la distancia de Feng-Rao para un amplio conjunto de valores del semigrupo y compararlos con los resultados generados por la función `FengRaoDistance`.

FengRaoDistanceArf

Consideremos el semigrupo numérico,

```
gap> s:=NumericalSemigroupBySmallElements([0,12,24,32,36,40]);;
gap> TestFengRaoDistanceArf(s);
```

¿Correcto?	m	FengRaoDistance(S,1,m)	FengRaoDistanceArf(S,m)
true	12	2	2
true	24	2	2
true	32	2	2
true	36	2	2
true	40	2	2
true	41	2	2
true	42	2	2
true	43	2	2
true	44	2	2
true	45	2	2
true	46	2	2
true	47	2	2
true	48	2	2
true	49	2	2
true	50	2	2
true	51	2	2
true	52	4	4

Cuadro 5.1: FengRaoDistanceArf, valores de $m \in S \cap [2, 52]$

¿Correcto?	m	FengRaoDistance(S,1,m)	FengRaoDistanceArf(S,m)
true	53	4	4
true	54	4	4
true	55	4	4
true	56	4	4
true	57	4	4
true	58	4	4
true	59	4	4
true	60	4	4
true	61	4	4
true	62	4	4
true	63	4	4
true	64	6	6
true	65	6	6
true	66	6	6
true	67	6	6
true	68	6	6
true	69	6	6
true	70	6	6
true	71	6	6

Cuadro 5.2: FengRaoDistanceArf, valores de $m \in S \cap [53, 71]$

Resultado correcto?	m	FengRaoDistance(S,1,m)	FengRaoDistanceArf(S,m)
true	72	8	8
true	73	8	8
true	74	8	8
true	75	8	8
true	76	10	10
true	77	10	10
true	78	10	10
true	79	10	10
true	80	11	11
true	81	12	12
true	82	13	13
true	83	14	14
true	84	15	15
true	85	16	16
true	86	17	17
true	87	18	18
true	88	19	19
true	89	20	20
true	90	21	21
true	91	22	22
true	92	23	23
true	93	24	24
true	94	25	25
true	95	26	26
true	96	27	27
true	97	28	28
true	98	29	29
true	99	30	30

Cuadro 5.3: FengRaoDistanceArf, valores de $m \in S \cap [72, 99]$

FengRaoDistance2

Consideremos el mismo semigrupo que el caso anterior, en este caso calculamos la segunda distancai de Feng-Rao.

Resultado correcto?	m	FengRaoDistance(S,2,m)	FengRaoDistanceArf2(S,m)
true	12	3	3
true	24	3	3
true	32	3	3
true	36	3	3
true	40	3	3
true	41	3	3
true	42	3	3
true	43	3	3
true	44	3	3
true	45	3	3
true	46	3	3
true	47	3	3
true	48	3	3
true	49	3	3
true	50	3	3
true	51	4	4
true	52	6	6
true	53	6	6
true	54	6	6
true	55	6	6
true	56	6	6
true	57	6	6
true	58	6	6

Cuadro 5.4: FengRaoDistanceArf2, valores de $m \in S \cap [12, 58]$

Resultado correcto?	m	FengRaoDistance(S,2,m)	FengRaoDistanceArf2(S,m)
true	59	6	6
true	60	6	6
true	61	6	6
true	62	6	6
true	63	8	8
true	64	9	9
true	65	9	9
true	66	9	9
true	67	9	9
true	68	9	9
true	69	9	9
true	70	9	9
true	71	11	11
true	72	12	12
true	73	12	12
true	74	12	12
true	75	13	13
true	76	14	14
true	77	15	15
true	78	15	15
true	79	16	16

Cuadro 5.5: FengRaoDistanceArf2, valores de $m \in S \cap [59, 79]$

TestFengRaoDistanceOrdinary

Examinemos los resultados para un semigrupo ordinario, que definimos a continuación,

```
gap> s:=NumericalSemigroupBySmallElements([0,7]);;
gap> IsOrdinary(s);
true
```

¿Resultado correcto?	m	FengRaoDistance(S,2,m)	FengRaoDistanceOrdinary2(S,m)
true	12	3	3
true	13	4	4
true	14	5	5
true	15	6	6
true	16	7	7
true	17	8	8

Cuadro 5.6: FengRaoDistanceOrdinaryf2, valores de $m \in [c, 2c-2]$

FengRaoDistanceSymmetric

Probamos, para el siguiente grupo simétrico, la función para calcular valores de la distancia de Feng Rao de elementos $m \in [c, 2c-1]$

```
gap> s:=NumericalSemigroup(3,7);;
gap> IsSymmetric(s);
true
```

¿Resultado correcto?	m	FengRaoDistance(S,1,m)	FengRaoDistanceSymmetric(S,m)
true	17	6	6
true	18	7	7
true	19	9	9
true	20	9	9
true	21	10	10
true	22	12	12
true	23	12	12
true	24	13	13
true	25	14	14
true	26	15	15
true	27	16	16
true	28	17	17

Cuadro 5.7: FengRaoDistanceSymmetric, valores de $m \in [c, 2c-1]$

FengRaoDistanceGeneralized

En ese caso, calculamos, para valores de $r = 2, 3, 7$, las distancias generalizadas que este método es capaz de calcular (Como dijimos, no es todos los elementos del intervalor $[c, 2c-2]$). Consideramos el mismo semigrupo anterior, $s = \langle 3, 7 \rangle$.

¿Ccorrecto?	m	FengRaoDistance(S,2,m)	FengRaoDistanceSymmetricGeneralized(S,2,m)
true	14	6	6
true	17	9	9
true	18	10	10
true	20	12	12
true	21	13	13
true	23	15	15
true	24	16	16
true	25	17	17
true	26	18	18
true	27	19	19
true	28	20	20

Cuadro 5.8: FengRaoDistanceSymmetricGeneralized, $r = 2$ valores de $m \in [c, 2c-1]$ con $(m - 2 * g + 1) \in s$

¿Correcto?	m	FengRaoDistance(S,3,m)	FengRaoDistanceSymmetricGeneralized(S,3,m)
true	14	9	9
true	17	12	12
true	18	13	13
true	20	15	15
true	21	16	16
true	23	18	18
true	24	19	19
true	25	20	20
true	26	21	21
true	27	22	22
true	28	23	23

Cuadro 5.9: FengRaoDistanceSymmetricGeneralized, $r = 3$ valores de $m \in [c, 2c-1]$ con $(m - 2 * g + 1) \in s$

¿Correcto?	m	FengRaoDistance(S,7,m)	FengRaoDistanceSymmetricGeneralized(S,7,m)
true	14	15	15
true	17	18	18
true	18	19	19
true	20	21	21
true	21	22	22
true	23	24	24
true	24	25	25
true	25	26	26
true	26	27	27
true	27	28	28
true	28	29	29

Cuadro 5.10: FengRaoDistanceSymmetricGeneralized, $r = 7$ valores de $m \in [c, 2c-1]$ con $(m - 2 * g + 1) \in s$

FengRaoDistanceTwoGenerators

Hacemos pruebas para el semigrupo con dos generadores $s := \langle 4, 5 \rangle$ usando el método `FengRaoDistanceTwoGenerators(S,m)`.

```
gap> s:=NumericalSemigroup(4,5);
<Numerical semigroup with 2 generators>
```

¿Correcto?	m	FengRaoDistance(S,1,m)	FengRaoDistanceTwoGenerators(S,m)
true	17	8	8
true	18	8	8
true	19	8	8
true	20	9	9
true	21	10	10
true	22	12	12
true	23	12	12
true	24	13	13
true	25	14	14
true	26	15	15
true	27	16	16

Cuadro 5.11: FengRaoDistanceTwoGenerators, valores de $m \in [c, 2c-1]$

Conclusión

En este proyecto hemos presentado los conceptos matemáticos básicos (de semigrupos, ideales y distancias de Feng Rao) para poder introducir una serie de resultados teóricos (como los teoremas que describen como calcular las distancias de Feng-Rao) en los cuales nos hemos basado para desarrollar el código de este proyecto. Estos resultados están sacados de diversas fuentes bibliográficas, que han sido la base de este trabajo.

Debido a las circunstancias particulares del Doble Grado, he podido usar como punto de partida la investigación del primer trabajo de fin de grado (de matemáticas). En dicho trabajo estudié conceptos de semigrupos numéricos y sus ideales, códigos correctores lineales y códigos algebraico geométricos en un punto. Con esta base, he podido realizar la labor de investigación directamente, sin tener que estudiar los conceptos básicos necesarios para poder entender la bibliografía. Así, he podido usar resultados de múltiples artículos y también he podido dedicar más tiempo a la labor de desarrollo y prueba del software.

El proyecto ha sido orientado a la creación de una contribución al paquete de GAP "NumericalSpgs", y por tanto se centra en los conceptos que, Pedro A. Garcia ha considerado que era deseable implementar para incorporarlos a la librería "NumericalSpgs". Estos conceptos son: la descomposición de ideales relativos y propios en componentes irreducibles, las caracterizaciones de los semigrupos a través de la secuencia ν y τ y la operación $\oplus$; y finalmente, las distancias de Feng-Rao.

Establecimos, al principio del trabajo, unos objetivos, que como podemos ver se han alcanzado, habiendo implementado todas las funciones requeridas. Mientas escribo este texto, parte del código implementado ya ha sido añadido al repositorio, mientras que el resto esta siendo revisado. Muchas de estas funciones formarán parte de la siguiente versión de la librería "NumericalSpgs".

Este trabajo ha sido la primera experiencia de trabajo para un cliente, desarrollando un producto software que va a tener una aplicación real. También es el primer es mi primera contribución que hago a un proyecto de código abierto.

En el calculo de la distancia de Feng-Rao, gracias a las pruebas, vemos que la nueva versión de la función `FenRaoDistance` es más rápida. Habiendo excluido de la versión final los dos algoritmos cuyos resultados eran inferiores, en todos los casos la nueva versión de la función es más rápida que la anterior.

En este trabajo, he usado conocimientos adquiridos en los dos grados, aplicando conocimiento de múltiples asignaturas. Algunos son conocimientos directos, de álgebra, códigos o programación. Mientras que otras son capacidades interdisciplinares, desarrolladas a lo largo de los cursos del los grados.

La combinación de el estudio teórico y práctico de los temas del trabajo me han llevado a familiarizarme con los conceptos de semigrupos, ideales, secuencias ν y τ, códigos correctores y distancias de Feng-Rao (entre otros). También he aprendido a usar el sistema y el lenguaje GAP, extender mi conocimiento en el uso de LaTeXy de Github.

En conjunto, el trabajo ha conllevado la aplicación de conocimientos de muchas áreas de estudio de ambas carreras, habiendo realizado un trabajo de investigación razonablemente completo (usando como fuentes artículos académicos), así como realizar un proyecto de desarrollo software no trivial, con una aplicación más allá de ser un ejercicio formativo.

Lineas de trabajo futuras

En base a la bibliografía estudiada, podemos plantear una serie de objetivos que podían expandir el alcance del trabajo, o como objetivos para futuros trabajos relacionados.

- **Números de Feng Rao:** Introdujimos los números de Feng-Rao en el capítulo 4, y se conocen algunos casos en lo cuales el cálculo de estos números es más sencillo que el caso general. Una función más rápida que calcule estos números haría el cálculo de las distancias de Feng-Rao más rápido, para ciertos casos (como el caso $m \geq 2c - 1$).

- Problema **Wire-Tap Channel II:** Es un tema interesante sobre el que me hubiera gustado poder investigar más. Como hemos dicho, está relacionado los pesos de Hamming generalizados y habría sido posible continuar el trabajo hablando de dichos pesos.

- Estudio y optimización del caso general en el cálculo de la distancia de Feng-Rao. Quizá sea posible encontrar un algoritmo más rápido o hacer optimizaciones en el caso general de la función `FengRaoDistance`. También sería muy útil para evitar desbordamientos de memoria en casos donde $r >> 0$, poder reescribir el algoritmo como uno no recursivo.

- **Códigos correctores**: Uno de los temas del trabajo de matemáticas que no hemos tratado en este trabajo es el de códigos correctores. GAP tiene capacidades limitadas de cálculo simbólico (a diferencia de otros lenguajes como SAGE). Habría sido interesante haber trabajado en un caso práctico con códigos correctores lineales. En particular, implementar los dos algoritmos de decodificación para códigos AG descritos en [15].

- Investigar formas de optimizar el código existente, pues es posible que existen formas de reescribirlo de forma más eficiente.

- Un objeto de estudio importante en el TFG de matemáticas ha sido el semigrupo de Weiestrass, con el cual no hemos trabajado. Seria interesamte implementar una función en GAP que, dada la cuerva algebraica, calculase el semigrupo de Weiestrass asociado.

Apéndice A

Manual de usuario

Vamos a explicar como usar software desarrollado para este trabajo. El código puede descargarse del repositorio [14]. Todo este código está en el lenguaje GAP, por lo que es necesario instalar el sistema GAP, y el paquete "NumericalSpgs".

A.0.1. Instalación

Descargar e instalar GAP, "NumericalSpgs" y el código del trabajo

El sistema GAP puede descargarse de la página oficial [9], donde se dan las instrucciones de descarga `https://www.gap-system.org/Download/`. También es posible usar GAP desde SAGE. Los pasos básicos para descargarlo, consisten en:

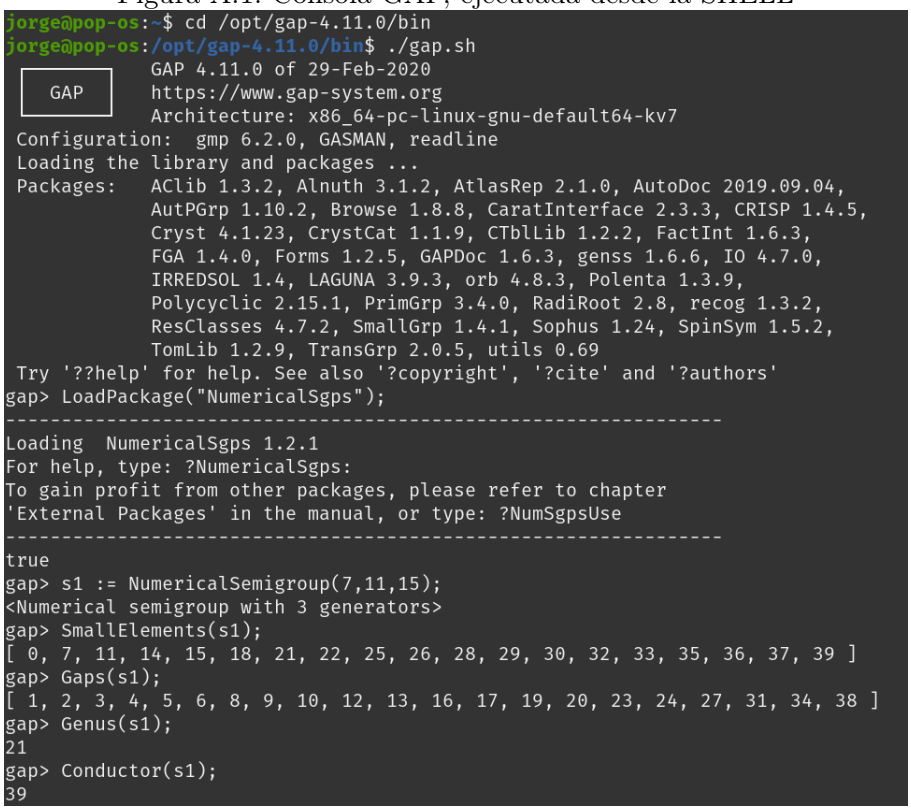
- Seleccionar una versión de GAP la página web `https://www.gap-system.org/Download/`, y descargarlo.
- Descomprimir el archivo.
- A continuación, para sistemas sistemas UNIX, Linux y macOS, ejecutar (desde el directorio descomprimido) el script `./configure` y el comando `make`.
- En UNIX, Linux y macOS, moverse a `pkg`, y ejecutar el script `../bin/BuildPackages.sh`. Este script creará la mayoría de los paquetes que necesitan ser compilados (si algo no funciona en este caso, mirar el archivo README del paquete correspondiente).
- En Windows no es necesario compilar, pues la mayoría de los paquetes ya incluyen ejecutables `.exe` en los archivos comprimidos.
- El paquete "NumericalSpgs" está incluido con la instalación por defecto, pero puede ser descargado si fuera necesario de Github [5]. En el manual de GAP se encuentran las instrucciones para instalar un paquete `https://www.gap-system.org/Manuals/doc/ref/chap76.html#X82473E4B8756C6CD`

- Finalmente, para usar el código del repositorio, basta con descargar los archivos en un directorio de trabajo. Cargaremos los archivos al ejecutar GAP.

Ejecutar GAP y usar el código

Tras realizar la instalación de GAP, y descargar el código del repositorio, basta con que ejecutemos el script `gap.sh` que encontraremos en el subdirectorio `bin`. Por ejemplo, en mi instalación: El siguiente paso es cargar la librería:

Figura A.1: Consola GAP, ejecutada desde la SHELL




```
gap> LoadPackage("NumericalSgps");
```

```
Loading  NumericalSgps 1.2.2
For help, type: ?NumericalSgps:
To gain profit from other packages, please refer to chapter
'External Packages' in the manual, or type: ?NumSgpsUse
```

---

true

Para cargar cualquiera de las funciones de este trabajo, usaremos la función `Read(<path>)`, donde `<path>` es la ruta del de dicha función en la instalación. Veamos un ejemplo, para cargar la función de descomposición de ideales:

```
gap> Read("/home/jorge/Escritorio/TFG/
TFG-informatica/ideals/IdealDecomposition.g");
```

También se puede cargar todo el directorio:

```
gap> Read("/home/jorge/Escritorio/TFG/
TFG-informatica/ideals");
```

Notemos que, algunas de estas funciones dependen de otras, luego es necesario cargar estas primero.

A.0.2. Descripción de las funciones

Ideales

En el repositorio, hay un directorio llamado `ideals`. Este contiene tres funciones. La función `IdealByDivisorClosedSet` es una función de "NumericalSpgs" que será incluida en una futura versión, aquí la hemos incluido por motivos de compatibilidad. Las otras dos funciones corresponden con lo métodos de descomposición de ideales en irreducibles.

- `IdealDecomposition(I)`: Dado I, un ideal relativo de un cierto semigrupo numérico, devuelve la lista de ideales irreducibles (componentes $\mathbb{Z}$-irreducibles) en los cuales el ideal se descompone.

  Input I, un ideal de un cierto semigrupo numérico.

  Output Una lista finita L, de ideales $\mathbb{Z}$- irreducibles tales que $\cap_{l_i \in L} l_i = I$.

  Ejemplo:

```
gap> s:=NumericalSemigroup(3,5);;
gap> I:=[-5,2]+s;;
gap> L:=IdealDecomposition(I);
[ <Ideal of numerical semigroup>,
<Ideal of numerical semigroup> ]
gap> I=Intersection(L[1],L[2]);
true
gap> SmallElements(L[1]); SmallElements(L[2]);
SmallElements(I);
[ -8, -5, -3, -2, 0 ]
[ -10, -7, -5, -4, -2 ]
[ -5, -2, 0 ]
```

- `ProperIdealDecomposition(I)` Da la descomposición de I, un ideal propio de un cierto semigrupo en componentes irreducibles.

  Input: I, un ideal propio de un cierto semigrupo numérico. Es decir, `IsIntegral(I)` debe ser cierto.
  Output: Una lista finita L, de ideales propios e irreducibles (de la forma $S \setminus D(x)$) tales que $\cap_{l_i \in L} l_i = I$.

  Ejemplo:

```
gap> s:=NumericalSemigroup(3,5);;
gap> I:=[15,11]+s;;
gap> L:=ProperIdealDecomposition(I);
[ <Ideal of numerical semigroup>, <Ideal of numerical semigroup> ]
gap> SmallElements(L[1]); SmallElements(L[2]); SmallElements(I);
[ 5, 8, 10, 11, 13 ]
[ 6, 9, 11, 12, 14 ]
[ 11, 14 ]
gap> I=Intersection(L[1],L[2]);
true
```

Caracterización de semigrupos

Tratamos el contenido del directorio `SemigroupCharacterization`. Aquí están todas las funciones discutidas en el capítulo 3, sobre las secuencias ν, τ y la operación $\oplus$.

- `Nu(i,S)`: calcula el valor de la secuencia ν del elemento i-ésimo del semigrupo S. Con índices comenzando en 1, calcula $\nu(S[i]) = \nu_i$

  Input: S, un semigrupo numérico e i, un entero positivo.

  Output: El valor de la secuencia ν asociado al elemento i-ésimo del semigrupo numérico (índices empezando en 1).

  Ejemplo:

```
gap> s:=NumericalSemigroup(3,7);;
gap> Nu(1,s); Nu(2,s); Nu(3,s); Nu(4,s); Nu(5,s);
1 2 3 2 4
```

- `NuSequence(S)`: Calcula la secuencia ν del semigrupo S. En concreto, calcula los elementos de dicha secuencia $\{\nu_1, \ldots, \nu_{2c-g}\}$ (con índices empezando en 1, c y g el conductor y el género del semigrupo). Para $i > 2c - g$, sabemos que $\nu_{i+1} = \nu_i + 1$.

  Input: Un semigrupo numérico, S.

  Output: Una lista, formada por los primeros $2c - g$ elementos de la ν secuencia del semigrupo S.

  Ejemplo:

```
gap> s:=NumericalSemigroup(3,7);;
gap> NuSequence(s);
[ 1, 2, 3, 2, 4, 4, 5, 6, 3, 6, 8, 6, 7, 10, 9, 10, 12, 12 ]
```

- `SemigroupFromNu(Nu)`: Dada una lista, `Nu`, que forme parte del principio de la secuencia ν, calcula el semigrupo numérico S, asociado a dicha sucesión. (Basta con los primeros $2c - g$ elementos de la sucesión).

  Input: Una lista, correspondiente a los primeros k elementos de una sucesión ν. Necesariamente, $k \geq 2c - g$.

  Output: Un semigrupo numérico, cuya sucesión ν es la que se ha pasado como Input.

  Ejemplo:

```
gap> nu:=[ 1, 2, 3, 2, 4, 4, 5, 6, 3, 6, 8,
6, 7, 10, 9, 10, 12, 12 ];;
gap> S:=SemigroupFromNu(nu);
<Numerical semigroup>
gap> NuSequence(S)=nu;
true
```

- `TauSemigroup(i,S)` calcula el valor de la secuencia τ del elemento i-ésimo del semigrupo S. Con índices comenzando en 1, calcula $\tau(S[i]) = \tau_i$

  Input: S, un semigrupo numérico e i, un entero positivo.

  Output: El valor de la secuencia τ asociado al elemento i-ésimo del semigrupo nu-

mérico (índices empezando en 1).

Ejemplo:

```
gap> s:=NumericalSemigroup(3,7,19);;
gap> TauSemigroup(1,s);TauSemigroup(2,s); TauSemigroup(3,s);
TauSemigroup(4,s); TauSemigroup(5,s);
TauSemigroup(10,s); TauSemigroup(11,s);
0 0 1 0 1 2 3
```

- `TauSequence(S)`: Calcula la secuencia τ del semigrupo S. En concreto, calcula los elementos de dicha secuencia $\{\tau_1, \ldots, \tau_{2c-g+1}\}$ (con índices empezando en 1, c y g el conductor y el género del semigrupo). Para $i > 2c - g$, sabemos que $\tau_{i+1} = \tau_i$ si i es impar, y $\tau_{i+1} = \tau_i + 1$ si es par.

  Input: Un semigrupo numérico, S.

  Output: Una lista, `Tau`, formada por los primeros $2c - g + 1$ elementos de la τ secuencia del semigrupo, S.

  Ejemplo:

```
gap> s:=NumericalSemigroup(3,7,19);;
gap> TauSequence(s);
[ 0, 0, 1, 0, 1, 1, 2, 2, 3, 2, 3, 3, 4, 4, 5, 4, 5, 5, 6 ]
```

- `SemigroupFromTau(Tau)`: Dada una lista, que forme parte del principio de la secuencia τ, calcula el semigrupo numérico S, asociado a dicha sucesión. (Basta con los primeros $2c - g$ elementos de la sucesión).

  Input: Una lista, correspondiente a los primeros k elementos de una sucesión τ. Necesariamente, $k \geq 2c - g$.

  Output: Un semigrupo numérico, cuya sucesión τ es la que se ha pasado como Input.

  Ejemplo:

```
gap> tau:=[ 0, 0, 1, 0, 1, 1, 2, 2, 3, 2,
3, 3, 4, 4, 5, 4, 5, 5, 6 ];;
gap> SemigroupFromTau(tau);
```

```
<Numerical semigroup>
gap> tau=TauSequence(SemigroupFromTau(tau));
true
```

- `Oplus(i,j,S)`: Calcula $k = i \oplus j$.

  Input: Los índices i y j, enteros positivos y un semigrupo numérico, S.

  Output: $k = i \oplus_S j$. Es decir, un entero positivo k, tal que $S[k] = S[i] + S[j]$.

  Ejemplo

```
gap> s:=NumericalSemigroup(5,7,11);;
gap> Oplus(2,3,s);
6
gap> s[2]+s[3]=s[6];
true
```

- `SemigroupFromOplus(O)`: Calcula el semigrupo, a partir de la operación $\oplus$. Dicha operación es caracterizada como una lista de ternas. En el este cálculo usamos la función `NuFromOplus` en dicho cálculo. Para pruebas, se puede usar la función `createO` en el directorio de tests para generar una lista de ternas a partir de un semigrupo.

  Input: Una lista de ternas O, cada uno de sus elementos de la forma $(i, j, i \oplus j)$. Dicha lista debe tener suficientes elementos para caracterizar el semigrupo.

  Output: Un semigrupo numérico, cuya operación $\oplus$ corresponde con O.

  Ejemplo

```
s:=NumericalSemigroup(5,7,11);
gap> O:=createO(s,30,30);;
gap> S:=SemigroupFromOplus(O);
<Numerical semigroup>
gap> s=S;
true
```

Distancias de Fen-Rao

Consideremos el contenido del directorio `FengRaoDistance`, donde está todas las funciones relacionadas con el cálculo de las distancias de Feng-Rao.

- `FengRaoDistance2(s,r,m)`: Calcula la distancia de Feng-Rao generalizada de orden r; $\delta_{FR}^r(m)$. Incorpora todas las mejoras realizadas en el trabajo.

  Input: s, un semigrupo numérico, r, un entero positivo, y un elemento del semigrupo numérico, $m \in s$.

  Output: La distancia r-ésima del elemento m en el semigrupo s: $\delta_{FR}^r(m)$.

  Ejemplo:

```
gap> s:=NumericalSemigroupBySmallElements([0, 12, 24, 32, 36, 40]);
<Numerical semigroup>
gap> IsArf(s);
true
gap> FengRaoDistance2(s,2,69); FengRaoDistance2(s,2,55);
FengRaoDistance2(s,2,79);
9 6 16
gap> FengRaoDistance(s,2,69); FengRaoDistance(s,2,55);
FengRaoDistance(s,2,79);
9 6 16
```

- `FengRaoDistanceArf(S,m)` Calcula primera distancia de Feng-Rao (clásica) de un semigrupo de tipo Arf.

  Input: s, un semigrupo numérico de tipo Arf, un elemento del semigrupo numérico, $m \in s$.

  Output: La distancia clásica de Feng-Rao del elemento m en el semigrupo s: $\delta_{FR}(m)$.

- `FengRaoDistance2(S,m)` Calcula segunda distancia de Feng-Rao de un semigrupo de tipo Arf.

  Input: s, un semigrupo numérico de tipo Arf, un elemento del semigrupo numérico, $m \in s$, con $m \in [c, 2c-2]$.

  Output: La distancia de Feng-Rao generalizada de orden 2 del elemento m en el semigrupo s: $\delta_{FR}^2(m)$.

- `FengRaoDistanceBruteForce(S,m)` Calcula la primera distancia de Feng-Rao, de forma no recursiva.

Input: s, un semigrupo numérico, un elemento del semigrupo numérico, $m \in s$.

Output: La distancia clásica de Feng-Rao del elemento m en el semigrupo s: $\delta_{FR}(m)$.

- `FengRaoDistanceOrdinary2(S,m)` Calcula segunda distancia de Feng-Rao de un semigrupo de tipo Ordinario.

  Input: s, un semigrupo numérico de tipo ordinario, un elemento del semigrupo numérico, $m \in s$, con $m \in [c, 2c-2]$.

  Output: La distancia de Feng-Rao generalizada de orden 2 del elemento m en el semigrupo s: $\delta^2_{FR}(m)$.

- `FengRaoDistanceSymmetric(S,m)` Calcula la primera distancia de Feng-Rao de un elemento de un semigrupo simétrico en el intervalo $[c+1, 2c-2]$
  Input: s, un semigrupo numérico de tipo Simétrico, un elemento del semigrupo numérico, $m \in s$.

  Output: La distancia clásica de Feng-Rao del elemento m en el semigrupo s: $\delta_{FR}(m)$.

- `FengRaoDistanceSymmetricGeneralized(S,r,m)` Calcula la distancai de Feng-Rao generalizada del elemento m, con las condiciones impuestas.$\delta^r_{FR}(m)$.

  Input: s, un semigrupo numérico, r, un entero positivo, un elemento del semigrupo numérico, $m \in s$, con $m \in [c+1, 2c-2]$ y $m - 2g + 1 \in s$.

  Output: La distancia r-ésima del elemento m en el semigrupo s: $\delta^r_{FR}(m)$.

- `FengRaoDistanceTwoGenerators(S,m)` Calcula la primera distancia de Feng-Rao de m, para un semigrupo con un semigrupo con dos generadores.

  Input: s, un semigrupo numérico con dos generadores, y un elemento del semigrupo numérico, $m \in s$, con $m < 2c-1$ y $m - 2g + 1 \in s$.

  Output: La distancia clásica del elemento m en el semigrupo s: $\delta_{FR}(m)$.

Bibliografía

[1] P.A. García-Sánchez Abdallah Assi Marco D'Anna. *Numerical semigroups and applications*. Springer, 2020. Cap. Chapter 3, Ideals.

[2] Maria Bras-Amorós. *Numerical Semigroups and Codes*. E. Martinez-Moro, 2013. Cap. Chapter 5 of Algebraic Geometry Modeling in Information Theory.

[3] J.I. Farrán y C. Munuera. "Goppa-like bounds for the generalized Feng–Rao distances". En: *Discrete Applied Mathematics* 128 (2003), 145–156.

[4] Antonio Campillo, Jose Ignacio Farrán y Carlos Munuera. "On the parameters of algebraic geometry codes related to Arf semigroups". En: *IEEE Transactions on information theory* (2000).

[5] M. Delgado, P. A. Garcia-Sanchez y J. Morais. *NumericalSgps, A package for numerical semigroups, Version 1.2.2*. https://gap-packages.github.io/numericalsgps. Refereed GAP package. 2020.

[6] M. Delgado, P. A. Garcia-Sanchez y J. Morais. *numericalsgps– a package for numerical semigroups*. URL: https://www.gap-system.org/Manuals/pkg/NumericalSgps-1.2.1/doc/manual.pdf.

[7] José I. Farrán y Antonio Campillo. "Computing Weierstrass semigroups and the Feng Rao distance from singular plane models". En: *Elsevier* (1999). URL: https://core.ac.uk/download/pdf/81171302.pdf.

[8] José I. Farrán, Pedro A. García-Sánchez y Benjamín A. Heredia. "On the second Feng-Rao distance of Algebraic Geometry codes related to Arf semigroups". En: *Springer* (2018).

[9] *GAP – Groups, Algorithms, and Programming, Version 4.11.0*. The GAP Group. 2020. URL: https://www.gap-system.org.

[10] *GAP - Reference Manual*. Ver. 4.11.0. 2020. URL: https://www.gap-system.org/Manuals/doc/ref/manual.pdf.

[11] P.A. García-Sánchez J.C. Rosales. *Numerical Semigroups*. Vol. 20. Springer, 2009.

[12] J. I. Farran M. Delgado, P. A. García-Sánchez y D. Llena. "On the weight hierarchy of codes coming from semigroups with two generators". En: *IEEE Transactions on information theory* (2013).

[13] L. H. Ozarow y A. D. Wyner. "Wire-Tap Channel II". En: *AT&T Bell Laboratories Technical Journal* 63.10 (1984), págs. 2135-2157. DOI: https://doi.org/10.1002/j.1538-7305.1984.tb00072.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1984.tb00072.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1984.tb00072.x.

[14] Jorge Angulo Rodriguez. *Repositorio, código de semigrupos, TFG informática.* URL: https://github.com/Ataraxta/TFG-informatica.

[15] Jorge Angulo Rodriguez. *Trabajo de Fin de Grado, Semigrupos numéricos, códigos AG en un punto y pesos de Hamming Generalizados.* 2020.

[16] Christoph Kirfel y Ruud Pellikaan. "The minimum distance of codes in an array coming from telescopic semigroups". En: *IEEE Transactions Information Theory, vol. 41* (1995).

[17] V. K. Wei. "Generalized Hamming weights for linear codes". En: *IEEE Transactions on Information Theory* 37.5 (1991), págs. 1412-1418. DOI: 10.1109/18.133259.