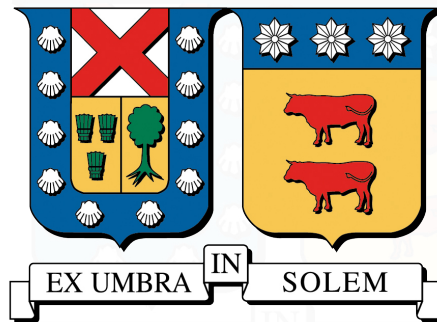


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAISO - CHILE



Aplicación Para Encontrar Sub Variedades Abelianas

SVEN CHRISTIAN MARTIN KLEIN PLARRE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRÓNICO

PROFESOR GUÍA : DR. WERNER CREIXELL F.
PROFESOR CORREFERENTE : DRA. ANITA M. ROJAS

JULIO 2024

AGRADECIMIENTOS

Infinitas gracias a mis padres por tenerme paciencia y apoyarme incondicionalmente todos estos años gracias a los cuales me he podido convertir en el hombre que hoy soy. A mis hermanas por estar ahí para mí. A Alejandra Larrain y a Eduardo Correa por ayudarme en mis momentos más difíciles y por enseñarme a entenderme a mí mismo. También a Amalia Pizarro-Madariaga por compartir sus apuntes de criptografía de manera desinteresada.



Índice de Contenidos

1. Introducción	1
2. Trabajo a desarrollar y resultados esperados	3
3. Desarrollo de la Propuesta	4
3.1. Descomposición del Sistema	4
3.2. Demostración de la Propiedad del Espacio Elipsoide	5
3.3. Determinando los Límites del Espacio	7
3.4. Proceso de Búsqueda	9
3.4.1. Método "full scan"	10
3.4.2. Método "elipsoid looping"	10
3.5. Funcionamiento del programa	11
3.6. Ejemplo	11
4. Conclusiones y Trabajo a Futuro	14
5. ANEXOS	15
5.1. Código	15
5.2. Demostración del Espacio Elipsoide en Wolfram Mathematica	21
Bibliografía	24

Índice de Figuras

3.1. Espacio de las soluciones para el sistema de orden 2 con variables $d2 = 1, t1 = 0.5 + i\sqrt{2}/2, t2 = -0.5$ y $t3 = 0.5 + i\sqrt{2}/2$	6
3.2. Ejemplo de otros tipos de superficie cuadrática	6
3.3. Representación en 2 dimensiones del proceso para encontrar máximos y mínimos.	8
3.4. Ejemplo de la elipse 3.3 con los puntos cuya gradiente equivale a un valor unitario.	8
3.5. Puntos máximos y mínimos de cada eje obtenidos con el método anteriormente descrito para una superficie de parámetros $d2 = 3, \tau_1 = 4i/5 + 7/5, \tau_2 = 3i/5 + 9/5$ y $\tau_3 = 6i/5 + 3/5$	9
3.6. Gráfico de puntos evaluados en una sección transversal de un elipsoide. Se muestran con las flechas azules la secuencia en que fueron evaluados los puntos rojos.	11
3.7. Gráfico de la superficie del ejemplo.	12
3.8. Lista de soluciones para la superficie del ejemplo.	13
3.9. Lista de soluciones para la superficie del ejemplo usando el método "ellipsoid looping".	13

RESUMEN

En el ámbito de la criptografía el mundo se encuentra al borde de un cambio grande. Con el advenimiento de los computadores cuánticos los métodos tradicionales de cifrado que han estado en uso desde hace varias décadas quedarán rápidamente obsoletos. Algunas de las técnicas propuestas para nuevos algoritmos, llamados *postcuánticos*, utilizan curvas algebraicas de género dos (sobre cuerpos finitos) tales que su variedad Jacobiana es isógena a un producto de (dos) curvas elípticas. Con ello se generalizan métodos que utilizan curvas elípticas (que son curvas algebraicas de género 1 y por ende equivalentes a sus Jacobianas) que están en uso en la actualidad.

Un primer paso para construir este tipo de curvas algebraicas es hacerlo sobre el cuerpo de los números complejos. Como las variedades Jacobianas son variedades abelianas, se usa que allí hay teoría para determinar cuándo una variedad abeliana A de dimensión g se descompone como producto de subvariedades abelianas de dimensión menor. En el caso $g = 2$, como producto de (dos) curvas elípticas.

El método que aquí implementamos computacionalmente, usa la *matriz de períodos* de A para generar un sistema de ecuaciones no lineales en varias variables. La variedad A es isógena a un producto de subvariedades sí y sólo si dicho sistema tiene solución en los racionales. El tamaño del sistema depende de la **dimensión** g de la variedad A . En el presente documento se trabaja con variedades de dimensión 2; es decir, *superficies abelianas*, las cuales entregan 4 parámetros de entrada que definen las ecuaciones de un sistema de 6 variables pertenecientes a los racionales y 3 ecuaciones.

En este ámbito, para comprobar si una superficie abeliana dada se descompone como un producto de curvas elípticas, se debe resolver sobre los racionales un sistema de ecuaciones dependiente de un grupo de parámetros de entrada, que vienen de la superficie, que lo determina.

El tema del trabajo es la búsqueda de soluciones en el dominio de los números racionales para el sistema de ecuaciones determinado por los parámetros entregados en la matriz de períodos de una superficie abeliana A . Se debe encontrar al menos una tupla de valores racionales para que se compruebe que el sistema tiene solución. El trabajo no incluye encontrar esa matriz de períodos, es un dato que se recibe y con el cual se construye y resuelve el sistema.

Lo primero que se hizo fue reducir el sistema a una ecuación cuadrática de 3 dimensiones. Una vez realizado esto se demostró que por las propiedades del sistema, esta ecuación cuadrática forma un elipsoide, por lo que el espacio de las soluciones es acotado en los reales. A continuación se ideó un método basado en "apuntar" el gradiente del sistema de ecuaciones en la misma dirección que los ejes coordenados para calcular los límites del espacio de las soluciones en donde realizar la búsqueda.

Luego se explica el proceso de búsqueda en el propio espacio y las dificultades encontradas al implementarlo y al obtener resultados. Este proceso consiste en realizar una subdivisión del espacio tomando muestras de este según un diferencial entre muestras dado por el usuario entre los límites máximos y mínimos de las variables siendo evaluadas. Estas muestras son evaluadas en las ecuaciones para comprobar si el resultado es válido y lo entregan si así lo es.

Finalmente se va al detalle del funcionamiento del programa: qué funciones y utilidad entrega, cómo funciona, cómo se usa y qué resultados entregará.

ABSTRACT

In the field of cryptography, the world is on the brink of a big change. With the advent of quantum computers, traditional encryption methods that have been in use for several decades will quickly become obsolete. Some of the techniques proposed for new algorithms, called *post-quantum*, use algebraic curves of genus two (over finite fields) such that their Jacobian manifold is isogeneous to a product of (two) elliptic curves. With this, methods that use elliptic curves (which are algebraic curves of genus 1 and therefore equivalent to their Jacobian curves) that are in use today are generalized.

A first step to build this type of algebraic curves is to do it on the field of Complex Numbers. Since Jacobian manifolds are abelian manifolds, it is used that there is theory to determine when an abelian manifold A of dimension g decays as a product of abelian submanifolds of smaller dimension. In the case $g = 2$, as a product of (two) elliptic curves.

The method that we implement computationally uses the *period matrix* of A to generate a system of nonlinear equations in several variables. Manifold A is isogenic to a product of submanifolds if and only if said system has a solution in rationals. The size of the system depends on the **dimension** g of variety A . In this document we work with varieties of dimension 2; that is, *abelian surfaces*, which provide 4 input parameters that define the equations of a system of 6 variables belonging to the rationals and 3 equations.

In this field, to check if a given abelian surface breaks down as a product of elliptic curves, a system of equations dependent on a group of input parameters, which come from the surface, that determines it, must be solved over the rationals.

The subject of the work is the search for solutions in the domain of rational numbers for the system of equations determined by the parameters given in the period matrix of an abelian surface A . At least one tuple of rational values must be found in order to prove that the system has a solution. The work does not include finding that period matrix, it is a data that is received and with which the system is built and solved.

The first thing that was done was to reduce the system to a 3-dimensional quadratic equation. Once this was done, it was shown that due to the properties of the system, this quadratic equation forms an ellipsoid, so the space of the solutions is bounded in the real numbers. Next, a method based on "pointing" the gradient of the system of equations in the same direction as the coordinate axes was devised to calculate the limits of the space of solutions where to perform the search.

Then the search process in the space itself is explained, as well as the difficulties encountered in implementing it and obtaining results. This process consists of carrying out a discrete subdivision of the space, taking samples from it and evaluating them. The differential between samples is given by the user. The samples are taken between the maximum and minimum limits of the variables being evaluated. These samples are then evaluated in the original equations to check if the result is valid and they deliver it if it is.

Finally, the details about how the program works: what functions and utility it provides, how it works, how it is used and what results it will deliver.

1 | Introducción

La ciencia de la criptografía se dedica a estudiar formas de cifrar mensajes, de tal forma de que estos puedan ser enviados a través de un canal público e inseguro (como el internet). El objetivo de esta acción es que estos no sean legibles por un tercero y sólo por el destinatario del mensaje.

En estos momentos la mayoría de las comunicaciones en internet están encriptadas por medio del protocolo https, que usa el método RSA (Rivest, Shamir y Adleman) (1) de intercambio de claves públicas y privadas. Este protocolo funciona utilizando la clave pública de un dominio o usuario para cifrar un paquete de datos. Esta clave corresponde a un número primo de gran tamaño. El paquete de datos enviado sólo puede ser descifrado utilizando la clave privada que únicamente el receptor del mensaje posee. Este método lleva en uso desde 1976 pues el cifrado y descifrado de mensajes tiene relativa baja complejidad computacional. Sin embargo, la obtención de la clave privada a partir del mensaje cifrado y la clave pública tiene una complejidad muy alta. El uso principal del algoritmo es el intercambio de claves simétricas al establecer una comunicación entre emisor y receptor.

En 1985 Neal Koblitz y Victor Miller propusieron de forma independiente el uso de curvas elípticas para el cifrado de mensajes (2). En (3) se hace una explicación de lo que es una curva elíptica. La definición y uso de curvas elípticas no se encuentra dentro del alcance de este proyecto. En resumidas cuentas una curva elíptica son los puntos que satisfacen una ecuación polinomial del tipo:

$$y^2 = ax^3 + b$$

Lo importante es a que conjunto pertenecen los valores de a y b , así como las soluciones (x, y) . Para criptografía se necesita que el grupo tenga un número finito de puntos, pues los cálculos sobre números reales o complejos son lentos e imprecisos pues a fin de cuentas hay un proceso de "redondeo".

Éstos algoritmos de encriptación anteriormente mencionados son seguros con la tecnología computacional actual. En este momento, el mejor algoritmo para atacar una comunicación encriptada es el algoritmo de Pollard (4). Este algoritmo es capaz de resolver el problema en aproximadamente $O(\sqrt{p})$ pasos. Es decir que si se desencripta un mensaje cuya clave prima de menor tamaño es de 256 bits ($p = 2^{256}$), el algoritmo es capaz de descifrar la clave en 2^{128} pasos.

En 1997, Peter Shor ideó un algoritmo basado en computación cuántica (5) capaz de descifrar estos mensajes en tiempo polinomial. Esto implica que si estos procesadores estuvieran comercialmente disponibles, todos los sistemas de encriptación actuales quedarían obsoletos inmediatamente. Con el avance en el desarrollo de esta tecnología de hardware el NIST (Instituto Nacional de Estándares y Tecnología) convoca a un concurso a fines del 2017 para que la comunidad someta algoritmos resistentes a ataques cuánticos. Los seleccionados de este concurso son conocidos como algoritmos post cuánticos (6). Entre ellos se encuentra el algoritmo llamado *Supersingular Isogeny Diffie-Hellman* (SIDH). Este toma ingredientes del (ECDH) pero basa su seguridad en un problema diferente al logaritmo discreto en curvas elípticas: el llamado "Problema de la Isogenea Supersingular" (7).

Finalmente, en 2022 apareció un artículo (8) donde se propone el ataque mencionado en la sección anterior al SIDH. Este ataque se basa fuertemente, como ellos mismos dicen en la segunda línea del abstract, en un artículo de Ernst Kani (9). En este artículo se estudian curvas algebraicas de género 2 sobre los complejos (u otro cuerpo algebraicamente cerrado) y la descomposición de su variedad Jacobiana como producto de curvas elípticas.

Este es justamente el tópico de este trabajo de título: la implementación computacional de un método constructivo, basado en la resolución de un sistema de 6 incógnitas y 3 ecuaciones (2 lineales y una no lineal) sobre los racionales.

Esto tiene el objetivo de descomponer una variedad abeliana de dimensión 2 en producto de curvas elípticas.



2 | Trabajo a desarrollar y resultados esperados

El sistema que resulta de la teoría para descomponer variedades abelianas (10) consiste en 3 ecuaciones y 6 variables.

$$-d_2 = d_2 a_{13} + a_{24}, \quad (2.1)$$

$$0 = (\tau_1 \tau_3 - \tau_2^2) a_{12} - d_2 a_{14} \tau_1 + d_2 a_{13} \tau_2 - a_{24} \tau_2 + a_{23} \tau_3 + d_2 a_{34} \quad (2.2)$$

$$0 = a_{14} a_{23} - a_{13} a_{24} + a_{12} a_{34}. \quad (2.3)$$

$$(a_{12}, a_{13}, a_{14}, a_{23}, a_{24}, a_{34}) \in \mathbb{Q}^6$$

El cual resulta de considerar la matriz de períodos Π_A la superficie abeliana A que se desea descomponer. Es decir, los parámetros de entrada están ordenado matricialmente.

$$\Pi_A = \begin{pmatrix} 1 & 0 & \tau_1 & \tau_2 \\ 0 & d_2 & \tau_2 & \tau_3 \end{pmatrix}.$$

En donde d_2 es un número natural y τ_1, τ_2 y τ_3 son números complejos tales que la matriz Z_A , llamada *matriz de Riemann de A*, formada por las partes imaginarias $\Im(\tau_j)$ de los τ_j ; es decir,

$$Z_A = \begin{pmatrix} \Im(\tau_1) & \Im(\tau_2) \\ \Im(\tau_2) & \Im(\tau_3) \end{pmatrix}$$

es definida positiva.

El objetivo del proyecto es encontrar una solución en los Racionales para las variables del sistema. Es completamente indispensable para el contexto matemático del proyecto que el resultado no sea una aproximación o que tenga algún tipo de error ya sea cuantizable o no. Este debe ser exacto. Cualquier error, por pequeño que sea hará que el cálculo de la subvariedad abeliana correspondiente a la tupla solución encontrada falle. Si esto ocurre, dicha (*cuasi*-)solución no corresponda a una subvariedad y por ende no determine una descomposición de la superficie A de partida.

El programa para encontrar soluciones al sistema de ecuaciones será desarrollado en SageMath(11). SageMath es un sistema algebraico computacional basado en Python construido sobre paquetes matemáticos como Numpy y Sympy.

3 | Desarrollo de la Propuesta

Para desarrollar el sistema en una forma en que se pueda encontrar soluciones reales para este, se utilizarán las ecuaciones dadas para reducir el sistema a una ecuación. Esta ecuación corresponde a una superficie cuadrática real de dimensión 3 que forma una superficie elipsoide.

$$Ax^2 + By^2 + Cz^2 + Dyz + Exz + Fxy + Gx + Hy + Jz + K = 0 \quad (3.1)$$

3.1. Descomposición del Sistema

Para reducir el sistema de ecuaciones a una forma cuadrática se eliminan variables utilizando las ecuaciones del sistema. Esto tiene el objetivo de expresar variables como una función y así reemplazarla en la ecuación final. Empezando con la primera ecuación 2.1.

$$-d_2 = d_2 a_{13} + a_{24} \quad (3.2)$$

$$a_{24} = -d_2 - a_{13} d_2 \quad (3.3)$$

La ecuación 2.2 es una ecuación compleja, por lo que se debe descomponer en una parte real y una parte imaginaria. Para esto se utilizarán las equivalencias. Y el reemplazo que se acaba de obtener de la variable a_{24} .

$$\tau_1 = \tau_{1r} + i\tau_{1i}$$

$$\tau_2 = \tau_{2r} + i\tau_{2i}$$

$$\tau_3 = \tau_{3r} + i\tau_{3i}$$

$$0 = a_{34}d_2 - a_{14}d_2\tau_1 + a_{13}d_2\tau_2 - (-d_2 - a_{13}d_2)\tau_2 + a_{23}\tau_3 + a_{12}(\tau_1\tau_3 - \tau_2) \quad (3.4)$$

Primero se analiza la parte imaginaria de la ecuación 3.4 para despejar a_{23} .

$$0 = -a_{14}d_2\tau_{1i} + d_2\tau_{2i} + 2a_{13}d_2\tau_{2i} - 2a_{12}\tau_{2i}\tau_{2r} + a_{23}\tau_{3i} + a_{12}\tau_{1r}\tau_{3i} + a_{12}\tau_{1i}\tau_{3r} \quad (3.5)$$

$$a_{23} = (a_{14}d_2\tau_{1i} - d_2\tau_{2i} - 2a_{13}d_2\tau_{2i} + 2a_{12}\tau_{2i}\tau_{2r} - a_{12}\tau_{1r}\tau_{3i} - a_{12}\tau_{1i}\tau_{3r})/\tau_{3i} \quad (3.6)$$

Se continua con la parte real utilizando los reemplazos ya obtenidos para despejar a_{34} .

$$0 = a_{34}d_2 - a_{14}d_2\tau_{1r} + a_{12}\tau_{2i}^2 + d_2\tau_{2r} + 2a_{13}d_2\tau_{2r} - a_{12}\tau_{2r}^2 - a_{12}\tau_{1i}\tau_{3i} + \frac{a_{14}d_2\tau_{1i}\tau_{3r} - d_2\tau_{2i}\tau_{3r} - 2a_{13}d_2\tau_{2i}\tau_{3r} + 2a_{12}\tau_{2i}\tau_{2r}\tau_{3r} - a_{12}\tau_{1i}\tau_{3r}^2}{\tau_{3i}} \quad (3.7)$$

$$a_{34} = \frac{a_{14}d_2\tau_{1r}\tau_{3i} - a_{12}\tau_{2i}^2\tau_{3i} - d_2\tau_{2r}\tau_{3i} - 2a_{13}d_2\tau_{2r}\tau_{3i} + a_{12}\tau_{2r}^2\tau_{3i} + a_{12}\tau_{1i}\tau_{3i}^2}{\tau_{3i}d_2} + \frac{-a_{14}d_2\tau_{1i}\tau_{3r} + d_2\tau_{2i}\tau_{3r} + 2a_{13}d_2\tau_{2i}\tau_{3r} - 2a_{12}\tau_{2i}\tau_{2r}\tau_{3r} + a_{12}\tau_{1i}\tau_{3r}^2}{\tau_{3i}d_2} \quad (3.8)$$

Finalmente se integran todas las transformaciones en la ecuación 2.3 para obtener un sistema tridimensional cuadrático en función de las variables racionales a_{12} , a_{13} y a_{14} .

$$0 = a_{13}^2 d_2 + a_{13} d_2 + \frac{a_{14}(a_{14} d_2 \tau_{1i} - d_2 \tau_{2i} - 2a_{13} d_2 \tau_{2i} + 2a_{12} \tau_{2i} \tau_{2r} - a_{12} \tau_{1r} \tau_{3i} - a_{12} \tau_{1i} \tau_{3r})}{\tau_{3i}} \\ + a_{12} \frac{a_{14} d_2 \tau_{1r} \tau_{3i} - a_{12} \tau_{2i}^2 \tau_{3i} - d_2 \tau_{2r} \tau_{3i} - 2a_{13} d_2 \tau_{2r} \tau_{3i} + a_{12} \tau_{2r}^2 \tau_{3i} + a_{12} \tau_{1i} \tau_{3i}^2}{\tau_{3i} d_2} \\ - \frac{a_{14} d_2 \tau_{1i} \tau_{3r} + d_2 \tau_{2i} \tau_{3r} + 2a_{13} d_2 \tau_{2i} \tau_{3r} - 2a_{12} \tau_{2i} \tau_{2r} \tau_{3r} + a_{12} \tau_{1i} \tau_{3r}^2}{\tau_{3i} d_2} \quad (3.9)$$

Es posible ahora descomponer la ecuación obteniendo de esta los factores de la ecuación 3.1 para poder construir la forma matricial de la ecuación. Los factores simplificados son los siguientes:

$$A = \frac{-\tau_{2i}^2 \tau_{3i} + \tau_{2r}^2 \tau_{3i} + \tau_{1i} \tau_{3i}^2 - 2\tau_{2i} \tau_{2r} \tau_{3r} + \tau_{1i} \tau_{3r}^2}{\tau_{3i} d_2} \\ B = d_2 \\ C = \frac{d_2 \tau_{1i}}{\tau_{3i}} \\ D = \frac{-2d_2 \tau_{2i}}{\tau_{3i}} \\ E = \frac{2\tau_{2i} \tau_{2r} - \tau_{1r} \tau_{3i} - \tau_{1i} \tau_{3r}}{\tau_{3i}} + \frac{d_2 \tau_{1r} \tau_{3i} - d_2 \tau_{1i} \tau_{3r}}{\tau_{3i} d_2} = \frac{2\tau_{2i} \tau_{2r} - 2\tau_{1i} \tau_{3r}}{\tau_{3i}} \\ F = \frac{-2d_2 \tau_{2r} \tau_{3i} + 2d_2 \tau_{2i} \tau_{3r}}{\tau_{3i} d_2} = 2 \frac{\tau_{2i} \tau_{3r} - \tau_{2r} \tau_{3i}}{\tau_{3i}} \\ G = \frac{-d_2 \tau_{2r} \tau_{3i} + d_2 \tau_{2i} \tau_{3r}}{\tau_{3i} d_2} = \frac{\tau_{2i} \tau_{3r} - \tau_{2r} \tau_{3i}}{\tau_{3i}} \\ H = d_2 \\ J = \frac{-d_2 \tau_{2i}}{\tau_{3i}} \\ K = 0$$

Una vez reducido el sistema de grado 2 a la mínima cantidad de variables independientes es incluso posible graficar el espacio de las soluciones como se muestra en el ejemplo de la figura 3.1.

Ya obtenida la ecuación reducida, es posible empezar a desarrollar el modelo de búsqueda en el espacio de las soluciones. Pero antes es necesario conocer las características de este espacio para poder definir el área de búsqueda. En el ejemplo de la figura 3.1 la sub-variedad produce un elipsoide, pero una superficie cuadrática también puede producir un hiperboloide y un paraboloide, los cuales no son acotados. Así que antes de continuar con la búsqueda de soluciones se debe demostrar que el dominio de las soluciones es acotado.

3.2. Demostración de la Propiedad del Espacio Elipsoide

Para realizar la demostración de que al reducir variables el sistema restante forma un elipsoide, se utilizará la ecuación cuadrática del sistema y se transformará en su forma matricial, la cual se muestra a continuación:

$$\begin{pmatrix} A & F/2 & E/2 \\ F/2 & B & D/2 \\ E/2 & D/2 & C \end{pmatrix} \quad (3.10)$$

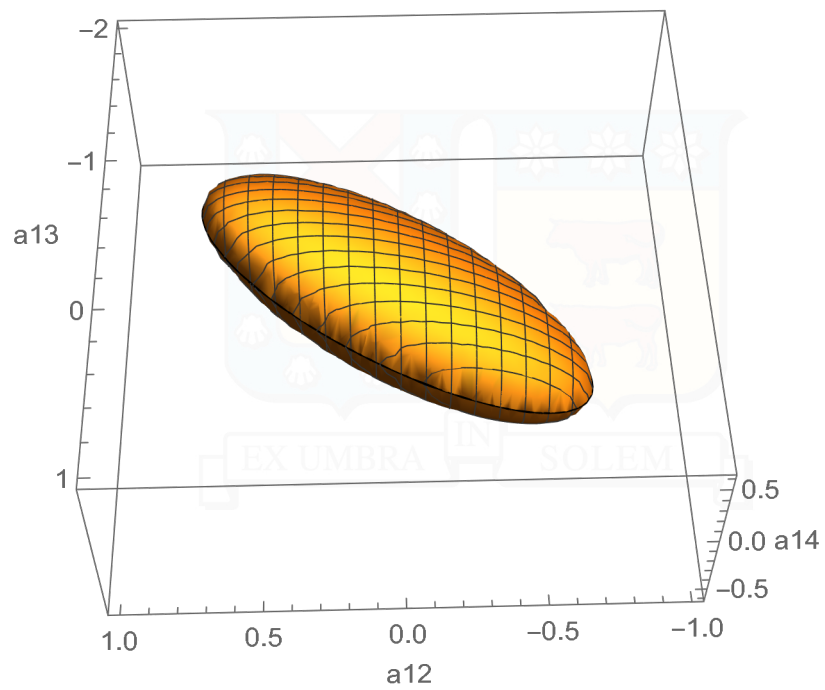
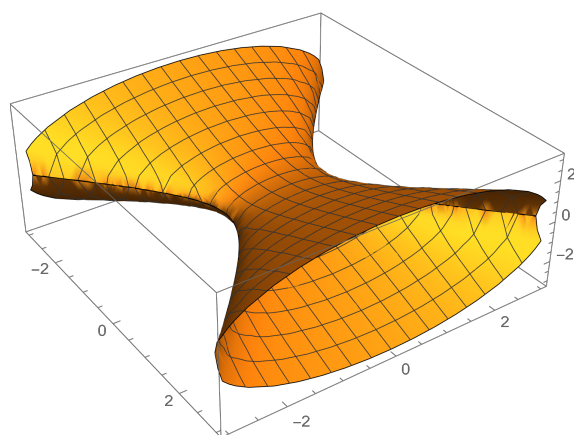
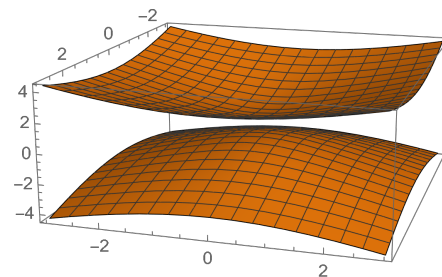


Figura 3.1: Espacio de las soluciones para el sistema de orden 2 con variables $d2 = 1, t1 = 0.5 + i\sqrt{2}/2, t2 = -0.5$ y $t3 = 0.5 + i\sqrt{2}/2$



(a) Hiperboloide.



(b) Paraboloid.

Figura 3.2: Ejemplo de otros tipos de superficie cuadrática

Para comprobar que la figura es un elipsoide se debe cumplir que la matriz 3.10 es definida positiva. Es decir se deben cumplir las siguientes inecuaciones.

$$A > 0$$

$$A \cdot B - (F/2)^2 > 0$$

$$A \cdot B \cdot C + D \cdot E \cdot F/4 - A \cdot (D/2)^2 - B \cdot (E/2)^2 - C \cdot (F/2)^2 > 0$$

Si a demás se toman en cuenta las restricciones de entrada para los parámetros d_2 , τ_{1i} , τ_{2i} , τ_{3i} ,

$$d_2 > 0, d_2 \in \mathbb{N}$$

$$\tau_{1i} > 0$$

$$\tau_{1i}\tau_{3i} > \tau_{2i}^2$$

Entonces al evaluar el sistema en Wolfram Mathematica, se obtiene que las condiciones para que se cumpla la propiedad de la forma elipsoide.

$$d_2 > 0 \& \left(\left(\tau_{2i} < 0 \& \tau_{1i} > 0 \& \tau_{3i} > \frac{\tau_{2i}^2}{\tau_{1i}} \right) | \left(\tau_{2i} == 0 \& \tau_{1i} > 0 \& \tau_{3i} > 0 \right) | \left(\tau_{2i} > 0 \& \tau_{1i} > 0 \& \tau_{3i} > \frac{\tau_{2i}^2}{\tau_{1i}} \right) \right) \quad (3.11)$$

Esto significa que si d_2 y τ_{1i} son positivos, para cualquier valor de τ_{2i} se cumple la propiedad si el producto entre τ_{1i} y τ_{3i} sea mayor a τ_{2i}^2 , las cuales son las condiciones de entrada para dichos parámetros. Entonces se puede demostrar que el espacio de las soluciones para éstas forma un elipsoide y es por ende acotado. Si a demás se toma en cuenta que todas las transformaciones utilizadas para reducir variables son lineales, entonces esta conclusión es válida para las 6 variables.

3.3. Determinando los Límites del Espacio

Sabiendo que el espacio de las soluciones es acotado y tiene forma de elipsoide, el siguiente paso es determinar las dimensiones de éste. Se necesita encontrar un prisma rectangular que contenga al elipsoide encontrando los valores máximos y mínimos de cada una de las variables siendo evaluadas. Las cuales son en este caso a_{12} , a_{13} y a_{14} .

Lo que se busca es un prisma rectangular que contenga al elipsoide. Este prisma se puede construir buscando los planos tangentes de la figura que son, a su vez, tangentes a los ejes del espacio. Entonces la forma de encontrar la dimensión máxima en cada eje del elipsoide es encontrando los puntos en que la normal de este tenga la misma dirección que los ejes. Es decir, si la función que define al elipsoide es $F(a_{12}, a_{13}, a_{14})$, encontrar los puntos en donde:

$$\Delta F = \left[\frac{dF}{da_{12}}, \frac{dF}{da_{13}}, \frac{dF}{da_{14}} \right]$$

$$\Delta F = [1, 0, 0]$$

$$\Delta F = [-1, 0, 0]$$

$$\Delta F = [0, 1, 0]$$

$$\Delta F = [0, -1, 0]$$

$$\Delta F = [0, 0, 1]$$

$$\Delta F = [0, 0, -1]$$

Este proceso de búsqueda de máximos y mínimos se puede ver más claramente en la figura 3.3. Sin embargo, hay un problema y este es que la magnitud del gradiente sin normalizar no es un valor unitario, sino un número que depende de la curvatura de la figura en ese punto. Si se intenta obtener la normal igualando al valor unitario se obtendrá un punto que no pertenece a la superficie, como se ve en el ejemplo de la figura 3.4. En este ejemplo se intentaron obtener los máximos y mínimos de la figura igualando el gradiente a un vector unitario y se obtuvieron los 4 puntos

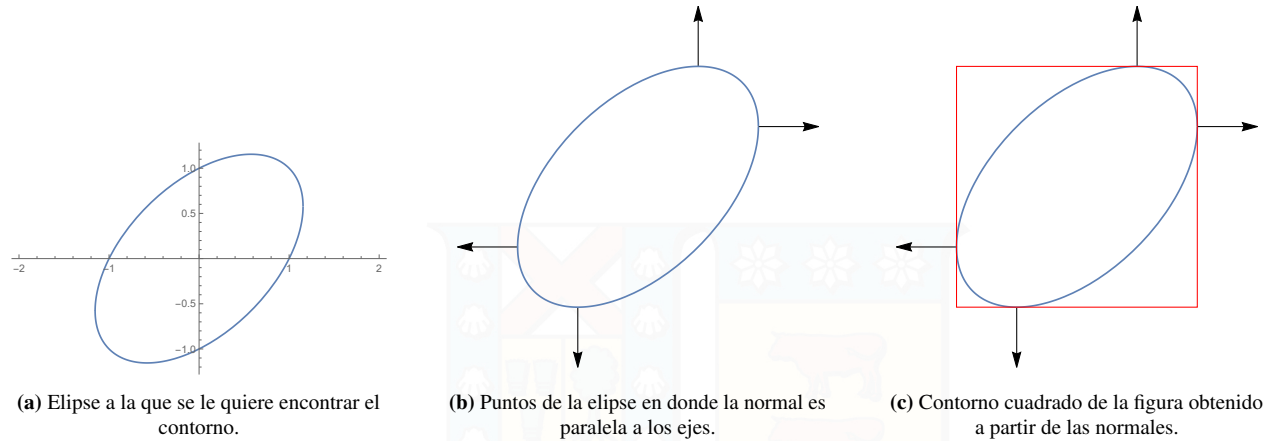


Figura 3.3: Representación en 2 dimensiones del proceso para encontrar máximos y mínimos.

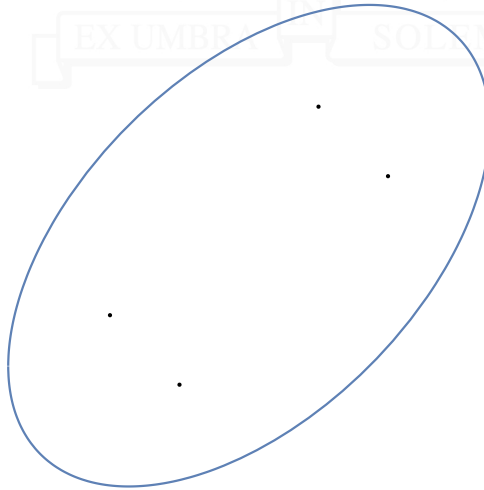


Figura 3.4: Ejemplo de la elipse 3.3 con los puntos cuya gradiente equivale a un valor unitario.

marcados dentro de la figura. La solución a este problema es evaluar la función de superficie en vez de las componentes no nulas del gradiente, como se muestra a continuación:

$$\begin{aligned} \left[F, \frac{dF}{da_{13}}, \frac{dF}{da_{14}} \right] &= [0, 0, 0] \\ \left[\frac{dF}{da_{12}}, F, \frac{dF}{da_{14}} \right] &= [0, 0, 0] \\ \left[\frac{dF}{da_{12}}, \frac{dF}{da_{13}}, F \right] &= [0, 0, 0] \end{aligned}$$

La primera ecuación tiene como soluciones los puntos en donde a_{12} es máximo y mínimo, lo mismo pasa para a_{13} y a_{14} en las dos ecuaciones siguientes. Se puede ver un ejemplo de una superficie que corresponde a una variedad con sus mínimos y máximos marcados utilizando este método en la figura 3.5.

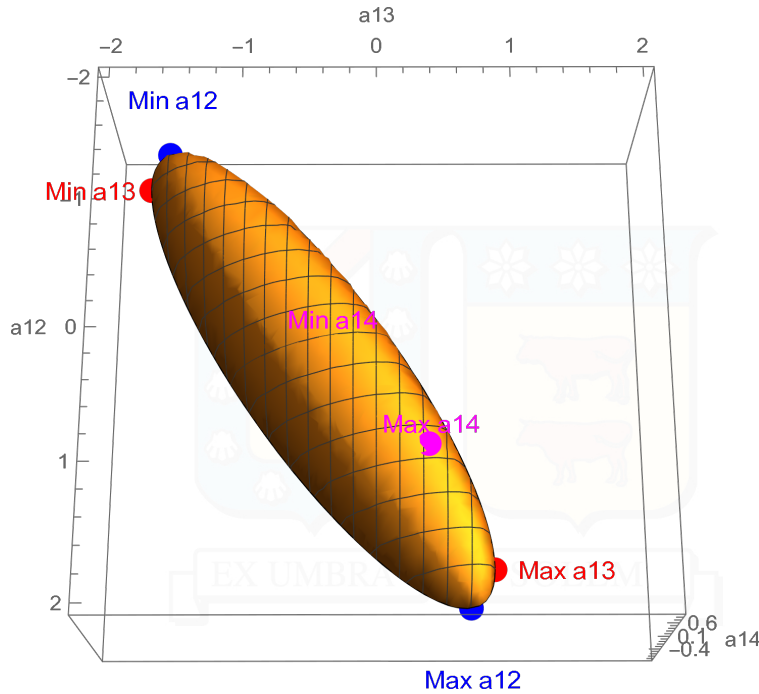


Figura 3.5: Puntos máximos y mínimos de cada eje obtenidos con el método anteriormente descrito para una superficie de parámetros $d2 = 3$, $\tau_1 = 4i/5 + 7/5$, $\tau_2 = 3i/5 + 9/5$ y $\tau_3 = 6i/5 + 3/5$.

3.4. Proceso de Búsqueda

Con los límites del espacio obtenidos, se puede proceder a realizar la búsqueda. Debido a que se requiere asegurar de que el resultado pertenece a los racionales, se realiza la búsqueda subdividiendo el espacio en valores discretos. La diferencia entre los espacios está dada como una potencia de 10 cuyo exponente está dado por el usuario. Es decir, el espacio está subdividido en cubos de lado 10^{-n} , en donde n es el nivel de precisión de la búsqueda que determina el usuario.

Sin embargo, existe todavía el problema que presentan los errores de punto flotante. Estos son errores de cuantización de números racionales en un espacio de memoria. Un número racional que puede expresarse con un número finito de dígitos en un sistema decimal, puede generar un número periódico en binario. También se puede producir este error en un número racional con más valores decimales que los que puede representar el equipo. Al colocar estos números en una ranura de memoria no se puede guardar todo el valor del número y este se trunca, lo cual produce un error entre el valor guardado y el valor representado. Estos errores son el principal obstáculo en la resolución del problema y obligan a estructurar el programa de tal manera de que se entreguen resultados completamente exactos.

Entre las varias modificaciones que proporciona el kernel de Sage Math a Python, se cuenta con una clase `sage.rings.rational.Rational` que representa una fracción. Mientras que Python normalmente traspasa una fracción a un valor flotante, Sage Math mantiene el numerador y denominador originales intactos. Esto permite trabajar con enteros dentro de fracciones al realizar el proceso de búsqueda, lo cual elimina completamente los errores de aproximación y de punto flotante.

Se diseñaron dos métodos de búsqueda para encontrar soluciones en el espacio de la superficie. Estos métodos de búsqueda buscará en primer lugar una igualdad exacta con la función de superficie y 0. Luego realiza la transformación inversa de las variables a_{23} (3.6), a_{24} (3.3) y a_{34} (3.8), para evaluar los 6 puntos en las ecuaciones originales 2.1, 2.2 y 2.3. Si un resultado cumple con todos estos requisitos, entonces es retornado como válido.

3.4.1. Método "full scan"

El primer método realiza una evaluación de todos los puntos del espacio, por esto llamado "full scan". En este método se toman todos los puntos entre máximos y mínimos y se evalúan en la ecuación de superficie 3.1. Este método tiene la ventaja de que garantiza encontrar todas las soluciones si es que existen para un cierto nivel de precisión. Sin embargo tiene como desventaja si alta cantidad de cálculos y evaluaciones que a demás escala cubica-mente con el orden de entrada.

3.4.2. Método "elipsoid looping"

En segundo lugar se diseñó un método denominado "elipsoid looping" que es usado por defecto al evaluar la superficie en el espacio. Este utiliza las propiedades del espacio sean:

- La función de superficie es cuadrática.
- La función forma un elipsoide cerrado.
- Los valores dentro y fuera del elipsoide que marca el 0 de la función de superficie tienen signos opuestos.
- Al discretizarse, un elipsoide puede ser subdividido en un conjunto de elipses bidimensionales.
- Cada una de estas elipses tiene cuatro cuadrantes y la curvatura de cada cuadrante tiene una forma cuadrática predecible.

Se toman estas propiedades para generar un algoritmo que toma muestras de dos puntos diferentes en el espacio y los compara entre si de acuerdo a un conjunto de reglas. Tras comparar estos dos puntos se lleva uno de estos al valor del segundo, mientras que este se mueve una distancia determinada por el orden de evaluación en una dirección determinada por el resultado de la evaluación anterior.

Este método divide la superficie en una serie de elipses bidimensionales a lo largo de la variable a_{12} , evaluando cada elipse una tras otra en las variables a_{13} y a_{14} . Se crean dos puntos ubicados en la parte exterior del elipse entre el borde del primer y cuarto cuadrante. El primer punto se desplaza hacia el borde del elipse y entonces se evalúa el valor obtenido de ambos puntos en la ecuación de superficie de acuerdo a las siguientes reglas:

- Si la multiplicación del resultado de ambos puntos evaluados en la ecuación de superficie es mayor a 0 y el valor absoluto del resultado de del punto que va por delante es menor al subsiguiente, quiere decir que ambos puntos se encuentran en la misma parte del espacio (ya sea en el interior o exterior del elipse) y que al avanzar se están acercando a este. Se continua avanzando en la misma dirección.
- Si la multiplicación del valor en la ecuación de superficie es menor a 0, esto significa que uno de los puntos se encuentra en el interior del elipse y otro en el exterior. Se cambia la dirección de movimiento para el siguiente punto (se cambia la variable que se modifica de a_{13} a a_{14} o viceversa).
- Si la multiplicación del valor en la ecuación de superficie es igual a 0, esto significa que se ha encontrado una solución. Esta se comprueba, guarda y se realizan los mismos pasos que el ítem anterior.
- Si la multiplicación del resultado de ambos puntos evaluados en la ecuación de superficie es mayor a 0 y el valor absoluto del resultado de del punto que va por delante es mayor al subsiguiente, significa que los puntos siendo evaluados se están alejando de la superficie determinada por las soluciones. Esto sucede cuando el proceso de búsqueda pasa de un cuadrante a otro y requiere un cambio en la forma en que se decide a que dirección llegar.

La forma en que se determina que dirección ir depende del cuadrante del elipse en donde se realiza la búsqueda. Estas son para cada cuadrante:

- Primer cuadrante: se avanza en dirección a_{13} negativa y a_{14} positiva.
- Segundo cuadrante: se avanza en dirección a_{13} negativa y a_{14} negativa.
- Tercer cuadrante: se avanza en dirección a_{13} positiva y a_{14} negativa.
- Cuarto cuadrante: se avanza en dirección a_{13} positiva y a_{14} positiva.

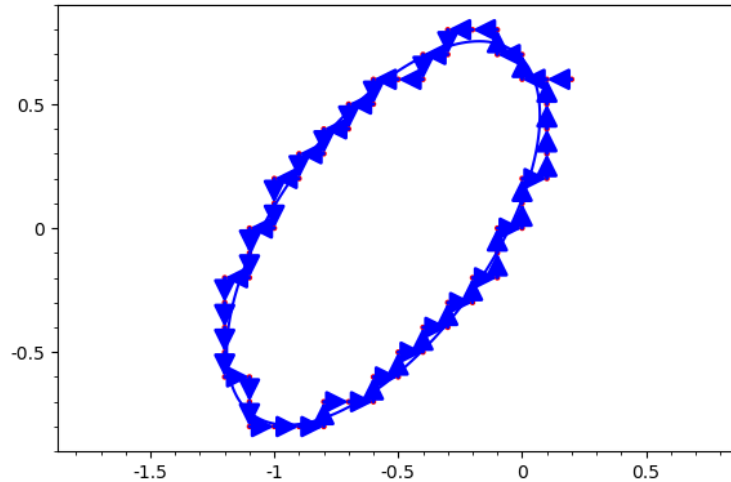


Figura 3.6: Gráfico de puntos evaluados en una sección transversal de un elipsoide. Se muestran con las flechas azules la secuencia en que fueron evaluados los puntos rojos.

Al evaluar puntos usando estas reglas se genera una serie de puntos evaluados que giran al rededor del área de posibles soluciones del elipsoide, como se muestra en la figura 3.6. Al terminar de evaluar una elipse se continua con la siguiente utilizando como punto de partida el último punto evaluado movido en dirección a 13 positiva hasta asegurarse de que este se encuentra en la posición de partida necesaria fuera del elipse en el primer cuadrante. Este método es más rápido que el método anterior "full scan" a demás de escalar cuadrática-mente en vez de cubica-mente al solo evaluar la superficie de las soluciones en vez de todo el espacio. Sin embargo existe la posibilidad de que este método falle en encontrar algunas soluciones en ciertos casos.

Vale mencionar que existe aún la rara posibilidad de que al ingresar como parámetros de entrada a la superficie números irracionales se obtengan falsos positivos o negativos. Por esto mismo queda a responsabilidad del usuario de juzgar la validez de los resultados. Por último, el programa arroja resultados más fiables cuando se le entregan los parámetros de entrada en formato de fracción en vez de decimales.

3.5. Funcionamiento del programa

A continuación se presenta en pseudocódigo el proceso de búsqueda en el espacio de las soluciones para el problema.

3.6. Ejemplo

Considere la superficie abeliana A de matriz de períodos Π_A :

$$\begin{bmatrix} 1 & 0 & 4/5*w + 7/5 & 3/5*w + 9/5 \\ 0 & 3 & 3/5*w + 9/5 & 6/5*w + 3/5 \end{bmatrix}$$

w es el número complejo i , es decir $w^2 = -1$.

El primer paso es crear el objeto de la superficie

```
1 superficie = Superficie2(3,4*I/5+7/5,3*I/5+9/5,6*I/5+3/5)
```

Partiendo con una búsqueda en el espacio partido cada 1/10 se utiliza el método `.resolucion(1)`

```
1 superficie.resolucion(1, method="full_scan")
```

Algorithm 1 Algoritmo de búsqueda

```

 $n \leftarrow \text{tamaño\_del\_paso}$ 
 $\text{step} \leftarrow 1/10^n$ 
 $\text{contador} \leftarrow 0$ 
 $x \leftarrow \text{min\_a}_{12}$ 
while  $x \leq \text{max\_a}_{12}$  do
   $y \leftarrow \text{min\_a}_{13}$ 
  while  $y \leq \text{max\_a}_{13}$  do
     $z \leftarrow \text{min\_a}_{14}$ 
    while  $z \leq \text{max\_a}_{14}$  do
      if  $(x,y,z)$  pertenecen a la superficie then
         $\text{variables} \leftarrow \text{transformacion inversa}(x,y,z)$ 
        if  $\text{variables}$  son solución de las ecuaciones then
           $\text{print}(\text{variables})$ 
           $\text{contador}++$ 
        end if
      end if
       $z \leftarrow z + \text{step}$ 
    end while
     $y \leftarrow y + \text{step}$ 
  end while
   $x \leftarrow x + \text{step}$ 
end while
 $\text{print}(\text{contador})$ 

```

La respuesta a este método será un gráfico como el de la figura 3.7 y una lista con los resultados encontrados como la mostrada en 3.8. Se lograron comprobar la validez de las tuplas de soluciones mostradas en estas imágenes.

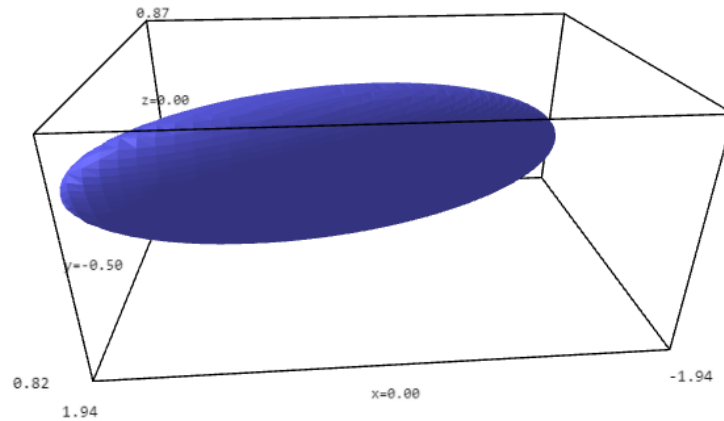


Figura 3.7: Gráfico de la superficie del ejemplo.

a12	a13	a14	a23	a24	a34
-9 / 5	-9 / 5	-3 / 5	27 / 10	12 / 5	3 / 2
-9 / 5	-8 / 5	-3 / 5	21 / 10	9 / 5	9 / 10
-3 / 2	-9 / 5	-3 / 5	27 / 10	12 / 5	9 / 5
-3 / 2	-3 / 2	0 / 1	3 / 1	3 / 2	3 / 2
-3 / 2	-13 / 10	-3 / 5	6 / 5	9 / 10	3 / 10
-3 / 2	-1 / 1	0 / 1	3 / 2	0 / 1	0 / 1
-9 / 10	-8 / 5	-3 / 5	21 / 10	9 / 5	9 / 5
-9 / 10	-9 / 10	-3 / 5	0 / 1	-3 / 10	-3 / 10
-3 / 5	-3 / 5	3 / 5	3 / 2	-6 / 5	3 / 10
-3 / 5	-2 / 5	3 / 5	9 / 10	-9 / 5	-3 / 10
-3 / 10	-13 / 10	-3 / 5	6 / 5	9 / 10	3 / 2
-3 / 10	-3 / 5	-3 / 5	-9 / 10	-6 / 5	-3 / 5
-3 / 10	-3 / 5	3 / 5	3 / 2	-6 / 5	3 / 5
-3 / 10	-1 / 10	3 / 5	0 / 1	-27 / 10	-9 / 10
0 / 1	-1 / 1	0 / 1	3 / 2	0 / 1	3 / 2
0 / 1	0 / 1	0 / 1	-3 / 2	-3 / 1	-3 / 2
3 / 10	-9 / 10	-3 / 5	0 / 1	-3 / 10	9 / 10
3 / 10	-2 / 5	-3 / 5	-3 / 2	-9 / 5	-3 / 5
3 / 10	-2 / 5	3 / 5	9 / 10	-9 / 5	3 / 5
3 / 10	3 / 10	3 / 5	-6 / 5	-39 / 10	-3 / 2
3 / 5	-3 / 5	-3 / 5	-9 / 10	-6 / 5	3 / 10
3 / 5	-2 / 5	-3 / 5	-3 / 2	-9 / 5	-3 / 10
9 / 10	-1 / 10	3 / 5	0 / 1	-27 / 10	3 / 10
9 / 10	3 / 5	3 / 5	-21 / 10	-24 / 5	-9 / 5
3 / 2	0 / 1	0 / 1	-3 / 2	-3 / 1	0 / 1
3 / 2	3 / 10	3 / 5	-6 / 5	-39 / 10	-3 / 10
3 / 2	1 / 2	0 / 1	-3 / 1	-9 / 2	-3 / 2
3 / 2	4 / 5	3 / 5	-27 / 10	-27 / 5	-9 / 5
9 / 5	3 / 5	3 / 5	-21 / 10	-24 / 5	-9 / 10
9 / 5	4 / 5	3 / 5	-27 / 10	-27 / 5	-3 / 2

30

Figura 3.8: Lista de soluciones para la superficie del ejemplo.

Por otra parte se puede llamar al método "ellipsoid_looping" llamándolo como parámetro de entrada o, en caso de no ingresar ningún método se usará este por defecto. Este método retorna un gráfico como el del proceso anterior 3.7 y una lista de resultados 3.9.

```
superficie.resolucion(1)
```

a12	a13	a14	a23	a24	a34
-9 / 5	-8 / 5	-3 / 5	21 / 10	9 / 5	9 / 10
-3 / 2	-3 / 2	0 / 1	3 / 1	3 / 2	3 / 2
-3 / 2	-9 / 5	-3 / 5	27 / 10	12 / 5	9 / 5
-3 / 2	-13 / 10	-3 / 5	6 / 5	9 / 10	3 / 10
-3 / 2	-1 / 1	0 / 1	3 / 2	0 / 1	0 / 1
-9 / 10	-8 / 5	-3 / 5	21 / 10	9 / 5	9 / 5
-9 / 10	-9 / 10	-3 / 5	0 / 1	-3 / 10	-3 / 10
-3 / 5	-2 / 5	3 / 5	9 / 10	-9 / 5	-3 / 10
-3 / 5	-3 / 5	3 / 5	3 / 2	-6 / 5	3 / 10
-3 / 10	-1 / 10	3 / 5	0 / 1	-27 / 10	-9 / 10
-3 / 10	-3 / 5	3 / 5	3 / 2	-6 / 5	3 / 5
-3 / 10	-3 / 5	-3 / 5	-9 / 10	-6 / 5	-3 / 5
0 / 1	-1 / 1	0 / 1	3 / 2	0 / 1	3 / 2
0 / 1	0 / 1	0 / 1	-3 / 2	-3 / 1	-3 / 2
3 / 10	-2 / 5	3 / 5	9 / 10	-9 / 5	3 / 5
3 / 10	-9 / 10	-3 / 5	0 / 1	-3 / 10	9 / 10
3 / 10	-2 / 5	-3 / 5	-3 / 2	-9 / 5	-3 / 5
3 / 5	-3 / 5	-3 / 5	-9 / 10	-6 / 5	3 / 10
3 / 5	-2 / 5	-3 / 5	-3 / 2	-9 / 5	-3 / 10
9 / 10	-1 / 10	3 / 5	0 / 1	-27 / 10	3 / 10
9 / 10	3 / 5	3 / 5	-21 / 10	-24 / 5	-9 / 5
3 / 2	3 / 10	3 / 5	-6 / 5	-39 / 10	-3 / 10
3 / 2	0 / 1	0 / 1	-3 / 2	-3 / 1	0 / 1
3 / 2	1 / 2	0 / 1	-3 / 1	-9 / 2	-3 / 2
3 / 2	4 / 5	3 / 5	-27 / 10	-27 / 5	-9 / 5
9 / 5	3 / 5	3 / 5	-21 / 10	-24 / 5	-9 / 10

26 soluciones.

Figura 3.9: Lista de soluciones para la superficie del ejemplo usando el método "ellipsoid looping".

4 | Conclusiones y Trabajo a Futuro

En el presente trabajo se ha logrado idear una forma de simplificar el problema presentado a un sistema de 3 variables. Se ha demostrado, además, que las soluciones de este sistema y valores posibles de las 6 variables en los reales son acotados y forman un elipsoide. A partir de este sistema se ideó un método de búsqueda en el espacio para buscar soluciones.

A futuro se trabajará en un método para buscar resultados en variedades A de dimensión 3. Estas variedades entregan los parámetros para el sistema de ecuaciones en una matriz de entrada Π_A

$$\Pi_A = \begin{pmatrix} 1 & 0 & 0 & \tau_{11} & \tau_{12} & \tau_{13} \\ 0 & d_2 & 0 & \tau_{12} & \tau_{22} & \tau_{23} \\ 0 & 0 & d_3 & \tau_{13} & \tau_{23} & \tau_{33} \end{pmatrix},$$

de tamaño 3×6 . Lo que da origen a un sistema no lineal de 15 variables y 9 ecuaciones. Las soluciones consisten en tuplas de la forma

$$(a_{ij})_{1 \leq i < j \leq 6} \in \mathbb{Q}^{15}.$$

5 | ANEXOS

5.1. Código

El programa está basado en una clase llamada "Superficie 2", que se inicializa con los valores de d_2 , τ_1 , τ_2 y τ_3 . Los parámetros de esta clase son las partes reales e imaginarias de las variables de la subvariedad, junto con los valores necesarios para construir la ecuación 3.1.

El código está hecho para y fue probado en el kernel de SageMath 9.0 de Python.

```

1 class Superficie2:
2     def __init__(self,d2,t1,t2,t3):
3         self.d2 = d2
4         self.t1r = t1.real()
5         self.t1i = t1.imag()
6         self.t2r = t2.real()
7         self.t2i = t2.imag()
8         self.t3r = t3.real()
9         self.t3i = t3.imag()
10        self.aa = (-self.t2i*self.t2i*self.t3i + self.t2r*self.t2r*self.t3i + self.t1i*self.t3i*
self.t3i - 2*self.t2i*self.t2r*self.t3r + self.t1i*self.t3r*self.t3r)/(self.t3i*d2)
11        self.bb = d2
12        self.cc = d2*self.t1i/self.t3i
13        self.dd = -2*d2*self.t2i/self.t3i
14        self.ee = 2*(self.t2i*self.t2r - self.t1i*self.t3r)/self.t3i
15        self.ff = 2*(self.t2i*self.t3r - self.t2r*self.t3i)/self.t3i
16        self.gg = (self.t2i*self.t3r - self.t2r*self.t3i)/self.t3i
17        self.hh = d2
18        self.ii = -d2*self.t2i/self.t3i

```

El primer método que contiene la clase es utilizado para comprobar que la superficie generada es un elipsoide y la matriz imaginaria de entrada es definida positiva. Se llama al intentar hacer una búsqueda de soluciones para el sistema para comprobar que el sistema es válido.

```

1 #comprueba que la matriz de la superficie sea definida positiva
2 #OUTPUT: True si la matriz del poligono es definida positiva. False si no lo es
3 def selfCheck(self):
4     if(self.aa<=0):
5         return False
6     if(self.aa*self.bb-pow(self.ff/2,2)<=0):
7         return False
8     if(self.aa*self.bb*self.cc + self.dd*self.ee*self.ff/4 - self.bb*pow(self.ee/2,2) - self.
aa*pow(self.dd/2,2) - self.cc*pow(self.ff/2,2) <= 0):
9         return False
10    if(self.t1i<=0):
11        return False
12    if(self.t1i*self.t3i-pow(self.t2i,2)<=0):
13        return False
14    return True

```

El segundo método contiene la ecuación de la superficie y se usa para comprobar que una tupla a_{12} , a_{13} y a_{14} pertenezca a esta.

```

1 #metodo que calcula el valor de la funcion de la superficie debe igualarse a 0 para pertenecer
  a esta
2 #INPUT: Los valores a12, a13 y a14 que se estan evaluando en la ecuacion dada por los
  parametros de la subvariedad
3 #OUTPUT: El valor de la funcion de la superficie del elipsoide
4 def calculo(self,a12,a13,a14):
5     return float(a12*a12*self.aa + a13*a13*self.bb + a14*a14*self.cc + a13*a14*self.dd + a12*
      a14*self.ee + a12*a13*self.ff + a12*self.gg + a13*self.hh + a14*self.ii)

```

El tercer método retorna la ecuación de superficie como una función de variables simbólicas 'var'.

```

1 #metodo que retorna la ecuacion de superficie con variables 'var' a12, a13 y a14
2 #OUTPUT: La ecuacion de superficie eq(a12,a13,a14)
3 def ecuacion(self):
4     a12,a13,a14 = var('a12,a13,a14')
5     return a12*a12*self.aa + a13*a13*self.bb + a14*a14*self.cc + a13*a14*self.dd + a12*a14*
      self.ee + a12*a13*self.ff + a12*self.gg + a13*self.hh + a14*self.ii

```

El siguiente método es utilizado para encontrar los límites máximos y mínimos de la superficie. Retorna un arreglo de 3*2*3 con las coordenadas de los puntos en donde se encuentran los valores mínimos y máximos de a_{12} , a_{13} y a_{14} . En ese orden.

```

1 #metodo para encontrar minimo y maximo en cada eje de la figura
2 #retorna: arreglo 3*2*3 con las coordenadas de los maximos y minimos de a12,a13 y a14 [[a12min,
  a12max],[a13min,a13max],[a14min,a14max]]
3 #funciona encontrando una gradiente de la figura que este en la direccion de cada eje.
4 #para encontrar cada min y max se evalua la derivada parcial de cada variable que no se esta
  buscando y se iguala a 0
5 #junto con la funcion de la superficie.
6 def bounds(self):
7     var('x y z')
8     minmaxx = solve((x^2*self.aa + y^2*self.bb + z^2*self.cc + y*z*self.dd + x*z*self.ee + x*y
      *self.ff + x*self.gg + y*self.hh + z*self.ii==0,
9         2*y*self.bb + z*self.dd + x*self.ff + self.hh==0,
10        2*z*self.cc + y*self.dd + x*self.ee + self.ii==0),x,y,z)
11     minmaxy = solve((2*x*self.aa + z*self.ee + y*self.ff + self.gg==0,
12        x^2*self.aa + y^2*self.bb + z^2*self.cc + y*z*self.dd + x*z*self.ee + x*y
      *self.ff + x*self.gg + y*self.hh + z*self.ii==0,
13        2*z*self.cc + y*self.dd + x*self.ee + self.ii==0),x,y,z)
14     minmaxz = solve((2*x*self.aa + z*self.ee + y*self.ff + self.gg==0,
15        2*y*self.bb + z*self.dd + x*self.ff + self.hh==0,
16        x^2*self.aa + y^2*self.bb + z^2*self.cc + y*z*self.dd + x*z*self.ee + x*y
      *self.ff + x*self.gg + y*self.hh + z*self.ii==0),x,y,z)
17     limitesx = sorted([[minmaxx[0][0].rhs(),minmaxx[0][1].rhs(),minmaxx[0][2].rhs()],
18        [minmaxx[1][0].rhs(),minmaxx[1][1].rhs(),minmaxx[1][2].rhs()]],key = itemgetter(0)
19 )
20     limitesy = sorted([[minmaxy[0][0].rhs(),minmaxy[0][1].rhs(),minmaxy[0][2].rhs()],
21        [minmaxy[1][0].rhs(),minmaxy[1][1].rhs(),minmaxy[1][2].rhs()]],key = itemgetter(1)
22 )
23     limitesz = sorted([[minmaxz[0][0].rhs(),minmaxz[0][1].rhs(),minmaxz[0][2].rhs()],
24        [minmaxz[1][0].rhs(),minmaxz[1][1].rhs(),minmaxz[1][2].rhs()]],key = itemgetter(2)
25 )
26     return [limitesx,limitesy,limitesz]

```

El método reversa es usado para obtener el arreglo de las 6 variables $[a_{12}, a_{13}, a_{14}, a_{23}, a_{24}, a_{34}]$ correspondiente a las variables de búsqueda $[a_{12}, a_{13}, a_{14}]$. Utiliza las ecuaciones 3.3, 3.6 y 3.8.

```

1 #metodo para deshacer las simplificaciones. retorna las 6 variables originales
2 #INPUT: las variables a12, a13 y a14 para una subvariedad dada.
3 #Output: un arreglo con las variables [a12,a13,a14,a23,a24,a34]
4 def reversa(self,a12,a13,a14):
5     a24 = -self.d2 -a13*self.d2
6     a23 = (a14*self.d2*self.t1i - self.d2*self.t2i - 2*a13*self.d2*self.t2i + 2*a12*self.t2i*
      self.t2r - a12*self.t1r*self.t3i -
7     a12*self.t1i*self.t3r)/self.t3i
8     a34 = (a14*self.d2*self.t1r*self.t3i - a12*pow(self.t2i,2)*self.t3i - self.d2*self.t2r*
      self.t3i - 2*a13*self.d2*self.t2r*self.t3i +

```

```

9  a12*pow(self.t2r,2)*self.t3i + a12*self.t1i*pow(self.t3i,2) - a14*self.d2*self.t1i*self.t3r +
    self.d2*self.t2i*self.t3r +
10  2*a13*self.d2*self.t2i*self.t3r - 2*a12*self.t2i*self.t2r*self.t3r + a12*self.t1i*pow(self.t3r,2)
    )/(self.d2*self.t3i)
11  return [a12,a13,a14,a23,a24,a34]

```

El método plt genera un gráfico de la superficie. El gráfico es retornado como un objeto.

```

1  #Grafica en 3D la sub variedad en las coordenadas de a12, a13 y a14
2  #INPUT: El arreglo 3*2 de los limites
3  #Output: Un grafico 3D.
4  def plt(self,limites):
5      var('a12 a13 a14')
6      return implicit_plot3d(self.aa*a12^2 + self.bb*a13^2 + self.cc*a14^2 + a13*a14*self.dd +
    a12*a14*self.ee + a12*a13*self.ff + a12*self.gg + a13*self.hh + a14*self.ii==0,
7      (a12,limites[0][0],limites[0][1]),(a13,limites[1][0],limites[1][1]),
    (a14,limites[2][0],limites[2][1]))

```

El método "comprobación" es usado para evaluar una tupla de las variables (a_{12} , a_{13} , a_{14} , a_{23} , a_{24} , a_{34}) en el sistema de ecuaciones. Retorna "True" en caso de que la tupla sea solución y "False" en caso contrario.

```

1  #Evalua las 3 ecuaciones originales con los valores de de variables entregados en los
    parametros de la sub variedad.
2  #INPUT: a12,a13,a14,a23,a24,a34 cada uno por si mismo. las variables a evaluar.
3  #Funciona mejor al ingresarlas en forma fraccional
4  #OUTPUT: entrega True si las 3 ecuaciones son exactamente iguales a 0. False en cualquier otro
    caso.
5      a = SR(a23)
6      b = SR(a24)
7      c = SR(a34)
8      eq1 = self.d2*a13+a24+self.d2
9      eq2 = (((self.t1r+I*self.t1i)*(self.t3r+I*self.t3i)-(self.t2r+I*self.t2i)^2)*a12
10     -self.d2*(self.t1r+I*self.t1i)*a14 + self.d2*(self.t2r+I*self.t2i)*a13
11     -(self.t2r+I*self.t2i)*a24 + (self.t3r+I*self.t3i)*a23 + self.d2*a34)
12     eq3 = a14*a23-a13*a24+a12*a34
13     #se comprueba que las 3 ultimas variables sean numeros racionales y que no tengan simbolos
    como raices o
14     #expresiones sinusoidales provenientes de los parametros de entrada que pueden ser
    irracionales.
15     if(not a.is_numeric()or not b.is_numeric() or not c.is_numeric()):
16         return False
17     if(not eq1.is_zero()):
18         return False
19     if(not eq2.is_zero()):
20         return False
21     if(not eq3.is_zero()):
22         return False
23     return True

```

Finalmente el último método llamado "resolucion" realiza una búsqueda en el espacio utilizando los métodos anteriores. Después de revisar que los parámetros de entrada sean correctos, se calculan los límites de la figura. A continuación se procede utilizar el mecanismo de búsqueda ingresado por el usuario. Cuando se encuentra una solución que satisface a la función de superficie, se utiliza el método "reversa" para obtener un arreglo $[a_{12}, a_{13}, a_{14}, a_{23}, a_{24}, a_{34}]$ correspondiente al punto. Las variables se evalúan con el método "comprobación" y si forman una solución se imprimen y se guardan en una cola de la librería "deque". Cuando el programa acaba se imprimen la cantidad de soluciones encontradas.

El método toma como parámetro de entrada un entero mayor o igual a 1, el cual define el orden de la discretización del espacio a evaluar. Por ejemplo: un valor de entrada 1 dividirá el espacio en 10^1 , por lo que al hacer la búsqueda se tomará una muestra del espacio para evaluar en la función cada 0.1. Un valor de 2 en 10^2 , etc. También se puede ingresar como parámetro de entrada el algoritmo de búsqueda deseado. El método imprime un gráfico de la superficie, los resultados encontrados y el número de soluciones encontradas.

```

1  #Realiza una evaluacion igualando a 0 la ecuacion de superficie y las 3 ecuaciones originales para
    encontrar soluciones.

```

```

2 #INPUT: el orden o cantidad de digitos significativos que tendra la cuadrícula en la que se
  subdividirá el espacio.
3 #orden: el tamaño de la discretización con que se toma cada muestra  $10^{\text{orden}}$ 
4 #method: el método que se usa para realizar la búsqueda. "ellipsoid_looping" es por defecto y usa
  el método inventando
5 #comparando los valores tomados con 2 punteros. El método "full_scan" realiza una búsqueda
  completa de todos los puntos
6 #entre los límites.
7 #point_limit: entrega un número máximo de iteraciones para el método "ellipsoid_looping" si no se
  entrega ninguno, se usa
8 #por defecto  $10000^{\text{orden}}$ 
9 #OUTPUT: retorna 0 al terminar. muestra el gráfico de la subvariedad e imprime por la consola
  todos los resultados obtenidos.
10 #Imprime a demás la cantidad de resultados encontrados.
11 def resolucion(self,orden, method = "ellipsoid_looping", point_limit = -1):
12     if(not self.selfCheck()):
13         print("Error en los parametros de la curva.")
14         return -1
15     if(orden<1| (not isinstance(orden,sage.rings.integer.Integer))):
16         print("Error en el parametro de entrada.")
17         return -1
18     spacing = 1/pow(10,orden)
19     nro_soluciones = 0
20     soluciones = deque()
21     lim = self.bounds()#se ordena el arreglo 3*2*3 limt en un arreglo 3*2 con los maximos y
  minimos de cada variable
22     #[[min a12, max a12],[min a13, max a13],[min a14, max a14]]
23     limites = [[lim[0][0][0],lim[0][1][0]], [lim[1][0][1],lim[1][1][1]], [lim[2][0][2],lim
  [2][1][2]]]
24     show(self.plt(limites))
25     limites = [[QQ(round(limites[0][0],orden)),QQ(round(limites[0][1],orden))],
26               [QQ(round(limites[1][0],orden)),QQ(round(limites[1][1],orden))],
27               [QQ(round(limites[2][0],orden)),QQ(round(limites[2][1],orden))]]
28     if(method=="full_scan"):
29         x = limites[0][0]
30         print('{:^7}|{:^7}|{:^7}|{:^7}|{:^7}|{:^7}|'.format("a12","a13","a14","a23","a24","
  a34"))
31         while x <= limites[0][1]:
32             y = limites[1][0]
33             while y <= limites[1][1]:
34                 z = limites[2][0]
35                 while z <= limites[2][1]:
36                     if(self.calculo(x,y,z).is_zero()):
37                         answer = self.reversa(x,y,z)
38                         if(self.comprobacion(x,y,z,answer[3],answer[4],answer[5])):
39                             nro_soluciones+=1
40                             soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
  [4],answer[5]])
41                     print('
  |{:^3}/{: ^3}|{:^3}/{: ^3}|{:^3}/{: ^3}|{:^3}/{: ^3}|{:^3}/{: ^3}|{:^3}/{: ^3}|'.format(x.numerator
  (),x.denominator(),y.numerator(),y.denominator(),z.numerator(),z.denominator(),QQ(answer[3]).
  numerator(),QQ(answer[3]).denominator(),QQ(answer[4]).numerator(),QQ(answer[4]).denominator(),
  QQ(answer[5]).numerator(),QQ(answer[5]).denominator()))
42                     z=z+spacing
43                     y=y+spacing
44                     x=x+spacing
45                 print(nro_soluciones)
46                 return soluciones
47             if(method == "ellipsoid_looping"):
48                 if(point_limit < 0):
49                     point_limit = pow(10000,orden)
50                 x = QQ(round(lim[0][0][0],orden))
51                 if(abs(x)>abs(lim[0][0][0])):
52                     x = x + spacing
53                 print('{:^7}|{:^7}|{:^7}|{:^7}|{:^7}|{:^7}|'.format("a12","a13","a14","a23","a24","
  a34"))
54                 p1=((QQ(round(lim[0][0][1],orden)),QQ(round(lim[0][0][2],orden))))#se toman muestras
  comparando los valores en el punto p1 y p2

```

```

55     p2=((QQ(round(lim[0][0][1],orden)),QQ(round(lim[0][0][2],orden))))#p1 va por delante
    de p2
56     n=0
57     #este loop mueve el punto 1 hasta el borde del circulo. al traspasarlo se devuelve al
    punto interior mas cercano al borde
58     while(self.calculo(x,p1[0],p1[1]).abs()<=self.calculo(x,p2[0],p2[1]).abs() and n<
    point_limit):
59         p2=p1
60         temp = list(p1)
61         temp[0] = temp[0]+spacing
62         temp[1] = temp[1]+spacing
63         p1=tuple(temp)
64         n=n+1
65     p1 = p2
66     #una vez se encontro el borde desplazandose en sentido a13+, se desplaza el punto 1 en
    sentido a14- hasta atravesar el circulo
67     #esto asegura que el punto de partida este en el cuadrante 2, en a13 positivo y a14
    positivo justo afuera del circulo
68     while(self.calculo(x,p1[0],p1[1]).abs()<=self.calculo(x,p2[0],p2[1]).abs() and n<
    point_limit):
69         p2=p1
70         temp = list(p1)
71         temp[1] = temp[1]-spacing
72         p1=tuple(temp)
73         n=n+1
74     p1=p2
75
76     while(x<limites[0][1][0] and n<point_limit):
77         k = 0#k marca el cuadrante en que se realiza la busqueda.
78         direccion = True#determina la variable que se esta usando para avanzar, ya sea a13
    o a14. cuando cambia hay un cambio de direccion
79         temp = list(p1)
80         temp[0] = temp[0]+spacing
81         p1=tuple(temp)
82         while(self.calculo(x,p1[0],p1[1]).abs()<=self.calculo(x,p2[0],p2[1]).abs() and
    self.calculo(x,p1[0],p1[1])*self.calculo(x,p2[0],p2[1])>0):
83             p2=p1
84             temp = list(p1)
85             temp[0] = temp[0]+spacing
86             p1=tuple(temp)
87             n=n+1
88         while(k<4 and n<point_limit):#el punto de partida deberia estar justo en el inicio
    del segundo cuadrante y se recorren en el sentido del reloj
89             p2=p1
90             temp = list(p1)
91             if(direccion):
92                 if(k==0):#el sistema esta en la parte inferior del circulo se va a la
    izquierda
93                     temp[0] = temp[0]-spacing
94                 elif(k==1):
95                     temp[1] = temp[1]-spacing
96                 elif(k==2):#el sistema esta en la parte superior del circulo, se va a la
    derecha
97                     temp[0] = temp[0]+spacing
98                 elif(k==3):
99                     temp[1] = temp[1]+spacing
100            else:
101                if(k==0):#acciones para la parte derecha del circulo
102                    temp[1] = temp[1]+spacing
103                elif(k==1):
104                    temp[0] = temp[0]-spacing
105                elif(k==2):
106                    temp[1] = temp[1]-spacing
107                elif(k==3):
108                    temp[0] = temp[0]+spacing
109            p1=tuple(temp)
110            n=n+1
111            if(self.calculo(x,p1[0],p1[1])*self.calculo(x,p2[0],p2[1])<=0):#significa que

```

```

hubo un cruce en la frontera del circulo o se encontro una solucion
112         direccion = not direccion
113         if(self.calculo(x,p1[0],p1[1]).is_zero()):
114             answer = self.reversa(x,p1[0],p1[1])
115             if(self.comprobacion(x,p1[0],p1[1],answer[3],answer[4],answer[5])):
116                 nro_soluciones +=1
117                 soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
118 [4],answer[5]])
119                 print('
120 |{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|'.format(x.numerator
121 (),x.denominator(),p1[0].numerator(),p1[0].denominator(),p1[1].numerator(),p1[1].denominator()
122 ,QQ(answer[3]).numerator(),QQ(answer[3]).denominator(),QQ(answer[4]).numerator(),QQ(answer[4])
123 .denominator(),QQ(answer[5]).numerator(),QQ(answer[5]).denominator()))
124             elif(self.calculo(x,p1[0],p1[1]).abs()>(self.calculo(x,p2[0],p2[1]).abs())):#
125 si el punto 1 es mayor al 2 entonces se esta alejando del
126 #circulo y se debe cambiar de cuadrante
127 #las siguientes comparaciones son usadas para evaluar un punto en la
128 diagonal
129 #del punto en donde se realiza el cambio de cuadrante. A veces al cambiar
130 de cuadrante se
131 #pierde un resultado justo en el punto evaluado por los ifs siguientes
132 if(k==0 and self.calculo(x,p1[0]+spacing,p1[1]+spacing).is_zero()):
133     answer = self.reversa(x,p1[0]+spacing,p1[1]+spacing)
134     if(self.comprobacion(answer[0],answer[1],answer[2],answer[3],answer
135 [4],answer[5])):
136         nro_soluciones +=1
137         soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
138 [4],answer[5]])
139         print('
140 |{:~3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|'.format(answer[0].
141 numerator(),answer[0].denominator(),answer[1].numerator(),answer[1].denominator(),answer[2].
142 numerator(),answer[2].denominator(),QQ(answer[3]).numerator(),QQ(answer[3]).denominator(),QQ(
143 answer[4]).numerator(),QQ(answer[4]).denominator(),QQ(answer[5]).numerator(),QQ(answer[5]).
144 denominator())
145         elif(k==1 and self.calculo(x,p1[0]+spacing,p1[1]-spacing).is_zero()):
146             answer = self.reversa(x,p1[0]+spacing,p1[1]-spacing)
147             if(self.comprobacion(answer[0],answer[1],answer[2],answer[3],answer
148 [4],answer[5])):
149                 nro_soluciones +=1
150                 soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
151 [4],answer[5]])
152                 print('
153 |{:~3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|'.format(answer[0].
154 numerator(),answer[0].denominator(),answer[1].numerator(),answer[1].denominator(),answer[2].
155 numerator(),answer[2].denominator(),QQ(answer[3]).numerator(),QQ(answer[3]).denominator(),QQ(
156 answer[4]).numerator(),QQ(answer[4]).denominator(),QQ(answer[5]).numerator(),QQ(answer[5]).
157 denominator())
158                 elif(k==2 and self.calculo(x,p1[0]-spacing,p1[1]-spacing).is_zero()):
159                     answer = self.reversa(x,p1[0]-spacing,p1[1]-spacing)
160                     if(self.comprobacion(answer[0],answer[1],answer[2],answer[3],answer
161 [4],answer[5])):
162                         nro_soluciones +=1
163                         soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
164 [4],answer[5]])
165                         print('
166 |{:~3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|{:^3}/{:~3}|'.format(answer[0].
167 numerator(),answer[0].denominator(),answer[1].numerator(),answer[1].denominator(),answer[2].
168 numerator(),answer[2].denominator(),QQ(answer[3]).numerator(),QQ(answer[3]).denominator(),QQ(
169 answer[4]).numerator(),QQ(answer[4]).denominator(),QQ(answer[5]).numerator(),QQ(answer[5]).
170 denominator())
171                     elif(k==3 and self.calculo(x,p1[0]-spacing,p1[1]+spacing).is_zero()):
172                         answer = self.reversa(x,p1[0]-spacing,p1[1]+spacing)
173                         if(self.comprobacion(answer[0],answer[1],answer[2],answer[3],answer
174 [4],answer[5])):
175                             nro_soluciones +=1
176                             soluciones.append([answer[0],answer[1],answer[2],answer[3],answer
177 [4],answer[5]])
178                             print('

```

```

148 |{:^3}/{:^3}|{:^3}/{:^3}|{:^3}/{:^3}|{:^3}/{:^3}|{:^3}/{:^3}|{:^3}/{:^3}|'.format(answer[0].
149 numerator(), answer[0].denominator(), answer[1].numerator(), answer[1].denominator(), answer[2].
150 numerator(), answer[2].denominator(), QQ(answer[3]).numerator(), QQ(answer[3]).denominator(), QQ(
151 answer[4]).numerator(), QQ(answer[4]).denominator(), QQ(answer[5]).numerator(), QQ(answer[5]).
152 denominator())
153
154 k+=1
155 #p1=p2
    x = x+spacing
    print(nro_soluciones, ' soluciones.')
    return soluciones
else:
    print("Error de metodo.")
    return -1

```

5.2. Demostración del Espacio Elipsoide en Wolfram Mathematica

En primera instancia se tienen las 3 ecuaciones originales.

$$eq1[a13_, a24_] = d2 + d2 * a13 + a24;$$

$$eq2[a12_, a13_, a14_, a23_, a24_, a34_] =$$

$$((t1r + I * t1i) * (t3r + I * t3i) - (t2r + I * t2i)^2) * a12 + d2 * (t2r + I * t2i) * a13 - d2 * (t1r + I * t1i) * a14 +$$

$$(t3r + I * t3i) * a23 - (t2r + I * t2i) * a24 + d2 * a34;$$

$$eq3[a12_, a13_, a14_, a23_, a24_, a34_] = a14 * a23 - a13 * a24 + a12 * a34;$$

Se simplifica a24 usando la primera.

$$a24 = a24 /. Solve[eq1[a13, a24] == 0, a24]$$

$$\{-d2 - a13d2\}$$

Se simplifica a23 usando la parte imaginaria de la segunda.

$$eq2i[a12_, a13_, a14_, a23_, a34_] = ComplexExpand[Im[eq2[a12, a13, a14, a23, a24, a34]]]$$

$$\{-a14d2t1i + d2t2i + 2a13d2t2i - 2a12t2it2r + a23t3i + a12t1rt3i + a12t1it3r\}$$

$$a23 = a23 /. Solve[eq2i[a12, a13, a14, a23, a34] == 0, a23]$$

$$\left\{ \frac{a14d2t1i - d2t2i - 2a13d2t2i + 2a12t2it2r - a12t1rt3i - a12t1it3r}{t3i} \right\}$$

Se simplifica a34 usando la parte real de la segunda.

$$eq2r[a12_, a13_, a14_, a34_] = ComplexExpand[Re[eq2[a12, a13, a14, a23, a24, a34]]]$$

$$\left\{ a_{34}d_2 - a_{14}d_2t_{1r} + a_{12}t_{2i}^2 + d_2t_{2r} + 2a_{13}d_2t_{2r} - a_{12}t_{2r}^2 - a_{12}t_{1i}t_{3r} + \frac{a_{14}d_2t_{1i}t_{3r}}{t_{3i}} \right. \\ \left. - \frac{d_2t_{2i}t_{3r}}{t_{3i}} - \frac{2a_{13}d_2t_{2i}t_{3r}}{t_{3i}} + \frac{2a_{12}t_{2i}t_{2r}t_{3r}}{t_{3i}} - \frac{a_{12}t_{1i}t_{3r}^2}{t_{3i}} \right\}$$

a34 = a34/.Solve[eq2r[a12, a13, a14, a34] == 0, a34]

$$\left\{ \frac{1}{d_2t_{3i}} \left(a_{14}d_2t_{1r}t_{3i} - a_{12}t_{2i}^2t_{3i} - d_2t_{2r}t_{3i} - 2a_{13}d_2t_{2r}t_{3i} + a_{12}t_{2r}^2t_{3i} + a_{12}t_{1i}t_{3i}^2 - a_{14}d_2t_{1i}t_{3r} + d_2t_{2i}t_{3r} \right. \right. \\ \left. \left. + 2a_{13}d_2t_{2i}t_{3r} - 2a_{12}t_{2i}t_{2r}t_{3r} + a_{12}t_{1i}t_{3r}^2 \right) \right\}$$

Simplify[eq3[a12, a13, a14, a23, a24, a34]]

$$\left\{ \frac{1}{d_2t_{3i}} \left(a_{14}d_2^2t_{1i} + a_{13}d_2^2t_{3i} - a_{14}d_2(d_2t_{2i} + 2a_{13}d_2t_{2i} - 2a_{12}t_{2i}t_{2r} + 2a_{12}t_{1i}t_{3r}) + a_{13}d_2(d_2t_{3i} \right. \right. \\ \left. \left. - 2a_{12}t_{2r}t_{3i} + 2a_{12}t_{2i}t_{3r}) + a_{12}(-d_2t_{2r}t_{3i} + d_2t_{2i}t_{3r} + a_{12}(-2t_{2i}^2t_{3i} + t_{2r}^2t_{3i} - 2t_{2i}t_{2r}t_{3r} + t_{1i}(t_{3i}^2 + t_{3r}^2))) \right) \right\}$$

A partir de esta expresión es posible obtener los parámetros de la ecuación polinomial.

aa[d2_, t1i_, t2r_, t2i_, t3r_, t3i_] = (-t2i^2 * t3i + t2r^2 * t3i - 2 * t2i * t2r * t3r + t1i * (t3i^2 + t3r^2))/(t3i * d2);
(*a12^2*)

bb[d2_] = d2; (*a13^2*)

cc[d2_, t1i_, t3i_] = d2 * t1i/t3i; (*a14^2*)

dd[d2_, t2i_, t3i_] = -2 * d2 * t2i/t3i; (*a13a14*)

ee[t1i_, t2r_, t2i_, t3r_, t3i_] = (2 * t2i * t2r - 2 * t1i * t3r)/(t3i); (*a12a14*)

ff[t2r_, t2i_, t3r_, t3i_] = (-2 * t2r * t3i + 2 * t2i * t3r)/(t3i); (*a12a13*)

gg[t2r_, t2i_, t3r_, t3i_] = (t2i * t3r - t2r * t3i)/t3i; (*a12*)

hh[d2_] = d2; (*a13*)

ii[d2_, t2i_, t3i_] = -d2 * t2i/t3i;

(*a14*)

Se utilizan estos parámetros para comprobar que la matriz del polinomio sea definida positiva, tomando en cuenta a demás los requisitos para los parámetros de entrada.

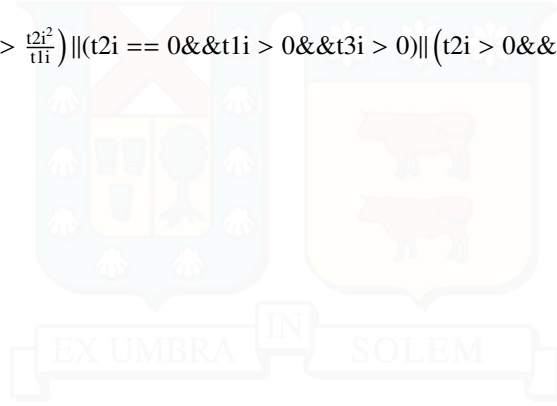
Reduce[aa[d2, t1i, t2r, t2i, t3r, t3i] > 0 &&

aa[d2, t1i, t2r, t2i, t3r, t3i] * bb[d2] - (ff[t2r, t2i, t3r, t3i]/2)^2 > 0 &&

aa[d2, t1i, t2r, t2i, t3r, t3i] * bb[d2] * cc[d2, t1i, t3i] +

$\text{ff}[t2r, t2i, t3r, t3i] * \text{dd}[d2, t2i, t3i] * \text{ee}[t1i, t2r, t2i, t3r, t3i]/4 -$
 $\text{bb}[d2] * (\text{ee}[t1i, t2r, t2i, t3r, t3i]/2)^2 - \text{aa}[d2, t1i, t2r, t2i, t3r, t3i] * (\text{dd}[d2, t2i, t3i]/2)^2 -$
 $\text{cc}[d2, t1i, t3i] * (\text{ff}[t2r, t2i, t3r, t3i]/2)^2 > 0 \&\& d2 > 0 \&\& t1i > 0 \&\& t1i * t3i - t2i^2 > 0,$
 $\{t3i\}, \text{Reals}]$

$d2 > 0 \&\& \left((t2i < 0 \&\& t1i > 0 \&\& t3i > \frac{t2i^2}{t1i}) \parallel (t2i == 0 \&\& t1i > 0 \&\& t3i > 0) \parallel (t2i > 0 \&\& t1i > 0 \&\& t3i > \frac{t2i^2}{t1i}) \right)$



Bibliografía

- [1] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, p. 120–126, feb 1978. [Online]. Available: <https://doi.org/10.1145/359340.359342> 1
- [2] G. Seroussi, “Elliptic curve cryptography,” in *1999 Information Theory and Networking Workshop (Cat. No.99EX371)*, 1999, pp. 41–. 1
- [3] A. Pizarro, “Criptografía post-cuántica y curvas elípticas,” Apuntes del Minicurso dictado en el Encuentro Nacional de Ingeniería Matemática, 2022, pontificia Universidad Católica de Chile. 1
- [4] J. M. Pollard, “Monte carlo methods for index computation (mod p),” *Mathematics of Computation*, vol. 32, no. 143, pp. 918–924, 1978. [Online]. Available: <http://www.jstor.org/stable/2006496> 1
- [5] P. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134. 1
- [6] D. J. Bernstein and T. Lange, “Post-quantum cryptography,” *Nature*, vol. 549, no. 7671, pp. 188–194, Sep 2017. [Online]. Available: <https://doi.org/10.1038/nature23461> 1
- [7] E. V. Flynn and Y. B. Ti, “Genus two isogeny cryptography,” Cryptology ePrint Archive, Paper 2019/177, 2019, <https://eprint.iacr.org/2019/177>. [Online]. Available: <https://eprint.iacr.org/2019/177> 1
- [8] W. Castryck and T. Decru, “An efficient key recovery attack on sidh,” Cryptology ePrint Archive, Paper 2022/975, 2022, <https://eprint.iacr.org/2022/975>. [Online]. Available: <https://eprint.iacr.org/2022/975> 1
- [9] E. Kani, “The existence of curves of genus two with elliptic differentials,” *Journal of Number Theory*, vol. 64, no. 1, pp. 130–161, 1997. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0022314X97921053> 1
- [10] R. Auffarth, H. Lange, and A. M. Rojas, “A criterion for an abelian variety to be non-simple,” *J. Pure Appl. Algebra*, vol. 221, no. 8, pp. 1906–1925, 2017. [Online]. Available: <https://doi.org/10.1016/j.jpaa.2016.10.020> 2
- [11] “Página web de sage math,” <https://www.sagemath.org/>. 2