

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES
UNIVERSIDAD POLITÉCNICA DE MADRID

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es
www.industriales.upm.es



POLITÉCNICA

INDUSTRIALES

05 TRABAJO FIN DE GRADO

Álvaro Morales Sánchez

TRABAJO FIN DE GRADO

DISEÑO INTELIGENTE DE DISPOSITIVOS MÉDICOS PERSONALIZADOS: EXPLORANDO LA EFICIENCIA DE LA SUPERELIPSE Y EL CASCO CONVEXO

SEPTIEMBRE 2023

Álvaro Morales Sánchez

DIRECTOR DEL TRABAJO FIN DE GRADO:
William Solórzano Requejo,
Andrés Díaz Lantada

*“No se puede conectar los puntos mirando hacia adelante;
solo puedes conectarlos mirando hacia atrás”.*

Steven Jobs

“No importa lo lento que vayas mientras no pares.”

Confucio

Agradecimientos

Me gustaría aprovechar esta oportunidad para agradecer todo el apoyo recibido durante el desarrollo de este trabajo.

A mis padres, por todo el cariño dado y educación impartida.

A mis abuelas y tía, ejemplos de sencillez y amor en mi día a día.

A mis amigos más cercanos, tanto del colegio como de San Alfonso, por las risas, momentos, historias y todo lo que hemos compartido, compartimos y compartiremos.

A mis compañeros de Scan F.C., por haber disfrutado no solo de la etapa universitaria a su lado, también haber compartido amistad y aventuras.

A José y Marta, ejemplos y consejeros en muchos aspectos de la vida.

A mis compañeros de UPM Racing y resto de compañeros con los que he compartido esta etapa universitaria, por el compañerismo, ayuda y conocimiento que he obtenido de ellos.

Quiero agradecer vuestra ayuda, por si no lo hubiese dicho antes, a todos aquellos que acogisteis mis quejas u os pusisteis en mi piel en momentos más delicados.

Por último, dar las gracias a mi tutor William. Su confianza, conocimiento y amabilidad son pilares de este trabajo. Eres un claro ejemplo de trabajo y desempeño, y te deseo lo mejor en el futuro.

Resumen

Este estudio se enfoca en mejorar la automatización del diseño de prótesis apoyado en la inteligencia artificial y métodos numéricos. Se exploran enfoques distintos, centrándose el análisis en la superelipse. Al continuar la línea de investigación de ajuste elíptico, se consideró dicha geometría como extensión directa. Pero sucede algo particular para la elipse, que no se puede extender a la superelipse. La elipse presenta la ventaja de lograr al menos un ajuste cerrado basado en la distancia algebraica expresada con su polinomio implícito. No sucede lo mismo con la superelipse. No pudiendo aplicar los conocimientos de otros trabajos anteriores a dicha geometría.

A pesar de dicho descubrimiento, se continuó investigando la optimización de los parámetros de la superelipse para el ajuste de curvas procedentes de imágenes tomográficas. Tras la resolución de numerosos problemas en el desarrollo del modelo, se llegó a dar una mejora sustancial, con respecto al ajuste elíptico. Además de hallarse una manera rápida de complementarlo empleando los *inliers* de dicho ajuste como datos. Aunque dicha mejora no conlleve cuestionarse el tiempo necesario para la obtención de los resultados, siendo mucho menor empleando el ajuste elíptico.

Por otro lado, y de manera simultánea, se estudia la aplicación del concepto de casco convexo o *Convex Hull* al ajuste de curvas. Resultando una función sencilla de implementar que a través del aporte de un parámetro puede adaptar los resultados del ajuste según el criterio del usuario. Significa dejar un margen entre el ajuste y el contorno, que de primeras no se da por la precisión del ajuste, para la correcta implementación o montaje de futuras prótesis.

Palabras clave: métodos numéricos, diseño personalizado, superelipse, *Convex Hull*.

Códigos UNESCO

1201.01 Geometría Algebraica

1204.01 Geometría Afín

1204.06 Geometría Euclídea

1206.01 Construcción de Algoritmos

1206.07 Interpolación, Aproximación y Ajuste de Curvas

1206.08 Métodos Iterativos

3314.02 Prótesis

Abstract

This study focuses on improving the automation of prosthesis design supported by artificial intelligence and numerical methods. Different approaches are explored, focusing most of the analysis on the superellipse. By continuing the research line of elliptic fitting, this geometry could be considered as a direct extension. But there is something particular for the ellipse, which cannot be extended to the superellipse. The ellipse has the advantage of achieving at least a closed fit based on the algebraic distance expressed with its implicit polynomial. This affirmation is not true for the superellipse. Because of that, it was not possible to apply the knowledge of previous work to this geometry.

Despite this discovery, the research continued optimising the parameters of the superellipse for fitting curves from tomographic images. After solving numerous problems in the development of the model, a substantial improvement over the elliptical fit was achieved. In addition, a quick way was found to complement it by using the inliers of the elliptical fit as data. Although this improvement does not entail questioning the time needed to obtain the results, which is much less using the elliptical adjustment.

On the other hand, and simultaneously, the application of the Convex Hull concept to curve fitting is studied. The final function is simple to implement, and it can adapt the fitting results with the contribution of a parameter according to the user's criterion. It means leaving a margin between the fit and the contour, which at first is not given by the precision of the fit, for the correct implementation or assembly of future prostheses.

Keywords: numerical methods, custom design, superellipse, Convex Hull.

UNESCO Codes

1201.01 Algebraic Geometry

1204.01 Affine geometry

1204.06 Euclidean Geometry

1206.01 Construction Algorithms

1206.07 Interpolation, Approximation and Curve Fitting

1206.08 Iterative Methods

3314.02 Prosthesis

Índice

Agradecimientos.....	4
Resumen	5
Abstract	6
Introducción	9
Objetivos	10
Capítulo 1: Antecedentes y Estado del Arte	11
1.1 Antecedentes.....	11
1.2 Estado del arte	13
Capítulo 2: Marco Teórico	16
2.1 Geometrías.....	16
2.1.1 Óvalo	16
2.1.2 Superelipse	17
2.1.3 Convex Hull.....	19
2.2 Ajuste y optimización.....	21
2.2.1 Distancia algebraica y mínimos cuadrados.....	21
2.2.2 Algoritmo de Levenberg-Marquardt.....	23
2.2.3 Método de Powell.....	26
2.2.4 Método Nelder-Mead	28
2.2.5 RANSAC	31
Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas	33
3.1 Óvalo	33
3.2 Superelipse (polinomio implícito)	33
3.3 Superelipse (optimización de parámetros)	36
3.3.1 Contexto	37
3.3.2 Versiones y desarrollo de funciones	39
3.3.3 Flujo de trabajo.....	44
Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos..	55
4.1 Planteamiento	55
4.2 Desarrollo	56
Capítulo 5: Resultados	58
5.1 Optimización de parámetros de la superelipse	58
5.2 Optimización de parámetros de Convex Hull.....	64
Conclusiones	66

Líneas Futuras	67
Bibliografía.....	68
Planificación temporal y presupuesto	71
Planificación temporal	71
Presupuesto	72
Evaluación de impactos.....	74
Análisis de aspectos legales y éticos	75
Contribución a los Objetivos de Desarrollo Sostenible	75
Índice de figuras	76
Índice de tablas.....	77
Abreviaturas	78
Apéndices	79
Apéndice 1: Función superelipsen4_fit	79
Apéndice 2: Función superellipse.....	79
Apéndice 3: Función objetivo a minimizar	80
Apéndice 4: Función comparación de áreas	80
Apéndice 5: Función para calcular el área total encerrada por del contorno de la imagen ..	80
Apéndice 6: Función para estimar el área de una superellipse	81
Apéndice 7: Función para obtener la media de un ajuste	81
Apéndice 8: Función para obtener el mayor error de un ajuste	81
Apéndice 9: Función minimizacion_angulo.....	81
Apéndice 10: Función minimizacion_simple	84
Apéndice 11: Modificación de RANSAC	85
Apéndice 12: Función ajuste completa.....	86
Apéndice 13: Función convexhull_adaptable1	87
Apéndice 14: Función ordenacion_graham.....	88
Apéndice 15: Función convexhull_adaptable2.....	88

Introducción

En la continua mejora de la calidad de vida se encuentra una relación con la tecnología, avanzando o progresando unidas o apoyadas entre sí. Posiblemente, el campo en el que dicha relación sea más evidente sea el de la medicina. Dentro de este, destacamos la creación y aplicación de prótesis. Estos dispositivos logran recuperar funciones o partes del ser humano que en algún momento se han visto limitadas o dañadas. La implementación de los avances científicos y tecnológicos en su desarrollo fue toda una revolución. Sin embargo, como todo lo asociado o dependiente de la tecnología o ingeniería, está en constante actualización o mejora. Nunca se frena el avance o aplicación de conocimientos.

Uno de los aspectos más cuestionados de las prótesis es su ajuste. Esto sucede por la importancia que tiene en el desarrollo de su función y su vida útil. Un ajuste incorrecto o mejorable afecta en: la comodidad, aparición de dolores o molestias, dificultad en su empleo, no satisfacer correctamente las funciones mecánicas a las que se somete, desgaste excesivo o rotura y muchos otros. Actualmente, se dispone de los medios para optimizar los ajustes. El problema es su implementación. Como sucede en otros campos, la economía y el tiempo son factores más que importantes para justificar el empleo de cualquier metodología, técnica o solución. Por esto, la mayoría de las prótesis poseen un diseño genérico, aunque no sea la mejor de las soluciones, son económicas y fáciles de diseñar. Por tanto, para que un método pueda implementarse en la industria debe reducir tiempos de diseño y costes de fabricación.

Enunciado el problema, se busca a través de la tecnología, una posible solución. Resaltando de nuevo la relación salud-tecnología. Actualmente, la inteligencia artificial y automatización de procesos son la principal inversión de tiempo y conocimientos por parte de los ingenieros. Afectando al crecimiento de sus aplicaciones y mejorando su desarrollo exponencialmente. Con este trabajo se busca continuar la línea de investigación comenzada por William Gabriel Solórzano Requejo con su estudio: “*Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares*”.

Los resultados y la aplicación del trabajo antecedente a este estudio sólo están enfocados en dos casos concretos de prótesis. Dicho trabajo soluciona con creces el problema mencionado, pero deja abierto el uso o estudio de nuevas geometrías. Por tanto, este TFG buscará nuevas geometrías que puedan abarcar el mayor tipo de prótesis posibles. Concretamente, se estudiará la implementación de los conceptos: superelipse y *Convex Hull*. En esta investigación se encontrará toda la información necesaria, desde la base teórica a resultados obtenidos de la implementación de dichas geometrías, para el desarrollo de un algoritmo de automatización de ajuste de prótesis.

Objetivos

Objetivo general

Desarrollar funciones de geometrías cerradas adaptables (superelipse y *Convex Hull*), y así poder emplearlas para la personalización de cualquier tipo de prótesis.

Objetivos específicos

- Comprender correctamente y apoyarse en los antecedentes para poder aplicar lo que ya se ha aprendido con otros estudios y avanzar más fácilmente.
- Programar varias funciones que desempeñen de manera automática, en la mayor brevedad posible, el ajuste de las geometrías.
- Demostrar y explicar las mejoras adquiridas con el trabajo.
- Establecer líneas futuras de investigación que puedan ayudar a desarrollar más este estudio, o aprovechen los conocimientos adquiridos.

Capítulo 1: Antecedentes y Estado del Arte

1.1 Antecedentes

La idea principal de este trabajo es seguir la línea de investigación planteada en [1], que a su vez continuaba el estudio de [2] y [3]. En dicho documento se desarrolló una aplicación de ajuste elíptico para el diseño de prótesis que se adaptasen a la anatomía del paciente. Concretamente dicho trabajo se focalizó en el estudio morfológico del fémur proximal y el diseño de un vástago corto personalizado. Para el estudio realizado a lo largo de este informe se empleará la misma metodología de trabajo, que resumiremos en el siguiente párrafo, pese a no centrarse en una parte fisiológica concreta, como sucedía en los trabajos anteriores.

Para entender el flujo de trabajo que se ha seguido en el estudio, basado en [1] debe mencionarse la metodología de trabajo seguida para el diseño de dispositivos médicos personalizados (Figura 1).

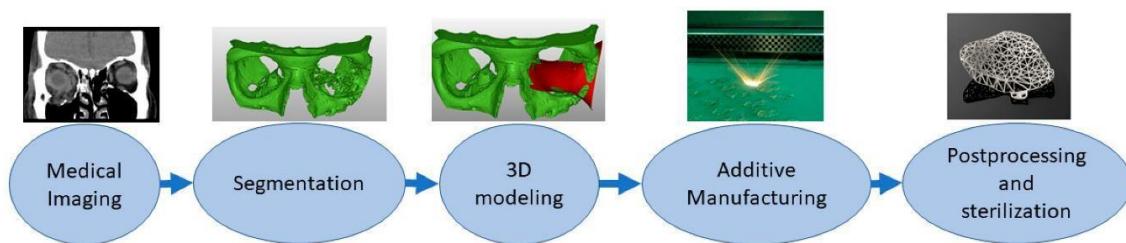


Figura 1. Flujo de trabajo para prótesis personalizadas. Fuente: “*Additive Manufacturing Processes in Medical Applications*” [4].

A fin de obtener diseños personalizados se comienza creando un modelo digital basado en técnicas de imagen médica, sistemas de escaneo 3D, software de modelado CAD o combinaciones de ellos. El modelo deberá representar la anatomía del paciente, para ello se requiere la segmentación de imágenes médicas, a través de la selección adecuada de véxoles (Figura 2).

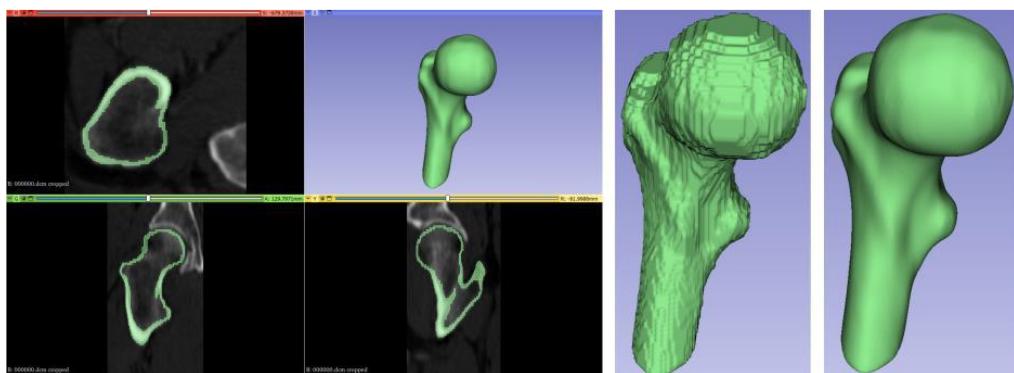


Figura 2. Segmentación de un fémur proximal. Fuente: “*Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares*” [1].

En [1] se aprovecha la inteligencia artificial para lograr una nueva estrategia de obtención de modelos precisos en menos tiempo. La metodología de obtención del modelo comienza por el procesado de imágenes médicas obtenidas por tomografía (TC) con el algoritmo *subpixel_edges*, transformando en un sistema de coordenadas los pixeles de la imagen correspondiente. El siguiente paso consistirá en diferenciar los bordes internos y externos, para lo que se emplea el algoritmo de *clustering DBSCAN* (Figura 3).



Figura 3. (A) Imagen Tomográfica, (B) Imagen en sistema de coordenadas - algoritmo *subpixel_edges* y (C) separación de los contornos exterior e interior – DBSCAN. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].

Como la mayoría de los dispositivos médicos o prótesis se ajustan al interior se trabajará con dicho borde, cuyas coordenadas son conocidas tras pasar por DBSCAN. A dicho contorno interno es donde se ajusta matemáticamente una elipse o círculo en [1], y donde se buscará ajustar las nuevas geometrías estudiadas en este trabajo. El ajuste de círculos y elipses mencionado se basa en la distancia algebraica y el método de mínimos cuadrados. Como la base de dichos ajustes y las nuevas geometrías son iguales, se desarrollará más acerca de la distancia algebraica y el método de mínimos cuadrados en el Capítulo 2: Marco Teórico. Para asegurarse que la curva a ajustar permanezca en todo momento interna al contorno interior se emplea tanto en [1] como este trabajo unas modificaciones del algoritmo RANSAC. Pese a ser muy similares ambas, pero no exactamente iguales, se aprovecha de nuevo el Capítulo 2: Marco Teórico para desarrollar más acerca de dicho algoritmo. Tras obtener una geometría ajustada definida, bien con las coordenadas conocidas, o parámetros representativos (radios, centros, etc) se aprovecha dicha información mediante software de modelado CAD y poder convertir o crear archivos STL. Con la importación de las geometrías se logra aprovechar la expansión de la manufactura aditiva en la medicina y en los dispositivos como prótesis, base del trabajo [1]. Con el uso de esta estrategia y todos los algoritmos implicados se logra un modelado más efectivo y automatizado.

El propósito de este documento se centra en la parte del flujo de trabajo que va desde la imagen tomográfica a la geometría ajustada. No se busca mejorar las técnicas de obtención de coordenadas a partir de vóxeles, ni en la obtención de una prótesis concreta completa. Usando las herramientas desarrolladas en [1] se buscan nuevas geometrías que ajusten de manera automatizada al mayor número de imágenes médicas posibles. En [1] se concluyó que la elipse era el mejor ajuste automatizado para las imágenes personalizadas de fémur y arteria. Con este trabajo se quiere ver la influencia de otras geometrías, en comparación con la elipse, además de buscar una generalización a cualquier órgano o imagen.

1.2 Estado del arte

Más adelante, en el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas y en el Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos se desarrollará más acerca de las geometrías de estudio. Pero para poder situar al lector en este apartado, se enunciarán brevemente las geometrías destacadas en el trabajo y relación con otros estudios académicos o actuales que han servido de inspiración o ayuda. Las principales geometrías o ideas para ajustar se basan en: óvalos, superelipses y polígonos cuyos vértices sean los puntos de la curva a ajustar.

La primera geometría mencionada, el óvalo, parte de una idea del tutor debido a su gran similitud con la elipse, de la cual se trajeron los mejores resultados en [1]. Investigando su geometría, comparación con elipse, empleo de la distancia algebraica y mínimos cuadrados, no se llegó a encontrar ningún artículo o trabajo centrado en el ajuste de óvalos a imágenes médicas mediante inteligencia artificial. En el Capítulo 2: Marco Teórico se explicará lo necesario acerca de esta geometría y en el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas todo sobre su implementación en el ajuste de imágenes médicas.

Por el contrario, de las superelipses se encontró mucha información y artículos relacionados con el ajuste a datos o empleo en el diseño de prótesis. Es más, la idea de enfocar la investigación en esta geometría vino inspirada de algunos de los artículos que se mencionarán a continuación, y de nuevo, por su gran relación con la famosa elipse. En el Capítulo 2: Marco Teórico se desarrollará su relación con la elipse y se explicará todo lo necesario acerca de dicha geometría. Pese a encontrar los artículos que se mencionarán a continuación, sigue sin ser un caso de estudio tan concreto como el de este informe o el de [1]. Los artículos tratan del empleo en el diseño de prótesis, matemática del ajuste a datos, teorías y errores de ajustes; pero en ningún caso se enfocan en el ajuste automatizado de dicha geometría a distintas imágenes médicas.

La idea de la superelipse comenzó gracias al estudio [5] de la implementación de estas en prótesis de cadera y fémur, similares a las de [1]. En [5] se ve una clara similitud con [1] ya que el modelo se realiza por capas, aunque en este caso de superelipses (Figura 4). El principal objetivo es el mismo, optimizar la geometría de la prótesis para encontrar el mejor ajuste, y, por tanto, minimizar las tensiones entre hueso y prótesis, y con ello el desgaste de esta. Pero dicha optimización o ajuste no se apoya en la inteligencia artificial ni busca la personalización de las prótesis. Se basa en una metodología que mezcla matemática y diseño de cuerpos sólidos o CAD. Primero, sitúa el centro del hueso y de la superelipse a ajustar en la misma posición y después, apoyándose en la ecuación de la superelipse, fijando un grado y longitudes de los semiejes se busca la curva de Bézier o *spline* que ajuste mejor.

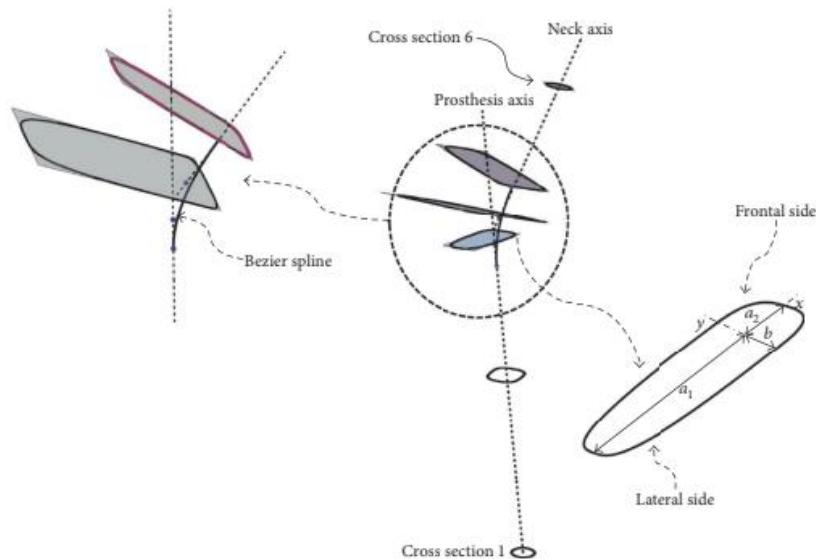


Figura 4. Resumen del desarrollo del ajuste de superelipses en "*Minimizing Stress Shielding and Cement Damage in Cemented Femoral Component of a Hip Prosthesis through Computational Design Optimization*" [6].

Al hablar de diseño, se encontró un artículo que abordaba el concepto de superelipsoide y su aplicación en el campo de la mecánica [6]. El artículo se centra en una extensión del concepto de superelipsoide, que es una figura geométrica paramétrica utilizada para describir objetos con formas que se asemejan a combinaciones de elipses y rectángulos. Esta generalización busca aplicar esta forma geométrica a problemas en el campo de la mecánica. Con este artículo se confirma el uso de la superelipse en el ámbito del diseño, aunque en nuestro caso no es propiamente mecánico. Y se empieza a discutir el empleo de los parámetros para el ajuste, que será un punto importante a lo largo de este trabajo.

Además de los artículos basados en el diseño con superellipse, se encontró la mayor fuente de inspiración en artículos que empleaban la superellipse para el ajuste de datos. Su uso es muy global, ya que los datos pueden ser nubes de puntos y con dichos trabajos lo que se busca es agrupaciones para minimizar costes computacionales u otros fines. Pero, pese a no darse un ajuste tan específico como en los anteriores artículos centrados en el diseño, se logró recopilar información acerca de las matemáticas de un ajuste de superellipse a datos. Al final, el caso de estudio de este trabajo es similar, ya que la imagen médica se define por coordenadas de los puntos de su contorno.

En el primer informe de este estilo “*Superellipse fitting to partial data*” [7], de Xiaoming Zhang y Paul L.Rosin, se explora cómo ajustar superelipses a datos parciales y cómo adaptar el concepto de superellipse a escenarios donde la información es limitada. Una superellipse es una figura geométrica que combina propiedades de elipses y rectángulos, y se utiliza para describir formas curvas y angulares. El enfoque principal aquí es cómo adaptar este concepto a datos que pueden no contener toda la información necesaria para ajustar una superellipse de manera precisa. Para ello se busca modificar la función objetivo (1), basada como en [1] en la distancia algebraica y en mínimos cuadrados, y emplear dicha nueva función en la optimización. Debido a la falta de información o imprecisión en los datos, la optimización se basa en emplear un algoritmo de Levenberg-Marquardt, del que se desarrollará más en el Capítulo 2: Marco Teórico, junto con una iteración proveniente de un ajuste elíptico previo. Esto último es un buen punto a tener en cuenta en este trabajo, ya que gracias a [1] se logra el ajuste elíptico y puede ser de gran utilidad.

$$Q_0(x, y) = \left[\frac{(x - x_c) \cos \theta - (y - y_c) \sin \theta}{a} \right]^{\frac{2}{\epsilon}} + \left[\frac{(y - y_c) \cos \theta + (x - x_c) \sin \theta}{b} \right]^{\frac{2}{\epsilon}} - 1 \quad (1)$$

Siguiendo la línea de modificar la ecuación (1) se encuentra otro estudio [8] muy parecido.

En la misma línea, esta “*Curve Segmentation and Representation by Superellipses*” [9] de Paul L.Rosin and Geoff A.W. West con un enfoque muy parecido, pero empleando la distancia euclídea y un método de optimización basado en la técnica de Powell.

En un último trabajo titulado “*Superellipse Fitting for the Recovery and Classification of Mine-Like Shapes in Sidescan Sonar Images*” [10], de Esther Dura. Judith Bell and Dave Lane, se plantea usar un método de optimización distinto a los anteriores. Se ve innecesario usar la técnica de Powell ya que se considera como un gasto computacional innecesario. Para ello se plantea usar Nelder-Mead, obteniendo unos resultados satisfactorios.

Los informes mencionados ofrecen una nueva visión para el ajuste. Pudiendo partir de una función objetivo (1) y las modificaciones o simplificaciones de los distintos trabajos. Además, de disponer de tres nuevas herramientas de optimización a valorar: Levenberg-Marquardt, Powell y Nelder-Mead. Todo ello se verá reflejado y desarrollado en capítulos posteriores.

Por otro lado, acerca de la idea de ajustarse a la curva con un polígono cuyos vértices sean los puntos de la curva no se llega a encontrar ningún caso de estudio práctico de actualidad o utilidad. La única fuente de información es toda aquella relacionada con el Convex Hull y dicho campo de la geometría computacional [11]. El concepto de Convex Hull también será definido en el Capítulo 2: Marco Teórico. Lo que sí se llega a encontrar es una herramienta de gran utilidad para el desarrollo del trabajo, y es que dentro de la librería *SciPy* [12] de Python® hay una función *ConvexHull* [13] que facilita y simplifica el código a desarrollar en el Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos.

Capítulo 2: Marco Teórico

La importancia de este capítulo es siempre grande en trabajos de investigación o estudio. Sin un marco teórico trabajado no se concibe llegar a valorar todas las opciones, o encontrar un resultado conforme a lo que se busca. Para el caso de este trabajo, ajuste de geometría a imágenes médicas de manera automatizada o apoyándose en IA, es necesario conocer al detalle las geometrías que se emplean y las técnicas de ajuste u optimización. Así pues, se decide dividir este apartado en dos secciones principales: geometrías y ajuste y optimización. Dentro del primero se desarrollarán los parámetros más importantes o aspectos a tener en cuenta para el ajuste de cada una de las geometrías empleadas en el documento. Y en el segundo se expandirá más acerca de la metodología de ajuste y los métodos de optimización.

2.1 Geometrías

2.1.1 Óvalo

Como ya se ha mencionado previamente, una de las causas del estudio del óvalo es su similitud con la elipse. Por ello, se definirá genéricamente dicha geometría y se relacionará con la elipse que tan buenos resultados obtuvo en [1].

Un óvalo, definido por la RAE como: “curva cerrada, con la convexidad vuelta siempre a la parte de afuera, de forma parecida a la de la elipse, y simétrica respecto de uno o de dos ejes” [14]. O también definido como: “forma geométrica convexa y redondeada, que se asemeja al perfil de un huevo de ave en su sentido más amplio. Incluye a circunferencias y elipses como casos especiales, con dos ejes de simetría en lugar de solamente uno o ninguno.” [15]. Podrían mencionarse otros ejemplos, pero se comprende la similitud entre el óvalo y la elipse. En la siguiente figura (Figura 5) se pueden observar dichas similitudes.

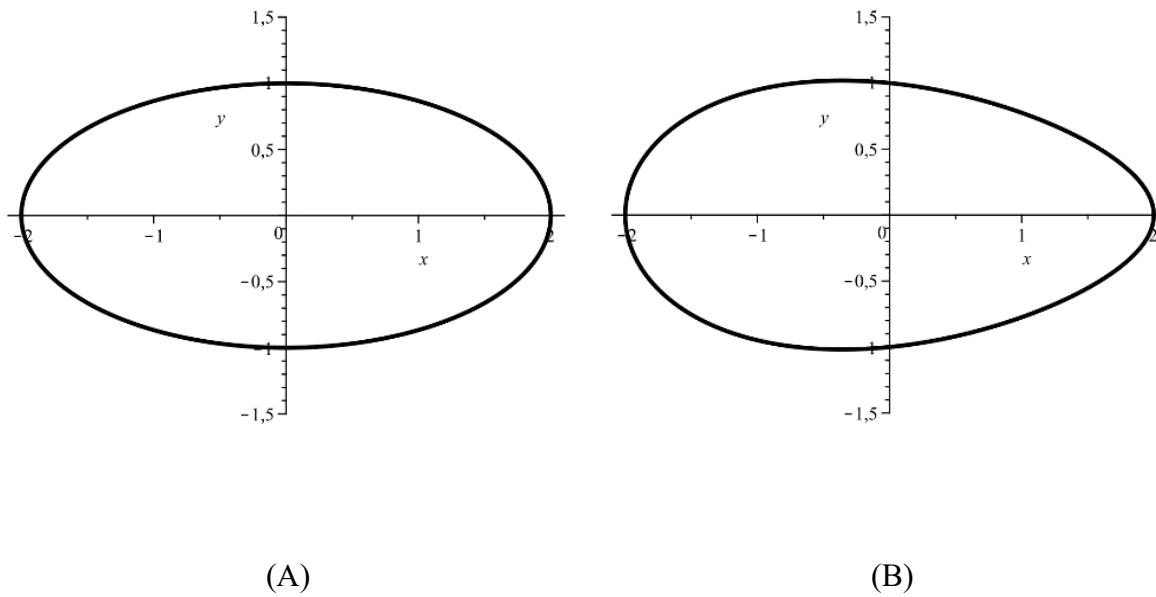


Figura 5. (A) Geometría de una elipse y (B) geometría de un óvalo. Fuente: Wikipedia [15].

Comparando visualmente las imágenes anteriores, se puede ver que cambia la forma y simetría de ambas curvas. La elipse es un caso concreto en el que se cumple que la suma de las distancias de los dos focos, a cualquier punto de la curva, es constante. Si dicha suma se cambia por una suma ponderada, multiplicando cada distancia de los focos por un valor, se habla de cualquier otro óvalo genérico. Tras esta primera particularidad de la elipse, se podría mencionar otra en la propia ecuación de la curva. En la ecuación genérica para el caso de la elipse (2), “a” y “b” corresponden a las longitudes de sus semiejes, y son constantes. En cambio, quitando para el caso particular de la elipse, en la ecuación del óvalo (3) se sustituye “a” y “b” por distintas ecuaciones.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (2)$$

$$\frac{x^2}{4} + \frac{y^2}{1 - 0.2 * x} = 1 \quad (3)$$

Observando las imágenes y teniendo en cuenta las diferencias recién mencionadas. Para ajustar una elipse, los parámetros de los semiejes definirán bastante la geometría final, al ser constantes en la ecuación (2). Por otro lado, en un óvalo, al ser funciones varían respecto de otros valores. Dicho de otra forma, el óvalo dispone de más datos a modificar para variar su geometría y buscar un mejor ajuste.

En el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas se aplicará la teoría desarrollada acerca de distancia algebraica y mínimos cuadrados a esta geometría, de nuevo comparando brevemente con el caso de la elipse.

2.1.2 Superelipse

Una superelipse o también conocida como curva de Lamé es una representación geométrica definida por la ecuación (4), donde $n > 0$ y “a” y “b” corresponden a los semiejes de la curva:

$$\left| \frac{x}{a} \right|^n + \left| \frac{y}{b} \right|^n = 1 \quad (4)$$

Se puede observar que (2) y (4) son muy similares. Como sucedía con el óvalo, la superelipse también tiene relación teórica y geométrica con la elipse, por lo que es interesante continuar el estudio con dicha geometría. Como se verá posteriormente, la geometría de la superelipse es similar a la de la elipse con una curvatura más o menos pronunciada dependiendo del grado “n” (Figura 6). Por esto se considera un caso útil de estudio, ya que si la elipse ajustaba bien en [1], el cambio de curvatura podría mejorar el ajuste.

Pese a atribuirse al científico Piet Hein (1905-1996) el descubrimiento de la superelipse fue el matemático Gabriel Lamé (1795-1870) quien generalizó dicha ecuación [16].

Como sucede en muchas otras geometrías, pueden diferenciarse casos específicos según los valores que tomen sus parámetros. Los casos más comunes para los distintos valores que toma “n” son [17]:

- $0 < n < 1$: Superelipse de forma similar a la de una estrella con lados cóncavos (curvados hacia el centro). Destaca el caso del astroide donde $a = b$ y $n = 2/3$.
- $n = 1$: La curva es un rombo con esquinas $(\pm a, 0)$ y $(0, \pm b)$
- $1 < n < 2$: Similar al primer caso, pero los lados tienen forma convexa (curvados hacia afuera).
- $n = 2$: Elipse. Además, como caso particular para $a = b$, se trata de un círculo.
- $n > 2$: Curva con geometría similar a la de un rectángulo donde sus curvas toman una cierta curvatura dependiendo del grado.
- $n = \infty$. Sería un rectángulo.

Todos los tipos están comprendidos dentro del perímetro que une los puntos $(\pm a, 0)$ y $(0, \pm b)$, o el caso del rectángulo con dichos vértices.

Pese a haber hecho distintas distinciones a partir del grado “n”, hay dos más genéricas. Para $n < 2$ se habla de hipoelipses y para $n > 2$ se denominan hiperelipses. El estudio que se llevará a cabo se centra en las hiperelipses. Aunque en dicho grupo incluiremos las elipses para tener una comparación más precisa de los resultados.

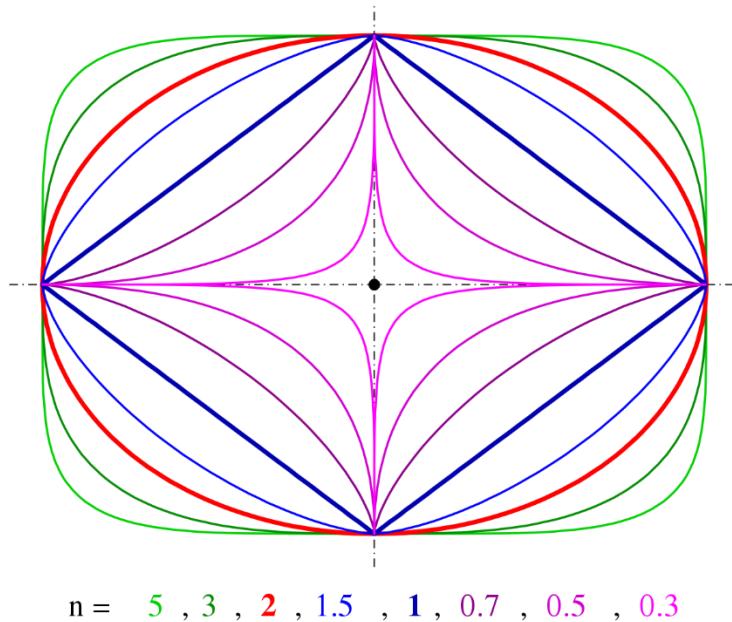


Figura 6. Representación de distintas superelipses variando su grado “n”. Fuente: Wikipedia [18].

Pese a ser (4) la ecuación general que define una superelipse es necesario realizar unas modificaciones para automatizar el ajuste con ayuda de algoritmos,

- Posicionar el centro de la superelipse respecto del sistema global o contorno de la imagen médica. Esto simplemente afecta a (4) añadiendo la distancia entre las variables x e y , coordenadas de los puntos de la curva, al centro correspondiente, posición del centro de la superelipse respecto del sistema global, denominándolo por ejemplo (x_c, y_c) .
- Establecer relación de rotación entre las coordenadas de la superelipse y el sistema global o contorno de la imagen médica. Para ello simplemente hay que usar la matriz de transformación estudiada en cinemática plana (5). Únicamente es necesario transformar o representar la rotación, puesto que el desplazamiento de los centros ya se ha expresado en el punto anterior. Para el caso a estudiar, se necesita de una matriz de

rotación para dos dimensiones (5) ya que se trabaja con geometrías pertenecientes a dicho espacio.

$$R(\Theta) = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \quad (5)$$

Se pasa a denominar “x” e “y” a las coordenadas locales de la superelipse. Multiplicando el vector posición $[x \ y]$ por la matriz anterior y sustituyendo las “x” e “y” de (4) por dicho resultado, además de tener en cuenta la modificación anterior, se obtiene la ecuación de la superelipse (6). Dicha ecuación (6) sería la que se buscara ajustar a la curva correspondiente. Que es casi idéntica a (1) con el único cambio en el exponente.

$$\left[\frac{(x - x_c)\cos\Theta - (y - y_c)\sin\Theta}{a} \right]^n + \left[\frac{(y - y_c)\cos\Theta + (x - x_c)\sin\Theta}{b} \right]^n = 1 \quad (6)$$

Variando el grado “n”, se pueden estudiar geometrías similares a la elipse pero que varían su forma y área, pudiendo resultar en un mejor ajuste. Dicho grado, interesa que tome únicamente valores pares, esto debido a que la simetría que ofrece el grado par hace que, a la hora de buscar el ajuste, y rotar la curva, no se vea alterada dicha curva y su forma.

Al escoger dicha restricción, la superelipse tiene simetría y se mantiene invariante respecto a rotaciones de 180° . Todo esto se ve mejor al trasladarlo a sus ecuaciones paramétricas (7) y (8):

$$x(\Theta) = x_c + a * \cos(\Theta)^n \quad (7)$$

$$y(\Theta) = y_c + b * \sin(\Theta)^n \quad (8)$$

Al tomar “n” como valor par, los términos trigonométricos de dichas ecuaciones (7) y (8) implican una simetría respecto de los ejes x e y. Es decir, al tomar un ángulo (Θ) y su opuesto ($-\Theta$), dichas funciones paramétricas producirán los mismos valores “x” e “y”, confirmando la simetría respecto de los ejes locales de dicha superelipse. Por tanto, para un “n” par la curva puede rotar sin verse alterada.

De nuevo, en el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas se verá aplicada la teoría desde dos enfoques distintos. El primero de ellos se basa en encontrar la ecuación implícita de (6) y aplicar el ajuste por distancia algebraica y mínimos cuadrados como se realizó en [1]. El segundo consiste en optimizar los parámetros de (6) para lograr el mejor ajuste.

2.1.3 Convex Hull

Posteriormente en el Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos se explicará cómo se llegó a la idea de emplear el concepto de *Convex Hull* en el ajuste a imágenes médicas. Este apartado se centra en su definición y aspectos importantes para entender su concepto.

En español es conocido como envoltura o casco convexo, y es un concepto geométrico y matemático que hace referencia a la forma geométrica o polígono más pequeña y convexa que contiene a un conjunto de puntos dados en un espacio euclíadiano [19]. Se puede simplificar como el polígono convexo más pequeño que abarca todos los puntos del conjunto. Para el caso tridimensional, poliedro (Figura 7).

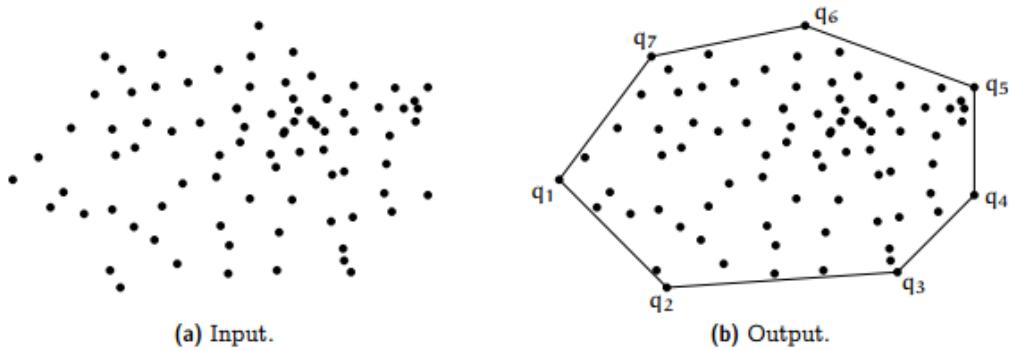


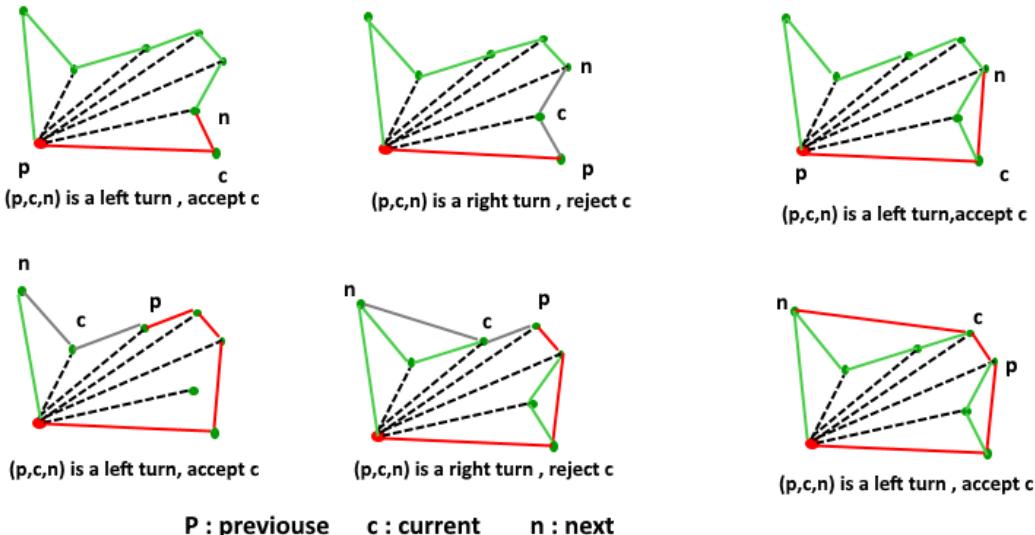
Figura 7. Ejemplo de Convex Hull. Fuente: Medium [20].

A este concepto se le pueden asociar distintos atributos o características [21]:

- Convexidad: todos los ángulos interiores del polígono o poliedro son menores o iguales a 180 grados. La forma siempre se curva hacia afuera, sin concavidades.
- Eficiencia: la envoltura convexa es un concepto importante en la geometría computacional y se utiliza en una variedad de aplicaciones debido a su eficiencia en el cálculo y su utilidad en problemas prácticos.
- Menor encerramiento: la envoltura convexa tiene el área (en 2D) o el volumen (en 3D) mínimo posible mientras incluye todos los puntos en su interior.
- Complejidad: la complejidad del cálculo de la envoltura convexa puede variar dependiendo de la cantidad de puntos y la distribución espacial de los mismos.
- Cálculo: existen varios algoritmos para calcular la envoltura convexa, como el algoritmo Graham-Scan, Jarvis March y QuickHull.

Para comprender mejor su aplicación se explica la base del algoritmo de Graham-Scan [22] y que tendrá influencia en el Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos. Dicho algoritmo funciona de la siguiente manera (Figura 8):

- Selecciona un punto base. Comienza seleccionando el punto más bajo de todos los puntos como el punto base.
- Ordena por ángulo. (Parte del algoritmo que se verá más reflejada en nuestro trabajo). Ordena los puntos restantes según su ángulo polar con respecto al punto base. Esto crea una lista de puntos ordenados en sentido antihorario alrededor del punto base.
- Pila. Almacena en una pila los puntos según los procesa. En la parte inferior se situarán el punto base y el de menor ángulo.
- Recorrido. Para verificar que los puntos se recorren en el orden correcto va verificando si forman una vuelta a derechas o izquierdas con respecto de los puntos superiores de la pila. Si es a derechas eliminan los puntos de la pila hasta formar una vuelta a izquierdas.
- Al final del procesado, los puntos de la pila formaran el *Convex Hull*.



In the above algorithm and below code , a stack of points is used to store convex hull points. With reference to the code ,**p** is next-to-top in stack, **c** is top of stack and **n** is point[i]

Figura 8. Esquema explicativo del algoritmo de Graham-Scan para hallar el Convex Hull. Fuente: GeeksforGeeks [22].

Esta metodología, y el uso del concepto de *Convex Hull* está bastante extendida. Dentro de sus aplicaciones destacan: gráficos para computadora, robótica, geografía, procesamiento de imágenes, diseño industrial, estadística, procesamiento de rutas y otros.

2.2 Ajuste y optimización

2.2.1 Distancia algebraica y mínimos cuadrados

Como bien se explica en el TFM “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” esta metodología se basa en la representación mediante el polinomio implícito de la cónica a ajustar. Para desarrollar dichos conceptos se toma dicho TFM como referencia, tomando la elipse como cónica a ajustar [1]. Para las geometrías a estudiar en el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas se sigue la misma metodología cambiando el polinomio implícito, siendo todos muy similares.

Para el caso de la elipse, definiremos el polinomio implícito de segundo orden (Q) por un vector de coeficientes ($v = [A \ B \ C \ D \ E \ F]^T$):

$$Q(p, v) = [x^2 \ xy \ y^2 \ x \ y \ 1] * \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = 0 \quad (9)$$

Denominando $P = \{p_1, p_2, \dots, p_n\}$ al conjunto de puntos del contorno de la imagen tomográfica, entonces el vector de coeficientes debe ajustarse a él empleando la distancia algebraica (D_A) debido a la simplificación en cálculos y recursos computacionales [23]. Para

obtener dicha distancia matemáticamente únicamente habrá que reemplazar las coordenadas de un punto $p_i = (x_i, y_i)$ en el polinomio Q (9). Conque si p_i pertenece, en este caso a la elipse, su distancia será 0.

$$D_A(p_i, v) = Q(p_i, v) = A * x_i^2 + B * x_i * y_i + C * y_i^2 + D * x_i + E * y_i + F \quad (10)$$

Observando la Figura 9, se aprecia que la distancia algebraica geométricamente es la medida resultante de la intersección entre un vector trazado desde el centro de la elipse hasta el punto p_i y la propia elipse. Además, esta distancia aporta información adicional que será de gran utilidad en el apartado o uso de RANSAC, que se analizará posteriormente. Ya que dependiendo del signo de dicha distancia el punto será interior o exterior a la cónica. Si su signo es negativo significa que es interior y, por tanto, si fuese positivo, externo.

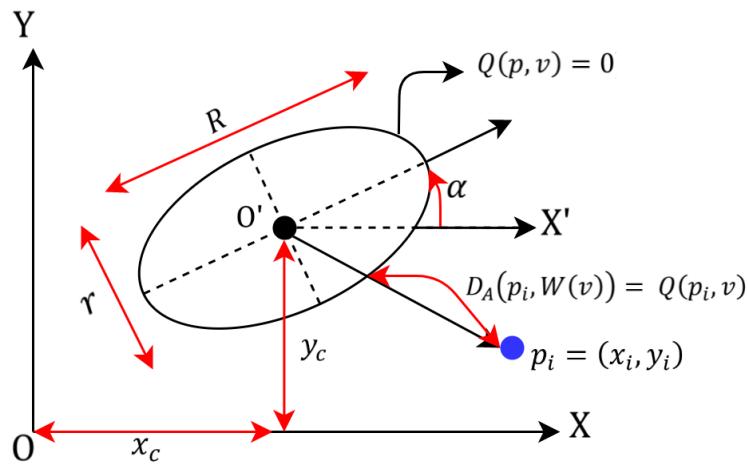


Figura 9. Parámetros de la elipse y distancia algebraica. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].

Para optimizar el ajuste se emplea la técnica de mínimos cuadrados, minimizando el cuadrado de la distancia algebraica entre los puntos P y la elipse Q. Pudiéndose expresar como la norma al cuadrado del producto entre la matriz de diseño D_p , que contiene información de P, y el vector v [1].

$$D_p = \begin{bmatrix} x_1^2 & x_1 * y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2 * y_2 & y_2^2 & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & x_n * y_n & y_n^2 & x_n & y_n & 1 \end{bmatrix} \quad (11)$$

$$\min \sum_{i=1}^n D_A(p_i, v)^2 = \min \sum_{i=1}^n Q(p_i, v)^2 = \min \|D_p v\|^2 \quad (12)$$

Es posible que una solución a este planteamiento sea $v = \bar{0}$. Para ello se evita restringiendo el vector de coeficientes $\|v\|^2 = 1$, evitando que todos sus términos sean nulos [24]. Tomando esta condición y aplicando multiplicadores de Lagrange, se obtiene la función lagrangiana L (13) a optimizar.

$$L = \|D_P v\|^2 - \lambda(\|v\|^2 - 1) = v^T D_P^T D_P v - \lambda(v^T v - 1) \quad (13)$$

Para minimizar (13), se iguala el gradiente de L con respecto a v a 0:

$$\nabla_v L = 0 \leftrightarrow 2D_P^T D_P v - 2\lambda v = 0 \quad (14)$$

$$D_P^T D_P v = \lambda v \quad (15)$$

De la ecuación (15) se obtiene un problema de vectores propios, entonces λ y v corresponden a un valor y un vector propio de $D_P^T D_P$. Si $D_P^T D_P v = \lambda v$, (12) pasa a ser:

$$\min \|D_P v\|^2 = \min v^T D_P^T D_P v = \min \lambda \|v\|^2 = \min \lambda \quad (16)$$

Por lo tanto, el vector de coeficientes (v) que minimiza la distancia resulta ser el vector propio de $D_P^T D_P$ asociado al valor propio más pequeño (λ). Utilizando este vector v , se define la elipse Q , y se pueden establecer cinco parámetros que la caracterizan: las coordenadas de su centro (x_c, y_c), los tamaños de los semiejes mayor (R) y menor (r), y el ángulo (α) que provoca una rotación en sentido contrario a las agujas del reloj (Figura 9). Para hallar dichos parámetros se debe hacer uso de la función *elliptical_parameters*, tras emplear *elliptical_fit* [1].

2.2.2 Algoritmo de Levenberg-Marquardt

Como se mencionó previamente en el Estado del arte, en el trabajo “*Superellipse fitting to partial data*” [7] de Xiaoming Zhang y Paul L.Rosin se buscaba ajustar la ecuación (1) modificada empleando este algoritmo. Para ver su aplicación en este proyecto se buscó información acerca de su funcionamiento o base matemática.

El algoritmo de Levenberg-Marquardt (LMA o LM), también conocido en matemáticas o computación como el método de mínimos cuadrados amortiguados (DLS), se utiliza para resolver problemas de mínimos cuadrados no lineales. Fue desarrollado independientemente por Kenneth Levenberg y Donald Marquardt, combinando el algoritmo de Gauss-Newton (GNA) y el método de descenso de gradiente. Debido a esta combinación, el LMA encuentra una solución incluso comenzando alejado del mínimo final, lo que significa que es más robusto que el GNA. Pero al combinarse con el descenso de gradiente es significativamente más lento que el GNA. Puede verse de manera general como un algoritmo de Gauss-Newton enfocado en una región de confianza [25].

El objetivo principal del algoritmo es encontrar los parámetros que minimiza la diferencia entre los valores observados (puntos del contorno) y los predichos (curva a ajustar) por un modelo matemático no lineal. Antes de resumir su funcionamiento o desarrollo se destacan algunos aspectos claves.

- Actualización de parámetros. Se inicia el algoritmo con una estimación inicial y en cada iteración se ajustan los parámetros para minimizar la función costo. Dicha función será una media de la distancia entre los valores observados respecto a los predichos por el modelo.
- Matriz Hessiana. El algoritmo utiliza la matriz Hessiana, generalización de la matriz jacobiana que contiene las segundas derivadas parciales de la función costo respecto de los parámetros a minimizar. Dicha matriz se emplea para ajustar la magnitud de los cambios en los parámetros.

- Control de dirección de cambio. El algoritmo ajusta los parámetros en la dirección que reduce la función de costo. Si la dirección resultante mejora la función de costo, se acepta el cambio. Si el cambio empeora la función de costo, el algoritmo intenta controlar la dirección y la magnitud del cambio.
- Parámetro de Marquardt. El parámetro de Marquardt (denotado como λ) controla la mezcla entre el método de Gauss-Newton (que es más rápido, pero puede divergir) y el método de descenso de gradiente (que es más seguro, pero más lento). Un valor pequeño de λ tiende hacia el método de Gauss-Newton, mientras que un valor grande tiende hacia el Gradiente Descendente.
- Convergencia. El algoritmo de Levenberg-Marquardt es iterativo y converge a medida que los parámetros se ajustan para reducir la función de costo. La convergencia puede depender de la elección inicial de los parámetros y el valor de λ .

A continuación, se desarrolla técnicamente su funcionamiento [26] [27]:

- Formulación del problema. Definición de la función no lineal a minimizar: $f(x_1, x_2, \dots, x_n)$. Esta función suele representar el error entre los valores observados y los valores predichos por un modelo no lineal.
- Inicialización de los parámetros. Se inicia con una estimación x_0 de los parámetros que se deseen ajustar. En [7] se toman los parámetros de un ajuste elíptico como dicha estimación. En este trabajo se trabajará con la misma estimación.
- Cálculo de la Jacobiana. La Jacobiana J es una matriz que contiene las derivadas parciales de f con respecto a cada uno de los parámetros. En cada iteración, se calcula J evaluando las derivadas parciales en los valores actuales de los parámetros.
- Cálculo del residuo. El residuo r es un vector que contiene las diferencias entre los valores observados y los valores predichos por el modelo no lineal. Se calcula como $r = \text{valores observados} - \text{valores predichos}$.
- Cálculo del paso: Se busca un paso de actualización Δx para los parámetros que reduzca el error. Este paso se calcula resolviendo el sistema de ecuaciones $J^T J \Delta x = J^T r$, donde J^T es la matriz transpuesta de J .
- Control del paso: Se introduce un parámetro llamado λ (lambda) que ajusta el tamaño del paso. Si λ es pequeño, el algoritmo se comporta como el método de Gauss-Newton; si es grande, se comporta como el método del Gradiente Descendente. Esto permite combinar las ventajas de ambos métodos.
- Actualización de parámetros: Los parámetros se actualizan con el paso calculado: $x_i \rightarrow x_i + \Delta x_i$ para cada i .
- Criterio de convergencia: Se verifica si el cambio en los parámetros o el valor de la función es lo suficientemente pequeño para detener el algoritmo. Si no se cumple el criterio de convergencia, se vuelve al paso 3 y se repiten las iteraciones.

El algoritmo permanecerá iterando hasta cumplir con el criterio de convergencia. El principal objetivo es encontrar la dirección que ajuste los parámetros disminuyendo el error de manera efectiva. Con lambda λ se puede ajustar el equilibrio entre velocidad y garantía de convergencia estable (Figura 10).

Este algoritmo está ampliamente integrado en áreas de optimización numérica, estadística ajuste de curvas y estimación de parámetros. Al estar tan integrado en diversos campos de estudio se ve implementado en librerías y herramientas de programación. Concretamente dentro de la librería *SciPy* [12] de Python® se encuentra la función *scipy.optimize.least_squares* que emplea este algoritmo, especificando method: 'lm', para optimizar una función [28].

Pese a valorar la implementación de dicha función en el trabajo, en principio se descarta debido a la complejidad que supone trabajar con la hessiana de (6). Como quedan dos algoritmos por estudiar, el descarte no supone una decisión definitiva. También se puede valorar como caso de estudio para líneas futuras.

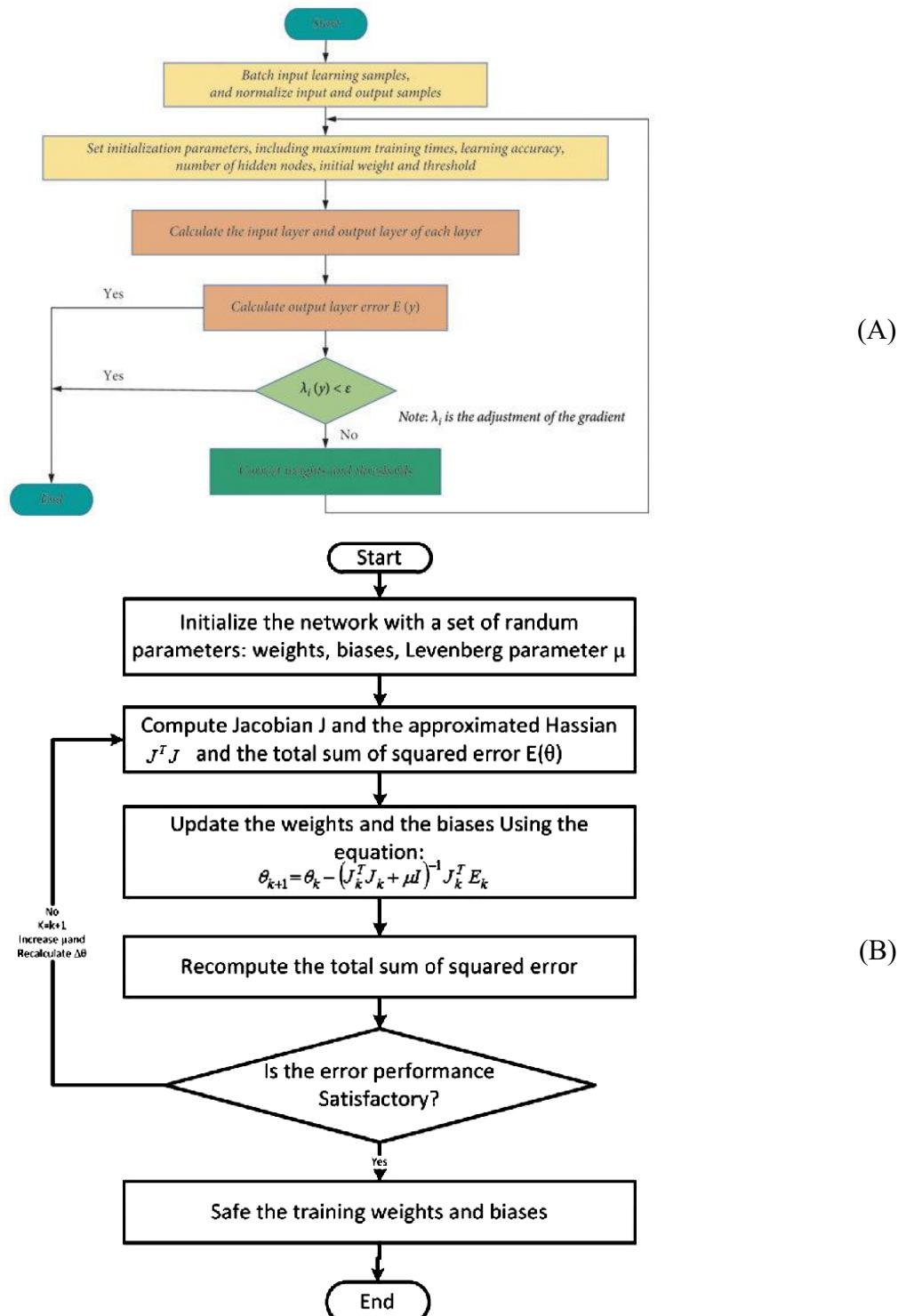


Figura 10. (A) Flujograma básico del algoritmo de Levenberg-Marquardt y (B) flujograma más técnico de dicho algoritmo. Fuentes: “Investigation on the Displacement Ductility Coefficient of Reinforced Concrete Columns Strengthened with Textile-Reinforced Concrete” [29] y “Charged Particle Pseudorapidity Distributions for Pb-Pb and Au-Au Collisions Using Neural Network Model” [30].

2.2.3 Método de Powell

De nuevo, volviendo al Estado del arte, en el trabajo “*Curve Segmentation and Representation by Superellipses*” [9] de Paul L.Rosin and Geoff A.W. West se trabajaba con dicho método. Siguiendo la línea del apartado anterior, se informará acerca de este método.

Como el algoritmo anterior, este método es un algoritmo de optimización no lineal para encontrar el mínimo de una función de dimensión múltiple. Desarrollado por Michael J.D. Powell, es especialmente útil cuando la función objetivo no es convexa y puede tener múltiples mínimos locales.

Su principal diferencia con otros métodos de optimización que requieren del cálculo de derivadas, como Levenberg-Marquardt (basado en el método de Newton y de descenso de gradiente), es que este no necesita información sobre las derivadas de la función. Se basa en una estrategia de exploración que sigue la dirección de los ejes principales del espacio de búsqueda, que ahora se desarrollará [31]:

- Inicialización. Se comienza con un punto inicial x_0 y un conjunto de direcciones iniciales d_1, d_2, \dots, d_n que se utilizarán para explorar el espacio de búsqueda. Estas direcciones generalmente son los ejes de coordenadas, pero podrían ser otras direcciones específicas
- Iteración. En cada iteración, el algoritmo realiza los siguientes pasos:
 - a) Minimización unidimensional. Se minimiza la función $f(x_0 + \lambda d_i)$ en cada dirección d_i , donde λ es el tamaño del paso. Se utiliza un método unidimensional (como la búsqueda en línea de Brent) para encontrar el valor de λ que minimiza la función en esa dirección.
 - b) Actualización de posición. Se actualiza x_0 a $x_0 + \lambda d_i$ en la dirección que resultó en la menor función objetivo.
 - c) Actualización de direcciones. Se actualizan las direcciones d_i . La nueva dirección d_n se establece en la dirección entre x_0 anterior y el nuevo x_0 , y las demás direcciones se desplazan hacia adelante en el orden.
 - d) Fin de la iteración. Se repiten los pasos anteriores hasta que se alcance un criterio de terminación, como por ejemplo un número máximo de iteraciones o convergencia satisfactoria. El algoritmo converge cuando las direcciones d_i convergen hacia la dirección que conduce al mínimo.

El método de Powell busca iterativamente en múltiples direcciones para encontrar el mínimo. A medida que avanza, ajusta las direcciones para adaptarse a la forma de la función en el espacio de búsqueda (Figura 11). Esto permite que el método explore regiones más amplias y evite quedar atrapado en mínimos locales. Es particularmente útil para funciones con múltiples mínimos locales y que no son altamente no lineales. Sin embargo, puede ser menos eficiente en términos de velocidad de convergencia en comparación con los métodos que utilizan información sobre las derivadas de la función. A pesar de esto, el método de Powell puede ser una herramienta valiosa para problemas de optimización en los que las derivadas no son fáciles de calcular o no están disponibles.

Es importante tener en cuenta que, aunque el método de Powell es robusto en muchos casos, su rendimiento puede variar según la naturaleza de la función objetivo y la calidad de las direcciones iniciales. Por lo tanto, es recomendable probar varias estrategias de inicialización y evaluar la convergencia antes de aplicar el método en situaciones prácticas.

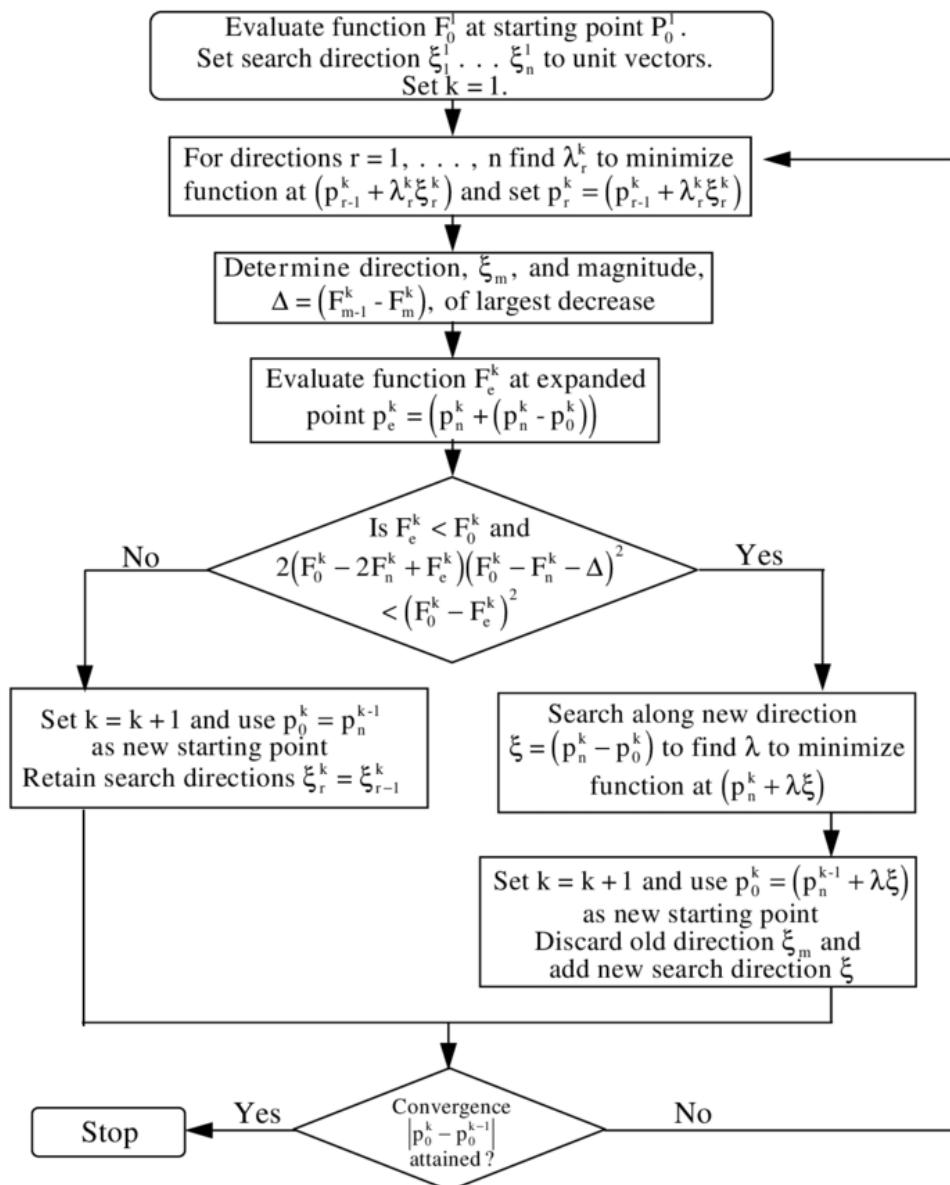


Figura 11. FlujoGRAMA del método de Powell. Fuente: “*A Manual for Use of MTDFREML – a Set of Programs to Obtain Estimates of Variances and Covariances (draft)*” [32]

En principio este método sería implementable ya que no es necesario hallar ni las derivadas de (6) ni exige nada más específico y complejo. El problema que surge es estimar las direcciones. Como ya se ha mencionado previamente, el punto de partida de este trabajo es el ajuste elíptico, que puede proporcionar un x_0 pero no una o varias direcciones para comenzar a iterar. Como en la librería *SciPy* de Python® se dispone de la función *scipy.optimize.minimize* con *method = ‘Powell’*, se realizaron unas pruebas para sondear el algoritmo. No se llegaba a obtener una solución para los parámetros de la superelipse ya que terminaba el número de iteraciones sin llegar a estar cerca de la convergencia. Seguramente será por no proporcionar direcciones específicas que guíen al algoritmo. Podría plantearse como una futura línea de estudio.

2.2.4 Método Nelder-Mead

En el Estado del arte se menciona un último algoritmo para optimizar. Concretamente se obtiene la idea de “*Superellipse Fitting for the Recovery and Classification of Mine-Like Shapes in Sidescan Sonar Images*” [10] de Esther Dura, Judith Bell and Dave Lane, donde lo compara con alguno de los anteriores. En esta sección se detalla su funcionamiento.

Desarrollado por John Nelder y Roger Mead es un algoritmo de optimización no lineal, como los anteriores, que se utiliza para encontrar el mínimo (o máximo) de una función en múltiples dimensiones, siendo especialmente útil para problemas donde no se conocen las derivadas de la función objetivo [33].

Este método se basa en construir un simplex (polígono en 2D, tetraedro en 3D, etc) en el espacio de búsqueda. Dicho simplex se modifica y mueve en cada iteración para explorar distintas regiones del espacio y dar con el mínimo de la función.

Los pasos o desarrollo técnico de este algoritmo son [34] (Figura 12):

- Inicialización del simplex. Se comienza con un simplex inicial en el espacio de los parámetros a optimizar. El simplex destaca por ser un polígono de “n+1” vértices en “n” dimensiones, donde “n” es el número de parámetros a optimizar. Los vértices de dicho simplex se eligen alrededor de la estimación inicial “ x_0 ” de dichos parámetros, influyendo significativamente en la convergencia.
- Iteración. En cada iteración el algoritmo realiza los siguientes pasos:
 - a) Ordenamiento de puntos. Se ordenan los puntos en función de los valores que toman en la función objetivo. Se considera el mejor punto a aquel con el menor valor, el segundo menor el segundo mejor, y así sucesivamente.
 - b) Cálculo del centroide. Se calcula el centroide del simplex excluyendo el peor vértice.
 - c) Reflejo. Se calcula el reflejo del peor vértice con respecto al centroide. El reflejo es una reflexión del peor vértice a través del centroide. Si el valor de la función en el vértice reflejado es mejor que el segundo peor pero no mejor que el mejor, se procede al siguiente paso. Si el valor es el mejor hasta ahora, se procede a la expansión.
 - d) Expansión. Si la reflexión es muy buena (mejor que todos los vértices excepto el mejor), se procede a la expansión. La expansión implica extender el simplex en la dirección del reflejo. Si la expansión mejora la función objetivo, el vértice expandido reemplaza al peor vértice.
 - e) Contracción. Si la reflexión no es suficientemente buena, se realiza una contracción hacia el centroide. Si la contracción es mejor que el peor vértice, pero no mejor que el segundo peor, el vértice contraído reemplaza al peor vértice.
 - f) Reducción. Si ninguna de las operaciones anteriores resulta en una mejora significativa, se procede con la reducción. La reducción implica contraer todo el simplex alrededor del mejor vértice.
 - g) Fin de la iteración. Se repiten los pasos anteriores hasta alcanzar un criterio, máximo de iteraciones o de convergencia. Esta se alcanza cuando el simplex se ajusta alrededor de un mínimo local o global de la función (Figura 13).

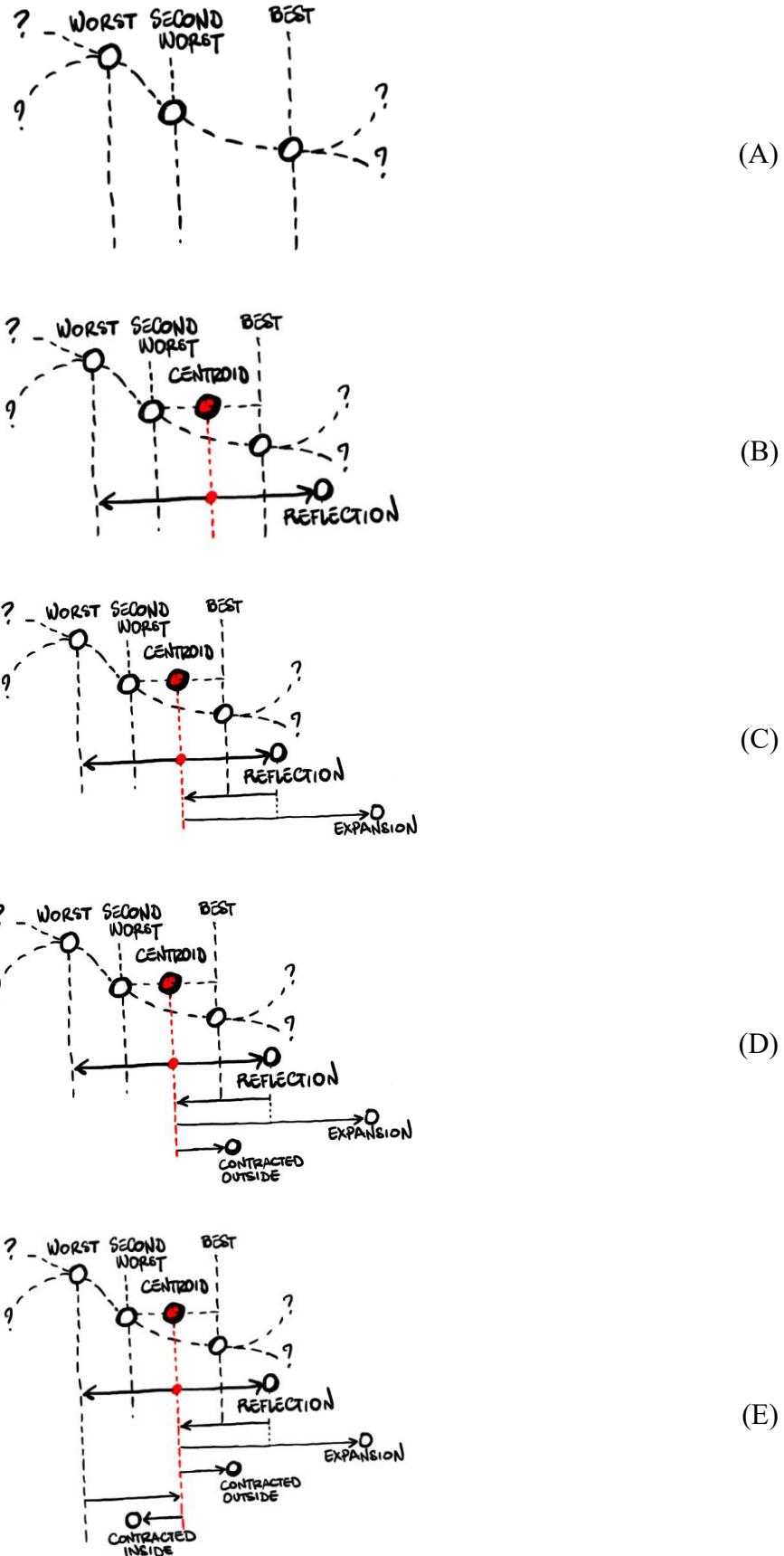


Figura 12. Pasos de una iteración de Nelder-Mead: (A) Ordenamiento de vértices (B) Calculo de centroide y reflejo (C) Expansión (D) Contracción (E) Reducción. Fuente: “Breaking down the Nelder Mead algorithm” [34].

Tras analizar su funcionamiento en detalle, se observa que el método de Nelder-Mead es adecuado para problemas no lineales, donde calcular derivadas sea costoso o complejo. Uno de sus problemas es que no garantiza la convergencia global y puede quedar atrapado en mínimos locales. Este problema es mejor desarrollarlo en el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas. Otro posible problema de este algoritmo es la velocidad de convergencia. Esta puede variar según la forma de la función objetivo y la estimación inicial. Para obtener un tiempo razonable se harán diversas pruebas o ajuste. En general este algoritmo es el que menos inconvenientes tiene y el más factible de emplear. Más adelante, en Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas se mencionará su aplicación y comparación directa con los dos anteriores. Para aplicarlo en el estudio se recurre a la librería *SciPy* de Python® donde se encuentra la función *scipy.optimize.minimize* con method = ‘*Nelder-Mead*’ [35].

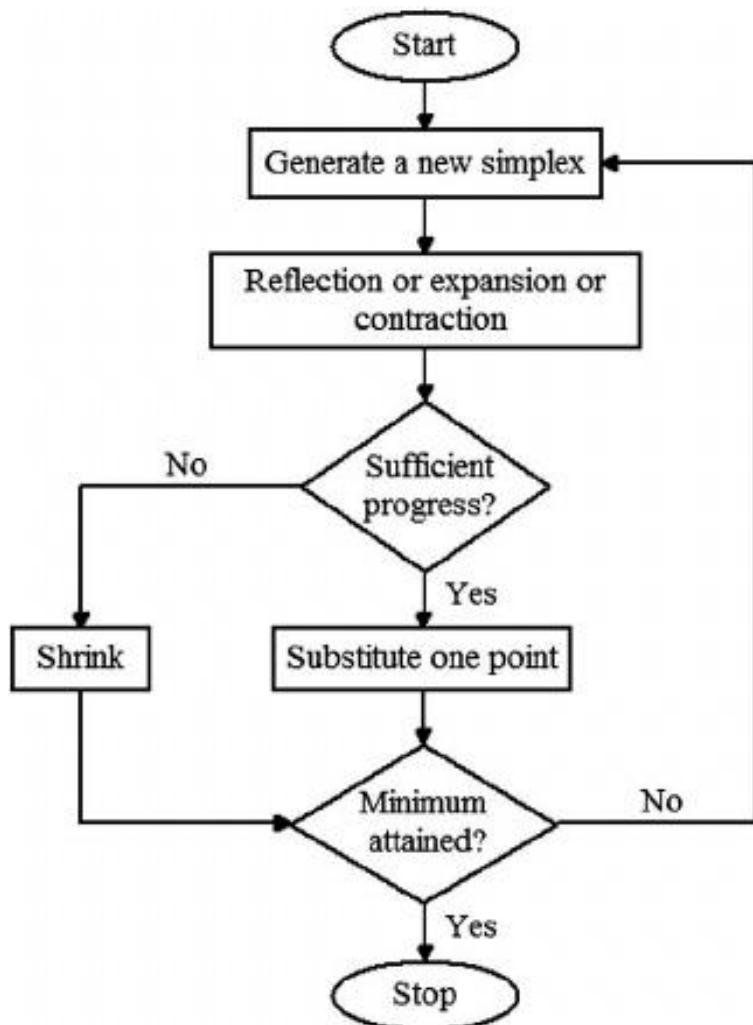


Figura 13. Flujograma del algoritmo Nelder-Mead. Fuente: “*Analysis and Evaluation of Optimization Algorithms Application for Parameter Estimation of Muskingum Flood Routing Models in Rivers*” [36].

2.2.5 RANSAC

En los Antecedentes se mencionó la aplicación de este algoritmo en [1] y que será modificado para este trabajo. Se aprovecha la base de [1] para explicar detalladamente su función y, a continuación, se indican las modificaciones acompañadas con el flujograma del código empleado.

RANSAC es el acrónimo en inglés, en español el algoritmo se denomina “Consenso de muestras aleatorias”. Es un método iterativo y no determinista, publicado por Fischler y Bolles, que a partir de un conjunto de datos con valores atípicos estima los parámetros de un modelo [37].

Para ilustrar su funcionamiento, imagine un conjunto de puntos similar a los que se muestran en la Figura 14. De manera inmediata, uno puede notar que todos los puntos, a excepción de uno, siguen una línea recta. Esto plantea la pregunta de cómo un computador podría copiar el análisis humano. En este sentido, el método RANSAC propone desarrollar una función de coste basada en la distancia mínima cuadrática, que suma las distancias entre los puntos y la línea ajustada. De manera iterativa, se selecciona un subconjunto de puntos para ajustarlos a una línea, eligiendo finalmente la línea que resulta en el costo más bajo.

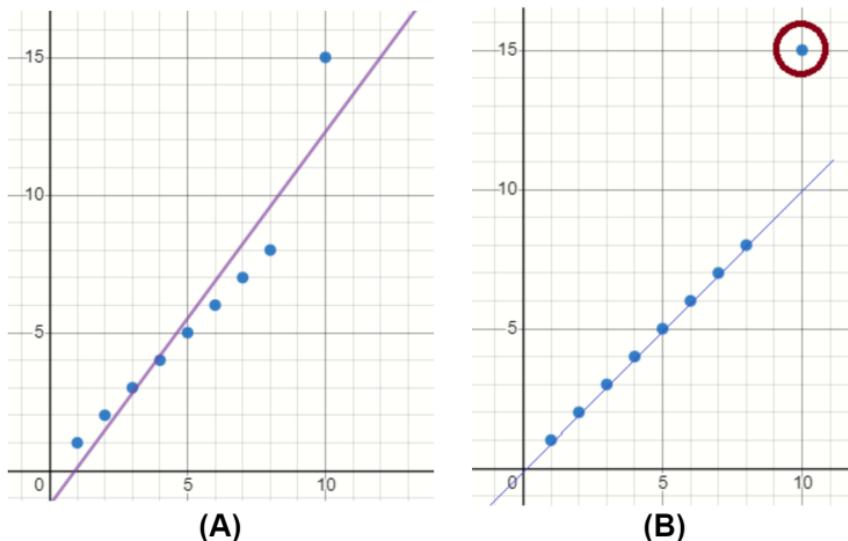


Figura 14. Regresión lineal (A) convencional y (B) con RANSAC. Fuente "Outlier detection using the RANSAC algorithm" [38].

El empleo al detalle de RANSAC en la Figura 14 es [38]:

- Seleccionar una muestra de “n” puntos del conjunto aleatoriamente (población).
- Utilizar para determinar la ecuación que se ajusta a los “n” puntos la regresión de mínimos cuadrados.
- Determinar y almacenar la media de la distancia de cada punto de la población a la recta. Si la media es menor que la mejor puntuación de iteraciones anteriores, se descartará la ecuación lineal más antigua y se selecciona la actual.
- Vuelta al primer paso y comienzo de una nueva iteración hasta completar el número de iteraciones especificado. Al final de dichas iteraciones habrá una ecuación lineal que supone la mejor recta.

Por la propia naturaleza iterativa del algoritmo, no determinista, es posible que el modelo no sea el mejor posible. No obstante, es un algoritmo más que útil para implementar con los ajustes geométricos correspondientes. Con este algoritmo aseguramos que el implante no sobrepase el

contorno al que se ajusta la prótesis, condición necesaria. Todos los ajustes estudiados y a estudiar necesitan de una integración de este algoritmo posterior al ajuste (Figura 15).

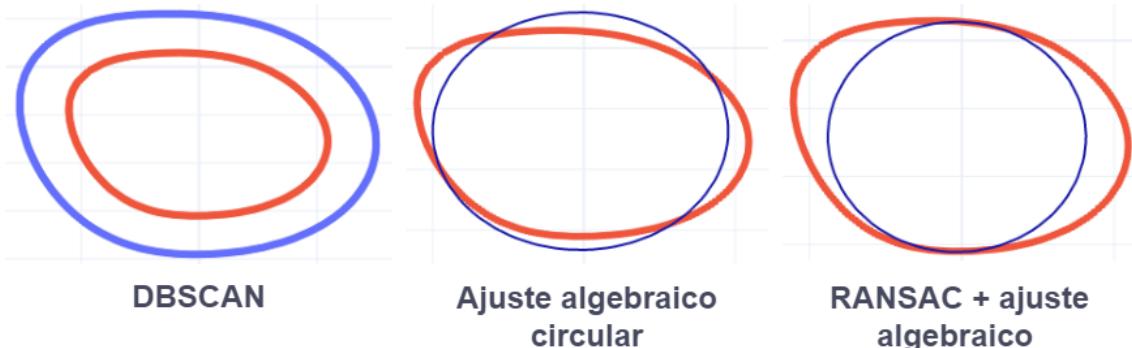


Figura 15. Resultados de la integración de RANSAC al ajuste algebraico circular. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].

En el marco teórico de la distancia algebraica se destacó la utilidad que ofrecía el signo de esta. Pues aquí se ve empleado, ya que el algoritmo RANSAC modificado aprovecha dicho signo para determinar si la curva se sitúa dentro o fuera del contorno.

El algoritmo RANSAC modificado (Apéndice 11: Modificación de RANSAC), que es el que se emplea en este trabajo, sigue el procedimiento descrito para el caso de la regresión lineal. Selecciona una muestra aleatoriamente de “n” puntos (*sample*) que conforman el borde interno (data) y se ajusta la curva que se adapta a ellos (*param_current*). La distancia algebraica, en este caso la llamada a la función superelipse, será la función de coste, con signo positivo si el punto está fuera y negativo si estuviese dentro de esta. Por ello, si la distancia de un punto de la muestra a la curva fuese menor o igual a cero se almacena (*inliers*). Si hubiese tres o más puntos almacenados (mínimo de puntos necesarios para representar una geometría en un plano 2D), se obtiene el vector de parámetros que se amolda a ellos (*param_new*) y se halla la distancia algebraica de dicha población, extrayendo su media (*mean*) y desviación típica (*std*). Si la resta de estas (*mean-std*) fuese mayor a 0 se almacenan los parámetros (*l_param.append*). Suponiendo que la distancia algebraica de la población sigue una distribución normal, al cumplirse esa condición se garantiza que al menos el 84% de puntos del contorno se encuentran fuera de la curva ajustada. Por si se desobedeciese dicha condición, se hace otro filtro a los *inliers*. El algoritmo se ejecuta tantas veces como haya determinado el usuario (*max_iter*) y da como resultados los parámetros de la superelipse ($x_c, y_c, a, b, \Theta, n$), con la métrica (media menos desviación estándar) más alta (*param_def*).

En el Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas se detallará más acerca de los hiperparámetros, *max_iter* y *n*, empleados en cada caso de estudio.

Capítulo 3: Automatización en el diseño de implantes personalizados basado en curvas

3.1 Óvalo

Como se explicó en el apartado de *Distancia algebraica y mínimos cuadrados*, dicha metodología se basa en la ecuación implícita y el vector de coeficientes correspondientes. Aquí es donde surge el problema para el estudio del óvalo. Como hemos mencionado anteriormente en el marco teórico del *Óvalo*, al sustituir los valores “a” y “b” en (2), constantes en la elipse, por distintas funciones (3), no se puede llegar a obtener una ecuación implícita como tal. Ya que dependiendo de las funciones y variables que se empleen, la forma de la ecuación implícita que podría definir un óvalo varía tanto en grado como número de variables. Por lo que no es posible disponer de una ecuación genérica con la que trabajar y la cual minimizar. Se tendría que usar un método iterativo para delimitar rangos o emplear una estimación inicial que nos asegure llegar a una solución, pero de nuevo, dependiendo de muchos factores. Se opta por dejar ese posible caso de estudio para un futuro o usar otra metodología en esta geometría.

3.2 Superelipse (polinomio implícito)

Para la comprensión del planteamiento en el que se profundiza a continuación es necesario el apoyo en la teoría explicada en el apartado correspondiente del Capítulo 2: Marco Teórico, destacando de ahí la ecuación (6), sin olvidar el razonamiento empleado para llegar a dicha ecuación. Agregándole la información de la metodología desarrollada en *Distancia algebraica y mínimos cuadrados*. No olvidar la causa de que únicamente se empleen valores pares de “n”, para tener superelipses con simetría par.

Teniendo en cuenta la ecuación (6), y siguiendo la justificación de que “n” sea par. Al expandir dicha ecuación, juntar los términos con mismo grado en “x”, “y” o los monomios que combinan “x” e “y”, agrupar los multiplicadores o términos que multiplican a cada variable o monomio (cosenos, senos y otras variables) se obtiene un polinomio implícito para la superelipse, similar a (9). Como en (9), también se dispone de un vector de coeficientes del estilo ($v = [A \ B \ C \ D \ E \ F]^T$), con mayor número de coeficientes. En este caso dichos coeficientes vendrán de la agrupación de los multiplicadores mencionados recientemente. Hallada la ecuación implícita correspondiente, se seguirá el mismo proceso que el estudiado en el apartado *Distancia algebraica y mínimos cuadrados*.

Como prueba inicial, se intenta una primera aproximación al método seguido en [1] para una superelipse de grado $n = 4$. Previo a la creación de las funciones correspondientes se emplea la herramienta de cálculo simbólico SymPy de Python® [39]. Con ella se puede expandir la ecuación (6) evitando errores de cálculo y sin descuidar algún monomio. Tras comprobar que ningún término se anule. Juntando términos de mismo grado y nombrando los coeficientes según letras del alfabeto, en vez de incluir los términos expandidos, resulta un polinomio implícito de la siguiente forma:

$$\begin{aligned}
 D_A(p_i, v) = & Q(p_i, v) \\
 = & A * x_i^4 + B * x_i^3 * y_i + C * x_i^2 * y_i^2 + D * x_i * y_i^3 + E * y_i^4 + F * x_i^3 + G \\
 & * x_i^2 * y_i + H * x_i * y_i^2 + I * y_i^3 + J * x_i^2 + K * x_i * y_i + L * y_i^2 \\
 & + M * x_i + N * y_i + O
 \end{aligned} \quad (17)$$

Para seguir los pasos de [1] se han modificado lo que eran sus funciones *elliptical_fit*, *elliptical_distance* y se ha editado la función de representación. Las dos primeras se crearon de cero, siendo una adaptación teniendo en cuenta (17) y su número de términos, y en la función representación únicamente se ha editado una de sus líneas. El resultado es la función *superelipsen4_fit* copiada en Apéndices (Apéndice 1: Función *superelipsen4_fit*).

Comparando, se concluye que lo único que cambia del proceso es el polinomio implícito. Con distintos grados ocurrirá lo mismo, siendo cada vez un polinomio más extenso y laborioso de emplear. Sucediendo lo mismo con las distintas funciones a emplear.

Con esta primera prueba, *superelipsen4_fit*, se obtienen los siguientes resultados:

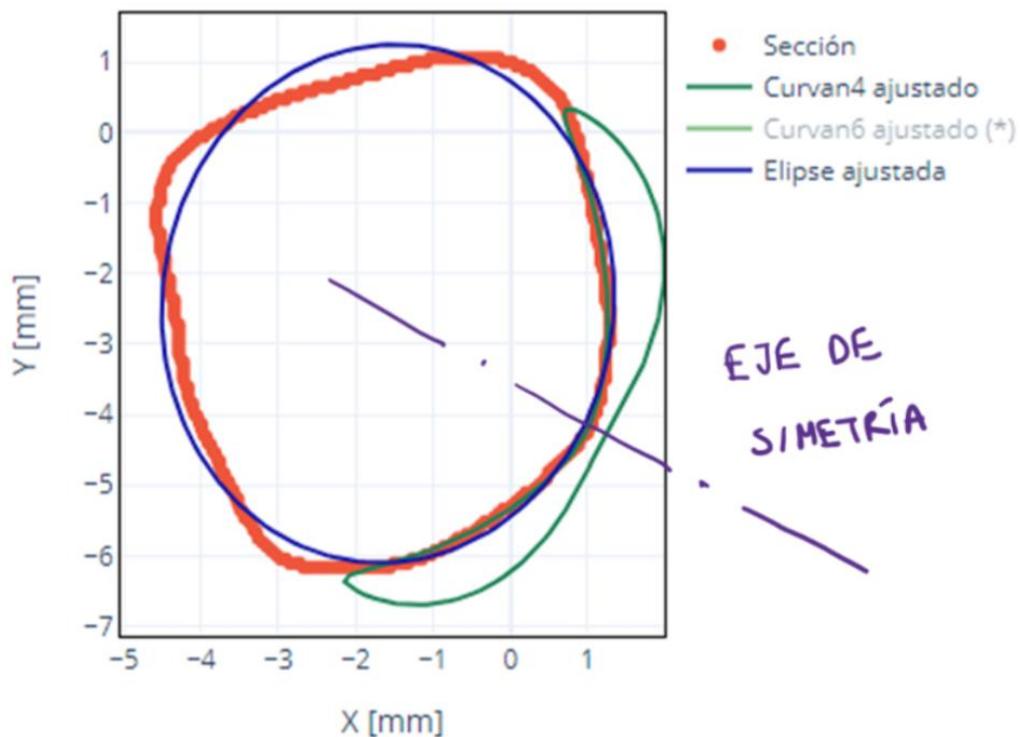


Figura 16. Representación del ajuste con *superelipsen4_fit*.

```

[-0.0225851  0.00735299 -0.03249703  0.00088134 -0.01003004 -0.12891629
-0.13022245 -0.09937817 -0.09628616 -0.00295439 -0.50782853 -0.11491556
 0.52053184  0.54251157 -0.33042055]

```

Figura 17. Vector de coeficientes resultante del ajuste anterior.

Pese a lograr obtener un primer resultado, logrando hallar el vector de coeficientes (Figura 17) que determina el mejor ajuste, y la geometría de dicha solución (Figura 16). A la hora de representar los valores, se observa que el ajuste no es correcto. La forma de la curva no coincide con una superellipse, pese a cumplir parte de la simetría (simetría en uno de sus ejes) mencionada anteriormente, no se obtiene la forma correspondiente a una superellipse de grado $n = 4$.

Puede que intentar ajustar al contorno interno de una imagen tomográfica no haya sido el mejor ejemplo para comprobar el correcto funcionamiento del algoritmo. Para ello, con el apoyo de la herramienta GeoGebra [40], se representa una superelipse de grado $n = 4$ cualquiera y se muestrean “ m ” puntos, siendo “ m ” el número de coeficientes a buscar. Con eso se podrá resolver un sistema “ m ” x “ m ” y encontrar la solución. Sacando las coordenadas de dichos puntos y colocándolos en orden se comprueba que el código encuentra una superelipse de grado $n = 4$ que ajuste a los puntos (Figura 19). Pero comparando los coeficientes (Figura 18), no coinciden. Puede ser por varias razones, como unidades de Θ , distancias proporcionales, etc. Pasando esto último por alto, se ha logrado comprobar que la función puede ajustar una superelipse de grado 4.

```
[ 2.39810999e-05  9.88481605e-06  1.05536739e-05 -3.72352795e-07
 5.06807874e-06  1.06937162e-03  6.43256082e-04  1.78975137e-04
 3.48055063e-04  2.03196512e-02  9.10463722e-03  9.75821508e-03
 1.75711328e-01  1.37951346e-01  9.74424055e-01]
```

Figura 18. Vector de coeficientes para el ajuste empleando *superelipsen4_fit* a $m = 15$ puntos.

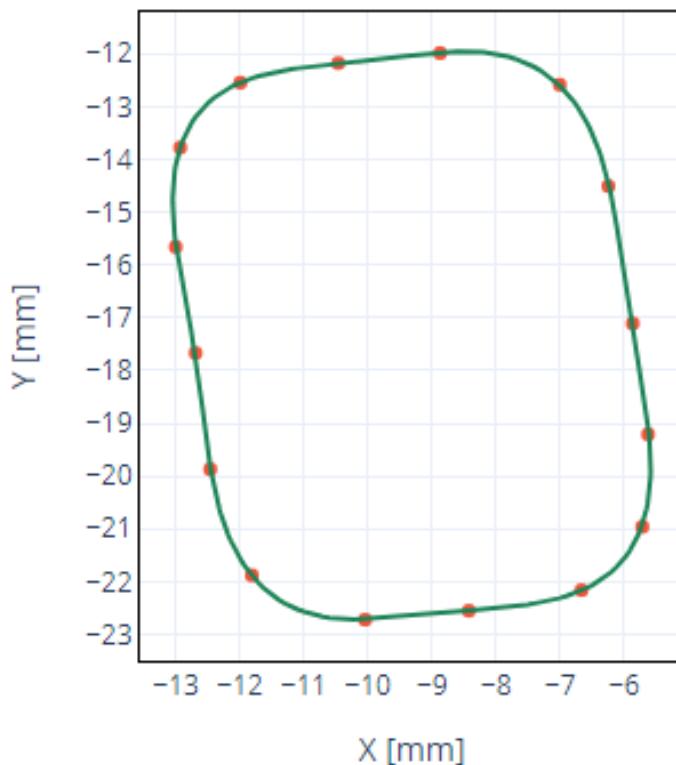


Figura 19. Representación del ajuste con *superelipsen4_fit* a $m = 15$ puntos.

La siguiente prueba será similar, pero en vez de puntos que se sabe que pertenecen a una superelipse, serán aleatorios. De nuevo, al menos se deben emplear el mismo número de puntos que coeficientes a ajustar. Tras probar con puntos aleatorios, se observa que no ajusta ni la función de la superelipse de grado cuatro ni la de la elipse [1]. Como la función *elliptical_fit* debería ajustar, se crea un código que ordene en sentido horario los puntos aleatorios respecto del centroide de estos. Esto puede probar si dicha condición afecta. Con los puntos ordenados, la elipse devuelve una posible solución, pero la superelipse sigue sin ajustar, resultando una curva similar a una parábola. Lo mismo sigue sucediendo con un mayor número de puntos.

Probando también para $n = 6$, revisando detenidamente los monomios y su agrupación al expandir la ecuación (6), se comprobó que el fallo no venía a la hora de encontrar o formar la ecuación implícita a ajustar. Por lo que se comenzó a buscar algún caso similar de estudio o información que pudiese ayudar a entender lo que sucede.

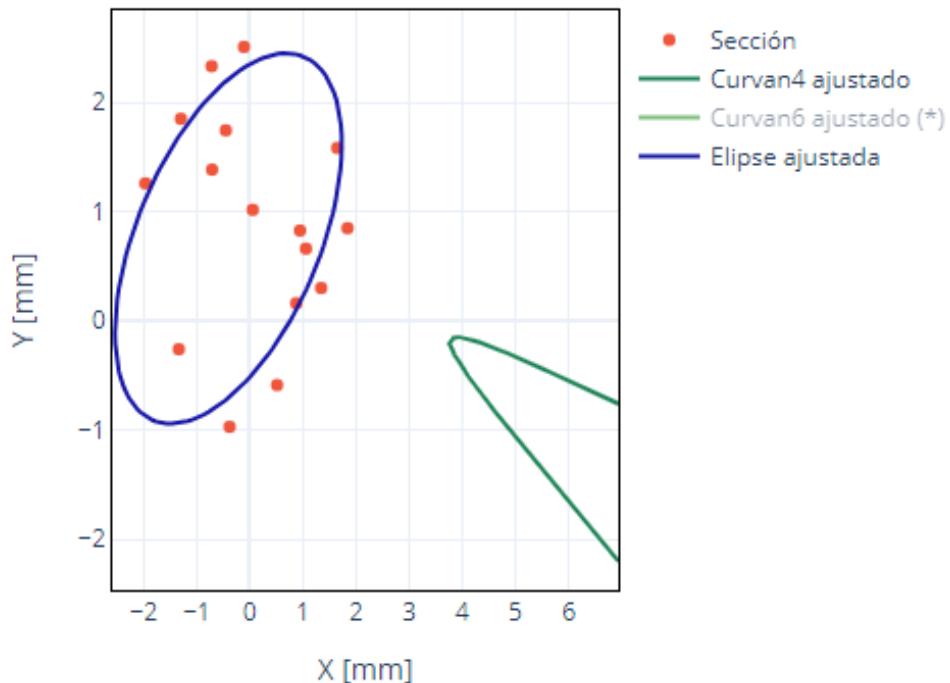


Figura 20. Intento de ajuste con *superelipsen4_fit* a $m = 15$ puntos aleatorios.

Investigando estudios similares, se encontró el *paper “Fitting Superellipses”* [41], de Paul L. Rosin, acerca del ajuste de superelipses. El trabajo estudia el ajuste a tendencia de datos, pero se puede extrapolar o comparar directamente con el ajuste a curvas. En dicho documento, el apartado 4.1, confirma la similitud del estudio al emplear también la distancia algebraica en el ajuste. Y en dicho apartado resalta que el ajuste para el caso de la elipse, en particular, dispone de una solución para la minimización de forma cerrada. Para el resto de superelipses, o grado $n > 2$, puede no darse. Es por eso por lo que aparecen geometrías similares a paráolas. Para ello, el propio documento da ideas de cómo solucionar el problema, pero a la vez expresa los posibles errores o anomalías en resultados [41].

3.3 Superelipse (optimización de parámetros)

Con los resultados obtenidos se plantea cambiar el enfoque de trabajo de *Distancia algebraica* y *mínimos cuadrados*. La base de la optimización será la misma, la distancia algebraica. Pero no se seguirá con el planteamiento basado en autovalores, ya que se ha comprobado que no se obtiene el resultado correcto. La optimización pasa a enfocarse en los parámetros que definen la ecuación (6) de la superelipse. La ecuación (6) será la nueva distancia algebraica, trabajando directamente sobre esta en vez de un polinomio implícito. Dicha función cumple la función de distancia algebraica, puesto que al sustituir las coordenadas de los puntos correspondientes en los “x” e “y” de (6), se obtiene: un valor igual a cero si el punto pertenece a la superelipse concreta, positivo si es externo a la curva o negativo si fuese interior.

3.3.1 Contexto

Debido a la justificación explicada en el Capítulo 2: Marco Teórico, se debe trabajar con grados de superelipses pares, por la simetría y ventajas que ofrecen. Para limitar el estudio se opta por un límite superior de $n = 10$. A partir de dicho valor, el cambio de curvatura de sus vértices es despreciable visualmente, acercándose cada vez más a un rectángulo. Dependiendo de la imagen a ajustar esto podría suponer una ventaja o no, fin de este estudio.

Empleando como base el trabajo realizado en [1], se requiere de una función que defina la geometría a ajustar y, a la vez, la distancia algebraica. Para ello se traslada la ecuación (6) a una función *superellipse* (Apéndice 2: Función superellipse). Primero, habría que diferenciar entre variables y datos. Los datos serán las coordenadas “x” e “y” de los puntos de la curva a la que se quiere ajustar dicha geometría. Las variables serán los parámetros que definen la superellipse y se buscan optimizar. En este caso: $x_c, y_c, a, b, \Theta, n$. Por tanto, en la definición y llamada a la función habrá dos entradas: un vector de *variables* con $(x_c, y_c, a, b, \Theta, n)$ y *data*, con las coordenadas (x, y) de los puntos. Más adelante, en el apartado de versiones se desarrollará la llamada a esta función y como afecta a los datos. Como es de esperar, dicha función devuelve la distancia algebraica con su signo, o un vector con todas si los datos contienen a más de un punto. Por lo que esta función es de gran importancia en el estudio y otras funciones derivarán de estas.

Tomando como base los trabajos mencionados en el Estado del arte se busca el ajuste de la superellipse mediante métodos de optimización de sus variables. Para ello se detalló en Ajuste y optimización los posibles métodos a emplear. En dicho apartado se explicó cada método en particular en profundidad. Ahora sólo se resumen (Tabla 1) y enuncian las ventajas de cada uno y las causas de la elección final.

Tabla 1. Resumen métodos de optimización.

<i>Método de Levenberg-Marquardt</i>	<i>Método de Powell</i>	<i>Método de Nelder-Mead</i>
Empleado para problemas de ajustes, basado en la aproximación de la función objetivo con una función lineal y cuadrática.	Basado en un algoritmo de línea de base y una matriz de direcciones.	Método de optimización no lineal que no requiere información sobre las derivadas de la función objetivo.
Especialmente útil cuando se conocen las derivadas parciales de la función objetivo.	Para mejorar la convergencia aprovecha las propiedades de las funciones cuadráticas en varias direcciones.	Estrategia basada en el empleo de un simplex para converger hacia un mínimo local.
Eficaz, al basarse en la información derivativa a la hora de encontrar el mínimo local.	Eficiente para problemas de varias dimensiones. Debido a la combinación de optimización unidimensional con multidimensional.	Al no utilizar información derivativa es adecuado para funciones objetivo no suaves o con discontinuidades.
Sensible a las condiciones iniciales. Convergiendo a mínimos locales.	No es necesario proporcionar derivadas.	En problemas altamente dimensionales o con valles profundos el algoritmo tarda en converger, es lento.
	Adecuado para funciones objetivo con valles estrechos o curvaturas complejas.	Dependiendo del simplex inicial puede quedar atrapado en mínimos locales.

Tras resumir las distintas metodologías, se enuncian las principales razones por las que se cree que el método de Nelder-Mead es el más adecuado para este trabajo.

- La superelipse se considera una curva no suave, por sus esquinas pronunciadas debido a “n”. Dicho método puede trabajar con este tipo de funciones.
- No se requiere del uso de las derivadas o hessiana. Puede ser costoso y laborioso, por lo que se ahorra su uso. Esta ventaja de Powell y Nelder-Mead sobre Levenberg-Marquardt, ayuda en la decisión.
- Para un enfoque multidimensional alto podría ser más útil el método de Powell. Pero esto implicaría un conocimiento de direcciones a estimar. Como se carece de dicho conocimiento, y el caso de estudio implica un número de parámetros moderado, es más sencilla la aplicación de Nelder-Mead.

Para emplear este método se recurre a la librería *SciPy* de Python® donde se encuentra la función *scipy.optimize.minimize* con *method = ‘Nelder-Mead’* [35]. Habrá que proporcionarle una serie de entradas a dicha función para resolver la optimización con dicho método.

La primera entrada es *fun* donde habrá que especificar la función que se quiere minimizar o la función objetivo. Como idea general, el objetivo será minimizar la distancia entre los puntos del contorno de la imagen médica con la superelipse correspondiente. Siguiendo esa idea podría definirse la función objetivo como el sumatorio de las distancias de los puntos del contorno a la superelipse a optimizar. Empleando la función *superellipse* (Apéndice 2: Función superellipse) para obtener dichas distancias. Por tanto, se busca minimizar el sumatorio, para obtener la superelipse más próxima a toda la sección de la curva.

Este planteamiento es mejorable, gracias al enfoque de los mínimos cuadrados, elevando cada término del sumatorio al cuadrado. Como en cualquier problema de optimización o minimización se busca un resultado global, lo más rápido posible y con el menor error posible. Con este cambio en la función objetivo se logra:

- Convertir la función en estrictamente positiva, simplificando el análisis matemático y asegurando una única solución mínima si la función objetivo es convexa.
- Penalizar más los errores grandes. Al elevar al cuadrado el error, o diferencia entre valores predichos y observados, se amplifica el impacto de las diferencias más grandes. Teniendo más impacto en la función objetivo, deseable para la optimización.
- Aumentar la velocidad de convergencia. La función objetivo cuadrática suaviza la curva en comparación con su predecesora. Pese a que Nelder-Mead no necesita las derivadas, estas y los gradientes son más fáciles de calcular.
- Al dar un mayor peso a los valores atípicos (errores grandes) se gana robustez. El ajuste del modelo se ve afectado más por ellos, influyendo en la solución. Dicha solución se adaptará mejor a ruido o mediciones dispares.

Aplicando este detalle se obtiene la función objetivo *fobjetivo* (Apéndice 3: Función objetivo a minimizar), que será implementada en la variable o entrada correspondiente a *fun*.

La siguiente variable o término que se necesita detallar a la hora de llamar a *scipy.optimize.minimize* es x_0 . Corresponde a una iteración o estimación inicial. Como ya hemos mencionado, y se volverá a hacer debido a su importancia, influye mucho en la convergencia a un mínimo local. Como inspiración de otros estudios se tomarán en general los parámetros hallados con *elliptical_fit* y *elliptical_parameters* [1]. Pese a que se menciona como general, en los subapartados y flujo de trabajo se detallará más de su uso.

Como ya se mencionó habrá que seleccionar el método de optimización o minimización: method: ‘Nelder-Mead’, y no es necesario emplear la jacobiana y hessiana.

En todo algoritmo de optimización para delimitar su búsqueda se suele añadir un rango a los parámetros que se quieren optimizar. Dichos rangos se juntan en un vector y se especifica en *bounds*. En el siguiente subapartado se especifican los rangos con los que se han trabajado y como se han ido acotando para mejorar resultados.

Para terminar, habría que añadir *options* donde se detalla el número máximo de iteraciones a llevar a cabo. De nuevo, dicho valor ha sido modificado a lo largo del estudio.

3.3.2 Versiones y desarrollo de funciones

Compartida la información anterior, se da comienzo a la prueba de distintos algoritmos y primeros ajustes, pasando a resumir los principales problemas de código o planteamiento que han ido surgiendo y como se han solucionado.

- Función Superelipse (Apéndice 2: Función superelipse)

Se mencionaron las entradas necesarias y su inspiración en la forma de (6). En principio era tan sencillo como nombrar a cada parámetro con la componente correspondiente del vector *variables* y copiar la ecuación (6) en el código. Pero, como se trabajaba en [1] y se verá en el flujo de trabajo, tras un primer ajuste, hay que emplear una modificación de RANSAC (Apéndice 11: Modificación de RANSAC), y así lograr que el ajuste sea interno al contorno. Pues en dicha función se necesita llamar a *superelipse* para saber el signo de la distancia algebraica de cada punto y ver si es interior o no. Por lo que la función trabajará con dos formatos de *data*. Para el ajuste trabajará con una matriz compuesta de vectores cuyas componentes son las coordenadas “x” e “y” de cada punto. Y en la llamada a RANSAC, trabajará punto a punto con un vector y solo sus coordenadas (x, y). Es por esto por lo que en dicha función se observan dos diferenciaciones.

- Minimización inicial

En los siguientes puntos se van a resaltar las modificaciones a las entradas o variables empleadas en *scipy.optimize.minimize*, junto con su uso, para obtener el mejor ajuste posible.

En la primera prueba no se especifica mucho el rango de los parámetros, únicamente se acota el grado entre 2 y 10. Como iteración inicial se toman los parámetros del ajuste elíptico de [1] añadiéndole el grado correspondiente, comenzando por el 4. Al comprobar que la función ajustaba correctamente, entendiendo por correctamente que fuese capaz de devolver una superelipse, cumpliendo las propiedades como la simetría mencionada, se repitió dicha llamada a la función cambiando en cada caso el grado. Confirmando que la solución fuese un vector con las variables ($x_c, y_c, a, b, \Theta, n$) que definen la curva ajustada.

Tras realizar las primeras pruebas se observa que la función únicamente encuentra soluciones dentro del grado que se especifica en la estimación inicial. Ya se mencionó en la teoría, pero uno de los problemas que tiene el método de Nelder-Mead es que puede quedar atrapado en un mínimo local, omitiendo gran parte de la región de estudio, dependiendo de los valores que se le propicien.

- Minimización (con vector de estimaciones iniciales)

Para esquivar este primer problema se plantea la opción de usar un vector de vectores como estimación inicial en lugar de un único vector. Antes se definía un vector con los parámetros de la elipse, añadiéndole el grado correspondiente a cada prueba. Ahora dicha modificación formaría parte de cada vector dentro del vector *initial_guesses*. Lo único que se ganó fue en la automatización de las pruebas, ya que siempre acababa resultando un ajuste con el grado correspondiente al del último vector.

Como solución a este contratiempo, se añadió, en el bucle que minimizaba cada vector dentro de *initial_guesses*, una comparación. Para comparar, se llamaba a una nueva función que devolvía la media de los errores o distancias del ajuste al contorno, usando la ecuación (6). Y dentro del bucle se guardaba el ajuste con menor media. Más tarde, al unificar y hacer limpieza de códigos se vio que el fin de dicha función era tan simple como usar *mean* de la función *superellipse*, usando como vector *variables* aquel correspondiente a los parámetros ajustados. Por lo que la función desde un principio era innecesaria.

Con estas pequeñas modificaciones se empezaron a dar soluciones distintas en las pruebas. Al comparar con la elipse y sus parámetros tanto visualmente, como los parámetros, se comprobaba que los resultados estaban lejos de los esperados.

- Minimización global

Para intentar aproximarse a los resultados que se esperaban o buscaban se empezaron a modificar los rangos de los valores. En las primeras pruebas se empleaban los valores más altos y bajos de las coordenadas (*x*, *y*) de los puntos que definían el contorno.

A la hora de crear los rangos, es necesario conocer dichas coordenadas. Si a esto le sumas el bucle mencionado anteriormente, empieza a surgir una mezcla de funciones y datos. Como solución, y para mejorar la comprensión del código de futuros lectores, se unificó todo en una función *minimización_global*. Dicha función tenía como entradas: *initial_guesses*, *fobjetivo*, *data* y la función empleada para comparar. Más tarde, se eliminaron *fobjetivo* y, como se mencionará, la inutilidad de la de comparación.

Para los rangos de x_c , y_c , a , b se hallaban las coordenadas en “x” y en “y” más alta de los puntos del contorno. Las cuales se multiplicaban por un factor positivo o negativo dependiendo de si era el límite superior o inferior del intervalo. El ángulo tenía de cero a 360 grados, por lo que su intervalo incluía cualquier posible solución. El grado continuaba limitado entre 2 y 10.

Mejorados los resultados, se comenzó a pasar los ajustes resultantes por la función RANSAC (Apéndice 11: Modificación de RANSAC). Como con el caso de la función *superellipse*, en *minimizacion_global*, habría que separar en dos casos, y cada uno con dos situaciones distintas. La primera de ellas es que se pasaba de llamar a la función con *initial_guesses* a volver a usar de entrada un único *initial_guess* correspondiente al resultado de un primer empleo de *minimización_global*. Por lo que esta función debería de ver la dimensión de *initial_guesses* (nombre dentro de la función, no afecta a su dimensión) para ver si usar el bucle o no. Después, como los rangos de las fronteras de búsqueda dependen de la entrada *data* para delimitar los intervalos, habrá que separar entre el uso de la función en el primer ajuste y cuando es llamada por RANSAC. La solución se basa en la dimensión de *data* como anteriormente, pero en ambos casos se agrupó la coordenada o coordenadas en unas variables “X” e “Y”. Porque si no, para el caso RANSAC, al solo entrar un punto, este proporcionaría el mínimo y máximo valor de las coordenadas “x” e “y”.

Solucionados dichos detalles se obtuvieron los primeros resultados post RANSAC. Se observó que dichos resultados solían coincidir o estar muy próximos a la elipse. Y en caso de resultar de otro grado no eran del todo buenos visualmente. Con los parámetros de cada ajuste delante se notó que el ángulo era muy próximo al ajuste elíptico en todos los casos. Este detalle parece mínimo, pero influye bastante en el antes y después de RANSAC, ya que las geometrías de distinto grado podrían ajustar mejor con la rotación. De nuevo la convergencia local distorsiona negativamente los resultados.

- Minimización ángulo (Apéndice 9: Función minimizacion_angulo)

El planteamiento por el que se opta para hacer frente a dicha problemática es la creación de una variable *angulo* dentro del código que varie su valor para ampliar el rango de búsqueda. Todo ello en una nueva función *minimización_angulo*, basada en *minimización_global*. En esta, se recurre a un bucle *for* para que dicha variable tome los distintos valores. Al estimar qué valores darle y qué salto tomar se observó que el rango de búsqueda, (0, 360) que se había marcado, era inútil, puesto que al ser simétrica la geometría y poder cambiar las longitudes de sus ejes, con especificar (0,180) ya se valorarán todas las soluciones posibles. Después, se vio que dicho rango era incoherente con los valores empleados para *angulo* en el bucle. Para reducir el tiempo de optimización se acotó el bucle de prueba de ángulos. Para ello, se empleaba el ángulo de *initial_guess*, proveniente de otro ajuste, y se sacaban los límites del bucle con la suma de más/menos 60° a dicho valor (pudiendo generar, y por tanto tomar *angulo*, valores negativos). Pasando a probar ajustes dentro de ese rango con saltos de 15°. Importantísimo, que fue un error corregido posteriormente, las funciones *np.cos*, *np.sin* y *scipy.optimize.minimize* trabajan mejor y normalmente con radianes. Para el caso del ángulo en *bounds* se especificó el rango de soluciones: (-np.pi, np.pi).

Era de esperar que esta implementación aumentase el tiempo necesario para que se hallase el ajuste correspondiente. Especialmente con la implementación de dicha función dentro de RANSAC (Apéndice 11: Modificación de RANSAC). Como se mencionó en el apartado teórico de Ajuste y optimización, aquí se desarrollará más acerca de los hiperparámetros: *max_iter* y *n*. En un principio se empleaban los valores de [1]: 5000 y 20, respectivamente. Pero al llegar a este punto del estudio, donde el tiempo de iteración se incrementó en exceso, se comenzó a bajar el número de iteraciones subiendo el número de puntos que compusiesen la muestra. De todas formas, para este caso, no tuvo apenas influencia. El código llegaba al punto de detenerse por excesivo tiempo en funcionamiento.

Hasta ahora, el flujo de trabajo consistía en usar *minimizacion_angulo* con los parámetros de la elipse ajustada y filtrada con RANSAC, y posteriormente emplear la modificación de RANSAC (Apéndice 11: Modificación de RANSAC), llamando de nuevo a *minimización_angulo*. Este último paso era el que suponía un exceso de tiempo. Para no perder la ventaja de considerar distintas opciones de ángulos, se optó por sacar la variable *angulo*, y que de esta manera pasase a ser una entrada de la función. Pudiendo sacar el bucle de esta. Aunque de primeras parece que no tiene influencia, se ahorró bastante tiempo. Antes, al disponer de la prueba de ángulos dentro de *minimización_angulo*, en cada iteración de RANSAC se llamaba más de una vez a dicha función. Por tanto, en cada llamada valora todos los ángulos. Por lo que en una iteración de RANSAC se estudiaban mínimo dos veces seis valores de ángulos. En cambio, al sacar dicho bucle, en cada llamada a RANSAC sólo se estudiaba un único valor. Pasando de realizar las *max_iter* con distintos valores varias veces, a realizar seis casos distintos de *max_iter* cada uno. Viendo el ahorro de tiempo que esto suponía se copió la metodología con el estudio del grado. No se eliminan las dos opciones posibles, dependiendo de la entrada *initial_guesses* y su dimensión, por si en un futuro se necesitasen. Al estar adaptada al caso de dimensión 1, se

aprovecha un bucle externo que llama uno a uno a las estimaciones iniciales de cada grado. Para no perder la función de *minimización_angulo*, que será empleada de una manera concreta (reflejado en el apartado de flujo de trabajo), se crea *minimización_simple* (Apéndice 10: Función *minimizacion_simple*).

- Comparación

Tras unas pruebas de esta metodología, se obtenían únicamente soluciones similares a la elipse, ósea, grado 2. Revisando el código, siendo más exhaustiva dicha revisión en la parte de la comparación de medias, se encontró el error.

La propia comparación mediante la media de los ajustes de distintos grados es incorrecta. En el fondo, cada grado implica una función *superelipse* distinta, con distinto “n”. Por ello, la comparación se basaría en varias funciones, lo que no es del todo correcto. Esto no quiere decir que la función *superelipse* sea incorrecta para su empleo en el trabajo o ajuste. Dicha función cumple con lo que se espera: con su llamada en *fobjetivo* se busca ajustar una *superelipse* definida por sus parámetros al contorno correspondiente y con su llamada en RANSAC (Apéndice 11: Modificación de RANSAC) se diferencia entre puntos internos o externos al contorno. A la hora de comparar, pasaría como con la minimización o ajuste, habría que estudiar por grados separados. En ambos casos la “n” influye en los resultados. Pero eso ya se tuvo en cuenta debido a que *scipy.optimize.minimize* se veía muy influenciado en la convergencia por su estimación inicial y el grado de esta. En cambio, para la comparación de ajustes, con la menor media se buscaba el mejor ajuste sin importar el grado. Y ese planteamiento es erróneo.

Para aclarar la situación, en todas las llamadas a *superelipse* se sustituirían en (6) los valores de las coordenadas de (x, y) y de las variables ($x_c, y_c, a, b, \theta, n$) tras ser optimizadas. Dicha función devolverá una distancia entre la *superelipse* y el contorno. Pero esa distancia dependerá de las variables, no estando normalizada o no siendo real. Concretamente, de la variable “n”, que afecta a los dos paréntesis de la ecuación (6). Y en dichos paréntesis se incluye, entre otros, una resta de las coordenadas de los puntos con el centro de la curva. Pues dicha resta, según el grado n, tendrá un aporte numérico al resto de la ecuación (6) más o menos alto. Variando entre un amplio rango de valores. Pensar en que 0.5 elevado a un exponente 2 es 62 veces mayor que elevado a 8. Pues esto sucederá con las mediciones de las distancias, y por tanto de su media. A pesar de lo anterior, *superelipse* serviría como medición y no influye negativamente en la minimización, porque la idea es buscar que siempre sea la menor o cero. Podría plantearse emplear una comparación por superficies, pero de nuevo habría que tener cuidado con la influencia del factor “n” en su cálculo. Todo lo mencionado tampoco implica que se descarte el uso de la media a nivel comparación de ajustes. Lo único que no se puede emplear entre ajustes de distinto grado. A continuación, se desarrollarán tres funciones empleadas, o estudiadas, para lograr la mejor comparación de ajustes del mismo grado. No olvidar que siempre se representan los resultados y dicha representación es la mejor manera de comparar. Pero lo que se busca es la automatización e independencia del diseñador.

Lo normal para comparar optimizaciones, como se ha explicado, sería comparar la media de los resultados. Otra opción bastante empleada es comparar el error más grande o la mayor desviación. Además, para este trabajo, interesaría comparar las áreas de cada ajuste o su comparación respecto al contorno proporcionado por la imagen médica. Para lograr automatizar dichas comparaciones dentro de las funciones encargadas de la optimización se crean tres nuevas funciones.

La primera de ellas *comparacion_areas* (Apéndice 4: Función comparación de áreas) calcula el área interna al contorno de la imagen, donde se va a ajustar la superelipse, y estima el área de cada superelipse. Para calcular el área del contorno se llama a la función *area_contorno* que proporcionándole las coordenadas (x, y) de sus puntos devuelve el área que encierran. Previamente se probaron funciones basadas en el método de cálculo de áreas de polígonos y en el método del trapecio. Al principio, comparando con el área de las superelipses se creía que dichos métodos proporcionaban estimaciones aptas, pero no era así. Esto sucedía, ya que las versiones anteriores de *area_superellipse* basadas entre otras ideas en la regla de Simpson o en las ecuaciones paramétricas de x (7) e y (8), tenían dependencia del factor de n, y lo que esto conllevaba. Se buscó solucionar antes la versión de *area_superellipse* para lograr una comparativa razonable de los valores de *area_contorno*. Aunque esta última en verdad tiene menos importancia, puesto que sea cual sea el valor que devuelva será el mismo en todas las comparaciones. Por lo que es más importante disponer de una función *area_superellipse* que funcione correctamente y no tenga dependencia de “n”. Como solución final se llegó a la actual *area_superellipse* (Apéndice 6: Función para estimar el área de una superelipse) basada en el método de Monte Carlo [42]. De esta forma, especificando un número de *muestras* la función genera un rango y dependiendo de los valores que estén dentro o no se estimará un área. Como es de esperar, a mayor número de muestras mayor precisión. Probando la función para distintos “n” y tomando como base las representaciones de las correspondientes superelipses, se consideraron las estimaciones proporcionadas bastante acertadas entre ellas. Con esto, se ajustó la función *area_contorno* (Apéndice 5: Función para calcular el área total encerrada por del contorno de la imagen) empleando ConvexHull, que más tarde se detallará por ser un caso de estudio.

La segunda función, *mean_error* (Apéndice 7: Función para obtener la media de un ajuste) nos devuelve la media de los errores o distancias del ajuste definido en *variables* con respecto al contorno. Hay que recordar que esta y la siguiente función solo pueden emplearse para la comparación entre superelipses del mismo grado.

La tercera, *error_max* (Apéndice 8: Función para obtener el mayor error de un ajuste) es similar a la anterior, pero devolviendo el máximo error. En el siguiente apartado se detallará cuál de las tres funciones realiza una comparación más fiable y por tanto ayuda a proporcionar el mejor ajuste.

- Minimizacion simple (Apéndice 10: Función *minimizacion_simple*)

En sus primeras versiones recopilaba las adaptaciones para poder emplear un bucle externo y funcionar como *minimizacion_angulo* (Apéndice 9: Función *minimizacion_angulo*). Como se ha explicado, el bucle de estudio de grados es eliminado por la imposibilidad de comparar directamente distintos grados. Deben estudiarse los grados por separado. Y con el bucle de ángulos externo a la función, se obtenía una función de minimización muy simplificada. Esto suponía una gran ventaja para implementar en RANSAC (Apéndice 11: Modificación de RANSAC), ya que en cada una de sus iteraciones solo estudiaría un caso. Empleando RANSAC para cada valor de ángulo. Por lo que *minimizacion_simple* tendrá una gran importancia en la metodología de estudio de las distintas imágenes. Hay algunos valores dentro de *bounds* que han sido modificados o adaptados para encontrar resultados o mejorarlos. Concretamente los factores que delimitan los rangos de “a” y “b” al multiplicar los de *initial_guess*.

- Bounds y options

Por último, hay que destacar que se omitió la relación de los rangos con los puntos del contorno. Como la estimación inicial influye demasiado en el resultado, se aprovechó para acotar el rango entre 0.8 y 1.2 del valor inicial de cada una de las variables. Y a la hora de entrar en `scipy.optimize.minimize` se entraba con valores aleatorios dentro de dicho rango, a excepción de las coordenadas del centro, que apenas variaban, el ángulo, que tiene su propio estudio aparte, y el grado, que está fijado de inicio. Con esto se obliga al algoritmo a buscar entre otras opciones y evitar una convergencia local marcada ya de inicio. Pero, a la vez, el espacio de búsqueda está limitado a parámetros coherentes donde se sabe que habrá solución. Lo que tocaría después sería comparar dichas soluciones empleando las funciones descritas anteriormente.

En `options` de `scipy.optimize.minimize` se pasó de 1000 iteraciones a 500, ahorrando un tiempo significativo y obteniendo los mismos resultados.

3.3.3 Flujo de trabajo

En el subapartado anterior se ha hablado más acerca del desarrollo de las funciones y los problemas que se han ido resolviendo. Ahora, se enfocará en su empleo y como aprovechar dichas funciones para conseguir los mejores resultados.

La idea general sería encontrar un primer ajuste muy preciso, aunque no esté completamente dentro del contorno. Y con ese ajuste, emplear RANSAC (Apéndice 11: Modificación de RANSAC) para que encuentre una solución interior. Básicamente el procedimiento seguido en [1], pero con otras funciones.

Para obtener el primer ajuste se ha pasado por numerosas pruebas con distintas funciones, datos de ajuste, rangos de búsqueda, hiperparámetros de RANSAC (`max_iter` y `n`) y estimaciones iniciales. El desarrollo de las funciones se ha expresado en el anterior punto, explicando el empleo de una primera función `minimizacion`, seguido por: `minimizacion_global`, `minimizacion_angulo` (Apéndice 9: Función `minimizacion_angulo`) y hasta `minimizacion_simple` (Apéndice 10: Función `minimizacion_simple`), apoyada en bucles externos. Llegando a la conclusión de que la mejor opción es estudiar distintos ángulos para cada grado por separado. Función realizada por `minimizacion_angulo` sin necesidad de un bucle externo como sucedería con `minimizacion_simple`. Por lo que se buscará el primer ajuste con dicha función.

Para ello, basado en los *papers* mencionados se empleó como estimación inicial un ajuste elíptico anterior. Además de ser el antecedente de estudio y disponer de las herramientas para hallarlo. Con el que se obtenían resultados, visualmente, bastante buenos pero mejorables. A nivel de parámetros eran casi idénticos a los de la estimación inicial o elipse. Por ejemplo, uno de los semiejes podría aumentar su tamaño, pero al ser el de la elipse menor se estancaba. Como ya se ha mencionado, al ver la alta influencia de la estimación inicial en la convergencia local se planteó emplear el ajuste circular como opción y ampliar el rango de búsqueda de solución para cada parámetro. Sucedieron varios problemas, uno relacionado con el ángulo y el otro con los semiejes. El círculo a diferencia de la elipse no dispone de orientación definida por el ángulo, porque al girarlo sigue siendo la misma geometría y ajusta exactamente igual. Por lo que del ajuste circular previo se partía sin información del ángulo. Lo que obligaba a ampliar el rango del bucle y aun así no garantizaba buenas soluciones. Esto último viene relacionado con los parámetros “a” y “b”, o semiejes. Que pese a partir ambos de una única estimación o radio, ya que el círculo no dispone de dos ejes de distintas longitudes, con unos rangos de soluciones bastante amplios, se quedaban estancados de nuevo en el valor del radio. Para

intentar buscar en otros subespacios, dentro del espacio definido de soluciones que pueden tomar “a” y “b”, y llegar así a otras soluciones, se emplea la función *random* aplicada al rango completo. Pero al generar valores *random* para cada ángulo del bucle pues salían resultados variados pero malos. Por lo que se decidió volver al empleo de la elipse.

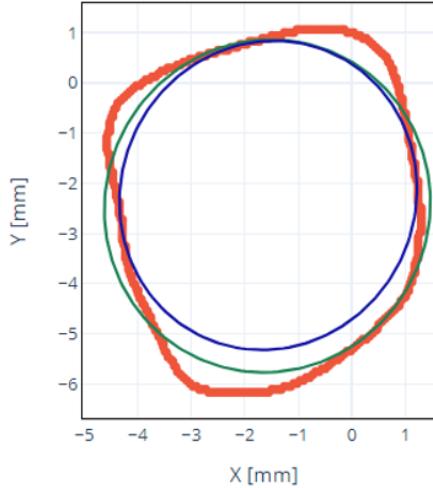
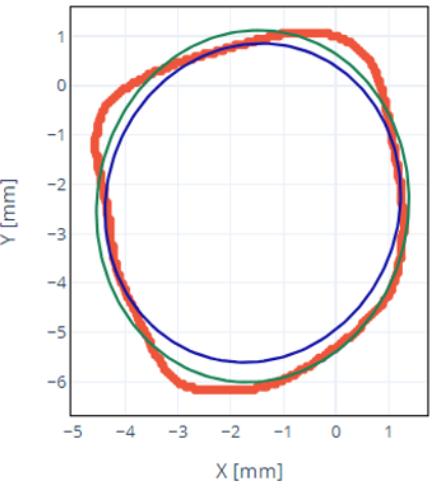
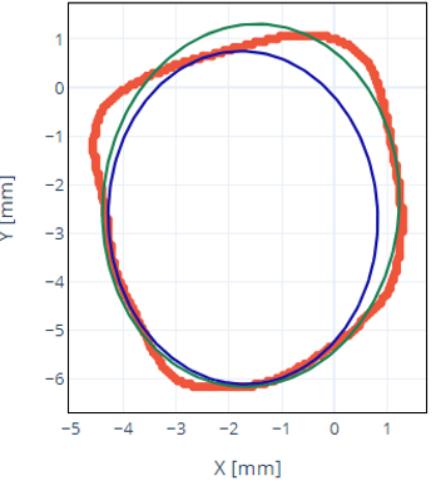
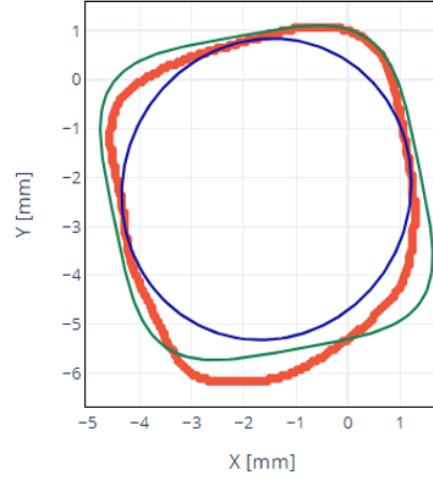
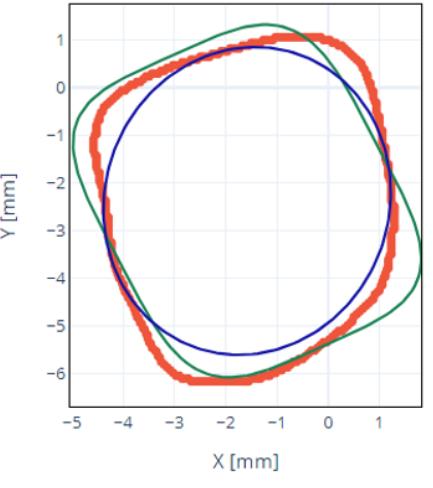
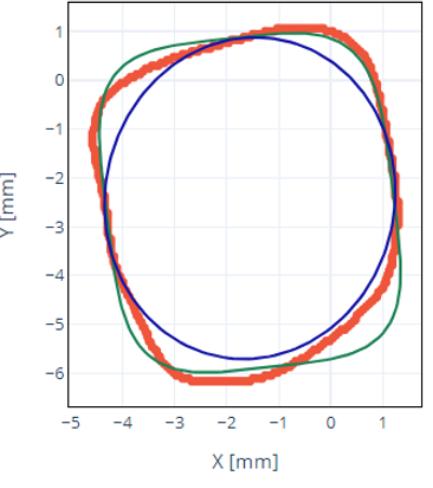
Cuando se habla del ajuste elíptico se hace referencia al resultado tras pasar el filtro de la función RANSAC. Se intentó ajustar con el previo a RANSAC, pero los resultados eran peores y más alejados de la geometría. Para aprovechar la iteración inicial, pero intentar modificarla para buscar más opciones se acotó el rango de búsqueda de los semiejes a 0.9 del valor inicial, 1.1 del valor inicial), además del bucle para los ángulos que ya se implementaba. Con esa pequeña diferencia en los límites se valoraban bastantes más opciones gracias a *random* pero dentro de unos límites razonables para eliminar resultados dispares y malos. Buscando una ínfima mejora de los parámetros de la elipse como una longitud de un eje algo mayor. Esto se aplicó tanto a *minimizacion_angulo* para hallar ese ajuste previo, como posteriormente a *minimizacion_simple*. Tras añadir estas modificaciones se comprobó que *minimizacion_angulo* era la mejor solución para explorar al máximo las opciones y encontrar el mejor ajuste previo posible. Para ello se apoyó en las tres funciones de comparación.

Se llevaron a cabo numerosos ensayos para poder comparar los métodos de comparación y seleccionar el más adecuado para este primer ajuste. Para comparar objetivamente dichas funciones, a parte de los resultados visuales, se empleará un parámetro de área igual a la diferencia entre el ajuste de la superelipse y el contorno de la imagen médica. El valor más cercano a cero, independientemente del signo, se traduce en el mejor ajuste. De esta manera, en cada columna de la tabla se reflejan los ajustes hallados empleando, dentro de *minimizacion_angulo*, las distintas funciones de comparación. Para estas primeras pruebas se emplea una sección del fémur.

En la tabla siguiente (Tabla 2) se verán reflejados dichos resultados. Como conclusión, se obtiene que el mejor método de comparación, para emplear dentro de *minimizacion_angulo*, es el basado en la media de todos los puntos. Con los valores de A (mm^2) delante, se observa, que es algo mejor que la comparación de áreas, pero claramente ambos mejores que el de máximo error.

- Sección
- Superelipse ajustada
- Elipse ajustada (*)

Figura 21. Leyenda de los resultados

Grado	Comparación de área (mm^2)	Comparación de media	Comparación de error max
n=2			
A (mm^2)	1.9512	0.465	0.679
n=4			
A (mm^2)	- 0.009	1.563	3.934

n=6			
A (mm^2)	0.851	- 0.877	- 1.476
n = 8			
A (mm^2)	- 0.233	0.041	3.343

n=10			
A (mm^2)	- 0.253	- 0.041	- 0.222

Tabla 2. Tabla comparativa de los ajustes con *minimizacion_angulo* y cada función comparación.

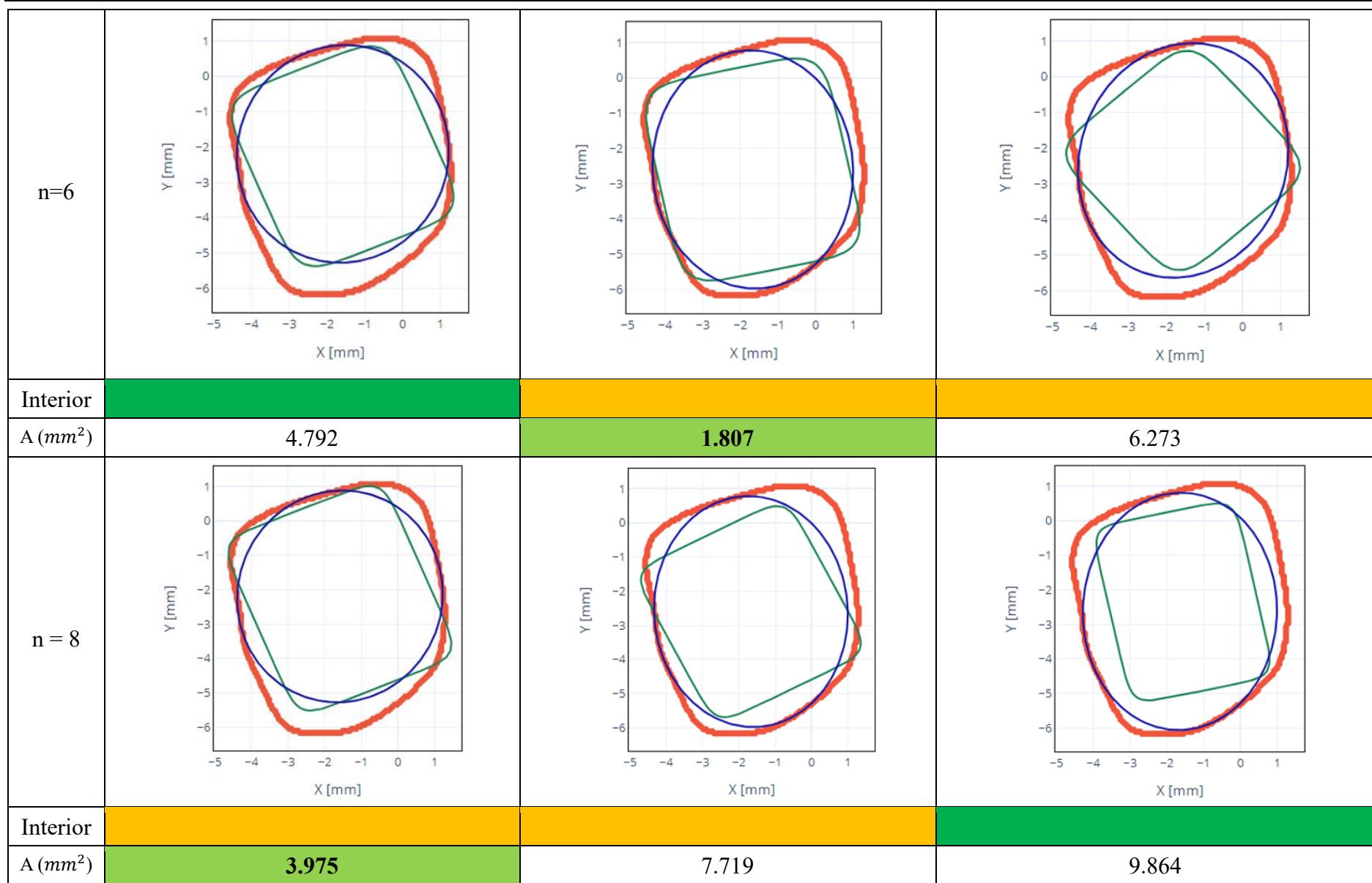
Una vez obtenido un ajuste previo bueno, se pasa a aplicar RANSAC (Apéndice 11: Modificación de RANSAC). Se incorporó de primeras la misma función *minimizacion_angulo*. Pese a distintas pruebas modificando *max_iter* y *n*, no se llegaba a bajar de las 3 horas. Es aquí donde se incorporará *minimizacion_simple*. Se logró reducir el tiempo más de lo previsto, pero muchas veces no llegaba a proporcionar una solución. Ya que un ajuste previo a RANSAC puede no llegar a ajustar interiormente, y más con el incremento de grado ya que se va asemejando a un rectángulo. Visualmente se aprecia que los primeros ajustes deben reducirse en tamaño y muchas veces rotar para lograr encajar dentro de la imagen. Para ello, además de modificar los rangos, de “a” y “b”, a (0.7 del ajuste previo, 0.9 del ajuste previo) se necesita estudiar distintos ángulos. Como solución, se crea una función auxiliar con un bucle que varie el ángulo de la estimación inicial que entraría en RANSAC, y por tanto en *minimizacion_simple*. Como para algunos ángulos puede que no se encuentre solución se debe añadir unas líneas de código que impidan el frenado del bucle. Como se observará a continuación, muchas veces los resultados son exageradamente internos. Esto es porque se ha necesitado modificar el intervalo anterior a (0.5 del ajuste previo, 0.8 del ajuste previo) para asegurarse la aparición de una solución. A veces, por no modificarlo aparece alguna solución ligeramente externa. Pero ya se analizará en el apartado de resultados y en las conclusiones lo que se haría.

Remarcando que hubo que hacer numerosas pruebas para encontrar los valores de *max_iter* y *n* que lograsen resultados sin problemas, y buenos, y en un tiempo coherente. Llegándose a 1000 y 100, durando cada optimización aproximadamente una hora.

En vez de compartir la función donde se incorporará el bucle y *minimizacion_simple*, en el (Apéndice 12: Función ajuste completa) se incorporará lo mencionado más la llamada a *minimizacion_angulo* que hallaba el primer ajuste.

Al igual que con el funcionamiento de *minimizacion_angulo* para sacar el ajuste inicial, en esta parte también se deben comparar las funciones encargadas de reflejar el mejor ajuste. Se repite la manera de comparar, pero en este caso el área reflejada será la diferencia entre el ajuste elíptico a mejorar y la superelipse solución. Por lo tanto, que dicha área sea negativa significa que el ajuste de la superelipse es mejor. Además, se añade una línea reflejando si el ajuste está completamente dentro. Observando la tabla (Tabla 3) se llega a la conclusión de que el mejor ajuste para esta fase es la comparativa de áreas. El ajuste de máximo error puede ser útil para geometrías que ajusten razonablemente bien, pero sean difícil de encajar dentro en zonas o picos concretos. Y el ajuste que compara las medias es peligroso. Porque los valores son próximos al de áreas, pero visualmente se ve que muchos de sus ajustes están en el límite de ajustar internamente.

Grado	Comparación de área (mm^2)	Comparación de media	Comparación de error max
n=2			
Interior			
A (mm^2)	- 3.257	1.536	- 0.541
n=4			
Interior			
A (mm^2)	1.438	2.339	2.297



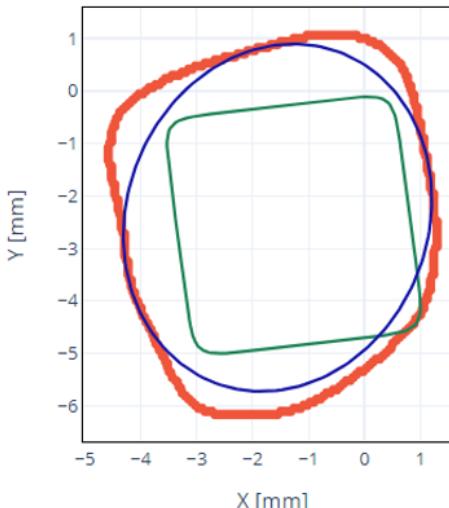
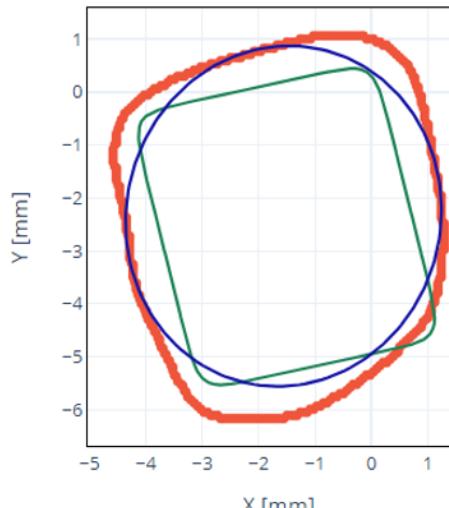
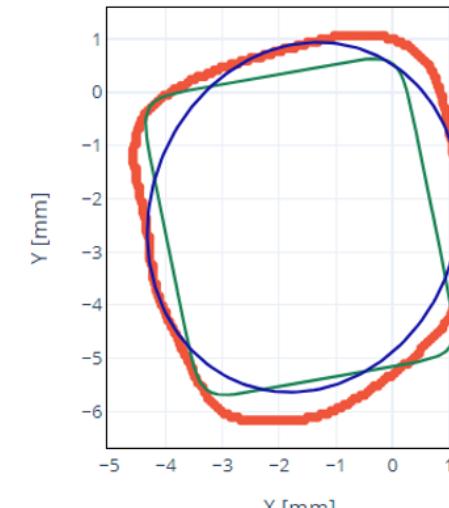
n=10			
Interior			
A (mm^2)	8.615	6.764	3.431

Tabla 3. Tabla comparativa de los ajustes con RANSAC (*minimizacion_simple*) y cada función comparación.

En una reunión con el tutor, donde se compartieron los resultados adquiridos hasta el momento, se expresó la sorpresa y descontento con algunos ajustes que no se encontraban completamente dentro del contorno. Para solucionar esto surgió la idea de probar a ajustar respecto a los *inliers* del ajuste elíptico, en vez de los propios puntos de la imagen médica.

Los *inliers* son puntos de las elipses ajustadas a la muestra que están dentro del contorno. La función RANSAC empleada en [1] es capaz de proporcionar dichos datos como hacía con los parámetros del ajuste elíptico. Por lo que, si se busca un ajuste a ellos, será interior. Además, es fácil encontrar una solución porque muchos vienen de otras soluciones y reduce el número de puntos a estudiar. Y si el ajuste no fuese tan preciso y superase sus límites, no tendría por qué superar los de la imagen, puede haber margen. Este planteamiento podría reducir tiempos y mejorar resultados. Continuando con la imagen médica empleada hasta ahora se prueba la metodología completa explicada anteriormente para los distintos grados.

En la tabla (Tabla 4) de la siguiente página se ve que los ajustes no difieren mucho de los otros. A nivel de duración, el tiempo que se ahorra no es el suficiente como para cambiar la metodología. Pero hay que destacar un resultado. Para el caso $n = 2$, en el primer ajuste con *minimizacion_angulo* se da una solución completamente interior sin emplear RANSAC, en dos segundos, y con mayor área que la elipse ajustada. Con RANSAC el tiempo es muy elevado y no mejora en muchos casos el ajuste elíptico. En el Capítulo 5: Resultados se comentará y compartirán los resultados de la prueba de este caso específico, para distintas imágenes médicas.

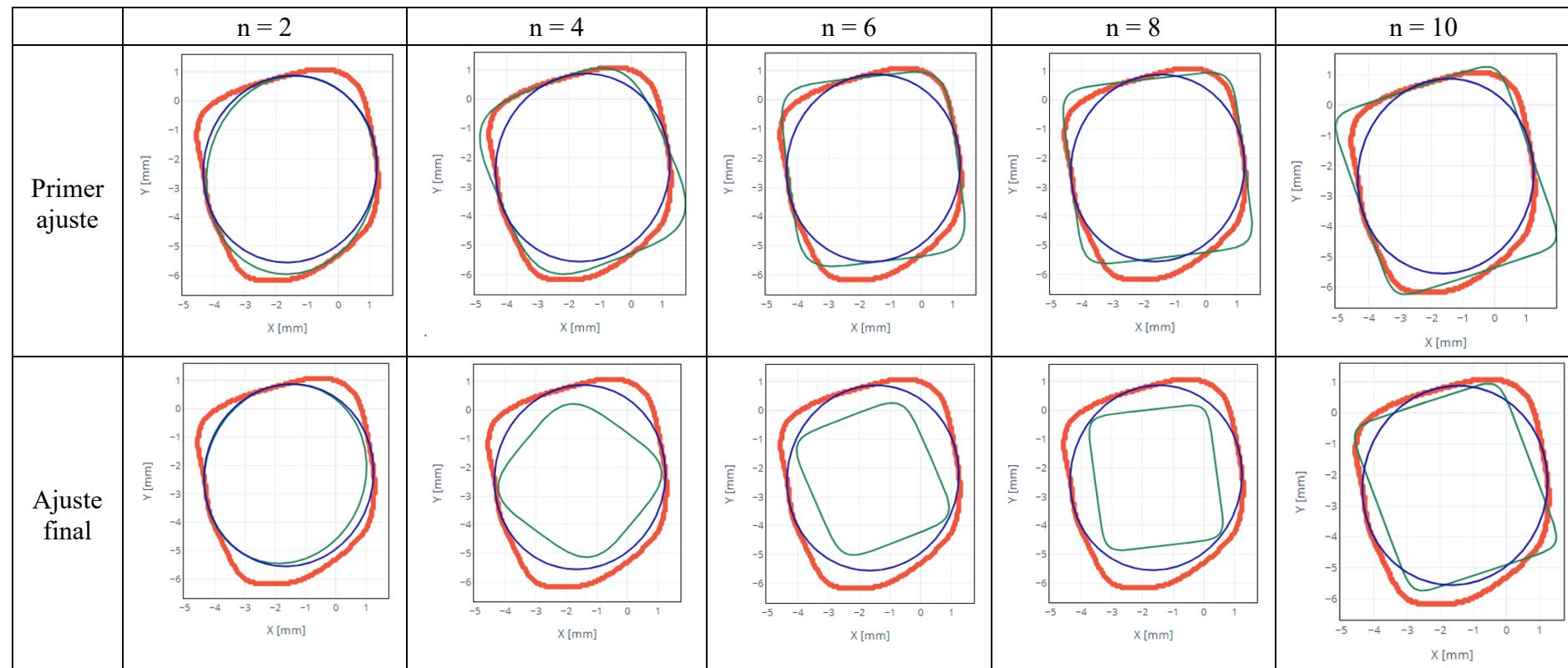


Tabla 4. Resultados de ajustes a los *inliers* del ajuste elíptico: primer ajuste (previo a RANSAC) y ajuste final (con RANSAC).

Capítulo 4: Automatización en el diseño de implantes personalizados basado en polígonos

4.1 Planteamiento

Con la idea de la superelipse ya se estaba abarcando una gran cantidad de curvas. Al variar sus parámetros se podría lograr múltiples opciones con propiedades y geometrías diversas. Pero siempre se quiere abarcar más. No se encontró ninguna otra curva cerrada que pudiese adaptarse y generalizarse de manera sencilla para lograr su optimización.

De ahí se traslada el pensamiento a valorar los polígonos. Por polígono se entiende aquella geometría plana formada por segmentos y vértices. Pensando que se dispone únicamente del contorno a ajustar, definido por las coordenadas (x, y) de los puntos que lo componen, surgió el siguiente planteamiento.

Se parte de las coordenadas de todos los puntos. Pues se comienza cogiendo una muestra aleatoria de “ n ” puntos. Se unen dichos puntos con el más próximo. Se trazaría la perpendicular a dicha unión por la mitad hasta cortar o llegar a un punto del contorno. Se volvería a buscar la unión de cada punto con el que este situado más próximo, y se repetiría lo anterior. Estos pasos se repetirían hasta la definición de convergencia que se especifique. Por ejemplo, hasta que se tuviesen “ n ” puntos. O hasta que el área de dicho polígono fuese superior al mejor ajuste de otra geometría. Esta idea fue inspirada en una modificación del método de Newton-Raphson. Todo reflejado en el siguiente boceto (Figura 22).

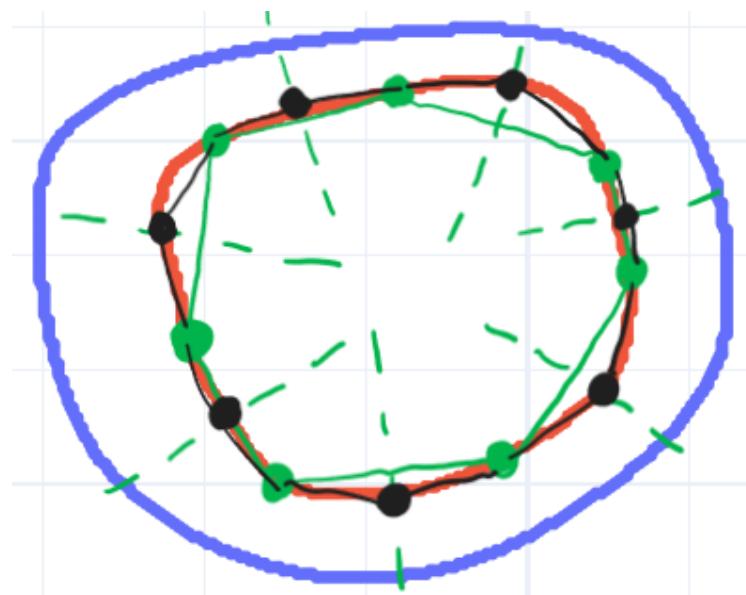


Figura 22. Boceto con el desglose de la idea en un contorno real.

Como el desarrollo de dicho código desde cero sería complejo y laborioso se buscó alguna función para reciclar o usar, y de esta manera, ahorrarse pasos y problemas. En dicha búsqueda se cruzó la idea del *Convex Hull*, y ejemplos de funciones, tanto de usuarios como de las propias librerías. La teoría acerca de este concepto está desarrollada en el Capítulo 2: Marco Teórico.

4.2 Desarrollo

Se optó por implementar la función *ConvexHull*. Esta se puede importar de la librería *SciPy*, concretamente del paquete *Spatial*, de Python® [13].

El fin de esta idea es hacer que el casco convexo coincidiese con el contorno o que se aproximase. Para, ello como primera función, se llamaba a *ConvexHull* para sacar el casco convexo de los datos proporcionados. Una vez hallado dicho casco habrá distintas metodologías para ordenar los puntos que lo componen: elegir el punto más a la izquierda [11], hallar un centroide, etc. En esta primera función se opta por calcular el centro. Después con ayuda de la función *np.arctan2* de *Numpy* [43], se halla el ángulo de cada punto con respecto al centro anterior. Se reordenan los puntos y se devuelve el casco convexo ordenado en sentido horario.

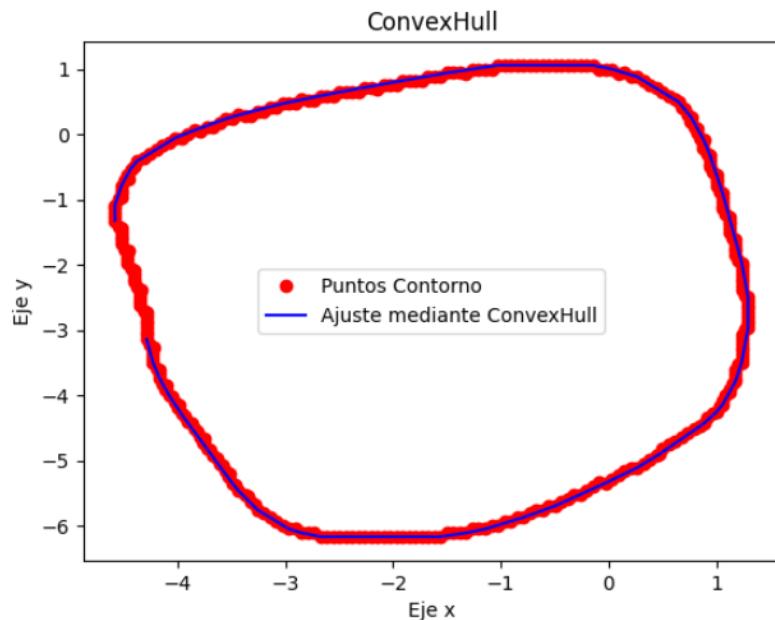


Figura 23. Primera solución con ConvexHull

El primer resultado es mejor de lo esperado (Figura 23), con el único defecto de no devolver un casco completamente cerrado. Y por ser detallistas, que el ajuste sea tan bueno podría ser perjudicial para la instalación de la prótesis, por lo que convendría buscar una manera de especificar una especie de separación. Se busca, por tanto, una función que permita editar dicho parámetro.

Para lograr la separación, se sigue un proceso similar. Primero se halla el casco, pasándose posteriormente, de nuevo, al centro. Se ordenan los puntos en sentido antihorario. Aquí empieza lo novedoso, se calcula el vector de unión entre el centro y cada uno de los puntos. Con dichos vectores se dispondrá de la dirección que los une. Dependiendo del valor y signo que tome *valor*, la separación se dilatará o contraerá. Es importante que se normalicen dichas distancias para que dicho valor afecte por igual a todos los vectores, por eso el siguiente paso consiste en normalizar los vectores, dividiendo por su longitud. El resultado de esta primera función

Se buscan otras opciones, pero lo único que se acaba encontrando es otra manera de ordenar los puntos basado en el algoritmo de ordenación de Graham. Comparando sus áreas con la del contorno, se ve su gran similitud (Figura 24). En estas funciones seguía sin unirse o cerrarse completamente la curva. Para ello hubo que forzar en una línea de código la unión entre el primer y el último punto del casco.

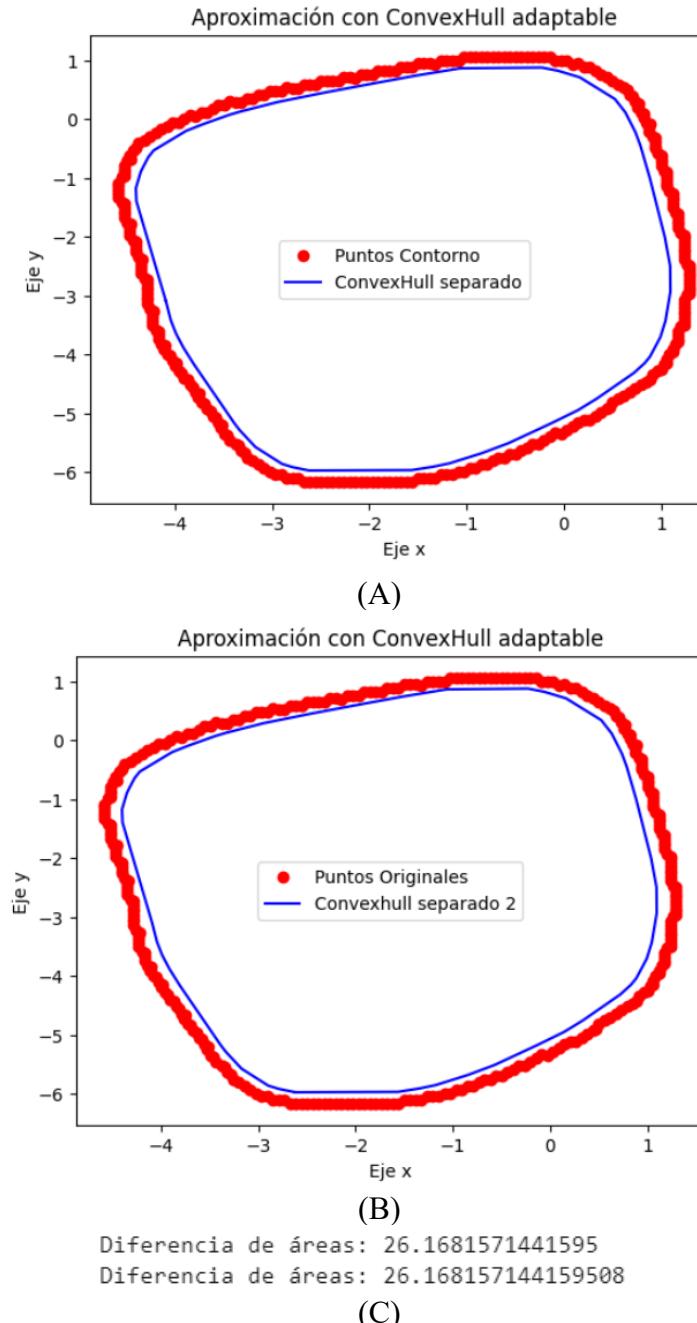


Figura 24. (A) Resultados función *convexhull_adaptable1*, (B) resultados *convexhull_adaptable2* y (C) comparación por áreas de ambas funciones.

Se comparten en Apéndices ambas funciones por si fuesen necesarias para líneas futuras. Ambas tienen como entrada los puntos del contorno *data* y el *valor* o parámetro de separación.

Capítulo 5: Resultados

En este capítulo se adjuntarán los resultados de las funciones y el flujo de trabajo comentado a lo largo del trabajo. Para poder comentar y comparar distintas circunstancias se van a estudiar tres imágenes médicas, correspondientes a una tibia, arteria y fémur; que poseen geometrías complejas. Estas, en principio, no son similares a ninguna geometría definida o conocida, dificultando su ajuste.

5.1 Optimización de parámetros de la superelipse

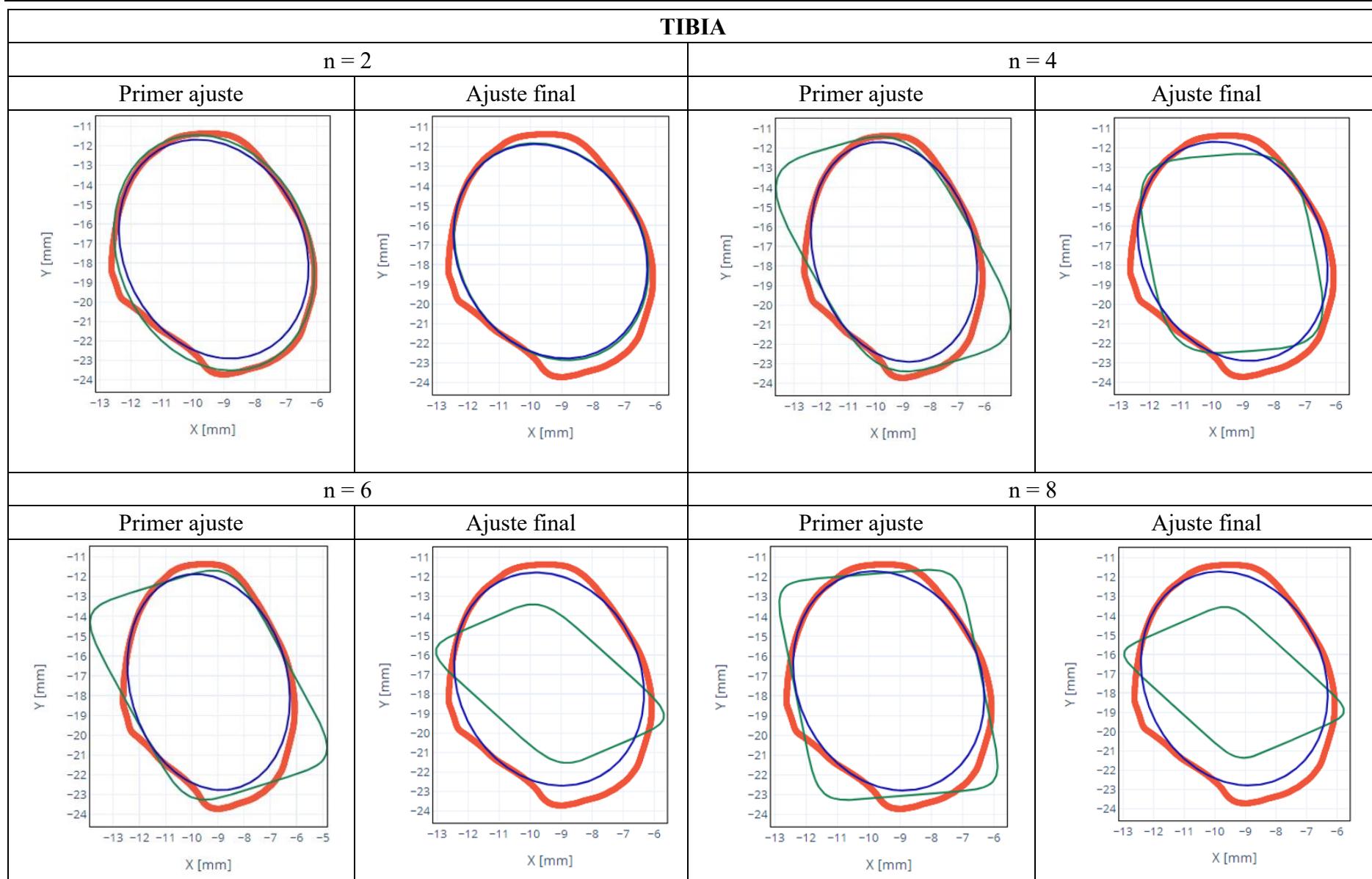
La primera tabla (Tabla 5) reúne los ajustes de cada grado siguiendo la metodología de ajuste previo con *minimizacion_angulo* (Apéndice 9: Función *minimizacion_angulo*) y posterior empleo de la modificación de RANSAC (Apéndice 11: Modificación de RANSAC).

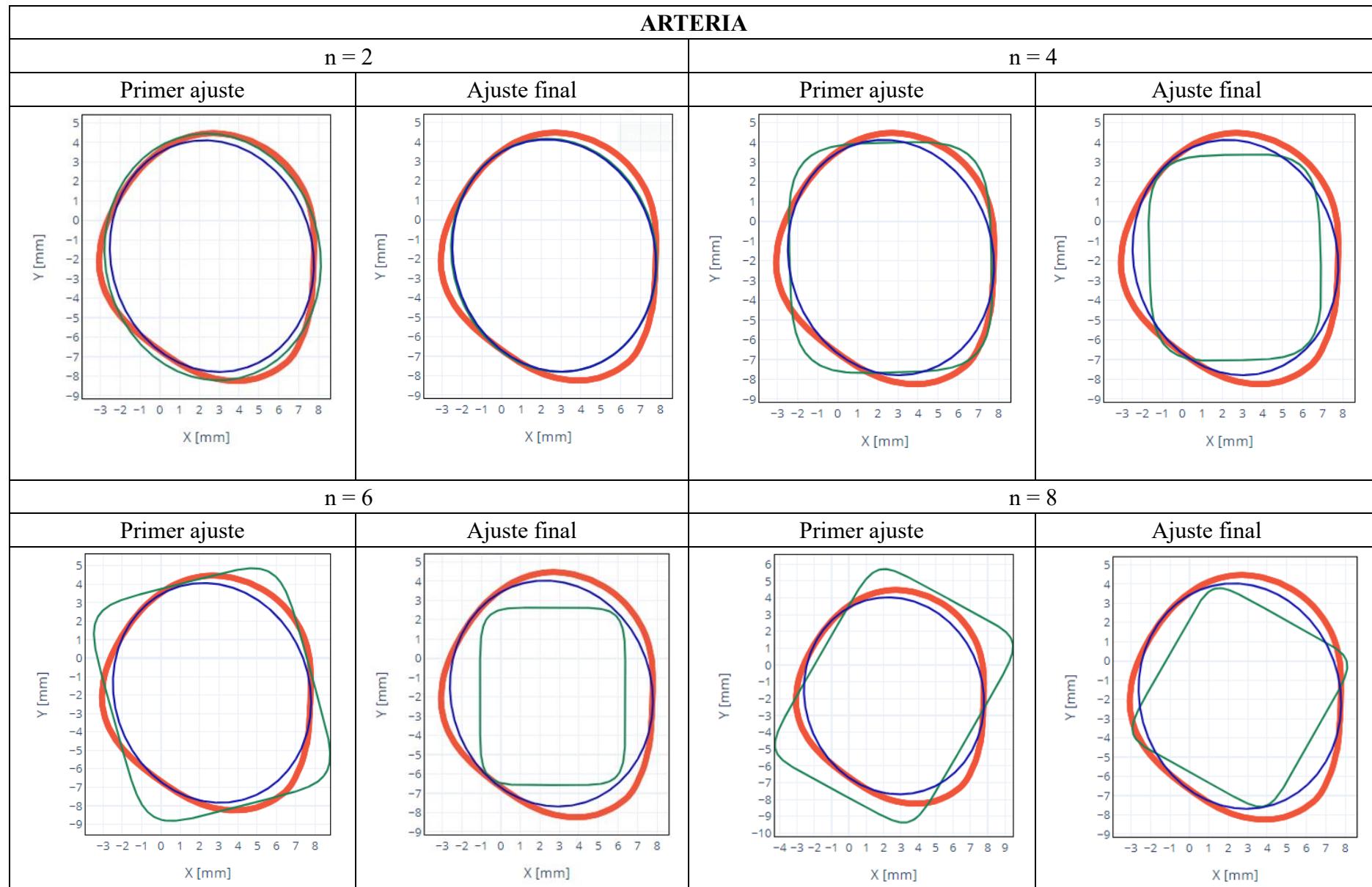
Como se puede observar en dichas imágenes, ya algunos de los primeros ajustes tienen bastantes zonas de su geometría lejana al contorno correspondiente. Por lo que incrementa la dificultad de encontrar una solución interna a la curva. Muchas resultan en un ajuste “fallido” ya sea por tener zonas externas o por estar muy alejadas de mejorar los resultados de la elipse.

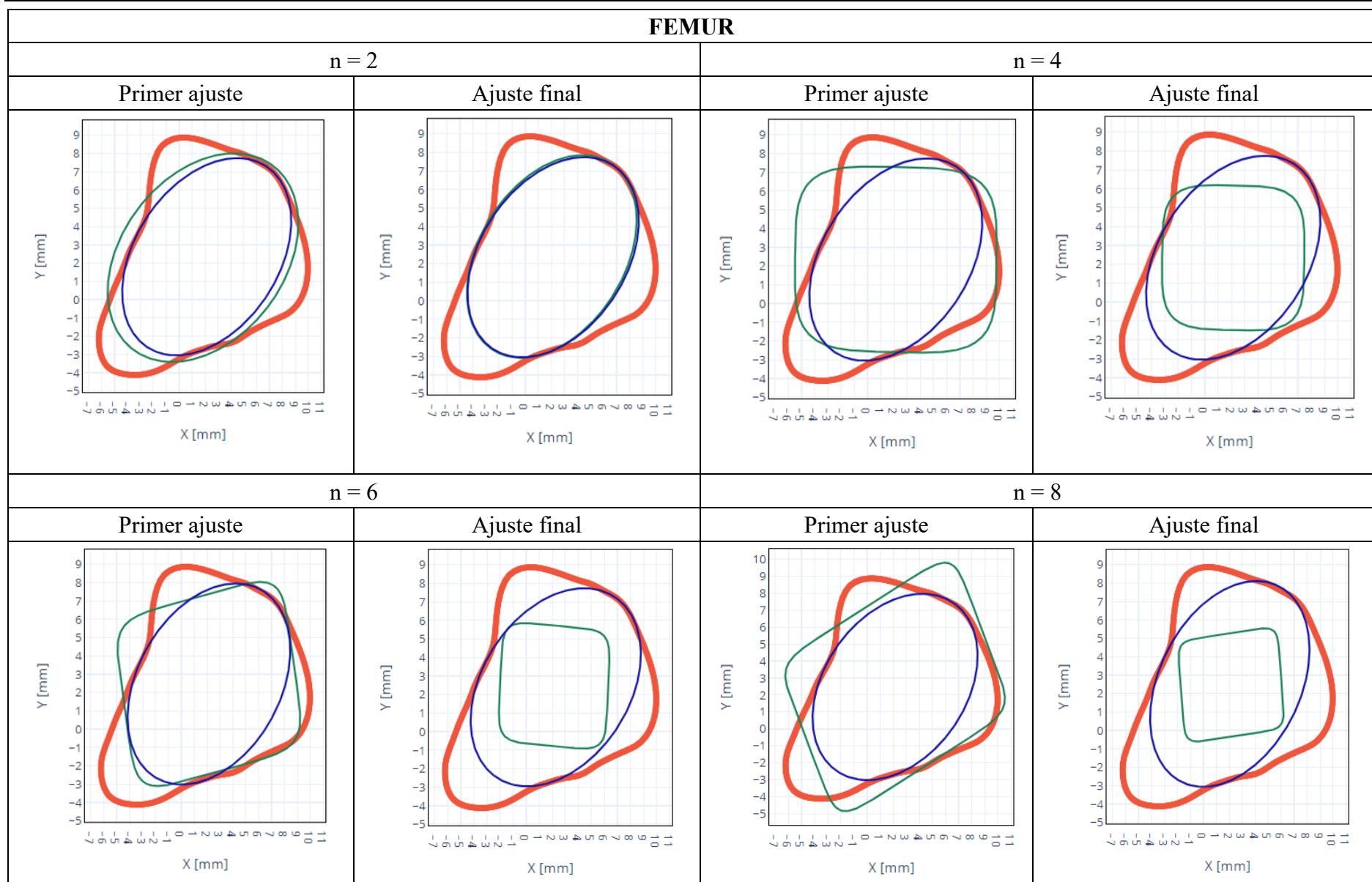
Además, hay que destacar del uso de las funciones, que algunas han visto modificados los rangos de las variables o incrementando el número de ángulos de estudio. Por lo que las funciones deben adaptarse a las geometrías o grados que se buscan ajustar. Por ejemplo, para lograr los ajustes de $n = 10$ de las tres imágenes, se ha tenido que delimitar los rangos de “a” y “b” a (0.5 del valor del primer ajuste, 0.9 del valor del primer ajuste) respectivamente, muy lejano al código inicial y empleado en el resto de los grados de manera general. Esto es debido a su geometría, al ser más próxima a la del rectángulo es más difícil que toda su área sea interior. Por lo que habrá que reducir su tamaño en mayor porcentaje, comparado con el ajuste previo.

Otro punto negativo es el tiempo, en algunos casos aumentó el tiempo estimado para llegar a la solución final. Se esperaba que fuese para todas las imágenes de una hora, pero para los grados más altos de estas imágenes se aproximaban más a las dos horas. Esto influye bastante en la futura implementación del trabajo. Puesto que únicamente las superelipses de grado 2 incrementan el área de ajuste, pero lo hacen empleando bastante más tiempo. El resto de los grados puede que ofrezcan mejores resultados estructurales, se podría estudiar en otros trabajos, pero sigue siendo un gasto de tiempo de ajuste innecesario.

Por el contrario, en la segunda tabla del apartado (Tabla 6) se dan resultados positivos. Empleando únicamente *minimizacion_angulo*, y en cuestión de segundos, se obtienen superelipses de grado 2, o sea elipses, mejores que las anteriores visualmente, por su colocación, e incrementando el área. Todo ello gracias al ajuste a los *inliers* del ajuste elíptico previo.







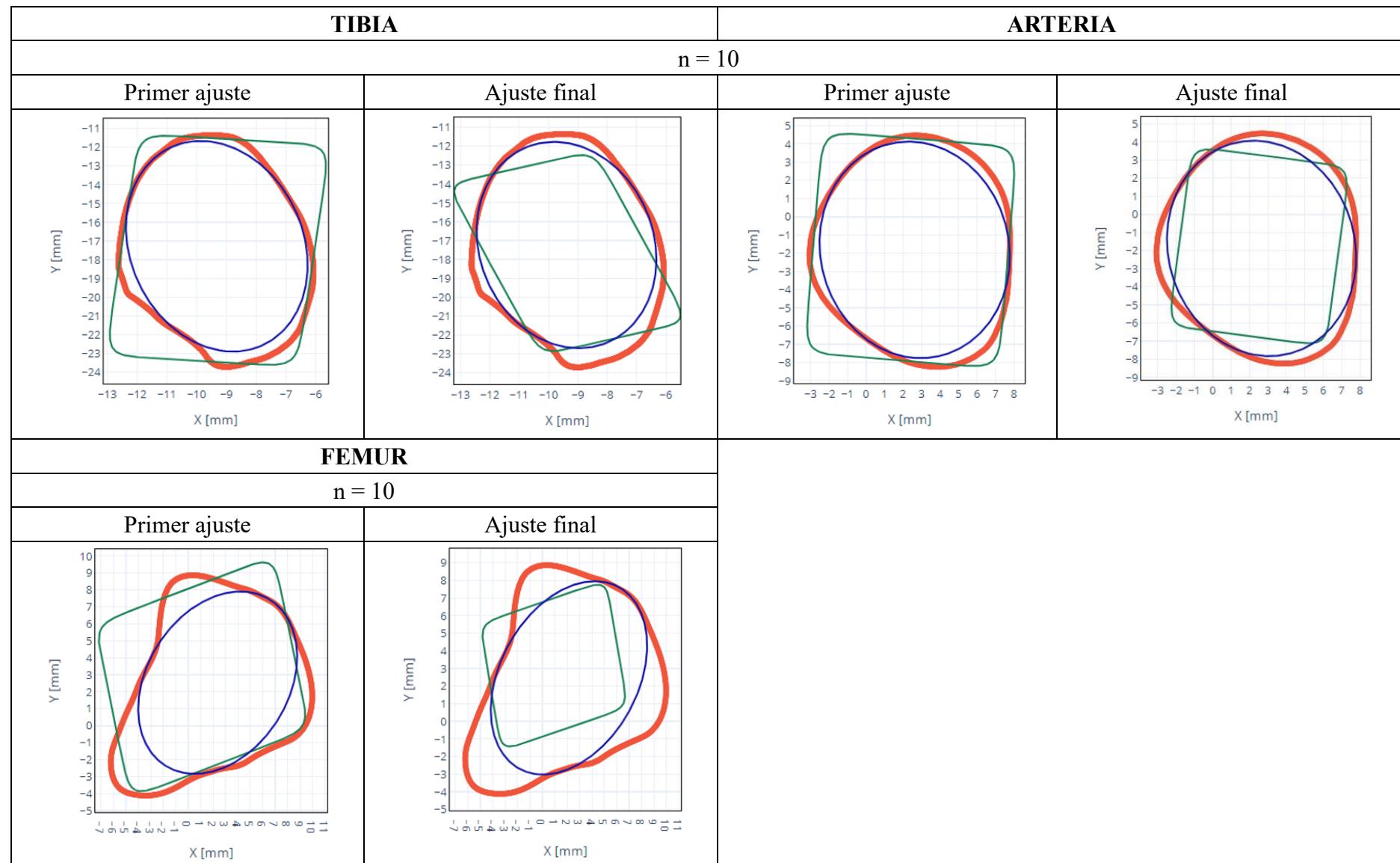


Tabla 5. Resultados de ajustes de superelipses a tibia, arteria y fémur.

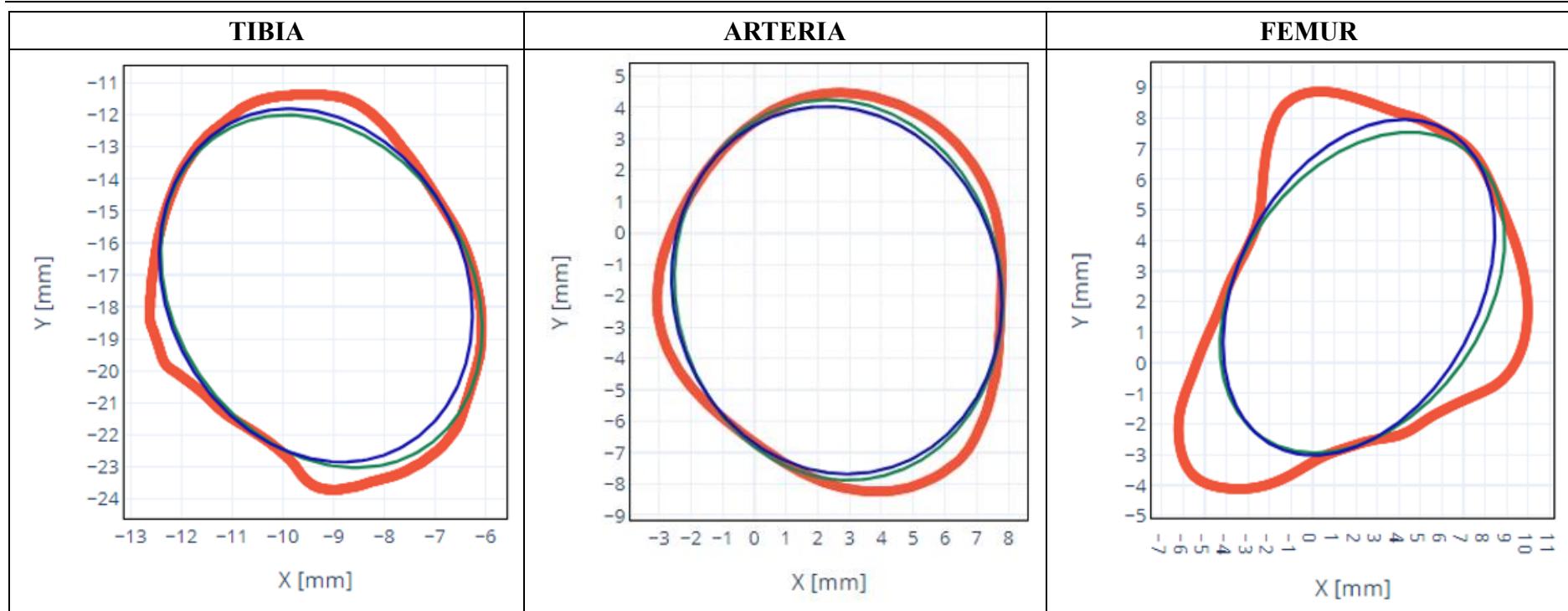


Tabla 6. Resultados ajuste a *inliers* de tibia, arteria y fémur.

5.2 Optimización de parámetros de Convex Hull

Como para el caso de la superellipse, se pasa a probar las funciones explicadas con anterioridad a tres casos más complejos. Pese a que se demostró que ambas funciones eran muy similares, y se obtenían los mismos resultados, se volvió a comprobar por si unas imágenes distintas afectasen más de lo esperado. En la siguiente tabla se puede confirmar que ambas trabajan y resuelven el ajuste de manera idéntica.

Por otro lado, si que hay que modificar el parámetro de la distancia en todos los casos, expresados en la misma tabla. Alguno, como el caso de la arteria, no necesitaría del uso de dicho parámetro. En cambio, para el caso del fémur es necesario alejar la curva, más que el resto, debido a que una zona del casco convexo tiene menor curvatura que el contorno de la imagen y saldría un poco por una de las esquinas de la geometría. Pero no es más que un detalle, no se debería catalogar como problema para su aplicación o uso.

Los resultados de la Tabla 7 siguen la leyenda de la Figura 25.

- Sección
- Elipse ajustada
- ConvexHull

Figura 25. Leyenda de los resultados con *Convex Hull*.

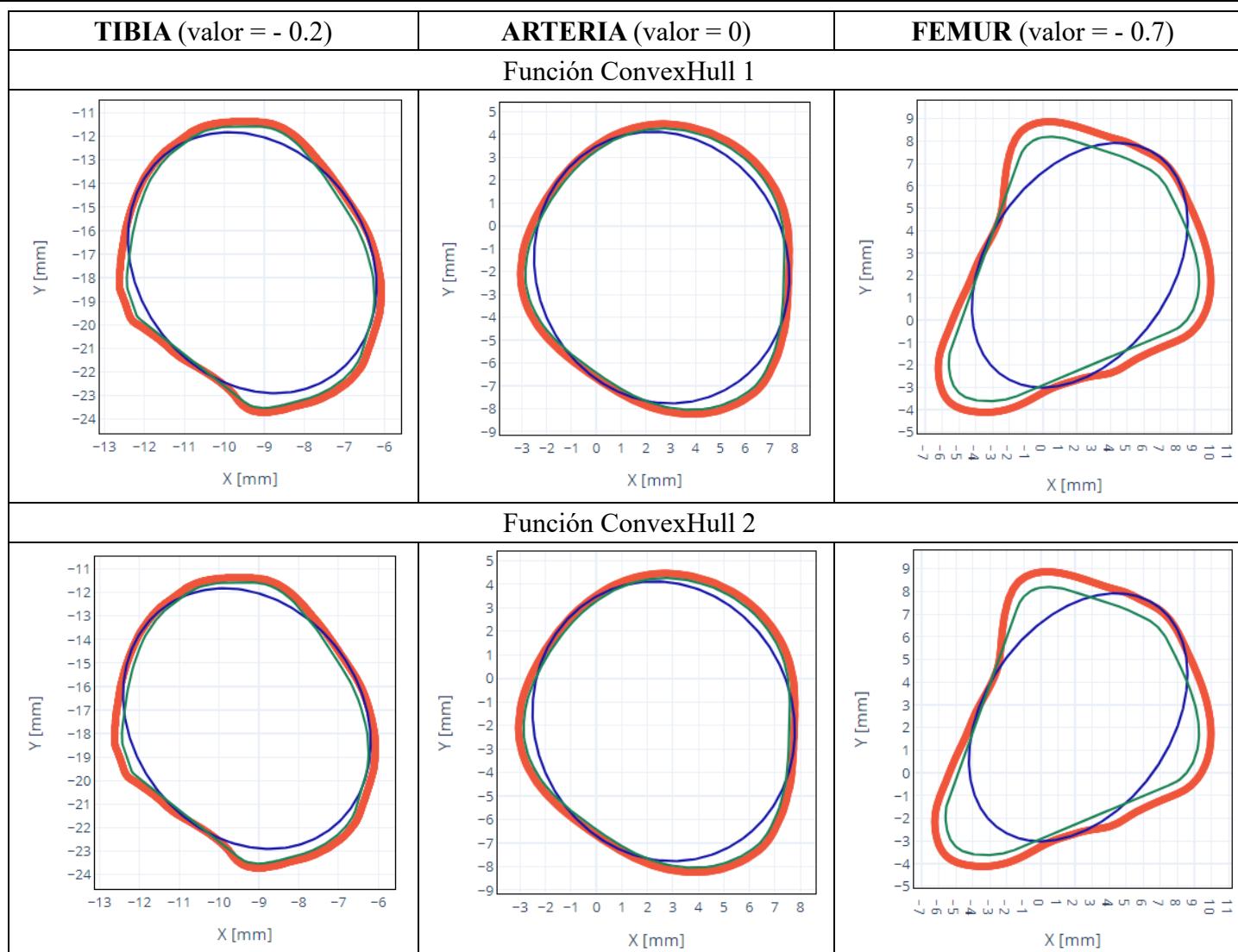


Tabla 7. Resultado de las dos versiones de *ConvexHull* aplicadas a tibia, arteria y fémur

Conclusiones

Este trabajo de investigación se centra en la mejora y ampliación del estudio de la automatización del diseño de prótesis apoyado en la inteligencia artificial y métodos numéricos. Para lograrlo se desarrollan distintos enfoques. Sin embargo, el caso del óvalo fue descartado antes de comenzarlo debido a la complejidad escondida de dicha geometría.

El mayor enfoque del trabajo se puso en la superelipse. Como se partía de un estudio de aplicación de la elipse, se pensaba ampliar dicho estudio a distintos grados, ya que en el fondo una elipse es un caso concreto de la superelipse. Lo que se desconocía es la ventaja que ofrece la elipse para encontrar una solución de forma cerrada a través del uso de la distancia algebraica en su ajuste. En resumen, el ajuste por mínimos cuadrados y distancia algebraica, expresada esta con el polinomio implícito correspondiente, solo asegura solución para el caso de la elipse, y no cuando $n \neq 2$.

Ese pequeño detalle, desconocido previamente, no supuso el abandono a dicha geometría. De esta manera, se dio comienzo a un estudio detallado de la optimización de los parámetros de una superelipse para minimizar la distancia algebraica a una curva, en este caso proveniente de una imagen tomográfica. Tras numerosas funciones y pruebas, se llegó a la conclusión de que no era viable la aplicación del modelo desarrollado. Dicho modelo necesitaba de mayor cantidad de tiempo con respecto al ajuste elíptico, para mejorar únicamente las soluciones de grado 2. Para el resto de los grados habría que estudiar si se da un mejor comportamiento mecánico, pese a un ajuste peor y necesitar de mayor tiempo. En cambio, hay que mencionar que puede ser de utilidad el ajuste de superelipses de caso $n = 2$ a los *inliers* de un ajuste elíptico previo. Lo que viene a ser otro ajuste elíptico que gracias a la función *minimizacion_angulo* consigue mejorar, en segundos, el ajuste elíptico previo.

Por otro lado, se planteó otro estudio simultaneo basado en un polígono formado por el casco convexo o *convex hull* de la curva a ajustar. De esta parte del trabajo se obtuvieron mejores resultados. La idea es fácil de implementar, y a través de la entrada de un parámetro auxiliar se puede ajustar la geometría a la calidad buscada en los resultados.

Con la puesta en funcionamiento de estos ajustes se permitirá generalizar el diseño de implantes médicos personalizados. Ampliando su aplicación a: prótesis de rodilla, húmero y fémur, y stents coronarios o válvulas aórticas.

Líneas Futuras

Continuando el estudio iniciado en este trabajo, será necesario implementar en un software de diseño asistido por ordenador, o CAD, el código estrictamente necesario. Se lograría una mayor automatización del proceso de diseño y obtener un fichero STL con el que obtener la prótesis final. Para ello, previamente, habría que adaptar o implementar el código a la aplicación web propuesta en el trabajo “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” de William Gabriel Solórzano Requejo [1]. Añadiendo en dicha web las nuevas geometrías de estudio, para que sean una opción más a valorar por el algoritmo a la hora de trabajar con cualquier imagen médica.

Gracias a dicha implementación se podría observar la calidad de los ajustes, comparando a través de la web todas las geometrías de su base de datos, de manera real. Se pasaría a imprimir una versión a escala de la prótesis y el órgano al que debería ajustarse para su implementación. Con este caso real se comprobaría la utilidad y resultado del ajuste.

Con el apoyo del software de diseño o CAD, para el caso del ajuste por *Convex Hull*, se podría implementar una manera de suavizar sus bordes y facilitar el encaje de la futura prótesis con la estructura del paciente. Y con la creación de un modelo final, y la ayuda del software correspondiente, se podría comparar el comportamiento estructural que aportaría cada una de las geometrías. Aunque un ajuste en principio no sea el más exacto a nivel geométrico, puede resultar siendo una mejor solución mecánica. Los resultados mecánicos son de igual o mayor importancia en la futura prótesis, especialmente para el correcto desarrollo de las funciones del organismo.

Para evitar el descarte o mejorar el ajuste de superelipses, se podrían plantear nuevas metodologías de estudio: empleo de nuevos algoritmos (recocido simulado, optimización por enjambre de partículas-PSO, etc), separación de secciones del contorno (cambiando el *cluster DBSCAN* por *MiniBatchKMeans*) y ajustar por zonas, ajustar la superelipse con dos mitades con distintos parámetros; esta última idea puede tener varios enfoques. En la actualidad ya se está trabajando con una primera idea donde un eje es fijo, además del grado, y el otro se divide en dos semiejes de distintas longitudes. Pero se podría desarrollar más y además modificar el grado de cada una de las mitades que son separadas por el eje fijo.

Bibliografía

- [1] W. G. S. Requejo, «Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares».
- [2] W. G. Solórzano Requejo, «Innovación en el diseño personalizado de vástagos femorales cortos», *Universidad de Piura*, oct. 2021, Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: <https://pirhua.udep.edu.pe/handle/11042/5159>
- [3] W. Solórzano-Requejo, C. Ojeda, y A. Díaz Lantada, «Innovative Design Methodology for Patient-Specific Short Femoral Stems», *Materials (Basel)*, vol. 15, n.º 2, p. 442, ene. 2022, doi: 10.3390/ma15020442.
- [4] M. Salmi, «Additive Manufacturing Processes in Medical Applications», *Materials*, vol. 14, n.º 1, p. 191, ene. 2021, doi: 10.3390/ma14010191.
- [5] A. Ait Moussa, J. Fischer, R. Yadav, y M. Khandaker, «Minimizing Stress Shielding and Cement Damage in Cemented Femoral Component of a Hip Prosthesis through Computational Design Optimization», *Advances in Orthopedics*, vol. 2017, pp. 1-12, 2017, doi: 10.1155/2017/8437956.
- [6] B. Y. Ni, I. Elishakoff, C. Jiang, C. M. Fu, y X. Han, «Generalization of the super ellipsoid concept and its application in mechanics», *Applied Mathematical Modelling*, vol. 40, n.º 21-22, pp. 9427-9444, nov. 2016, doi: 10.1016/j.apm.2016.06.011.
- [7] X. Zhang y P. L. Rosin, «Superellipse fitting to partial data», *Pattern Recognition*, vol. 36, n.º 3, pp. 743-752, mar. 2003, doi: 10.1016/S0031-3203(02)00088-2.
- [8] «KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS».
- [9] P. L. Rosin, «Curve segmentation and representation by superellipses», *IEE Proc., Vis. Image Process.*, vol. 142, n.º 5, p. 280, 1995, doi: 10.1049/ip-vis:19952140.
- [10] E. Dura, J. Bell, y D. Lane, «Superellipse Fitting for the Recovery and Classification of Mine-Like Shapes in Sidescan Sonar Images», *IEEE J. Oceanic Eng.*, vol. 33, n.º 4, pp. 434-444, oct. 2008, doi: 10.1109/JOE.2008.2002962.
- [11] «Convex Hull using Jarvis' Algorithm or Wrapping», *GeeksforGeeks*, 13 de julio de 2013. <https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/> (accedido 1 de septiembre de 2023).
- [12] «Spatial algorithms and data structures (scipy.spatial) — SciPy v1.11.2 Manual». <https://docs.scipy.org/doc/scipy/reference/spatial.html> (accedido 1 de septiembre de 2023).
- [13] «scipy.spatial.ConvexHull — SciPy v1.11.2 Manual». <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html> (accedido 1 de septiembre de 2023).
- [14] R.- ASALE y RAE, «óvalo | Diccionario de la lengua española», «Diccionario de la lengua española» - Edición del Tricentenario. <https://dle.rae.es/óvalo> (accedido 1 de septiembre de 2023).
- [15] «Óvalo», *Wikipedia, la enciclopedia libre*. 1 de mayo de 2023. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: <https://es.wikipedia.org/w/index.php?title=%C3%93valo&oldid=150897011>
- [16] «Superellipse _ AcademiaLab». <https://academia-lab.com/enciclopedia/superellipse/> (accedido 1 de septiembre de 2023).

- [17] «Superellipse -- from Wolfram MathWorld». <https://mathworld.wolfram.com/Superellipse.html> (accedido 1 de septiembre de 2023).
- [18] «Superellipse», *Wikipedia, la enciclopedia libre*. 16 de agosto de 2023. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: <https://es.wikipedia.org/w/index.php?title=Superellipse&oldid=153110277>
- [19] «Convex hull», *Wikipedia*. 13 de agosto de 2023. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Convex_hull&oldid=1170129966
- [20] H. Sikchi, «Convex Hulls: Explained», *Medium*, 22 de abril de 2017. <https://medium.com/@harshitsikchi/convex-hulls-explained-baab662c4e94> (accedido 1 de septiembre de 2023).
- [21] M. De Berg, O. Cheong, M. Van Kreveld, y M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. doi: 10.1007/978-3-540-77974-2.
- [22] «Convex Hull using Graham Scan», *GeeksforGeeks*, 24 de julio de 2013. <https://www.geeksforgeeks.org/convex-hull-using-graham-scan/> (accedido 1 de septiembre de 2023).
- [23] A. Fitzgibbon, M. Pilu, y R. B. Fisher, «Direct least square fitting of ellipses», *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 21, n.º 5, pp. 476-480, may 1999, doi: 10.1109/34.765658.
- [24] A. Marin-Hernandez, F. González, y H. Ríos-Figueroa, «Ajuste de modelos geométricos por métodos de consenso de muestras aleatorias», 2016, pp. 1-11.
- [25] «Algoritmo de Levenberg-Marquardt», *Wikipedia, la enciclopedia libre*. 14 de diciembre de 2022. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Algoritmo_de_Levenberg-Marquardt&oldid=147920251
- [26] «Wikiwand - Algoritmo de Levenberg-Marquardt», *Wikiwand*. https://www.wikiwand.com/es/Algoritmo_de_Levenberg-Marquardt (accedido 1 de septiembre de 2023).
- [27] «Algoritmos de mínimos cuadrados (ajuste de modelos) - MATLAB & Simulink - MathWorks España». <https://es.mathworks.com/help/optim/ug/least-squares-model-fitting-algorithms.html> (accedido 1 de septiembre de 2023).
- [28] «scipy.optimize.least_squares — SciPy v1.11.2 Manual». https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html (accedido 1 de septiembre de 2023).
- [29] B. Wang, S. Yin, y M. Liu, «Investigation on the Displacement Ductility Coefficient of Reinforced Concrete Columns Strengthened with Textile-Reinforced Concrete», *Advances in Civil Engineering*, vol. 2021, pp. 1-12, dic. 2021, doi: 10.1155/2021/3152619.
- [30] Department of Physics, Faculty of Education, Ain Shams University (Roxy, Cairo, Egypt) *et al.*, «Charged Particle Pseudorapidity Distributions for Pb-Pb and Au-Au Collisions Using Neural Network Model», *Ukr. J. Phys.*, vol. 58, n.º 8, pp. 709-718, ago. 2013, doi: 10.15407/ujpe58.08.0709.
- [31] «Método de Powell - OPTIMIZACIÓN EN SISTEMAS DINÁMICOS». <https://1library.co/article/m%C3%A9todo-powell-optimizaci%C3%B3n-sistemas-din%C3%A9micos.y4geo8ry> (accedido 1 de septiembre de 2023).

- [32] K. Boldman, L. A. Kriese, L. Van Vleck, C. P. Tassell, y S. Kachman, «A Manual for Use of MTDFREML – a Set of Programs to Obtain Estimates of Variances and Covariances (draft)», *USDA, ARS*, ene. 1995.
- [33] «Método Nelder-Mead», *Wikipedia, la enciclopedia libre*. 26 de enero de 2021. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=M%C3%A9todo_Nelder-Mead&oldid=132712944
- [34] «Breaking down the Nelder Mead algorithm · Mathias Brandewinder blog». <https://brandewinder.com/2022/03/31/breaking-down-Nelder-Mead/> (accedido 1 de septiembre de 2023).
- [35] «scipy.optimize.minimize — SciPy v1.11.2 Manual». <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html> (accedido 1 de septiembre de 2023).
- [36] R. Barati, «Analysis and Evaluation of Optimization Algorithms Application for Parameter Estimation of Muskingum Flood Routing Models in Rivers», 2014, doi: 10.13140/RG.2.2.20181.86244.
- [37] M. A. Fischler y R. C. Bolles, «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography», *Commun. ACM*, vol. 24, n.º 6, pp. 381-395, jun. 1981, doi: 10.1145/358669.358692.
- [38] saurabh dasgupta, «Outlier detection using the RANSAC algorithm», *MLearning.ai*, 31 de enero de 2020. <https://medium.com/mllearning-ai/outlier-detection-using-the-ransac-algorithm-de52670adb4a> (accedido 1 de septiembre de 2023).
- [39] «SymPy». <https://www.sympy.org/en/index.html> (accedido 1 de septiembre de 2023).
- [40] «Suite Calculadora - GeoGebra». <https://www.geogebra.org/suite> (accedido 1 de septiembre de 2023).
- [41] P. L. Rosin, «Fitting superellipses», *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, n.º 7, pp. 726-732, jul. 2000, doi: 10.1109/34.865190.
- [42] «Método de Montecarlo», *Wikipedia, la enciclopedia libre*. 16 de enero de 2023. Accedido: 1 de septiembre de 2023. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=M%C3%A9todo_de_Montecarlo&oldid=148634952
- [43] «numpy.arctan2 — NumPy v1.25 Manual». <https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html> (accedido 1 de septiembre de 2023).
- [44] «Sign Up - Udacity». <https://auth.udacity.com/sign-up> (accedido 1 de septiembre de 2023).

Planificación temporal y presupuesto

Para la elaboración de este trabajo se ha empleado una gran cantidad de tiempo y recursos. En esta sección se desglosará, a nivel general, la manera en la que se ha empleado el tiempo e invertido los recursos.

Planificación temporal

Como en cualquier otro proyecto o investigación, dividido en distintas fases, la línea temporal puede dividirse en períodos según la tarea y su duración. Los períodos de este caso podrían dividirse en:

1. Gestión del proyecto

Dediqué el mes de octubre a preguntar e informarme sobre los temas disponibles o propuestos. Hasta hablar con William, que me propuso una primera idea de investigación bastante interesante y que me podría ayudar a ampliar mis conocimientos en otros campos. En la segunda reunión el tema cambio, pero la base y conocimientos seguían la misma línea. Desde aquella reunión a finales de octubre podría decirse que comenzó el trabajo.

2. Estudio de antecedentes

Al continuar este trabajo la línea de investigación comenzada por William con “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares”, era de gran importancia comprender dicho estudio en detalle. Se aprovecho el poco tiempo del que se disponía en noviembre para esta tarea.

3. Estado del arte

De nuevo la limitación, tanto de conocimientos como de tiempo, afecto en la selección de la siguiente tarea. Comenzó la búsqueda de proyectos relacionados, actuales o que sirvieran como base de inicio del desarrollo. Para esta tarea se emplearon los meses de diciembre y enero. Era importante dedicarle el mayor tiempo posible para aprovechar la mayor cantidad de recursos, pero fijando el inicio del segundo cuatrimestre como final de la etapa, para así poder comenzar con el desarrollo y disponer de suficiente tiempo para modelos y resolución de los problemas que pudiesen surgir.

4. Formación

El inicio del desarrollo de los modelos o algoritmos tuvo que postponerse debido a la falta de formación para dar su comienzo. Con un previo inicio, se notó la carencia de conocimientos tanto de programación con Python®, como de las geometrías a emplear. Se dedicó la primera semana de febrero a realizar un curso intensivo de Python® a través de Udacity [44]. Continuando con una segunda semana de pruebas y adaptación al entorno de Google Colab, gracias a unos vídeos tutoriales proporcionados por el tutor. Para iniciar el desarrollo de los algoritmos se estudiaron en detalle las geometrías superelipse y *Convex Hull*, además de las herramientas de optimización.

5. Modelos y resultados

Este periodo puede dividirse en tres, donde alguno de ellos comparte parte de la línea temporal. El primer modelo fue el de la superelipse basada en su polinomio implícito. Dicho estudio necesito de los meses de marzo y gran parte de abril. A finales de abril, cuando se dejó de ver futuro a dicho modelo se comenzó a buscar una nueva metodología. Llegándose a plantear el

estudio de la superelipse a través de sus parámetros. Este comenzó a finales de abril llegando hasta finales de julio o inicios de agosto. Ya que, a la hora de obtener resultados, hubo que ir modificando o mejorando el modelo. Viendo su prolongación inesperada en el tiempo, en junio, se decidió trabajar en paralelo con el modelo de *Convex Hull*. Llegando a obtener los resultados antes que el otro modelo, para mediados de julio.

6. Redacción memoria

Comenzando el 16 de agosto hasta finales de la semana del 21-27 de agosto se trasladó todo el desarrollo del trabajo a una primera versión del TFG. El tutor fue receptor de dicha versión para poder revisarla y plantear cambios, modificaciones o sugerencias. Posteriormente, se trabajó en la entrega final.

7. Preparación de la defensa

Tras disponer de la versión definitiva del TFG se dio comienzo a la preparación de la presentación. En esta etapa destacan: preparación de la presentación, pruebas de la defensa por parte del alumno y defensa a modo de ejemplo al tutor, para recibir las últimas críticas constructivas de cara a la defensa oficial. Todo ello en el mes de septiembre.

Para resumir esta planificación y desarrollo se realizó el siguiente diagrama de Gantt simplificado:

Tabla 8. Diagrama de Gantt simplificado

Tareas	oct-22	nov-22	dic-22	ene-23	feb-23	mar-23	abr-23	may-23	jun-23	jul-23	ago-23	sep-23
Gestión del proyecto												
Estudio antecedentes												
Estado del arte												
Formación												
Superelipse (polinomio implícito)												
Superelipse (paramétrizable)												
ConvexHull												
Memoria												
Defensa												

Presupuesto

En este apartado se llevaba a cabo una estimación del presupuesto del trabajo. Para ello se diferenciarán los costes materiales y los recursos humano.

Costes materiales

En esta recopilación de recursos se ve reflejado: el hardware empleado, el software necesario y otros posibles gastos indirectos.

Tabla 9. Costes materiales

Concepto	Importe	Amortización	Coste
Ordenador	1000	20%	200
Curso Udacity	0	-	0
Google Colab	0	-	0
Gastos eléctricos	160	-	160
Total			360

Costes de recursos humanos

Todos ellos expresados de manera unitaria por hora de trabajo. Siendo el coste unitario del tutor de 30 €/h y el del alumno de 12 €/h.

Tabla 10. Costes de recursos humanos

Concepto	Unidades	Precio unitario	Coste
Reuniones con el tutor	12	30	360
Reuniones con el tutor	12	12	144
Documentación	40	12	480
Formación	70	12	840
Modelos y resultados	290	12	3480
Memoria	50	12	600
Total			5904

Costes totales

El resultado total de la estimación es: **6264 €.**

Tabla 11. Costes totales

	Coste
Material	360
Humano	5904
Total	6264

Evaluación de impactos

En esta sección, se evaluarán los impactos obtenidos con el desarrollo del proyecto. Analizando los efectos, beneficios y resultados en relación con los objetivos y metas planteados inicialmente. Analizando especialmente el impacto técnico.

En cuanto al impacto en la de automatización de prótesis con la ayuda de la inteligencia artificial, no se han logrado cumplir con las altas expectativas. Como se ha visto, la implementación de la superelipse no es del todo automática en comparación con el resto de las técnicas, además de emplearse más tiempo para obtener resultados. La comparativa de tiempos sería el defecto más importante para su aplicación. Pero se ha logrado avanzar mucho en cuanto a formación e información de dicha geometría. También se ha recopilado información acerca de nuevas librerías y funciones que pueden ser implementadas en otras geometrías. Y se han recopilado los problemas o errores que han ido surgiendo para que en posibles casos de estudios se eviten y así se logre avanzar más eficazmente. Por el contrario, con el desarrollo del *Convex Hull* se han obtenido mejores resultados de lo esperado y lo que parecía una idea bastante compleja se ha logrado implementar con facilidad.

A nivel personal se ha logrado una gran adquisición de conocimientos teóricos. Con este trabajo se buscaba empezar la formación en inteligencia artificial y el trabajo con el entorno Python®, cosa que se ha cumplido mejor de lo esperado. También hay que destacar el aprendizaje o adaptación a nuevos e inesperados problemas, tomando nota de errores cometidos y como lograr evitarlos en el desarrollo de futuros trabajos, o a nivel profesional y personal.

Se espera, que al igual que este trabajo seguía una línea de estudio, este pueda inspirar a otros o continuar con dicha línea. El estudio de la mejora en la creación de prótesis, apoyado en la inteligencia artificial, debería pasar a ser un tema emergente y no dejar que investigaciones como esta se dejen estancadas.

Análisis de aspectos legales y éticos

En el desarrollo de este proyecto se ha cumplido en todo momento con la normativa específica acerca de la elaboración de un Trabajo de Fin de Grado, tanto de la UPM como la de la escuela. Cumpliéndose de la misma manera con los estatutos de la Universidad Politécnica de Madrid y cualquier documento anexo a este como puede ser el código ético de la Escuela Técnica Superior de Ingenieros Industriales.

Se ha llevado un correcto uso de la información ajena. En todo momento se ha respetado el derecho de autor o propiedad intelectual, citando o haciendo referencia a trabajos que han servido de inspiración o han tenido un papel importante en el desarrollo del trabajo. No ha sido necesaria prestar una especial atención a la protección de datos puesto que no se ha trabajado con datos personales o de personas ajena. Las imágenes médicas empleadas en el estudio son representaciones de un modelo digital. Para ello se han empleado modelos digitales (archivos STL) de libre acceso acompañados de la aplicación Chitubox®, programa gratuito, para obtener sus imágenes tomográficas.

Para el código, basado en el lenguaje Python®, se ha empleado la aplicación Google Colab. Al disponer de una cuenta de Google se tiene acceso a una versión gratuita que permite trabajar hasta con tres trabajos en línea al mismo tiempo. Podría valorarse pagar por la versión Pro, pero en este caso dicha herramienta ha cumplido con las necesidades de ejecución y administración de código.

Contribución a los Objetivos de Desarrollo Sostenible

Como llevan marcados desde 2015, los Objetivos de Desarrollo Sostenible, son una ruta establecida por las Naciones Unidas de cara a la Agenda 2030. Además de los desafíos que se tratan de resolver en cada tarea o trabajo, nunca hay que perder de vista dichos objetivos, por lo que conviene interconectarlos o tenerlos presente lo máximo posible en el día a día. Para este caso particular, un Trabajo de Fin de Grado, conviene relacionarlo lo máximo posible con uno de ellos. El fin de este trabajo impulsa una mejora en la salud, por lo que es aplicación directa del ODS número 3 “Salud y bienestar”. Otros ODS que podrían verse afectados positivamente, aunque en menor medida, son: el ODS número 9 “Industria, innovación e infraestructura”, ya que es una relación directa del desarrollo de cualquier TFG de la rama de ingeniería, y el ODS número 12 “Producción y consumo responsables”. Este último viene relacionado indirectamente con la mejora que se busca de las prótesis. Aunque directamente afecte al ODS 3, al mejorar la salud, indirectamente afecta al 12. Ya que la mejora de las prótesis no solo mejora la vida del individuo, sino que también la de la propia prótesis. Con un mejor diseño se reducen los fallos o desgastes, logrando reducir la producción o cambio de prótesis por fallo o envejecimiento.



Figura 26. ODS relacionados con el TFG

Índice de figuras

Figura 1. Flujo de trabajo para prótesis personalizadas. Fuente: “Additive Manufacturing Processes in Medical Applications” [4].	11
Figura 2. Segmentación de un fémur proximal. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].....	11
Figura 3. (A) Imagen Tomográfica, (B) Imagen en sistema de coordenadas - algoritmo <i>subpixel_edges</i> y (C) separación de los contornos exterior e interior – DBSCAN. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].....	12
Figura 4. Resumen del desarrollo del ajuste de superelipses en "Minimizing Stress Shielding and Cement Damage in Cemented Femoral Component of a Hip Prosthesis through Computational Design Optimization" [6].	14
Figura 5. (A) Geometría de una elipse y (B) geometría de un óvalo. Fuente: Wikipedia [15].....	16
Figura 6. Representación de distintas superelipses variando su grado “n”. Fuente: Wikipedia [18]. ...	18
Figura 7. Ejemplo de Convex Hull. Fuente: Medium [20].....	20
Figura 8. Esquema explicativo del algoritmo de Graham-Scan para hallar el Convex Hull. Fuente: GeeksforGeeks [22].....	21
Figura 9. Parámetros de la elipse y distancia algebraica. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].	22
Figura 10. (A) Flujograma básico del algoritmo de Levenberg-Marquardt y (B) flujograma más técnico de dicho algoritmo. Fuentes: “Investigation on the Displacement Ductility Coefficient of Reinforced Concrete Columns Strengthened with Textile-Reinforced Concrete” [29] y “Charged Particle Pseudorapidity Distributions for Pb-Pb and Au-Au Collisions Using Neural Network Model” [30].	25
Figura 11. Flujograma del método de Powell. Fuente: “A Manual for Use of MTDFREML – a Set of Programs to Obtain Estimates of Variances and Covariances (draft)” [32]	27
Figura 12. Pasos de una iteración de Nelder-Mead: (A) Ordenamiento de vértices (B) Calculo de centroide y reflejo (C) Expansión (D) Contracción (E) Reducción. Fuente: “Breaking down the Nelder Mead algorithm” [34].....	29
Figura 13. Flujograma del algoritmo Nelder-Mead. Fuente: “Analysis and Evaluation of Optimization Algorithms Application for Parameter Estimation of Muskingum Flood Routing Models in Rivers” [36].	30
Figura 14. Regresión lineal (A) convencional y (B) con RANSAC. Fuente "Outlier detection using the RANSAC algorithm" [38].....	31
Figura 15. Resultados de la integración de RANSAC al ajuste algebraico circular. Fuente: “Diseño asistido por inteligencia artificial de dispositivos médicos personalizados: aplicación a prótesis vasculares y articulares” [1].	32
Figura 16. Representación del ajuste con superelipsen4_fit.....	34
Figura 17. Vector de coeficientes resultante del ajuste anterior.....	34
Figura 18. Vector de coeficientes para el ajuste empleando superelipsen4_fit a m = 15 puntos.....	35
Figura 19. Representación del ajuste con superelipsen4_fit a m = 15 puntos.....	35
Figura 20. Intento de ajuste con superelipsen4_fit a m = 15 puntos aleatorios.....	36
Figura 21. Leyenda de los resultados	45
Figura 22. Boceto con el desglose de la idea en un contorno real.....	55
Figura 23. Primera solución con ConvexHull	56
Figura 24. (A) Resultados función <i>convexhull_adaptable1</i> , (B) resultados <i>convexhull_adaptable2</i> y (C) comparación por áreas de ambas funciones.	57
Figura 25. Leyenda de los resultados con Convex Hull.....	64
Figura 26. ODS relacionados con el TFG	75

Índice de tablas

Tabla 1. Resumen métodos de optimización.....	37
Tabla 2. Tabla comparativa de los ajustes con minimizacion_angulo y cada función comparación	48
Tabla 3. Tabla comparativa de los ajustes con RANSAC (minimizacion_simple) y cada función comparación	52
Tabla 4. Resultados de ajustes a los inliers del ajuste elíptico: primer ajuste (previo a RANSAC) y ajuste final (con RANSAC).....	54
Tabla 5. Resultados de ajustes de superelipses a tibia, arteria y fémur	62
Tabla 6. Resultados ajuste a inliers de tibia, arteria y fémur	63
Tabla 7. Resultado de las dos versiones de ConvexHull aplicadas a tibia, arteria y fémur	65
Tabla 8. Diagrama de Gantt simplificado.....	72
Tabla 9. Costes materiales	72
Tabla 10. Costes de recursos humanos	73
Tabla 11. Costes totales	73

Abreviaturas

<i>Abreviatura</i>	<i>Significado de la abreviatura</i>
TC	Tomografía computarizada
3D	Tridimensional
CAD	Diseño asistido por ordenador
DBSCAN	Algoritmo de agrupamiento espacial basado en densidad
STL	Stereolithography
RANSAC	Algoritmo de consenso de muestras aleatoria
2D	Bidimensional
LMA o LM	Algoritmo de Levenberg-Marquardt
DLS	Método de mínimos cuadrados amortiguados
GNA	Algoritmo de Gauss-Newton
PSO	Optimización por enjambre de partículas

Apéndices

Apéndice 1: Función superelipsen4_fit

```
def superelipsen4_fit(data):  
    Dx = []  
    for i in range(0, data.shape[0], 1):  
        x = data[i][0]  
        y = data[i][1]  
        Dx.append([x**4, x**3*y, x**2*y**2, x*y**3, y**4, x**3, x**2*y,  
        x*y**2, y**3, x**2, x*y, y**2, x, y, 1])  
    D = np.dot(np.transpose(Dx), np.array(Dx))  
    Eva, Eve = np.linalg.eig(D)  
    posicion = np.where(Eva == np.amin(Eva)) #tuple  
    coef = Eve[:, posicion[0][0]]  
    return coef
```

Apéndice 2: Función superellipse

```
def superellipse(variables, data):  
    xc = variables[0]  
    yc = variables[1]  
    a = variables[2]  
    b = variables[3]  
    teta = variables[4]  
    n = variables[5]  
  
    cos_teta = np.cos(teta)  
    sin_teta = np.sin(teta)  
  
    Sol = []  
  
    if np.ndim(data) == 1: # Opción para vector  
        x = data[0]  
        y = data[1]  
  
        term1 = (((x - xc) * cos_teta - (y - yc) * sin_teta) / a)  
        ** (n)  
        term2 = (((y - yc) * cos_teta + (x - xc) * sin_teta) / b)  
        ** (n)  
  
        sol = term1 + term2 - 1  
  
        Sol = np.append(Sol, sol)
```

```
        # Opción para matriz
    else:
        for i in range(data.shape[0]):
            x = data[i][0]
            y = data[i][1]

            term1 = (((x - xc) * cos_teta - (y - yc) * sin_teta) / a)
            ** (n)
            term2 = (((y - yc) * cos_teta + (x - xc) * sin_teta) / b)
            ** (n)

            sol = term1 + term2 - 1

            Sol = np.append(Sol, sol)

    return Sol
```

Apéndice 3: Función objetivo a minimizar

```
# Función objetivo para minimizar

def fobjective(variables, data):
    predicción = superelipse(variables, data)
    return np.sum((predicción) ** 2)
```

Apéndice 4: Función comparación de áreas

```
def comparacion_areas(data, variables):
    return area_contorno(data) - area_superelipse(variables)
```

Apéndice 5: Función para calcular el área total encerrada por del contorno de la imagen

```
import numpy as np
from scipy.spatial import ConvexHull

def area_contorno(data):
    contorno = ConvexHull(data)
    return hull.volume
```

Apéndice 6: Función para estimar el área de una superelipse

```
def area_superellipse(variables, muestras=1000000):
    xc, yc, a, b, theta, n = variables

    # Generar muestras aleatorias

    x_m = np.random.uniform(xc - a, xc + a, muestras)
    y_m = np.random.uniform(yc - b, yc + b, muestras)

    # Verificar si cada muestra está dentro de la superelipse

    in_superellipse = (np.abs(((x_m - xc) * np.cos(theta) - (y_m - yc)
    * np.sin(theta)) / a) ** n + np.abs(((y_m - yc) * np.cos(theta) + (x_m
    - xc) * np.sin(theta)) / b) ** n ) <= 1

    # Calcular el área estimada

    area_estimada = np.sum(in_superellipse) / muestras * 4 * a * b

    return area_estimada
```

Apéndice 7: Función para obtener la media de un ajuste

```
def mean_error(data, variables):
    prediccion = superellipse(variables, data)
    return np.mean(prediccion)
```

Apéndice 8: Función para obtener el mayor error de un ajuste

```
def max_error(data, variables):
    prediccion = superellipse(variables, data)
    return np.max(prediccion)
```

Apéndice 9: Función minimizacion_angulo

```
def minimizacion_angulo(data, initial_guesses):

    best_fit = []
    total_diff = []
    difs = []
```

```
param = []

options = {'maxiter': 500}

if np.ndim(initial_guesses) == 1: # Opción para vector

    # Rango para xc (coordenada x del centro)
    xc_range = (-1*abs(initial_guesses[0]), 1*abs(initial_guesses[0]))

    # Rango para yc (coordenada y del centro)
    yc_range = (-1*abs(initial_guesses[1]), 1*abs(initial_guesses[1]))

    # Rango para a (semieje en el eje x)
    a_range = (0.9*abs(initial_guesses[2]),
    1.1*abs(initial_guesses[2]))

    # Rango para b (semieje en el eje y)
    b_range = (0.9*abs(initial_guesses[3]),
    1.1*abs(initial_guesses[3]))

    # Rango para teta (ángulo de rotación)
    teta_range = (-np.pi, np.pi)

    # Rango para eps (buscamos hiperelipse >1)
    grado_range = (2, 10)

    bnds = (xc_range, yc_range, a_range, b_range, teta_range,
grado_range)

    for angulo in np.arange(initial_guesses[4] - np.deg2rad(30),
initial_guesses[4] + np.deg2rad(30), np.deg2rad(15)):
        result = scipy.optimize.minimize(fobjectivo,
[initial_guesses[0], initial_guesses[1], random.uniform(a_range[0],
a_range[1]), random.uniform(b_range[0], b_range[1]), angulo,
initial_guesses[5]], args=(data,), method='Nelder-Mead', bounds=bnds,
options=options) ##Cambiando angulos
        #print(result.x)
        if result.success:
            #area_diff = abs(comparacion_areas(data, result.x))
            mae_diff = abs(mean_absolute_error(data, result.x))
            #max_diff = abs(max_error(data, result.x))

            #total_diff = area_diff + mae_diff + max_diff
            total_diff = mae_diff
            difs.append(total_diff)
            param.append(result.x)

best_fit = param[difs.index(min(difs))]
```

```
else:

    # Rango para xc (coordenada x del centro)
    xc_range = (-1*abs(initial_guesses[0][0]),
1*abs(initial_guesses[0][0]))

    # Rango para yc (coordenada y del centro)
    yc_range = (-1*abs(initial_guesses[0][1]),
1*abs(initial_guesses[0][1]))

    # Rango para a (semieje en el eje x)
    a_range = (0.9*abs(initial_guesses[0][2]),
1.1*abs(initial_guesses[0][2]))

    # Rango para b (semieje en el eje y)
    b_range = (0.9*abs(initial_guesses[0][3]),
1.1*abs(initial_guesses[0][3]))

    # Rango para teta (ángulo de rotación)
    teta_range = (-np.pi, np.pi)

    # Rango para eps (buscamos hiperelipse >1)
    grado_range = (2, 10)

    bnds = (xc_range, yc_range, a_range, b_range, teta_range,
grado_range)

    for initial_guess in initial_guesses:
        for angulo in np.arange(initial_guess[4] - np.deg2rad(30),
initial_guess[4] + np.deg2rad(30), np.deg2rad(15)):
            result = scipy.optimize.minimize(fobjectivo,
[initial_guess[0], initial_guess[1], random.uniform(a_range[0],
a_range[1]), random.uniform(b_range[0], b_range[1]), angulo,
initial_guess[5]], args=(data,), method='Nelder-Mead', bounds=bnds,
options=options) ##Cambiando angulos
            #print(result.x)
            if result.success:
                #area_diff = abs(comparacion_areas(data, result.x))
                mae_diff = abs(mean_absolute_error(data, result.x))
                #max_diff = abs(max_error(data, result.x))

                #total_diff = area_diff + mae_diff + max_diff
                total_diff = mae_diff
                difs.append(total_diff)
                param.append(result.x)

best_fit = param[difs.index(min(difs))]
```

```
    return best_fit
```

Apéndice 10: Función minimizacion_simple

```
def minimizacion_simple(data, initial_guess):  
  
    best_fit = []  
  
    # Rango para xc (coordenada x del centro)  
    xc_range = (-1*abs(initial_guess[0]), 1*abs(initial_guess[0]))  
  
    # Rango para yc (coordenada y del centro)  
    yc_range = (-1*abs(initial_guess[1]), 1*abs(initial_guess[1]))  
  
    # Rango para a (semieje en el eje x)  
    a_range = (0.7*abs(initial_guess[2]), 1*abs(initial_guess[2]))  
  
    # Rango para b (semieje en el eje y)  
    b_range = (0.7*abs(initial_guess[3]), 1*abs(initial_guess[3]))  
  
    # Rango para teta (ángulo de rotación)  
    teta_range = (-np.pi, np.pi)  
  
    # Rango para eps (buscamos hiperelipse >1)  
    grado_range = (2, 10)  
  
    bnds = (xc_range, yc_range, a_range, b_range, teta_range,  
            grado_range)  
  
    options = {'maxiter': 500}  
  
    result = scipy.optimize.minimize(fobjectivo, [initial_guess[0],  
                                                initial_guess[1], random.uniform(a_range[0], a_range[1]),  
                                                random.uniform(b_range[0], b_range[1]), initial_guess[4],  
                                                initial_guess[5]], args=(data,), method='Nelder-Mead', bounds=bnds,  
                                options=options)  
    #print(result.x)  
    if result.success:  
        best_fit = result.x  
  
    else:  
        best_fit = initial_guess  
  
    return best_fit
```

Apéndice 11: Modificación de RANSAC

```
def ransacselpip(max_iter, n, data, initial_guess):

    inliers = np.empty((0, 2)) # inliers, matriz vacia

    l_epoch = []
    l_parm = []
    l_param = []
    l_inliers = []

    for epoch in range(max_iter):
        ## Muestras
        spr = [] # se almacenan los index que se deben eliminar de la
        matriz de inliers
        sample = data[np.random.choice(data.shape[0], n, replace=False)] # N corresponde al tamaño de la muestra
        param_current = minimizacion_simple(sample, initial_guess) # Distancia algebraica con la muestra para determinar que puntos son
        inliers
        for i in range(sample.shape[0]):
            distance = superelipse(param_current, sample[i])
            if distance <= 0: # para asegurarse que la superelipse queda
            dentro de la sección interna, propiedades de la distancia algebraica
                inliers = np.append(inliers, sample[i].reshape((1,2)),
            axis=0) # reshape para poder agregar datos a la matriz

        # Eliminar líneas duplicadas en INLIERS
        inliers = np.unique(inliers, axis=0)
        # Verificación de inliers (segunda comprobación)/ necesito tres
        puntos para definir la superelipse
        if inliers.shape[0] >= 3: # condición para tener los suficientes
        puntos y hacer el análisis
            param_new = minimizacion_simple(inliers, initial_guess) #
            ajuste con los inliers
            # Media y desviación estándar de todos los puntos con el ajuste
            realizado con los inliers
            mean = np.mean(superelipse(param_new, data))
            std = np.std(superelipse(param_new, data))
            if mean - std >= 0:
                l_epoch.append(epoch + 1)
                l_parm.append(mean - std)
                l_param.append(param_new)
                l_inliers.append(inliers)
            else:
                for j in range(inliers.shape[0]):
                    distance = superelipse(param_new, inliers[j])
```

```
        if distance >= 0: #para asegurarse que la elipse queda
            dentro de la sección, tanto como sea posible
                spr.append(j) # se agregan los index de los puntos que
            deben eliminarse
            inliers = np.delete(inliers, spr, axis=0) #para eliminar los
            puntos

        print(epoch)

    if l_parm:
        # Para encontrar el valor máximo
        max_value = max(l_parm)
        max_inliers = l_inliers[l_parm.index(max_value)]
        param_def = l_param[l_parm.index(max_value)]

    else: #para el caso en el que no haya solución devuelva param_def
        como array en vez de error o nada (y poder continuar el bucle externo)
        max_value = np.array([])
        max_inliers = np.array([])
        param_def = np.array([])

    return param_def
```

Apéndice 12: Función ajuste completa

```
def ajuste_superelipse(data, initial_guess):

    total_diff = []
    difs = []
    param = []
    best_fit = []

    primer_ajuste = minimizacion_angulo(I, initial_guess)

    for angulo in np.arange(primer_ajuste[4] - np.deg2rad(30),
                           primer_ajuste[4] + np.deg2rad(30), np.deg2rad(15)):
        ajuste = ransacselip(max_iter = 1000, n = 100, data = I,
                             initial_guess = [primer_ajuste[0], primer_ajuste[1], primer_ajuste[2],
                                              primer_ajuste[3], angulo, primer_ajuste[5]])
        if ajuste.size == 0: # Verificar si ajuste es un array vacío,
            para en ese caso pasar al siguiente parámetro del bucle en vez de
            frenarlo
            continue

        area_diff = abs(comparacion_areas(I, ajuste))
        #mae_diff = abs(mean_absolute_error(I, ajuste))
        #max_diff = abs(max_error(data, ajuste))
```

```
#total_diff = area_diff + mae_diff + max_diff
total_diff = area_diff
difs.append(total_diff)
param.append(ajuste)

best_fit = param[difs.index(min(difs))]

print(best_fit)

return best_fit
```

Apéndice 13: Función convexhull_adaptable1

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import ConvexHull

def convexhull_adaptable1(data, valor):
    # Calcular el convex hull de los puntos
    hull = ConvexHull(data)
    hull_puntos = data[hull.vertices]

    # Calcular el centro
    centro = np.mean(hull_puntos, axis=0)
    angulos = np.arctan2(hull_puntos[:, 1] - centro[1], hull_puntos[:, 0] - centro[0])
    indices_orden = np.argsort(angulos)
    hull_puntos_orden = hull_puntos[indices_orden]

    # Calcular los vectores
    vectores = hull_puntos_orden - centro

    # Normalizar los vectores
    normas = np.linalg.norm(vectores, axis=1)
    vectores_norm = vectores / normas[:, np.newaxis]

    # Desplazar los puntos del convex hull
    puntos_separados = hull_puntos_orden + valor * vectores_norm

    # Cerrar la curva
    puntos_separados = np.vstack([puntos_separados,
                                   puntos_separados[0]])

    return puntos_separados
```

Apéndice 14: Función ordenacion_graham

```
def ordenacion_graham(data):
    def angulo_polar(p):
        return np.arctan2(p[1] - start[1], p[0] - start[0])

    # Encontrar y seleccionar el punto más bajo como punto de partida
    start = data[np.argmin(data[:, 1])]

    # Ordenar los puntos en función del ángulo polar con respecto al
    # punto de partida
    puntos_orden = sorted(data, key=angulo_polar)

    return np.array(puntos_orden)
```

Apéndice 15: Función convexhull_adaptable2

```
def convexhull_adaptable2(data, valor):
    # Calcular el convex hull de los puntos
    hull = ConvexHull(data)
    hull_puntos = data[hull.vertices]

    # Utilizar ordenacion_graham para ordenar los puntos en sentido
    # antihorario
    hull_puntos_orden = ordenacion_graham(hull_puntos)

    # Calcular el centro
    centro = np.mean(hull_puntos_orden, axis=0)

    # Calcular los vectores
    vectores = hull_puntos_orden - centro

    # Normalizar los vectores
    normas = np.linalg.norm(vectores, axis=1)
    vectores_norm = vectores / normas[:, np.newaxis]

    # Desplazar los puntos del convexhull
    puntos_separados = hull_puntos_orden + valor * vectores_norm

    # Cerrar la curva
    puntos_separados = np.vstack([puntos_separados,
                                   puntos_separados[0]])

    return puntos_separados
```