ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE LA SALUD

CONVERSIÓN DE CÓDIGO DE MATLAB A R DE OTOLAB (ANÁLISIS DE IMÁGENES DE OTOLITOS DE PECES)

CODE CONVERSION FROM MATLAB TO R FROM OTOLAB (IMAGE ANALYSIS OF FISH OTOLITHS)

Realizado por
ANDREA CARMONA RODRÍGUEZ

Tutorizado por
ENRIQUE NAVA BARO

Departamento
INGENIERÍA DE COMUNICACIONES
UNIVERSIDAD DE MÁLAGA

MÁLAGA, Julio 2020

Agradecimientos

A mi familia por apoyarme siempre
Y a mis amigas por hacerlo divertido

Resumen:

En este trabajo final de grado se desarrolla la conversión de código de Matlab a R del programa OTOLAB para el análisis de imágenes de otolitos de los peces. El estudio se ha centrado especialmente en el módulo de análisis morfométrico y en elaborar la estrategia óptima para la portabilidad del resto del código.

Para desarrollar dicho código, se ha utilizado tanto Matlab como RStudio. Para ello, se han estudiado ambos lenguajes y su interoperabilidad. Se ha aprendido el funcionamiento los diversos algoritmos de análisis digital de imágenes y las funciones desarrolladas sobre ello en ambos lenguajes o la producción de algunas de ellas cuando ha sido necesario. Una vez estudiadas las funciones, se tradujo el código y se produjeron *scripts* para realizar una serie de pruebas y validar la calidad de este.

Palabras claves: otolito, análisis digital de imágenes, R, Matlab.

Abstract:

The aim of this graduate thesis is to analyse fish's otolith images using a code Matalab R conversion with the programme OTOLAB. Morphometric data analysis and strategy have been used to maximize portability with the rest of the code using both Matlab and RStudio.

These programmes develop the languages used and its interoperability. We have been progressing through algorithms and images digital analysis and functions with both languages being used or a function production if necessary. Once we have assessed these functions, we have translated the code and we have produced scripts in order to assess it and to validate it.

Keywords: otolith, digital image analysis, R, Matlab.

Acrónimos

ETSI	Escuela Técnica Superior de Ingeniería
UMA	Universidad de Málaga
TFG	Trabajo Fin de Grado
GUI	Graphical User Interface
DNI	Documento Nacional de Identidad
JPEG	Joint Photographic Experts Group
TIFF	Tagged Image File Format
GNU	Gnu is Not Unix
GPL	General Public License
CRAN	The Comprehensive R Archive Network
SO	Sistema Operativo
GIS	Geographic Information System
FTP	File Transfer Protocol
RGB	Red Green and Blue
CLAHE	Contrast Limited Adaptive Histogram Equalization

Índice de contenidos

Índice de figuras

CAPÍTULO 1

Introducción

En la actualidad, los otolitos de los peces son una parte muy importante del oído interno de los peces óseos. Se utilizan en diversos tipos de estudios asociados a múltiples disciplinas: ecología trófica, paleontología, arqueología, ecomorfología, filogenia y biología pesquera. Otra rama en la que destaca el uso de los otolitos es la biomedicina, en la que se ha estudiado su posibilidad como biominerales. Esto se ha utilizado como recurso en la regeneración ósea y dental.

La diversidad de aplicaciones de los otolitos y todas sus potencialidades han propiciado la necesidad de una sinergia entre dos campos, la investigación y la ingeniería. Un claro ejemplo es el software de análisis de imágenes de otolitos que se presenta en este trabajo, con el desarrollo de la conversión de Matlab a R.

1.1. Objetivos

El objetivo de este trabajo es realizar una conversión del código abierto OTOLAB, desarrollado conjuntamente por la Universidad de Málaga y el Instituto Español de Oceanografía para el análisis de imágenes de otolitos de peces. Este software tiene una estructura modular, con tres aplicaciones principales actualmente: registro, segmentación y análisis morfométrico de imágenes, así como una interfaz gráfica de usuario común a todas ellas. En este TFG se ha realizado la conversión de código del módulo de análisis morfométrico y se ha elaborado una estrategia óptima para la portabilidad del resto del código.

Para ello se utilizarán tanto los conocimientos en Matlab adquiridos en diversas asignaturas del grado en Ingeniería de la Salud y aspectos aprendidos con anterioridad en Imágenes Biomédicas como los formatos de imagen y sus posibilidades de compresión, el estudio sobre implementación y aplicación de algoritmos de tratamiento digital de imagen para la mejora de la calidad , como también el uso de algoritmos para segmentar, describir y cuantificar imágenes.

Por último, se realizarán pruebas de evaluación y comparación del software desarrollado en R.

Por tanto, los objetivos serán:

- Familiarización con los entornos de programación Matlab y R, con énfasis en los algoritmos, toolboxes o paquetes relacionados con el tratamiento digital de imágenes.
- Estudio de herramientas que faciliten la portabilidad automática entre Matlab y R, tales como matconv o matlab.R, y evaluar su aplicabilidad a algoritmos de tratamiento digital de imágenes.
- Estudio de los diversos algoritmos implicados en el análisis digital de imágenes.
- Comparativa de funciones propias de librerías en R de tratamiento digital de imágenes, con Image Processing Toolbox de Matlab.
- Creación de diversas funciones de análisis digital, inexistentes en R.
- Conversión del código de análisis morfométrico de OTOLAB a R. Desarrollo de las funciones de OTOLAB.
- Realización de pruebas de evaluación, *scripts*, gráficas para facilitar la comparación del software desarrollado en R.

1.2. Estado del arte

El desarrollo de un pez y las condiciones en las que se encuentran durante este desarrollo, dejan su huella en una pequeña estructura localizada en el oído interno del pez: el otolito. Podría decirse que un otolito es comparable a la huella digital o DNI de un humano, ya que no hay dos iguales. En cambio, al igual que los anillos concéntricos de los troncos de los árboles, las diferentes precipitaciones de carbonato cálcico en torno a un núcleo inicial que conforman el otolito anuncian aún más aspectos que una huella digital: además de la edad podemos conocer en qué circunstancias se desarrolló, que condiciones ambientales se dieron, si sufrió algún déficit nutritivo durante su vida e incluso si luchó contra algún elemento contaminante.

Los diversos campos en los que los otolitos destacan con una gran utilidad son ecología trófica, es decir, el estudio de los organismos y sus interacciones alimenticias (presa-depredador) en un ecosistema, paleontología, arqueología, ecomorfología o morfología ecológica, lo cual es el estudio de la relación entre el papel ecológico de un individuo y sus adaptaciones morfológicas y la filogenia (ciencia que estudia la relación de parentesco entre especies o taxones en general y biología pesquera). Otra rama del uso de los otolitos, más alejada de la biología marina, es el de la biomedicina, estudiando la posibilidad de los otolitos como biominerales, utilizado como recursos en la regeneración ósea y dental.

Por la importancia biológica de los otolitos se creó OTOLAB, desarrollado para el análisis de estas microestructuras. Fue desarrollado originalmente en Matlab por ser este el lenguaje más común en la ingeniería. Sin embargo, las particularidades de trabajo en biología marina, como pueden ser trabajos de campo, es decir, proyectos de investigación donde se lleva la teoría al entorno donde se aplica o identifica, combinado con la amplia difusión de software de uso libre y código abierto y variabilidad de arquitecturas hace deseable que OTOLAB pueda ser portado a R, solventando cierta inestabilidad de las aplicaciones desarrolladas en Matlab que contienen las interfaces gráficas de usuario (GUI) diferentes según la versión del programa y arquitectura del ordenador, que dificultan su extensión con nuevas aplicaciones de biología marina.

Además, la conexión entre los datos obtenidos del análisis morfométrico de las imágenes de otolitos de peces con técnicas de *machine learning*, ampliamente desarrolladas en lenguaje R, hacen atractiva su portabilidad a R en lugar de otros lenguajes de uso libre, tales como Python.

1.3. Estructura de la memoria

• En el **Capítulo 1** se hace una breve introducción, estado del arte, y una estructura de la memoria.

• En el **Capítulo 2** se realiza una revisión sobre los órganos de los sentidos de los peces, el sistema vestibular y los otolitos.

• En el **Capítulo 3** se describe el código de la aplicación OTOLAB en Matlab, y la que se va a utilizar, RStudio, y se realiza una comparativa entre ambas, explicando la motivación

de esta conversión.

• En el **Capítulo 4** se estudia la interoperabilidad entre Matlab y RStudio.

• En el **Capítulo 5** se explica la metodología seguida.

• En el **Capítulo 6** se expone la evaluación de las diferencias en las principales funciones utilizadas en Matlab frente a R y la creación de las inexistentes.

• En el **Capítulo 7** se realizan una serie de pruebas y el proceso de validación de los resultados.

• En el **Capítulo 8** se expone la conclusión y las líneas futuras.

CAPÍTULO 2

Otolitos

El grupo al que todos conocemos como "peces" incluye alrededor de 30.000 especies de seres vivos.Una de los grupos en los que se pueden clasificar los peces son los vertebrados, distribuidos a su vez en distintas clases. Entre estas, se pueden encontrar Myxini (peces bruja, suponen el 0,25 % de los vertebrados), Petromyzontida (Cephalaspidomorphi, también conocidos como lampreas, suponen el 0.14 %) , Chondrichthyes (tiburones,rayas y quimeras son el 3,47 %), Actinopterygii (son el grupo de peces oseos y tienen el mayor porcentaje con un 96,12 %) y por último, Sarcopterygii (pecespulmonados son el 0,03 %).

Los peces constituyen el grupo que contiene el número de especies más elevado de vertebrados y que colonizó ampliamente los diversos ambientes acuáticos del planeta. El agua del planeta ocupa una superficie total de 510.000 103 Km^2 , de los cuales podemos diferenciar el 97,5 % está en los océanos y solo un 2,5 % es agua dulce. La heterogeneidad de estos ambientes acuáticos y sus propiedades es altamente diversa y, salvando las aguas subterráneas, los glaciares y los casquetes polares, en todos localizamos peces. Por lo que destaca la volubilidad de los peces, el éxito de esta colonización en una gama tan amplia de distintos hábitats es debido a su sistema sensorial, cuya especialidad es la percepción de estímulos en el medio. Por tanto, es vital entender a los peces como "maquinas sensoriales" muy eficientes, y el otolito es una parte de este engranaje [18].

2.1. Órganos de los sentidos de los peces

En términos sensoriales, los peces poseen distintos mecanismos, entre los que se encuentran la fotorrecepción, la quimiorrecepción, percepción electromagnética y mecanorrecepción.

La fotorrecepción (lo que se conoce en humanos como visión) se realiza a través de los ojos de los peces. Los ojos, que por norma general poseen el mismo componente en todos los vertebrados, perciben las imágenes y las interpretan. Además, los peces también pueden observar colores y, según la especie, hasta luz ultravioleta e infrarroja. No obstante se debe tener cuenta que el entorno visual en el agua varía mucho según una serie de factores como son la luminosidad, la presencia de partículas y otros elementos, lo que conlleva que la visión no sea siempre el sentido más beneficioso para los peces [18].

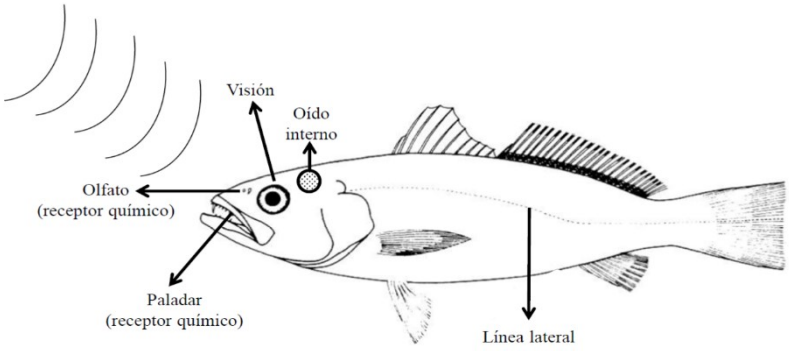


Figura 2.1: Órganos de los sentidos de los peces [18].

El siguiente mecanismo sensorial, denominado quimiorrecepción, envuelve al olfato y el paladar. Por un lado, el olfato depende tanto de la solución de diferentes sustancias químicas en el agua como del contacto de dicha solución con la roseta olfativa (un epileto sensorial situado en las fosas nasales). Por otro lado, el paladar se relaciona con la percepción de los alimentos y las células sensoriales. Estas se encuentran en la boca, barbilla y labios.

La quimiorrecepción o percepción química es vital en peces ya que, gracias a esta máquina sensorial, es capaz de reconocer aletas pélvicasaletas pélvicas (también denominadas pares) o identificar áreas en las especies con filopatria natal (tendencia a permanecer en el mismo territorio), para evitar depredadores.

El electromagnetismo en los peces es fundamentalmente la capacidad de percibir y emitir campos magnéticos y eléctricos, usados en la comunicación del ataque y la defensa. Depende de células especializadas en el tegumento (piel del pez).

Los mecanorreceptores son los órganos que perciben sonidos, el equilibrio y la presión. Comenzando con el sistema de la línea lateral, está formado por células pilosas, que a su vez se agrupan en neuromastatos, capaces de captar diferentes vibraciones de distintos orígenes. El aparato vestibular más concreto en las células pilosas situadas en el oído interno se encargan de la audición y del equilibrio, también está formado de más elementos que se explican con más profundidad a continuación [18].

2.2. Los otolitos y su ubicación en el aparato vestibular

Los otolitos de los peces son unos cuerpos calcáreos situados en el interior del aparato vestibular (oído interno de los peces teleósteos) parecidos a las otoconias de otros vertebrados. Son más grandes que las otoconias y tienen una morfología compleja y diferente según la especie. De igual modo que las otoconias, bajo la gravedad, los otolitos también impulsan las células sensoriales del vestíbulo mandando un impulso nervioso al cerebro del animal donde se interpreta la posición relativa del individuo comprándolo a la dirección de la gravedad, lo cual sustenta el equilibrio del pez y ejerce de sensor de profundidad. Otra función conocida es la detección de vibraciones. En los peces teleósteos hay tres pares de otolitos y cada uno tiene una localización, función, tamaño, forma y ultraestructura diferente al otro. Estos tres pares de otolitos son llamados *lapillus, sagitta y asteriscus.*

(Figura 2.2) [2].

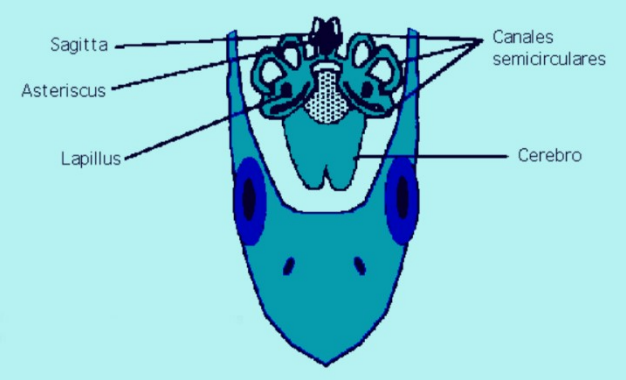


Figura 2.2: Organos de los sentidos de los peces [2].

Como se menciona anteriormente, el aparato vestibular de los peces se encuentra en su oído interno, en el interior de dicho aparato se encuentran los otolitos. El aparato vestibular en la mayoría de los casos, excepto si son peces planos, tiene simetría bilateral, y se divide en dos, una parte superior denominada saco dorsal, y una parte inferior llamada saco ventral. Luego, los otolitos *lapilli* se encuentran anteriormente en la parte superior y los sagittae y asterisci están situados normalmente muy cerca entre sí en la parte inferior. El vestíbulo que contiene los tres pares de otolitos se denominan utrículo, sáculo y lagena, respectivamente de los otolitos *lapilli*,*sagittae* y *asterisci*. Los *sagittae* suelen ser normalmente los más utilizados para el estudio, debido a que son más grandes respecto a los otros [18].

2.3. Uso de los otolitos

Los otolitos tienen unos anillos de crecimiento con gran similitud a los observados en los troncos de los árboles (Fig 2.3). El primer investigador en darse cuenta fue Reibisch en 1899. Los anillos son bandas translúcidas alteradas con bandas más opacas. El estudio de la sucesión de bandas informaba sobre la edad del pez.

Igualmente, al tener cada especie una forma y tamaño particular también han sido utilizados para la diferenciación entre especies, utilizando otolitos fósiles para derivar sus antiguos propietarios. Hasta la década de los setenta la "lectura"de otolitos se circunscribía a los anillos anuales (dos: uno translúcido y otro más opaco). Por tanto, solo servían para calcular la edad de los peces adultos. Pasados los setenta, se descubrió los anillos de crecimiento diario, a partir de este punto los otolitos tenían una precisión infinitamente mayor para el cálculo de edad, y se empezó a poder estudiar la edad también de los individuos jóvenes y larvas y lo más importante, su composición química: aparte de informar de los cambios fisiológicos en el organismo, también informaban de algunos cambios ambientales.

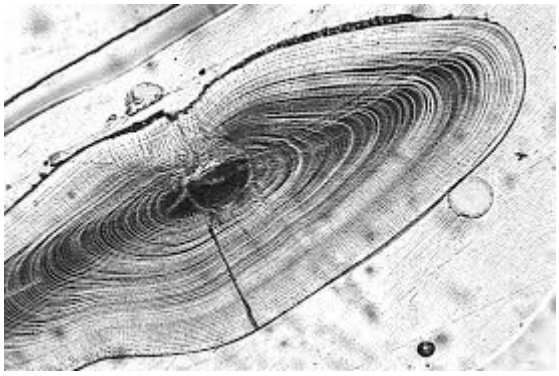


Figura 2.3: Otolito visto desde un microscopio [18].

A pesar de que los otolitos están formados principalmente por carbonato cálcico (90 %) cambios fisiológicos, como puede ser la nutrición del individuo o ciertos factores ambientales parecen favorecer la precipitación de otros elementos en pequeñísimas cantidades: estroncio, bario, magnesio, cadmio, cobalto, cobre, silicio, fosfato, boro, entre otros [2].

En definitiva, los otolitos aparte de informar sobre la edad de un individuo, informan también del ambiente en el que vive y las variaciones que ese entorno ha sufrido durante un periodo de tiempo. Este aspecto es aún más importante en especies migratorias ya que el otolito puede mostrar información de la ruta seguida. Más concretamente, los isótopos tanto de oxígeno como de carbono y estroncio anuncian a los investigadores sobre la temperatura del agua en las que el individuo habitó [2].

CAPÍTULO 3

Tecnologías empleadas

3.1. Matlab

Matlab (Matrix Laboratory) es un lenguaje de programación dedicado especialmente a la computación matemática diseñado especialmente para ingenieros y científicos. El entorno de escritorio tiene una forma natural de expresar matemática computacional como álgebra lineal, análisis de datos, procesamiento de señales e imágenes. Matlab presenta una solución específica de la aplicación llamada *Toolboxes*. Las cajas de herramientas proporcionan un conjunto de funciones de Matlab que resuelven un conjunto específico de problemas. Hay varias áreas donde están disponibles las cajas de herramientas, tales como procesamiento de señal digital, sistemas de control, red neuronal, simulaciones, aprendizaje profundo y muchas otras [3].

El software inicial, OTOLAB, está producido en Matlab, como ya se ha mencionado. Matlab es un lenguaje de alto rendimiento para la informática técnica. Integra el cálculo, la visualización y la programación en un entorno fácil de usar donde los problemas y las soluciones se expresan en una notación matemática familiar.

Este software es idóneo ya que proporciona un conjunto de herramientas como solución para una amplia clase de problemas en el procesamiento digital de imágenes. Concretamente Matlab ofrece un extenso conjunto de funciones para el procesamiento digital de imágenes. Estas funciones, así como la expresión del lenguaje de Matlab, hacen que las operaciones del procesamiento digital de imágenes se escriban de una manera intuitiva y clara. Por ello, se puede decir que es un software ideal para la resolución de problemas de procesamiento digital de imágenes y, por tanto, uno de los más utilizados en este campo [3].

Matlab sigue siendo de los lenguajes más utilizados para la implementación de algoritmos de cálculo científico, disponiendo de multitud de librerías que ayudan a ello. Destaca su potencia computacional y su capacidad para trabajar de manera vectorizada y matricial. Se trata del lenguaje comúnmente más enseñado en las universidades para impartir asignaturas de cálculo científico y análisis numérico. Además muchas empresas utilizan Matlab como herramienta principal para el desarrollo de software en ámbitos diversos como el análisis de datos o el desarrollo de modelos matemáticos, entre muchos otros. Este hecho permite que Matlab siga siendo siempre uno de los lenguajes más solicitados en el mercado de contratación de nuevos trabajadores.

3.2. R

R es un lenguaje y entorno de programación de libre distribución para análisis estadístico y gráfico normalmente asociado a tareas estadísticas, porque posee una gran variedad de librerías sobre tratamiento estadístico. Es uno de los lenguajes más utilizado en la investigación estadística. R forma parte del sistema GNU y se distribuye con la licencia GNU GPL. Presenta una gran variedad de herramientas estadísticas como son los modelos lineales y no lineales, análisis de series temporales, test estadísticos, algoritmos tanto de clasificación como de agrupamiento, etc. Otra característica importante para destacar es que puede integrarse con distintas bases de datos y, por otro lado, una gran capacidad gráfica, con la que se permite generar gráficos de alta calidad. Por último, su funcionalidad más trivial es la posibilidad de usarla como herramienta para el cálculo numérico.

R consiste en un lenguaje y un entorno de tiempo de ejecución con gráficos. Actúa como un depurador que tiene acceso a ciertas funciones del sistema y la capacidad de ejecutar programas almacenados en archivos de *script*.

La herramienta se basa en lo que se conoce como un proyecto colaborativo, fruto de las aportaciones de la comunidad e impulsando a los usuarios al desarrollo y ampliación del lenguaje, a la creación de funciones e incluso de librerías como conjunto de funciones, las cuales se han utilizado para el desarrollo del proyecto. Asimismo, R facilita la interacción del código con otros lenguajes de programación y con diferentes tipos de bases de datos, el intercambio de datos entre Matlab y R ha sido muy útil en la realización de pruebas para el trabajo, y la interacción de C con R también ha sido trivial para la creación de una función específica (ambas cosas se explican con detalle más adelante). Estas características provocan que sea muy utilizado tanto en universidades como empresas, ya que aporta al usuario una serie de cualidades muy eficaces respecto al tratamiento de datos de un tamaño significativo, la visualización de estos y la gran cantidad de herramientas estadísticas.

En la actualidad, es un lenguaje con gran relevancia debido al significativo auge de la minería de datos y la inteligencia artificial. Un significante número de empresas poseen grandes equipos de trabajadores cuya tarea está estrechamente relacionado con ambas metodologías, siendo R uno de los lenguajes preferidos por multitud de compañías para llevar a cabo dichas tareas.

RStudio es la herramienta que se ha utilizado para implementar tanto las funciones como los *scripts* para ejecutar dichas funciones, ya que es más completo que la consola de R y permite hacerlo de una manera más visual. RStudio es en concreto un entorno de desarrollo integrado para R, siendo una interfaz comercial que presenta una versión gratuita entre otras opciones de pago más avanzadas y con mayor número de utilidades disponibles. RStudio incluye una consola, editor de sintaxis que apoya la ejecución de código, y también tiene herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

El entorno de trabajo se visualiza mejor en la Figura 3.1:

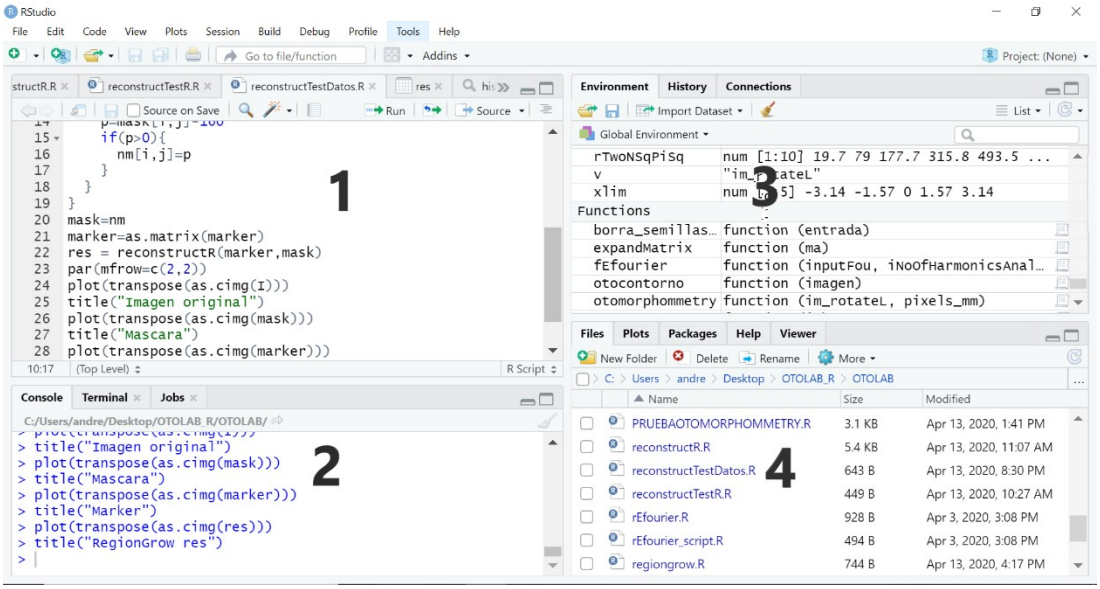


Figura 3.1: Interfaz de RStudio.

1. Esta sección es fundamentalmente el área de trabajo, donde se escribe el código que más tarde se ejecuta.
2. La consola, se puede escribir código y ejecutarlo dándole directamente a la tecla enter, también es donde se ejecuta el código cuando se ejecuta lo trabajado en la sección 1.
3. Esta sección sirve para abrir, visualizar y/o cargar archivos de datos, también se puede ver el historial de y hacer conexiones.
4. En la cuarta sección se pueden ver los archivos, se visualizan las gráficas que se ha ido programando, se observan los distintos paquetes instalados y seleccionados, y por último se visualizan los gráficos.

Por último, cabe destacar que hay una gran cantidad de librerías que no vienen por defecto en R, las cuales incorporan muchas funciones de gran utilidad. Estas se pueden instalar gratuitamente mediante el repositorio de paquetes, CRAN (*The Comprehensive R Archive Network*), de forma gratuita.

Al trabajar con ambos programas, es fácil darse cuenta de su gran cantidad de similitudes, ya que ambos ofrecen funciones matemáticas y estadísticas y procesan datos estructurados y no estructurados. A continuación, se realiza una comparativa entre ambos, destacando diversas características.

3.3. Comparativa entre Matlab y R

Al trabajar con ambos programas es fácil darse cuenta de su gran cantidad de similitudes, ya que ambos ofrecen funciones matemáticas y estadísticas y procesan datos estructurados y no estructurados. A continuación, se concreta una comparativa entre ambos, centrada en diversas características

Facilidad de aprendizaje

Matlab es un lenguaje de programación fácil de usar. Sus cajas de herramientas, las cuales realizan varias funcionalidades, facilitan aún más el uso para diversas tareas. Por otro lado, R es bastante complicado para los principiantes en lenguajes de programación (de hecho, es conocido por tener una empinada curva de aprendizaje). Sin embargo, tras hacer un uso reiterado, la complicación se atenúa. Además, R-Commander y R-Studio (las nuevas versiones de GUI para R) han beneficiado el aprendizaje a la comunidad.

Coste

La licencia de Matlab tiene un coste que varía en función del uso y productos escogidos. En cambio R es una plataforma colaborativa, de código abierto, es decir, completamente gratis para todos.

Actuación

Cuando se trata de tareas informáticas técnicas, estadísticas y aprendizaje automático Matlab es más rápido que R. Sin embargo, un desarrollador competente en R puede lograr resultados más rápido y mejorar el rendimiento [4].

Funcionalidades

Matlab se usa en diversas aplicaciones como procesamiento de imágenes, manipulación de matrices, aprendizaje automático y procesamiento de señales, mientras que R se usa principalmente para análisis estadísticos y procesamiento de datos [4].

En este TFG se pretende demostrar que R también es compatible con algunas tareas, más concretamente con el procesamiento digital de imágenes.

Soporte y documentación

Matlab tiene un soporte y documentación envidiable, la documentación es posible visualizarla en línea y sin salir del escritorio de Matlab, incluye inmensos ejemplos de código y cuenta con más de 200 expertos, los cuales se dedican al soporte técnico resolviendo dudas y solucionando problemas de la comunidad. En cambio, R es un lenguaje de código abierto, y se ayuda de forma colaborativa entre usuarios. Los desarrolladores son los que aportan el soporte y la documentación, lo cual lo hace un poco menos eficiente.

Visualización

R y Matlab son muy valiosos a la hora de visualizar datos y enseñar los resultados. R consta de tres motores gráficos más importantes : *base, lattice, ggplot2. Lattice* y ggplot2 son muy parecidos funcionalmente y los usuarios eligen entre uno u otro. Los gráficos de base son los predeterminados y los más fáciles de utilizar. Tanto *lattice* como *ggplot2* proporcionan una interfaz más variable proporcionando especificar las variables a trazar, cómo se muestran y las propiedades visuales generales.

A parte de estos tres motores de visualización existen diversas librerías que se han utilizado para la implementación del código, como son imager, EBImage, jpeg, entre otras.

En comparación, Matlab también posibilita el tratamiento de aplicaciones con funciones de GUI, es decir, interfaz gráfica de usuario, permitiendo personalizar el trazado de funciones en 2D y 3D tanto de forma interactiva como programática. Otro aspecto del que consta Matlab es de Simulink, un paquete adicional el cual es un entorno de programación gráfica

para modelar, simular y analizar sistemas dinámicos multidominio. La interfaz principal de Simulink es una herramienta gráfica de diagramación de bloques.

Sistema operativo (OS)

Ambos funcionan en los tres sistemas operativos (SO) de consumo, es decir, Windows, Linux y Mac. R además es igual independientemente de la plataforma. Hay pruebas en CRAN que aseguran que los paquetes funcionan correctamente sin diferencias respecto al SO en el que se trabaje.

Una ventaja de R frente a Matlab es que no tiene ninguna versión con licencia, por consiguiente, es más fácil la interoperabilidad de un programa, respecto a distintos ordenadores, por lo que no sería relevante la versión de R instalada. En cambio, en Matlab sí se pueden encontrar diferencias a la hora de ejecutar una aplicación entre una licencia y otra.

Interfaces con otros idiomas

Matlab proporciona una integración flexible de dos vías con otros lenguajes de programación, lo que significa:

- Puede llamar a Matlab desde otro idioma.
- Desde Matlab puede llamar a bibliotecas escritas en otros lenguajes de programación.
- El código Matlab se puede convertir a código C / C ++.

Del mismo modo, las interfaces R se han desarrollado para varios idiomas. Puede llamar al código escrito en otros idiomas desde R y el código en R puede ser llamado desde otros idiomas. Asimismo con C / C ++ desde R y a R desde Python [4].

Por último, también hay diversas formas de interoperabilidad entre Matlab y R, estas se comentaran en profundidad más tarde.

3.4. Motivación

OTOLAB fue desarrollado originalmente en Matlab por ser este el lenguaje más común en la ingeniería. Sin embargo, hay diversos aspectos que fundamentan la utilidad de convertir este código en R, que se explicara más detalladamente en esta sección.

Una de las principales características que hacen esta traducción deseable es el hecho de que R sea un software de uso libre y gratuito para todos los usuarios. R posee código abierto, lo cual mejor el estudio de muchas funciones, ya que en Matlab no es posible la visualización del código de funciones ya predefinidas y, por consiguiente, es más difícil crear diversas versiones de dichas funciones.

Otro aspecto para solventar es cierta inestabilidad de las aplicaciones desarrolladas en Matlab que contienen interfaces gráficos de usuario (GUI) diferentes según la versión del programa y arquitectura del ordenador, que dificultan su extensión con nuevas aplicaciones y su uso.

Además cada día se suman universidades y centros de investigación científica, incluyendo las ciencias médicas que utilizan R en su día a día, demostrando sus capacidades y fiabili-

dad para el análisis estadístico y tratamiento de datos frente a soluciones privativas [1]. En los últimos años, el uso de R ha ido creciendo enérgicamente, como se puede comprobar en un artículo escrito por *mappingGis* (web especialista en formación GIS online) en la cual mediante el uso de una herramienta llamada *Stack Overflow Trends tool*, en la que se fundamentan de las preguntas mensuales que se hacen en la conocida web *Stack Overflow* sobre los diversos lenguajes de programación, podemos intuir su uso. Se adjunta la gráfica de la figura 3.2 que se visualiza a continuación, en la que se ve un claro crecimiento del uso de R en los últimos años. También se ve una comparativa con el uso de otros programas como panda, tensorflow y Matlab, en el que se ve claramente que, según este estudio, R ha tenido un crecimiento mayor en su uso respecto a Matlab [10].

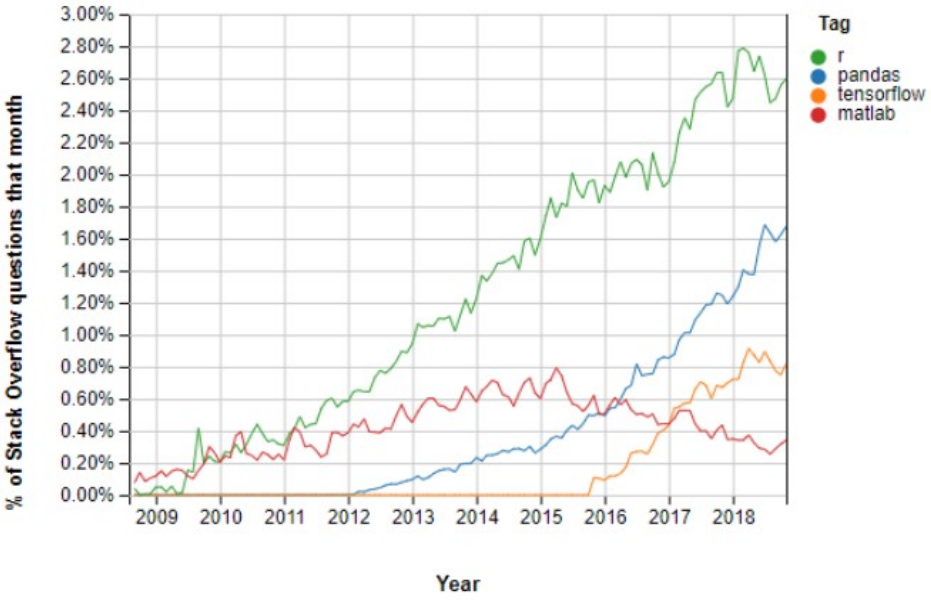


Figura 3.2: Gráfica del uso de distintos lenguajes de programación [10].

CAPÍTULO 4

Interoperabilidad entre Matlab y R

Tras una búsqueda exhaustiva de información, se definen distintas formas de interoperabilidad entre Matlab y R, explicando una descripción de cada una de ellas, su instalación y por último, las pruebas que se han realizado para ver la viabilidad de cada una de estas opciones.

4.1. matlabR

El primer paquete es matlabr, el cual es principalmente un paquete para llamar a Matlab desde R.

Está compuesto por distintos comandos, que son los siguientes:

- *add_path*: Crea una ruta para añadir a la ruta de Matlab.
- *get_matlab*: Intenta encontrar la ruta de Matlab.
- *have_matlab*: Envía un booleano igual a *true* o *false*, dependiendo si Matlab es accesible desde R.
- *rmat_to_matlab_mat*: Convierte una matriz de R a Matlab.
- *run_matlab_code*: Esta función ejecuta el código de Matlab.
- *rvec_to_matlab*: Esta función prueba un carácter de R y devuelve su estado/valor.
- *rvec_to_matlabcell*: Convierte el vector en R, en una celda de Matlab.
- *rvec_to_matlabclist*: Su funcionalidad es tomar un vector en R y convertirlo en una lista de celdas [11].

Como el principal interés es traducir de Matlab a R, y no viceversa, hay funciones de este paquete que no interesan como por ejemplo las tres últimas: *rvec_to_matlab*, *rvec_to_matlabcell*, *rvec_to_matlabclist*, y también *rmat_to_matlab_mat* , ya que traduce la matriz de R a Matlab y no a la inversa. En cambio, sí resulta interesante el estudio de las demás funciones ya que permiten una conexión entre Matlab y RStudio.

Instalación del paquete

matlabr se encuentra en *GitHub*, una forma de instalarlo es con el siguiente comando desde R:

»devtools::install_github("muschellij2/matlabr")

El siguiente paso se realiza solo en el caso de que Matlab no esté en su directorio habitual, o si se está utilizando un sistema basado en GUI, como sería el caso de RStudio. Para ello se utiliza el comando *options* y se detalla el directorio donde se encuentra Matlab.

» library(matlabr)

»options(matlab.path=('/ProgramData/Microsoft/Windows/Start Menu/Programs/MATLAB R2019b')

Para corroborar la instalación de este paquete, se puede usar un comando de este, *have_matlab()*, el cual devuelve un booleano con *true* si se puede realizar la conexión con Matlab, y *false* si en caso contrario, no se puede producir dicha conexión. También se prueba el comando *get_matlab()* y efectivamente devuelve la ruta de acceso donde se encuentra Matlab.

Pruebas

Con la instalación se ha podido comprobar que efectivamente los tres primeros comandos que definimos (*add_path, have_matlab, get_matlab*) son fáciles de usar y permiten una conexión inmediata.

Ahora se prueba *run_matlab_code* para ver si realmente se puede realizar la conversión de un lenguaje a otro, para ello se hace un ejemplo sencillo como la suma de dos escalares y una matriz. Para ello, se realiza lo siguiente en R:

» code = c("x = 1",

" y=2;", "z=x+y", " a = [1 2 3; 4 5 6; 7 8 9]", "save('test.txt', 'x', 'a', 'z', -ascii')")

» res = run_matlab_code(code)

Efectivamente al introducirlo en R da una ruta donde se encuentra un *script* de Matlab con x, y, z y se abre directamente en Matlab. Para comprobar si el test.txt funciona, se introduce en R el comando *file.exists* y se verifica que la creación ha sido correcta. Por último, se confirma si ha ocurrido bien la conversión leyendo este mismo fichero:

» output = readLines(con = "test.txt") » print(output)

Y la salida en R es la siguiente:

[1] "1.0000000e+00

[2] "1.0000000e+00 2.0000000e+00 3.0000000e+00

[3] "4.0000000e+00 5.0000000e+00 6.0000000e+00

[4] "7.0000000e+00 8.0000000e+00 9.0000000e+00

[5] "3.0000000e+00

En [1] se refiere al valor de x, en [2] se ha encontrado el valor de a, y en [5] el valor de z, por ello se puede concluir que se ha realizado bien la conversión.

4.2. R.matlab

La función principal es llamar a Matlab desde R y así poder utilizar diversos recursos como funciones, *scripts*, variables, entre otros. El funcionamiento se basa en crear un servidor en Matlab y un objeto cliente en RStudio, creando así una conexión cliente-servidor.

Métodos:

- *as.character*: Obtiene una cadena que describe la conexión Matlab actual.
- *close* : Cierra la conexión al servidor Matlab.
- *evaluate* : Evalúa una expresión Matlab.
- *finalize* : Finaliza el objeto si se elimina.
- *getOption* : Obtiene el valor de una opción.
- *getVariable*: Obtiene una o varias variables de Matlab.
- *isOpen*: Comprueba si la conexión al servidor Matlab está abierta.
- *open*: Abrer una conexión con el servidor Matlab.
- *readResult* : Lee los resultados del servidor Matlab.
- *setFunction* : Define una función Matlab.
- *setOption* : Establece el valor de una opción.
- *setVariable*: Determina una o varias variables de Matlab.
- *setVerbose*: Establece el nivel detallado para obtener más detalles sobre el acceso a Matlab.
- *startServer*: Método estático que inicia un servidor Matlab.
- *writeCommand*: Escribe(envía) un comando al servidor Matlab.
- *readMat*: Lee un dato con extensión .mat [6].

Instalación del paquete

Para instalarlo se tiene que ejecutar un *script* llamado *MatlabServer.m* que inicia un "servidor"minimalista de Matlab. Cuando se inicia el servidor escucha las conexiones del número de puerto especificado por la variable de entorno 'MATLABSERVER_PORT'.

Para diferenciar cuando se ha usado R o el servidor de Matlab se distinguirá por colores usando el azul para R y el verde para el servidor de Matlab. El siguiente paso es abrir R e instalar el paquete R.matlab y utilizarlo como librería, seguidamente se inicializará desde RStudio el servidor de Matlab. Estos pasos son respectivamente los siguientes comandos:

» install.package(R.matlab)

» library(R.matlab)

» Matlab$startServer()

Para comunicarse con Matlab se crea un objeto cliente de Matlab y se chequea el estado (en este punto aún debe estar desconectado) de la siguiente forma:

» matlab = Matlab()

» print(matlab)

Por último, para finalizar con la instalación se conecta con el servidor de Matlab, se confirma que dicho servidor esté abierto y funcionando y se corrobora el estado (en este momento, ya debe estar conectado). Cada paso es correspondiente a los sucesivos comandos.

» isOpen = open(matlab)

» if (!isOpen)

» throw("MATLAB server is not running: waited 30 seconds.")

» print(matlab)

En este punto ya se habría conectado Matlab con R.

Pruebas

Se empezará escribiendo una expresión sencilla en el servidor de Matlab, como una suma y una matriz. Y se muestra con Matlab.

» evaluate(matlab, " A=1+2;", "B=ones(2,20);")

» evaluate(matlab, " A")

Una vez se ha escrito en el servidor, se trasladan a R y se ve si se visualiza esa variable con el comando *getvariable*, se guarda en una variable que se ha llamado data y se visualiza data.

» data = getVariable(matlab, "A", "B"))

» data

Efectivamente se traslandan dos constantes (el escalar y la matriz) a R.

A continuación, se prueba a realizar una función sencilla en el servidor de Matlab, para ver si se envía correctamente la información a R. La función consiste en realizar la media y la variación de un número dado. Se tiene que introducir en el servidor:

» setFunction(matlab,

» function [media,desviacion] = estadistica(x)

» n = length(x);

» media = sum(x)/n;

» desviacion = sqrt(sum((x-media).^2/n));

```
» end
» ");
```

Y se visualiza el resultado en R:

```
» x= c(1, 2, 3, 4, 5)
» evaluate(matlab, "[m,d]=estadistica(x);")
» resultado = getVariable(matlab, c("m", "d"))
» print(resultado)
```

Se comprueba tambien que se realiza bien el traspaso de información.

Para acabar, se cierra la conexión con Matlab y se ve su estado, que tendrá que ser apagado.

```
» close(matlab)
» print(matlab)
```

Y en último lugar, se ha dejado una de las funciones más útiles a la hora de realizar el TFG, *readMat*. Se ha utilizado de una forma menos laboriosa que muchas otras de este paquete. Simplemente se guarda el dato obtenido en Matlab en la carpeta donde se esta trabajando en R (se debe comprobar que queda con la extensión .mat) y se abre con este comando poniendo entre paréntesis el nombre del dato. En el siguiente ejemplo se visualiza la lectura de un dato exportado de Matlab llamado *input.mat*.

```
»data=readMat(input.mat)
»input=data[[1]]
```

4.3. matconv

Este paquete proporciona un traductor para el código Matlab / Octave en código R. Proporciona opciones de diferentes estructuras de datos y la conversión de funciones. Con el inconveniente de que para su uso se debe cumplir principalmente con la guía de estilo estándar.

Tiene diversos métodos, que se explican a continuación:

mat2r: Es la función de controlador de máximo nivel. Para una conversión más complicada se pueden usar conversores de funciones (*makeFunctionMap*) o convertidores de datos (*makeDataMap* y *makeSliceMap*), los cuales se pueden añadir al controlador principal, igualmente se explicarán de forma más específica posteriormente. Principalmente esta función genera una lista con el código convertido como rCode y el código original como matCode, lo cual permite ver las diferencias entre los dos y qué está haciendo exactamente el programa. Los argumentos que se pueden añadir son la ruta de archivo del código de Matlab que se quiere convertir, la ruta de archivo donde se desea guardar la solución de esta conversión, los conversores de funciones y/o los conversores de estructuras. Por último, hay un parámetro llamado *Verbose* que da información extra [15].

makeDataMap, y *makeSliceMap*: En ambas su funcionalidad es la conversión de estructuras, señalando el carácter que limita la estructura tanto por la derecha como por la izquierda, el nombre de la estructura en R y el nombre de la estructura en Matlab que debe convertirse. Por ejemplo, en el caso de una matriz, los caracteres delimitante serían por la derecha y por la izquierda corchetes ([ ]) respectivamente, y en esta prueba en ambos lenguajes el nombre de la estructura es *matrix* . La diferencia entre ambas es que *makeDataMap* sería más bien para cualquier tipo de estructura; coge las líneas de código de Matlab y realiza la conversión, en cambio, *makeSliceMap*, se centra más en cadenas (*strings*) y las convierte.

makeFunctionMap: Convierte los argumentos de una función de Matlab. Los valores a introducir para el uso de esta función serán el nombre de la función y el argumento, con el cual se quiere usar la función. Por ejemplo, si nuestra función es para realizar la media y se llama "media", los valores a introducir serán media (ya que es el nombre de la función) y un vector con los valores a los cuales se quieren realizar la media.

Instalación del paquete

En este paquete se señala en azul el código en R, en naranja el código utilizado para realizar la conversión y por último el verde para el código correspondiente a Matlab.

La instalación en este caso es trivial, introduciendo los siguientes comandos:

»install.packages("matconv")

Si se desea obtener una versión más actualizada, se puede instalar directamente la del desarrollador así:

»install.packages("devtools")

devtools::install_github("sidjai/matconv")

Para usarlo solo hay que poner el siguiente comando:

»library(matconv)

También es recomendable utilizar la biblioteca rmarkdown.

Pruebas

Se han realizado diversos ejemplos. Primero se comienza con el controlador principal, mat2r, traduciendo una función que lee, ya que multiplica por 3, pero luego lo divide, lo importante es ver los cambios que se realizan. Hay que destacar que en Matlab hay una distinción entre archivos de *script* y archivos de funciones y se puede ver cómo cambia en R según si es un *script* o una función, también como los objetos se recompilan y como aparece una declaración *return*. Lo último que cabe destacar es el parámetro *varagin* que en Matlab se utiliza para entradas predeterminadas o con longitud desconocida, lo más cercano en R es "..." que se convertirá en una lista como la variable *varagin*

» matIn = c("function [dat] =

+ " didThing = 1*3;",

+ "dat = didThing / 3;",

+ end")

» mat2r(matIn, verbose = 0)

$matCode

[1] “function [dat] = xlsReadPretty(varargin)

[2] “didThing = 1*3;

[3] “dat = didThing / 3;

[4] “end

$rCode

[1] “xlsReadPretty = function(...) {

[2] “tvarargin = list(...)

[3] “didThing = 1*3

[4] “dat = didThing / 3

[5] “treturn(dat)

[6] ”}

Ahora se utiliza el conversor *makeDataMap*, destacando los límites de la matriz, consiguiendo así la traducción de una matriz en Matlab a R.

» dataMap = makeDataMap(”[”, ”]”, ”matrix”)

» dataMap(“A = [1,2,3;4,5,6]”)

[1] “A = matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

A continuación, se prueba con una estructura más compleja, como es el array de celdas de Matlab, se comprueba que también realiza una buena conversión.

» dataMap = makeDataMap(rClass = “list”, matClass =“cell”)

» dataMap(“B= 1, “1.2’; 5, 6, 8”)

[1] “B = list(list(1, “1.2’), list(5, 6, 8))

Se continúa, pasando ahora a lo que se llama en Matlab *structure*, y se intenta obtener un dato de esta estructura, analizando su homólogo en RStudio, utilizando *SliceMap*.

» sliceMap = makeSliceMap(matClass = “structure”, rClass = “list”)

» sliceMap(“resultado = colegio.estudiante.nombre”)

[1] “resultado = colegio[[‘estudiante’]][[‘nombre’]]

Tras diversas pruebas, se puede concluir que todas las funciones de matconv funcionan pero es laborioso escribirlo.

Para finalizar, y tras intentar integrarlo en el TFG para facilitar la traducción del código, se ha llegado a diversas conclusiones sobre estos paquetes. La única función que ha sido beneficiosa en el proyecto, ha sido la función *readMat*, del paquete R.Matlab, ya que en algunas ocasiones, a la hora de crear una función en R, que en Matlab no existiera o ver si

una función predefinida en Matlab, hacía exactamente lo mismo que otra función encontrada en una librería de R. Es conveniente verla con exactamente los mismos datos,por lo que se traspasó la información de Matlab a R. Tambien se utilizó, para trasladar datos en Matlab que se representaran en imágenes y así poder realizar en RStudio, la imagen diferencia. Esta función, es muy fácil de utilizar, guardando los datos obtenidos en Matlab, en la carpeta del proyecto en R, y utilizando el comando *readMat.*.

Respecto a todo lo demás, ha sido descartado, por una sencilla razón, una vez que has aprendido ambos idiomas, es menos laborioso hacer la conversión que escribirlo entre los comandos y asegurarte de que cumple la guía de estilo estándar, ya que solo suplementa cosas básicas. Lo más laborioso del TFG es descubrir que funciones ya predefinidas en Matlab son homólogas a alguna función definida en una librería en CRAN, y estas funciones más complejas o no esenciales de R, por tanto no las traduce.

CAPÍTULO 5

Metodología

Este capítulo se centra en explicar las diversas formas de trabajar que se repitierón constantemente durante el proyecto.

Los aspectos triviales y diferencias entre Matlab y R, se han aprendido siguiendo como manual la referencia [7]. La principal diferencia entre Matlab y R es que en R no se distingue entre filas y columnas, mientras en Matlab esta distinción existe. Un ejemplo sería la diferencia entre escribir una fila y columna en Matlab:

En Matlab una fila sería así:

```
v=[1 2 3 4]
```

Mientras una columna sería:

```
v=[1; 2; 3; 4]
```

Colocando los puntos y comas entre los elementos para el caso de las columnas y sin estos para las filas. Por el contrario, en R, en ambos casos sería:

```
v= c(1,2,3,4)
```

Dicha distinción conlleva a otros cambios, en R al insertar una matriz tienes que poner el número de filas o columnas. Por ejemplo, para definir una matriz con dos filas en R:

```
matrix(c(1,2,3,4,5,6), nrow=2, byrow=TRUE)
```

Y en Matlab sería:

```
[1 2 3 ; 4 5 6]
```

Estos son algunos entre muchos otros cambios que conlleva esta diferente forma de nombrar a matrices y vectores, aparte de las diferentes estructuras que hay en ambos lenguajes, por ejemplo una *struct* en Matlab es lo que se conoce en R como un *list*.

Otra de los aspectos que se han repetido durante todo el proceso es el cambio de paréntesis a corchetes. En Matlab para acceder a un elemento del vector, o acceder a un elemento de una matriz o acceder a un elemento de la matriz conociendo su índice, se utiliza paréntesis, en cambio, para esos usos en R se utiliza un corchete.

Por último, se destaca el uso de bucles y condicionales, ya que ha sido un recurso recurrente, como se muestra en la tabla 5.1:

Diferencias en bucles y condicionales	
Matlab	R

for
command1
command2
end
	for (i in v) {
	command1
	command2
	}

if cond
command1
command2
end
	if (cond) {
	command1
	command2
	}

if cond1
command1
elseif cond2
command2
elseif cond3
command3
else
command4
end
	if (cond) {
	command1
	command2
	} else {
	command3
	command4
	}
	*En R no existe la cláusula elseif

Tabla 5.1: Diferencias en bucles y condicionales.

En aspectos no fundamentales y más específicamente en el análisis digital de imágenes, la tarea ha sido mucho más difícil. El primer paso siempre ha sido examinar las referencias de la documentación de Matlab y el repositorio de CRAN (red de servidores FTP y web que almacena versiones idénticas y actualizadas de código y documentación de R). Si se encontraba alguna función similar o un conjunto de funciones que realizaran lo mismo en ambos lenguajes se usaban y si no, se creaban. Este paso será explicado detalladamente en el siguiente capítulo.

El proceso que se ha utilizado durante todo el proyecto para asegurar que en ambas plataformas se estaba realizando la conversión adecuada, era depurando en ambos lenguajes, para así poder estudiar línea a línea si se estaba realizando el mismo desarrollo.

Otro proceso de validación ha sido realizar pruebas con el mismo *input* o entrada utilizando la portabilidad de Matlab a R con archivos .mat y la librería R.matlab, como se explica en el capítulo anterior.

Una vez que se certifica que funciona con el mismo *input*, trasladado desde Matlab, se realizaba el *input* en R y se compara entre ambos lenguajes las diferencias.

Otro recurso utilizado es importar los datos y dibujar en RStudio la imagen diferencia entre la imagen creada con los datos de R y la imagen basada en los datos importados de Matlab, para así estudiar las diferencias entre las imágenes de las distintas plataformas.

Otra forma de verificar que se ha usado en los casos donde visualizando la imagen no se podía intuir si la función había sido correctamente traducida, se trazaba una matriz

ejemplo y se atendían a los cambios de los elementos de la matriz, que corresponden homólogamente a los píxeles de una imagen, siendo esta una forma más sencilla de visualizar.

Respecto a la forma de trabajar con las imágenes, uno de los principales cambios es leer una imagen, mientras en Matlab se usa un comando para todo tipo de imágenes, *imread*. En R, depende del tipo de imagen que se quiere abrir, en el proyecto, se ha utilizado fundamentalmente *readJPEG* para imágenes JPEG de la librería jpeg. Pero, por ejemplo, si se quisiera abrir un archivo tipo TIFF, hay una librería llamada justo así y se puede usar un comando llamado *readTIFF*.

Asimismo, para mostrar una imagen en Matlab basta con un comando llamado *imshow*.

```
imshow('image.jpg')
```

En cambio, en R, se puede hacer un *plot* , siempre si antes conviertes la imagen en un tipo *cimg* . Un tipo *cimg* es una clase para almacenar imágenes o video / datos hiperespectrales. Las imágenes tienen hasta 4 dimensiones etiquetadas x, y, z, c. X e y son las dimensiones espaciales habituales, z es una dimensión de profundidad (que correspondería al tiempo en una película), y c es una dimensión de color. Por tanto, es un array de 4 dimensiones numérico.

Este preprocesado consiste en utilizar la función *as.cimg* de la librería imager. Hay diversas formas de mostrar una imagen en R, pero se ha utilizado fundamentalmente la explicada anteriormente, que se detalla a continuación:

Estos primeros pasos son solo para obtener la librería necesaria para abrir la imagen, en este caso, es *jpeg* (si el formato es distinto sería otra) y la librería que se necesita para realizar la conversión al formato *cimg*.

```
install.packages("jpeg")
install.packages("imager")
library('jpeg')
library('imager')
```

Es importante que la imagen esté en el directorio donde se está trabajando. Para ello se puede utilizar el comado *setwd*, con el fin de introducir el directorio correcto:

```
setwd("C:/Users/andre/Desktop/OTOLAB_R/OTOLAB )
```

Los siguientes comandos corresponden a leer la imagen, nombrarla (se debe hacer si quieres utilizarla en siguientes pasos), cambiar el formato y graficar la imagen.

```
imagen=readJPEG('image.jpg')
imagen_cimg=as.cimg(imagen)
plot(imagen_cimg)
```

La diferencia a la hora de mostrarlo se visualiza claramente en la figura 5.1. Como se puede observar la figura en Matlab es más fácil de mostrar, y en R, aparece con unos ejes que no interesan en este caso y girada 90º a la derecha.

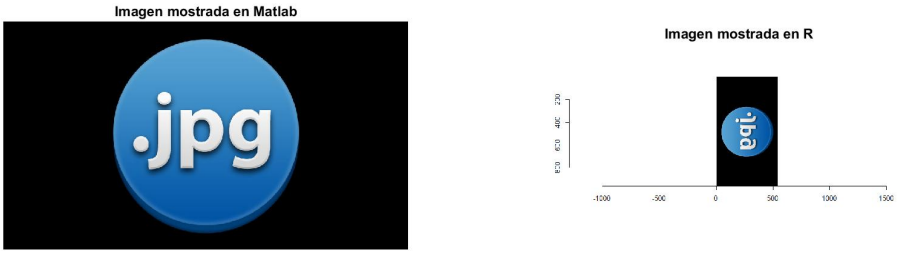


Figura 5.1: Diferencia de Matlab y R mostrando una imagen.

Para igualar la forma de mostrar las imágenes en ambos entornos, se ha utilizado *transpose* en R, del paquete *EBImage* y se han eliminado los ejes. Como ejemplo se corrige lo anterior y se enseña el resultado. Obviamente se debe instalar el paquete y utilizar la librería.

```
install.packages("EBImage")
library('EBImage')
imagen=readJPEG('image.jpg')
```

Se arregla la imagen girada:

```
imagen=transpose(imagen)
imagen_cimg=as.cimg(imagen)
```

Se eliminan los ejes:

```
plot(imagen_cimg,axes=NULL)
```

Figura 5.2: Diferencia de Matlab y R mostrando una imagen tras el preprocesado.

Se puede observar que en Matlab se muestran las imágenes más grandes al guardarlas, lo cual ha sido un inconveniente a la hora de obtener las imágenes diferencia para cuantificar el proceso de validación. Se ha tenido que importar los datos de Matlab a R para poder restarlas con exactamente el mismo tamaño.

Por último, si quieres dividir la figura actual en una cuadrícula de m por n dimensiones, en Matlab se utiliza *subplot* y en R se ha usado el parámetro *par* donde como parámetro debes adjuntar un vector que representa cómo quieres dividir la imagen.

Mientras en Matlab se realiza:

subplot(m,n,p) divide la figura actual en una cuadrícula de m por n y crea ejes en la posición especificada por p.

En R:

par(mfrow=c(m,n)) divide la figura en una cuadrícula m por n.

Respecto a la validación de los datos, el proceso a seguir fue el siguiente: De cada función que se tenía en Matlab, se crea la homóloga en R y para cada una de estas funciones se hizo un *script* tanto en Matlab como en R que mostraba los principales resultados resaltando la función de cada una, así pudimos comprobar que realizaban las mismas tareas. Este proceso se explicará detalladamente en el capítulo 7.

El siguiente capítulo se resume la comparativa en análisis digital de imágenes en ambos lenguajes, se estudia más a fondo que funciones existen parecidas en ambos lenguajes o que unión y/o modificaciones de estas se han tenido que realizar, del mismo modo que también se detalla en qué funciones no se ha encontrado el resultado esperado y se ha procedido a crearlas.

CAPÍTULO 6

Evaluación

A continuación, se explican las distintas funciones de análisis de imágenes utilizas en OTOLAB, detallando si hay una conversión directa. Y las diferencias entre estas, o en caso contrario, si no existe su homóloga y ha tenido que ser creada. En la tabla 6.1 se puede ver un resumen, de lo que se explicará posteriormente.

Conversión directa de funciones	
Matlab	R
bwlabel	bwlabel
regionprops	computeFeatures
imrotate	imrotate
imerode	erode
imdilate	dilate
strel	makeBrush
im2double	double
graytresh	otsu
bwboundaries	bwlabel+ocontour
Creación de funciones homólogas	
Matlab	Funciones creadas
bwmorph	Borrar_semillas_vecinas
imreconstruct	expandMatrix y reconstructR

Tabla 6.1: Estudio de las funciones homólogas en Matlab y R y creación de las inexistentes.

6.1. Funciones similares en ambos lenguajes

6.1.1. Componentes conectados

En Matlab existe una función que realiza dicha búsqueda con una imagen binaria, la función es bwlabel. Su forma de uso es la siguiente:

L = bwlabel(BW)

L = bwlabel(BW,conn)

[L,n] = bwlabel(...)

Donde los argumentos de entrada son los siguientes:

BW=es la imagen binaria 2D

conn=la conectividad del objeto

n= el número de objetos conectados encontrados

Y el argumento de salida:

L=Matriz de etiquetas de regiones contiguas, del mismo tamaño que BW.

En cambio en R se ha encontrado una función que realiza la misma tarea, de una librería llamada 'EBImage', y funciona igual que en Matlab. Se llama también *bwlabel*. En esta ocasión, no se puede elegir la conectividad (esto sería una restricción) tampoco devuelve el número de objetos conectados. Pero, al utilizarla, se descubre que, realizando el máximo de la matriz de etiquetas de regiones contiguas, ósea, del resultado, se obtiene el número de objetos conectados. Con los mismos argumentos que se ha descrito antes sería de la siguiente forma:

L=bwlabel(BW)

n=max(L)

A continuación, se muestra un ejemplo con el que se explica mejor el concepto de componentes conectados. Se toma una imagen inicial y se aplica la conectividad 4, la 8 y la diferencia entre estas. Por último se hace la conversión de los resultados a escala de grises para una más fácil visualización.

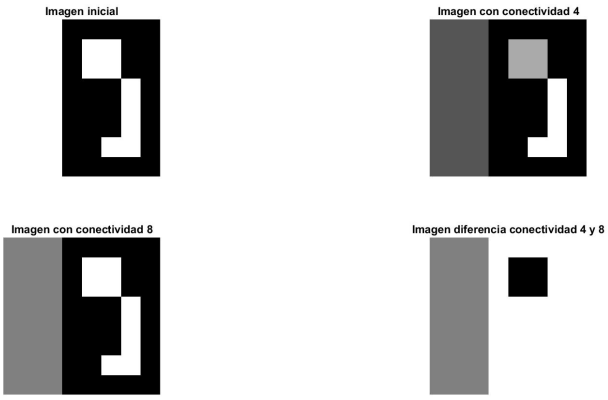


Figura 6.1: Ejemplo de componentes conectados.

Se puede observar como con conectividad 4, se visualiza el cuadrado, la forma de 'L' y el rectángulo de la izquierda de distintos colores, por lo que el número de objetos conectados serían 3, en cambio con conectividad 8, el cuadrado y la 'L' están ambos con el mismo color (en blanco), lo que significa que solo hay dos objetos conectados, el conjunto del cuadrado y la 'L' y el rectángulo de la izquierda. Esto es debido a la diferencia entre conectividad 4 y 8. Con conectividad 4, los píxeles están conectados si sus bordes se tocan, es decir se conectan a lo largo de la dirección horizontal o vertical, en cambio con conectividad 8

se incluyen a los píxeles conectados también las esquinas, por tanto están conectados en dirección horizontal, vertical y también diagonal.

Por último, se observa la imagen diferencia, claramente está formada por el cuadrado que tienen valores de píxeles diferentes (se pueden diferenciar disitintos valores de grises) y por el rectángulo de la izquierda, la forma de 'L' tenían los mismos valores de píxeles en ambos casos, por esta razón no se visualiza en la diferencia entre conectividades.

6.1.2. Descriptores

Tanto en Matlab como en R, hay una función que miden propiedades de la imagen. En Matlab es llamada *regionprops* y en R *computeFeatures*.

En Matlab *regionprops* se utiliza de la siguiente forma:

stats = regionprops(BW,properties)

El argumento de entrada es BW, la imagen binaria. Y hay diversas propiedades que se pueden adquirir (Tabla 6.2), las que se han utilizado en el TFG son las siguientes: *Area, Circularity, Eccentricity, MajorAxisLength , MinorAxisLength, Orientation,* y *Perimeter*.

Propiedades *regionprops*
Area
Perimeter
BoundingBox
Centroid
ConvexArea
ConvexHull
ConvexImage
Circularity
Eccentricity
EquivDiameter
EulerNumber
Extent
Extrema
FilledArea
FilledImage
Image
MaxFeretProperties
MinFeretProperties
MajorAxisLength
MinorAxisLength
Orientation
PixelIdxList
PixelList
Solidity
SubarrayIdx

Tabla 6.2: Propiedades de *regionprops*.

En R, como se menciona anteriormente también hay una función que mide las propiedades de una imagen, es de la misma librería que *bwlabel*, 'EBImage'. Esta librería ha sido muy útil en la realización del proyecto, dicha función se llama *computeFeatures* y las propiedades que engloba esta función se visualizan en tabla 6.3.

Propiedades de *computeFeatures*
mean
sd (standard deviation intensity)
mad
q* (quantile intensity)
computeFeatures.shape
Area
Perimeter
Radius.mean
Radius.sd
Radius.max
Radius.min
computeFeatures.moment
Cx (center of mass x)
Cy (center of mass y)
Majoraxis
Eccentricity
Theta

Tabla 6.3: Propiedades de *computeFeatures*.

En la Tabla 6.4 se pueden ver las funciones comunes en ambos lenguajes.

Descriptores comunes en ambos lenguajes.
Area
Perimeter
Eccentricity
MajorAxisLength
Orientation o Theta

Tabla 6.4: Propiedades de *regionprops*.

Algunas de las propiedades en común son típicos parámetros de una elipse, como la longitud del mayor eje y del menor (esta última solo se puede obtener con Matlab), que se refiere a la longitud (en píxeles) del eje principal o menor de la elipse respectivamente, la excentricidad que es la relación de la distancia entre los focos de la elipse y su longitud del eje principal, o la orientación o theta (en R) refiriéndose al ángulo entre el eje y el eje principal de la elipse.

Por otro lado, existen propiedades que aparecen en alguno de los lenguajes, pero no en ambos. Por ejemplo en Matlab, *ConvexArea* define al área del casco convexo de la región. El casco convexo de una región es la región más pequeña que cumple dos condiciones: (1) es convexa (2) contiene la región original y todas las que tienen que ver con la convergencia. Otro ejemplo sería filledImage que devuelve una imagen con los huecos rellenos, entre otras. En R, también hay parámetros que son inexistentes en Matlab como el radio mayor

y menor del objeto, para detectar la forma o la desviación estándar del radio medio y demás.

En ambos lenguajes te devuelve un parámetro de salida con las mediciones, y el parámetro de entrada es la imagen binaria otra vez. La diferencia es que, en R, no se puede elegir la propiedad, el parámetro de salida te da una lista con las propiedades de la función y debes acceder a ella según la posición.

En R hay distintas versiones, las que se han trabajado son *computeFeatures.shape*, y las propiedades que han sido interesantes son el área, y el perímetro. Un ejemplo de uso teniendo en cuenta que en *computeFeatures* la primera propiedad de la lista es el área es el que se expresa a continuación:

resultado=computeFeatures(BW)

área=resultado[1]

La otra versión que se ha aplicado, es *computeFeatures.moment*, de esta se obtuvo el eje mayor, la excentricidad y el centro de masa x e y.

El centro de masa vertical y horizontal no se ha utilizado en Matlab, ya que se ha usado porque son parámetros necesarios para usar *imrotate* en R.

También se ha usado *theta*, que es la orientación en Matlab. Se debe tener en cuenta que en R viene en radianes y en Matlab en grados, por lo que es necesaria la conversión.

Por último, cabe destacar que en R no se pueden obtener tantas propiedades como en la función predefinida en Matlab, como la circularidad y el eje menor, pero se encontró una forma sencilla de abarcar este problema ya que estas son dependientes de otras propiedades que si se pueden obtener. La circularidad y el eje menor, se calcularon como:

$$Circularity = \frac{4 * \pi * area}{perimeters^2} \tag{6.1}$$

$$MinorAxis = \sqrt{(eccentrity^2 - 1) * mayoraxis^2} \tag{6.2}$$

Un ejemplo del uso de los descriptores, se puede ver en el capítulo 7, más en detalle en las figuras 7.16 y 7.17 donde se usan estas funciones tanto en Matlab como en R, y se aprecian los mismos resultados en una larga lista de propiedades.

6.1.3. Rotación

Acerca de rotar una imagen, en Matlab es trivial, con el comando *imrotate*:

J = imrotate(I,angle)

J = imrotate(I,angle,method)

J = imrotate(I,angle,method,bbox)

Siendo:

I= una imagen

angle= ángulo

method= método de interpolación. Puede ser *nearest, bilinear o bicubic*, *bbox* se refiere a un cuadro delimitador que define el tamaño de la imagen de salida. Eligiendo entre crop, si se requiere que la imagen de salida sea igual a la imagen de entrada, recortando si hace falta o *loose*, la cual hace que la imagen de salida sea lo suficientemente grande como para contener toda la imagen rotada, con esta opción la imagen rotada es más grande que la original.

J se refiere al parámetro de salida, es decir, la imagen rotada.

En el trabajo, el ángulo utilizado, es la orientación recibida a través de *regionprops*. En R

En cambio, en R, se descubrió en una librería, que también se utilizó otras funciones, se llama 'Imager' y los parámetros de entrada son distintos.

J=imrotate(im, angle, cx, cy, interpolation = 1L, boundary = 0L)

El ángulo, de forma homóloga a Matlab, se obtiene gracias a *computeFeatures.moment* y seleccionando la orientación, el centro de x y de y se obtiene también por la función mencionada anteriormente . En interpolación se pueden elegir 0L, 1L y 2L siendo *nearest, linear* y *cubic* respectivamente, Y por último, las condiciones de los límites se pueden seleccionar también 0L, 1L y 2L que corresponde a *dirichlet, neumann* y *periodic*.

6.1.4. Morfología matemática

Elemento estructurante

La idea principal de la morfología binaria es examinar una imagen con una forma predefinida simple concluyendo sobre cómo esta forma encaja o no con las formas propias de la imagen. Esta simple 'sonda' es una imagen binaria, la cual es llamada elemento estructurante, es decir, un subconjunto del espacio o de la cuadrícula.

En Matlab, hay una función que crea el elemento estructurador morfológico que se llama *strel*, en el que el primer parámetro es la forma que puede adquiri *square, line, diamond, rectangle, octagon* entre otras y el segundo es el radio para definir el tamaño.

Del mismo modo, en R, en el paquete 'Morphology', se encuentra una función que llamada *makeBrush*, es la que realiza el elemento estructurante, tiene también como primer parámetro la forma que se puede elegir entre *box, disc, diamond, Gaussian, line,* y tres parámetros más que son *step* (indica si el pincel es binario), *sigma* (desviación estándar de la gaussiana), *angle* (ángulo), por defecto son respectivamente TRUE, 0.3 y 45.

Seguidamente se muestran ejemplos de elementos estructurantes creados en Matlab y en R.

Figura 6.2: Ejemplo de elemento estructurante creado con R.

Figura 6.3: Ejemplo de elemento estructurante creado con Matlab.

Erosionar, dilatar, apertura y cierre

Erosionar es una técnica utilizada en imagen digital para quitar los pixeles que no pertenecen al objeto, se toma un píxel de prueba (píxel estudio) y se van probando uno a uno, para ello si todos los pixeles vecinos al píxel de estudio pertenecen al objeto, entonces el píxel de estudio también pertenece al objeto. (Si alguno de los pixeles vecinos al píxel de estudio no pertenece al objeto entonces ese píxel de estudio tampoco pertenece al nuevo objeto).

Dilatar, al contrario, se utiliza si se quedan pixeles o pequeños grupos de ellos que se intuyen que deberían pertenecer al objeto pero que por diferentes motivos no quedaron identificados. Se incorporan al objeto, si alguno de los pixeles vecinos al píxel en estudio pertenece al objeto entonces el píxel de estudio también pertenece al objeto.

Apertura (*opening*) es una erosión seguida de una dilatación.

Cierre (*closing*) es una dilatación seguida de una erosión.

En Matlab, lenguaje inicial de OTOLAB, se realiza con la función *bwmorph*, ya que esta función tiene como utilidad realizar operaciones morfológicas, es un algoritmo más rápido para imágenes binarias y el elemento estructurante sería de tipo ones(3,3), creando el usuario la matriz. En la documentación de Matlab, se descubre que también se podría hacer con una función determinada *imerode*, e *imdilate* para dilatar, usando estas, es un algoritmo más lento que permite elegir el elemento estructurante mediante *strel* como en R con su función homóloga *makeBrush* y funciona con niveles de grises también. Esta versión es más parecida a R.

Erosionar con *bwmorph* se utiliza como en otras ocasiones el parámetro BW correspondiente a la imagen binaria y en operaciones deberemos poner *erode* y el número de veces a realizar, en este caso 1.

Para la abertura se utiliza una función llamada *imopen* con dos parámetros de entrada, la imagen y el elemento estructurante.

Para realizar la conversión a R, esta vez se encuentra la solución de nuevo en el paquete 'EBImage', más concretamente en 'Morphology'. La función utilizada es *erode*. Los parámetros de entrada son dos, la imagen binaria, y lo que se denomina *kern*, lo cual corresponde a una imagen objeto o una matriz que contiene el elemento estructurante. De la misma forma se haría para dilatar con la función *dilate*. Y, asimismo, para abertura o cierre que las funciones pertencen también a este paquete y sus nombres son *opening* y *closing* respectivamente.

A la hora de obtener el 'kern correcto' se utiliza nuevamente 'Morphology', específicamente

una función llamada *makeBrush*, como se explica anteriormente.

Como hay que fusionar dos funciones, a continuación, se muestra un ejemplo ilustrativo:

kern=makeBrush(size, shape=c('box', 'disc', 'diamond', 'Gaussian', 'line'), step=TRUE, sigma=0.3, angle=45)

J=erode(x, kern)

Siendo:

J= imagen erosionada

X= imagen binaria inicial

kern= elemento estructurante

La función *makeBrush*, para realizar un elemento estructurante será útil en diversas partes del proyecto.

Para ilustrar la morfología matemática, se visualiza la siguiente figura con la imagen inicial, el elemento estructurante, la imagen despues de erosionarla y, por último, la imagen dilatada.

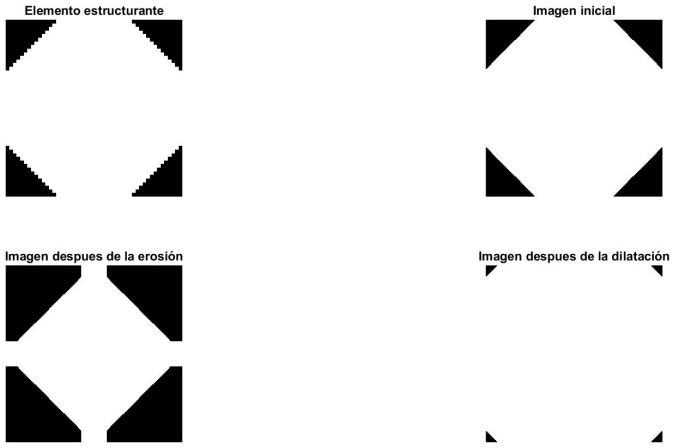


Figura 6.4: Ejemplo de morfología matemática.

6.1.5. Blanco y negro

Para convertir una imagen RGB o colores a escala de grises, en ambos lenguajes es muy sencillo.

Mientras en Matlab es I=rgb2gray(RGB). En R, la función se encuentra en un paquete llamado 'Imager', el cual se usa varias veces en este proyecto, la función escogida se llama *grayscale*.

BW= grayscale(RGB, method = "Luma", drop = TRUE)

RGB es la imagen en color, sobre el parámetro *method* se ha usado Luma, que realiza una luminancia aproximada, pero también es posible "XYZ", en cuyo caso se supone que la imagen está en el espacio de color sRGB y se usa la luminancia CIE. El tercer parámetro,

en este caso se asigna igual a *TRUE*, para que devuelva una imagen con un solo canal, de lo contrario, se mantienen los tres canales.

6.1.6. Imagen binaria

Una imagen binaria es una imagen en la que se pueden distinguir regiones blancas o negras, no hay escalas de grises. Mientras en Matlab hay una función llamada imbinarize, en R no se encuentra ninguna en ningún paquete, pero se realiza redondeando la imagen en escala de grises de la siguiente forma:

ImagenBinaria= round(imgray/max(imgray),0)

6.1.7. Conversión de tipos de imágenes

Para convertir la imagen en *double* precisión, es decir, escalar la salida de los tipos de datos enteros al rango [0, 1].

En Matlab se utiliza *im2double* y la imagen como parámetro de entrada y en R su homólogo es de un paquete llamado *base*, y la función es *as.double* y la imagen en blanco y negro como parámetro de entrada. Ambas obtienen como salida la imagen escalada.

De igual modo, hay más formatos para convertir como el uso de *cimg* en R, explicado en el capítulo anterior.

6.1.8. Segmentación

Calcula un umbral global a partir de la imagen en escala de grises utilizando el método de Otsu. El método de Otsu elige un umbral que minimiza la varianza intraclase de los píxeles en blanco y negro con umbral [17].

Matlab utiliza una función llamada *graytresh* con parámetro de entrada la imagen en escala de grises, y la salida el umbral.

En R se encuentra en 'EBImage', otsu, que realiza la misma función introduciendo el mismo parámetro de entrada.

6.1.9. Contorno

Específicamente rastrea los límites de la región en la imagen binaria.

Matlab utiliza la función *bwboundaries*, a la cual le introduces la imagen en blanco y negro y también le puedes añadir otros parámetros como la conectividad y *holes* o *noholes*, si quieres permitir agujeros o no.

En R no hay una función que rastreé los límites con cierta conectividad, por lo tanto se utiliza *bwlabel* (explicada anteriormente para los componentes conectados) y luego el resultado es el parámetro de salida de una función llamada *ocontour* perteneciente al paquete 'EBImage', introduciendole el resultado de *bwlabel* como parámetro de entrada.

Ambos procesos devuelven una matriz de celdas que muestran el contorno.

6.2. Funciones creadas

6.2.1. Borrar semillas vecinas

En Matlab se trata de una operación morfológica llamada 'shrink'. El uso de operaciones morfológicas se realiza del siguiente modo:

BW2 = bwmorph(BW,operation)

BW2 = bwmorph(BW,operation,n)

Siendo:

BW=imagen binaria

operation= operaciones morfológicas a realizar. Hay distintas como: *bothat* (dilatación seguida de erosión), *branchpoints* (puntos de rama de esqueleto), *clean* (quita los píxeles aislados), entre otras.

n= Número de veces que se realiza la operación

BW2= imagen binaria tras realizar la operación morfológicas n veces.

En este caso se va a estudiar *bwmorph*(S, 'shrink', Inf), siendo S la imagen binaria, *shrink*, la operación morfológica y se realiza infinitas veces. Su función es eliminar los píxeles para que los objetos sin agujeros se reduzcan a un punto y los objetos con agujeros se reduzcan a un anillo conectado a medio camino entre cada agujero y el límite exterior.

En R existe 'Morphology' para realizar operaciones morfológicas como se explica anteriormente, pero no se puede realizar *shrink* y al no encontrar ninguna similar en CRAN, se decide crear una función a la que nombramos como Borrar_semillas_vecinas. Esta función elimina los píxeles para que los objetos sin agujeros se reduzcan a un punto. Consta de un parámetro de entrada que es la imagen binaria, y un parámetro de salida que corresponde a la imagen binaria transformada. Para ello se hizó dos bucles anidados y se fue cambiando la primera fila, la esquina superior izquierda y más tarde la derecha y por último la primera fila general, se repitieron estos pasos con la última fila. Se realizó lo homólogo con las columnas y para acabar el caso general. El código se puede ver en el anexo A, listing A.1.

A continuación se realiza un ejemplo ilustrativo, mostrando una matriz de '0s' y '1s' y la matriz después de borrar_semillas_vecinas.

Figura 6.5: Ejemplo de borrar_semillas_vecinas.

6.2.2. Reconstrucción morfológica

La reconstrucción es muy utilizada y conocida en el procesamiento de imágenes binarias, donde simplemente se extraen los componentes conectadas de una imagen a partir de una imagen marcador. Este operador también puede ser definido en imágenes con niveles de grises, pudiéndose utilizar en distintas etapas del procesamiento de imágenes como filtrado, segmentación o extracción de características.

Más en profundidad, la reconstrucción morfológica se puede considerar conceptualmente como dilataciones repetidas de una imagen hasta que el contorno de la imagen marcador se ajusta bajo una segunda imagen, llamada imagen de máscara. En la reconstrucción morfológica los picos de la imagen del marcador se extienden o se dilatan. Cuando se habla de picos se refiere a los píxeles de alta intesidad (es decir, a los elementos más grandes de una matriz), del mismo modo se llaman valles a los píxeles de mínima intensidad (elementos menores de una matriz).

La reconstrucción morfológica tiene diversos usos dependiendo del marcador y máscara que se utilicen, algunos ejemplos son:

- Eliminación de objetos. Los marcadores se eligen de forma específica de manera que se eliminen los detalles que no se deseen. Mas tarde se realiza la reconstrucción por dilatación. El resultado consigue la preservación de las formas de los objetos que han superado al marcado. Nótese que en la apertura clásica este efecto no se conseguía.
- Eliminación de objetos que tocan el borde. Eligiendo como marcador la intersección del marco de la imagen con la imagen y aplicando reconstrucción por dilatación en la imagen.
- Identificación de líneas a través de reconstrucción por aperturas con elementos estructurantes lineales.

Matlab realiza la reconstrucción morfológica colocando el marcador debajo de la máscara. Por ello se puede considerar conceptualmente como dilataciones repetidas del marcador, hasta que el contorno de la imagen marcador encaje debajo de una segunda imagen, llamada imagen de máscara. Los elementos del marcador deben ser menores o iguales que los elementos correspondientes de la máscara. Si los valores en el marcador son mayores que los elementos correspondientes en la máscara, entonces se deben reconstruir los pixeles al nivel de la máscara antes de comenzar el procedimiento. Aparte de los parámetros de entrada máscara o marcador, se puede elegir la conectividad.

Por tanto la máscara se realiza aplicando *adapthisteq* a la imagen, para así realizar una ecualización de histograma adaptativo limitada por contraste (CLAHE). Sobre el marcador, se identifican objetos de alta intensidad en la imagen utilizando la erosión morfológica, para ello se forma un elemento estructurante de tamaño 5 y con forma de disco con *strel* y luego se realiza una erosión con *imerode* con la máscara y este elemento estructurante.

La función principal de CLAHE es diferenciar los picos altos en el histograma ya que normalmente son causados por regiones casi uniformes. Se impone un máximo de picos del histograma, lo que limita la cantidad de mejora de contraste y como consecuencia mejora el ruido.

Este proceso está basado en el algoritmo de Vincent [17].

Respecto a su uso en RStudio no se encontró en ninguna librería una función que realizará una reconstrucción morfológica, por lo tanto, se decidió realizarla usando el artículo de Vincent y su código en C.

Para ello programamos dos funciones. 'ExpandMatrix' y 'reconstruct' tanto en Matlab como en R y diferentes *scripts* para probarlas en ambos programas.

El pseudocódigo en el que se ha basado para realizar la propia reconstrucción morfológica es el que aparece en el artículo de Vincent:

Algoritmo 1 Pseudocódigo del algoritmo de Vincent [17].

- I:imagen de máscara (binaria o escala de grises)
- J:imagen de marcador, definida en el dominio $D_I, J \leq I$.
  La reconstrucción se determina directamente en J
- Escanear D_I en orden de trama:
  - Dejar p como el pixel actual;
- *J(p)* =*(max* $\{J(q), q \in N_G^+(p) \cup \{p\}\}) \wedge I(p)$
- Escanear D_I en orden anti-trama :
  - Dejar p como el pixel actual;
  - *J(p)* =*(max* $\{J(q), q \in N_G^-(p) \cup \{p\}\}) \wedge I(p)$
  - Si existe q $\in N_G^-(p)$ tal que J(q)< J(p) and J(q)<I(q)
    - *fifo_add(p)*
- Paso de propagación: mientras *fifo_empty() = false*
  - *p=fifo_first*
  - Por cada pixel q $\in N_G(p)$:
    - *J(q)<J(p) y I(q)≠ J(q)*
    - *J(q)=min*$\{J(p), I(q)\}$
    - *fifo_add(q)*

Primero se explicará *expandMatrix*, cuya función es expandir la matriz para poder trabajar con los bordes de la imagen en la reconstrucción morfológica. Es simplemente un preprocesado de la imagen, para poder trabajar con ella y así poder realizar la conectividad en los bordes. Primero se trabajó con las cuatro esquinas de la imagen y más tarde con el resto, creando una matriz de dimensión fila y columna un número mayor. Igualmente, esta función se puede visualizar en el apéndice A, listing A.2.

Reconstruct está basado en el pseudocódigo del algoritmo 1 , siendo *raster* y *antiraster* la forma de recorrer la matriz y NG^+ y NG^- representa a los pixeles rojos y azules, como se ha ilustrado en la Figura 6.6. Por último fifo es como la cola o lo que se conoce en programación como *tail*. También se puede ver el código en el anexo A, listing A.3.

Para las pruebas se utiliza en ambos entornos una imagen propia de Matlab, ya que es más fácil visualizar el resultado.

La primera prueba que se realizó fue comprobar si tras utilizar las dos funciones en la versión de Matlab, tanto el preprocesado con *expandMatrix* y *reconstruct*, es homólogo a utilizar la función propia de Matlab *imreconstruct*.

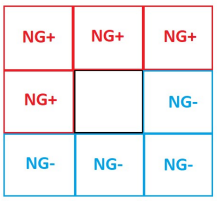


Figura 6.6: Ejemplo ilustrativo del algoritmo de Vincent.

En la figura 6.7 se puede ver como el *imreconstruct* del *toolbox* de Matlab proporciona un resultado muy parecido al obtenido en la función creada, que es un conjunto de las dos funciones explicadas anteriormente en lenguaje m y un *script* llamado TestReconstruct para mostrar los resultados.

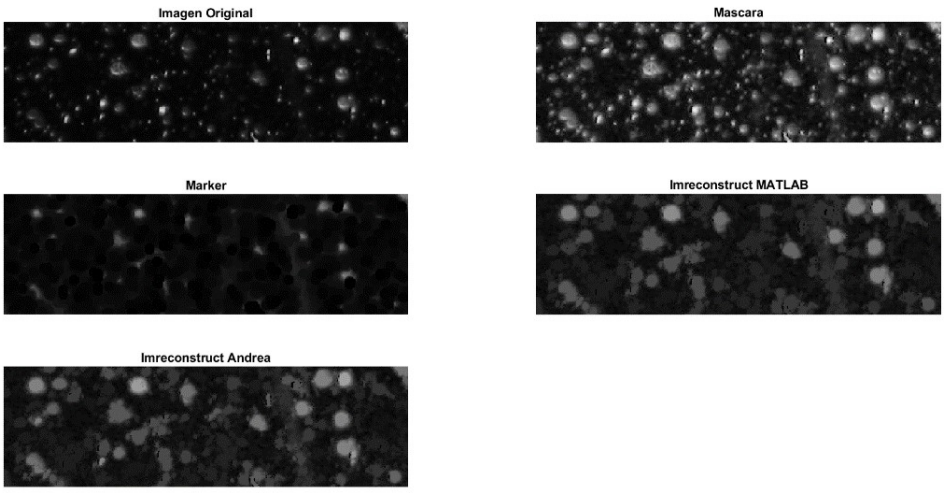


Figura 6.7: Imreconstruct predefinido en Matlab e imreconstruct creado en Matlab.

Una vez probado que el código en Matlab funcionaba correctamente, se realizó el siguiente paso, su traducción a R. Por tanto se dispuso también de estas funciones, pero en el lenguaje R, a parte de dos *scripts* diferentes, ya que se han realizado dos pruebas distintas.

La primera prueba fue importando los datos desde R, como se ha explicado en el capítulo 4 de interoperabilidad, para probar exactamente nuestras funciones (*expandMatrix, reconstructR*) con la misma máscara y marcador que en Matlab y el *script* correspondiente a esta prueba es *reconstructDatosR*. El resultado se visualiza en la figura 6.8.

El código es el mismo que en Matlab pero traducido a R, por lo consiguiente, realiza la misma función. Si no se observa exactamente el mismo resultado, es porque en R se destacan más los blancos, pero la matriz resultado es la misma en ambos lenguajes, fue una de las primeras comprobaciones.

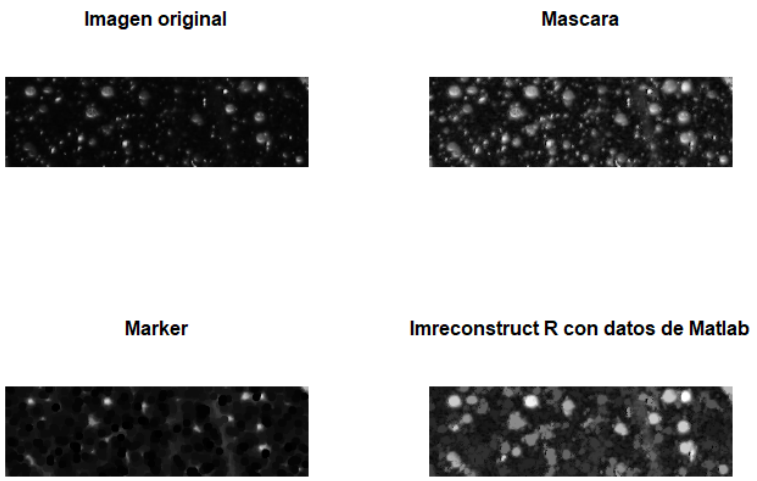


Figura 6.8: Imreconstruct predefinido en Matlab e imreconstruct creado en R.

La siguiente prueba requiere de un nuevo *script* al que se ha llamado reconstructTestDatos, y en esta prueba se realizó la máscara y el marcador, evidentemente lo más parecido posible al proceso trabajado en Matlab. El marcador es más trivial ya que se basa en formar un elemento estructurante y realizar una erosión y ambas funciones están predefinidas en R, como se explica en el apartado anterior 'Evaluación', concretamente en el subapartado 6.1.4 'Morfologia matemática'. En el caso de la máscara, aunque se probaron distintas funciones, ninguna realizaba una ecualización de histograma adaptativo limitada por contraste similar al de Matlab, por tanto, se realizó una ecualización del histograma con una función de R llamada *equalize* y luego se añadió un filtro basado en el histograma adaptativo de contraste, CLAHE.

La imagen resultante es la que se observa en la figura 6.9, en la cual se visualiza también el filtro y la máscara creados en R.

Claramente no es igual ni el marcador ni la máscara, ya que R tiende a resaltar más los blancos y hay variación en la forma en la que funcionan los comandos utilizados en Matlab y R, sin embargo, se puede observar finalmente en la figura 6.10 que son bastante parecidos, también se visualiza que en general hay variaciones al pintar las imágenes, ya que la original tampoco es exactamente igual y es la misma imagen en ambos casos.

Con estas pruebas se concluye que la función *expandMatrix* y *reconstruct* tanto en Matlab como en R son efectivas.

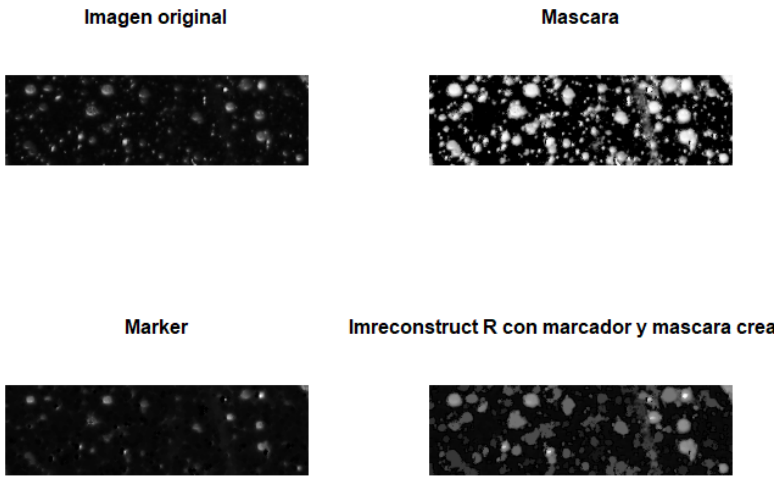


Figura 6.9: Imreconstruct en R con máscara y marcador creados.

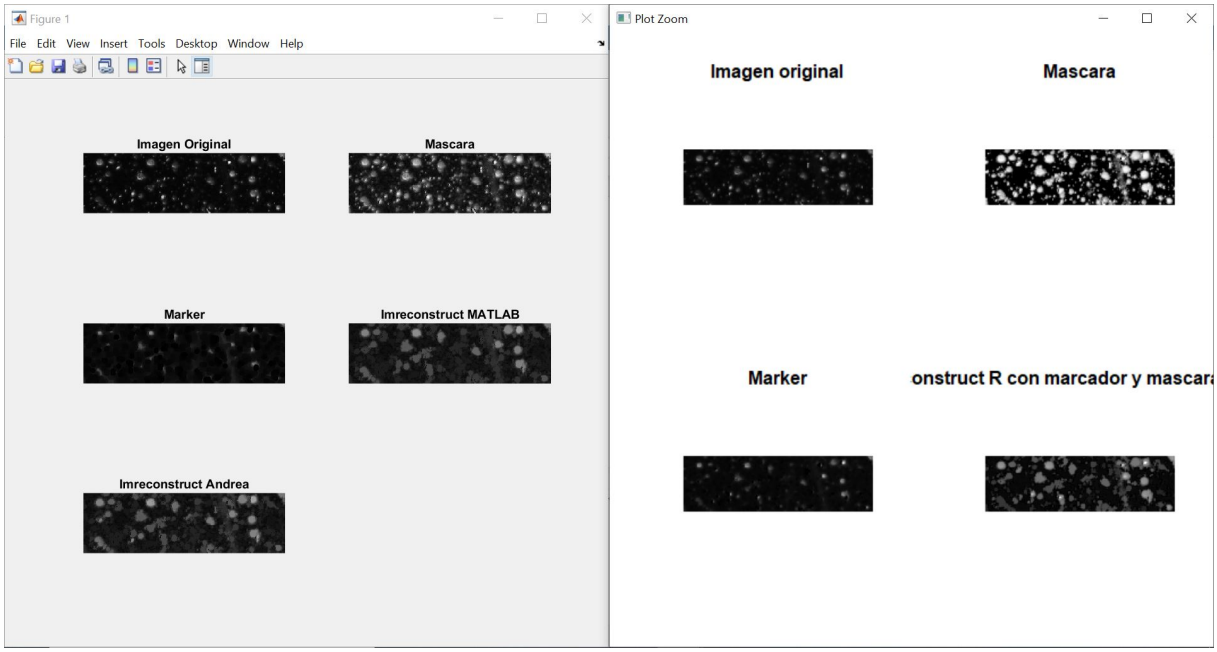


Figura 6.10: Imreconstruct con máscara y marcador creados en Matlab (derecha) y en R (izquierda).

CAPÍTULO 7

Proceso de validación

Las funciones correspondientes al programa OTOLAB, concretamente el módulo de análisis morfométrico a convertir son las que se proceden a explicar en este apartado.

La imagen original en la que se basan todas las pruebas es la siguiente:

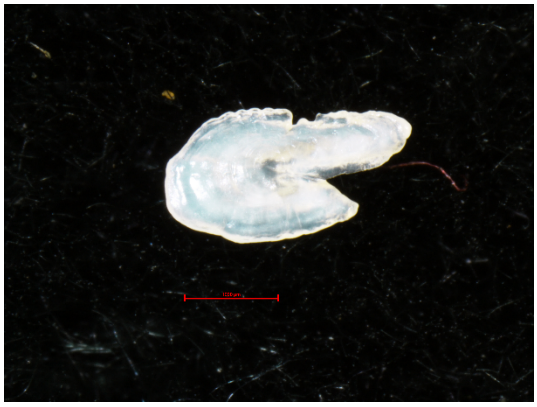


Figura 7.1: Imagen de prueba original de un otolito.

7.1. region grow

Su funcionalidad básica es hacer que la región crezca de forma iterativa, comparando todos los pixeles a los pixeles vecinos no asignados a ninguna región. La diferencia entre el valor de intensidad de un píxel y la media de la región vecina se usa como baremo. El píxel con la menor diferencia medida se asigna a la región.

Este proceso se detiene cuando la diferencia de intensidad entre la media de la región y el nuevo píxel se hace mayor que un cierto umbral que se predefine como parámetro de entrada.

En Matlab se deben introducir como parámetros la imagen, la máscara, y el umbral. Variando el umbral se obtienen distintos resultados, cuanto menor sea el umbral más formas crecen, en cambio si el umbral es muy alto solo pasan las formas más grandes.

La traducción a R de esta función fue de las más complejas ya que consta de las dos funciones que no están disponibles en R y que se han tenido que crear, como son *borrar_semillas_vecinas* e *imreconstruct* ambas explicadas con mayor detalle en el apartado 6.2.

El resultado que se obtiene tras la traducción y realizar un *script* prueba tanto en Matlab como en R, son la figuras 7.2, 7.4 en Matlab y las 7.3 y 7.5 en R. Para comprobar el resultado parece conveniente primero mostrar el resultado de region grow antes de la reconstrucción morfológica en la cual se puede observar mejor las regiones en ambos lenguajes. Luego esta imagen sirve como máscara en *imreconstruct* o en su homólogo como máscara en *reconstructR* y *expandMatrix* en R.

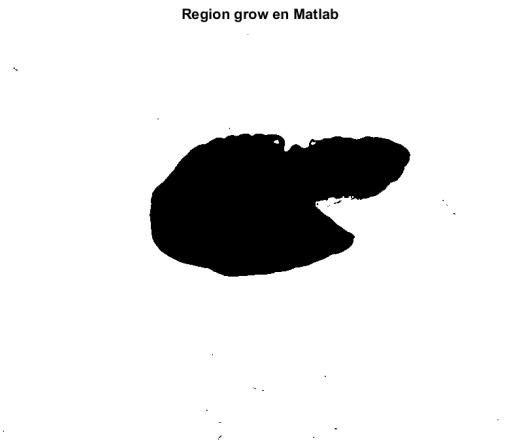


Figura 7.2: Region grow en Matlab.

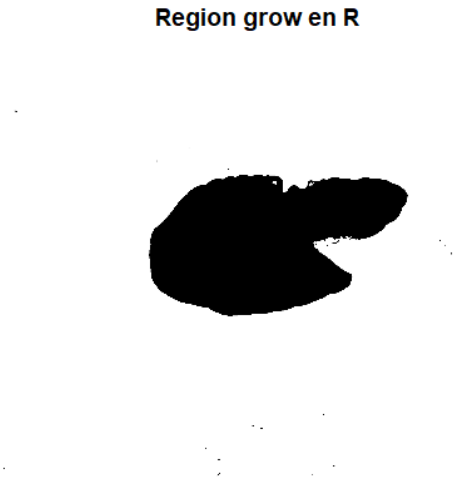


Figura 7.3: Region grow en R.

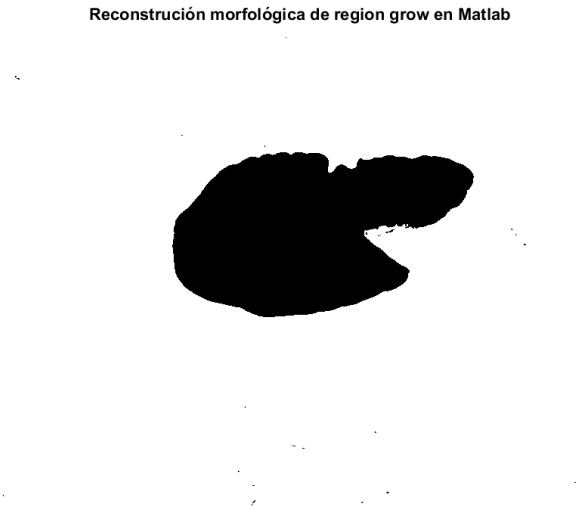


Figura 7.4: Reconstrución en Matlab.

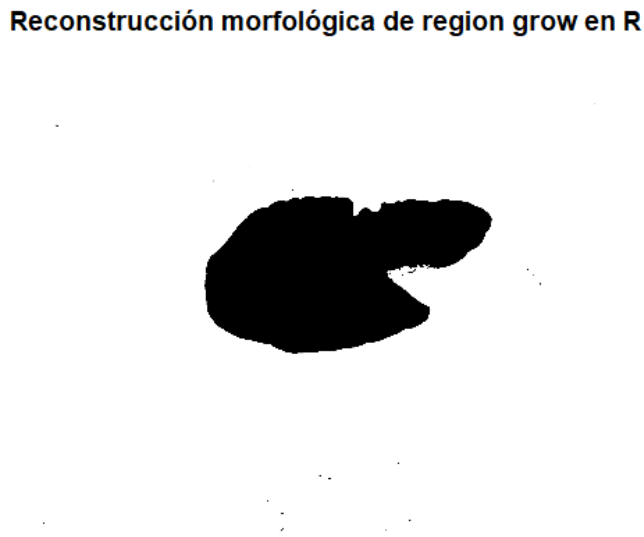


Figura 7.5: Reconstrución en R.

Para cuantificar la diferencia entre ambas imágenes, se han trasladado los datos de Matlab a R (mediante R.matlab como se explica anteriormente), pudiendo así restar la imagen producida en Matlab y en R y calcular su semejanza.

Los pasos a seguir fueron: trasladar los datos de Matlab a la carpeta de R con la extensión .mat; luego, instalar la librería R.matlab y leer los datos para poder guadarlos; a continuación se convirtieron ambas en matrices y, por último, se restaron, teniendo en cuenta que el dato de Matlab está traspuesto (diferencia también explicada anteriormente). Para restarla se utilizó un for restando elemento a elemento. El resultado obtenido se aprecia en la figura 7.6.

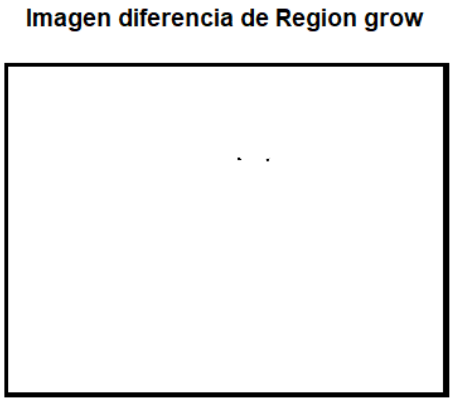


Figura 7.6: Diferencia region grow.

Se visualiza que no hay una diferencia notable, para confirmarlo se calculó el porcentaje de bits igual a '0s' que corresponden a donde la imagen es igual en ambos casos, ya que los puntos negros que se observan corresponden a los bits en los que la imagen no es igual (son '1s'). Se utilizó la siguiente formula, siendo c el número de elementos igual a '1s' en la matriz, y nelements el número de elementos totales.

equal = 100-((c/nelements)*100)

equal=99.98907 %

Con esto se puede concluir que las imágenes son idénticas al 99.99 %.

7.2. otocontorno

El cometido de otocontorno es fácil, su entrada es la imagen en blanco y negro y el resultado es el contorno de los objetos de la imagen. También realiza un giro según la orientación de la imagen. Las dificultades en este caso fue la obtención de una matriz con etiquetas conectadas, saber el número de objetos interesantes a estudiar, el uso de operaciones morfológicas y descubrir la forma de obtener las propiedades de la imagen en R.

Se realizó un *script* que pintará el contorno tanto en R como en Matlab, para comparar que se realiza del mismo modo. La figura 7.7 se puede observar el contorno realizado en Matlab y la 7.8 en R.

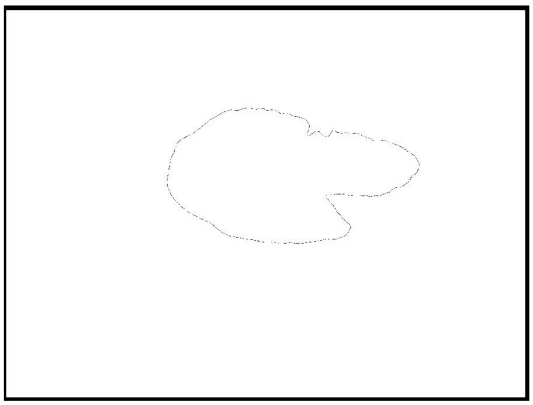


Figura 7.7: Contorno en Matlab.

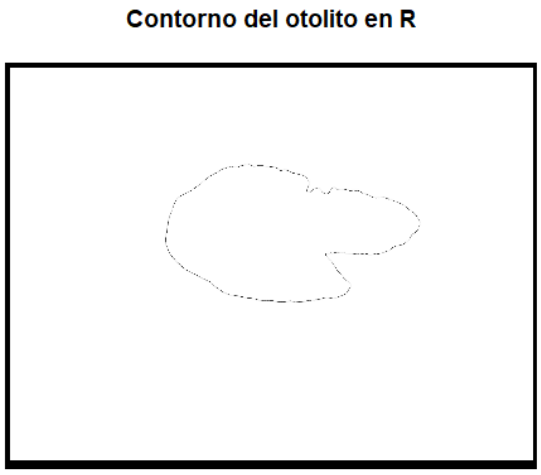


Figura 7.8: Contorno en R.

Por último se procedió a realizar la imagen diferencia entre Matlab y R como con la función anterior, region grow. Fue más díficil su resta ya que la imagen del contorno en Matlab y en R no tenían el mismo tamaño y costó descubrir el motivo. El motivo fue que en Matlab al utilizar la función *imrotate* cambia las dimensiones de la imagen, para evitarlo hay un parámetro de entrada llamado 'crop' que mantiene las dimensiones de la imagen. Una vez solucionado este error, se procedió a realizar los mismos pasos que con la función anterior, y la imagen diferencia resultado en este caso fue la figura 7.9.

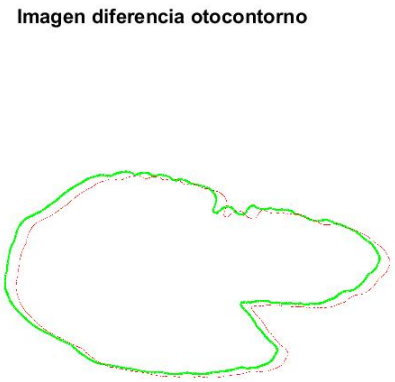


Figura 7.9: Imagen diferencia otocontorno.

Como se puede observar en este ejemplo, la función diferencia indica que en Matlab y en R no se obtiene exactamente la misma imagen, aunque si el mismo contorno, porque se visualiza en la figura 7.9 la misma forma en ambos casos y la misma rotación, pero en Matlab el otolito está desplazado mínimamente hacia la derecha. Por tanto, se concluye que en ambas se rota del mismo modo y se realiza el contorno, pero las imágenes no son iguales.

7.3. otosegment

La función de otosegment realiza una segmentación creciente de la región desde el píxel más oscuro. La diferencia con region grow, es que en region grow es a partir de un umbral que se establece y puede variar, en cambio, en esta función el baremo sería el píxel más oscuro.

Para ello, primero se busca el píxel más oscuro (el de menor valor) y se utiliza como semilla. Más tarde se obtiene el umbral y se utiliza región grow para identificar regiones, y a continuación se realiza un filtrado removiendo las regiones en las que el área es menor al 10 % de la imagen y también un filtrado morfológico para suavizar contornos, por último, se ejecuta un nuevo filtrado para evitar regiones nuevas conectadas.

Para su comprobación se realizaron nuevamente dos *scripts* en Matlab y en R, para ello se visualizó la imagen en blanco y negro, la imagen después de realizar región grow siendo el umbral el píxel más oscuro y por último la imagen después de realizar los filtrados morfológicos. Se pudo ver que en R y en Matlab los resultados no son idénticos. Esto se debe a que cuando se utiliza *bwlabel* en R compara entre distintas conectividades con una escala de grises y en Matlab en cambio, solo compara las conectividades con '0s' o '1s' lo

que produce que se vean distintas las gráficas.

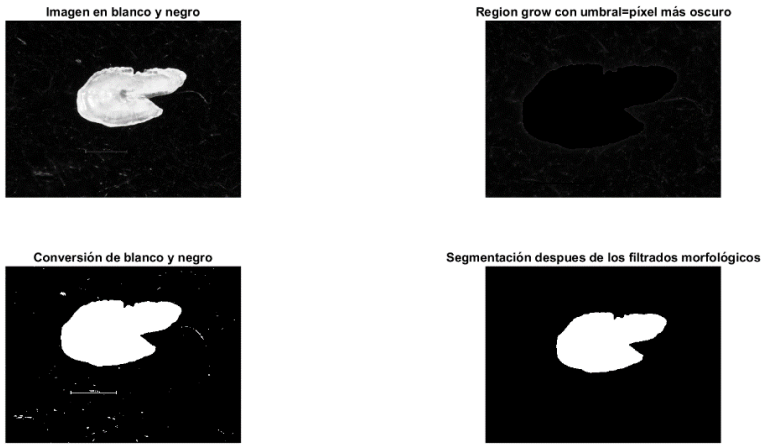


Figura 7.10: Otosegment en Matlab.

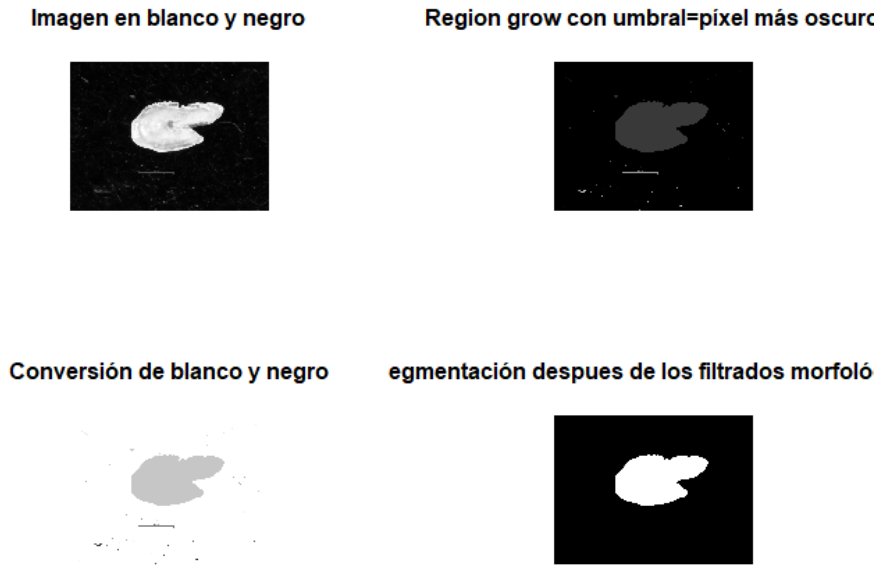


Figura 7.11: Otosegment en R.

Para validar los resultados se ha escogido la imagen final despues de los distintos filtrados en ambos lenguajes (Figura 7.12) se observa que la imagen a simple vista es igual. Pero como en las funciones anteriores se ha realizado la imagen diferencia, realizando exactamente el mismo proceso y obteniendo como resultado la Figura 7.13, donde nuevamente los píxeles blancos corresponden a la diferencia entre ambas imágenes. Del mismo modo que en region grow, se calcula el porcentaje de píxeles negros (correspondientes a donde las imagenes son identicas) y el porcentaje de similitud es 99.127 %.

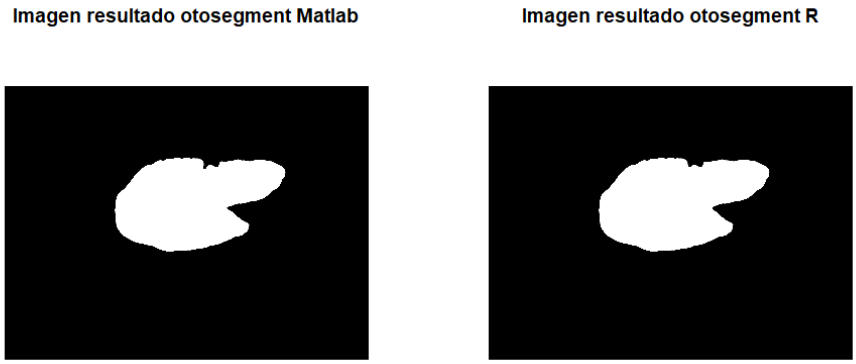


Figura 7.12: Imagen filtrada en R y en Matlab.

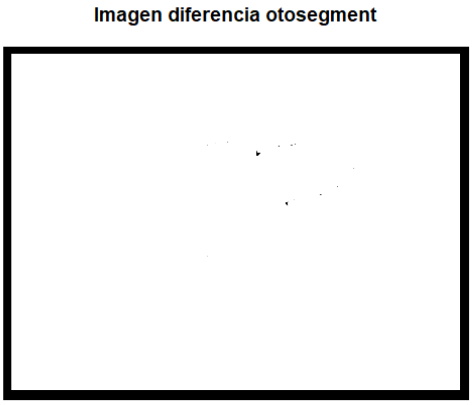


Figura 7.13: Imagen diferencia otosegment.

7.4. feFourier y reFourier

Estas dos funciones implican el sistema de descriptores de forma de Fourier elípticos descritos por primera vez por Kuhl y Giardina en [9]. fEfourier es la transformación directa que crea un "espectro de formasde un contorno cerrado x, y. rEfourier toma un número específico de armónicos del espectro y reconstruye el contorno x, y. Las funciones no solo son útiles para la creación de descriptores de formas, sino también para suavizar contornos o reducir un contorno arbitrario a un número específico de puntos.

En los descriptores de Fourier, se compararan formas las cuales tienen que estar ordenadas y ser una forma cerrada, por ello se utilizan coordenadas polares. Su función es comparar un píxel con el siguiente, se agrupan de 4 en 4 hasta el número de armónicos que deseas analizar, lo que conlleva a que el resultado sea un matriz 4*númeroArmónicosAnalizar.

Más profundamente y como explica el artículo, se describe un procedimiento de clasificación y reconocimiento que es directamente aplicable a clases de objetos que no pueden cambiar de forma y cuyas imágenes están sujetas a distorsiones del equipo sensorial, pero que pueden ocurrir en diferentes orientaciones, tamaños y traslaciones. Las características utilizadas en el procedimiento son normalizar coeficientes de Fourier derivados de códigos de cadena de los contornos de la imagen. La normalización se realiza de acuerdo a varias propiedades elípticas de los coeficientes de Fourier [9].

Esta traducción produjo problemas al ser larga, pero las conversiones eran de aspectos trivales en R. Lo más complicado fue uno de los parámetros de entrada de la función feFourier, ya que los parámetros son los siguientes *outline, iNoOfHarmonicsAnalyse, bNormaliseSizeState, bNormaliseOrientationState*, excepto el primero, los demás son fáciles de enterder, *iNoOfHarmonicsAnalyse* es el número de armónicos que quieres analizar, y los otros dos son booleanos para decidir si quieres normalizar el tamaño y/o la orientación. Sin embargo, *outline* tiene que ser el contorno de una región con una cierta conectividad, si se quiere estudiar el otolito, lo más logico es buscar el contorno más grande con la conectividad adecuada, y este procedimiento hacerlo en R y en Matlab. Esta cuestión costó resolverla, se realizaron diversas pruebas, como con otras funciones importando el mismo *input*, o creando los datos en ambas plataformas.

Lo que se observó es que con exactamente el mismo *input*, en este caso *outline*, daba el mismo resultado tanto en feFourier y reFourier (Figuras 7.9 y 7.10)

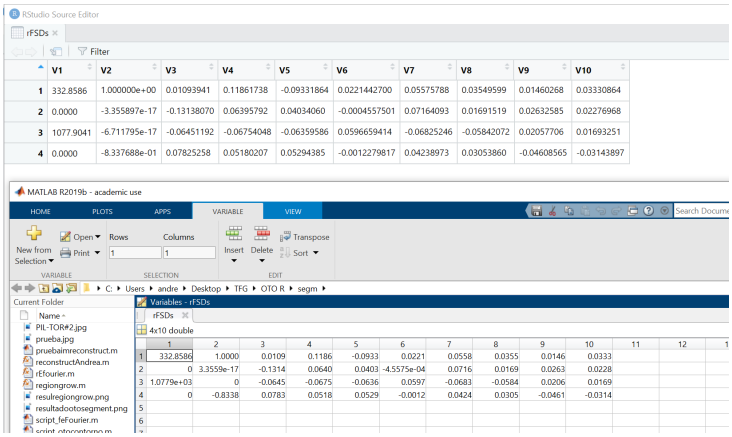
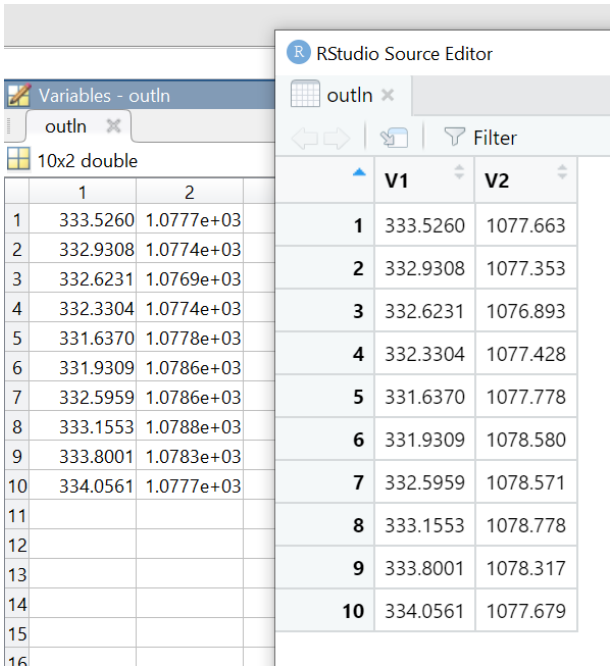


Figura 7.14: Resultado feFourier en Matlab y en R.



	1	2
1	333.5260	1.0777e+03
2	332.9308	1.0774e+03
3	332.6231	1.0769e+03
4	332.3304	1.0774e+03
5	331.6370	1.0778e+03
6	331.9309	1.0786e+03
7	332.5959	1.0786e+03
8	333.1553	1.0788e+03
9	333.8001	1.0783e+03
10	334.0561	1.0777e+03
11		
12		
13		
14		
15		
16		

	V1	V2
1	333.5260	1077.663
2	332.9308	1077.353
3	332.6231	1076.893
4	332.3304	1077.428
5	331.6370	1077.778
6	331.9309	1078.580
7	332.5959	1078.571
8	333.1553	1078.778
9	333.8001	1078.317
10	334.0561	1077.679

Figura 7.15: Resultado reFourier en Matlab y en R.

Sin embargo, obteniendo el *output* en ambas plataformas, no se ordenaban los datos de la misma forma, por lo que no se obtenian los resultados idénticos.

7.5. otomorphommetry

Esta función tiene diversos aspectos que han provocado problemas, pero se han sabido resolver. El primero es la función llamada como *bwboundaries* en Matlab, la cual rastrea los límites exteriores de los objetos con la conectividad indicada, no hay una función similar en R, pero si se obtuvo el mismo resultado, realizando primero la conectividad y luego el contorno del resultado. El siguiente problema fue que tiene una *struct* de 16 elementos, el equivalente más parecido sería en R una *list*, y por último, como se ha explicado en el anterior capítulo, en el apartado 6.1.2 no se pueden obtener las mismas propiedades de una imagen en los distintos lenguajes.

La verificación de que esta función es correcta se realiza a partir de los valores de los elementos de la estructura y dos gráficas: La primera muestra la distancia al centro de masa (d_L) frente al ángulo desde donde empieza a pintar (ϕ) y la segunda es un histograma de la distancia al centro de masas.

Se comprobaron la *struct* (Matlab) y la *list* (R) verificando que todos los valores sean similares. La estructura muestra diversos parámetros que son iguales en ambos lenguajes, como área, perímetro, diversas variables propias de una elipse como el eje mayor y el menor, excentricidad (grado de desviación de una sección cónica con respecto a una circunferencia), circularidad y otros como la media, la raíz cuadrada, momentos de distintos ordenes, la forma elíptica de Fourier y los descriptores de Shen.

Los descriptores Shen se crearon para un estudio en el que se analizaban las calcificaciones en mamografías; estos descriptores utilizaban medidas de compacidad, momentos y los

descriptores de Fourier (por ello son calculados también en la función), utilizando los tres descriptores de forma de Shen se pudo clasificar sin error las calcificaciones de las mamografías, aunque proviene del artículo [14] que mide la rugosidad de los contornos de calcificaciones, se podría aplicar tambien al otolito para ver sus distintas rugosidades.

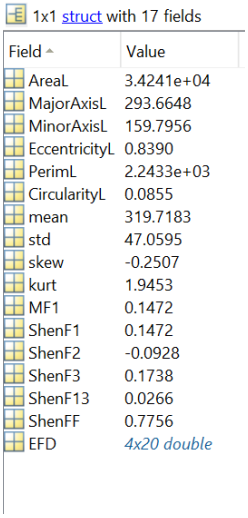


Figura 7.16: *Struct* en Matlab.

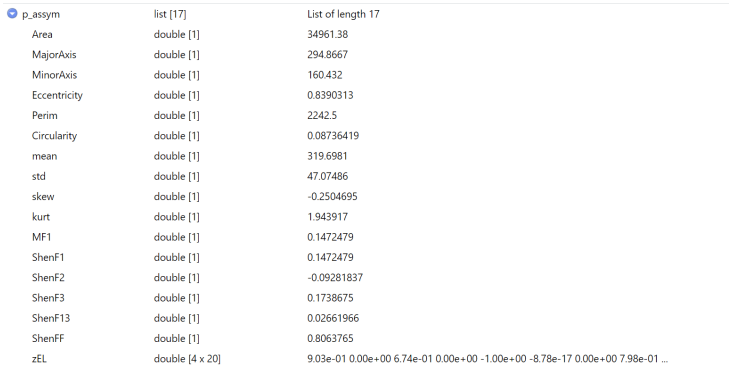


Figura 7.17: *List* en R.

Como se puede observar tanto los datos obtenidos en la *struct* o *list* son iguales. Por último, las siguientes figuras corresponde tanto a la gráfica de distancia al centro y los histogramas.

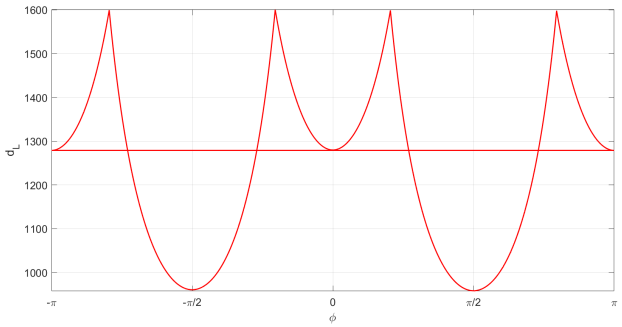


Figura 7.18: Distancia al centro en Matlab.

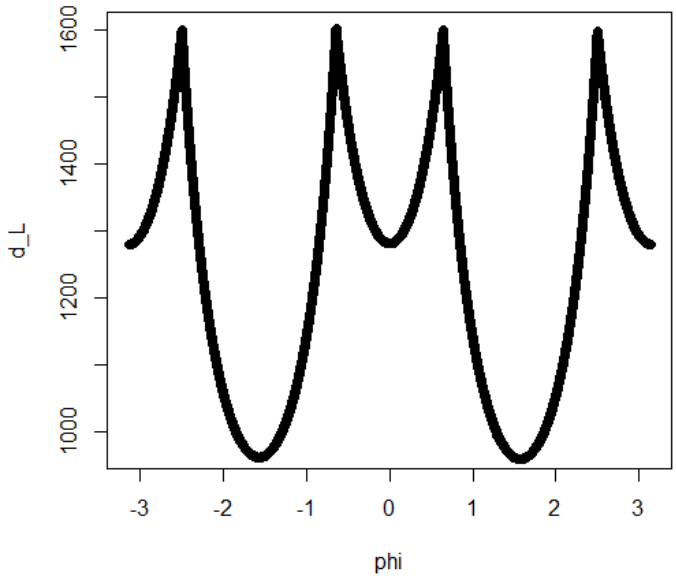


Figura 7.19: Distancia al centro en R.

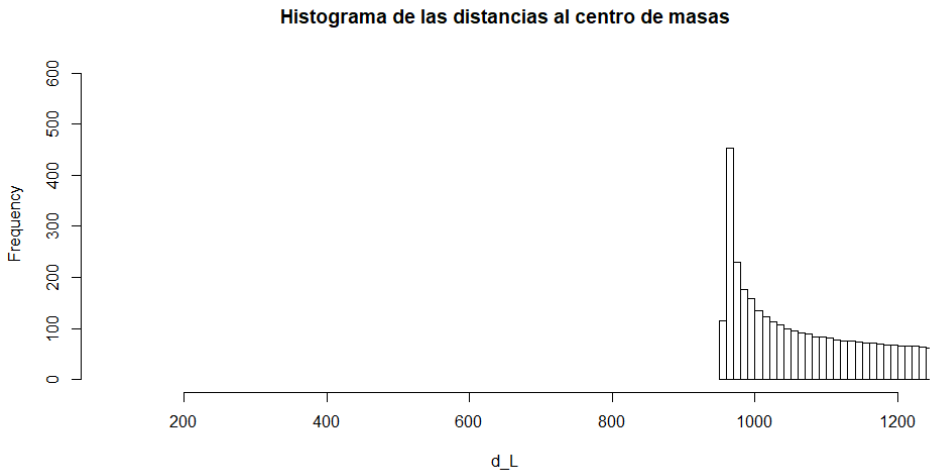


Figura 7.21: Histograma en R.

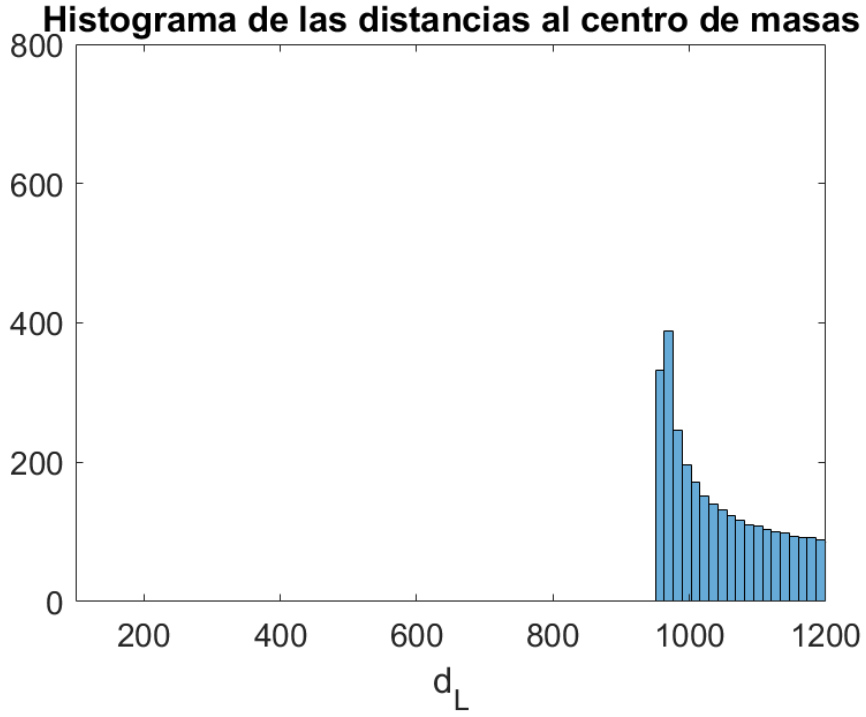


Figura 7.20: Histograma en Matlab.

Se puede observar que también se obtienen los mismos resultados en distancia al centro y el histograma obtiene la misma forma, exceptuando el primer valor.

CAPÍTULO 8

Resultados

8.1. Conclusiones y futuras líneas de trabajo

Los objetivos principales de este trabajo son desarrollar la metodología para conversión de Matlab a R, conocer en profundidad ambos lenguajes y saber detalladamente las diferencias. Más concretamente, estudiar las funciones predefinidas en ambos lenguajes y, como consecuencia, examinar los diversos algoritmos de análisis de imagen digital, creando cuando ha sido necesario funciones que suplieran alguna tarea no cumplimentada en la documentación para llegar al objetivo final, la conversión de código.

La realización de los objetivos ha favorecido en exceso los conocimientos adquiridos de ambos lenguajes destacando el aprendizaje de las diferencias y similitudes en una gran diversidad de tareas. Este estudio puede ayudar a futuros investigadores a decantarse por una herramienta u otra.

También cabe señalar todo lo aprendido en los diversos algoritmos de análisis digital, como la reconstrucción morfológica, el umbral de Otsu o la ecualización de histograma adaptativo limitado de contraste (CLAHE) entre otros. Aunque este programa esté enfocado al estudio de los otolitos de los peces, todas estas funciones se podrían usar en ámbitos de ingeniería biomédica. Por tanto, se realza el interés de su aprendizaje, y del conocimiento de su uso en ambos lenguajes.

Por último, el proceso de validación denota que es posible la conversión de todo el programa a R, ya que ha sido factible la conversión de este primer módulo y ahora se tiene una "guía"de cómo se podría proceder en las otras dos aplicaciones principales registro y segmentación. Quiero resaltar las ventajas también descritas de R, como el ser un software libre con gran portabilidad que se está abriendo camino entre un gran número de empresas de investigación científica. Tras este trabajo se verifica que, aunque R sea una plataforma enfocada en el análisis de datos y estadística, asimismo su uso tiene cabida en el análisis y estudio de las imágenes digitales.

Como líneas futuras de trabajo, se propone la conversión de los dos módulos restantes: registro y segmentación. En dicha conversión se debería seguir la misma metodología aplicada en este trabajo: el estudio de la documentación de ambos lenguajes para encontrar las similitudes en las funciones y la creación de nuevas funciones, en el casodel que no existan, basadas en el estudio de los algoritmos de la función que se quiere implementar

(como ha ocurrido en este proyecto). También es muy útil estudiar la interoperabilidad a la hora de comparar los resultados derivados de ambos programas, sobre todo como ya se ha mencionado anteriormente, el paquete R.matlab, más en concreto el comando readMat.

Al seguir esta metodología, se generarán nuevos retos y se ampliará la evaluación en ambos lenguajes de más algoritmos, inspeccionando más en profundidad las diferencias entre Matlab y R, teniendo así un estudio más detallado, lo que facilitaría aún más la conversión de código para futuros proyectos.

Bibliografía

[1] Artime C, Blanco N.(2013) Paquetes estadísticos con licencia libre (I). Revista electrónica de metodología aplicada . Disponible en: https://www.unioviedo.es/reunido/index.php/Rema/article/view/10307/9917

[2] Cárdenas, T. R., Luque, A. (1996). Otolitos: una introducción a su potencial uso para el estudio de la biología de los peces. Encuentros en la Biología, (32), 4.

[3] Delpiani, S. M., González-Castro, M., Díaz de Astarloa, J. M. (2012). El uso de otolitos y huesos de la cabeza para la identificación de dos especies del género Merluccius, en estudios de predador-presa. *Revista de biología marina y oceanografía*, 47(2), 351-357

[4] Hari Sindhuja (2020) R vs MATLAB: A Learner's Guide . Recuperado de : https://hackr.io/blog/r-vs-matlab

[5] Helfman, G., Collette, B. B., Facey, D. E., Bowen, B. W. (2009). The diversity of fishes: biology, evolution, and ecology. John Wiley Sons.

[6] Henrik Bengtsson,Andy Jacobson,Jason Riedy (2018) R.matlab: Read and Write MAT Files and Call MATLAB from Within R. Recuperado de: https://cran.r-project.org/web/packages/R.matlab

[7] Hiebeler, D. E. (2015). R and Matlab. CRC Press.

[8] Kovesi, P. D. (2000). MATLAB and Octave functions for computer vision and image processing. Centre for Exploration Targeting, School of Earth and Environment, The University of Western Australia, available from: http://www. csse. uwa. edu. au/ pk/research/matlabfns, 147, 230.

[9] Kuhl, F. P., Giardina, C. R. (1982). Elliptic Fourier features of a closed contour. Computer graphics and image processing, 18(3), 236-258.

[10] Morales, Aurelio(2019). Lenguajes de programación para GIS y sus tendencias de crecimiento Recuperado de: https://mappinggis.com/2012/11/lenguajes-de-programacion-gis

[11] Muschelli, John(2018) matlabr: An Interface for MATLAB using System Calls Recuperado de: https://www.rdocumentation.org/packages/matlabr

[12] Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., Zuiderveld, K. (1987). Adaptive histogram equalization and its variations. Computer vision, graphics, and image processing, 39(3), 355-368.

[13] Rodríguez, H. (2005). Imagen digital: conceptos básicos. Marcombo.

[14] Shen, L., Rangayyan, R. M., Desautels, J. L. (1994). Application of shape analysis to mammographic calcifications. IEEE transactions on medical imaging, 13(2), 263-274.

[15] Siddarta Jairam, David Hiebeler(2019) matconv: A Code Converter from the Matlab/Octave Language to R. Recuperado de: https://cran.r-project.org/web/packages/matconv

[16] Verzani, J. (2011). Getting started with RStudio.“ O’Reilly Media, Inc.”.

[17] Vincent, L. (1993). Morphological grayscale reconstruction in image analysis: applications and efficient algorithms. IEEE transactions on image processing, 2(2), 176-201.

[18] Volpedo, A. V., Vaz-dos-Santos, A. M. (2015). Métodos de estudios con otolitos: principios y aplicaciones. *Métodos de estudos com otólitos: princípios e aplicações. 1ª edição bilíngue, Buenos Aires*.

[19] Zahn, C. T., Roskies, R. Z. (1972). Fourier descriptors for plane closed curves. IEEE Transactions on computers, 100(3), 269-281.

Apéndices

APÉNDICE A

Funciones creadas

En el apéndice solo mostraremos las funciones creadas que no encontramos en R, estas mismas funciones en Matlab, y las funciones de OTOLAB en R y sus *scripts* de ejecución y los homólogos a estos en Matlab, no los añadiremos por no ampliar mucho la longitud de la memoria.

Listing A.1: Borrar semillas vecinas

```
borra_semillas_vecinas=function(entrada){
  i=0
  j=0
  entrada<-as.matrix(entrada)
  res<-entrada

  for (i in 1:nrow(res)){
    for (j in 1:ncol(res)){
      if (i==1 || i==nrow(res) || j==1 || j==ncol(res)){
        # ------------------------------------------------------------------------------------------------------------
        # ------------------------------------------------------------------------------------------------------------
        if (i==1){ ##### Primera fila
          # ------------------------------------------------------------------
          if (j==1){ # Esquina sup izq
            if (res[i,j]==1 && (res[i+1,j]==1 || res[i,j+1]==1  || res[i+1,j+1]==1)){
              res[i,j] = 0
            }
          }
          # ------------------------------------------------------------------
          if (j==ncol(res)){ # Esqina superior derecha
            if (res[i,j]==1 && (res[i,j-1]==1 || res[i+1,j]==1  || res[i+1,j-1]==1)){
              res[i,j] = 0
            }
          }
          # ------------------------------------------------------------------
          else{ # Primera fila general
            if (res[i,j]==1 && (res[i+1,j]==1 || res[i,j+1]==1 ||
                res[i,j-1]==1 || res[i+1,j+1]==1 || res[i+1,j-1]==1)){
              res[i,j] = 0
            }
          }
        }

        # ------------------------------------------------------------------------------------------------------------
        # ------------------------------------------------------------------------------------------------------------
        if (i==nrow(res)){ ##### Ultima fila
          # ------------------------------------------------------------------
          if (j==1){ # Esquina inferior izquierda
            if (res[i,j]==1 && (res[i-1,j]==1 || res[i,j+1]==1  || res[i-1,j+1]==1)){
              res[i,j] = 0
            }
          }
          # ------------------------------------------------------------------
          if (j==ncol(res)){ # Esqina inferior derecha
            if (res[i,j]==1 && (res[i,j-1]==1 || res[i-1,j]==1  || res[i-1,j-1]==1)){
              res[i,j] = 0
            }
          }
          # ------------------------------------------------------------------
          else{ # Ultima fila general
            if (res[i,j]==1 && (res[i-1,j]==1 || res[i,j+1]==1 || res[i,j-1]==1 ||
                                res[i-1,j+1]==1 || res[i-1,j-1]==1)){
              res[i,j] = 0
            }
          }
        }

        # ------------------------------------------------------------------------------------------------------------
        # ------------------------------------------------------------------------------------------------------------
```

```
        if (j==1){ ##### Primera columna
          # ------------------------------------------------------------------
          if (i==1){ # Esquina superior izquierda
            # Hecho
          }
          # ------------------------------------------------------------------
          if (i==nrow(res)){ # Esqina inferior izquierda
            # Hecho
          }
          # ------------------------------------------------------------------
          else{ # Primera columna general
            if (res[i,j]==1 && (res[i-1,j]==1 || res[i,j+1]==1 || res[i-1,j+1]==1 ||
                                res[i+1,j]==1 || res[i+1,j+1]==1)){
              res[i,j] = 0
            }
          }
        }

        # ------------------------------------------------------------------------------------------------------------
        # ------------------------------------------------------------------------------------------------------------
        if (j==ncol(res)){ ##### Ultima columna
          # ------------------------------------------------------------------
          if (i==1){ # Esquina superior izquierda
            # Hecho
          }
          # ------------------------------------------------------------------
          if (i==nrow(res)){ # Esquina inferior izquierda
            # Hecho
          }
          # ------------------------------------------------------------------
          else{ # Ultima columna general
            if (res[i,j]==1 && (res[i-1,j]==1 || res[i,j-1]==1 || res[i-1,j-1]==1 ||
                                res[i+1,j]==1 || res[i+1,j-1]==1)){
              res[i,j] = 0
            }
          }
        }
      }else{ ##### Caso general
        if (res[i,j]==1 && (res[i, j-1]==1 || res[i, j+1]==1 || res[i-1, j]==1 ||
        res[i+1, j]==1 || res[i-1, j-1]==1 || res[i-1, j+1]==1 || res[i+1, j-1]==1 || res[i+1, j+1]==1)){
          res[i,j] = 0
        }
      }

    }
  }
  res
}
```

Listing A.2: Preprocesado: expandir la matriz

```
expandMatrix <- function(ma){
  n = nrow(ma)
  m = ncol(ma)
  np1 = n+1
  np2 = n+2
  mp1 = m+1
  mp2 = m+2

  newM = matrix(0, np2, mp2)

  newM[1,1] = ma[1,1] # Top left corner
  newM[1,mp2] = ma[1, m] # Top right corner
  newM[np2,1] = ma[n, 1] # Bottom left corner
  newM[np2,mp2] = ma[n, m] # Bottom right corner


  newM[1, 2:mp1] = ma[1, 1:m] # First Row
  newM[np2, 2:mp1] = ma[n, 1:m] # Last Row
  newM[2:np1, 1] = ma[1:n, 1] # First column
  newM[2:np1, mp2] = ma[1:n, m] # Last column


  for (i in 1:n){
    for (j in 1:m){
      ip1 = i+1
      jp1 = j+1
      newM[ip1,jp1] = ma[i,j]
    }
  }

  return(newM)
}
```

Listing A.3: Reconstrucción morfológica

```
reconstructR <- function(marker, mask)
{

  n = nrow(marker)
  m = ncol(marker)
  II = expandMatrix(mask)
  JJ = expandMatrix(marker)

  mp1 = m+1
  mp2 = m+2
```

```
np1 = n+1
np2 = n+2

for (i in 2:np1){
  for (j in 2:mp1){

    ip1 = i+1
    jp1 = j+1
    im1 = i-1
    jm1 = j-1
    # Current
    v0 = JJ[i, j]
    # Down
    v1 = JJ[ip1, j]
    # Right
    v2 = JJ[i, jp1]
    # Right Down
    v3 = JJ[ip1, jp1]
    # Left Down
    v4 = JJ[ip1, jm1]

    points = c(v0, v1, v2, v3, v4)

    JJ[i,j] = max(points)

  }
}

read = 1
write = 1
#########################################################
# WARNING! TAIL DIMENSION NEEDS TO BE BIG ENOUGH #
# A MORE ELEGANT SOLUTION WOULD BE TO CREATE A   #
# DYNAMIC SIZE TAIL, FOR NOW WE ASSUME A LARGE   #
# NUMBER WILL BE SUFFICENT                       #
#########################################################
tail = matrix(0, 1e7, 2)

for (i in np1:-1:2){ # np1 originalmente
  for (j in mp1:-1:2){

    ip1 = i+1
    jp1 = j+1
    im1 = i-1
    jm1 = j-1
    # Current
    v0 = JJ[i, j]
    # Up
    v1 = JJ[im1, j]
    # Left
    v2 = JJ[i, jm1]
    # Left Up
    v3 = JJ[im1, jm1]
    # Right Up
    v4 = JJ[im1, jp1]

    points = c(v0, v1, v2, v3, v4)

    maximum = max(points)
    JJ[i,j] = maximum



    if ((v2<maximum) && (v2<II[i, jp1])){
      tail[write, 1] = i
      tail[write, 2] = j
      write = write + 1
    }

    else{

      if ((v1<maximum) && (v1<II[ip1, j])){
        tail[write, 1] = i
        tail[write, 2] = j
        write = write + 1
      }
      else{
        if ((v4<maximum) && (v4<II[ip1, jm1])){
          tail[write, 1] = i
          tail[write, 2] = j
          write = write + 1
        }
        else{
          if ((v3<maximum) && (v3<II[ip1, jp1])){
            tail[write, 1] = i
            tail[write, 2] = j
            write = write + 1
          }
        }
      }
    }
  }
}

while (read < write){
  i = tail[read,1]
  j = tail[read,2]
  maximum = JJ[i, j]
  read = read + 1
```

```
ip1 = i+1
jp1 = j+1
im1 = i-1
jm1 = j-1

# Left up
if (im1 > 1 && jm1 > 1){
  v1 = JJ[im1, jm1]
  if ((v1<maximum) && (II[im1, jm1]!=v1)){
    if (maximum<II[im1, jm1]){
      JJ[im1, jm1] = maximum
    }
    else{
      JJ[im1, jm1] = II[im1, jm1]
    }

    tail[write, 1] = im1
    tail[write, 2] = jm1
    write = write+1

  }
}

# Up
if (im1 > 1){
  v1 = JJ[im1, j]
  if ((v1<maximum) && (II[im1, j]!=v1)){
    if (maximum<II[im1, j]){
      JJ[im1, j] = maximum
    }
    else{
      JJ[im1, j] = II[im1, j]
    }

    tail[write, 1] = im1
    tail[write, 2] = j
    write = write+1

  }
}

# Right up
if (im1 > 1 && jp1 < mp2){
  v1 = JJ[im1, jp1]
  if ((v1<maximum) && (II[im1, jp1]!=v1)){
    if (maximum<II[im1, jp1]){
      JJ[im1, jp1] = maximum
    }
    else{
      JJ[im1, jp1] = II[im1, jp1]
    }

    tail[write, 1] = im1
    tail[write, 2] = jp1
    write = write+1

  }
}



# Left
if (jm1 > 1){
  v1 = JJ[i, jm1]
  if ((v1<maximum) && (II[i, jm1]!=v1)){
    if (maximum<II[i, jm1]){
      JJ[i, jm1] = maximum
    }
    else{
      JJ[i, jm1] = II[i, jm1]
    }

    tail[write, 1] = i
    tail[write, 2] = jm1
    write = write+1
  }
}

# Right
if (jp1 < mp2){
  v1 = JJ[i, jp1]
  if ((v1<maximum) && (II[i, jp1]!=v1)){
    if (maximum<II[i, jp1]){
      JJ[i, jp1] = maximum
    }
    else{
      JJ[i, jp1] = II[i, jp1]
    }

    tail[write, 1] = i
    tail[write, 2] = jp1
    write = write+1
  }
}

# Left Down
if (ip1 < np2 && jm1 > 1){
```

```
      v1 = JJ[ip1, jm1]
      if ((v1<maximum) && (II[ip1, jm1]!=v1)){
        if (maximum<II[ip1, jm1]){
          JJ[ip1, jm1] = maximum
        }
        else{
          JJ[ip1, jm1] = II[ip1, jm1]
        }

        tail[write, 1] = ip1
        tail[write, 2] = jm1
        write = write+1

      }
    }

    # Down
    if (ip1 < np2){
      v1 = JJ[ip1, j]
      if ((v1<maximum) && (II[ip1, j]!=v1)){
        if (maximum<II[ip1, j]){
          JJ[ip1, j] = maximum
        }
        else{
          JJ[ip1, j] = II[ip1, jm1]
        }

        tail[write, 1] = ip1
        tail[write, 2] = j
        write = write+1

      }
    }

    # Right Down
    if (ip1 < np2 && jp1 < mp2){
      v1 = JJ[ip1, jp1]
      if ((v1<maximum) && (II[ip1, jp1]!=v1)){
        if (maximum<II[ip1, jp1]){
          JJ[ip1, jp1] = maximum
        }
        else{
          JJ[ip1, jp1] = II[ip1, jp1]
        }

        tail[write, 1] = ip1
        tail[write, 2] = jp1
        write = write+1

      }
    }

  }

  res = matrix(0, n, m)
  res = JJ[2:np1, 2:mp1]
  return(res)

}
```

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA