UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR

INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

MASTER EN INGENIERÍA DE TELECOMUNICACIONES

Conectividad funcional dinámica en la enfermedad de Alzheimer

Autor:

D. Jesús Chaparro Mardaras

Tutores:

D. Jesús Poza Crespo

Pablo Núñez Novo

Valladolid, Septiembre de 2019

TÍTULO:	**Conectividad funcional dinámica en la enfermedad de Alzheimer**
AUTOR:	**D. Jesús Chaparro Mardaras**
TUTORES:	**D. Jesús Poza Crespo** **D. Pablo Núñez Novo**
DEPARTAMENTO:	**Departamento de Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

PRESIDENTE:	**Dr. D. Roberto Hornero Sánchez**
VOCAL:	**Dr. D. Javier Díaz Pernas**
SECRETARIO:	**Dr. D. Carlos Gómez Peña**
P. SUPLENTE:	**Dr. D. Miguel López-Coronado**
V. SUPLENTE:	**Dr. Dña. Miriam Antón Rodríguez**
S. SUPLENTE:	**Dr. Dña. María García Gadañón**

FECHA:	**Septiembre de 2019**
CALIFICACIÓN:	

Resumen del TFM

La enfermedad de Alzheimer es un grave problema de la sociedad actual. Debido a su difícil diagnóstico y su largo desarrollo a lo largo de la vida del paciente surge la idea de intentar aprovechar los medios existentes para mejorar el proceso de detección de esta enfermedad. Hasta el momento el diagnóstico certifica la enfermedad ya en una fase que afecta gravemente a la salud y el bienestar del paciente. Por ello hay que intentar detectarla en la fase denominada deterioro cognitivo leve. Tras varios estudios y trabajos desarrollados, se ha llegado a la conclusión de que hay ciertos patrones en el electroencefalograma que indican que la enfermedad se encuentra en dicha fase. Gracias al procesado de señal se pueden estudiar estos patrones y probablemente llegar a una solución para poder diagnosticar la enfermedad de Alzheimer a tiempo.

Debido al alto coste computacional de los métodos de conectividad funcional dinámica surge la motivación de buscar la forma de reducirlo. La forma más sencilla de abordar este problema implica un cambio de lenguaje y de entorno de computación. Al emplear Matlab, el sofware utilizado hasta el momento, el tiempo de computación supera las 3 horas para un único sujeto y un único método. Traducir estas operaciones de C++ a Python además de reducir el tiempo de computación tiene la ventaja de que el software de compilación de Python (Spyder) es de licencia abierta.

Al traducir el lenguaje actual (C++) a Python se aprecia una mejora notable en el tiempo de computación, en algunos casos se reduce de 3 horas a 22 minutos para el cálculo de 1 solo sujeto, aunque dependiendo del cálculo algunos emplean más tiempo. En cuanto al consumo de recursos, en Matlab se consume de media en torno al 85-90 % de la memoria durante todo el tiempo de cálculo con unos picos finales de 93 % de media, sin embargo Spyder tiene unos picos iniciales durante los primeros cálculos en torno al 90 % pero conforme avanza la computación se estabiliza en un 60-80 %. Se han obtenido resultados favorables en cuanto al tiempo de computación por lo que el resultado del trabajo ha sido satisfactorio en términos generales. A pesar de las dificultades de la traducción de un lenguaje a otro se ha podido llevar a cabo.

Palabras clave

Enfermedad de Alzheimer, EEG, conectividad funcional dinámica, Matlab, Python, coste computacional

Abstract

Alzheimer's disease is a serious problem in today's society. Due to its difficult diagnosis and its long development throughout the patient's life, the idea arises to try to take advantage of existing means to improve the process of detection of this disease. So far the diagnosis certifies the disease in a phase that seriously affects the health and welfare of the patient. That's the reason why we have to try to detect it in the phase known as mild cognitive impairment. After several studies and developed works, it has been concluded that there are certain patterns in the electroencephalogram that indicate that the disease is in this phase. Due to signal processing, these patterns can be studied and probably a solution can be found in order to diagnose Alzheimer's disease in time.

Due to the high computational cost of dinamic functional connectivity methods, the motivation arises to look for ways to reduce it. The simplest way to address this problem involves a change of language and computing environment. Using Matlab, the software used so far, the computing time exceeds 3 hours for a single subject and a single method. Translating these operations from C++ to Python in addition to reducing computing time has the advantage that Python compilation software (Spyder) is open license.

When translating the current language (C++) to Python is seen a significant improvement in computing time, in some cases reduced from 3 hours to 22 minutes for the calculation of 1 single subject, although depending on the calculation some spend more time. In terms of memory consumption, Matlab consumes on average around 85-90 % of the memory during the entire time of calculation with final peaks of 93 % on average, however Spyder has some initial peaks during the first calculations around 90 % but as it advances the computation is stabilized at 60-80 %. Favourable results have been obtained in terms of computation time so the result of the work has been satisfactory in general terms. In spite of the difficulties in translating from one language to another, it has been possible to do so.

Keywords

Alzheimer's disease, EEG, dynamic functional connectivity, Matlab Python, computational cost

Agradecimientos

A mi familia y a Cris por su inestimable apoyo. A mis tutores por su paciencia y ayuda.

Índice general

Índice de figuras

Capítulo 1

Introducción

El procesado de señal ha facilitado durante la historia la detección de varias enfermedades, tanto como cardiopatías, tumores y cualquier enfermedad o problema que pueda ser detectado mediante una señal, tanto acústica como eléctrica, óptica, magnética... Este procesado es un reto para la ingeniería y está en constante desarrollo como en el trabajo que se describe en las siguientes páginas.

1.1. Procesado de señales biomédicas

En el cuerpo humano se generan señales eléctricas como método de transmisión de información entre el sistema nervioso y el resto de órganos y tejidos que lo componen, las cuales transmiten la información en forma de variaciones de fase y amplitud.

Estas señales se emplean para el diagnóstico de distintas patologías por lo que las denominamos señales biomédicas y están clasificadas en varios tipos, dependiendo de la magnitud de medida:

- **Magnéticas**
- **Eléctricas**
- **Ópticas**
- **Acústicas**
- **Hidráulicas**

Gracias al avance de la tecnología y el desarrollo de los sistemas de obtención y análisis de este tipos de señales se ha podido profundizar en el origen y tratamiento de diversas patolog'ias. En este documento nos basaremos en el estudio de las señales de **EEG** (Electroencefalograma) y **MEG** (Magnetoencefalograma) para comprender la influencia de la enfermedad de Alzheimer en dichas señales y así poder agilizar el proceso de diagnóstico de dicha enfermedad con antelación suficiente para tomar las medidas adecuadas en el tratamiento.

Podemos clasificar el proceso de obtención y procesado de señales biomédicas de la siguiente manera [10]:

- Recopilación e inspección de las señales biomédicas.
    - Detección, muestreo, cuantificación y digitalización de la señal.
    - Procesado previo de la señal para eliminar señales no deseadas externas e internas (Pertenecientes a nuestro organismo).
    - Almacenamiento y/o transmisión de dicha señal.
- Procesamiento de la señal
    - Fragmentación de la señal
    - Conversión y filtrado de la señal
    - Elección de los patrones a detectar
- Clasificación
    - Obtención de características
    - Clasificado de la señal

En este documento se analizará el procesado tiempo-frecuencia mediante el lenguaje de programación **Python** de señales de magnetoencefalogramas de sujetos de control y pacientes con una probabilidad elevada de sufrir la enfermedad de Alzheimer (EA).

1.2. Contextualización del problema abordado

La enfermedad de Alzheimer (EA) es una enfermedad neurodegenerativa caracterizada por la aparición de síntomas de deterioro cognitivo y transtornos de conducta, los cuales empeoran a según se desarrolla la enfermedad a lo largo de los años. Se estima que la enfermedad puede empezar a afectar al paciente 30 años antes de tener síntomas clínicos en forma de neurodegeneración.

Durante las primeras fases de la enfermedad se produce un aumento de la acumulació de anillos neurofibrilares y placas seniles. Esta situación llega a un punto en el cual se empiezan a notar los síntomas de la EA. Dichos síntomas se pueden distinguir de los propios del envejecimiento normal pero no cumplen los requisitos para ser considerados EA [10].

El método principal para la obtención de las señales para estudiar la EA es el electroencefalograma (EEG), el cuál capta las diferencias de potencial entre dos puntos en la cabeza del paciente. El EEG ofrece información sobre el acoplamiento neuronal gracias a su alta resolución temporal, la cual permite analizar las rápidas variaciones de la conectividad neuronal [25].

Gracias a este sistema se realizó un estudio en el cual se cuantificó la dinámica de las medidas de red utilizando un enfoque de ventana deslizante(referencia nuñez), revelando

la existencia de una red central formada por un conjunto de regiones cerebrales que pueden desempeñar dinámicamente el papel de centros locales o globales.

Mediante los EEG se ha podido estudiar la denominada conectividad funcional estática, la cual asume que la interdependencia estadística entre las distintas regiones cerebrales permanece temporalmente estacionaria durante el estado de reposo. Estos estudios han aportado mucha información sobre las propiedades neurofisiológicas del cerebro, pero más tarde se propuso que el cerebro no permanecía en un estado de equilibrio durante el estado de reposo, sino que muestra patrones de conectividad fluctuantes, lo cuál se ha denominado conectividad funcional dinámica (dFC).

Dicha cuantificación podría proporcionar información relevante sobre las propiedades de las redes cerebrales [10]. La caracterización de dFC en pacientes con trastornos neurológicos podría utilizarse para descubrir las posibles alteraciones causadas por la EA. Esto podría ayudar a comprender mejor el sustrato neuronal y ser más efectivos al emplear los tratamientos a los pacientes [25].

1.3. Hipótesis

La principal hipótesis en la que se basa este trabajo es que mediante el estudio de la conectividad funcional dinámica de pacientes sanos y pacientes con EA se pueden diferenciar patrones neuronales que pueden ser indicativo del desarrollo de la EA. El procesado de estas señales está en constante desarrollo, como el que se ha realizado en este trabajo. Hasta ahora se han obtenido resultados empleando MATLAB, pero esta herramienta tiene varias limitaciones, entre ellas en gasto computacional. Por ello se desarrolló la idea de traducirlo a un lenguaje más potente (PYTHON) y analizar las diferencias de rendimiento, además de otras ventajas que se detallarán en este trabajo.

1.4. Objetivos

La finalidad principal de este proyecto es mejorar los algoritmos existentes para el procesado de las medidas de conectividad funcional. Dicha mejora se llevará a cabo traduciendo dichos algoritmos al lenguaje de programación **Python**, el cuál tiene varias ventajas que serán descritas en el capítulo 3. La mejora más notable es el tiempo de computación comparado con **Matlab**, el cuál se reduce notablemente, por otro lado **Python** tiene otra serie de desventajas que se describirán en esta memoria. El objetivo principal es llevar a cabo la traducción del código satisfactoriamente y analizar las diferencias de rendimiento. Para alcanzar este objetivo se han planteado también los siguientes objetivos específicos:

- Familiarizarse con la enfermedad de Alzheimer y los métodos de procesado necesarios para analizar la conectividad dinámica funcional.

- Estudiar y comprender los resultados y operaciones que realizan los métodos de análisis de conectividad implementados en MATLAB.
- Diseñar e implementar en PYTHON el toolbox que realice una traducción del código anterior y estudio de las funciones y librerías necesarias para dicho desarrollo.
- Analizar los resultados comparando el rendimiento del toolbox en PYTHON con el toolbox en MATLAB.
- Extraer conclusiones sobre el trabajo desarrollado.

1.5. Descripción de la memoria

En este TFM se describe el procedimiento llevado a cabo para el desarrollo de un toolbox en Python para intentar mejorar el rendimiento del código existente en Matlab. Para ello esta memoria se desarrolla en 5 capítulos los cuales se distribuyen de la siguiente manera:

- **Capítulo 2: Planteamiento del problema** En este capítulo se hará una descripción de la enfermedad de Alzheimer y sus fases, haciendo énfasis en el deterioro cognitivo leve. Por otro lado se describirán los métodos actuales de detección de la enfermedad (EEG) y los métodos de análisis que se emplean como medidas de conectividad, la conectividad funcional y los métodos estadísticos. Por último se analizan las limitaciones encontradas en la herramienta de computación empleada hasta ahora (MATLAB) para dichos métodos de análisis.
- **Capítulo 3: Diseño del toolbox en Python** En este capítulo se describe todo el proceso llevado a cabo para la traducción del toolbox anterior (MATLAB) al toolbox actual (PYTHON), así como las diferencias principales de implementación.
- **Capítulo 4: Resultados y discusión** Aquí se hará un análisis de los resultados obtenidos comparándolos con los del estudio anterior para analizar el rendimiento de este trabajo.
- **Capítulo 5: Conclusiones y líneas futuras** Por último, en este capítulo se detallarán las conclusiones a las que se han llegado y las posibles líneas de investigación que podrían continuarlo.
- **Bibliografía:** Indica las fuentes bibliográficas que se han consultado para la realización de este trabajo.

Capítulo 2

Planteamiento del problema

En este capítulo se hará una breve descripción de los principales factores en los que se basa este trabajo. En primer lugar se hablará de la enfermedad sobre la cual se ha desarrollado el estudio, EA. Posteriormente se describirá el medio de obtención de las señales de estudio y los métodos de análisis empleados. Por último se hará una descripción de las limitaciones del software empleado hasta el momento para calcular las medidas de conectividad funcional.

2.1. La demencia debida a la enfermedad de Alzheimer y el deterioro cognitivo leve

Como se ha comentado en el capítulo anterior (1) la enfermedad de Alzheimer es una enfermedad neurodegenerativa que afecta a un porcentaje muy alto de la población actual. Aunque su origen se desconoce, hay investigaciones que asocian la enfermedad con la aparición y el aumento de anillos neurofibrilares y placas seniles.

Su diagnóstico es complicado ya que aunque el aumento de dichos anillos y placas genere síntomas de la EA, éstos durante la primera fase de le enfermedad no cumplen los requisitos para ser considerados EA. Esta primera etapa de la enfermedad, se denomina deterioro cognitivo leve y es la más complicada de diagnosticar. Por ello, este trabajo como el anterior en el que se basa y posibles líneas futuras pretenden ayudar a la detección de esta fase. Posteriormente, la enfermedad afecta a las zonas del cerebro encargadas del razonamiento y el lenguaje sin afectar a la motricidad del paciente. Otros de los signos significativos es la aparición de amnesia anterógrada y de amnesia retrógrada. Según avanza la enfermedad aparecen otros síntomas como la apraxia, pérdida del sentido del humor, dificultades para la comprensión del lenguaje y otros trastornos de la comunicación. Antes de la atrofia global cerebral, aparecen síntomas similares a los de la depresión. El enfermo en estado de atrofia cerebral acaba padeciendo una incapacidad completa [10].

2.2. Electroencefalografía

Las señales de estudio, de duración 5 minutos, fueron obtenidas a través de un electroencefalograma gracias al Departamento de Neurofisiología Clínica del Hospital Universitario de Valladolid Río Hortega". Se empleó un EEG de 19 canales (XLTEK R, Natus Medical), con una frecuencia de muestreo de 200 Hz, siguiendo con las especificaciones del sistema internacional 10-20 de Fp1, Fp2, Fz, F3, F4,F7, F8, Cz, C3, C4, T3, T4, T5, T6, Pz, P3, P4, O1, y O2. Durante la obtención del EEG se establecieron una serie de premisas. Se realizó con los sujetos en estado de vigilia, tranquilos y sin abrir los ojos. Así mismo se localizaron en tiempo real los artefactos relacionados con el movimiento muscular, el movimiento de ojos y la somnolencia para separarlos del resto de la señal [25].

Una vez obtenida esta señal, se le realizó un preprocesado listado a continuación [25]:

- Eliminación de componentes con artefactos mediante un análisis preliminar independiente de los componentes.
- Filtrado FIR (Ventana Hamming, orden de filtrado 2000) limitar el contenido espectral a la banda de frecuencia ancha de[1 70] Hz y eliminar la potencia de ruido a 50 Hz.
- Eliminación de los artefactos restantes mediante visualización, seleccionando los primeros 60 segundos consecutivos sin artefactos para cada sujeto.

2.3. Métodos de análisis

En esta sección se describirán los distintos métodos de análisis empleados para profundizar en el estudio de la enfermedad de Alzheimer [25].

2.3.1. Medidas de conectividad funcional (MSCOH, PLI y AEC)

En esta sección se van a describir los distintos métodos que se han empleado para caracterizar la conectividad funcional entre señales de EEG.

Para dicha tarea hay varios métodos, de los cuales se han empleado **MSCOH** (*Magnitude Squared Coherence*), **PLI** (*Phase Lag Index*) y **AEC** (*Amplitude Envelope Correlation*). MSCOH se basa en la sensibilidad a la sincronía de fase y amplitud, PLI está basada prácticamente en fases y AEC está basada en amplitudes.

Cada uno de estos métodos, representa un proceso fisiológico subyacente distinto.

- MSCOH

  Se define Coherencia de la Magnitud al Cuadrado como el espectro cruzado estandarizado de dos series temporales complejas X e Y, normalizadas mediante la división del espectro cruzado por la raíz cuadrada del espectro de potencia de cada

serie temporal [29]. El valor absoluto de la coherencia, que es un número complejo, se conoce como coherencia (COH):

$$COHxy(f) = \frac{|SXY(f)|}{\sqrt{PX(f)PY(f)}} \tag{2.1}$$

donde SXY es el espectro cruzado de las señales X e Y , y PX y PY son sus respectivos espectros de potencia. La coherencia puede interpretarse como la cantidad de varianza en una señal que puede ser explicada por la otra, o viceversa[referencia]. MSCOH es el cuadrado de coherencia:

$$MSCOHxy(f) = \frac{|SXY(f)|^2}{PX(f)PY(f)} \tag{2.2}$$

MSCOH se puede considerar como medida de la consistencia de las diferencias de fase entre dos señales ponderadas por sus magnitudes. Esta medida está influenciada por la sincronía de fases. Las series de tiempo fueron ortogonalizadas para cada ventana de tiempo por separado antes de calcular MSCOH en orden. Para evitar la dispersión del campo y la conducción volumétrica, se han ortogonalizado con antelación las series temporales por cada ventana temporal por separado.

- PLI

El Índice de Desfase evalúa la asimetría de la distribución de la diferencia de fase instantánea entre las series temporales [32]. PLI se define como:

$$PLI = |\langle sign[sin(\Delta\Phi(tk))]\rangle| \tag{2.3}$$

Donde $\Delta\Phi(tk)$ es una serie temporal de diferencias de fase entre dos canales. Los valores de PLI están comprendidos entre 0 y 1, siendo cero indicativo de no acoplamiento o acoplamiento con retardo de fase. En cambio el 1 indica un acoplamiento de fase de retardo perfecto distinto de cero. Al contrario del resto de medidas de conectividad, PLI es insensible a los efectos de la conducción volumétrica.

- AEC

AEC es la correlación de la envolvente de amplitud, esto significa que indica la correlación de las magnitudes de dos tiempos para cada frecuencia de estudio.

AEC puede dar valores altos aunque las fases están distribuidas aleatoriamente. Para hacer el cálculo, en primer lugar se filtran las series temporales mediante un filtro pasa banda centrado en las frecuencias de intereés. Después mediante el valor absoluto de la señal analítica obtenemos la envolvente de potencia de cada señal. Para ello se emplea la transformada de Hilbert. Por último, se calcula la conectividad mediante la correlación de Pearson entre las envolventes de potencia de los registros. Como en MSCOH, se han ortogonalizados las series temporales para evitar la dispersión de campo o la conducción volumétrica.

2.3.2. Conectividad funcional dinámica

Para obtener unos datos fiables, aunque se consiga aislar el ruido en las medidas de conectividad, hay que tener en cuenta que las variaciones de señal obtenidas no son suficientes para considerarlo dFC. Los valores obtenidos no dejan de ser una estimación de los posibles valores reales .Únicamente mediante un análisis estadístico correcto podemos saber si las señales obtenidas reflejan una concetividad funcional dinámica. Para ello se realizan N versiones de cada una de las señales originales calculando en ellas las medidas de conectividad. De este modo se determina si el valor observado en la prueba tiene una Z estadísticamente significativa respecto al resto de versiones.Estos datos se obtienen de la desviación estandard de PLI, MSCOH y AEC (κPLI , $\kappa MSCOH$ y κAEC). Las versiones subrogadas de las señales originales tienen las mismas propiedades estadísticas que las señales originales, lo cual se lleva a cabo creando estas señales con una hipótesis nula adecuada. Para confirmar la detección de dFC, las señales subrogadas deben cumplir las propiedades de conectividad funcional estática y carecer de conectividad funcional dinámica. Obteniendo la hipótesis alternativa de presencia de dFC y la hipótesis nula de presencia de sFC.

En el estudio en el que se basa esta memoria [25], se obtuvieron las señales subrogadas a partir de la transformada de Fourier ajustada en amplitud (AAFT) de las señales originales.

2.3.3. Análisis estadístico

Para corregir los valores más elevados de κ no debidos a dFC y los menos elevados debidos a dFC, se normalizaron los valores de κPLI , $\kappa MSCOH$ y κAEC mediante los valores κ promedio de las series temporales subrogadas. Esta normalización nos permite comparar el nivel de conectividad dinámica de cada conexión.

En un primer momento se realizaron pruebas paramétricas para ver si la distribución de los valores anteriormente comentados cumplía las condiciones de estas pruebas. Se evaluó la normalidad con la prueba Shapiro-Wilk y la homocedasticidad con la prueba Levene. Al ver que no se cumplían las condiciones se volvieron a evaluar con pruebas no paramétricas.

La primera de las pruebas, para detectar la interacción entre grupos es las pruebas de Kruskal-Wallis de tres vías, calculada en cada banda de frecuencias. Solo se realizarían pruebas de permutación si se encontraran diferencias estadísticamente significativas entre grupos. Estas pruebas se utilizan para evaluar las diferencias espaciales entre grupos para el κ y el μ de cada medida de conectividad en esas bandas de frecuencia.

Posteriormente se empleó la FDR(False Discovery Rate) para evitar el error de tipo I. Con un nivel de significación de $\alpha = 0,05$, se empleó la FDR para controlar el número de conexiones entre regiones para un número concreto de conexiones [25].

Todo este procesamiento de señal se realizó con Matlab (versión R2017a Mathworks,

Natick, MA) y posteriormente con el lenguaje Python y el software Spyder (versión 3.3.2 Python 3.7.1) perteneciente a Anaconda(versión 1.9.7).

2.4. Limitaciones encontradas

Matlab, al contrario que Spyder, es un software de licencia cerrada al que solo se puede tener acceso mediante pago. Sin embargo, existen otras alternativas para el procesamiento de señal, aunque implican un cambio de lenguaje de programación. Matlab a pesar de ser una herramienta potente, tiene un tiempo de computación bastante alto dependiendo el tipo de cálculos que tenga que hacer. Otro de los principales inconvenientes de Matlab es su altísimo consumo de recursos, consumiento un 80-90 % de memoria con cada método de conectividad.

Capítulo 3

Diseño del toolbox en Python

En primer lugar se describirá la organización del código de matlab. En la figura 3.1 se muestra de qué módulos depende cada uno dentro del código.

El **scriptPrincipal** es el centro de todo el código, al inicio hace una llamada al script **configuracion**, en el cuál se define el directorio de guardado de los resultados, el tipo de medida de conectividad, la frecuencia de muestreo, si hay o no overlap, el tamaño de la ventana, el número de subrogaciones, el número de canales, el número de bandas y la longitud de los trials. Después en función del tipo de medida de conectividad seleccionada hace una llamada a un script o a otro.

En caso de querer estudiar la medida de conectividad MSCOH, directamente se hace una llamada al script **procesadoDFC mscoh**, el cuál hace dentro varias llamadas a funciones auxiliares. Estas funciones son **calculoMSCOH leakage**, **ortogonalization optimizada** y **subrogacionAAFT** la cuál hace una llamada a **AAFT** que a su vez llama a **phaseran alternative**.

En cambio, si queremos analizar las medidas de conectividad PLI y AEC, una vez configurado en el script **configuracion** el script **scriptPrincipal** hace las mismas llamadas anteriormente citadas a excepción de las relacionadas con la medida propiamente dicha, si la medidas es PLI el script **procesadoDFC aec pli** hace llamada al script **calculoAEC prefiltrado optimizada** en caso de que la medida sea AEC y si la medida es PLI hace la llamada al script **calculoPLI prefiltrado**.

Hay que tener en cuenta, cuando se realiza una traducción de MATLAB a PYTHON, que las funciones aunque hagan las mismas operaciones, la lectura de filas y de columnas no es la misma, por lo que hay que comprobar paso a paso cada una de ellas. En PYTHON existe una librería llamada NUMPY la cuál dispone de la mayoría de funciones de MATLAB implementadas en PYTHON, aunque en función de las necesidades para las que fueron diseñadas pueden dar resultados diferentes.

Otra gran diferencia respecto a Matlab es el recorrido de los índices. Matlab la primera posición la toma como un 1 sin embargo en spyder la primera posición es el 0. Es un detalle muy importante a tener en cuenta ya que todas las funciones emplean matrices y

los índices deben ser los adecuados para obtener el resultado deseado.

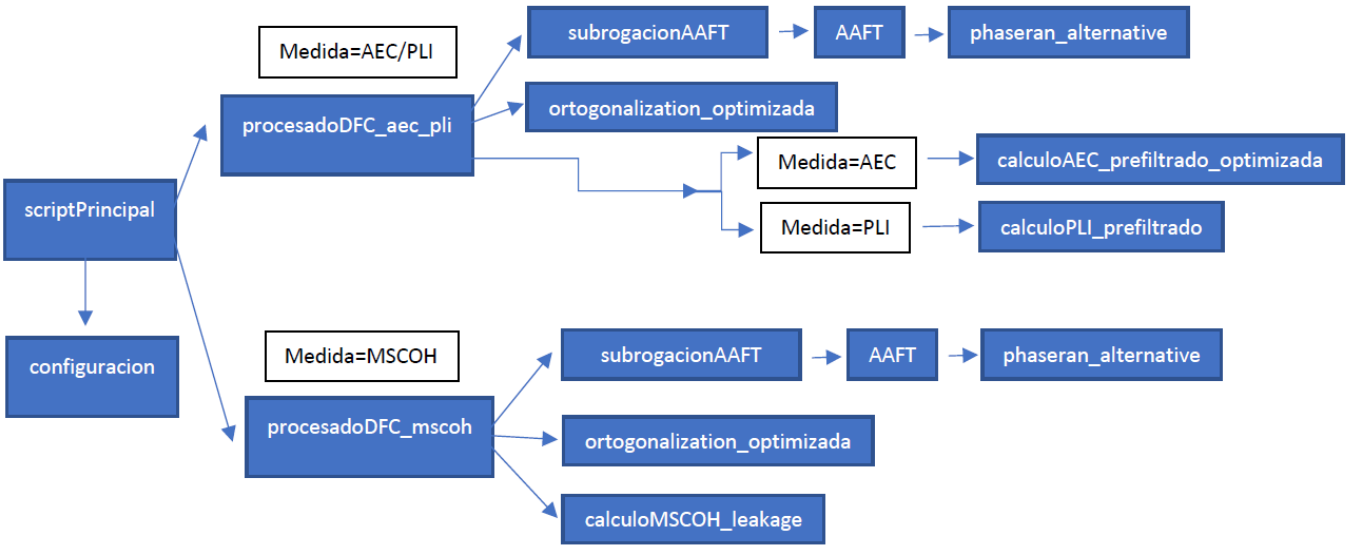

Figura 3.1: Distribución del código original.

El software empleado para la compilación del código en PYTHON es SPYDER, el cuál dispone de una consola similar a la de matlab con ventana de variables etc. Dispone de un indicador de consumo de memoria en tiempo real, el cuál indicará el rendimiento del código.

En la figura 3.2 se puede observar la consola del sofware SPYDER, donde abajo a la derecha dispone del indicador de consumo de memoria como se comentaba en el párrafo anterior.

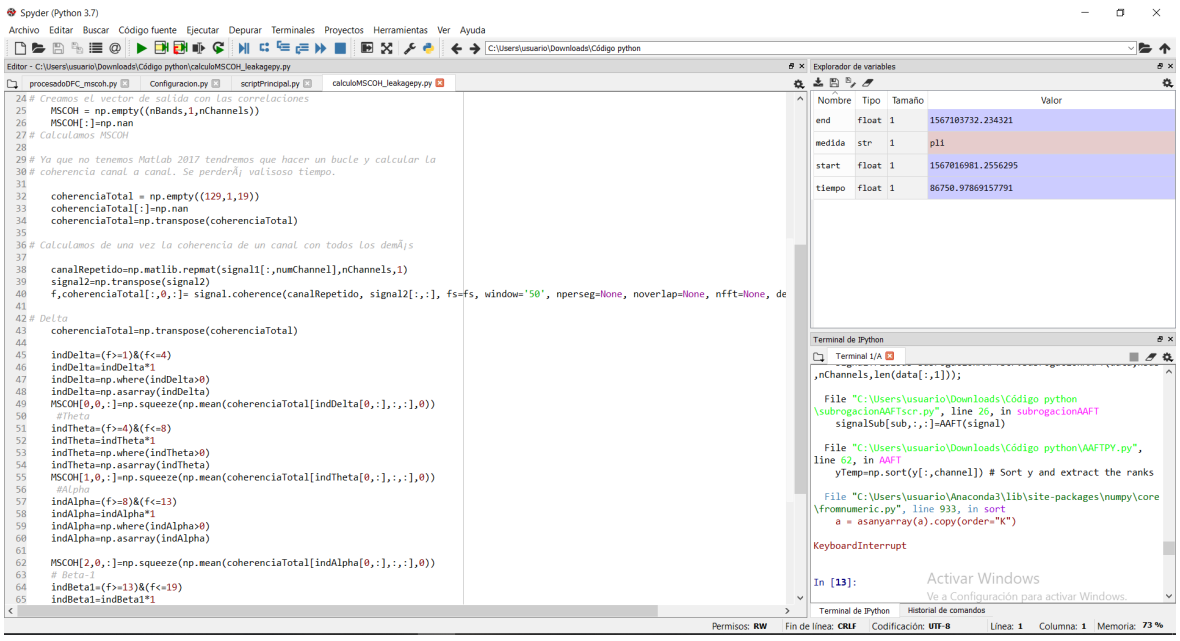

Figura 3.2: Consola SPYDER.py

A continuación se van a ir describiendo los cambios significativos que se han tenido que realizar en cada uno de los scripts independientemente del cambio del lenguaje:

- **scriptPrincipal**: Como se puede ver en la figura 3.3, los únicos cambios tienen que ver con la diferencia del lenguaje simplemente.

```
import Configuracion
#% PROCESADO
from Configuracion import medida

if 'aec' in medida or 'pli'in medida:

    import procesadoDFC_aec_pli

elif 'mscoh' in medida:

    import procesadoDFC_mscoh
```

Figura 3.3: scriptPrincipal.py

- **configuracion:**
  Como se puede ver en la figura 3.4, además de los cambios referidos al lenguaje, se ha añadido el **dirArch** el cuál es el directorio de los scripts empleados para el cálculo de los distintos métodos de conectividad, ya que para realizar el guardado de los resultados se va a cambiar de directorio.

```
# Configuracion
import os
dirArch="C:/Users/usuario/Desktop/PYTHON"
# Carpeta donde tenemos los .mat de las senyales
dirLoad = "C:/Users/usuario/Desktop/PYTHON/SignalsEEG_1_70Hz_60s/"

# Cargamos la carpeta con los sujetos


registro=os.listdir("C:/Users/usuario/Desktop/PYTHON/SignalsEEG_1_70Hz_60s/")

# La medida que queremos calcular
medida = 'aec' # pli, aec o mscoh
fs = 200 # frecuencia de muestre (200 Hz para senyales RIO HORTEGA)

overlap = 1 # si queremos que los calculos se hagan con ventanas con 50% de overlap (1) o no (0)

ventana = 15


if "0" in str(overlap):
    trials = 60/ventana
    trials=int(trials)
        # Carpeta donde queremos guardar los resultados
    dirSave = "C:/Users/usuario/Desktop/prueba/resultsnover" + str(ventana) + str(medida)

if "1" in str(overlap):

    trials = (60/ventana)*2-1
    trials=int(trials)
        # Carpeta donde queremos guardar los resultados
    dirSave = "C:/Users/usuario/Desktop/prueba/resultsnover" + str(ventana) + str(medida)


nSub = 500 # Numero de subrogaciones
nChannels = 19 # Numero de canales 19 (0-18)
nBands = 5 # Numero de bandas (no lo calculamos en gamma) 5 (0-4)
lengthTrials = ventana*fs # Longitud de los trials en muestras (15s: 3000 10s: 2000, 5s: 1000, 3s:600,
```

Figura 3.4: configuracion.py

- **procesadoDFC mscoh y procesadoDFC aec pli:**

  Estos dos scripts contienen la parte principal del cálculo de cada una de los métodos de conectividad. Los dos son iguales en cómputo general a excepción de las operaciones propias de cada método de conectividad. Contienen la carga de archivos así como las llamadas a todas las funciones que se describirán en las siguientes páginas para completar el cálculo y obtener los resultados de las medidas de conectividad. Además de las diferencias sutiles por el cambio de lenguaje, se ha tenido que adaptar los cálculos del p-valor y dividirlos en varias líneas, ya que python no interpreta bien las líneas con operaciones solapadas. Por otro lado para corregir los p-valores con la FDR (False Discovery Rate), se han hecho varias adaptaciones mediante concatenaciones y cambios de tamaño de matrices para poder obtener los datos en el formato deseado.

```
    # Calculamos el p-valor en la banda
compar=stdSubBanda >= stdSignalComp
divis=compar/nSub
suma=sum(divis)
pval[banda,:,:] = suma

    # Calculamos z-scores
zresta=stdSignal[banda,:,:]-np.mean(stdSubBanda,axis=0)
zdiv=zresta/np.std(stdSubBanda,ddof=0)
zscores[banda,:,:] = zdiv
# Corregimos los p-valores con FDR
index_FDR = []
pvalBand=np.squeeze(pval[banda,:,:])
index_FDR = nonzeroadaptada.noncero(np.triu(pvalBand,1))
pvalCor=np.concatenate(pvalBand.T)
index_FDRcor=np.matrix(index_FDR).T
index_FDRcor=index_FDRcor.astype(int).T
pFDR_aux=mne.stats.fdr_correction(pvalCor[index_FDRcor], alpha=0.05, method='indep')# Est
pFDR_aux=pFDR_aux[1]
pFDRMatriz=np.concatenate(pvalBand.T) #% Esto es para crear la matriz con las dimensiones
pFDRMatriz[index_FDRcor] = pFDR_aux # % Se asignan los nuevos valores
pFDRMatriz=pFDRMatriz.reshape(19,19)
pFDRMatriz=pFDRMatriz.T
pFDRMatriz=np.triu(pFDRMatriz,0)+np.transpose(np.triu(pFDRMatriz,1)) # Ponemos la parte i

pvalCorrected[banda,:,:] = pFDRMatriz;
```

Figura 3.5: procesadoDFC mscoh/aec pli.py

- **subrogacionAAFT:**

  Esta función es una traducción literal de la original. Para crear matrices con todos sus valores **nan**, en python primero se crea una matriz con datos aleatorios y después se cambian esos valores por **nan**.

```
from AAFTPY import AAFT
import numpy as np

def subrogacionAAFT(signal=None,nSub=None,nChannels=None,lengthTrials=None):
#SIGNALTRIALSUB Función que crea tantas señales subrogadas en fase de la señal de
#entrada como se indique
#   Variables de entrada:
#   signal: señal a subrogar
#   nSub: número de subrogaciones que se van a realizar
#   Variables de salida:
#   signalSub: matriz tamaño nSub x longitud x nCanales

# Creamos la matriz donde irán las subrogaciones
    signalSub = np.empty ((nSub,lengthTrials,nChannels))
    signalSub [:]=np.nan

    for sub in range (nSub):
        signalSub[sub,:,:]=AAFT(signal)

    return signalSub
```

Figura 3.6: subrogacionAAFT.py

- **AAFT:**

  Esta función calcula la transformada de Fourier ajustada por amplitud (Amplitude Adjusted Fourier Transformed). En este script la diferencia principal con el original radica en hacer uso del comando **random.seed()** para que el valor absoluto de la transformada de Fourier de las señales subrogadas coincida con el valor de la señal original y la fase sea distinta. Por otro lado cuando ordenamos los valores de la señal y obtenemos el índice, en matlab se hace directamente mediante el comando **sort**, sin embargo en python se utiliza **sort** para ordenar y **argsort** para obtener el índice de esos valores.

```
s=np.empty((12000,19))
y = x
yTemp=[]
T=[]
random.seed()# Do this so that each surrogate series has a different phase randomization
seed = np.random.rand() * 10000# get the current generator settings. This is so that all channels in this sur
# have the same phase randomization
# Esto es una prueba para ver si hace falta que sea igual para todos los
# canales
#normalAll=sort(randn(size(y,1),1));

for channel in range (len(y[1,:])):
    random.seed()   #Prevent all surrogates from being the exact same, only the phase randomization has to be

    normal = sorted(np.random.randn(len(y[:,1])))# Make n normal random deviates

    yTemp=np.sort(y[:,channel]) # Sort y and extract the ranks
    T=np.argsort(y[:,channel]) # Get a version of the signal sorted by amplitude
    r =np.argsort(T)
    T=np.sort(T)  # Sort the indexes of the amplitude sorted signal

    # Assign the ranks of y to the normal deviates and apply the phase
    #  randomization
    normal=np.matrix(normal).T
    c=[]
    normal = phaseran_alternative2(normal[r], c, seed)    # Apply phase randmization to the random deviates


    T = np.argsort(normal.T)    # Extract the ranks of the phase randomized normal deviates
    normal=np.sort(normal)  # Sort the phase randomized random deviates
    r = np.argsort(T) # Sort the indexes of the PR random deviates
    T=np.sort(T)

    ytemp1=np.matrix(yTemp[r]).T # Assign the ranks of the phase randomized normal deviates to y
                                 # and obtain the AAFT surrogates

    s[:,channel] = ytemp1.T   # Reorder the amplitude sorted signal by the PR random deviates
return s
```

Figura 3.7: AAFTPY.py

- **phaseran alternative:**
  Para esta función, ha habido que realizar adaptaciones para conseguir el formato adecuado de los datos, así como concatenaciones de vectores, inserción de datos dentro de los mismos. Gracias a estas adaptaciones se ha conseguido obtener los mismos resultados que en su homólogo en Matlab.

```
if math.remainder(N, 2) == 0:
    #random.shuffle(seed)                          # Make sur
    p1 = np.random.rand(h - 1, 1) * 2 * math.pi
    #p(2:N)=[p1' p(h+1) -flipud(p1)'];
    a=np.transpose(p1)
    a=np.append(a,0)
    b=np.transpose(- np.flipud(p1))
    b=np.append(b,0)
    d=np.concatenate((a,b))
    op2=np.delete(d,11999)
    op1=np.transpose(p[[e for e in range (1,N)]])
    p[[e for e in range (1,N)]] = np.add(op1, op2).T
    # Adjust the magnitudes
    m1=m[[e for e in range (0,h + 1)]]
    m2=np.flipud(m[[i for i in range (1,h)]])
    m = np.concatenate((m1,m2))
else:
    p1 = np.random.rand(h, 1) * 2 * math.pi
    #p(2:N)=[p1' p(h+1) -flipud(p1)'];
    a=p1
    b=- np.flipud(p1)
    op2=np.concatenate((a,b))
    op1=np.transpose(p[[e for e in range (1,N)]])
    p[[e for e in range (1,N)]] = np.add(op1, op2).T
    # Adjust the magnitudes
```

Figura 3.8: phaseranalternative.py

- **ortogonalization optimizada:**
  En esta función la principal diferencia de implementación es preparar las matrices

con los datos de cada canal y posteriormente multiplicarlas. En python es importante hacer las operaciones línea a línea intentando evitar las superposiciones de funciones, ya que puede que el resultado no sea correcto.

```
@author: jesuschaparro
"""
import numpy as np

def ortogonalization_optimizada(signal1=None,signal2=None):

#ORTOGONALIZACION Basada en el articulo de O'Neill et al. (2015)
# Realiza una ortogonalizacion de cada canal respecto a todos los demas

    signal_ort = np.empty(( len(signal1) , len(signal1[1,:]) , len(signal1[1,:]) ))
    signal_ort[:]=np.nan
    for channel1 in range (len(signal1[1,:])):
        signal_ort[:,channel1,channel1] = signal1[:,channel1]
        for channel2 in range (len(signal1[1,:])):
            if channel1 != channel2:
              signal1_canal = signal1[:,channel1]
              signal1_canal = np.array([signal1_canal]).T
              signal2_canal = signal2[:,channel2]
              signal2_canal = np.array([signal2_canal]).T
              a=np.transpose(signal2_canal)
              b=np.linalg.pinv(np.transpose(signal1_canal))
              beta=np.dot(a,b)
              h=signal2_canal-beta*signal1_canal
              signal_ort[:,channel2,channel1] = h.transpose()
    return signal_ort
```

Figura 3.9: ortogonalization optimizada.py

- **calculoAEC prefiltrado optimizada:**
  En este caso la diferencia fundamental es crear un vector a partir de una matriz para poder calcular la correlación entre las envolventes de la señal. Para poder obtener la señal Hilbert se hace uso de la librería de python **scipy.signal**.

```
import numpy as np
import scipy
import pandas as pd


def calculoAEC_prefiltrado_optimizada(signal1=None, signal2=None, nChannels=None, currentChannel=None):
    #Calculoc funcion que calcula la correlacion de la envolvente de amplitud de dos
    #senyales
    #   Variables de entrada:
    #   signal1, signal2: senyales sobre las que se va a calcular el parametro
    #   Variables de salida:
    #   AEC: correlacion de amplitud de envolvente.
    #   Por tanto, es un vector de nChannels elementos (un valor por
    #   canal)

    #% Creamos el vector de salida con las correlaciones
    A1 = np.empty ((1, nChannels))
    A1[:] = np.nan
    AEC=np.empty ((1, nChannels))
# Calculamos el logaritmo de la
# envolvente al cuadrado (power envelope). En un bucle lo hacemos banda por
# banda
# En Matlab 2012 no tenemos envelope, así que usamos Hilbert para
# calcular la senyal analítica y la sacamos a partir de ahí

    hilb1 = scipy.signal.hilbert(signal1[:,:])
    hilb2 = scipy.signal.hilbert(signal2[:,:])
    envelope1 = np.squeeze(abs(hilb1))
    envelope2 = np.squeeze(abs(hilb2))
    env1 = np.log(envelope1**2)
    env2 = np.log(envelope2**2)

        # Ahora calculamos la correlacion lineal entre las dos envolventes
    env1 = env1[:, currentChannel]
    env1 = np.array([env1]).T
    for co in range (len(env2[1,:])):
        A1=np.column_stack((env1,env2[:,co]))
        A1 =pd.DataFrame(A1)
        AEC[:,co]=A1[0].corr(A1[1])

    return AEC
```

Figura 3.10: calculoAEC prefiltrado optimizada.py

- **calculoPLI prefiltrado:**
  En esta función se han empleado los detalles anteriormente empleados. Para hacer una matriz de repeticiones de otra, en matlab **repmat**, se hace uso de la librería de python **numpy.matlib** la cuál tiene implementadas varias funciones de matlab.

```python
from scipy.signal import hilbert
import numpy as np


def calculoPLI_prefiltrado(signal = None,numChannels = None):
#CALCULOPLI Función que calcula el Phase Lag Index
#   Variables de entrada:
#   signal: señal sobre las que se va a calcular el parámetro. NtiempoxNchannels
#   Filtrar previamente al uso de esta función
#   Variables de salida:
#   PLI: Phase Lag index de las dos señales


    phaseSignal = np.angle(hilbert(signal));

# Optimizado por Saúl
    a=len(phaseSignal[:,1])
    b=numChannels*numChannels
    c=np.matlib.repmat(phaseSignal,numChannels,1)
    angles1 = np.reshape(c,(a,b),order="F")
    angles2 = np.matlib.repmat(phaseSignal,1,numChannels);

    PLI_vector = abs(np.mean(np.sign(np.sin(angles1-angles2)),axis=0))
    PLI = np.reshape(PLI_vector, [numChannels,numChannels])
    return PLI
```

Figura 3.11: calculoPLI prefiltrado.py

- **calculoMSCOH leakage:**
  Esta función calcula la magnitud al cuadrado de la de dos señales de entrada en las bandas de frecuencia convencionales (delta, zeta, alpha, beta-1,beta-2, gamma). Además de las diferencias del lenguaje, en cada una de las bandas de frecuencia, se han configurado los índices como matrices ya que sino el formato obtenido no permite obtener el resultado esperado en MSCOH. La función más problemática es mediante la cuál se obtiene la coherencia de las dos señales. En este punto aparece la limitación comentada anteriormente. Aunque la función **signal.coherence** está creada como la homóloga **mscohere** de Matlab, el resultado obtenido no es el mismo, generando una desviación de los resultados de gran magnitud. A pesar de varios intentos de distintas formas incluso empleando otra función específica para ese cometido, **matplotlib.mlab.cohere**,los resultados son muy diferentes.

```python
    MSCOH = np.empty((nBands,1,nChannels))
    MSCOH[:]=np.nan
# Calculamos MSCOH

# Ya que no tenemos Matlab 2017 tendremos que hacer un bucle y calcular la
# coherencia canal a canal. Se perderÃ¡ valioso tiempo.

    coherenciaTotal = np.empty((129,1,19))
    coherenciaTotal[:]=np.nan
    coherenciaTotal=np.transpose(coherenciaTotal)

# Calculamos de una vez la coherencia de un canal con todos los demÃ¡s

    canalRepetido=np.matlib.repmat(signal1[:,numChannel],nChannels,1)
    signal2=np.transpose(signal2)
    f,coherenciaTotal[:,0,:]= signal.coherence(canalRepetido, signal2[:,:], fs=fs, window='50', nperseg=None, noverlap=None, nfft=None, detrend='constant', axis=-1)

# Delta
    coherenciaTotal=np.transpose(coherenciaTotal)

    indDelta=(f>=1)&(f<=4)
    indDelta=indDelta*1
    indDelta=np.where(indDelta>0)
    indDelta=np.asarray(indDelta)
    MSCOH[0,0,:]=np.squeeze(np.mean(coherenciaTotal[indDelta[0,:],:,:],0))
    #Theta
    indTheta=(f>=4)&(f<=8)
    indTheta=indTheta*1
    indTheta=np.where(indTheta>0)
    indTheta=np.asarray(indTheta)
    MSCOH[1,0,:]=np.squeeze(np.mean(coherenciaTotal[indTheta[0,:],:,:],0))
    #Alpha
    indAlpha=(f>=8)&(f<=13)
    indAlpha=indAlpha*1
    indAlpha=np.where(indAlpha>0)
    indAlpha=np.asarray(indAlpha)
```

Figura 3.12: calculoMSCOH leakage optimizada.py

- **nonzeroadaptada:**

  Esta función ha sido diseñada exclusivamente para obtener el mismo resultado que en la funcion **nonzero** de MATLAB. A diferencia de la implementada en numpy, la de matlab recorre las columnas de arriba a abajo y de la izquierda hasta el final. En

python sin embargo, el recorrido lo hace fila a fila y muestra las posiciones de 0 a la longitud de la fila, volviendo a empezar en la siguiente. En matlab las posiciones se muestran en orden y cuando termina la columna la primera posición de la siguiente columna se indica como el número siguiente a la posición final de la anterior. Esta función recorre la matriz de la forma que se necesita y devuelve un vector con las posiciones en las que la matriz tiene un valor distinto de 0.

```
Created on Tue Jul 16 21:05:16 2019

@author: usuario
"""
import numpy as np

def noncero(signal1=None):

#devuelve un vector con las posiciones en las que los valores son distintos de 0,
#ordenandolo como un vector fila cuya composicion es el recorrido de cada columna de arriba a
#abajo
 re=np.empty((171))
 pos=0
 cont=0
 for b in range (19):
            for c in range (19):
                bo=signal1[c,b] != 0
                if (bo == True):
                    re[cont]=pos;
                    cont=cont+1
                pos=pos+1
 return re[:cont]
```

Figura 3.13: nonzeroadaptada.py

Capítulo 4

Resultados y discusión

4.1. Evaluación del toolbox

En este capítulo se irán comentando los resultados obtenidos en los tres cálculos de métodos de conectividad. Para poder evaluar el correcto funcionamiento se compararán los resultados de evaluar 75 sujetos pertenecientes a 3 grupos distintos, 25 sujetos de control, 25 sujetos con enfermedad de Alzheimer y 25 sujetos con deterioro cognitivo leve (Control, EA, DCL).

Los resultados se representarán en función de el parámetro κ para cada banda de frecuencias Delta, Zeta, Alfa Beta-1 y Beta-2 en sus versiones normalizadas y sin normalizar. La versión normalizada muestra una versión de los datos calculados mediante subrogaciones corregidos por la desviación típica estándar.

Se consiguen unos resultados similares a los originales con lo que el funcionamiento es correcto a excepción el cálculo en el método de conectividad MSCOH, el cuál debido a la imposibilidad de encontrar una función que calculase la coherencia entre dos señales de la misma forma que Matlab, ha generado unos valores muy lejanos a los esperados, por lo que no se tendrán en cuenta y será una futura línea de investigación. Estos resultados se representan mediante un script específico diseñado en matlab que muestra el parámetro κ respecto de distintas bandas de frecuencia, el cual se muestra en las siguientes imágenes:

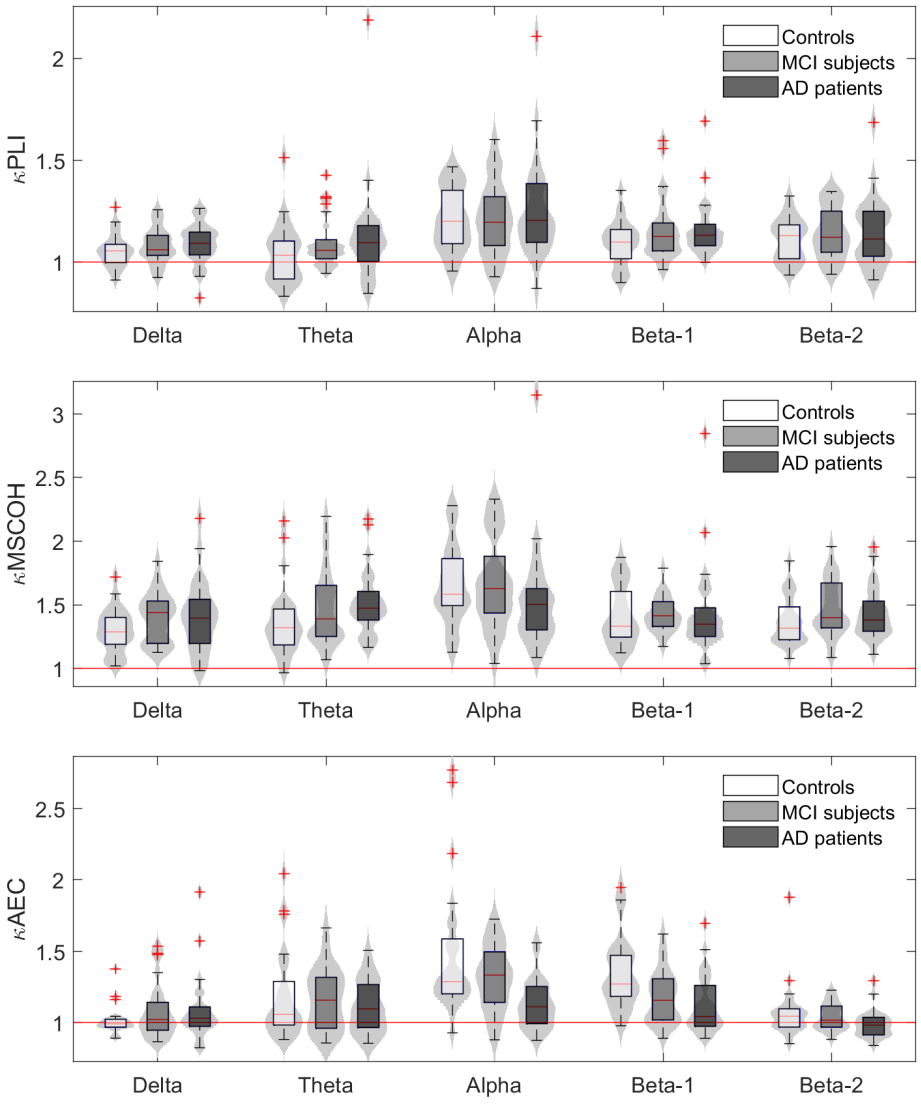

Figura 4.1: Gráficos que muestran los valores promedio normalizados de PLI, MSCOH y AEC en cada banda de frecuencias.

Los valores mostrados en la figura 4.1 están corregidos mediante desviación típica estandar (STD). Como se puede observar la mayoría de valores se encuentran por encima de la línea roja, lo cuál indica que hay valores mayores para los datos originales que para las réplicas subrogadas. Se obtienen valores estadísticamente significativos entre grupos ($p < 0{:}05$, prueba de Kruskal-Wallis con corrección de FDR) en las bandas alpha ($p < 0{:}0001$), beta-1 ($p < 0{:}0001$) y beta-2 ($p = 0{:}0399$) para MSCOH y PLI, sin embargo para AEC las diferencias significativas se encuentran en las bandas alpha ($p < 0{:}0001$) y beta-1 ($p < 0{:}0001$). Los 3 métodos de conectividad muestran diferencias significativas entre grupos en este caso. Tal y como sucede en Matlab los valores no normalizados muestran distribuciones similares a las de los normalizados como se muestra en la figura 4.2.

En la sección 4.3 se compararán estos resultados con los obtenidos con matlab para observar las diferencias.

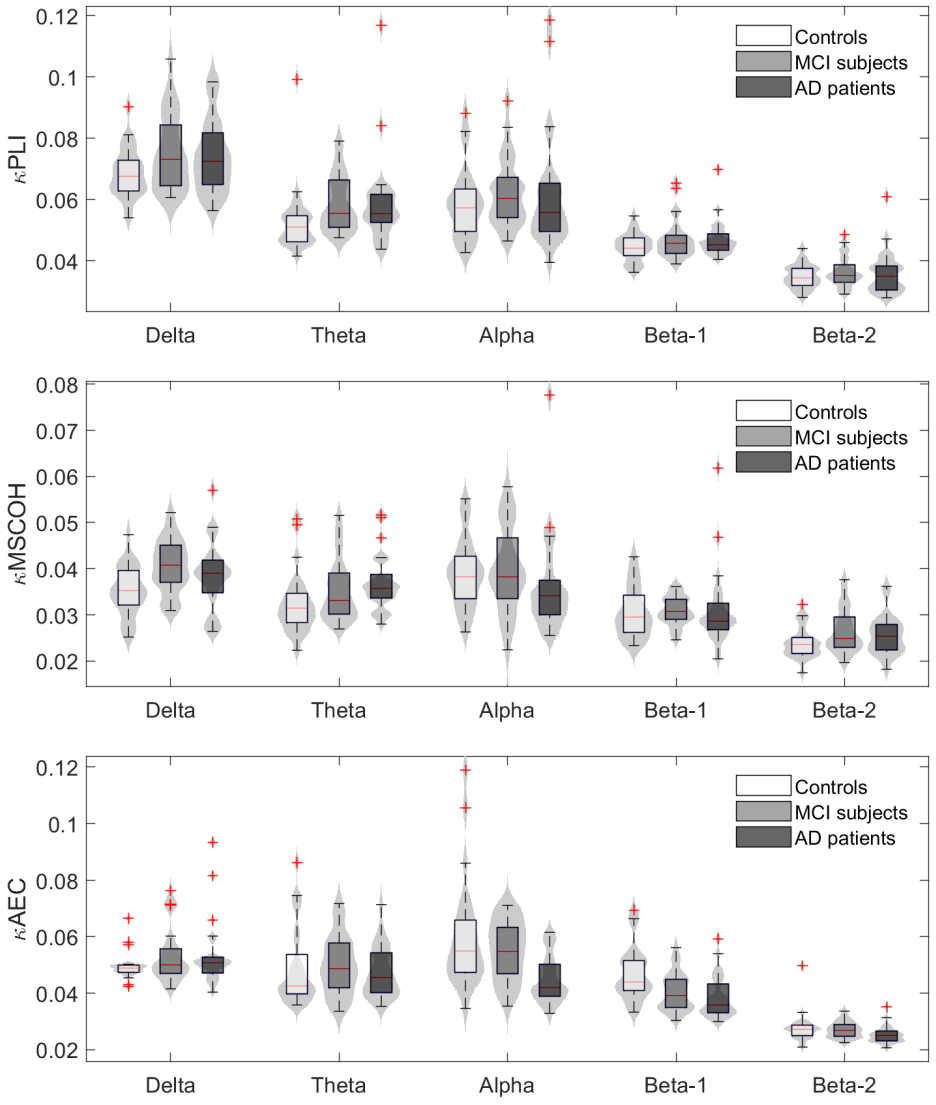

Figura 4.2: Gráficos que muestran los valores promedio sin normalizar de PLI, MSCOH y AEC en cada banda de frecuencias.

Se puede afirmar que se obtienen resultados muy similares a los originales, a excepción de MSCOH. Por lo tanto se han obtenido resultados favorables en dos de los tres cálculos.

4.2. Análisis de rendimiento

Para analizar el rendimiento del Toolbox, se ha realizado el cálculo de 75 sujetos como se comentaba en la sección 4.1 para cada una de las medidas de conectividad, tanto en MATLAB como en SPYDER. Se ha realizado el cálculo con varios ordenadores con las mismas características de RAM obteniendo resultados de rendimiento similares.

Los parámetros con los que se estudiará el rendimiento del toolbox en python son el porcentaje de memoria que consume el proceso de cálculo y el tiempo total empleado

para ello. Para comparar estos resultados se indican en las siguientes tablas. Para medir el porcentaje de memoria que consume el código se hará uso de la herramienta de consumo de memoria de SPYDER y para obtener el tiempo total de computación se han añadido distintos fragmentos de código que nos darán el tiempo total. En PYTHON la librería **time** nos indica esos valores y en MATLAB se ha empleado el código **tic-toc**.

A continuación se muestra una tabla con los valores del tiempo de computación de los 75 sujetos con los distintos softwares y lenguajes de programación:

	Método de cálculo		
Software	*MSCOH*	*PLI*	*AEC*
Matlab	6 días 7h 15'	1 día 7h 20'	3 días 21h 45'
Spyder	1 día 5h 5'	2 días 1h 21'	5 días 6h 13'

Tabla 4.1: Tiempos totales de procesamiento.

Tras obtener estos resultados se puede afirmar que en cómputo general con el código en python se ahorra tiempo, a pesar de que el cálculo del método de conectividad PLI y el cálculo con el método de conectividad AEC tardan significativamente más. El cálculo en el cuál se aprecia una mejora significativa es MSCOH aunque no de el resultado esperado. Esto puede ser debido al número de operaciones o al tipo, ya que en MSCOH se realizan los mismos cálculos que en código original. Por lo tanto se puede afirmar que los resultados obtenidos son favorables en cuanto al tiempo de cálculo de los 75 sujetos.

En la siguiente tabla se muestran los valores de consumo de memoria medio de cada uno de los procesos con cada uno de los softwares y lenguajes de programación:

	Método de cálculo		
Software	*MSCOH*	*PLI*	*AEC*
Matlab	61 %	86 %	90 %
Spyder	41 %	81 %	86 %

Tabla 4.2: Porcentaje de consumo de memoria medio.

Realmente aquí se observa la diferencia de rendimiento entre un código y otro. En todos los cálculos Spyder consume menos memoria, entre un 4 y un 10 % menos dependiendo del método de conectividad. Estos resultados son consumo de memoria medio, en el caso de Matlab durante la última parte del cálculo aumentaba el porcentaje en un 2-3 %, sin embargo Spyder se mantiene durante todo el cálculo con un porcentaje estable.

4.3. Comparación de patrones relacionados con el DCL y la EA

En esta sección se van a comparar los resultados obtenidos en este trabajo con los obtenidos en otras investigaciones para comprender la validez de los resultados.Principalmente

las diferencias más notables entre sujetos se encuentran en el cálculo κAEC por lo que se centrará la comparación en dichos resultados.

En el anterior trabajo desarrollado en Matlab [25], se encontraron diferencias estadísticamente significativas entre los grupos κAEC en las bandas alfa, beta-1 y beta-2, aunque en esta última en menor medida. No se encontraron diferencias significativas en κPLI o en κMSCOH tal y como ocurre en este trabajo. Los resultados para κAEC sugieren que la dFC se ve afectada por EA y DCL, con una pérdida de estabilidad en el acoplamiento presente en los grupos. Si se entiende la EA como un síndrome de desconexión neuronal, es de esperar que los patrones de conectividad de la EA se verán afectados e irán empeorando a lo largo de la progresión de la enfermedad [3]. Este es el caso, como se puede observar en la figura 4.9, ya que los valores promedio de kAEC muestran una clara tendencia a la reducción del dFC en los sujetos con DCL, así como una disminución más pronunciada de κAEC en los pacientes con EA en las bandas alfa, beta-1 y beta-2 [25]. Sin embargo, comparando con los resultados obtenidos en Python se puede observar en la figura 4.10 las variaciones notables entre grupos en las bandas comentadas se mantienen pero con una diferencia, el valor de datos de las señales subrogadas se equipara en mayor medida a los de las señales originales, a excepción del caso PLI que ocurre el contrario. Al comparar entre los resultados entre Matlab y Spyder, figuras 4.9 y 4.10, se puede ver como las diferencias entre sujetos se mantienen entre un cálculo y otro aunque el valor de la potencia normalizado se reduce en términos generales para Spyder.

En el cálculo MSCOH se obtiene una diferencia muy significativa al tener el código incompleto con una función que no da los resultados esperados por lo que estos datos no los podemos tener en cuenta ni comparar. En PLI no se observan diferencias sustanciales entre los sujetos en cada banda, por lo que no se analizarán dichos datos en profundidad.

Para κAEC, en Matlab, las mayores diferencias en dFC para la banda alfa sólo han encontraron entre los controles sanos y los pacientes con EA del mismo modo que en Spyder, en las bandas Alpha, Beta-1 y Beta-2. Tal y como se afirma en otros estudios, la EA está asociada con una disminución en la potencia en banda alfa [5] [26], así como con una pérdida de conectividad funcional [31] [22]. Si observamos las imágenes de los resultados podemos ver como esta característica está marcada sobre todo en κAEC, en el resto de cálculos no se puede afirmar ya que la potencia en la banda alfa en sujetos con EA es igual o mayor que en los sujetos de control.

Dichos resultados sugieren que también está presente en la EA una disminución del dFC . Por lo tanto, no sólo se reduce el FC en la EA, sino que se ralentiza, evolucionando a un ritmo más lento que los sujetos de control. Esto podría significar que la hipótesis de desconexión de la sFC podría extenderse al dominio dFC [22]. Dado que en esta banda sólo se encontraron tendencias no estadísticamente significativas hacia menor dFC en pacientes con DCL en comparación con los controles, esto sugiere que la dFC del DCL en la banda alfa no es tan anormal como en la EA. Esto podría implicar que un sistema FC menos dinámico es un signo de EA que no está presente en el DCL, a diferencia de la desconexión funcional estática que ya está presente en el DCL y que se vuelve más

marcada en la EA [5] [33].

También se han encontrado diferencias estadísticamente significativas entre los controles y los pacientes con EA en la banda beta-1, y los controles muestran una mayor AEC normalizada entre todos los pares regionales [25]. Los datos calculados con Matlab de los 75 sujetos, figura 4.11, no muestran diferencias estadísticamente significativas, así como en κPLI ni en Matlab ni en Spyder se observan mayores diferencias.

Se observaron algunas tendencias hacia una menor normalización de la κAEC en sujetos con DCL en comparación con controles sanos, lo que sugiere que el beta-1 dFC podría deteriorarse más temprano en el proceso de envejecimiento (DCL-EA) que el alfa dFC. Esto ocurre en el resto de cálculos realizados con Spyder pero es más notable en el κAEC. Al igual que en la banda Alfa, las diferencias estadísticamente significativas en la variabilidad fueron más marcadas entre los pacientes con DCL y los pacientes con EA que entre los controles y los pacientes con DCL, lo que sugiere además que la dFC podría alterarse en una etapa posterior en el continuo DCL-EA [25]. Este resultado se mantiene en los cálculos con Spyder solo que en función del cálculo se alejan en conjunto de la normalización como en κPLI o se acercan como en κAEC.

Estudios previos encontraron alteraciones tempranas del sFC en la banda beta. En particular, Gómez et al. encontraron decrementos inter e intrahemisféricos en la conectividad de Granger Causality para sujetos con DCL en comparación con los controles [17]. Babiloni et al. encontraron que la dirección parietal a frontal de la información ux es más débil en sujetos con DCL y pacientes con EA en comparación con los controles [2]. Este estudio [25], apoya así la noción de que el DCL es una etapa prodrómica de la EA. La banda beta ha sido tradicionalmente asociada con las funciones sensoriomotoras. Sin embargo, recientemente ha estado implicada en la preservación del estado actual del cerebro [30]. Es una de las bandas de frecuencia menos estudiadas en lo que se refiere a la EA y al DCL [18], pero algunos estudios sugieren que podría tener una importancia especial en las primeras etapas de la EA, con una pérdida de sincronización en la banda beta que refleja la patología especifica en la EA, en lugar de ser simplemente debida al envejecimiento [9]. Además, dado que una de las alteraciones más documentadas en las oscilaciones del EEG en la EA y el DCL es un desplazamiento de la potencia hacia frecuencias más bajas (Zeta y Delta) [34], podría esperarse una disminución de la estabilidad de la conectividad del EEG en las bandas de frecuencias más altas. De hecho, se encontró una correlación positiva entre la AEC normalizada y el poder relativo en pacientes con EA y especialmente en pacientes con DCL en la banda beta-1 , pero no se encontró ninguna correlación para los controles. De Haan et al. encontraron previamente una correlación entre el sFC y la potencia media, utilizando modelos computacionales [11].

Los resultados obtenidos en el estudio [25] podrían indicar que esta correlación entre potencia media y conectividad podría extenderse a dFC, tal vez sugiriendo que la alta correlación entre potencia relativa y kAEC en pacientes con DCL en la banda beta-1 es patológica y está relacionada con altos niveles de actividad que podrían conducir a daño neuronal [11]. La reducción de dFC encontrada en pacientes con EA fue especialmente prevalente en las conexiones entre las regiones izquierda-temporal, frontal y central. En varios estudios basados en el MEG [12] se han encontrado reducciones del sFC en pacientes con EA en estas regiones. En particular, las conexiones sFC de largo alcance entre

el hemisferio izquierdo y otras regiones parecen más afectadas por la EA según estos estudios [12], lo que concuerda con nuestros resultados en la banda beta-1.

Los resultados obtenidos en [25] sugieren que este podría ser también el caso de los dFC beta-1, ya que las conexiones que involucran al hemisferio izquierdo se redujeron significativamente en los pacientes con EA en comparación con los pacientes con DCL. Esto podría sugerir que el hemisferio izquierdo está más conectado que otras regiones durante el desarrollo de la EA. De hecho, un estudio de estado de reposo cerrado basado en MEG encontró una reducción de la desviación beta estándar (DE) de la actividad local en pacientes con EA en comparación con los controles de ancianos en el hemisferio izquierdo [22]. Esto puede indicar que una reducción en la activación local puede estar relacionada con una pérdida de dFC entre las áreas afectadas.

Basándonos en estos resultados se podría afirmar que obtenemos valores similares a los obtenidos en el estudio anterior [25] por lo que cumple en gran medida con el objetivo establecido. Comparando los datos con mayor fiabilidad, κAEC y κPLI con los obtenidos en Matlab, se podría decir que los valores de las señales subrogadas afectan de diferente forma en cada cálculo. En el caso de κAEC con mayores en Spyder que en Matlab, sin embargo en el caso de κPLI ocurre lo contrario.

Este trabajo ha mejorado en cómputo general las capacidades de computación tanto en gasto computacional como en tiempo invertido de los métodos de conectividad que permiten estudiar la influencia del estado DCL en el desarrollo de la enfermedad de Alzheimer. Además se realiza con un sofware de licencia libre que consume menos recursos.

Si se observan los resultados uno a uno, en términos temporales sólo obtenemos mejor rendimiento en el cálculo MSCOH, que , aunque no da como resultado los valores esperados, realiza las mismas operaciones que el cálculo original. Por otro lado en términos de consumo de memoria Spyder es más estable con un consumo menor aunque para ciertos cálculos obtenemos peores resultados temporales.

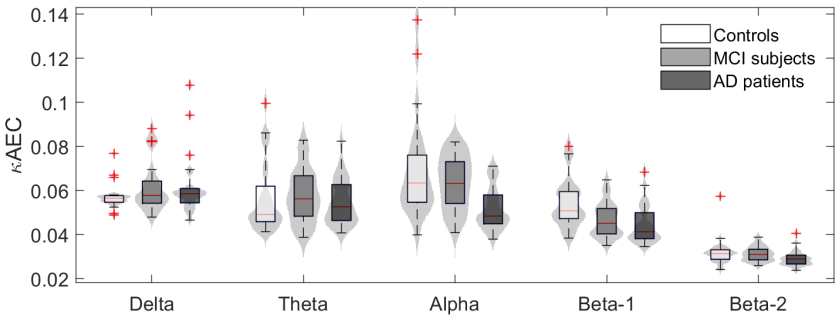

Figura 4.3: Gráfica que muestra los valores promedio sin normalizar de AEC en cada banda de frecuencias calculada con Matlab.

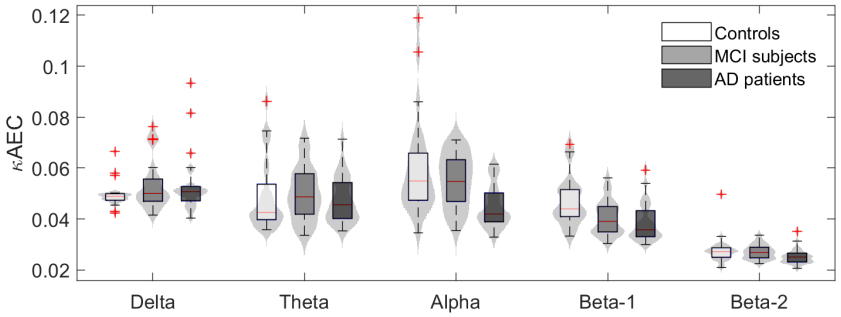

Figura 4.4: Gráfica que muestra los valores promedio sin normalizar de AEC en cada banda de frecuencias calculada con Spyder.

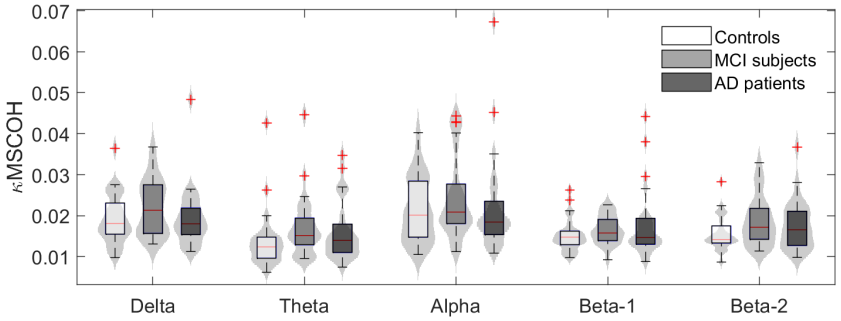

Figura 4.5: Gráfica que muestra los valores promedio sin normalizar de MSCOH en cada banda de frecuencias calculada con Matlab.

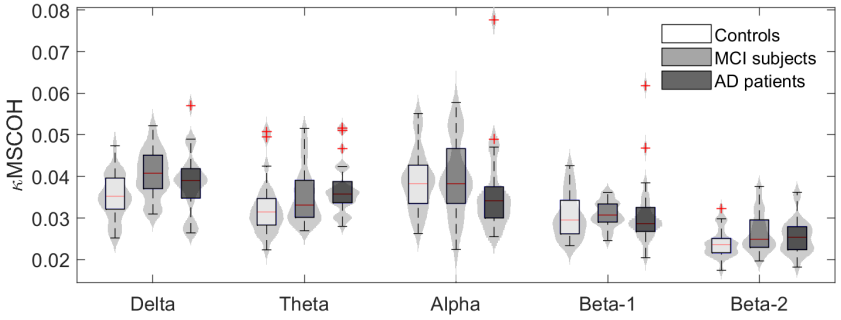

Figura 4.6: Gráfica que muestra los valores promedio sin normalizar de MSCOH en cada banda de frecuencias calculada con Spyder.

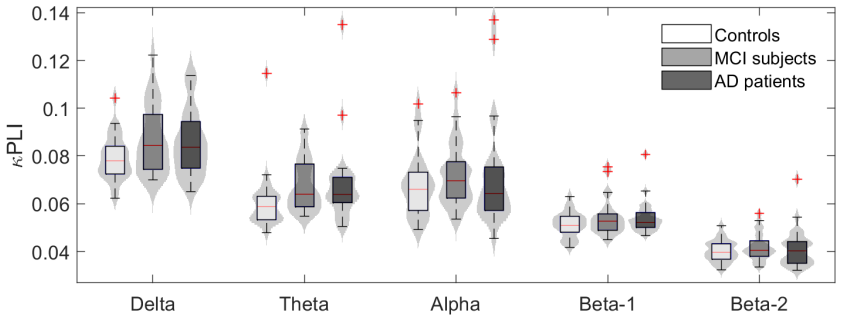

Figura 4.7: Gráfica que muestra los valores promedio sin normalizar de PLI en cada banda de frecuencias calculada con Matlab.

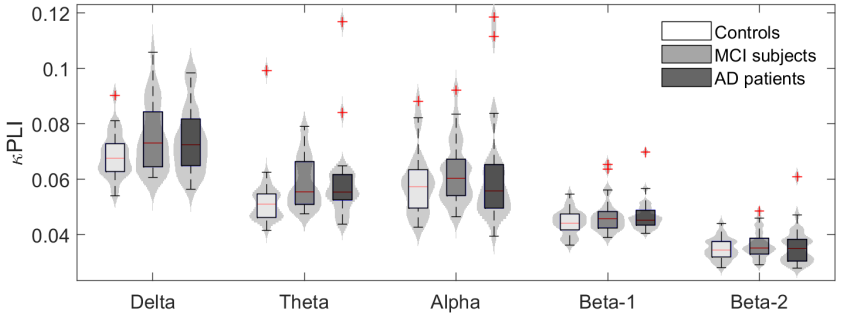

Figura 4.8: Gráfica que muestra los valores promedio sin normalizar de PLI en cada banda de frecuencias calculada con Spyder.

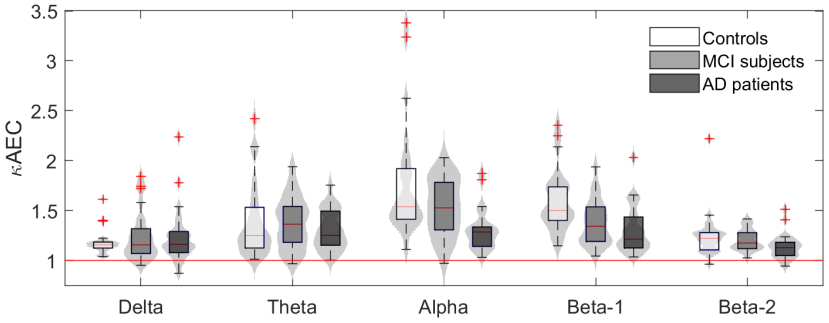

Figura 4.9: Gráfica que muestra los valores promedio normalizados de AEC en cada banda de frecuencias calculada con Matlab.

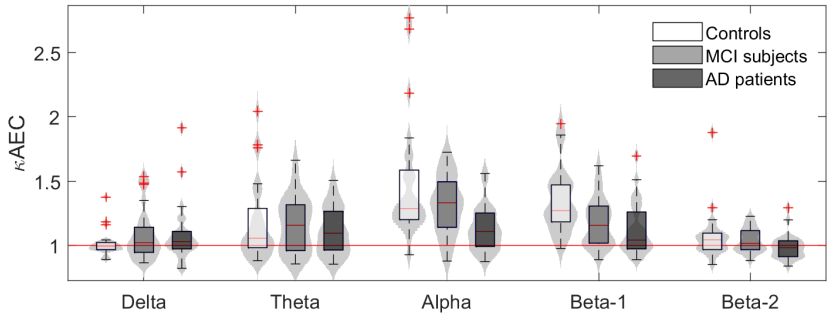

Figura 4.10: Gráfica que muestra los valores promedio normalizados de AEC en cada banda de frecuencias calculada con Spyder.

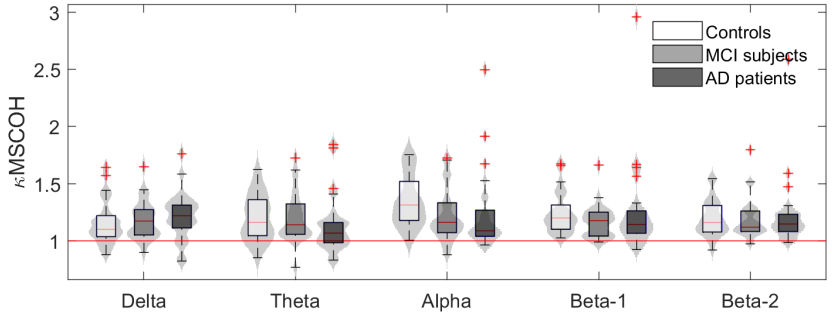

Figura 4.11: Gráfica que muestra los valores promedio normalizados de MSCOH en cada banda de frecuencias calculada con Matlab.

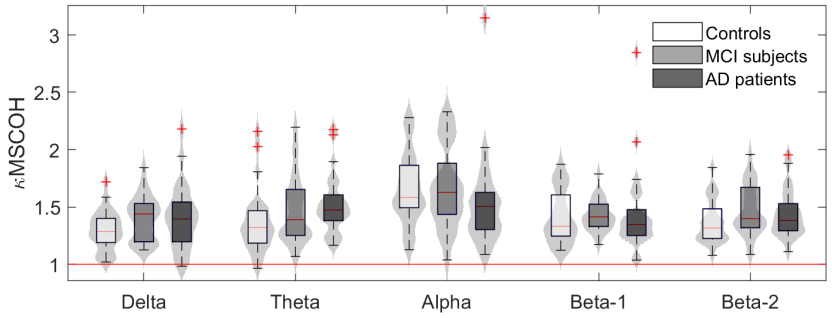

Figura 4.12: Gráfica que muestra los valores promedio normalizados de MSCOH en cada banda de frecuencias calculada con Spyder.

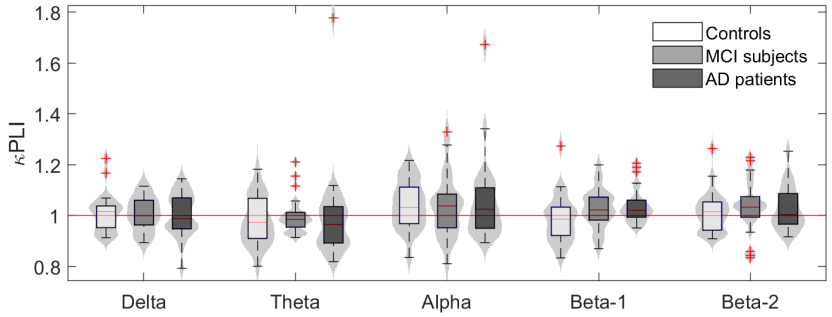

Figura 4.13: Gráfica que muestra los valores promedio normalizados de PLI en cada banda de frecuencias calculada con Matlab.

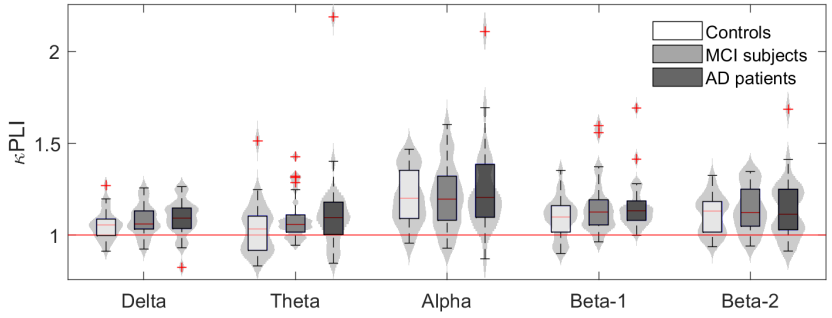

Figura 4.14: Gráfica que muestra los valores promedio normalizados de PLI en cada banda de frecuencias calculada con Spyder.

4.4. Limitaciones

Las principales limitaciones encontradas es la gran diferencia de resultados entre funciones que son análogas. Esto es debido a que aunque realice la misma operación la lectura de la matriz se hace en distinto orden, por otro lado si la finalidad de dicha función no era la misma que la de su análoga en matlab, las operaciones que realiza no son exactamente iguales. En el caso de algunas funciones como **nonzero**, la cuál muestra en qué posiciones la matriz tiene valores distintos de 0, se ha tenido que hacer una versión totalmente adaptada para obtener el mismo resultado que en matlab.

Por otro lado, cuando tienes un error en el código o las variables generadas no tienen el formato esperado para algunas funciones, spyder, el software empleado para la compilación de Python, no avisa o muestra un error difuso, por lo que hay que ir ejecutando línea a línea del código, comparándolo con Matlab, para ver si realmente las funciones dan los resultados esperados o si el formato de las variables es válido para esas funciones.

Otra limitación encontrada, la cuál ha alterado los resultados de este trabajo, es la función análoga de **mscohere** de MATLAB. No ha sido posible encontrar una función con

la que obtener los mismos resultados. Esta función solo afecta al cálculo de conectividad MSCOH produciendo un resultado muy lejano del original en MATLAB.

Capítulo 5

Conclusiones y líneas futuras

5.1. Grado de cumplimiento de los objetivos

En este trabajo se han estudiado los métodos de conectividad para poder detectar la enfermedad de Alzheimer en una fase temprana. Para ello se ha realizado una traducción de los cálculos hasta ahora empleados a un lenguaje con un rendimiento mejor. Para ello se establecieron los objetivos a seguir en el capítulo 1, de los cuales se han cumplido los siguientes:

- Se ha realizado la lectura de los documentos bibliográficos oportunos para profundizar en los conocimientos necesarios sobre la EA y sus métodos de análisis, comprendiendo las bases médicas y teóricas necesarias para el desarrollo de este trabajo.
- Se ha estudiado con profundidad los métodos de análisis de conectividad implementados en MATLAB y sus resultados, para comprender línea a línea la función que desempeña el código respecto a los cálculos de conectividad.
- Se ha desarrollado un toolbox en PYTHON realizando una traducción del código anterior y analizando todas las funciones y librerías que lo componen para que los resultados sean lo más parecidos posible a el código implementado en MATLAB.
- Se han analizado los resultados del toolbox de PYTHON comparandolos con MATLAB y se ha hecho un estudio del rendimiento de los mismos.
- De este trabajo se han obtenido varias conclusiones, que aportarán información a las líneas de investigación futuras.

5.2. Conclusiones

Tras analizar los resultados expuestos en el capítulo 4, se han extraído las siguientes conclusiones:

- Se ha demostrado que se puede obtener mayor rendimiento realizando un cambio de lenguaje de programación y de software de computación.
- Se ha demostrado que únicamente en términos generales se consigue una mejora en el tiempo de computación y en términos particulares en consumo de memoria.
- Aunque python dispone de librerías dedicadas al código de matlab, la traducción requiere de un proceso más detallado y complejo del código.
- No todas las funciones creadas en python homólogamente a matlab dan los mismos resultados que en matlab.
- Se obtienen resultados diferentes a pesar de realizar una traducción literal de un código a otro.
- Se pueden obtener buenos resultados con softwares de licencia libre obteniendo buenas cifras de rendimiento.

5.3. Líneas futuras

Aunque se han obtenido buenos resultados en el presente trabajo, uno de los 3 cálculos de conectividad no ha producido un resultado satisfactorio debido a una función en concreto. Una línea futura de investigación interesante sería conseguir adaptar dicha función para obtener unos resultados óptimos o en su defecto crear una función para obtener dichos resultados. Posteriormente se podrían comparar todos los resultados meticulosamente para comprobar cuál de los dos lenguajes ofrece unos mejores resultados. Otra posible línea futura podría ser modificar las funciones actuales realizadas mediante bucles y cambiarlas por operaciones vectoriales, lo cuál posiblemente reduzca el tiempo de procesamiento.

Bibliografía

[1] A.M. BASTOS, E. A. A tutorial review of functional connectivity analysis methods and their interpretational pitfalls. *Frontiers in Systems Neuroscience* (2016).

[2] BABILONI C, E. A. Directionality of eeg synchronization in alzheimer's disease subjects. *Neurobiology of Aging 30* (2009), 93–102.

[3] BOKDE A L W, E. A. Assessing neuronal networks: Understanding alzheimer's disease. *Progress in Neurobiology 89* (2009), 125–133.

[4] C. BABILONI, E. A. Resting state eyes-closed cortical rhythms in patients with locked-in-syndrome: An eeg study. *Clinical Neurophysiology 121* (2010), 1816–1824.

[5] C. BABILONI, E. A. Brain neural synchronization and functional coupling in alzheimer's disease as revealed by resting state eeg rhythms. 88–102.

[6] C. BABILONI, E. A. Abnormalities of resting state cortical eeg rhythms in subjects with mild cognitive impairment due to alzheimer's and lewy body diseases. *Journal of Alzheimer's Disease 62* (2018).

[7] C. BABILONI, E. A. Abnormalities of resting-state functional cortical connectivity in patients with dementia due to alzheimer's and lewy body diseases: an eeg study. *Neurobiology of Aging 65* (2018), 18–40.

[8] C. BABILONI, E. A. Levodopa may affect cortical excitability in parkinson's disease patients with cognitive deficits as revealed by reduced activity of cortical sources of resting state electroencephalographic rhythms. *Elservier, Neurobiology of Aging 73* (2019), 9–20.

[9] C.J. STAM, E. A. The organization of physiological brain networks. *Clinical Neurophysiology 123* (2012), 1067–1087.

[10] CRESPO, J. P. *Análisis tiempo-frecuencia de la actividad magnetoencefalográfica espontánea en la enfermedad de Alzheimer*. PhD thesis, Universidade de Valladolid, Spain, 2008.

[11] DE HAAN W, E. A. Activity dependent degeneration explains hub vulnerability in alzheimer's disease. *PLoS Computational Biology 8* (2012).

[12] ENGELS M M, E. A. Alzheimer's disease: The state of the art in resting-state magnetoencephalography. *Clinical Neurophysiology 128* (2017), 1426–1437.

[13] F. MIRAGLIA (DR.), E. A. Searching for signs of aging and dementia in eeg through network analysis. *Behavioural Brain Research 317* (2017), 292–300.

[14] G. DECO, E. A. Resting brains never rest: computational insights into potential cognitive architectures. *Trends in Neurosciences 36* (2013), 268–274.

[15] G.C. O'NEILL, E. A. Dynamics of large-scale electrophysiological networks: A technical review. *NeuroImage 180* (2018), 559–576.

[16] G.L. COLCLOUGH, E. A. How reliable are meg resting-state connectivity metrics? *NeuroImage 138* (2016), 284–293.

[17] GÓMEZ C, E. A. Alterations of effective connectivity patterns in mild cognitive impairment: an meg study. *Journal of Alzheimer's Disease Preprint* (2017), 1–12.

[18] GUNTEKIN B, E. A. Beta oscillatory responses in healthy subjects and subjects with mild cognitive impairment. *NeuroImage: Clinical 3* (2013), 39–46.

[19] J. DAUWELS, E. A. Diagnosis of alzheimer's disease from eeg signals: Where are we standing? *Curr Alzheimer Res 6* (2010), 487–505.

[20] J. THEDER, E. A. Testing for nonlinearity in time series: the method of surrogate data. *Physica D 58* (1992), 77–94.

[21] JONES D T, E. A. Non-stationarity in the "resting brain's" modular architecture. *PLoS ONE 7* (2012).

[22] KOELEWIJN L, E. A. Alzheimer's disease disrupts alpha and beta-band resting-state oscillatory network connectivity. *Clinical Neurophysiology 128* (2017), 2347–2357.

[23] M. FRASCHINI, E. A. The effect of epoch length on estimated eeg functional connectivity and brain network organisation. *Journal of Neural Engineering 13* (2016).

[24] M.X.COHEN. Where does eeg come from and what does it mean? *Trends in neurosciences 40* (2017), 208–218.

[25] P. NÚÑEZ, E. A. Characterizing the fast uctuations of dynamic resting-state electrophysiological functional connectivity: Reduced neuronal coupling variability in mild cognitive impairment and dementia due to alzheimer's disease. *Journal of Neural Engineering* (2019), 3–6.

[26] POZA J, E. A. Extraction of spectral based measures from meg background oscillations in alzheimer's disease. *Medical Engineering and Physics 29* (2007), 1073–1083.

[27] R. HINDRIKS, E. A. Can sliding-window correlations reveal dynamic functional connectivity in resting-state fmri? *NeuroImage 127* (2016), 242–256.

[28] R.M. HUTCHISON, E. A. Dynamic functional connectivity: Promise, issues, and interpretations. *NeuroImage 80* (2013), 360–378.

[29] ROACH B J, E. A. Event-related eeg time-frequency analysis: An overview of measures and an analysis of early gamma band phase locking in schizophrenia. *Schizophrenia Bulletin 34* (2008), 907–926.

[30] SPITZER B, E. A. Beyond the status quo: A role for beta oscillations in endogenous content (re)activation. *Eneuro 4* (2017), 93–102.

[31] STAM C J, E. A. Eeg synchronization in mild cognitive impairment and alzheimer's disease. *Acta Neurologica Scandinavica*.

[32] STAM C J, E. A. Phase lag index: Assessment of functional connectivity from multi channel eeg and meg with diminished bias from common sources. *Human Brain Mapping 28* (2007), 1178–1193.

[33] TÓTH B, E. A. Eeg network connectivity changes in mild cognitive impairment — preliminary results. *International Journal of Psychophysiology 92* (2014), 1–7.

[34] UHLHAAS P J, E. A. Neural synchrony in brain disorders: Relevance for cognitive dysfunctions and pathophysiology. *Neuron 52* (2006), 155–168.