UNIVERSIDAD
DE GRANADA

HERRAMIENTAS Y ESTRATEGIAS DE PARAMETER TUNING APLICADAS A LA NEUROCIENCIA COMPUTACIONAL

ANDRÉS ARCO LÓPEZ

Trabajo Fin de Grado

Grado en Ingeniería Informática

Tutores

Pablo Martínez Cañada

FACULTAD DE CIENCIAS

E.T.S. INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, a 16 de noviembre de 2022

Herramientas y estrategias de parameter tuning aplicadas a la neurociencia computacional

Andrés Arco López

Andrés Arco López. *Herramientas y estrategias de parameter tuning aplicadas a la neurociencia computacional*.

Trabajo de fin de Grado. Curso académico 2022-2023.

Responsables de tutorización

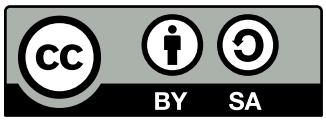
Pablo Martínez Cañada
Departamento de Ingeniería de Computadores, Automática y Robótica

Escuela Técnica Superior de Ingeniería Informática y Telecomunicación

Facultad de Ciencias

Grado en Ingeniería Informática

Universidad de Granada



Declaración de originalidad

D. Andrés Arco López

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2022-2023, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 16 de noviembre de 2022

Fdo: Andrés Arco López

Gracias en especial a mi tutor D. Pablo Martínez Cañada por estar dispuesto a ayudar en todo momento durante el desarrollo del trabajo a pesar de que este campo de machine learning (ML) era algo relativamente nuevo para mí y a mi familia y amigos por estar ahí siempre y apoyarme en los momentos más complicados a cerrar este ciclo de mi vida

RESUMEN

Actualmente conocemos menos del diez por ciento del funcionamiento del cerebro. La Neurociencia Computacional permite construir modelos de simulación del cerebro, que incluyen cientos de miles de neuronas, y ayudar así a neurocientíficos y médicos a descifrar la funcionalidad del cerebro.

En este proyecto se ha simulado un modelo de cerebro ampliamente usado para describir la dinámica neuronal del circuito cortical a nivel de micro- y meso-escala. A partir de la simulación de actividad neuronal, se ha generado la señal del electroencefalograma (EEG), una de las señales no-invasivas más conocida en el ámbito clínico.

La pregunta científica que hemos abordado en este proyecto ha sido si podemos estimar los parámetros del modelo (por ejemplo, el cociente entre excitación, E, e inhibición, I: E/I) que han dado lugar a las distintas propiedades de la señal del EEG. Para ello, hemos desarrollado herramientas de machine learning (ML) que permiten encontrar automáticamente las regiones de interés del espacio de parámetros del modelo cortical y que se relacionan con cambios en la señal del EEG. Estas herramientas podrán usarse para ayudar en el diagnóstico clínico de condiciones médicas y descodificar los parámetros del circuito cortical que son alterados.

Palabras clave: modelo neuronal, encefalograma, EEG, neuronas excitadoras, neuronas inhibidoras, NEST, regresión lineal, Ridge, K-NNeighbors, red neuronal.

ABSTRACT

We currently know less than ten percent of how the brain works. Computational Neuroscience makes it possible to build simulation models of the brain, which include hundreds of thousands of neurons, that are very helpful for neuroscientists and doctors to understand the functionality of the brain.

In this project, a widely used brain model has been simulated to describe the neural dynamics of the cortical circuit at the micro- and meso-scale level. From the simulation of neuronal activity, the electroencephalogram (EEG) signal has been generated, one of the most well-known non-invasive signals in the clinical field.

The scientific question that we have addressed in this project has been whether we can estimate the parameters of the model (for example, the ratio between excitation, E, and inhibition, I: E/I) that have given rise to the different properties of the signal of the EEG. To do this, we have developed machine learning (ML) tools that automatically find the regions of interest in the parameter space of the cortical model and that are related to changes in the EEG signal. These tools may be used to aid in the clinical diagnosis of medical conditions and decode the cortical circuitry parameters that are altered.

Keywords: neuronal model, encephalogram, EEG, excitatory neurons, inhibitory neurons, NEST, linear regression, Ridge, K-NNeighbors, neural network.

ÍNDICE GENERAL

1. INTRODUCCIÓN 11
   1.1. Contexto histórico . . . 11
2. CONCEPTOS BÁSCIOS 13
   2.1. Introduccion . . . 13
   2.2. Definición de neurociencia . . . 13
   2.3. Definición de electroencefalograma . . . 13
   2.4. Neuronas y tipos de neuronas . . . 14
      2.4.1. Estructura . . . 14
      2.4.2. Tipos de neuronas . . . 14
3. MODELO DE RED DE NEURONAS LIF (SPIKES) 16
4. PARALELIZACIÓN DE LAS SIMULACIONES EN UN SERVIDOR 18
   4.1. Análisis de los datos . . . 20
      4.1.1. Resultados de una simulación con los parámetros estándar . . . 21
      4.1.2. Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.005) . . . 22
      4.1.3. Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.200) . . . 23
      4.1.4. Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.450) . . . 24
      4.1.5. Resultados de una simulación con parámetro inh_exc_recurrent alterado (-0.25) . . . 25
      4.1.6. Resultados de una simulación con parámetro inh_exc_recurrent alterado (-4.50) . . . 26
      4.1.7. Resultados de una simulación con parámetro inh_exc_recurrent alterado (-8.00) . . . 27
5. CONCLUSIONES DEL ANÁLISIS DE LOS DATOS 29
6. ESTIMACIÓN DE PARÁMETROS DEL CIRCUITO NEURONAL A PARTIR DE LA SEÑAL DEL ENCEFALOGRAMA 30
   6.1. Algoritmo de regresión lineal simple . . . 31
      6.1.1. Machine Learning con algoritmo de regresión lineal simple . . . 33
      6.1.2. Machine Learning con algoritmo de regresión lineal simple aplicando autocorrelación . . . 34
   6.2. Algoritmo de regresión lineal de Ridge . . . 34
      6.2.1. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 0.5 . . . 36
      6.2.2. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 0.5 aplicando autocorrelación . . . 37

6.2.3. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1 . . . 38
6.2.4. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1 aplicando autocorrelación . . . 39
6.2.5. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1.5 . . . 40
6.2.6. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1.5 aplicando autocorrelación . . . 41
6.2.7. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 2 . . . 42
6.2.8. Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 2 aplicando autocorrelación . . . 43
6.2.9. Evolución de los resultados variando alfa para regresión lineal de Ridge . . . 44
6.3. Machine Learning con algoritmos no lineales . . . 45
6.3.1. Machine Learning con algoritmo no lineal de K-Nearest neighbour . . . 45
6.3.2. Machine Learning con algoritmo no lineal de K-Nearest neighbour aplicando autocorrelación . . . 47
6.3.3. Machine Learning con algoritmo no lineal - red neuronal . . . 48
6.3.4. Machine Learning con algoritmo no lineal - red neuronal aplicando autocorrelación . . . 50
6.4. Comparación de los resultados de errores cuadráticos medios . . . 51
7. CONCLUSIONES 58
7.1. Relevancia del TFG para la neurociencia y la medicina . . . 58
8. TRABAJO FUTURO 60

BIBLIOGRAFÍA 61

1

INTRODUCCIÓN

1.1 CONTEXTO HISTÓRICO

A finales del siglo XVIII se descubrió la actividad eléctrica en el sistema nervioso, dando pie a los análisis en el campo de la electrofisiología de las neuronas.

Con el desarrollo del microscopio y de las técnicas de fijación y tinción de los tejidos, la Anatomía del sistema nervioso experimentó un notable avance que culminó con la obra genial de Santiago Ramón y Cajal (1852-1934). Utilizando una técnica de impregnación argéntica desarrollada por el italiano Camillo Golgi (1843-1926), Cajal formuló la doctrina neuronal "el sistema nervioso está formado por células independientes, las neuronas, que contactan entre sí en lugares específico" y construyó un gran cuerpo de doctrina neuroanatómica.

Cajal fue un científico moderno, que no se limitó a describir estructuras estáticas, sino que se preguntó por los mecanismos que las gobiernan. Sus aportaciones a los problemas del desarrollo, la degeneración y la regeneración del sistema nervioso siguen siendo actuales.

A partir de la década de los sesenta del siglo pasado se dieron pasos agigantados en el estudio del cerebro, debido en gran medida a los avances tecnológicos. Por ejemplo, se desarrollaron escáneres que permitieron saber cómo es y cómo funciona este órgano. En años posteriores las investigaciones sobre él fueron enfocadas a la cognición humana (aprendizaje, memoria, percepción, etc.).

Como parte de este recorrido es posible establecer tres etapas: en la primera, que comprende hasta mediados de los ochenta, domina la metáfora del cerebro como un ordenador computacional; la segunda es la del conexionismo (modelos de redes neurales), en los años ochenta; y la tercera se ubica en los noventa, época conocida como la década del cerebro.

La década del cerebro se caracterizó por la mezcla de diversas ramas del conocimiento, cada una con un interés en particular respecto a alteraciones neurológicas como Parkinson, Alzheimer, neurofibromatosis, entre otras. Así, fue posible implicar al sector político y social en la investigación neurocientífica, desarrollar sistemas de

inversión federales y concienciar a la opinión pública sobre la importancia de las enfermedades neurológicas.

Ref. Emanuel Dzul

2

CONCEPTOS BÁSCIOS

2.1 INTRODUCCION

En éste capítulo se abordarán los conceptos clave de mayor importancia para facilitar el entendimiento del proyecto como pueden ser; el concepto de neurociencia, encefalograma o electroencefalograma, tipos y estructuras de las neuronas y por último se explicará el modelo de red de neuronas LIF utilizado durante el desarrollo del estudio.

2.2 DEFINICIÓN DE NEUROCIENCIA

La neurociencia es la ciencia que se ocupa del sistema nervioso y de cada uno de sus diversos aspectos y funciones especializadas. Aunque es una definición certera, para los expertos del Future Trends Forum (reunidos en Madrid en noviembre de 2019) se queda corta, concluyen que para ir más al detalle y teniendo en cuenta la complejidad de los procesos que suceden en el cerebro, se podría decir que la neurociencia surge con el objetivo de comprender el funcionamiento y la estructura del sistema nervioso desde distintas aproximaciones, mediante metodologías y técnicas diversas.

Ref. Cavada

2.3 DEFINICIÓN DE ELECTROENCEFALOGRAMA

Un encefalograma o electroencefalograma es una prueba que detecta la actividad eléctrica del cerebro mediante pequeños discos metálicos (electrodos) fijados sobre el cuero cabelludo. Las neuronas cerebrales se comunican a través de impulsos eléctricos y están activas todo el tiempo, incluso mientras duermes. Esta actividad se manifiesta como líneas onduladas en un registro de electroencefalograma.

Un electroencefalograma es uno de los estudios principales para diagnosticar la epilepsia. Un electroencefalograma también puede cumplir una función en el diagnóstico de otros trastornos cerebrales.

Ref. EEG

2.4 NEURONAS Y TIPOS DE NEURONAS

El cerebro humano tiene aproximadamente entre ochenta y cien mil millones de neuronas. Las redes neuronales son las encargadas de realizar las funciones complejas del sistema nervioso, es decir, que estas funciones no son consecuencia de las características específicas de cada neurona individual. Y como en el sistema nervioso realiza tantas tareas diversas y el funcionamiento de las diferentes partes del cerebro es tan complejo, estas células nerviosas también tienen que alcanzar un alto grado de complejidad, que consiguen especializándose y dividiéndose en diferentes tipos de neuronas.

2.4.1 *Estructura*

Por lo general la estructura de las neuronas está compuesta de las siguientes partes:

- **Soma:** El soma, también llamado pericarion, es el cuerpo celular de la neurona. Es donde se se encuentra el núcleo, y desde el cual nacen dos tipos de prolongaciones
- **Dendritas:** Las dendritas son prolongaciones que proceden del soma y parecen ramas o puntas. Reciben información procedente de otras células.
- **Axón:** El axón es una estructura alargada que parte del soma. Su función es la de conducir un impulso nervioso desde el soma hacia otra neurona, músculo o glándula del cuerpo. Los axones suelen estar cubiertos de mielina, una sustancia que permite una circulación más rápida del impulso nervioso.

2.4.2 *Tipos de neuronas*

Aunque existen distintas formas de clasificación de las neuronas en base a distintos criterios. Para el desarrollo de este trabajo se ha estudiado su clasificiacón en función del tipo de sinapsis.

Alrededor del ochenta por ciento de las neuronas son excitatorias. La mayoría de las neuronas tienen miles de sinapsis sobre su membrana, y cientos de ellas están activas simultáneamente. El que una sinapsis sea excitatoria o inhibitoria depende del tipo o tipos de iones que se canalizan en los flujos postsinápticos, que a su vez dependen del

tipo de receptor y neurotransmisor que interviene en la sinapsis (por ejemplo, el glutamato o el GABA, principal neurotransmisor inhibidor en el sistema nervioso central de mamíferos). Una descripción simplificada de neuronas excitatorias e inhibitorias es la siguiente:

- **Neuronas excitatorias:** Son aquellas que cuando producen un potencial de acción, activan otras neuronas.
- **Neuronas inhibitorias:** Son aquellas que realizan la función contraria, disminuyen la actividad de otras neuronas.

Ref. García-allen

3

MODELO DE RED DE NEURONAS LIF (SPIKES)

Una forma de comprender mejor el EEG (electroencefalograma) en términos de mecanismos de circuitos neuronales y vincular los modelos teóricos de las funciones cerebrales con los registros empíricos de EEG es comparar los datos de EEG con las predicciones cuantitativas obtenidas a través de simulaciones de modelos computacionales de redes de neuronas. Los modelos de red de neuronas de "tipo punto"de integración y Leaky-Integrate and Fire (LIF) son una herramienta importante actual en el modelado de la función cerebral. Estos modelos reducen la morfología de las neuronas a un solo punto en el espacio y describen la dinámica de las neuronas mediante un conjunto manejable de ecuaciones diferenciales acopladas. Estos modelos son lo suficientemente simples para ser entendidos a fondo, ya sea con simulaciones que son relativamente ligeras de implementar, o mediante enfoques analíticos.

A pesar de su simplicidad, generan una amplia gama de estados de red y dinámicas que se asemejan a las observadas en los registros corticales. Se han empleado para explicar satisfactoriamente un amplio espectro de diferentes mecanismos corticales y funciones corticales, como la codificación de información sensorial, la memoria de trabajo, la atención, la propagación de ondas, estados de vigilia no rítmicos, o la aparición de estados de altibajos. Sigue siendo una pregunta abierta cómo calcular EEG de manera realista a partir de modelos de red tan ampliamente utilizados de neuronas puntuales simples.

La actividad neuronal a menudo se registra a nivel de señales eléctricas agregadas. Estas señales se registran de forma invasiva en animales (por ejemplo, potenciales de campo local, LFP y electrocorticogramas, ECoG) o de forma no invasiva en humanos (por ejemplo, electroencefalogramas, EEG y magnetoencefalogramas, MEG).

Estas diferentes señales cerebrales agregadas comparten en gran medida las mismas fuentes neuronales y tienen importantes aplicaciones tanto en la investigación científica como en el diagnóstico clínico. Son fáciles de registrar, capturan muchos fenómenos agregados a nivel de circuito, incluidas señales integradoras sinápticas clave en diferentes niveles de organización, desde escalas cerebrales mesoscópicas a macroscópicas, y pueden revelar actividad oscilatoria en una amplia gama de frecuencias.

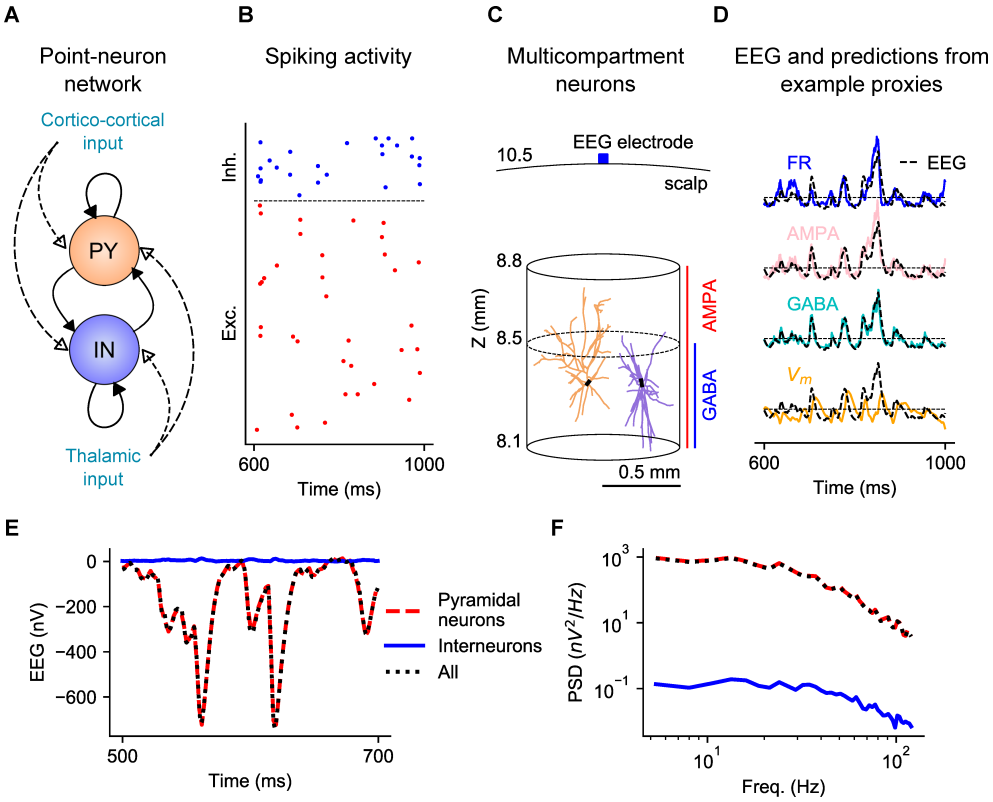


Figura 1: Descripción general de los modelos de red y cálculo de proxies y EEG de acuerdo con el trabajo realizado en Martínez-Cañada et al. [2021]

De acuerdo con la proporción de neuronas excitatorias e inhibidoras encontradas en la corteza cerebral, en nuestro modelo 4000 neuronas son excitatorias (es decir, sus proyecciones sobre otras neuronas forman sinapsis excitadoras similares a AMPA) y 1000 inhibitorias (es decir, sus proyecciones forman sinapsis similares a GABA), aleatoriamente conectados con una probabilidad de conexión entre cada par de neuronas de $0,2$. Todas las neuronas en el modelo reciben dos entradas externas (tanto una entrada talámica sensorial como una entrada intracortical, que representa ruido cortical) que fueron diseñadas para predecir algunos aspectos clave de la actividad neuronal en la corteza visual primaria durante la estimulación visual naturalista y la actividad espontánea.

Ref. Martínez-Cañada and Panzeri [2021]. Ref. Mazzoni [2010]

4

PARALELIZACIÓN DE LAS SIMULACIONES EN UN SERVIDOR

La paralelización puede mejorar la eficiencia de la ejecución de simulaciones a gran escala aprovechando máquinas multinúcleo/multiprocesador, clústeres de computadoras o supercomputadoras. En este caso, para las simulaciones se ha utilizado un servidor con sistema operativo Linux que cuenta con 32 núcleos disponibles y se ha hecho uso de la totalidad de ellos. A su vez, ha estado disponible una memoria RAM de 128Gb. Y el sistema tiene capacidad para almacenar hasta 2TB de información.

```
(TFG) estudiante1@bcilab:~$ lscpu
Arquitectura:                         x86_64
modo(s) de operación de las CPUs:     32-bit, 64-bit
Orden de los bytes:                   Little Endian
Address sizes:                        46 bits physical, 48 bits virtual
CPU(s):                               32
Lista de la(s) CPU(s) en línea:       0-31
Hilo(s) de procesamiento por núcleo: 2
Núcleo(s) por «socket»:               16
«Socket(s)»                           1
Modo(s) NUMA:                         1
ID de fabricante:                     GenuineIntel
Familia de CPU:                       6
Modelo:                               85
Nombre del modelo:                    Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz
Revisión:                             7
CPU MHz:                              842.328
CPU MHz máx.:                         3200,0000
CPU MHz mín.:                         800,0000
BogoMIPS:                             4200.00
Virtualización:                       VT-x
Caché L1d:                            512 KiB
Caché L1i:                            512 KiB
Caché L2:                             16 MiB
Caché L3:                             22 MiB
CPU(s) del nodo NUMA 0:               0-31
```

Figura 2: Información del servidor.

Gracias a servidor ha sido posible realizar una gran cantidad de simulaciones para disponer del mayor número de datos posibles modificando ciertos parámetros del modelo neuronal que se explicarán a continuación.

Para ejecutar las simulaciones se ha instalado NEST, que es un simulador de red neuronal de spikes utilizado en neurociencia computacional para simular dinámicas de interacción entre neuronas. NEST utiliza de forma nativa OpenMP (Documentation)

para ejecutarse en las computadoras multinúcleo y multiprocesador sin necesidad de bibliotecas adicionales.

En el modelo neuronal utilizado, estamos usando un total de 5000 neuronas de las cuales 4000 son excitatorias y 1000 son inhibitorias. En cada simulación se estudia el comportamiento de estas neuronas y su actividad neuronal en intervalos de 1 ms. Por lo tanto, obtenemos 1000 puntos de actividad neuronal simulada. Sin embargo se ha observado que en los 100 primeros ms aparecían resultados de actividad transitoria que podría afectar la interpretación de los resultados, por lo tanto, para cada una de las simulaciones se ha seleccionado el intervalo de 100 a 999 ms.

Para la obtención de de todo el conjunto de datos se han realizado un total de 480 simulaciones en los cuales se han ido variando cada una de las 5 variables de la red neuronal que son objeto de estudio de este proyecto.

Cuatro variables se corresponden con las conductancias sinápticas de pico, que determinan la intensidad de las conexiones entre los distintos tipos de neuronas.

- Conductancia sináptica de excitatorias a excitatorias (variable exc_exc_recurrent).
- Conductancia sináptica de excitatorias a inhibitorias (variable exc_inh_recurrent).
- Conductancia sináptica de inhibitorias a inhibitorias (variable inh_inh_recurrent).
- Conductancia sináptica de inhibitorias a excitatorias (variable inh_exc_recurrent).

```
#! ===========
#! Parameters
#! ===========

size_factor = 1.0 # downscale the size of the network
weight_factor = 1.0 # increase the synaptic weights to compensate it

Network_params = {
    "N_exc": int(4000.0/size_factor),
    "N_inh": int(1000.0/size_factor),
    "P": 0.2,
    "extent": 1.0,
    "exc_exc_recurrent": 0.178*weight_factor,
    "exc_inh_recurrent": 0.233*weight_factor,
    "inh_inh_recurrent": -2.70*weight_factor,
    "inh_exc_recurrent": -2.01*weight_factor,
    "th_exc_external": 0.234*weight_factor,
    "th_inh_external": 0.317*weight_factor,
    "cc_exc_external": 0.187*weight_factor,
    "cc_inh_external": 0.254*weight_factor
}
```

Figura 3: Código donde se representan los valores de referencia de las conductancias sinápticas del modelo.

La última variable v_0 que se corresponde con la tasa de entrada externa del modelo, la cual han variado entre dos valores; 1.5 spikes/seg y 2.5 spikes/s. Por lo tanto se han obtenido obtenido 240 simulaciones con el parámetro $v_0 = 1{,}5$ y otras 240 simulaciones con el parámetro $v_0 = 2{,}5$

```
External_input_params = {
    "v_0": 1.5,
    "A_ext": 0.0 ,
    "f_ext": 0.0 ,
    "OU_sigma": 0.4 ,
    "OU_tau": 16.0
}
```

Figura 4: Código donde se representan los parámetros que describen la entrada externa del modelo, entre ellos v_0 .

4.1 ANÁLISIS DE LOS DATOS

En primer lugar hemos estudiado las distintas salidas de actividad neuronal que genera el modelo y como pueden variar según la modificación de los parámetros. Para cada uno de los valores de los parámetros se han repetido las simulaciones 3 veces con el objetivo de promediar la aleatoriedad del modelo (e.g., las conexiones aleatorias entre neuronas).

Para cada una de las simulaciones se obtienen 9 archivos distintos correspondientes a distintas salidas del modelo (e.g., spikes, corrientes sinápticas AMPA y GABA) pero únicamente se han utilizado en este caso, los archivos .AMPA y .GABA que serán útiles para el cálculo del EEG, usando la siguiente fórmula que en trabajos previos ha demostrado aproximar bien el EEG, Martínez-Cañada et al. [2021].

$$EEG = |AMPA| + |GABA| \tag{1}$$

Los archivos .spikes que usaremos para visualización pero no para análisis posterior de resultados.

Se mostrará a continuación las gráficas obtenidas en puntos de relevancia gracias a los cuales va a ser posible observar diferencias notables en cada una de ellas.

4.1.1 *Resultados de una simulación con los parámetros estándar*

En esta simulación los valores de los parámetros de la red neuronal son los valores de referencia del modelo que en trabajos anteriores se ha demostrado que capturan gran parte de la dinámica cortical, Mazzoni et al. [2008].

- exc_exc_recurrent: 0.178
- exc_inh_recurrent: 0.233
- inh_inh_recurrent: -2.70
- inh_exc_recurrent: -2.01

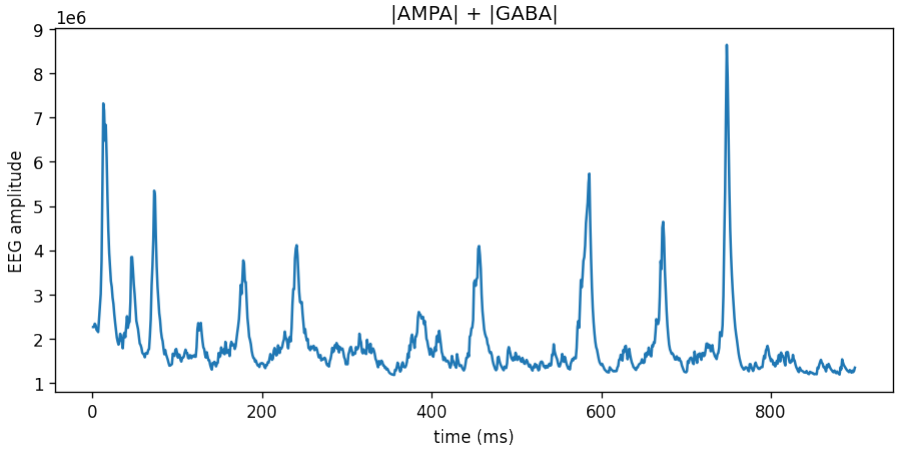


Figura 5: Gráfica EEG con parámetros estandar.

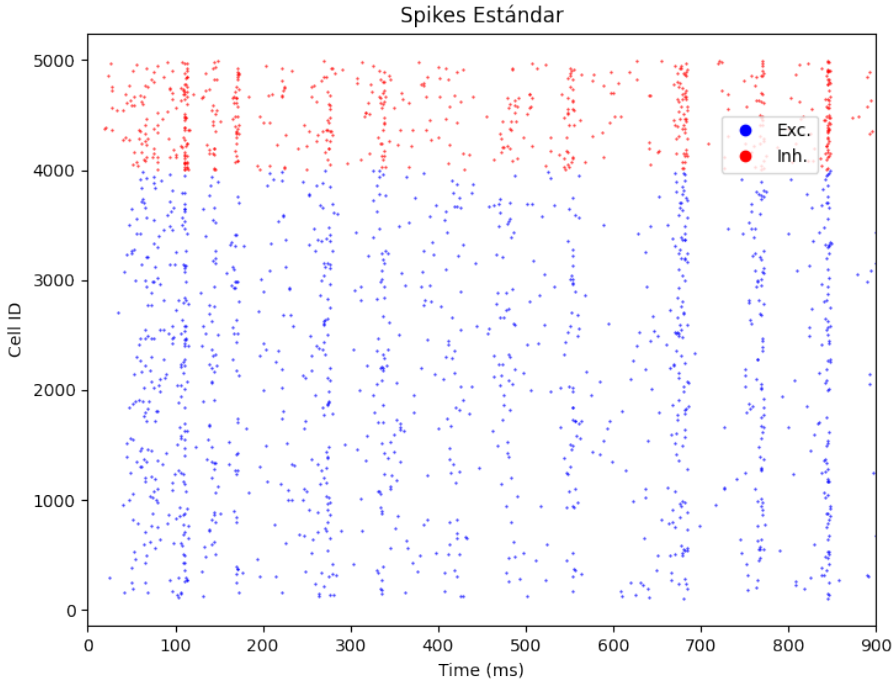


Figura 6: Gráfica de spikes con parámetros estandar.

4.1.2 *Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.005)*

Para esta simulación se ha realizado la ejecución con los valores estándar excepto para la variable exc_exc_recurrent que su valor se ha reducido al mínimo (0.005). Se observa una reducción generalizada de la actividad neuronal.

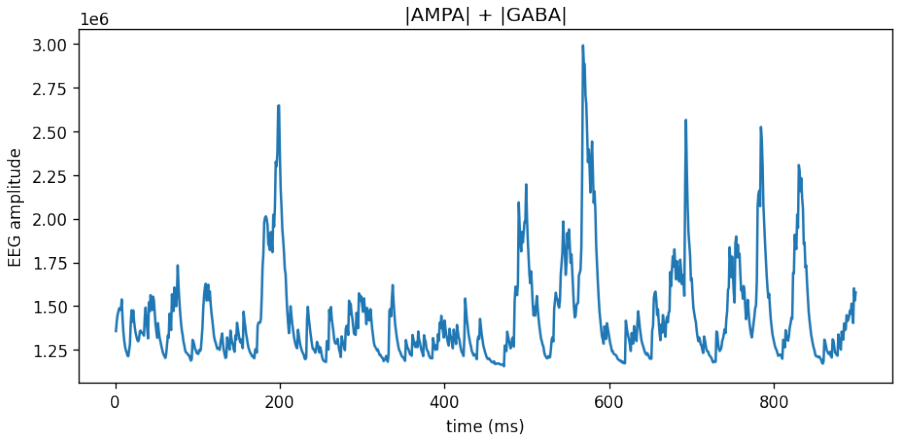


Figura 7: Gráfica EEG con parámetro exc_exc = 0.005.

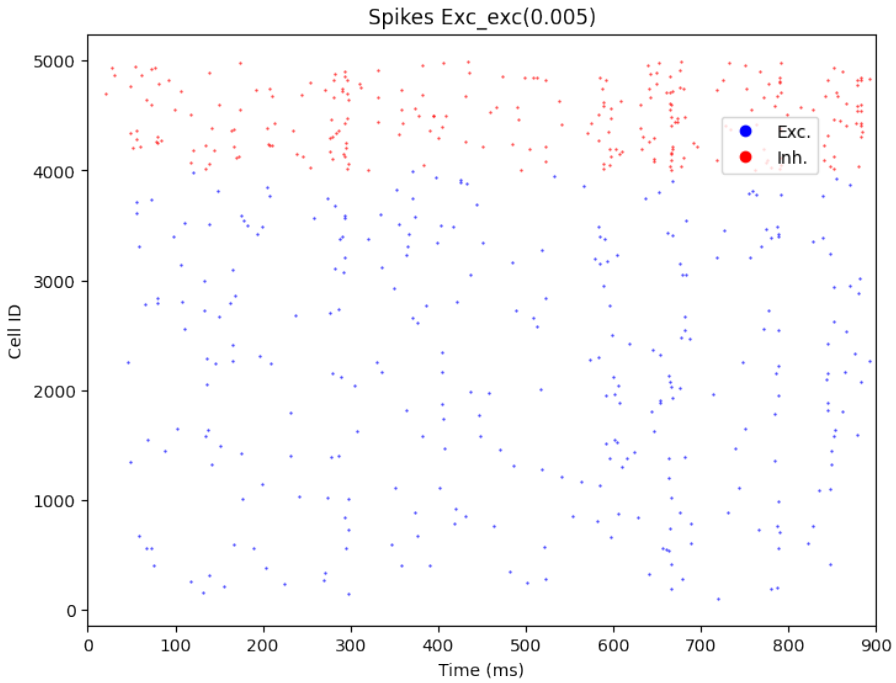


Figura 8: Gráfica de spikes con parámetro exc_exc = 0.005.

4.1.3 *Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.200)*

Para esta simulación se ha realizadao la ejecución con los valores estándar excepto para la variable exc_exc_recurrent que su valor se ha aumentado levemente con respecto al estándar (0.200). Se observa una leve sincronización que produce oscilaciones periódicas.

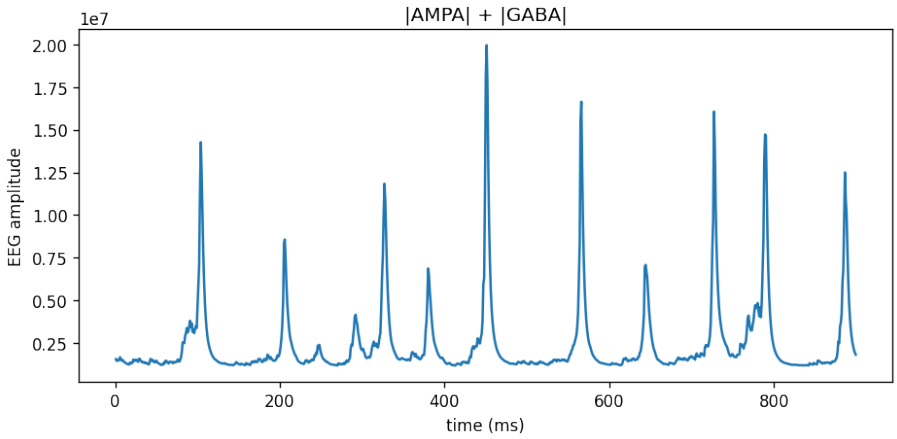


Figura 9: Gráfica EEG con parámetro exc_exc = 0.200.

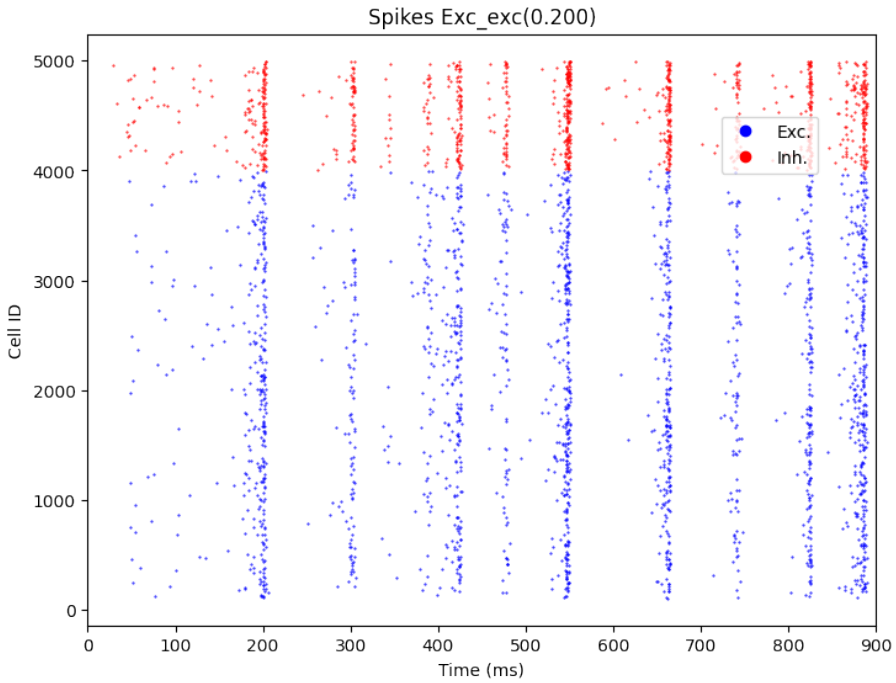


Figura 10: Gráfica de spikes con parámetro exc_exc = 0.200.

4.1.4 *Resultados de una simulación con parámetro exc_exc_recurrent alterado (0.450)*

Para esta simulación se ha realizadao la ejecución con los valores estándar excepto para la variable exc_exc_recurrent que su valor se ha aumentado considerablemente con respecto al estándar (0.450). Este correspondería con un escenario irrealista de sincronización muy elevada.

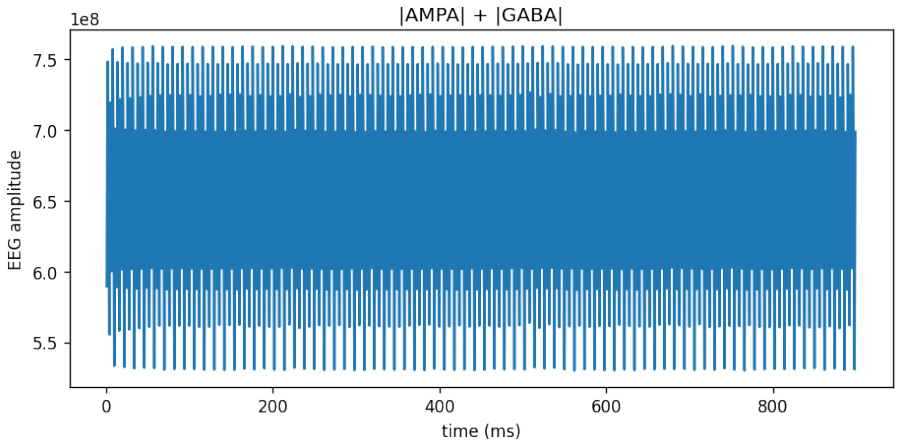


Figura 11: Gráfica EEG con parámetro exc_exc = 0.450.

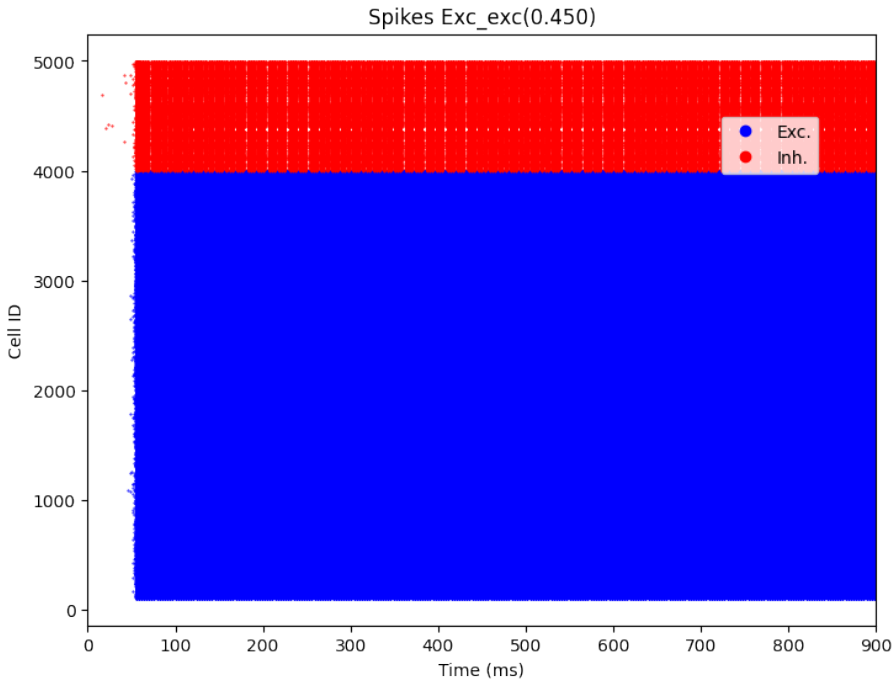


Figura 12: Gráfica de spikes con parámetro exc_exc = 0.450.

4.1.5 *Resultados de una simulación con parámetro inh_exc_recurrent alterado (-0.25)*

Para esta simulación se ha realizado la ejecución con los valores estándar excepto para la variable inh_exc_recurrent que su valor se ha aumentado considerablemente con respecto al estándar (-0.25). De nuevo observamos una respuesta demasiado sincronizada.

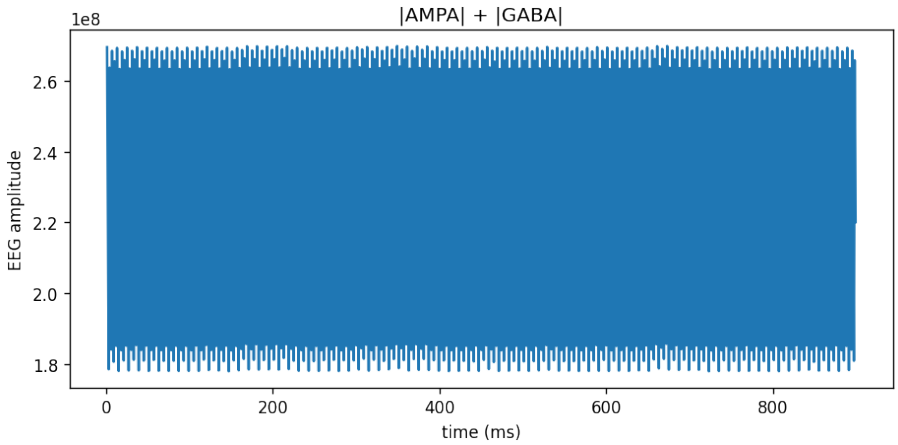


Figura 13: Gráfica EEG con parámetro inh_exc = -0.25.

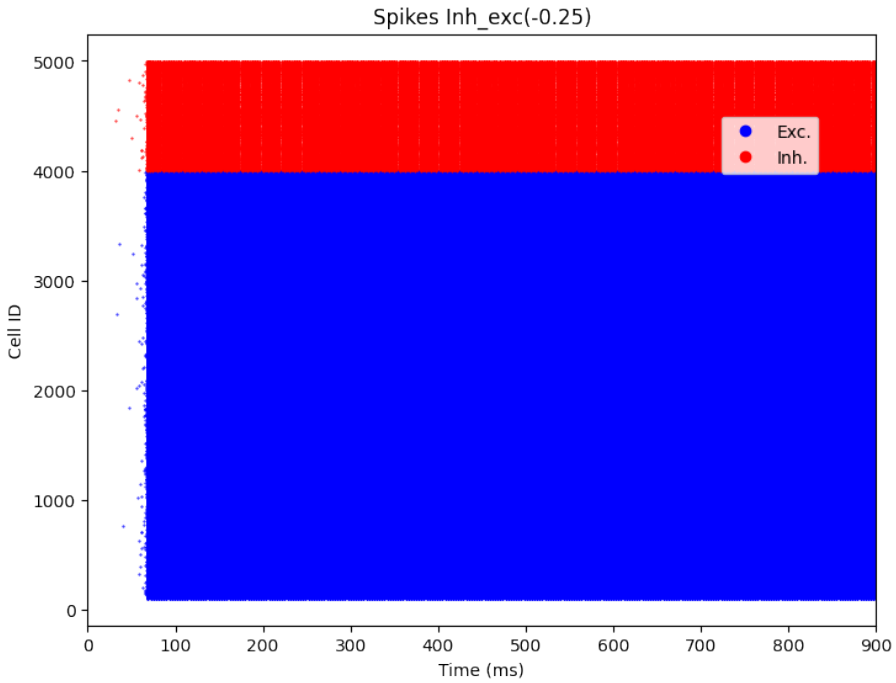


Figura 14: Gráfica de spikes con parámetro inh_exc = -0.25.

4.1.6 *Resultados de una simulación con parámetro inh_exc_recurrent alterado (-4.50)*

Para esta simulación se ha realizadao la ejecución con los valores estándar excepto para la variable inh_exc_recurrent que su valor se ha disminuido considerablemente con respecto al estándar (-4.50). La red produce una salida muy atenuada.

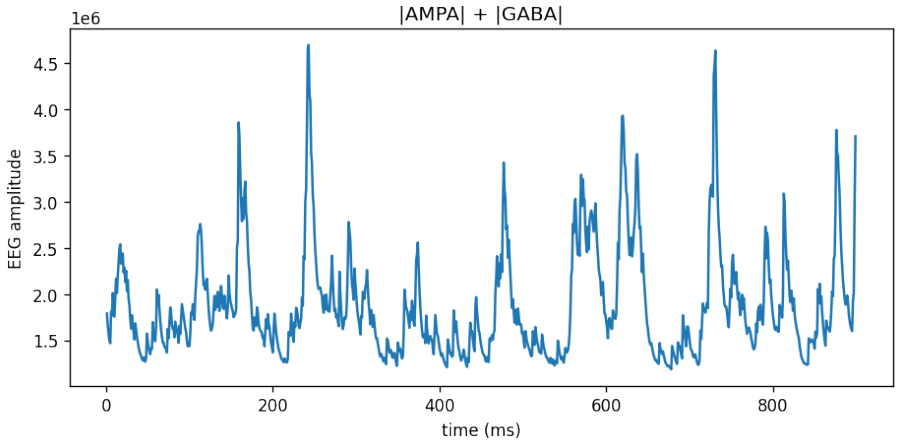


Figura 15: Gráfica EEG con parámetro inh_exc = -4.50.

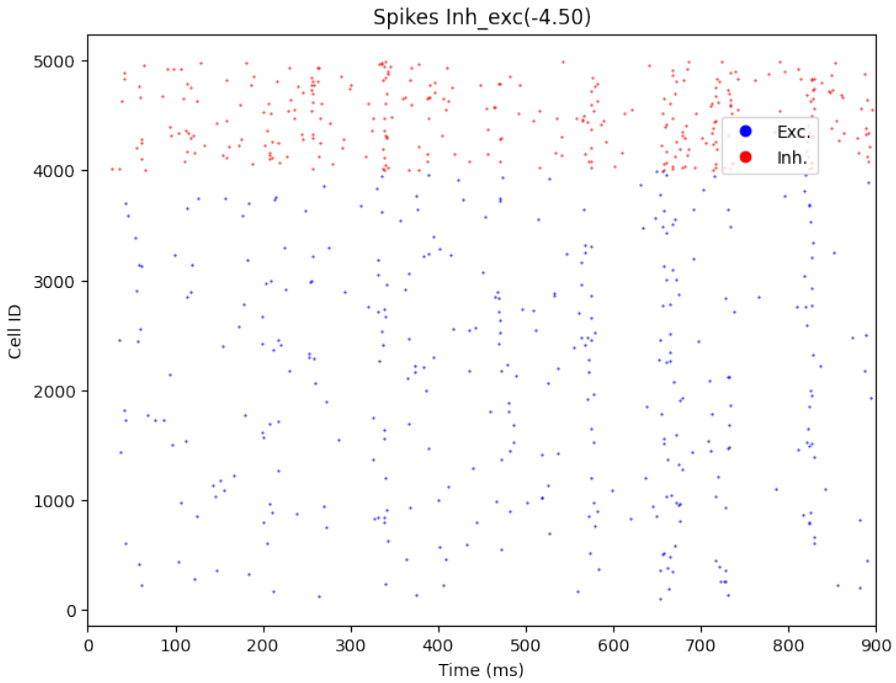


Figura 16: Gráfica de spikes con parámetro inh_exc = -4.50.

4.1.7 *Resultados de una simulación con parámetro inh_exc_recurrent alterado (-8.00)*

Para esta simulación se ha realizadao la ejecución con los valores estándar excepto para la variable inh_exc_recurrent que su valor se ha disminuido al mínimo con respecto al estándar (-8.00). A penas se diferencian puntos en los cuales se produzcan reacciones neuronales sincronizadas.

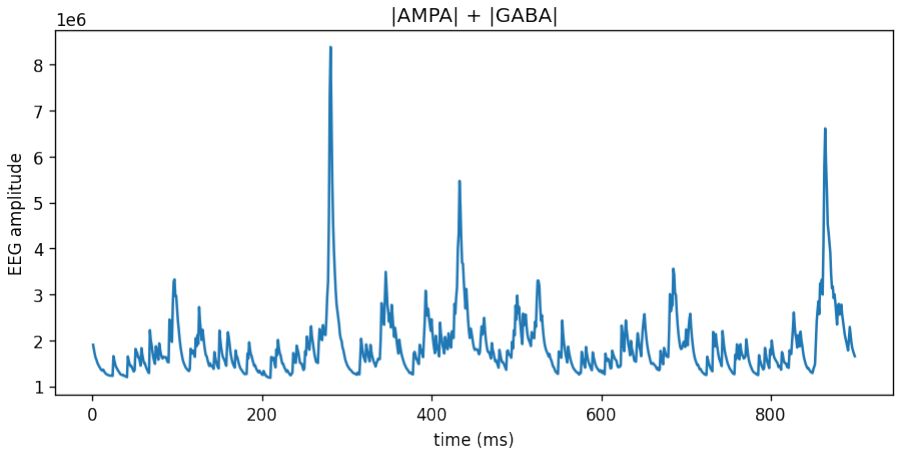


Figura 17: Gráfica EEG con parámetro inh_exc = -8.00.

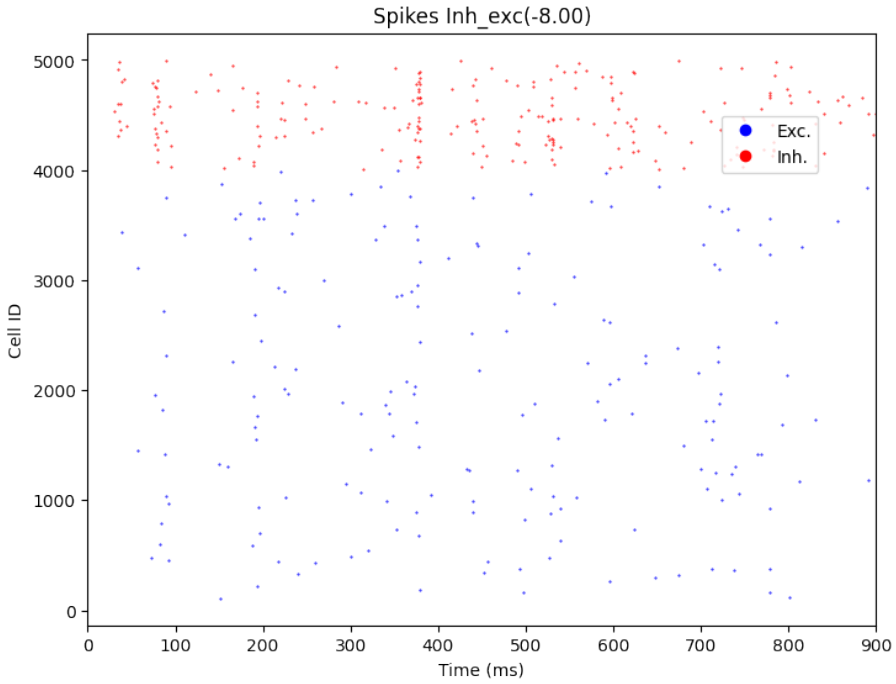


Figura 18: Gráfica de spikes con parámetro inh_exc = -8.00.

5

CONCLUSIONES DEL ANÁLISIS DE LOS DATOS

A partir del estudio de este tipo de gráficas, se pueden observar distintos tipos de comportamientos que en determinados casos se pueden asociar con resultados de cerebros de personas humanas que sufren algún tipo de enfermedad relacionada con el sistema nervioso. Un ejemplo de gráficas similares a las obtenida en las figura 6 y 16 podría ser la de una persona diagnosticada con trastorno del espectro autista (TEA) ya que a penas se diferencian puntos en los cuales se observan una gran actividad neuronal.

Esta información, además del estudio de anomalías genéticas, conduce a un modelo que postula que algunas formas de autismo son causadas por una mayor proporción de excitación/inhibición en los sistemas sensorial, mnemotécnico, social y emocional. El modelo postula además que el aumento de la proporción de excitación/inhibición puede ser causado por efectos combinatorios de variables genéticas y ambientales que inciden sobre un sistema neuronal dado, se ha estudiado en profundidad en Rubenstein [2003].

6

ESTIMACIÓN DE PARÁMETROS DEL CIRCUITO NEURONAL A PARTIR DE LA SEÑAL DEL ENCEFALOGRAMA

Uno de los objetivos de la realización de este proyecto es la estimación de parámetros del circuito neuronal a partir de la señal del encefalograma. Para ello, se ha programado en Python varios algoritmos de Machine Learning gracias a la librería sklearn obteniendo distintos resultados.

Los datos sobre los que se han trabajado han sido las 3 repeticiones de las simulaciones en las que se ha variado el valor de la variable exc_exc e inh_exc combinadas con los dos valores del parámetro v_0. Se han almacenado los valores del EEG en una variable X en la que en cada elemento tenemos un vector con todos los valores del EGG de esa simulación y otra variable Y multidimensional que se corresponden con los valores de los ratios de conductancias g calculados como:

$$g = \frac{g\_exc}{g\_inh} \tag{2}$$

y con con los valores de la tasa de entrada del tálamo v_0.

```
#30 exc_exc y 30 inh_exc, ext parameter 1.5
X = []
for trial in range(3):
    for k in np.arange(0,0.25,0.025):
        if(k==0):
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent(0.005)-'+str(trial+1)+'-0_rate_1.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent(0.005)-'+str(trial+1)+'-0_rate_1.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))
        else:
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent('+"{:.3f}".format(k)+')-'+str(trial+1)+'-0_rate_1.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent('+"{:.3f}".format(k)+')-'+str(trial+1)+'-0_rate_1.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))

for trial in range(3):
    for k in np.arange(1.50,4,0.25):
        if(k==0):
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-0.05)-'+str(trial+1)+'-0_rate_1.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-0.05)-'+str(trial+1)+'-0_rate_1.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))
        else:
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-'+"{:.2f}".format(k)+')-'+str(trial+1)+'-0_rate_1.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-'+"{:.2f}".format(k)+')-'+str(trial+1)+'-0_rate_1.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))

#30 exc_exc y 30 inh_exc, ext parameter 2.5
for trial in range(3):
    for k in np.arange(0,0.25,0.025):
        if(k==0):
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent(0.005)-'+str(trial)+'_rate_2.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent(0.005)-'+str(trial)+'_rate_2.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))
        else:
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent('+str(round(k,3))+')-'+str(trial)+'_rate_2.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_exc_exc_recurrent('+str(round(k,3))+')-'+str(trial)+'_rate_2.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))

for trial in range(3):
    for k in np.arange(1.50,4,0.25):
        if(k==0):
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-0.05)-'+str(trial)+'_rate_2.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-0.05)-'+str(trial)+'_rate_2.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))
        else:
            dataAMPA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-'+str(round(k,3))+')-'+str(trial)+'_rate_2.5.AMPA',"rb"),encoding='latin1' )
            dataGABA = pickle.load(open( '../Results/AMPA, GABA y spikes/trial_inh_exc_recurrent(-'+str(round(k,3))+')-'+str(trial)+'_rate_2.5.GABA',"rb"),encoding='latin1' )
            X.append(abs(dataAMPA[100:999]) + abs(dataGABA[100:999]))
```

Figura 19: Fuente de datos variable X

Antes de entrenar cada uno de los modelos se han normalizado haciendo uso de las herramientas de preprocesamiento proporcionadas por la librería sklearn. Además de dividir los datos entre datos de entrenamiento y datos de test para poder evaluar posteriormente mediante cross-validation cada uno de los modelos. Para ello elegimos el ochenta por ciento de datos como entrenamiento y el veinte por ciento restante para test.

A continuación, veremos los resultados obtenidos en forma de gráficas después de utilizar cada uno de los diferentes algoritmos.

6.1 ALGORITMO DE REGRESIÓN LINEAL SIMPLE

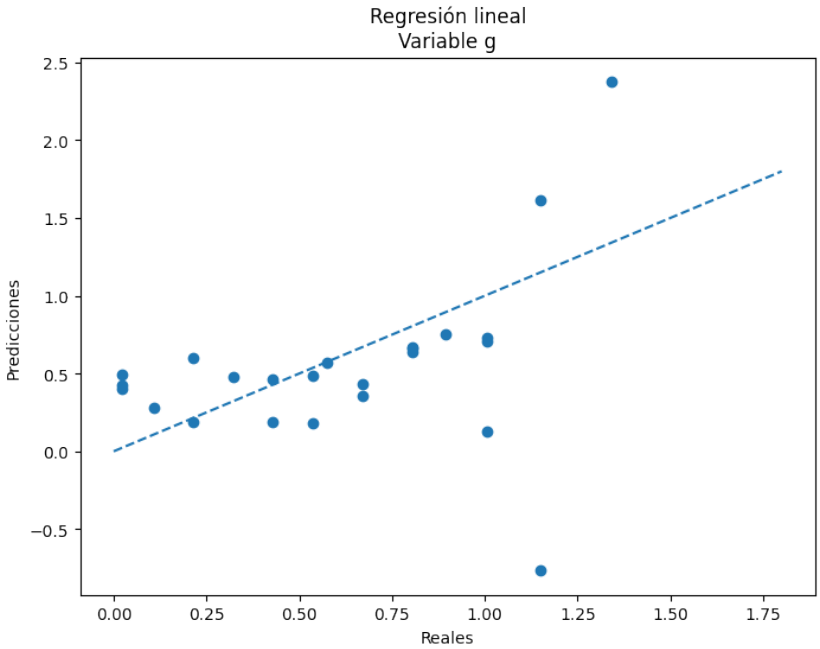
La regresión lineal simple es un método que usa la relación estadística entre dos variables cuantitativas en la que una variable, la variable de respuesta (o variable dependiente) puede ser predicha a partir de otra variable (la variable predictora o independiente).

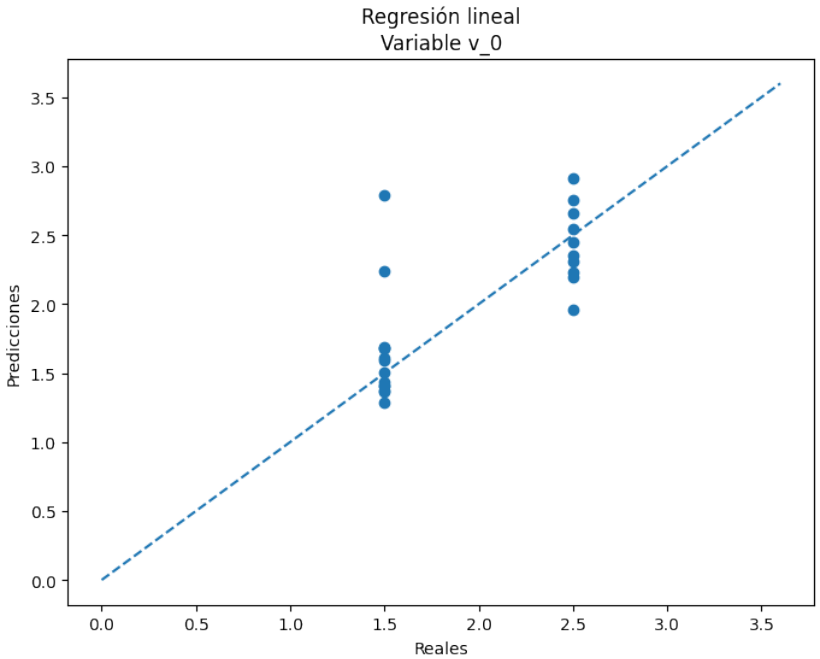
A continuación, se muestra el código utilizado para obtener el resultado de los errores cuadráticos medios después de 1000 ejecuciones para evitar aleatoriedad. Donde podemos diferenciar varias partes:
- Creación de los datos de entrenamiento y de tests.

- Escalado de entrada para ajustar los datos al modelo.
- Entrenamiento del modelo donde se llama a algoritmo de regresión lineal.
- Evaluación del modelo haciendo uso de los errores cuadráticos medios.

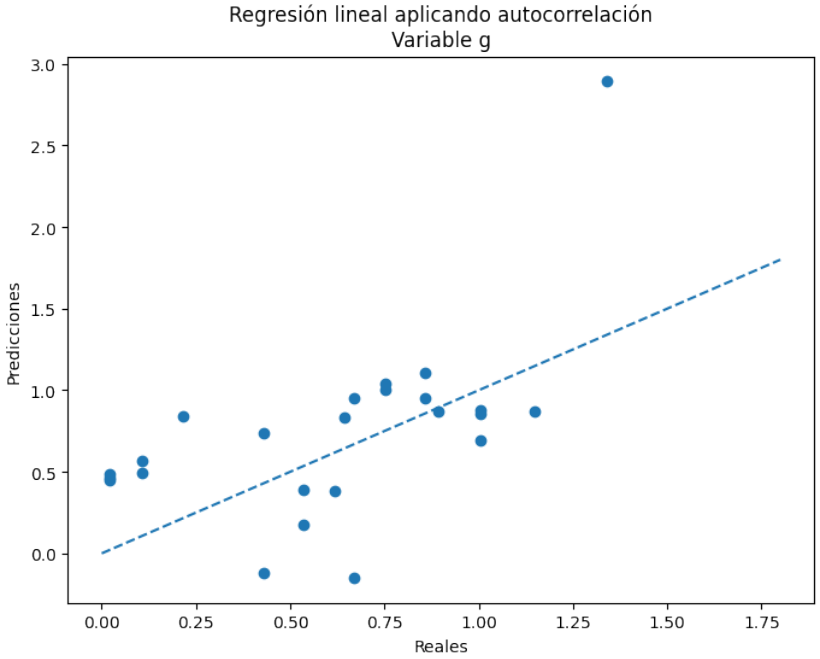
```
## Ejecutamos 1000 veces el algoritmo y obtenemos la media
avg_result_g = []
avg_result_v_0 = []
for times in range (100):
    for fold in range(10):
        #Creamos nuestro X e y de entrada y los sets de entrenamiento y test.
        X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)

        #Escalado de la entrada
        scaler = MinMaxScaler()
        scaler.fit(X_train)
        X_test = scaler.fit_transform(X_test)
        X_train = scaler.transform(X_train)

        #Entrenamiento del modelo
        reg = LinearRegression(fit_intercept=True, normalize=False).fit(X_train,y_train)

        #Predicciones
        y_pred = reg.predict(X_test)

        #Evaluacion del modelo
        y_test_np = np.array(y_test)
        y_pred_np = np.array(y_pred)
        avg_result_g.append(mean_squared_error(y_test_np[:,0], y_pred_np[:,0]))
        avg_result_v_0.append(mean_squared_error(y_test_np[:,1], y_pred_np[:,1]))

print("Media")
print(np.mean(avg_result_g))
print(np.mean(avg_result_v_0))
resultadoRegresionLineal = np.mean(avg_result_g)
resultadoRegresionLineal_v_0 = np.mean(avg_result_v_0)
```

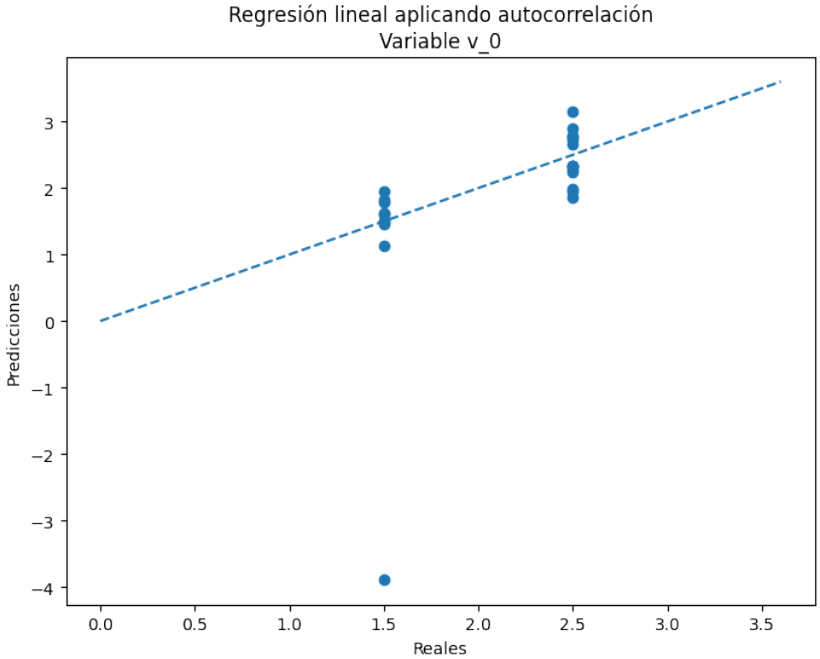
6.1.1 *Machine Learning con algoritmo de regresión lineal simple*





6.1.2 *Machine Learning con algoritmo de regresión lineal simple aplicando autocorrelación*





6.2 ALGORITMO DE REGRESIÓN LINEAL DE RIDGE

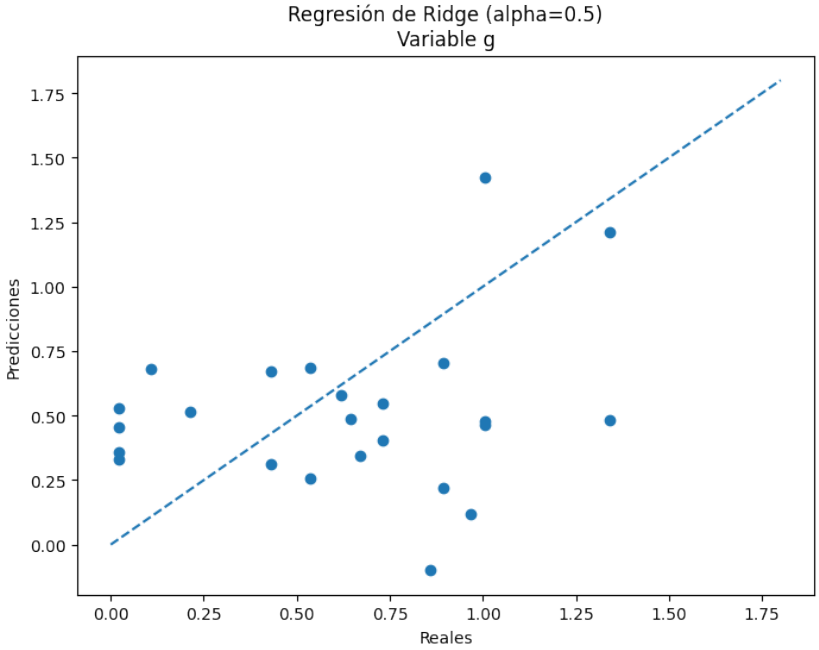
La regularización Ridge penaliza la suma de los coeficientes elevados al cuadrado. A esta penalización se le conoce como L2 y tiene el efecto de reducir de forma pro-

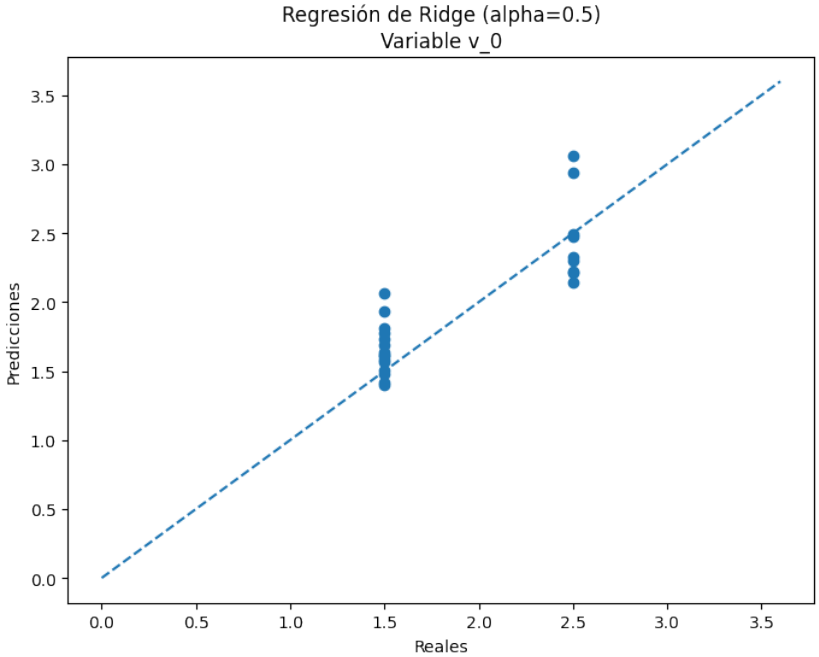
porcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a cero. El grado de penalización está controlado por lambda. Cuando lambda=0, la penalización es nula y el resultado es equivalente al de un modelo lineal por mínimos cuadrados ordinarios (OLS). A medida que lambda aumenta, mayor es la penalización y menor el valor de los predictores.

A continuación, se muestra el código utilizado para obtener el resultado de los errores cuadráticos medios después de 1000 ejecuciones para evitar aleatoriedad. Donde podemos diferenciar varias partes:
- Creación de los datos de entrenamiento y de tests.
- Escalado de entrada para ajustar los datos al modelo.
- Entrenamiento del modelo donde se llama a algoritmo de regresión de Ridge y se va variando alfa.
- Evaluación del modelo haciendo uso de los errores cuadráticos medios.

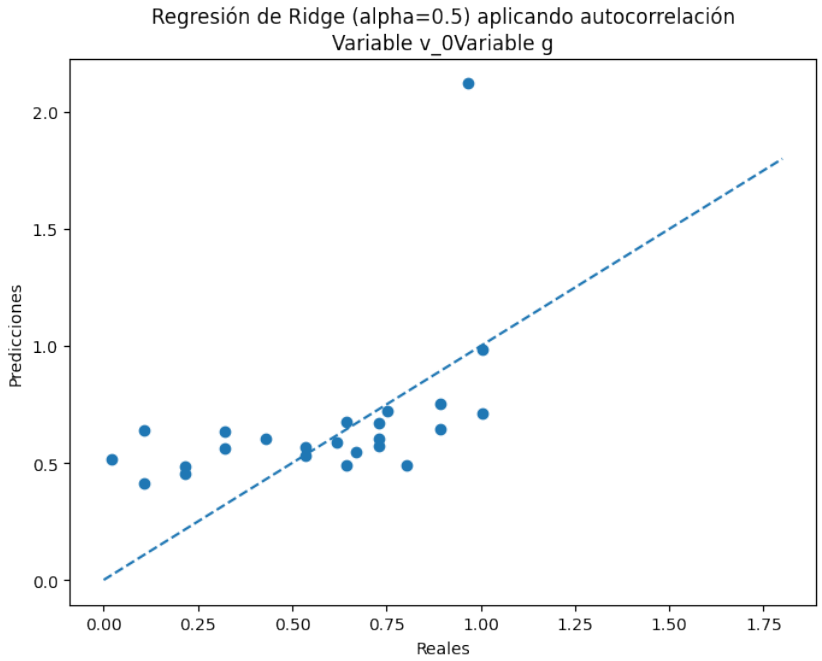
```
## Ejecutamos 1000 veces el algoritmo y obtenemos la media
avg_result_g = []
avg_result_v_0 = []
for times in range (100):
    for fold in range(10):
        #Creamos nuestro X e y de entrada y los sets de entrenamiento y test.
        X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)

        #Escalado de la entrada
        scaler = MinMaxScaler()
        scaler.fit(X_train)
        X_test = scaler.fit_transform(X_test)
        X_train = scaler.transform(X_train)

        #Entrenamiento del modelo
        reg = Ridge(alpha=0.5).fit(X_train,y_train)

        #Predicciones
        y_pred = reg.predict(X_test)

        #Evaluacion del modelo
        y_test_np = np.array(y_test)
        y_pred_np = np.array(y_pred)
        avg_result_g.append(mean_squared_error(y_test_np[:,0], y_pred_np[:,0]))
        avg_result_v_0.append(mean_squared_error(y_test_np[:,1], y_pred_np[:,1]))

print("Media")
print(np.mean(avg_result_g))
print(np.mean(avg_result_v_0))
resultadoRegresionLinealRidge0_5 = np.mean(avg_result_g)
resultadoRegresionLinealRidge0_5_v_0 = np.mean(avg_result_v_0)
```

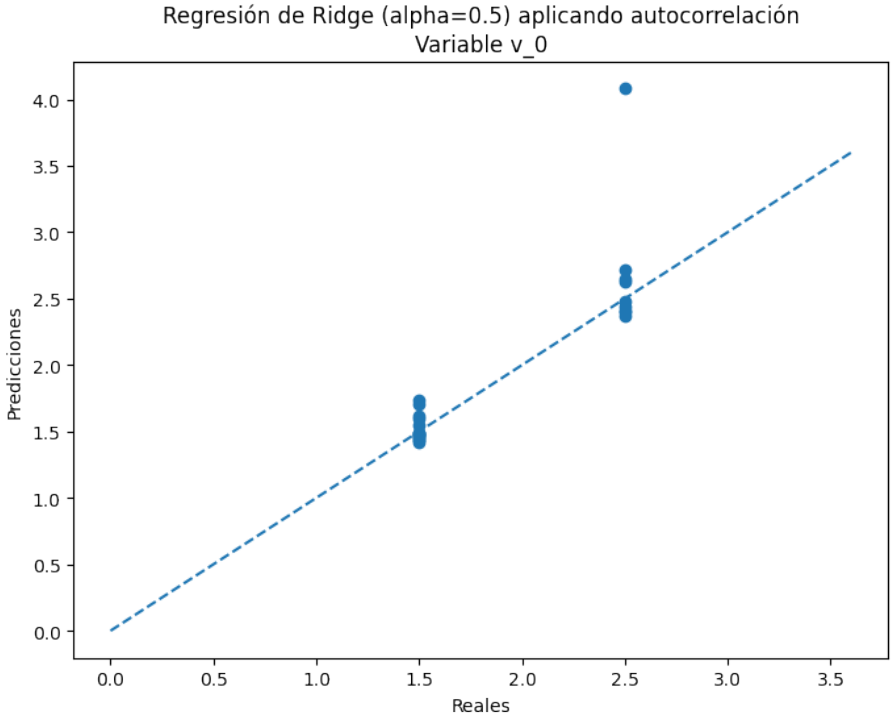
6.2.1 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 0.5*



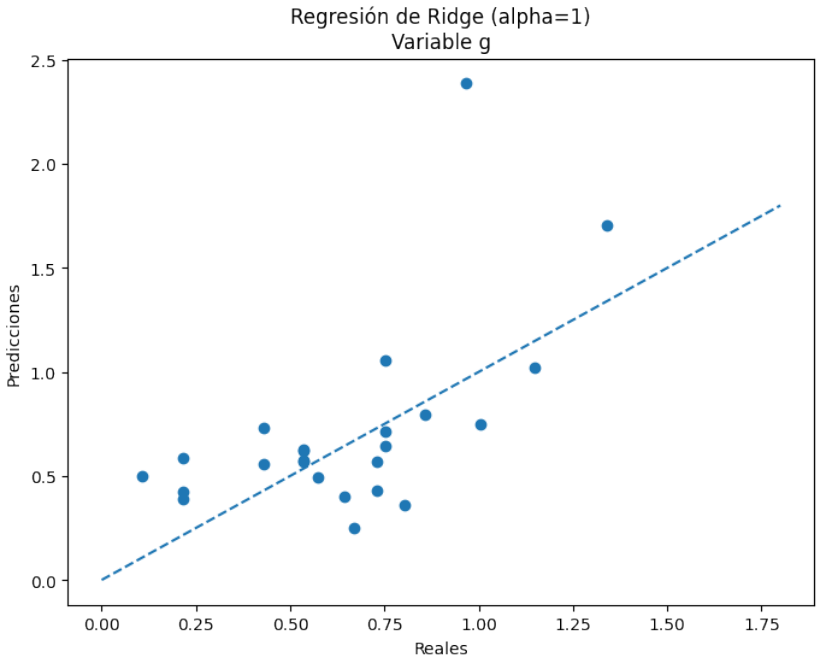


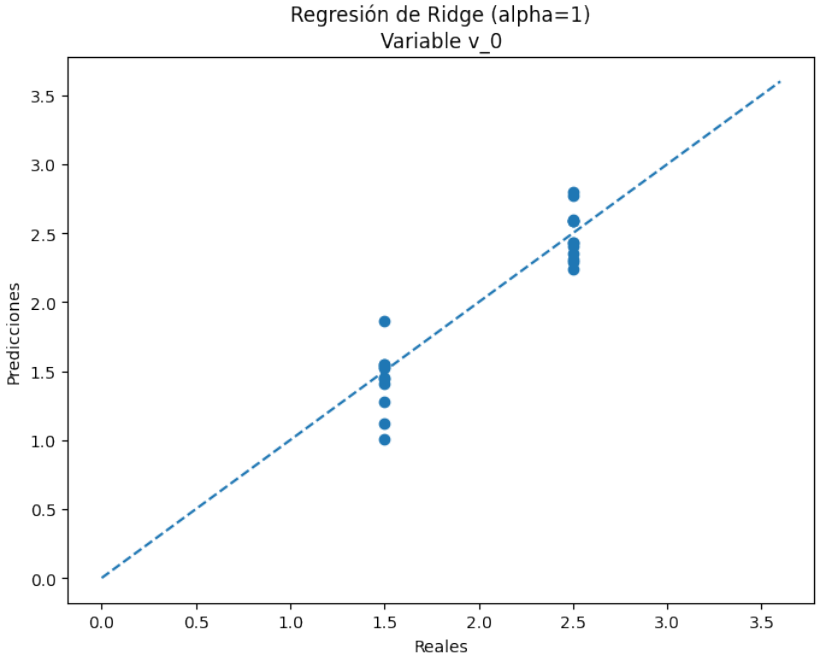
6.2.2 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 0.5 aplicando autocorrelación*



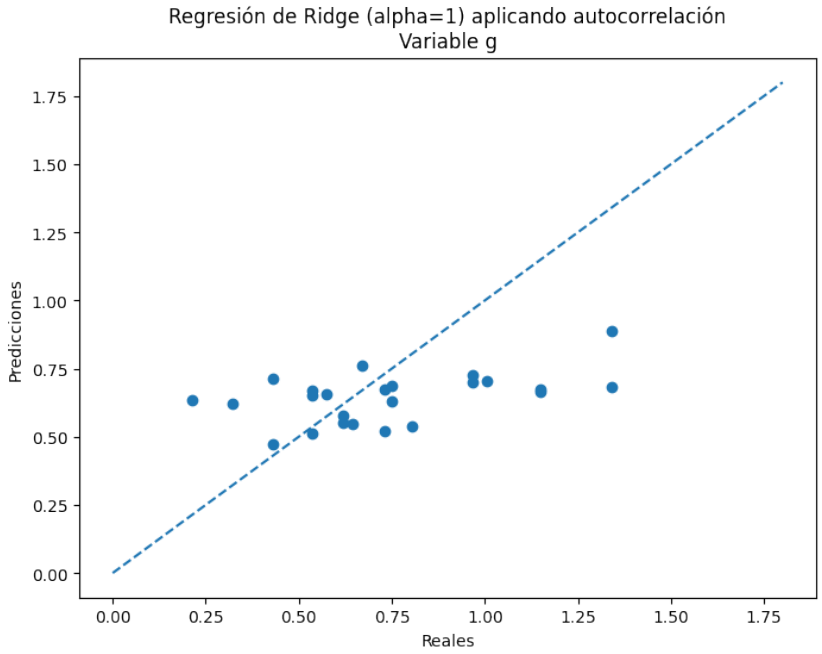


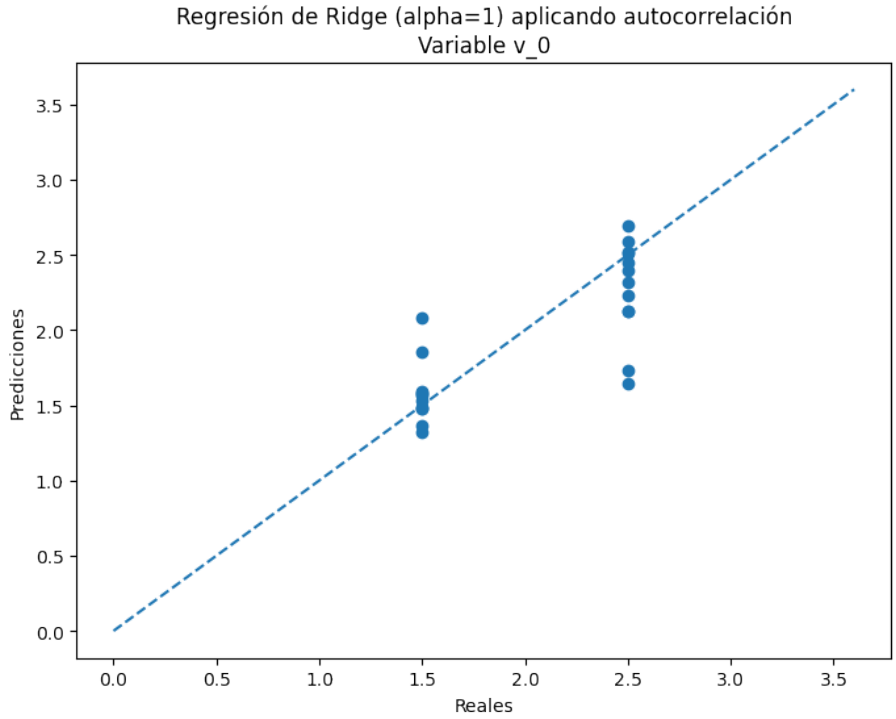
6.2.3 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1*



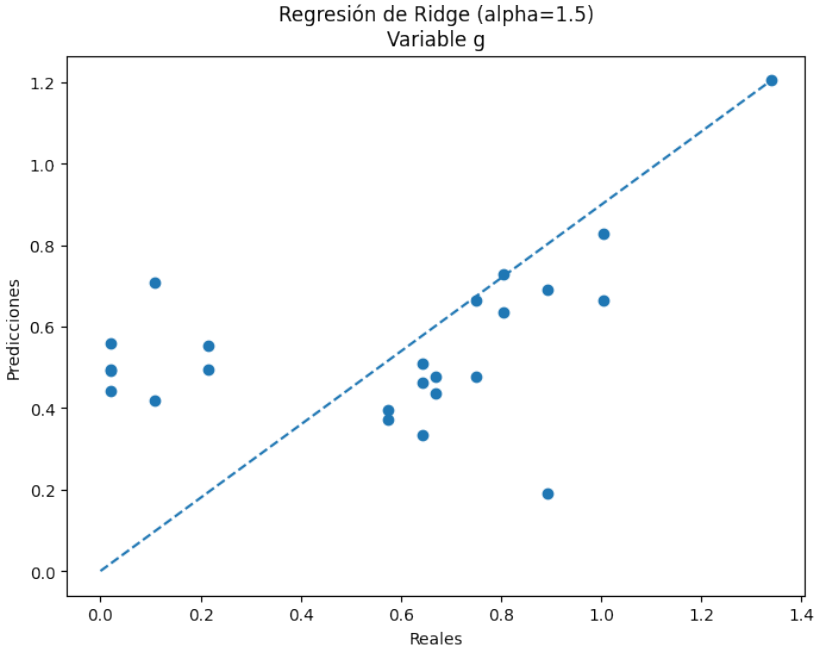


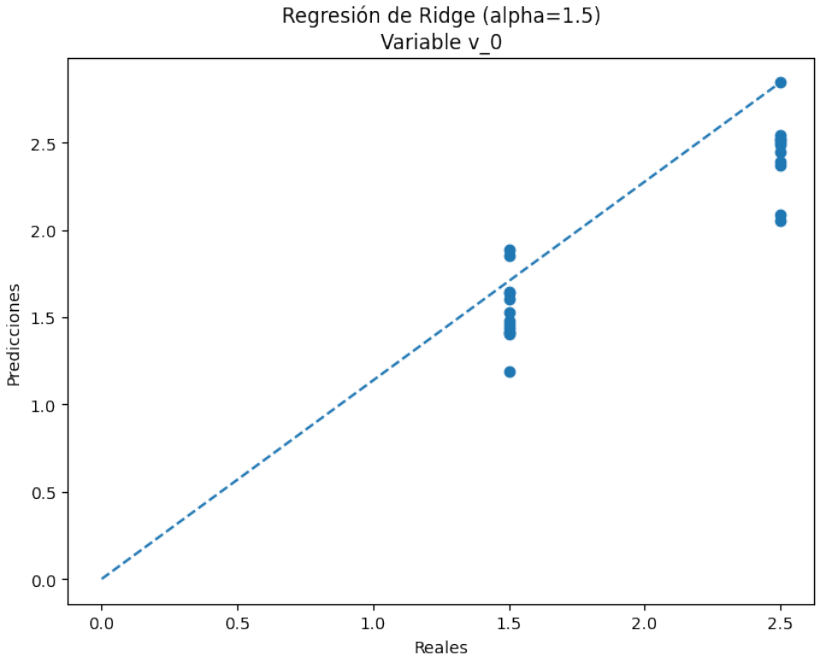
6.2.4 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1 aplicando autocorrelación*



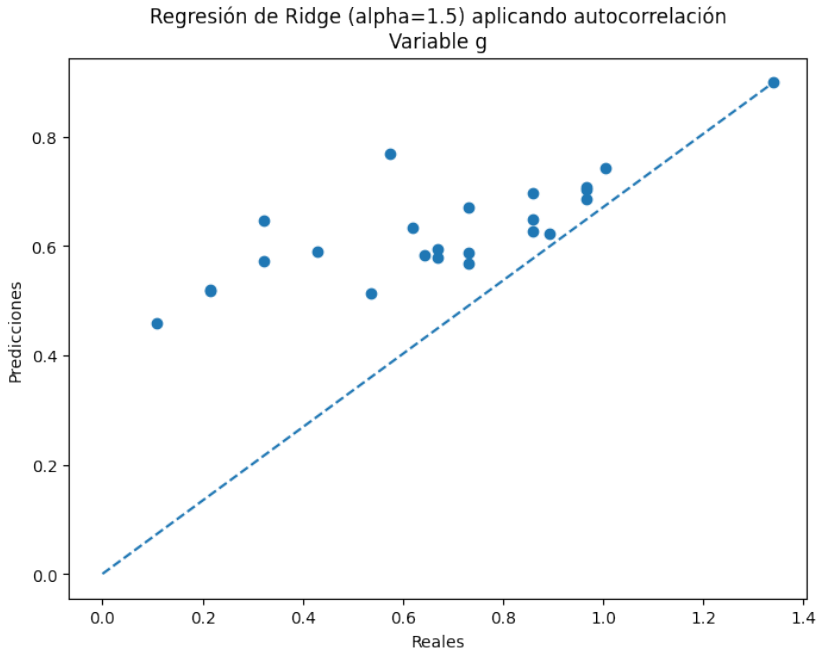


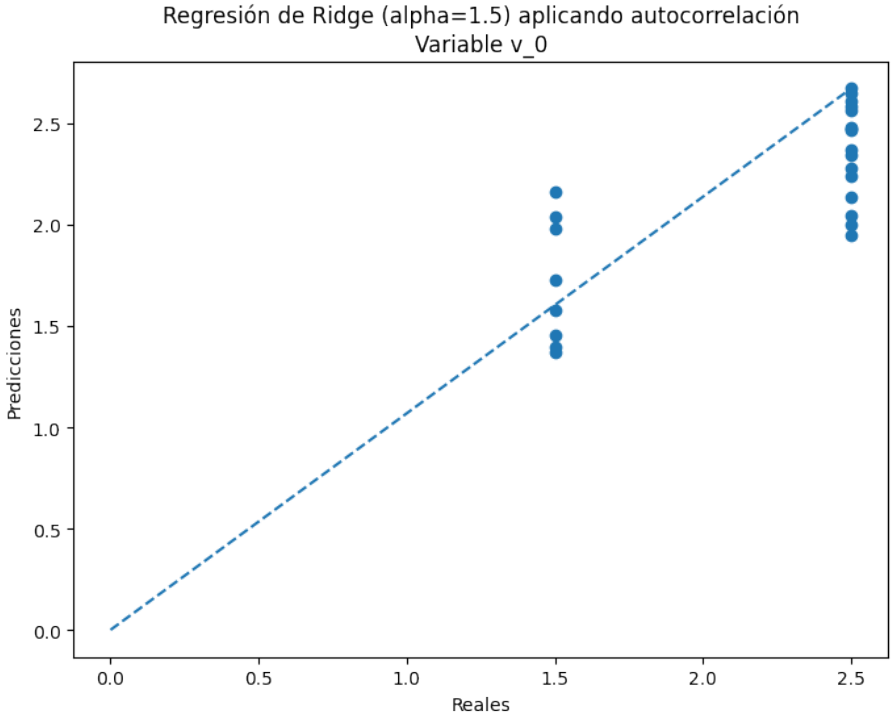
6.2.5 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1.5*



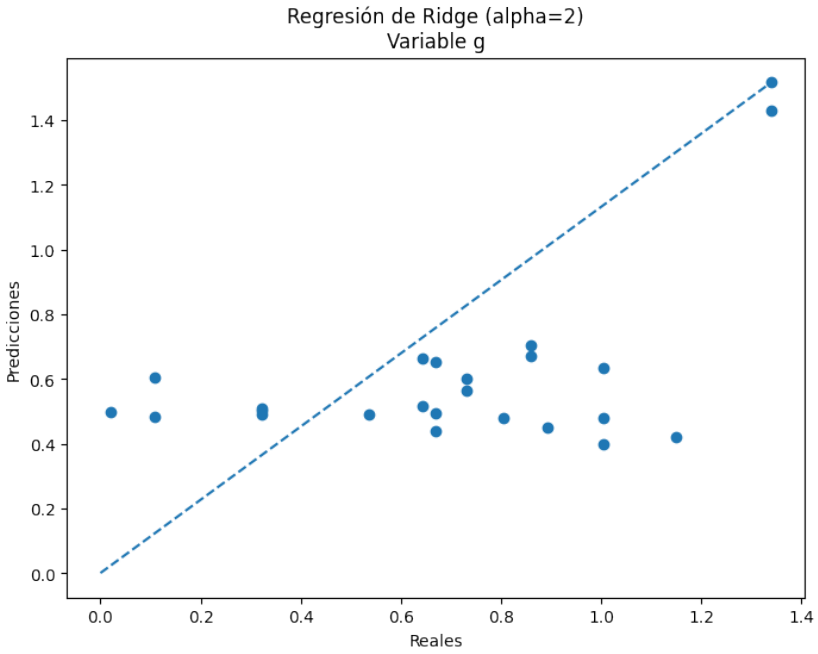


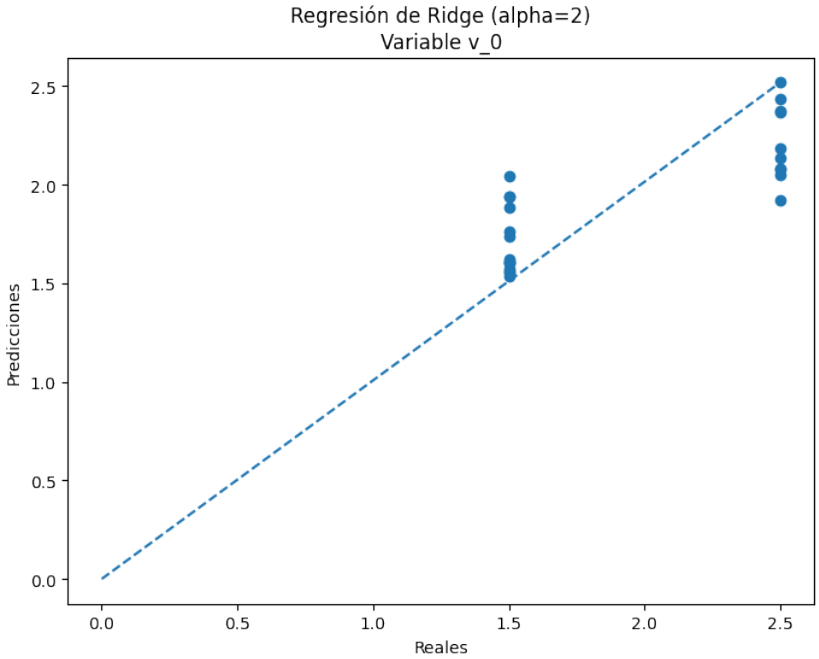
6.2.6 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 1.5 aplicando autocorrelación*



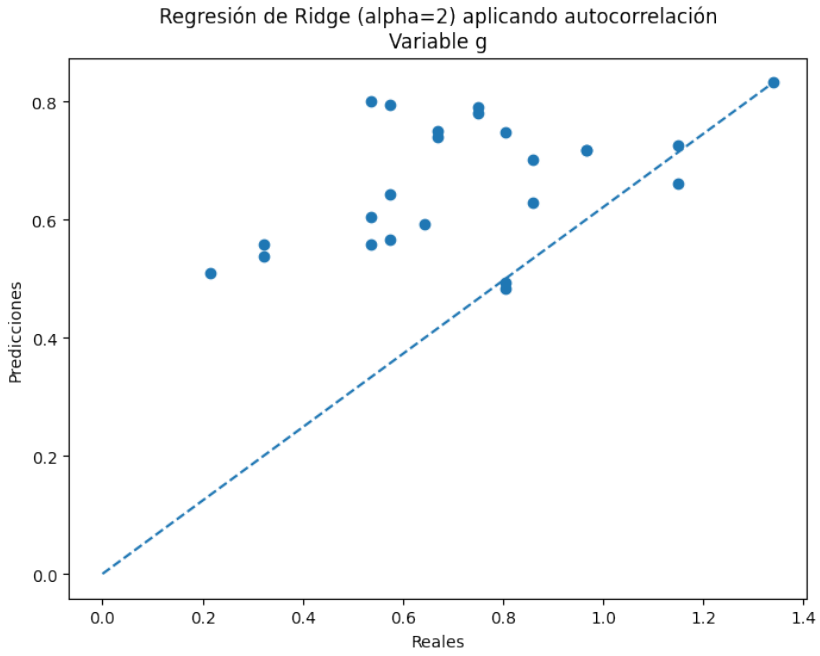


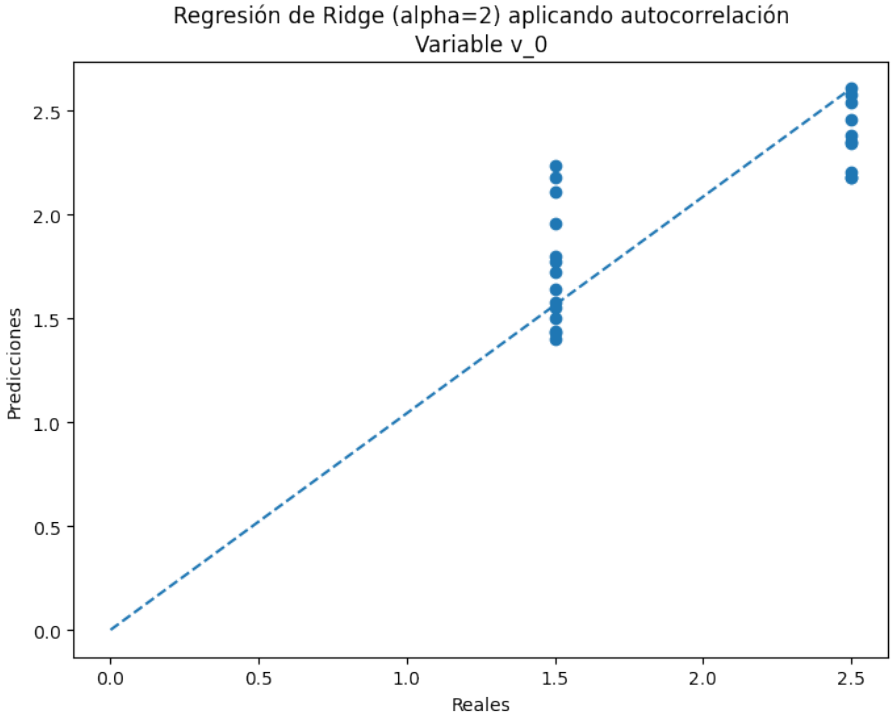
6.2.7 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 2*



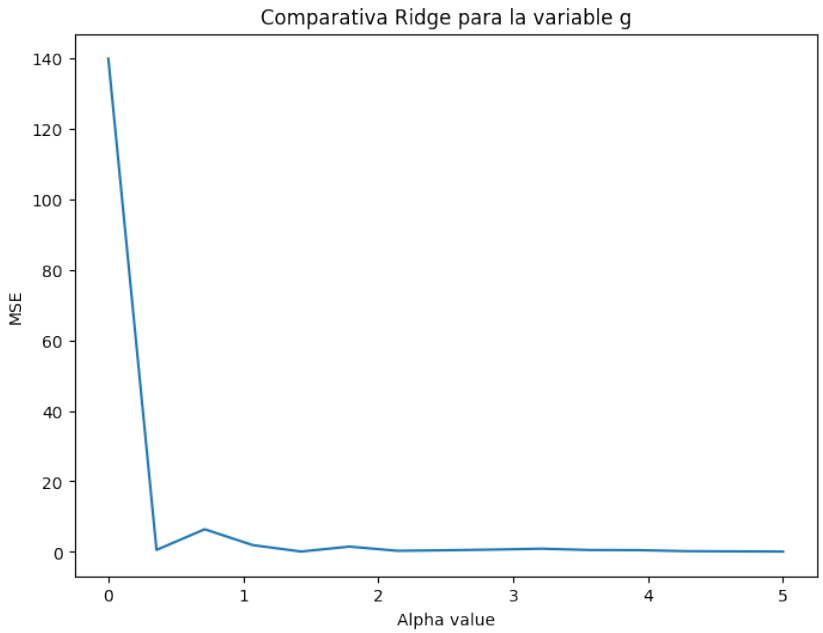


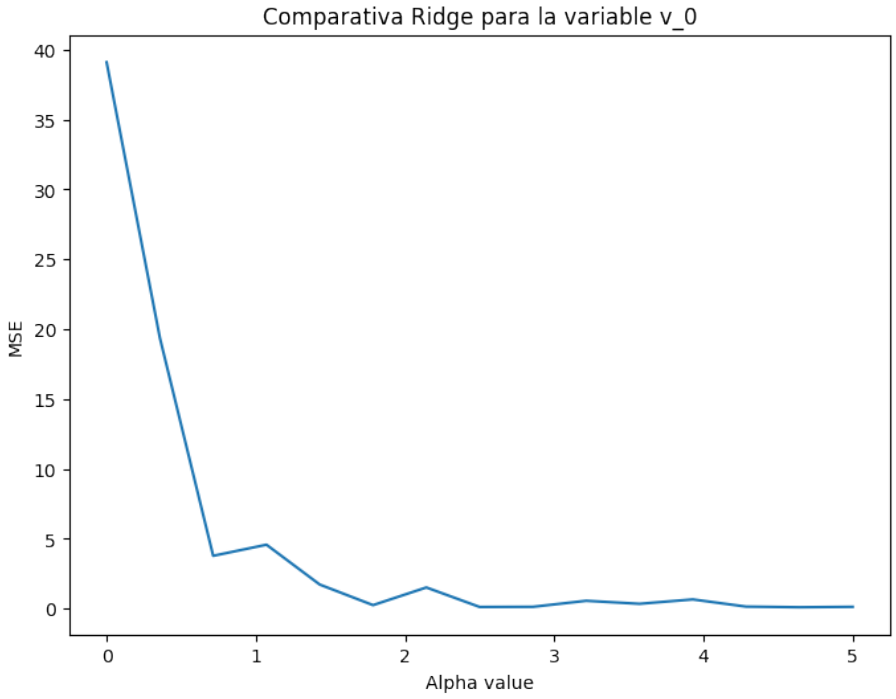
6.2.8 *Machine Learning con algoritmo de regresión lineal de Ridge con alfa = 2 aplicando autocorrelación*





6.2.9 *Evolución de los resultados variando alfa para regresión lineal de Ridge*





Como vemos en ambas gráficas cuando el valor de alfa es cero o cercano a cero, obtenemos un resultado parecido al de la regresión lineal simple, sin embargo, a medida que aumentamos el valor de alfa vamos obteniendo mejores resultados que estabilizan a partir de valores cercanos al 0,5.

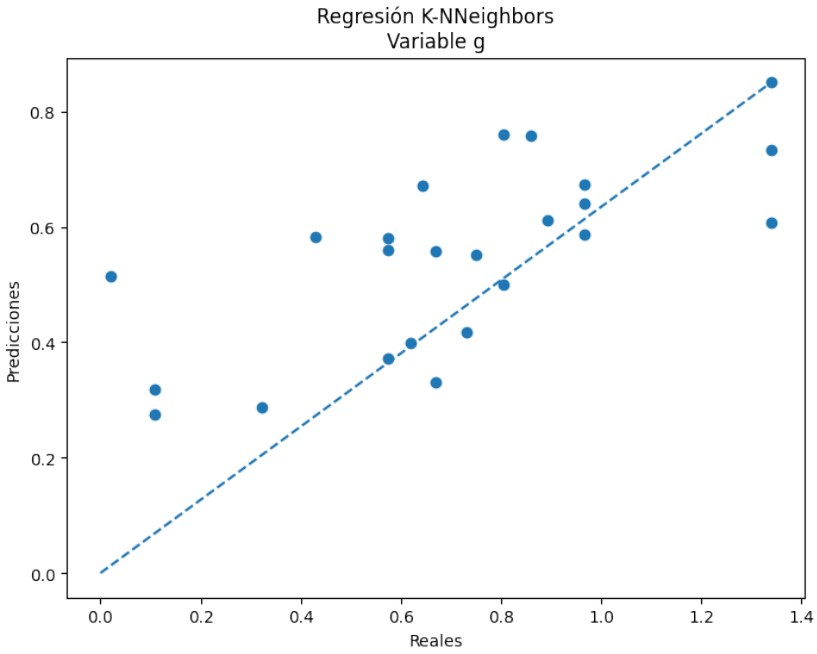
6.3 MACHINE LEARNING CON ALGORITMOS NO LINEALES

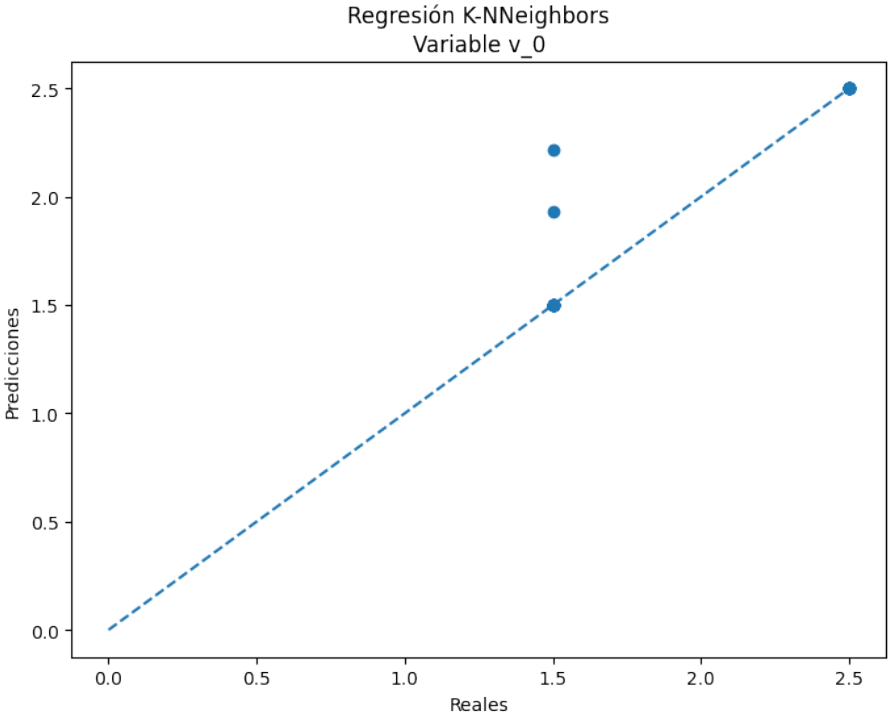
6.3.1 *Machine Learning con algoritmo no lineal de K-Nearest neightbour*

A continuación, se muestra el código utilizado para obtener el resultado de los errores cuadráticos medios después de 1000 ejecuciones para evitar aleatoriedad. Donde podemos diferenciar varias partes:

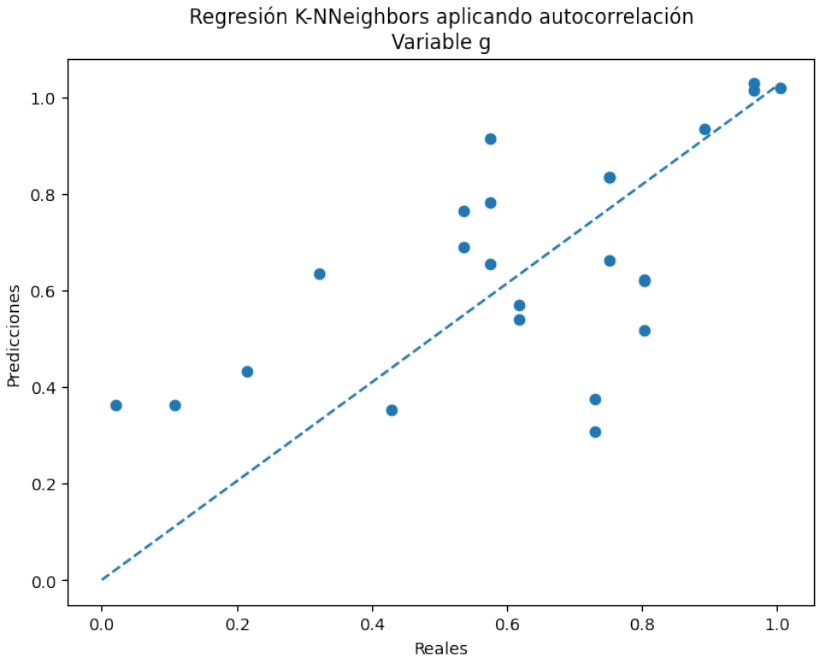
- Creación de los datos de entrenamiento y de tests.
- Escalado de entrada para ajustar los datos al modelo.
- Entrenamiento del modelo donde se llama a algoritmo de regresión de K-NNeighbors, en este caso con 7 vecinos.
- Evaluación del modelo haciendo uso de los errores cuadráticos medios.

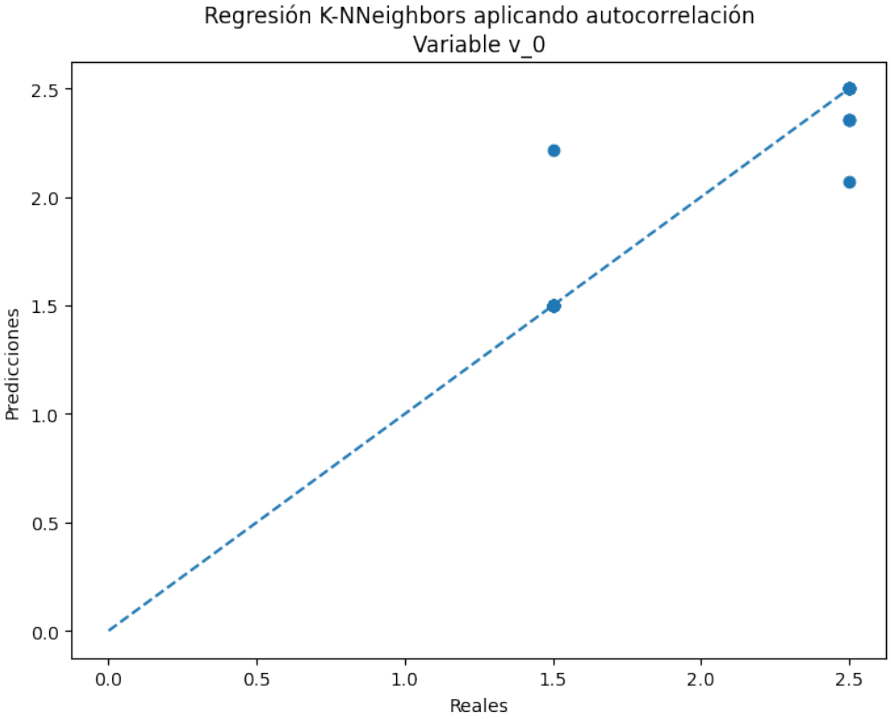
```
#Ejecutamos 1000 veces el algoritmo y obtenemos la media
avg_result_g = []
avg_result_v_0 = []
for times in range (100):
    for fold in range(10):
        #Creamos nuestro X e y de entrada y los sets de entrenamiento y test.
        X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)

        #Escalado de la entrada
        scaler = MinMaxScaler()
        scaler.fit(X_train)
        X_test = scaler.fit_transform(X_test)
        X_train = scaler.transform(X_train)

        #Entrenamiento del modelo
        knn = KNeighborsRegressor(n_neighbors=7).fit(X_train, y_train)

        #Predicciones
        y_pred = knn.predict(X_test)

        #Evaluacion del modelo
        y_test_np = np.array(y_test)
        y_pred_np = np.array(y_pred)
        avg_result_g.append(mean_squared_error(y_test_np[:,0], y_pred_np[:,0]))
        avg_result_v_0.append(mean_squared_error(y_test_np[:,1], y_pred_np[:,1]))

print("Media")
print(np.mean(avg_result_g))
print(np.mean(avg_result_v_0))
resultadoKNeighbors = np.mean(avg_result_g)
resultadoKNeighbors_v_0 = np.mean(avg_result_v_0)
```


Regresión K-NNeighbors
Variable g
Predicciones
Reales
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4


Regresión K-NNeighbors
Variable v_0
Predicciones
Reales
0.0
0.5
1.0
1.5
2.0
2.5

6.3.2 *Machine Learning con algoritmo no lineal de K-Nearest neightbour aplicando autocorrelación*



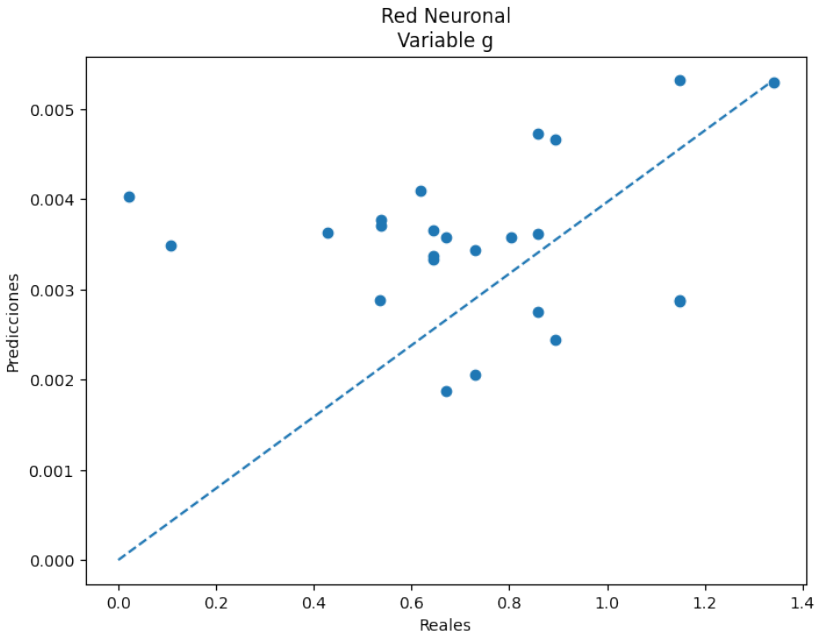


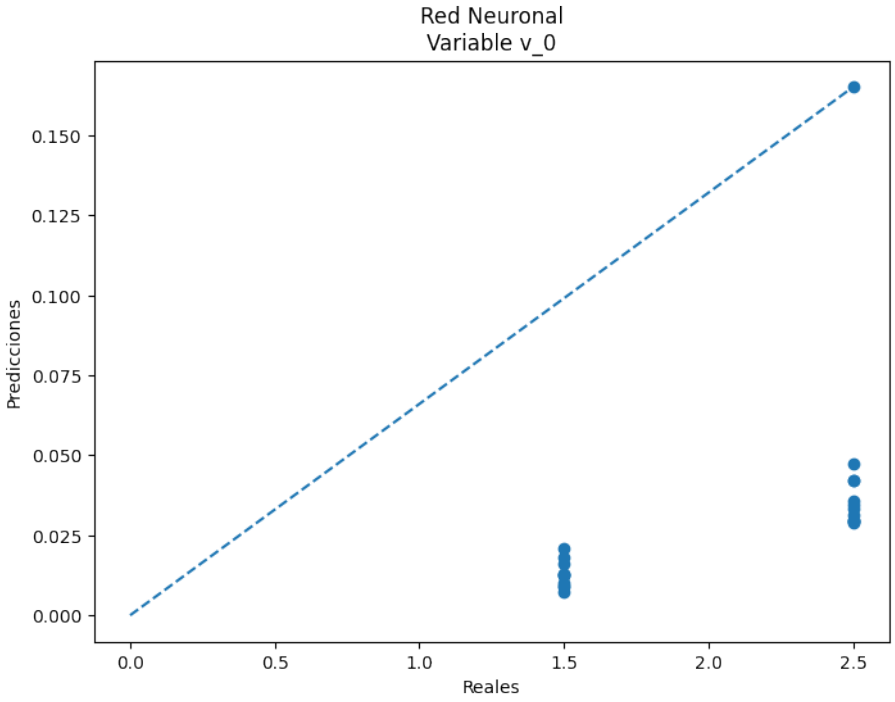
6.3.3 *Machine Learning con algoritmo no lineal - red neuronal*

A continuación, se muestra el código utilizado para obtener el resultado de los errores cuadráticos medios después de 1000 ejecuciones para evitar aleatoriedad. Donde podemos diferenciar varias partes:
- Creación de los datos de entrenamiento y de tests.
- Escalado de entrada para ajustar los datos al modelo.
- Entrenamiento del modelo donde se llama a la función baseline_model que se mostrará también a continuación y se han utilizadado como parámetros 100 unidades y 2 capas de profundidad.
- Evaluación del modelo haciendo uso de los errores cuadráticos medios.

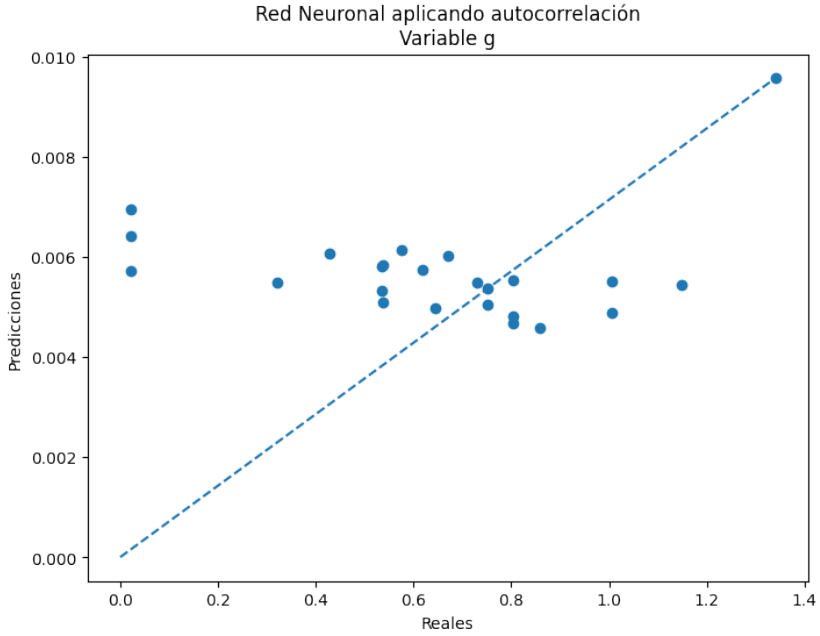
```
def baseline_model(NN_units, NN_layers, X_shape1, Y_shape1):
    # create model
    model = Sequential()
    # Add input layer
    model.add(Dense(NN_units, input_dim=X_shape1, kernel_initializer='normal',
                                    activation='relu'))
    # Add hidden layers
    for k in range(NN_layers):
        model.add(Dense(NN_units, kernel_initializer='normal',
                                    activity_regularizer=l2(0.0001)))
        model.add(Activation('relu'))
    # Add output layer
    model.add(Dense(int(Y_shape1), kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mse', optimizer='adam')

    return model
```

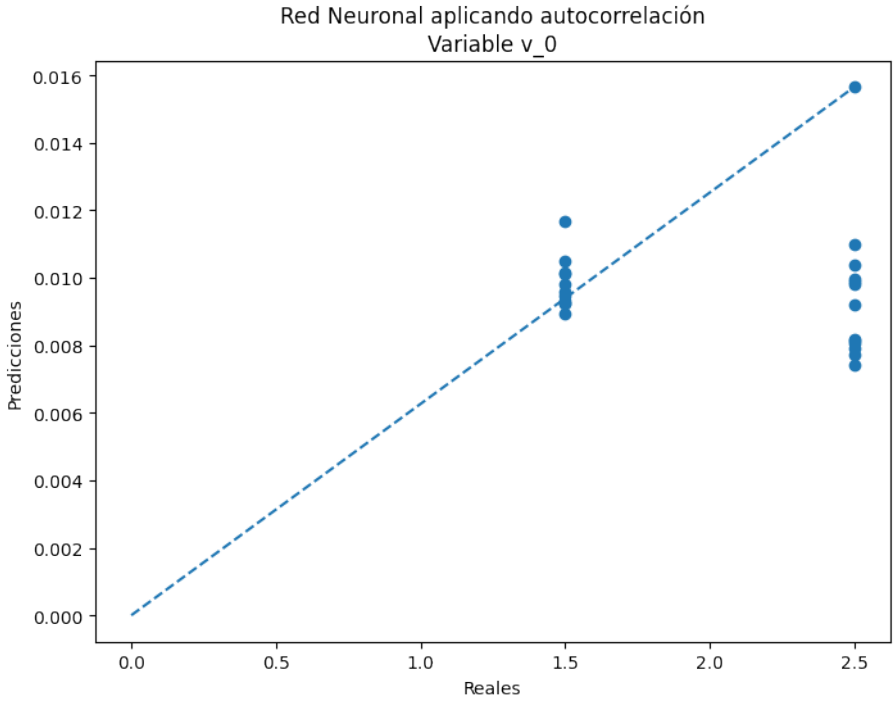
```
#Ejecutamos 100 veces el algoritmo y obtenemos la media
avg_result_g = []
avg_result_v_0 = []
for times in range (10):
    for fold in range(10):
        #Creamos nuestro X e y de entrada y los sets de entrenamiento y test.
        X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2)

        #Escalado de la entrada
        scaler = MinMaxScaler()
        scaler.fit(X_train)
        X_test = scaler.fit_transform(X_test)
        X_train = scaler.transform(X_train)

        #Entrenamiento del modelo
        nn = baseline_model(50, 2, len(X[0]),2)
        nn.fit(np.array(X_train), np.array(y_train))

        #Predicciones
        y_pred = nn.predict(X_test)

        #Evaluacion del modelo
        y_test_np = np.array(y_test)
        y_pred_np = np.array(y_pred)
        avg_result_g.append(mean_squared_error(y_test_np[:,0], y_pred_np[:,0]))
        avg_result_v_0.append(mean_squared_error(y_test_np[:,1], y_pred_np[:,1]))

print("Media")
print(np.mean(avg_result_g))
print(np.mean(avg_result_v_0))
resultadoRedNeuronal = np.mean(avg_result_g)
resultadoRedNeuronal_v_0 = np.mean(avg_result_v_0)
```





6.3.4 *Machine Learning con algoritmo no lineal - red neuronal aplicando autocorrelación*

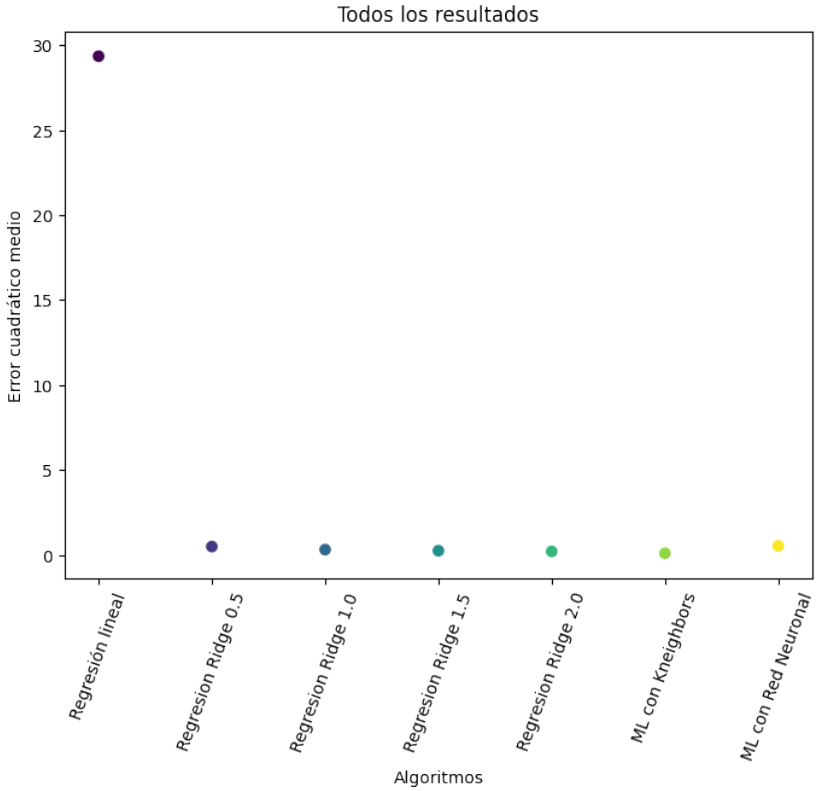




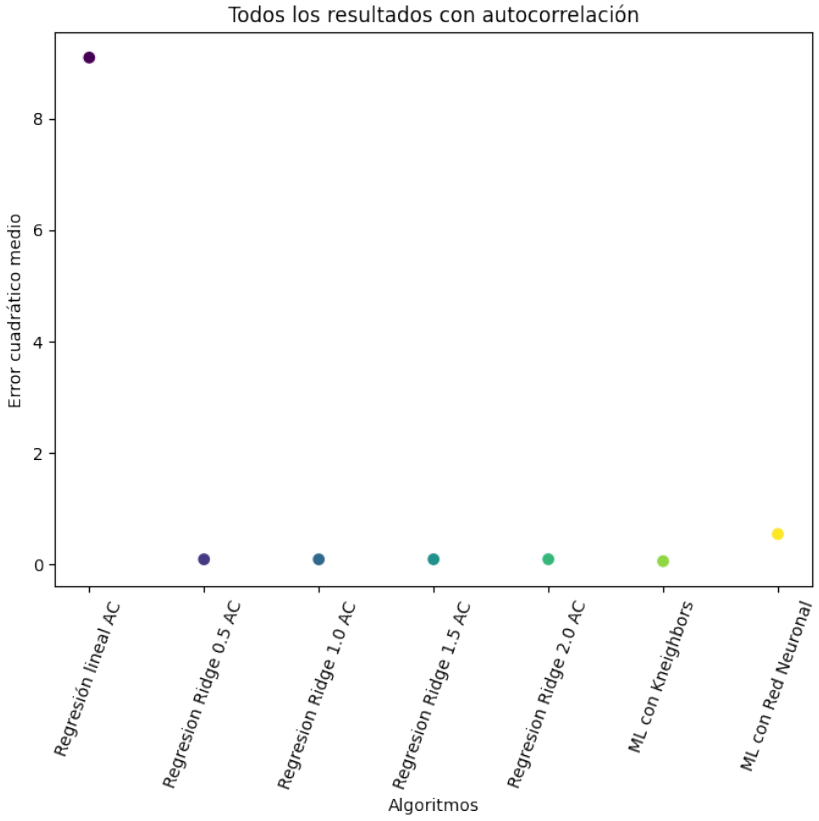
6.4 COMPARACIÓN DE LOS RESULTADOS DE ERRORES CUADRÁTICOS MEDIOS

Después de haber ejecutado un total de 1000 veces cada algoritmo con distintos datos de entrenamiento y de test se ha calculado la media de los errores cuadráticos de cada uno de ellos, tanto para la variable g como variable v_0. Dado que los resultados de g y v_0 son proporcionales, sólo se mostraran los resultados de una de ellas, en este caso, la variable g.

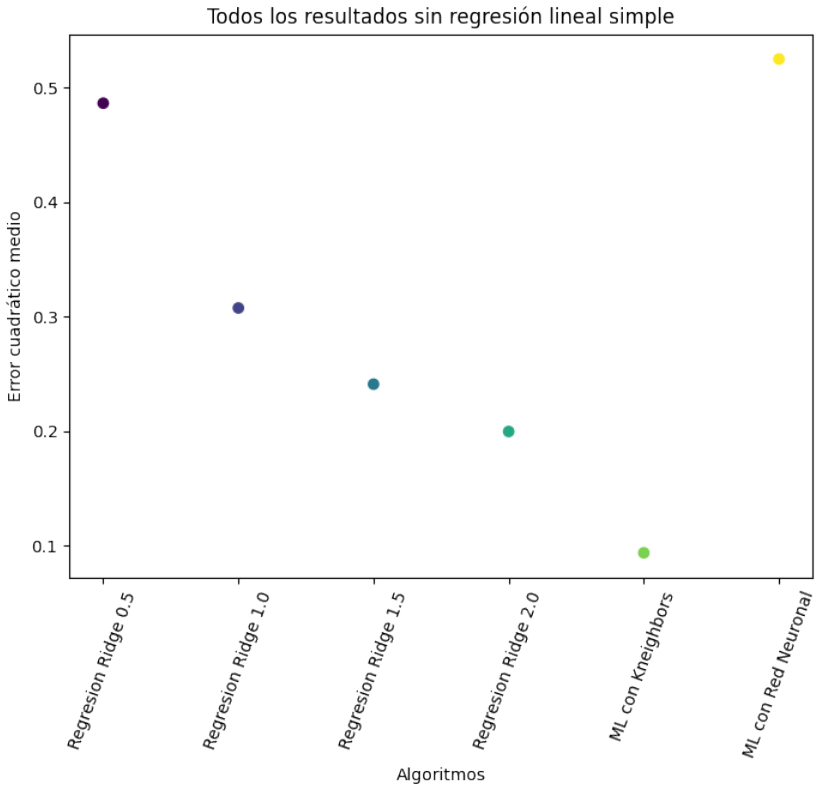
Una vez almacenados todos estos resultados se han ido comparando en diversas gráficas antes y después de aplicar autocorrelación, siendo los siguientes los resultados obtenidos.



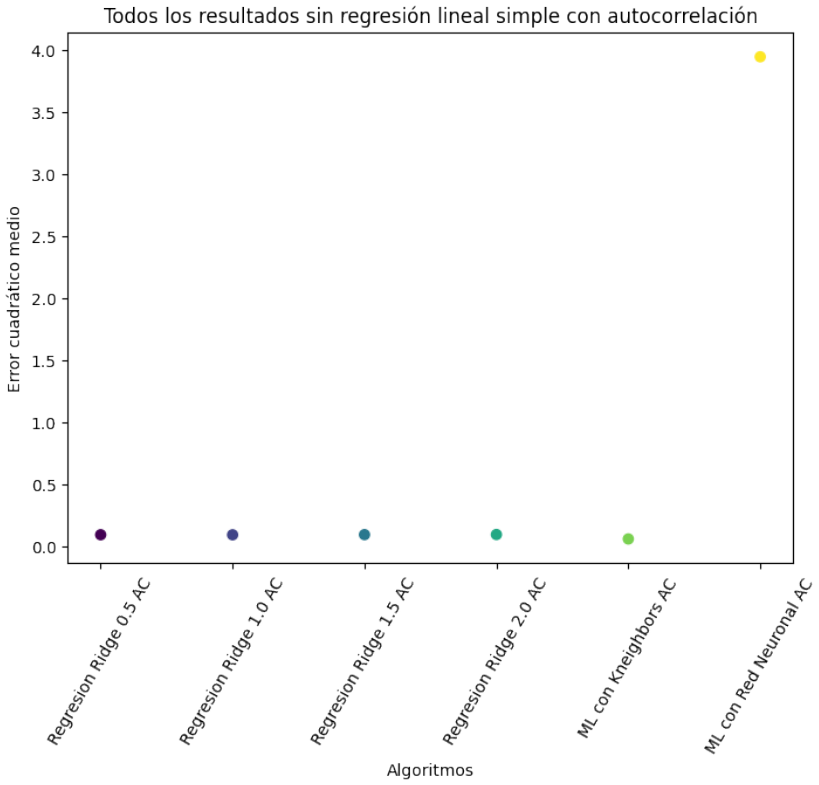
Se observa claramente que el resultado obtenido con regresión lineal es el peor resultado, llegando aproximadamente a un error cuadrático medio de treinta. Es el resultado esperado pues es el algoritmo más simple y el que menos complejidad algoritmica tiene.



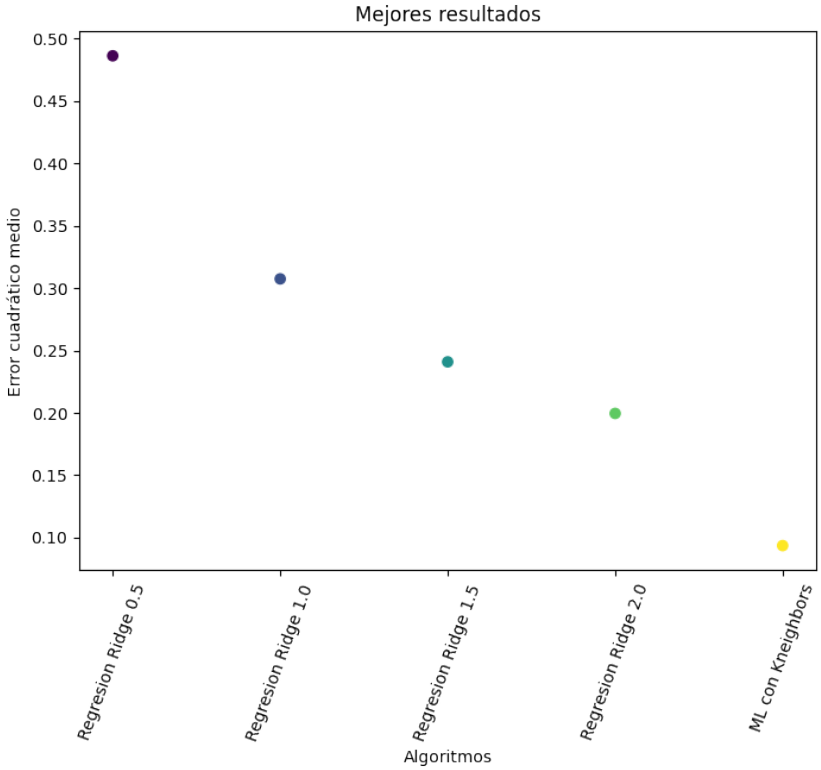
Aunque mejora considerablemente el error cuadrático medio después de aplicar autocorrelación, seguimos sin obtener un resultado capaz de ser comparado con el obtenido en el resto de algoritmos. Por ello, para la siguiente gráfica, será excluido para conseguir una mejor comparativa.

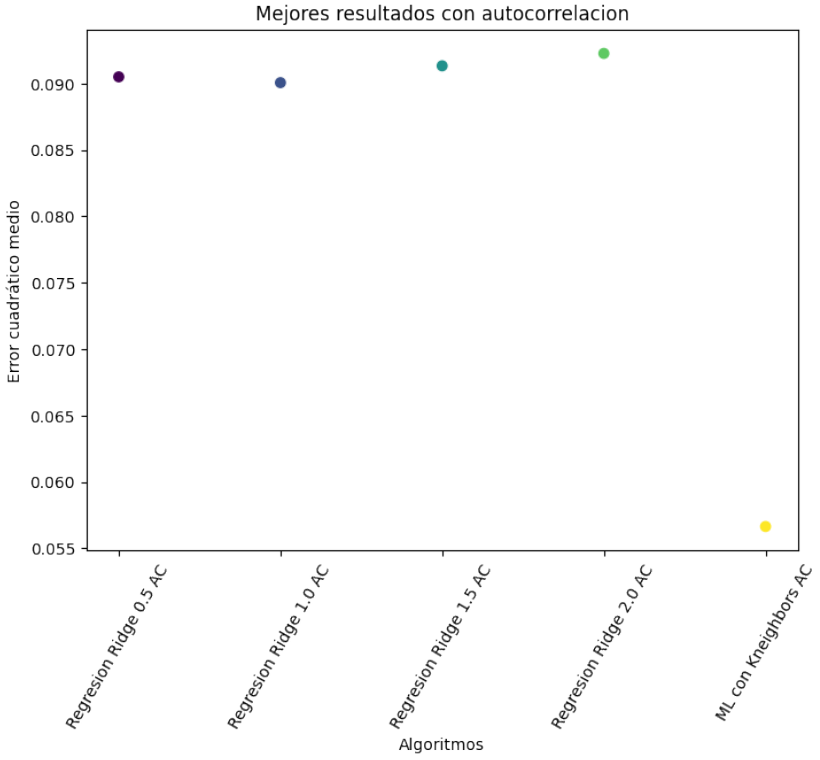


En esta gráfica ya si podemos observar errores cuadráticos medios por debajo de uno, lo cual son resultados bastane buenos, sin embargo, observamos que el algorimto de K-NNeighbors es el que obtiene el mejor resultado seguidos por los algoritmos de Ridge.



Después de aplicar autocorrelación, observamos que el algoritmo de machine learning basado en la Red Neuronal no es capaz de generar buenos resultados por lo tanto será excluido en la siguiente comparativa.





Al quedarnos únicamente con los cinco algoritmos con los que se han obtenido los mejores resultados, se puede observar que el algoritmo de K-NNeighbors es el que

obtiene mejores resultados siendo su error cuadrático medio de 0,1 antes de aplicar autocorrelación y muy próximo a 0,055, mientras que los algoritmos de Ridge únicamente se acercan a 0,1 después de haber aplicado autocorrelación.

7

CONCLUSIONES

7.1 RELEVANCIA DEL TFG PARA LA NEUROCIENCIA Y LA MEDICINA

Gracias a las simulaciones del modelo cerebral hemos sido capaces de, en primer lugar, obtener un gran conjunto de datos, entre ellos la señal del electroencefalograma, que es una de las señales no-invasivas más conocida en el ámbito clínico. En segundo lugar, analizando estas señales hemos sido capaces de detectar diferentes tipos de gráficas, algunas muy realistas y otras, sin embargo, que serían prácticamente imposible de obtener en la vida real pero que han sido de utilidad para entender entre que rangos de valores debemos de trabajar.

Posteriormente, hemos estudiado si es posible realmente estimar los parámetros del modelo, como el cociente de excitación, E, e inhibición: E/I, que ha dado lugar a las distintas propiedades de la señal del EEG. Mediante herramientas de machine learning (ML), hemos podido encontrar automáticamentelas regiones de interés del espacio de parámetros del modelo cortical y que se relacionan con cambios en la señal del EEG.

Es imporante remarcar que gracias a este estudio, hemos sido capaces de estudiar que las variaciones del parámetro E/I pueden influir directamente en el diagnóstico de algunas enfermedades como la epilepsia o el autismo. De tal forma que cuando se aumenta el cociente E/I, existe una mayor probabilidad de que se dén enfermedades como la epilepsia en la cual, hay un gran aumento de la actividad neuronal, o de igual forma al disminuir este cociente E/I, existe una mayor probabilidad de que dén enfermedades como el autismo, donde se puede se produce una disminución considerable de actividad neuronal, como se demuestra en el modelo del artículo Rubenstein [2003].

Gracias a la herramienta desarrollada en este tfg, hemos sido capaces de, usando un modelo del cerebro, poder leer señales del EEG y producir estimaciones de cambios en E/I. De tal forma que a partir de los resultados obtenidos en las simulaciones y tras hacer un estudio de los mejores algoritmos de machine learning, se ha podido obtener con bastante exactitud cual sería su correspondiente cociente E/I.

Finalmente, indicar también que hemos tenido algunas limitaciones durante el desarrollo del proyecto ya que el modelo cerebral usado es un modelo simplificado de una pequeña parte del cerebro y no tiene en cuenta conexiones con otras regiones del cerebro, como puede ser por ejemplo, los impulsos procedentes del tálamo. Además el modelo usa neuronas de tipo de LIF, que son neuronas simplificadas, que facilitan el cálculo del potencial con vistas a su implementación hardware, reduciendo el uso de recursos lógicos y optimizando la velocidad de operación del sistema. Por otro lado, la fórumla utilizada para generar el EEG, es una fórmula simplificada, con la que hemos sido capaces de mantenernos cercanos a los valores esperados pero que, sin embargo, no tiene en cuenta otros factores que pueden influir en su resultado.

8

TRABAJO FUTURO

Como continuación de este trabajo de fin de grado y como en cualquier otro proyecto de investigación, existen diversas líneas de investigación que quedan abiertas y en las que es posible continuar trabajando. En este caso, en el futuro se pueden trabajar en 2 líneas de mejora:

- Mejorar el modelo superando las limitaciones mencionadas anteriormente, haciendo uso de una mayor cantidad de parámetros (por ejemplo, impulsos provenientes del tálamo), los cuales se pueden modificar en las simulaciones con NEST y así obtener una mayor variedad de resultados. De ésta forma se podría estudiar que influencia tienen sobre el resto de parámetros estudiados.

- Aplicar el modelo a registros reales de señales del EEG tomados en pacientes con patologías clínicas. De esta forma, se podría observar si realmente el modelo es capaz de analizar los datos del EEG para estudiar la probabilidad de que un paciente sufra una determinada enfermedad.

BIBLIOGRAFÍA

Electroencefalografía (eeg). URL https://www.mayoclinic.org/es-es/tests-procedures/eeg/about/pac-20393875.

Carmen Cavada. Introducción histórica a la neurociencia. URL https://www.senc.es/introduccion-historica-a-la-neurociencia/.

OpenMP Documentation. *OpenMP Guide*. OpenMP. URL https://www.openmp.org/resources/refguides/.

Iván de Jesús Uscanga Uscanga Emanuel Dzul. El cerebro en el tiempo (recorrido de la neurociencia). URL https://www.uv.mx/cienciauv/blog/cerebroeneltiemponeurociencia/.

Jonathan García-allen. Tipos de neuronas: características y funciones. URL https://psicologiaymente.com/neurociencias/tipos-de-neuronas.

Noei S. Martínez-Cañada, P. and S. Panzeri. Methods for inferring neural circuit interactions and neuromodulation from local field potential and electroencephalogram measures. *Brain Informatics*, 8, 2021. doi: 10.1186/s40708-021-00148-y. URL https://doi.org/10.1186/s40708-021-00148-y.

Pablo Martínez-Cañada, Torbjørn V. Ness, Gaute T. Einevoll, Tommaso Fellin, and Stefano Panzeri. Computation of the electroencephalogram (eeg) from network models of point neurons. *PLOS Computational Biology*, 17:1–41, 04 2021. doi: 10.1371/journal.pcbi.1008893. URL https://doi.org/10.1371/journal.pcbi.1008893.

Alberto Mazzoni, Stefano Panzeri, Nikos K. Logothetis, and Nicolas Brunel. Encoding of naturalistic stimuli by local field potential spectra in networks of excitatory and inhibitory neurons. *PLOS Computational Biology*, 4, 12 2008. doi: 10.1371/journal.pcbi.1000239. URL https://doi.org/10.1371/journal.pcbi.1000239.

Whittingstall K. Brunel N. Logothetis N. K. Panzeri S. Mazzoni, A. Understanding the relationships between spike rate and delta/gamma frequency bands of lfps and eegs using a local cortical network model. page 956–972, 2010. doi: 10.1016/j.neuroimage.2009.12.040. URL https://doi.org/10.1016/j.neuroimage.2009.12.040.

Merzenich M. M. Rubenstein, J. L. Model of autism: increased ratio of excitation/inhibition in key neural systems. page 255–267, 2003. doi: 10.1034/j.1601-183x.2003.00037.x. URL https://doi.org/10.1034/j.1601-183x.2003.00037.x.