UNIVERSIDAD DE ANTIOQUIA
1803

ACONDICIONAMIENTO Y ANÁLISIS DE SEÑALES CEREBRALES

BIOSEÑALES Y SISTEMAS

YÉSIKA ALEXANDRA AGUDELO CC 1035875565
JULIANA MONCADA AGUDELO CC 1037646000

Con el objetivo de implementar una estrategia que permita extraer información neuronal de una señal EEG contaminada por ruido, se construyó una librería en Python, en la cual se adquirieron cuatro señales de interés para luego ser procesadas con filtros lineales, división por segmentos o épocas (cada uno por 2 segundos) y su respectiva detección de épocas con características estadísticas atípicas utilizando diversos métodos para su rechazo de artefactos, como valores extremos, tendencia lineal, improbabilidad y patrón espectral.

A) A continuación se explica paso a paso los códigos implementados para cumplir dicho fin.

- Como primer paso, se importan las librerías necesarias para el desarrollo del código, como numpy, para soporte para vectores y matrices (funciones matemáticas de alto nivel); matplotlib, para la generación de gráficos a partir de datos contenidos en arrays, listas o tuplas; LinearFIR, como código implementado para la etapa de filtrado (filtro tipo FIR); y scipy, para la utilización de módulos para optimización, funciones especiales, procesamiento de señales e imágenes.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import LinearFIR
4  from scipy.stats import kurtosis
5  from scipy import signal
6  from numpy import dtype
7  from scipy.stats import linregress
8
```

Figura 1. *Importación de librerías*

- Luego, se crea la función *'CargarSenal',* la cual utiliza un módulo de numpy llamado loadtxt, el cual permite cargar archivos de texto. Este recibe como argumentos:
    - Ruta→ el nombre de la señal.
    - Delimitador→ encargado de dividir la señal cada que encuentra una coma.
    - Eliminar→ salta las primeras líneas que se deseen.
    - Columnas→ número de columnas que se desean obtener.

Asimismo, se aplica el filtrado de la señal con el código hecho por el profesor John Fredy Ochoa, el cual implementa un filtro tipo FIR, que presenta una respuesta al impulso de duración finita, presentando la posibilidad de obtener una fase exactamente lineal.
De LinearFIR, se utiliza una función llamada *eegfiltnew,* la cual recibe como argumentos:

- Senal→ nombre de la señal a filtrar.
- Srate→ frecuencia de muestreo.
- Locutoff→ frecuencia de corte baja
- Hicutoff→ frecuencia de corte alta
- revfilt→ número de tipo de filtro
- Plot→ si se quiere mostrar o no las gráficas del filtro.

Como resultado, se obtiene la señal filtrada, con una frecuencia de muestreo de 250 Hz y con un paso de frecuencias de 1 a 50 Hz.

```
#Function to load signal
def CargarSenal(ruta,delimitador,eliminar,columnas):
    data=np.loadtxt(ruta,delimiter=delimitador,skiprows=eliminar,usecols=columnas)
    return data

my_data_1=CargarSenal('P1_RAWEEG_2018-11-15_Electrobisturí1_3min.txt',',',6,[1,2,3,4,5,6,7,8])
my_data_2=CargarSenal('P1_RAWEEG_2018-11-15_Electrobisturí2_2min.txt',',',6,[1,2,3,4,5,6,7,8])
my_data_3=CargarSenal('P1_RAWEEG_2018-11-15_FinProcedimiento_53min.txt',',',6, [1,2,3,4,5,6,7,8])
my_data_4=CargarSenal('P1_RAWEEG_2018-11-15_OjosCerrados_2min.txt',',',6, [1,2,3,4,5,6,7,8])

#Signals filter=LinearFIR
filtro1=LinearFIR.eegfiltnew(my_data_1, 250, 1, 50, 0, 0)
filtro2=LinearFIR.eegfiltnew(my_data_2, 250, 1, 50, 0, 0)
filtro3=LinearFIR.eegfiltnew(my_data_3, 250, 1, 50, 0, 0)
filtro4=LinearFIR.eegfiltnew(my_data_4, 250, 1, 50, 0, 0)
```

Figura 2. *Función para cargar la señal*

- Para el vector de tiempo, se utilizó el módulo de numpy *arange*, el cual devuelve valores espaciados uniformemente dentro de un intervalo dado, desde 0 hasta la longitud de los datos del filtro, con paso de (1/250=período).

```
#Vector time creation
tiempo_f1=np.arange(0,filtro1.shape[0]/250,1/250)
tiempo_f2=np.arange(0,filtro2.shape[0]/250,1/250)
tiempo_f3=np.arange(0,filtro3.shape[0]/250,1/250)
tiempo_f4=np.arange(0,filtro4.shape[0]/250,1/250)

#Creating the DC level for visualization
DC=[0,150,250,350,450,550,650,750]
```

Figura 3. *Creación del vector de tiempo y el nivel DC para su visualización.*

- Para la segmentación de la señal por épocas, se crea una función '*segmentacion',* la cual recibe como parámetros:
    - Filtro→ nombre de la señal a segmentar.
    - Tiempo→ vector de tiempo a utilizar.
    - Fs→ frecuencia de muestreo.
    - Época→ el número de segundos que requiere por segmento.

Como retorno, se obtienen dos array, uno con los datos de la señal segmentada y otro con el vector de tiempo segmentado. Esto se logra evidenciar en la figura 4.

```
34  #Function for signal segmentation
35  def segmentacion(filtro,tiempo,fs,epoca):
36
37      Dim=filtro.shape
38      Residuo=Dim[0]%(fs*epoca)
39      Total_Time=Dim[0]/fs
40      Segmentacion_Data=np.split(filtro[0:Dim[0]-Residuo,:], int(Total_Time//epoca))
41      time=np.split(tiempo[0:Dim[0]-Residuo], int(Total_Time//epoca))
42
43      return np.array(Segmentacion_Data),np.array(time)
44
45  #Segmentation function implementation
46  S1_array,time1=segmentacion(filtro1,tiempo_f1,250,2)
47  S2_array,time2=segmentacion(filtro2,tiempo_f2,250,2)
48  S3_array,time3=segmentacion(filtro3,tiempo_f3,250,2)
49  S4_array,time4=segmentacion(filtro4,tiempo_f4,250,2)
50
```

Figura 4. *Función para segmentar la señal*

- A continuación, se implementan los cuatro métodos para el rechazo de artefactos. En la figura 5, se muestra el método de rechazo por valores extremos, en donde se utiliza un umbral estándar de los datos, con el fin de seleccionar un rango establecido, para finalmente, eliminar los que están fuera de este.
  Para este objetivo, se implementó una función '*Rechazo'*, la cual recibe:
    - Umbralmaximo→ umbral superior del segmento.
    - Umbralminimo→ umbral inferior del segmento.
    - SegmentoArray→ Array del segmento a implementar.

```python
#FIRST METHOD: REJECTION FOR EXTREME VALUES

#This function is make for search the maximums segments and the minimum segments that don't satisfy the condition,
#and then, they are rejected.
def Rechazo (umbralmaximo,umbralminimo,SegmentoArray):
    Vunos=np.ones((SegmentoArray.shape[0],1))
    valormaximo=SegmentoArray.max(axis=1)
    valorminimo=SegmentoArray.min(axis=1)

    for i in range(0,8):
        index=np.where(valormaximo[:,i]>umbralmaximo)
        Vunos[index]=0
        index=np.where(valorminimo[:,i]<umbralminimo)
        Vunos[index]=0

    Rechazo=Vunos.transpose()[0]
    Data=SegmentoArray[Rechazo==1,:,:]
    size=Data.shape
    Signal=np.zeros((size[0]*size[1],size[2]),dtype=np.int)
    for i in range(0,size[2]):
        Signal[:,i]=Data[:,:,i].ravel()

    return Signal
```

Figura 5. *Función Primer método-Rechazo por valores extremos*

Esta función, al encontrar los máximos y los mínimos en el rango establecido, le asigna un índice cero a los que no cumplen con la condición y un uno a los que sí los satisfacen, eliminando finalmente estos primeros y dejando los segmentos 'marcados' con el 1. Retornando la señal que exclusivamente se encuentra en el umbral seleccionado.

En la figura 6 se muestra la implementación de la función y su respectiva gráfica.

```python
#SenalSeg1_Valores=Rechazo(75,-75,S1_array)
#SenalSeg2_Valores=Rechazo(75,-75,S2_array)
#SenalSeg3_Valores=Rechazo(75,-75,S3_array)
#SenalSeg4_Valores=Rechazo(75,-75,S4_array)

# plt.plot(SenalSeg4_Valores+DC)
# plt.title('REJECT FOR EXTREME VALUES')
# plt.xlabel('Time [s]')
# plt.show()
```

Figura 6. *Implementación de la función de la figura 6 y respectivo plot.*

En la figura 7 se muestran las gráficas resultantes de cada una de las señales EEG, con la aplicación del primer método de rechazo por valores extremos, con cada uno de los 8 canales.

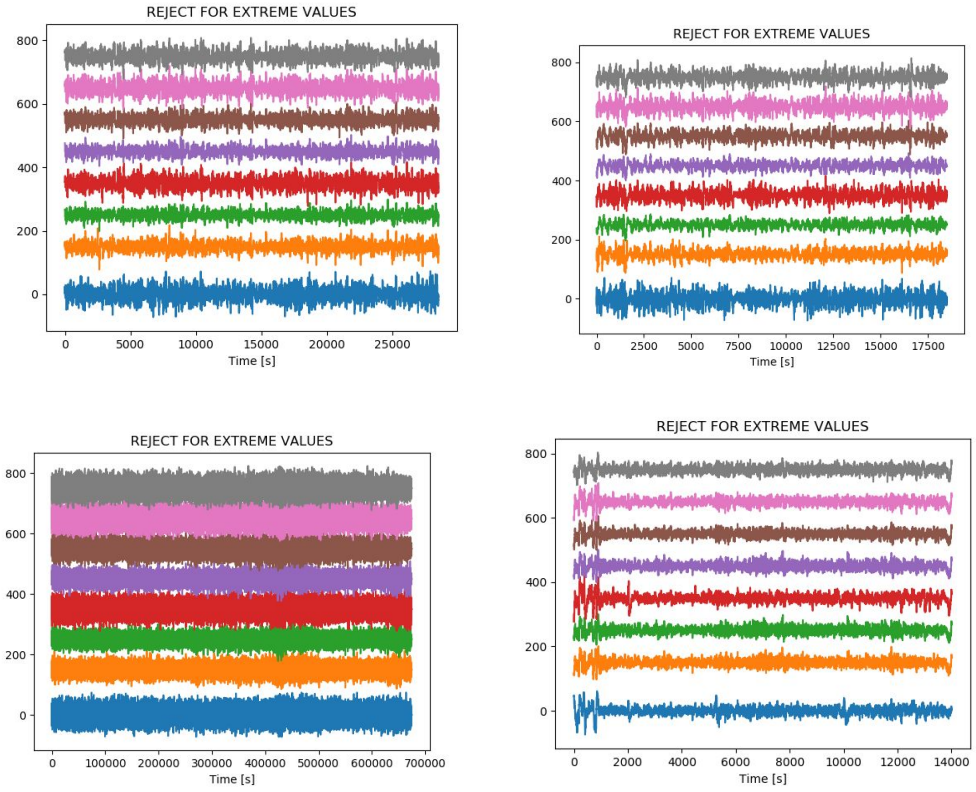

Figura 7*.Resultado de cada una de las señales aplicadas al primer método, en orden de izquierda a derecha.*

- El segundo método utilizado fue el rechazo por tendencia lineal, para el cual se creó una función *'Tendencia_Lineal',* la cual acepta o rechaza los datos en función de la pendiente mínima de una línea recta y su respectivo ajuste de la actividad del EEG. Esta recibe como argumentos:
    - SegmentoArray→ Array del segmento a implementar.
    - Umbralmax→ Umbral máximo
    - Umbralmin→ Umbral mínimo
    - time→ vector de tiempo

```
#SECOND METHOD: REJECT FOR LINEAL TENDENCY
def TENDENCIA_LINEAL(SegmentoArray,umbralmax,umbralmin,time):
    Dim=SegmentoArray.shape
    Unos=np.ones((Dim[0],1),dtype=np.int)
    Pendiente=np.zeros((Dim[0],Dim[2]),dtype=np.float32)

    for j in range (0,Dim[2]):
        for i in range(0,Dim[0]):
            Regresion=np.polyfit(time[j],SegmentoArray[i,:,j],1)
            Pendiente[i,j]=Regresion[0]



    for k in range(0,Dim[2]):
        index=np.where(Pendiente[:,k]>umbralmax)
        Unos[index]=0
        index=np.where(Pendiente[:,k]<umbralmin)
        Unos[index]=0

    Tendencia=Unos.transpose()[0]
    Data=SegmentoArray[Tendencia==1,:,:]
    size=Data.shape
    Signal=np.zeros((size[0]*size[1],size[2]),dtype=np.int)
    for i in range(0,size[2]):
        Signal[:,i]=Data[:,:,i].ravel()
    return Signal
```

Figura 8. *Función segundo método-Rechazo por tendencia lineal.*

Dicha función retorna la señal sin los segmentos rechazados. Esto se logró debido a que, si el umbral seleccionado no cumplía con la pendiente de la recta de regresión de los datos del EEG, se eliminaban para formar un nuevo segmento que sí cumplía con los valores establecidos.

En la figura 9 se visualiza la implementación de la señal y su código para la mostrar la gráfica en pantalla.

```

# SenalSeg1_Regresion=TENDENCIA_LINEAL(S1_array+DC,4,-4,time1)
# SenalSeg2_Regresion=TENDENCIA_LINEAL(S2_array+DC,4,-4,time2)
# SenalSeg3_Regresion=TENDENCIA_LINEAL(S3_array+DC,4,-4,time3)
# SenalSeg4_Regresion=TENDENCIA_LINEAL(S4_array+DC,4,-4,time4)
# plt.title('REJECT FOR LINEAL TENDENCY')
# plt.xlabel("Time[s]")
# plt.plot(SenalSeg1_Regresion+DC)
# plt.show()
```

Figura 9*. Implementación de la función de la figura 7 con su respectivo plot.*

En la figura 10 se observan los ocho canales de cada señal utilizada y su rechazo por tendencia lineal.

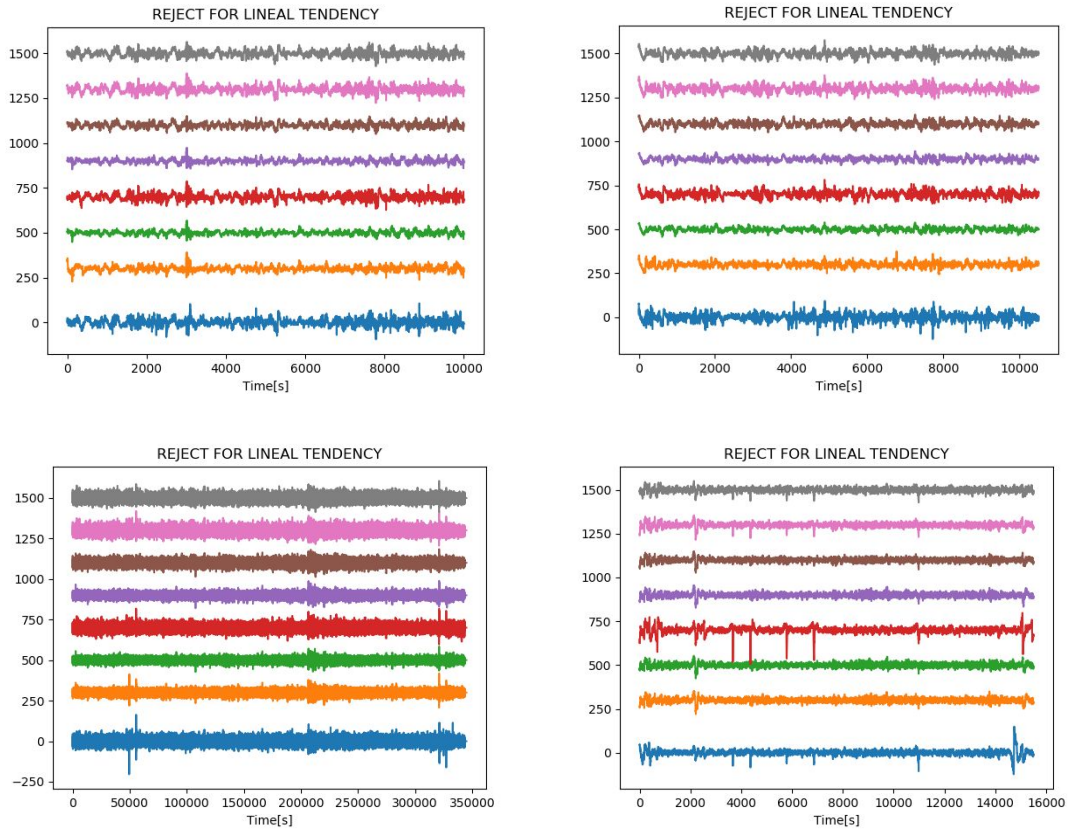

Figura 10.*.Resultado de cada una de las señales aplicadas al segundo método, en orden de izquierda a derecha.*

- Para el tercer método, se utilizó el rechazo por improbabilidad, la cual recibe los mismos argumentos que el método anterior, y en donde se utiliza la distribución de valores de datos y su curtosis para detectar los artefactos. Esta distribución, posee un valor estándar, con la cual se comparan los datos resultantes, con el fin de rechazar los datos que se encuentran muy alejados de este.

```
#THIRD METHOD: REJECT FOR IMPROBABILITY
#This function is created to search the maximums and minimums segments that don't satisfy the conditions...then, they are rejected.
def Kurtosis (umbralmaximo,umbralminimo,SegmentoArray):
    Unos=np.ones((SegmentoArray.shape[0],1))
    Kurt1=kurtosis(SegmentoArray,axis=1)

    for i in range(0,8):
        index=np.where(Kurt1[:,i]>umbralmaximo)
        Unos[index]=0
        index=np.where(Kurt1[:,i]<umbralminimo)
        Unos[index]=0

    Rechazo=Unos.transpose()[0]
    Data=SegmentoArray[Rechazo==1,:,:]
    size=Data.shape
    Signal=np.zeros((size[0]*size[1],size[2]),dtype=np.int)
    for i in range(0,size[2]):
        Signal[:,i]=Data[:,:,i].ravel()

    return Signal

# SenalSeg1_Kurtosis=Kurtosis(3,-3,S1_array)
# SenalSeg2_Kurtosis=Kurtosis(3,-3,S2_array)
# SenalSeg3_Kurtosis=Kurtosis(3,-3,S3_array)
# SenalSeg4_Kurtosis=Kurtosis(3,-3,S4_array)
# plt.plot(SenalSeg4 Kurtosis+DC)
```

Figura 11. *Función tercer método-Rechazo por improbabilidad.*

Como retorno, se obtiene la señal sin los datos fuera del rango establecido, y se obtienen los siguientes resultados (figura 12).
Los ocho canales de la señal 3, se visualizan de tal manera por la amplitud que representa.

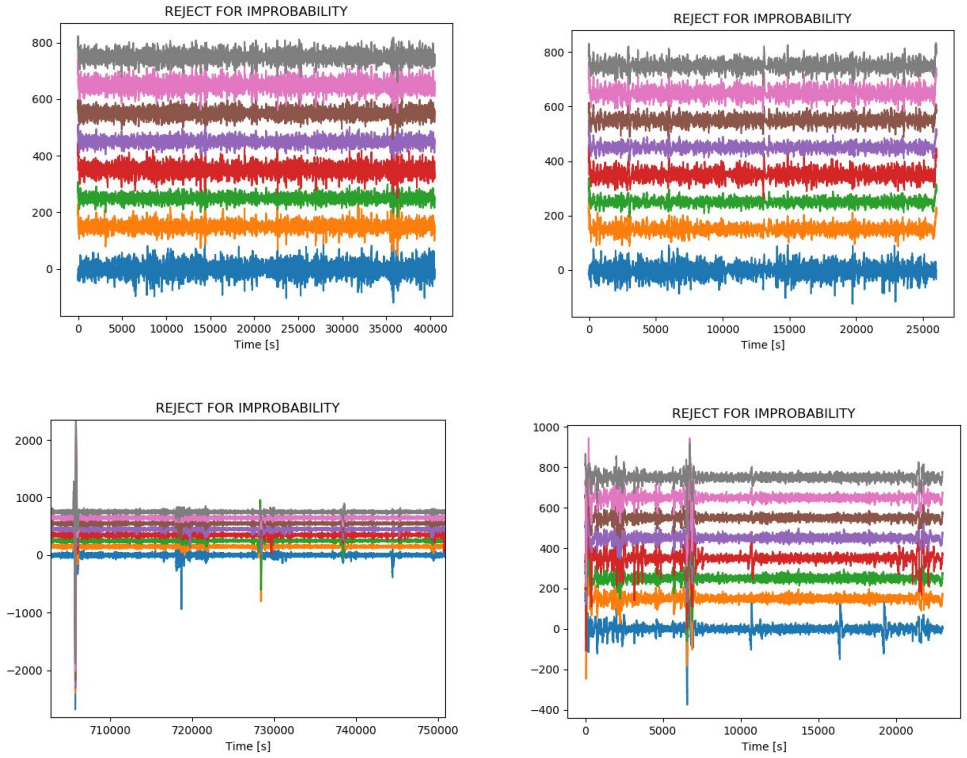

Figura 12.*.Resultado de cada una de las señales aplicadas al tercer método, en orden de izquierda a derecha.*

- Para el cuarto método, se utilizó el patrón espectral, en el cual, para la función *'PATRON_ESPECTRAL'* se necesitan los siguientes parámetros:
    - SegmentoArray→ Array del segmento a utilizar.
    - frec→ Frecuencia de muestreo.
    - Umbralmax→ Umbral máximo.
    - Umbralmin→ Umbral mínimo.

Este método se basa en el periodograma de Welch, el cual lo calcula para cada segmento dado con su respectiva media. A cada periodograma obtenido, se le resta la media calculada, y a partir de los umbrales dados, si esta resta se encuentra muy alejado de los valores, los rechaza y se queda con los datos en el rango establecido.

```python
#FOURTH METHOD: REJECT FOR SPECTRAL PATTERN
def PATRON_ESPECTRAL(SegmentoArray,frec,umbralmax,umbralmin):
    Dim=SegmentoArray.shape
    Unos=np.ones((Dim[0],1),dtype=np.int)
    Pmax=np.zeros((Dim[0],Dim[2]),dtype=np.float32)
    Pmin=np.zeros((Dim[0],Dim[2]),dtype=np.float32)


    for j in range (0,Dim[2]):
        for i in range(0,Dim[0]):
            F,Pxx= signal.welch(SegmentoArray[i,:,j],frec,"hanning",Dim[1])
            media=np.mean(Pxx)
            Pot=Pxx-media
            Pmax[i,j]=Pot.max()
            Pmin[i,j]=Pot.min()

    for k in range(0,Dim[2]):
        index=np.where(Pmax[:,k]>umbralmax)
        Unos[index]=0
        index=np.where(Pmin[:,k]<umbralmin)
        Unos[index]=0


    Patron=Unos.transpose()[0]
    Data=SegmentoArray[Patron==1,:,:]
    size=Data.shape
```

Figura 13. *Función cuarto método-Rechazo por patrón espectral.*

```python
    Signal=np.zeros((size[0]*size[1],size[2]),dtype=np.int)
    for i in range(0,size[2]):
        Signal[:,i]=Data[:,:,i].ravel()
    return Signal

#SenalSeg1_Patron= PATRON_ESPECTRAL(S1_array, 250,200,-5)
#SenalSeg2_Patron= PATRON_ESPECTRAL(S2_array, 250,200,-5)
#SenalSeg3_Patron= PATRON_ESPECTRAL(S3_array, 250,200,-5)
#SenalSeg4_Patron= PATRON_ESPECTRAL(S4_array, 250,200,-5)
#plt.plot(SenalSeg1_Patron+DC)
#plt.xlabel('Time [s]')
#plt.title('REJECT FOR SPECTRAL PATTERN')
#plt.show()
```

Figura 14. *Implementación de la función de la figura 10 y su respectivo plot.*

Las señales retornadas sin los valores fuera del umbral, se evidencian en la figura 15.

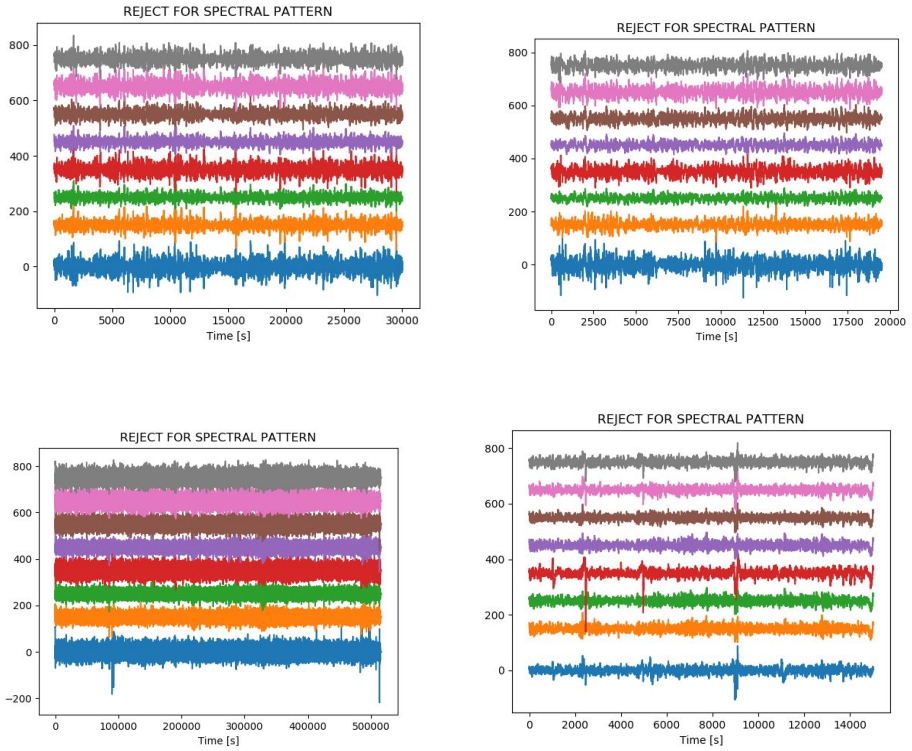

Figura 15.*Resultado de cada una de las señales aplicadas al cuarto método, en orden de izquierda a derecha.*

B) ESTRATEGIAS DE PREPROCESAMIENTO:

En este punto, se propone tres diferentes estrategias de preprocesamiento con diferentes órdenes de de aplicación. Con los mismos umbrales pero diferentes métodos utilizados.

- **Estrategia 1:** aplicación de todos los métodos de rechazo. La señal resultante se muestra a continuación:

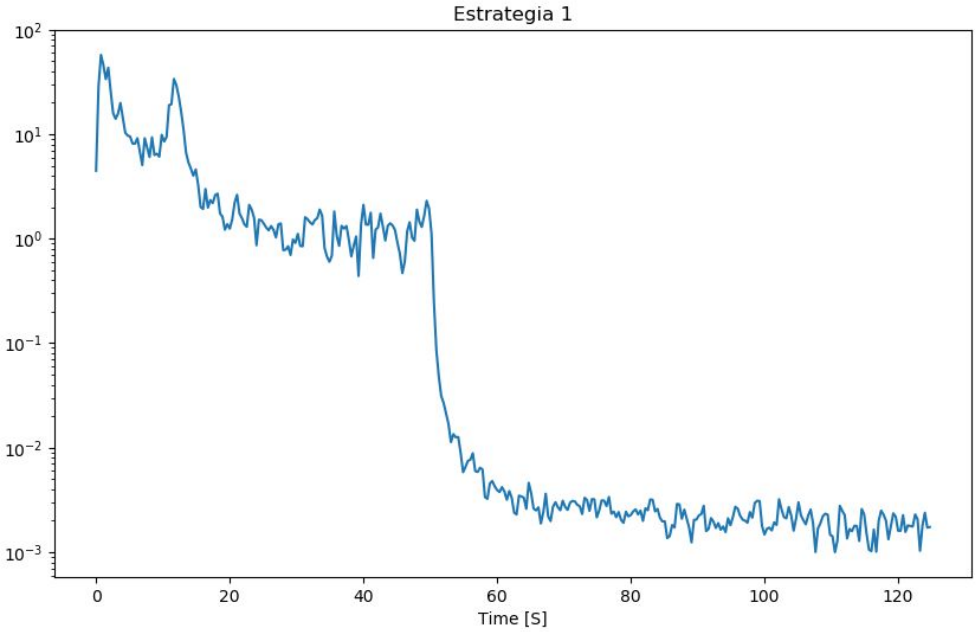

Figura 16. *Estrategia 1→ 4 métodos implementados.*

El código implementado para esta estrategia se muestra en la figura 17.

```
#ESTRATEGIA 1
S1_array,time1=segmentacion(filtro1,tiempo_f1,250,2)
SenalSeg1_Valores=Rechazo(75,-75,S1_array)
Valores_Extremos1,time_1=segmentacion(SenalSeg1_Valores, tiempo_f1, 250, 2)
TendenciaLineal=TENDENCIA_LINEAL(Valores_Extremos1, 4, -4, time_1)
TendenciaLineal1,time_2=segmentacion(TendenciaLineal, tiempo_f1, 250, 2)
Improbabilidad=Kurtosis(3,-3,TendenciaLineal1)
Improbabilidad1,time_3=segmentacion(Improbabilidad, time_2, 250, 2)
PatronEspectral=PATRON_ESPECTRAL(Improbabilidad1, 250, 200,-5)


Frec,Pxx= signal.welch(PatronEspectral[:,0],250, "hanning",len(PatronEspectral[:,0])/8)
plt.figure(figsize=(10,6))
plt.semilogy(Frec,Pxx)
plt.title('Estrategia 1')
plt.xlabel('Time [S]')
plt.show()
```

Figura 17. *Código de Estrategia 1→ 4 métodos implementados.*

- **Estrategia 2:** sin aplicación de patrón espectral. El código y la señal resultante se evidencian en las figuras 18 y 19.

```
#ESTRATEGIA 2
S1_array,time1=segmentacion(filtro1,tiempo_f1,250,2)
SenalSeg1_Valores=Rechazo(75,-75,S1_array)
Valores_Extremos1,time_1=segmentacion(SenalSeg1_Valores, tiempo_f1, 250, 2)
TendenciaLineal=TENDENCIA_LINEAL(Valores_Extremos1, 4, -4, time_1)
TendenciaLineal1,time_2=segmentacion(TendenciaLineal, tiempo_f1, 250, 2)
Improbabilidad=Kurtosis(3,-3,TendenciaLineal1)


Frec,Pxx= signal.welch(Improbabilidad[:,0],250, "hanning",len(Improbabilidad[:,0])/8)
plt.figure(figsize=(10,6))
plt.semilogy(Frec,Pxx)
plt.title('Estrategia 2')
plt.xlabel('Time [S]')
plt.show()
```

Figura 18. *Código de estrategia 2→ 3 métodos implementados, sin patrón espectral.*

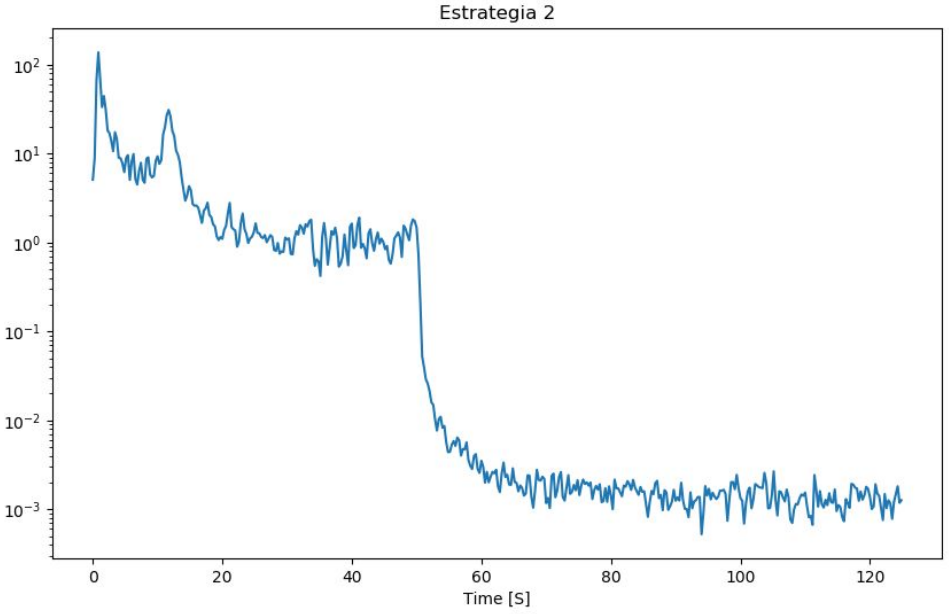

Figura 19. *Estrategia 2→ 3 métodos implementados, sin patrón espectral.*

- **Estrategia 3:** sin aplicación de patrón espectral ni improbabilidad. Los resultados de esta estrategia se evidencian a continuación.

```
#ESTRATEGIA 3
S1_array,time1=segmentacion(filtro1,tiempo_f1,250,2)
SenalSeg1_Valores=Rechazo(75,-75,S1_array)
Valores_Extremos1,time_1=segmentacion(SenalSeg1_Valores, tiempo_f1, 250, 2)
TendenciaLineal=TENDENCIA_LINEAL(Valores_Extremos1, 4, -4, time_1)


Frec,Pxx= signal.welch(TendenciaLineal[:,0],250, "hanning",len(TendenciaLineal[:,0])/8)
plt.figure(figsize=(10,6))
plt.semilogy(Frec,Pxx)
plt.title('Estrategia 3')
plt.xlabel('Time [S]')
plt.show()
```

Figura 20. *Código Estrategia 3→ 2 métodos implementados, sin patrón espectral ni improbabilidad.*

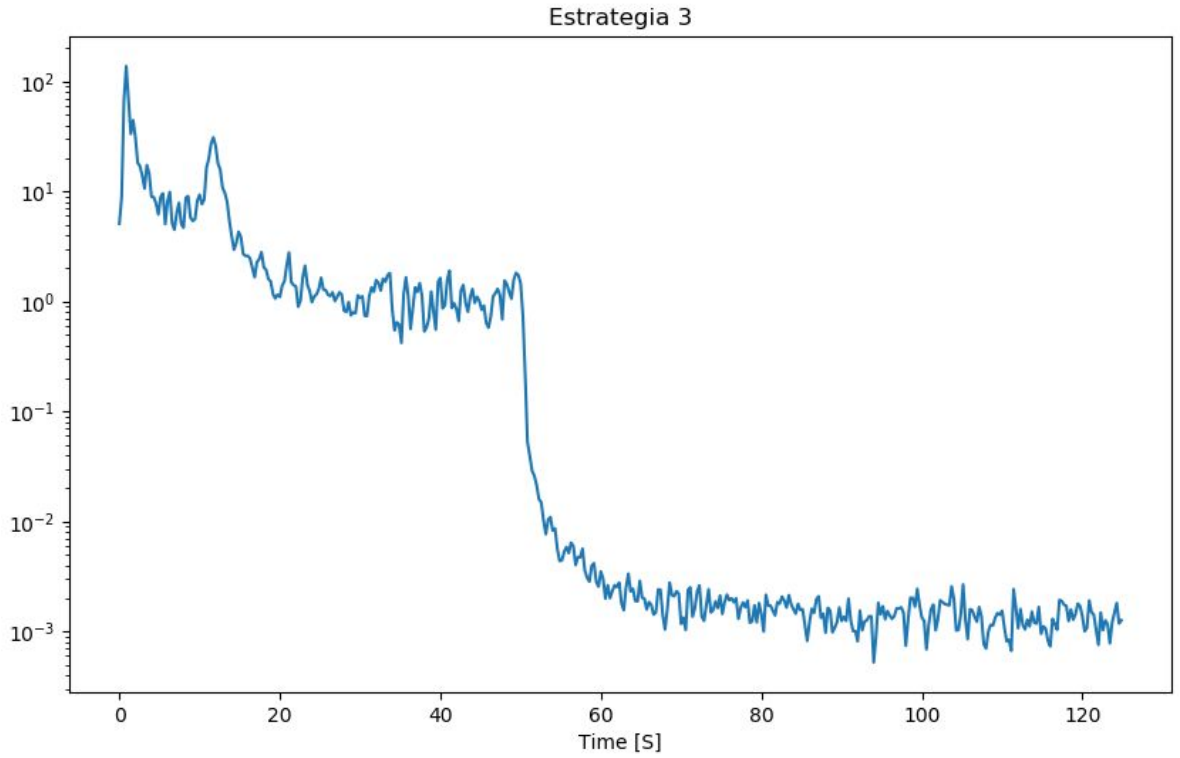

Figura 21. *Estrategia 3→ 2 métodos implementados, sin patrón espectral ni improbabilidad.*

Como se evidenció en los tres últimos gráficos, el cambio no fue muy notorio en las señales resultantes con los diferentes preprocesamientos implementados. Sin embargo, al aplicar únicamente el método de valores extremos, la señal queda con mucho ruido, como se logra evidenciar en la figura 22.

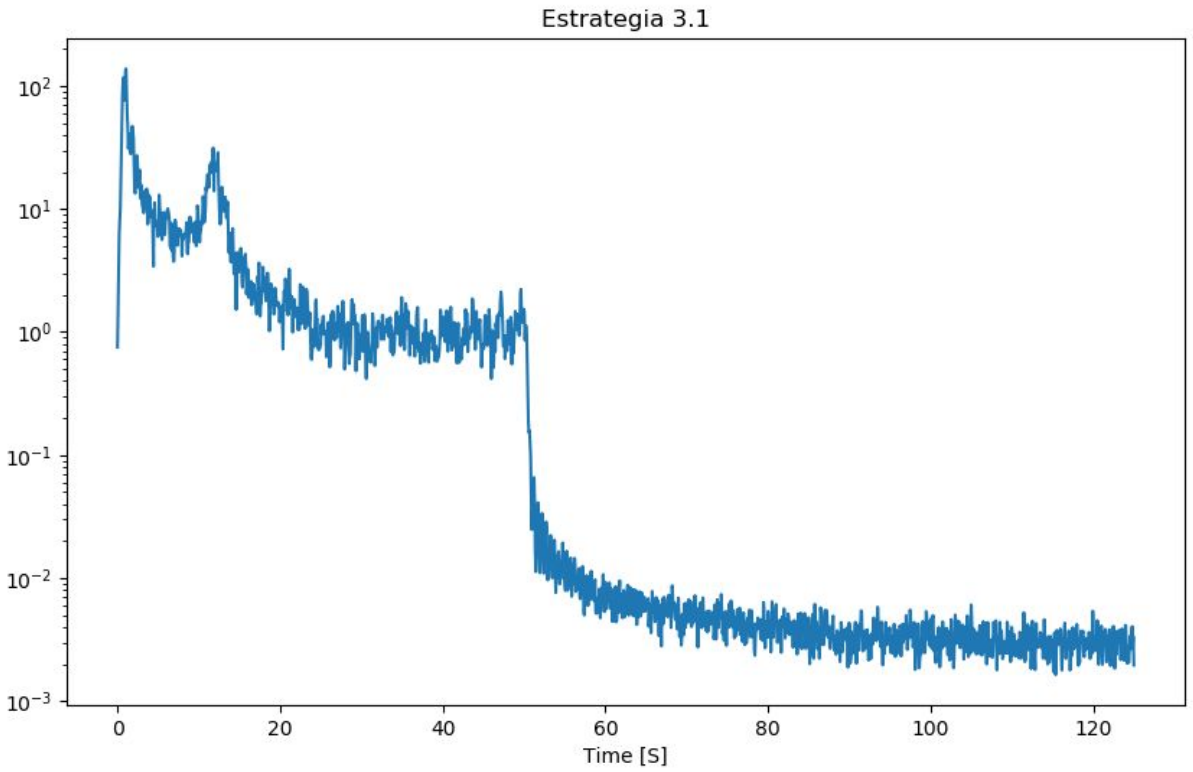

Figura 22. *Señal con el método de valores extremos.*

Finalmente, con la aplicación de valores extremos y tendencia lineal, no se obtuvieron cambios drásticos en la señal de salida.

Anotaciones importantes:

1) El diferente orden de aplicación de los métodos con el mismo umbral, no afecta notoriamente las señales de salida.
2) Si se aplican los métodos en el mismo orden pero con diferentes umbrales, se pierde mucha información de la señal en estudio.