UNIVERSIDAD DE GUANAJUATO

Verano2024(T3_4)_M?rquezSulca

July 29, 2024

1 £Una dimensión para comprender el Universo?

---

https://colab.research.google.com/drive/1rPpTs63aN9C6dX9cTdQafSfxcMc3wfpU?usp=sharing#scrollTo=

Revisaremos el análisis del bosque de Lyman alfa sobre una dimensión para ver qué nos puede enseñar en cosmología. El proyecto está basado en el realizado por la colaboración internacional Dark Energy Spectroscopic Instrument (DESI, https://desi.lbl.gov/) en su análisis del espectro de potencias del bosque de Lyman alfa sobre los datos EDR (Mon.Not.Roy.Astron.Soc. 526 (2023) 4).

1.1 Importamos las librerías necesarias para ejecutar nuestro código

[ ]:
```
import numpy as np
from PIL import Image
from skimage.io import imread
import matplotlib.pyplot as plt
from astropy.table import Table
from skimage.color import rgb2gray
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA
from scipy.stats import chi2_contingency

import os
from astropy.io import fits
from astropy.table import Table, Column
```

1.2 Carga de datos

Conectamos a Google Colab para utilizar los archivos necesarios.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

Aquí cargamos los datos, tenemos tres opciones en este caso, utilizar espectros simulados o utilizar espectros reales obtenidos de datos EDR, como se ha mencionado al inicio del cuadernillo, para distintos redshifts.

[ ]:
```
#Cargamos los datos

#Podemos cargar las simulaciones
path = '/content/drive/My Drive/Verano2024/lya100simqso.npy'
datos = np.load(path, allow_pickle=True)


#O los datos reales donde

  # Carpeta con redshifts de 2.2 a 2.4
path1 = '/content/drive/My Drive/Verano2024/2.2-2.4'

  # Carpeta con redshifts de 2.6 a 2.8
#path1 = '/content/drive/My Drive/Verano2024/2.6-2.8'
```

2 Simulación

Sólo corremos estas celdas si estamos usando los datos "lua100simqso.npy", o algún archivo que tenga el mismo formato, de no ser así, empezaremos a correr el código a partir del sigueinte título llamado "**Datos EDR**".

Vamos a ver la forma que tienen estos datos, aquí vemos que nuestro arreglo contiene 4 elementos, pero £Qué elementos?

[ ]:
```
datos.shape
```

[ ]:
```
(4,)
```

Vemos que tiene dos arreglos y dos tablas, en este caso nos interesan sólo los arreglos, a continuación veremos que es cada arreglo, mientras las tablas son sobre algunas otras característcas.

[ ]:
```
datos
```

[ ]:
```
array([array([[2.55549551e-10, 2.58901058e-10, 2.60270680e-10, ...,
               2.15351438e-02, 2.15326579e-02, 2.15301723e-02],
              [3.38791284e-07, 3.36706961e-07, 3.19156247e-07, ...,
               2.88823939e-02, 2.88785794e-02, 2.88747653e-02],
              [2.18650699e-03, 2.17598813e-03, 1.95036582e-03, ...,
               1.17539394e-02, 1.17523332e-02, 1.17507273e-02],
              ...,
              [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
               3.77623236e-03, 3.77587904e-03, 3.77552575e-03],
              [1.05741519e-10, 1.05750887e-10, 1.04808655e-10, ...,
               1.55758097e-02, 1.55741006e-02, 1.55723916e-02],
              [2.16776316e-20, 2.26113004e-20, 2.29430646e-20, ...,
               7.98594353e-01, 7.98492106e-01, 7.98389871e-01]])   ,
       array([  450.        ,   450.05625352,   450.11251406, ...,
```

```
            59986.3660422 , 59993.86480662, 60001.36450844])    ,
      <Table length=100>
      TARGETID OBJTYPE SUBTYPE TEMPLATEID ...    FLUX_Z     FLUX_W1     FLUX_W2
                                          ... nanomaggies nanomaggies
nanomaggies
       int64    str10   str10    int64    ...   float32     float32     float32
      -------- ------- ------- ---------- ... ----------- -----------
-----------
             0     QSO     LYA          0 ...    7.881542    9.522354
13.171676
             1     QSO     LYA          1 ...   7.0401125   11.520282
18.309992
             2     QSO     LYA          2 ...   5.0262227    9.369226
12.295799
             3     QSO     LYA          3 ...   6.9079533    9.059001
14.603897
             4     QSO     LYA          4 ...   3.9291627   11.385156
18.93713
             5     QSO     LYA          5 ...   27.419893   32.933903
45.21468
             6     QSO     LYA          6 ...   1.9643531    3.053256
4.997409
             7     QSO     LYA          7 ...   6.5826063   13.819599
17.866152
             8     QSO     LYA          8 ...   0.9898675   1.1609733
1.6924179
           ...     ...     ...        ... ...         ...         ...
...
            90     QSO     LYA         90 ...   1.4485104   2.2201788
3.117715
            91     QSO     LYA         91 ...   2.0194297   3.2754035
4.9964523
            92     QSO     LYA         92 ...  0.87984324   1.5301273
2.1927216
            93     QSO     LYA         93 ...   1.4994081   2.7117429
4.049831
            94     QSO     LYA         94 ...   1.3426052   3.6743593
5.664247
            95     QSO     LYA         95 ...   0.8495589   1.2201995
1.7599388
            96     QSO     LYA         96 ...    8.259461    13.02541
19.347546
            97     QSO     LYA         97 ...   1.0799193   1.4505165
2.0360146
            98     QSO     LYA         98 ...   4.7400827    8.314163
11.859535
            99     QSO     LYA         99 ...     138.924   272.25375
```

```
451.20517,
       <Table length=100>
       TARGETID MABS_1450            SLOPES          ... BAL_TEMPLATEID  DLA
                   mag                               ...
        int64    float32           float32[5]         ...     int16       bool
       -------- ---------- ------------------------- ... -------------- -----
              0 -24.917624  -1.3094226 .. -1.0764582 ...             -1 False
              1 -24.785715 -2.0494072 .. -0.94336027 ...             -1 False
              2  -24.24264  -1.9072899 .. -0.9067168 ...             -1 False
              3 -24.849638  -1.7697752 .. -0.6280984 ...             -1 False
              4 -23.854242  -1.8447511 .. -1.8040473 ...             -1 False
              5 -26.245047 -1.5411478 .. -0.57492065 ...             -1 False
              6 -23.379602  -1.8399305 .. -0.8939744 ...             -1 False
              7 -24.440105  -1.6965616 .. -0.6010259 ...             -1 False
              8 -22.741686   -1.378357 .. -1.1510594 ...             -1 False
            ...        ...                       ... ...            ...   ...
             90 -22.945038  -1.1108463 .. -1.3475752 ...             -1 False
             91  -23.43271 -1.4142272 .. -0.55008996 ...             -1 False
             92 -22.403606  -0.7985318 .. -0.7319957 ...             -1 False
             93 -22.999645  -1.8370581 .. -1.1905321 ...             -1 False
             94 -22.665716   -1.7573292 .. -0.877459 ...             -1 False
             95 -22.414503   -1.248135 .. -1.0109419 ...             -1 False
             96  -24.89478 -1.9792758 .. -0.72329646 ...             -1 False
             97 -22.742981  -1.4693078 .. -1.2514536 ...             -1 False
             98 -24.281157     -1.6692439 .. -1.1221 ...             -1 False
             99 -27.931751   -1.354525 .. -0.9756564 ...             -1 False],
      dtype=object)
```

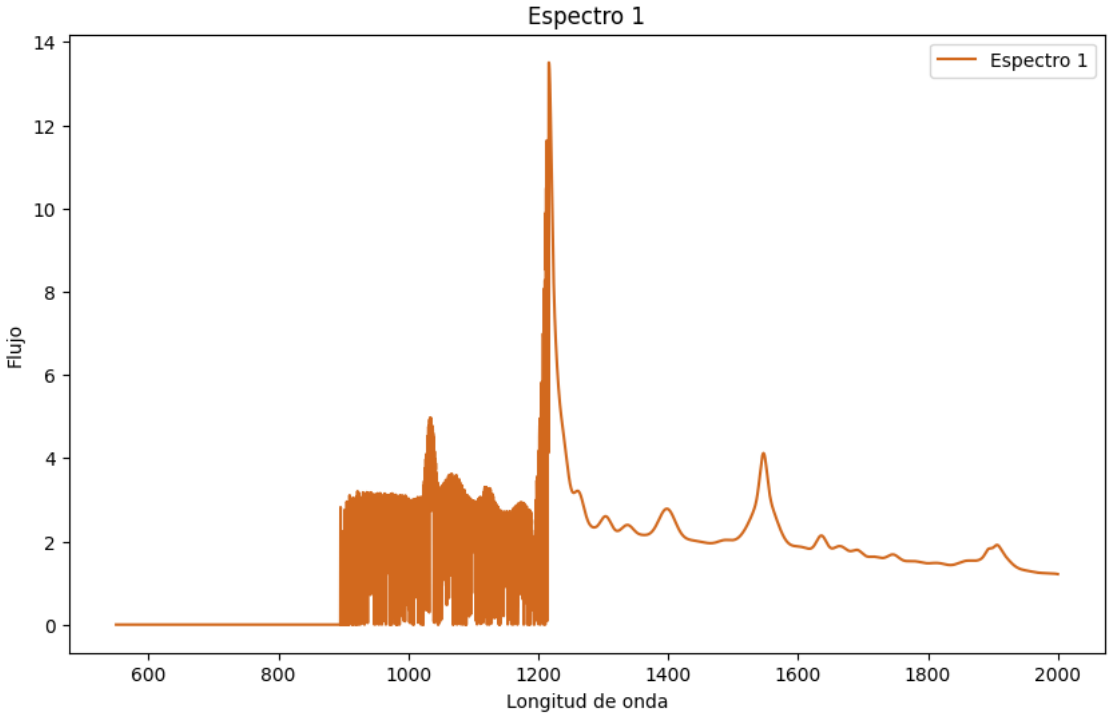
Entonces en "datos[0]" tenemos un arreglo de arreglos de la forma (100,39144), el cien es porque son 100 cuasares, aquí se contienen los flujos de cada espectro. \ Mientras en "datos[1]" tenemos un sólo arreglo también con 39,144 elementos, aquí se contiene la longitud de onda para todos los espectros, es decir, todos los epsectros abarcan la misma longitud de onda, sólo cambia el flujo.

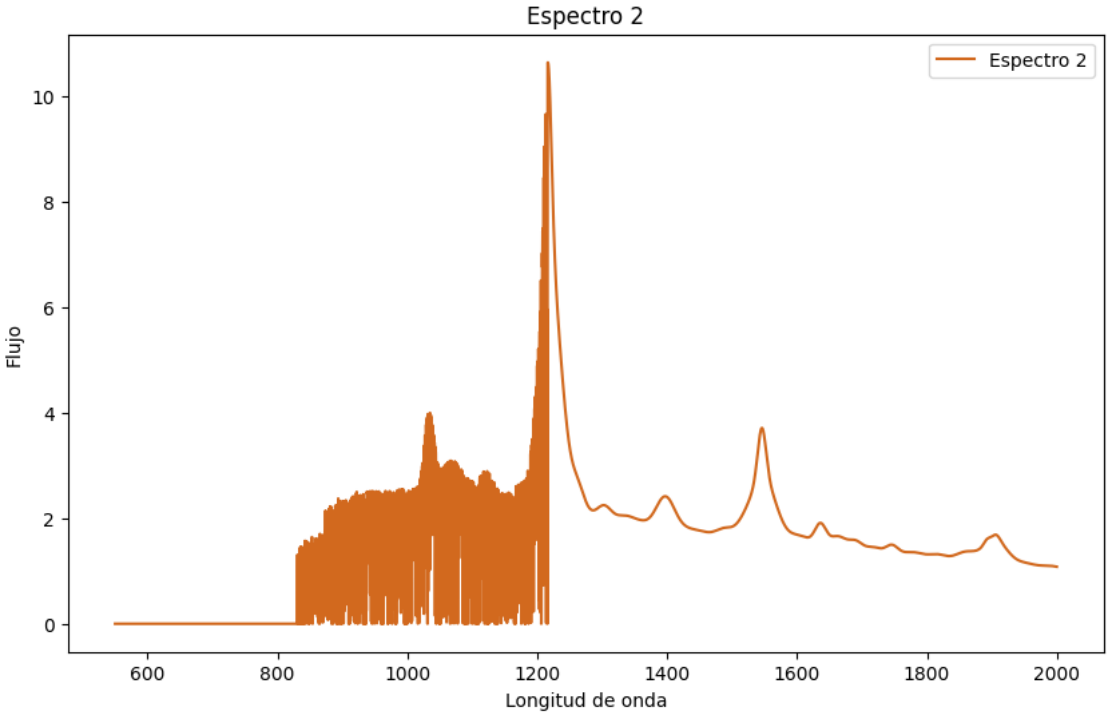
[ ]:
```
#Definimos qué es cada arreglo
longitudes_onda_sim = datos[1]
flujos_sin_normalizar_sim = datos[0]
```

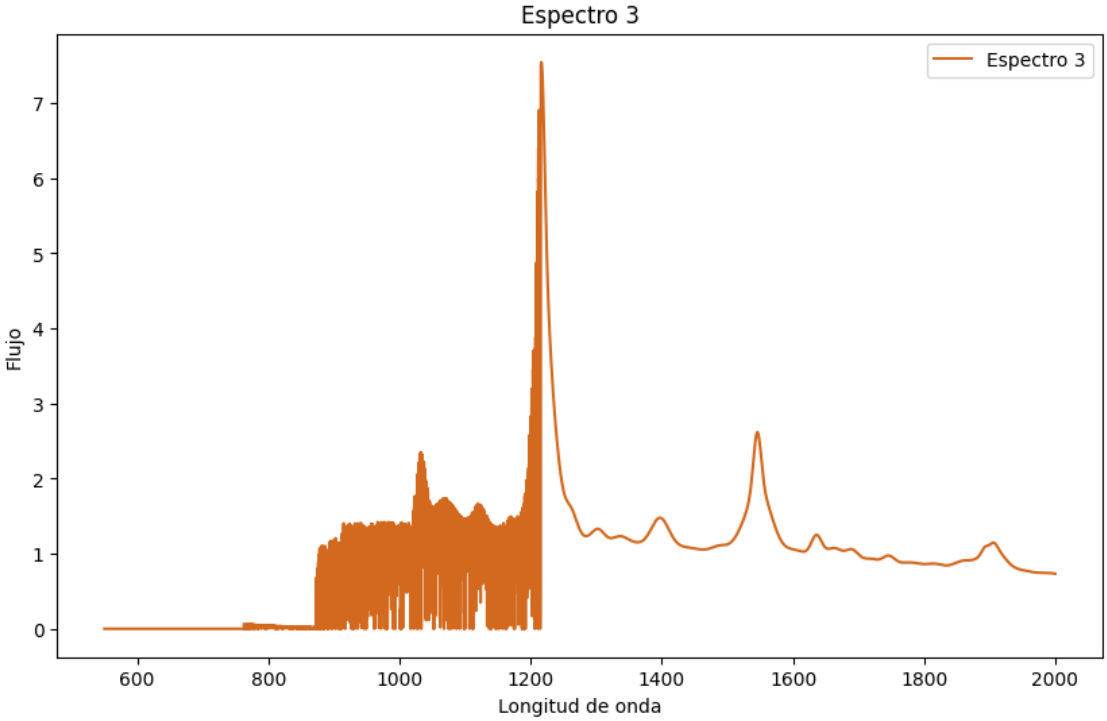
El rango de longitudes de onda es muy amplio (de 0 a 6000 Armstrong), esto no nos permite ver adecuadamente la región de Lyman Alfa, así que aplicaremos algo llamado "máscara" para sólo observar la región de 550 a 2000 Armstrongs, esto nos da una mejor visualización de nuestro espectro.

[ ]:
```
# Hacemos la máscara para los datos que nos resultan más útiles
mask1_sim = (longitudes_onda_sim >= 550) & (longitudes_onda_sim <= 2000)

# Aplicamos la máscara
longitudes_onda_mascara_sim = longitudes_onda_sim[mask1_sim]
flujos_mascara_sim = flujos_sin_normalizar_sim[:, mask1_sim]
```

```
# Graficamos los primeros 5
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara_sim, flujos_mascara_sim[i],␣
 ↪label=f'Espectro {i+1}', c='chocolate')
    plt.xlabel('Longitud de onda')
    plt.ylabel('Flujo')
    plt.title(f'Espectro {i+1}')
    plt.legend()
    plt.show()
```




Espectro 2
Espectro 2
Flujo
Longitud de onda
0
2
4
6
8
10
600
800
1000
1200
1400
1600
1800
2000


Espectro 3
Espectro 3
Flujo
Longitud de onda
0
1
2
3
4
5
6
7
600
800
1000
1200
1400
1600
1800
2000

Aquí podemos ver lo que es llamado el pico de Lyman Alfa, este es el pico más alto que vemos aproximadamente a la mitad de la gráfica y el pico de Lyman Beta, este está ubicado un poco a la izquierda y es considerablemente más pequeño que el de Lyman Alfa, pero sigue siendo distingible, todo lo que se encuentra en medio de ambos picos es llamado el **Bosque de Lyman Alfa**.

2.1 Pre-procesamiento

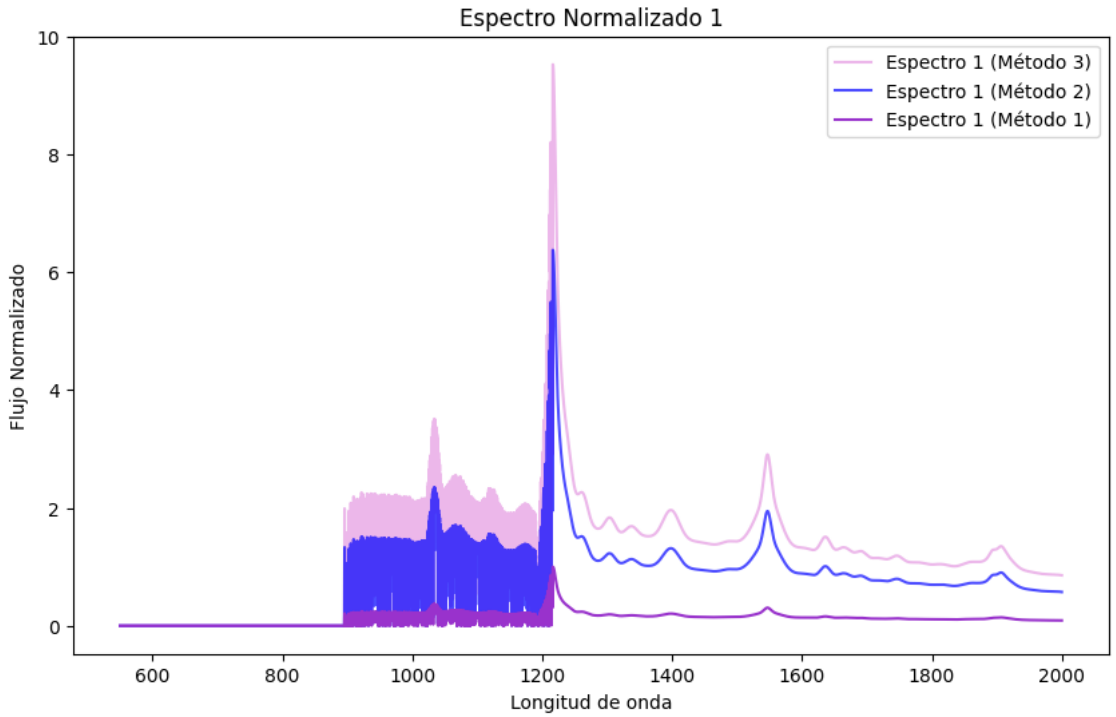
El pre-procesamiento son técnicas que aplicamos para suavizar o disminuir el ruido de nuestros datos antes de realizar cualquier análisis. \ Vamos a probar tres métodos diferentes para ello.

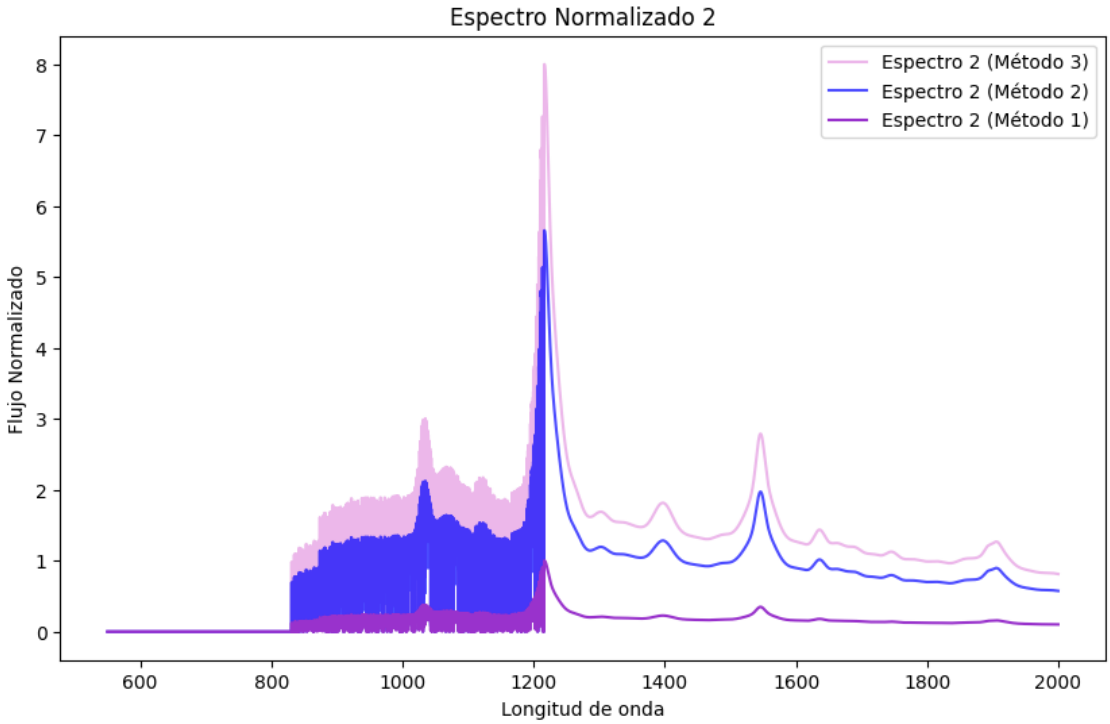
[ ]:
```
#Definimos las fucniones

# Usando una altura media entre el maximo y mínimo de cada espectro
def normalizar_altura_media(flujo):
    flujo_min = np.min(flujo, axis=1)[:, np.newaxis]
    flujo_max = np.max(flujo, axis=1)[:, np.newaxis]
    return (flujo - flujo_min) / (flujo_max - flujo_min) # NOTA: más bien era␣
↪máximo y el mínimo y quitarle esa distancia al cuásar, algo como restar␣
↪(maximo-min)/2


# Promedio de alturas para un pequeño rango en lambda
def normalizar_promedio_altura(flujo, wavelength, rango_min, rango_max):
    rango_mascara = (wavelength >= rango_min) & (wavelength <= rango_max)
    prom_flujo = np.mean(flujo[:, rango_mascara], axis=1)[:, np.newaxis]
    return flujo / prom_flujo


# Promedio de todas
def normalizar_promedio_todas(flujo):
    mean_flujo = np.mean(flujo, axis=1)[:, np.newaxis]
    return flujo / mean_flujo
```

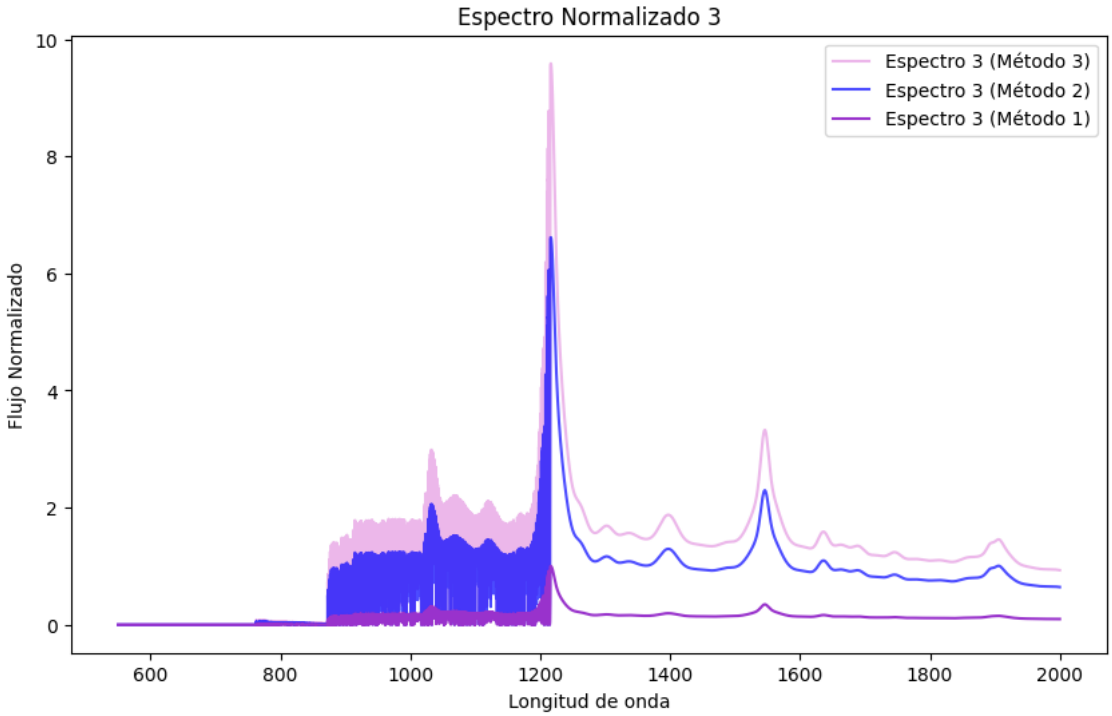
[ ]:
```
#Mandamos a llamar a las funciones que creamos
flujos_normalizados1 = normalizar_altura_media(flujos_mascara_sim)
flujos_normalizados2 = normalizar_promedio_altura(flujos_mascara_sim,␣
↪longitudes_onda_mascara_sim, 1400, 1500) #De preferencia más a la derecha
flujos_normalizados3 = normalizar_promedio_todas(flujos_mascara_sim)
```

[ ]:
```
#Graficamos los primeros 5 espectros juntos (el original y las tres␣
↪normalizaciones)
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara_sim, flujos_normalizados3[i],␣
↪label=f'Espectro {i+1} (Método 3)', c='orchid', alpha=0.5)
    plt.plot(longitudes_onda_mascara_sim, flujos_normalizados2[i],␣
↪label=f'Espectro {i+1} (Método 2)', c='blue', alpha = 0.7)
```

```
    plt.plot(longitudes_onda_mascara_sim, flujos_normalizados1[i],␣
↪label=f'Espectro {i+1} (Método 1)', c='darkorchid')
    plt.xlabel('Longitud de onda')
    plt.ylabel('Flujo Normalizado')
    plt.title(f'Espectro Normalizado {i+1}')
    plt.legend()
    plt.show()
```




Espectro Normalizado 2
Espectro 2 (Método 3)
Espectro 2 (Método 2)
Espectro 2 (Método 1)
Flujo Normalizado
Longitud de onda
0
1
2
3
4
5
6
7
8
600
800
1000
1200
1400
1600
1800
2000


Espectro Normalizado 3
Espectro 3 (Método 3)
Espectro 3 (Método 2)
Espectro 3 (Método 1)
Flujo Normalizado
Longitud de onda
0
2
4
6
8
10
600
800
1000
1200
1400
1600
1800
2000

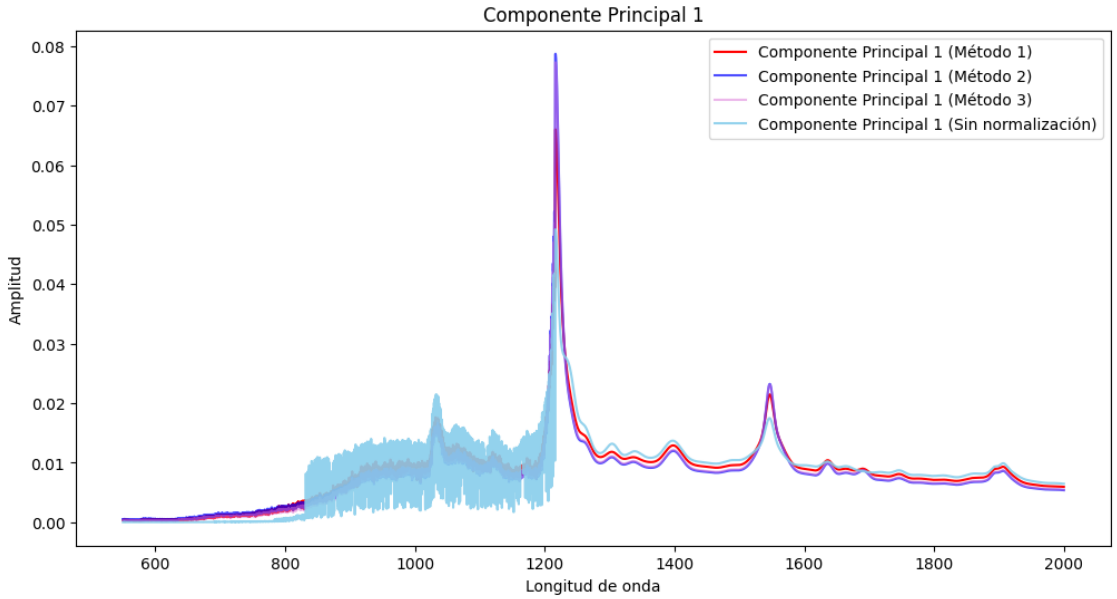
Aquí podemos ver cómo afecta a nuestros datos hacer pre-procesamiento y cómo dependiendo del método tenemos más o menos ruido. Esto nos ayuda a realizar un mejor análisis posterior. \
Ahora vamos a utilizar un método llamada empca: https://github.com/sbailey/empca. Este es un método de pca, £Qué es el pca? Bueno, su principal objetivo es reducir la dimensionalidad de un conjunto de datos mientras se conserva la mayor cantidad posible de varianza, es comúnmente utilizado en el campo de Machine Learning y ciencia de datos.

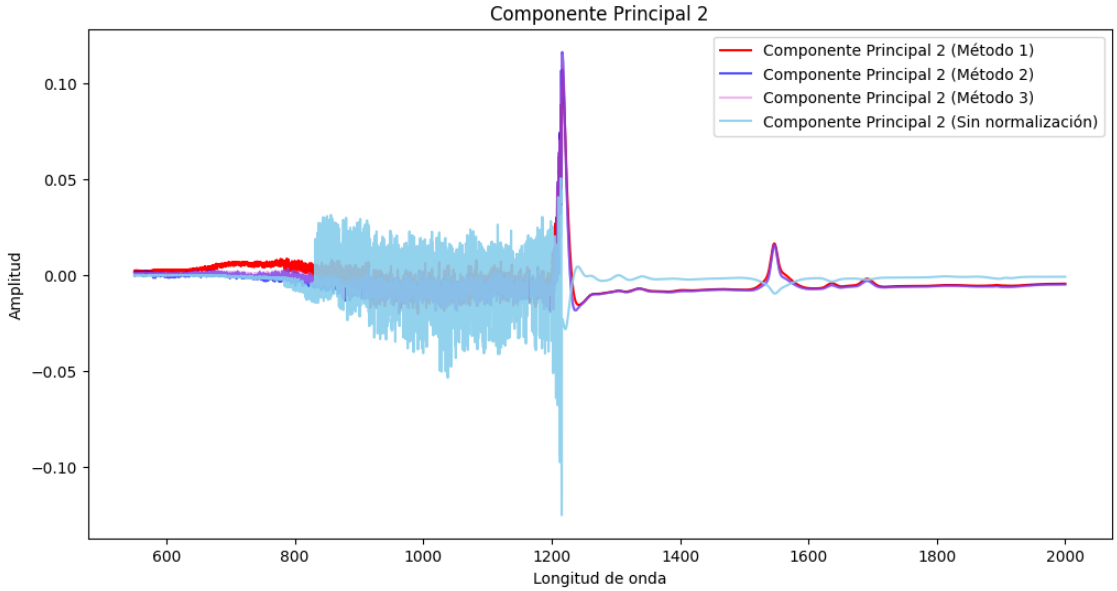
[ ]:
```
#Llamamos al código que importamos en collab
import empca
from empca import empca

#Definimos el número de eigenvectores que queremos y las iteraciones
nvec = 10
niter = 10

weights = np.ones_like(flujos_mascara_sim)

# Aplicar EMPCA para cada método de normalización y sin normalizar
modelo = empca(flujos_mascara_sim, weights, niter=niter, nvec=nvec)
modelo1 = empca(flujos_normalizados1, weights, niter=niter, nvec=nvec)
modelo2 = empca(flujos_normalizados2, weights, niter=niter, nvec=nvec)
modelo3 = empca(flujos_normalizados3, weights, niter=niter, nvec=nvec)
```

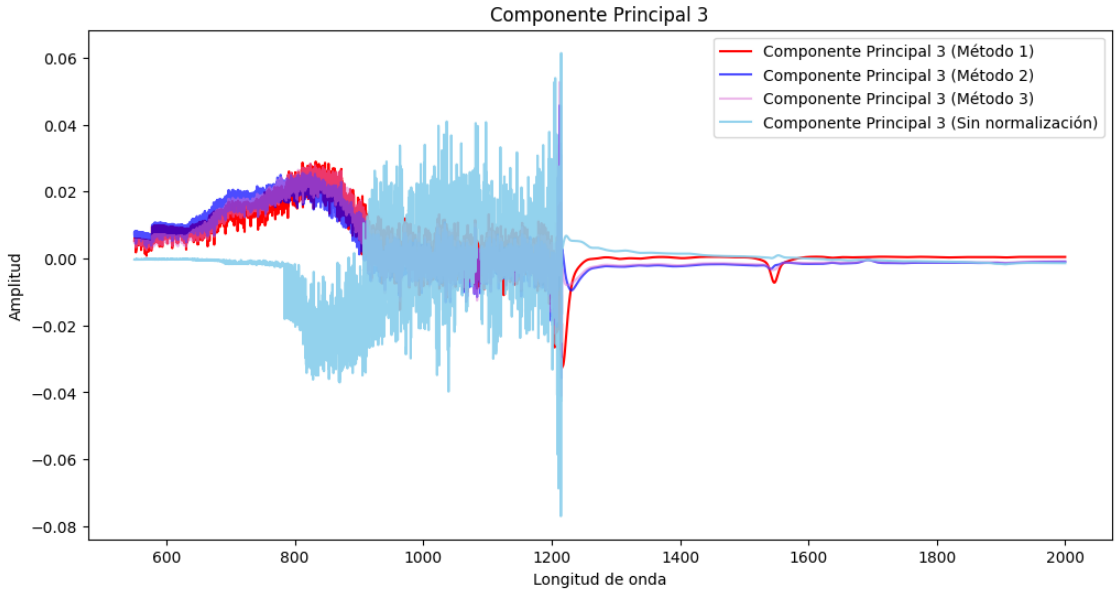
```
       iter        R2             rchi2
EMPCA  1/10       0.00524259     16.07258320
EMPCA  2/10       0.98263146      0.25414696
EMPCA  3/10       0.98999330      0.14607787
EMPCA  4/10       0.99073185      0.13526079
EMPCA  5/10       0.99097450      0.13172067
EMPCA  6/10       0.99111520      0.12966706
EMPCA  7/10       0.99121505      0.12820911
EMPCA  8/10       0.99129289      0.12707284
EMPCA  9/10       0.99135400      0.12618138
EMPCA 10/10       0.99139961      0.12551629
R2: 0.9914853976838974
       iter        R2             rchi2
EMPCA  1/10       0.01637930      0.02579071
EMPCA  2/10       0.84269716      0.00227931
EMPCA  3/10       0.87874709      0.00175536
EMPCA  4/10       0.88898105      0.00160709
EMPCA  5/10       0.89135222      0.00157275
EMPCA  6/10       0.89246964      0.00155657
EMPCA  7/10       0.89294912      0.00154963
EMPCA  8/10       0.89317174      0.00154641
EMPCA  9/10       0.89329864      0.00154457
EMPCA 10/10       0.89338597      0.00154330
R2: 0.8935101647995886
       iter        R2             rchi2
```

```
EMPCA  1/10       0.01994742      1.43894691
EMPCA  2/10       0.85806264      0.11494815
EMPCA  3/10       0.88600976      0.09190743
EMPCA  4/10       0.89094859      0.08792536
EMPCA  5/10       0.89279595      0.08643586
EMPCA  6/10       0.89369152      0.08571376
EMPCA  7/10       0.89417048      0.08532756
EMPCA  8/10       0.89444058      0.08510976
EMPCA  9/10       0.89459283      0.08498700
EMPCA 10/10       0.89467845      0.08491796
R2: 0.8948047727705232
       iter        R2             rchi2
EMPCA  1/10       0.01472524      2.47209912
EMPCA  2/10       0.85939178      0.19529715
EMPCA  3/10       0.88241647      0.16277066
EMPCA  4/10       0.88906493      0.15356576
EMPCA  5/10       0.89105468      0.15081132
EMPCA  6/10       0.89208757      0.14938150
EMPCA  7/10       0.89267093      0.14857396
EMPCA  8/10       0.89300600      0.14811014
EMPCA  9/10       0.89320866      0.14782959
EMPCA 10/10       0.89334323      0.14764330
R2: 0.8936439510539373
```

[ ]:
```python
for i in range(3):
    plt.figure(figsize=(12, 6))
    plt.plot(longitudes_onda_mascara_sim, modelo1.eigvec[i], label=f'Componente Principal {i+1} (Método 1)', c='red')
    plt.plot(longitudes_onda_mascara_sim, modelo2.eigvec[i], label=f'Componente Principal {i+1} (Método 2)', c='blue', alpha = 0.7)
    plt.plot(longitudes_onda_mascara_sim, modelo3.eigvec[i], label=f'Componente Principal {i+1} (Método 3)', c='orchid', alpha = 0.5)
    plt.plot(longitudes_onda_mascara_sim, modelo.eigvec[i], label=f'Componente Principal {i+1} (Sin normalización)' , c='skyblue', alpha = 0.9)
    plt.xlabel('Longitud de onda')
    plt.ylabel('Amplitud')
    plt.title(f'Componente Principal {i+1}')
    plt.legend()
    plt.show()
```


Componente Principal 1
Componente Principal 1 (Método 1)
Componente Principal 1 (Método 2)
Componente Principal 1 (Método 3)
Componente Principal 1 (Sin normalización)
Amplitud
Longitud de onda
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
600
800
1000
1200
1400
1600
1800
2000

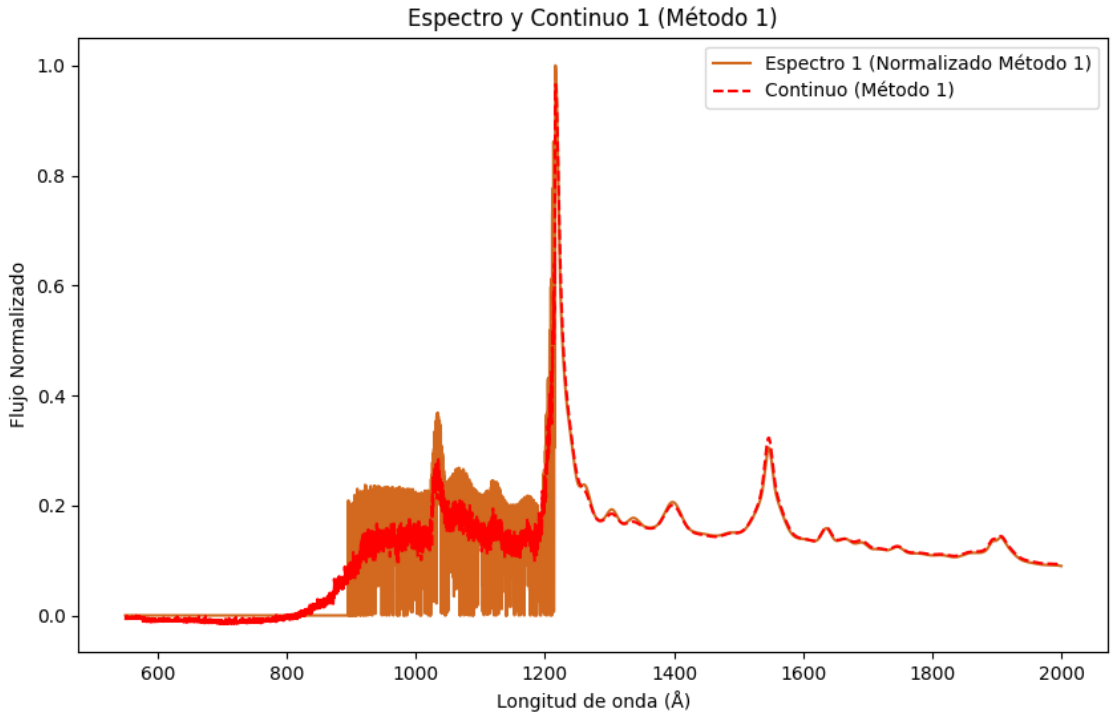

Componente Principal 2
Componente Principal 2 (Método 1)
Componente Principal 2 (Método 2)
Componente Principal 2 (Método 3)
Componente Principal 2 (Sin normalización)
Amplitud
Longitud de onda
−0.10
−0.05
0.00
0.05
0.10
600
800
1000
1200
1400
1600
1800
2000

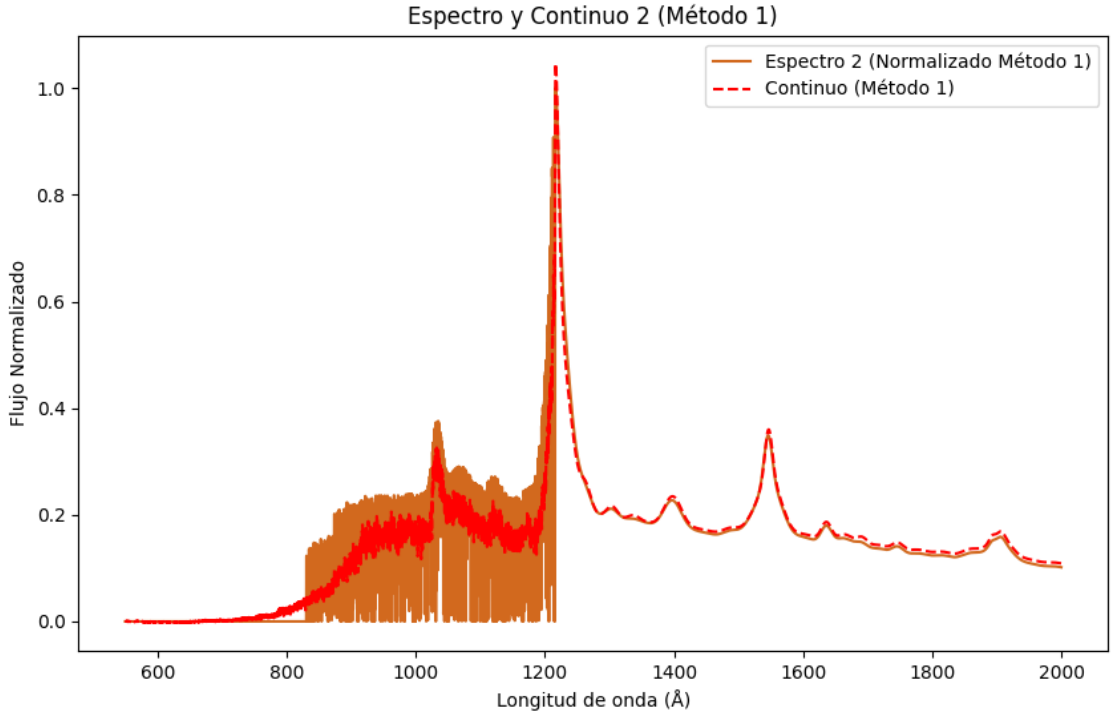


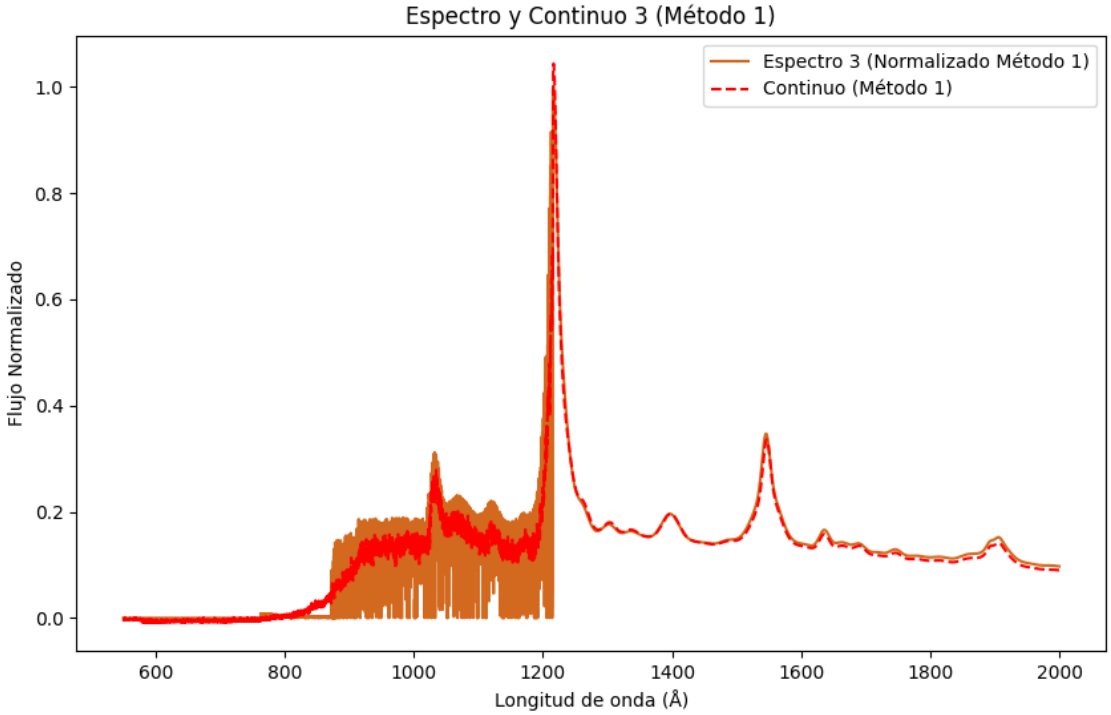
En estas gráficas podemos ver cómo la que llamamos "primera componente" resume bastante bien el comportamiento que estuvimos viendo en nuestras gráficas anteriores.

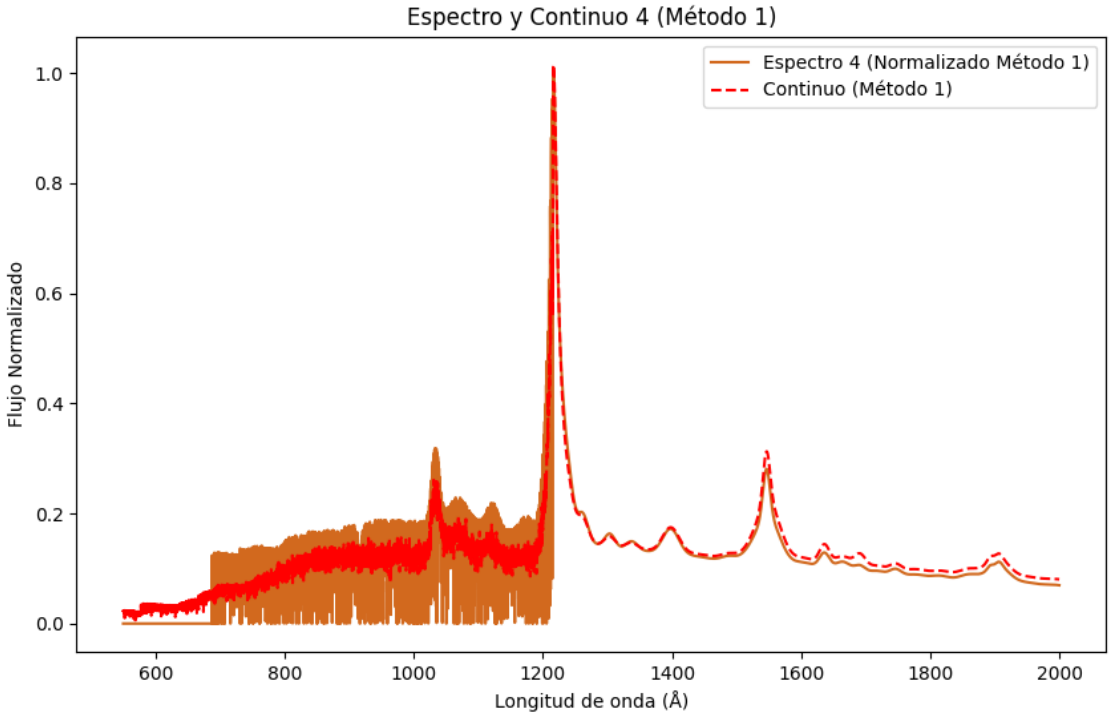
[ ]:
```
# Vamoa a rerear modeloos de continuo usando combinaciones lineales de componentes principales
def recrear_continuo(modelo, num_componentes, flujos):
    eigenvectores = modelo.eigvec[:num_componentes]
    proyecciones = np.dot(flujos, eigenvectores.T)
    continuo = np.dot(proyecciones, eigenvectores)
    return continuo

# Usamos el modeloo normalizado por el método 1 como ejemplo
num_componentes = 3
continuo1 = recrear_continuo(modelo1, num_componentes, flujos_normalizados1)
continuo2 = recrear_continuo(modelo2, num_componentes, flujos_normalizados2)
continuo3 = recrear_continuo(modelo3, num_componentes, flujos_normalizados3)

# Graficar algunos ejemplos de espectros normalizados junto con su modeloo de continuo
for i in range(5):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara_sim, flujos_normalizados1[i], label=f'Espectro {i+1} (Normalizado Método 1)', c='chocolate') #Estos colorcitos nada que ver
    plt.plot(longitudes_onda_mascara_sim, continuo1[i], label=f'Continuo (Método 1)', c='red', linestyle='--')
    plt.xlabel('Longitud de onda (Å)')
    plt.ylabel('Flujo Normalizado')
```

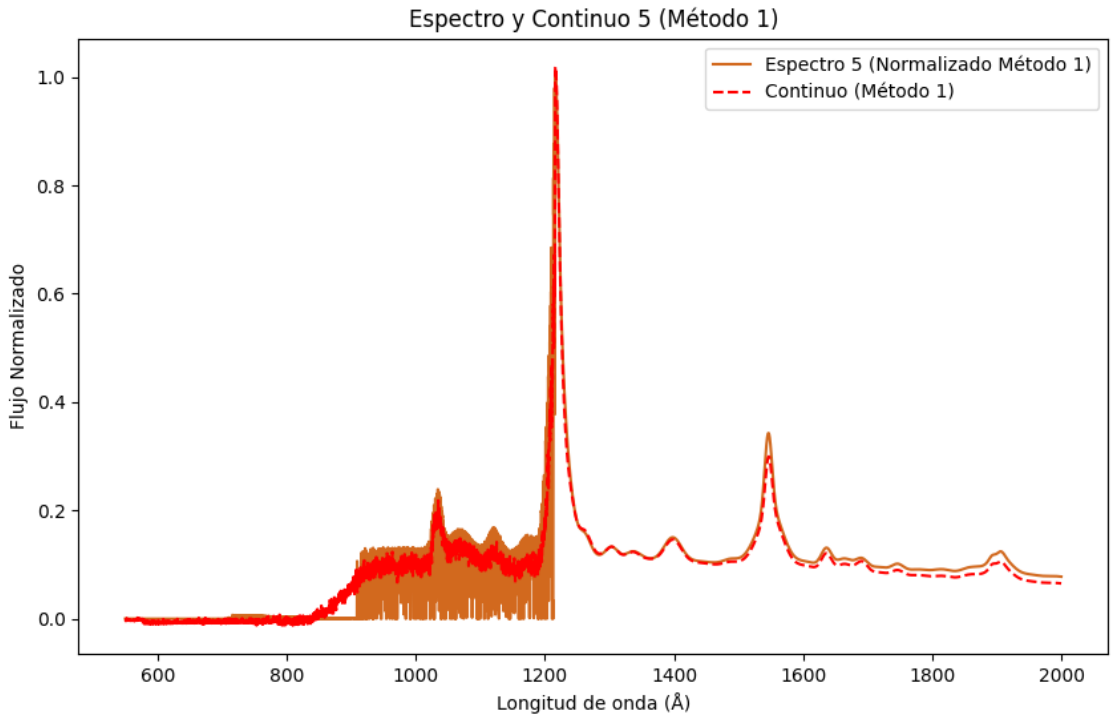
```
    plt.title(f'Espectro y Continuo {i+1} (Método 1)')
    plt.legend()
    plt.show()
```






Espectro y Continuo 3 (Método 1)
Espectro 3 (Normalizado Método 1)
Continuo (Método 1)
Flujo Normalizado
Longitud de onda (Å)
0.0
0.2
0.4
0.6
0.8
1.0
600
800
1000
1200
1400
1600
1800
2000

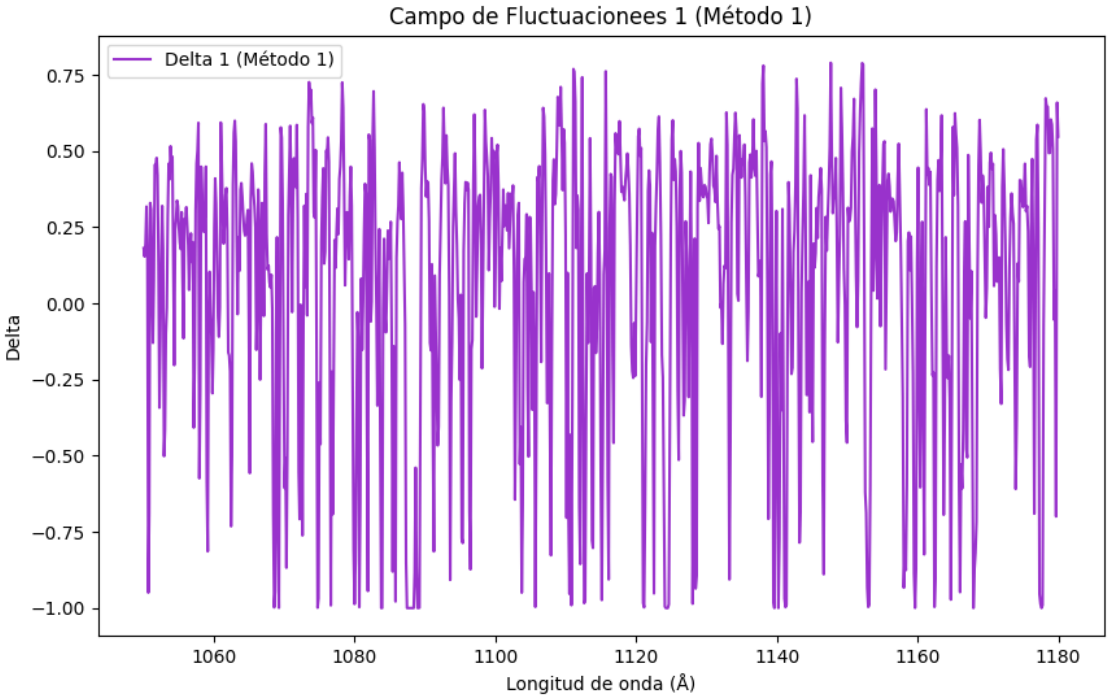

Espectro y Continuo 4 (Método 1)
Espectro 4 (Normalizado Método 1)
Continuo (Método 1)
Flujo Normalizado
Longitud de onda (Å)
0.0
0.2
0.4
0.6
0.8
1.0
600
800
1000
1200
1400
1600
1800
2000

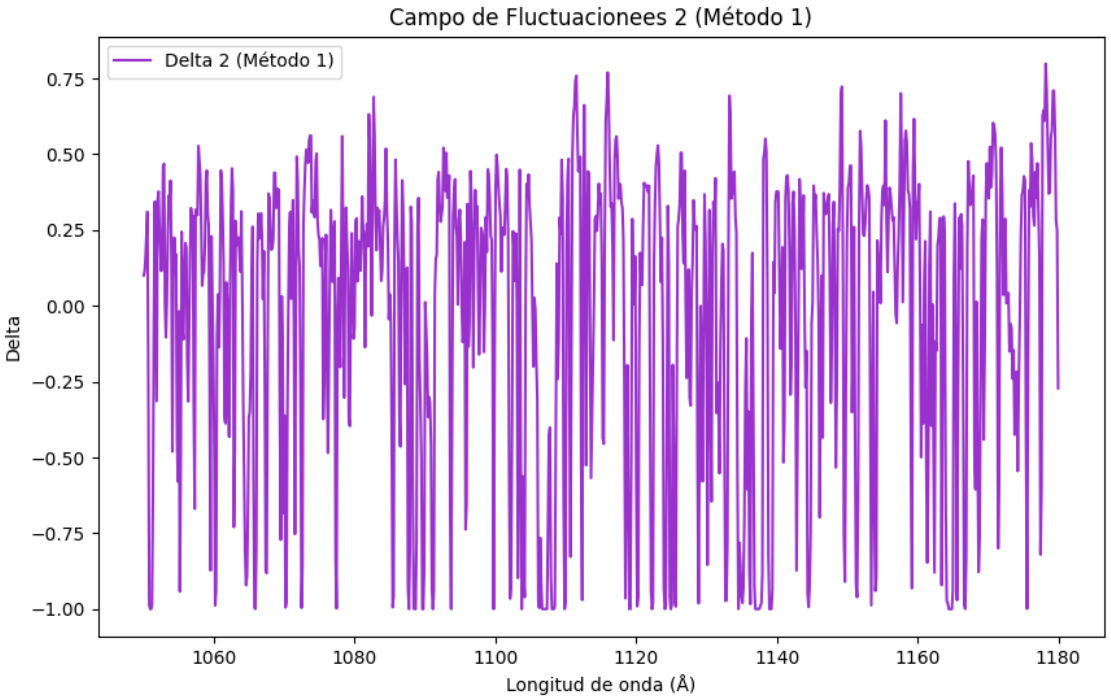


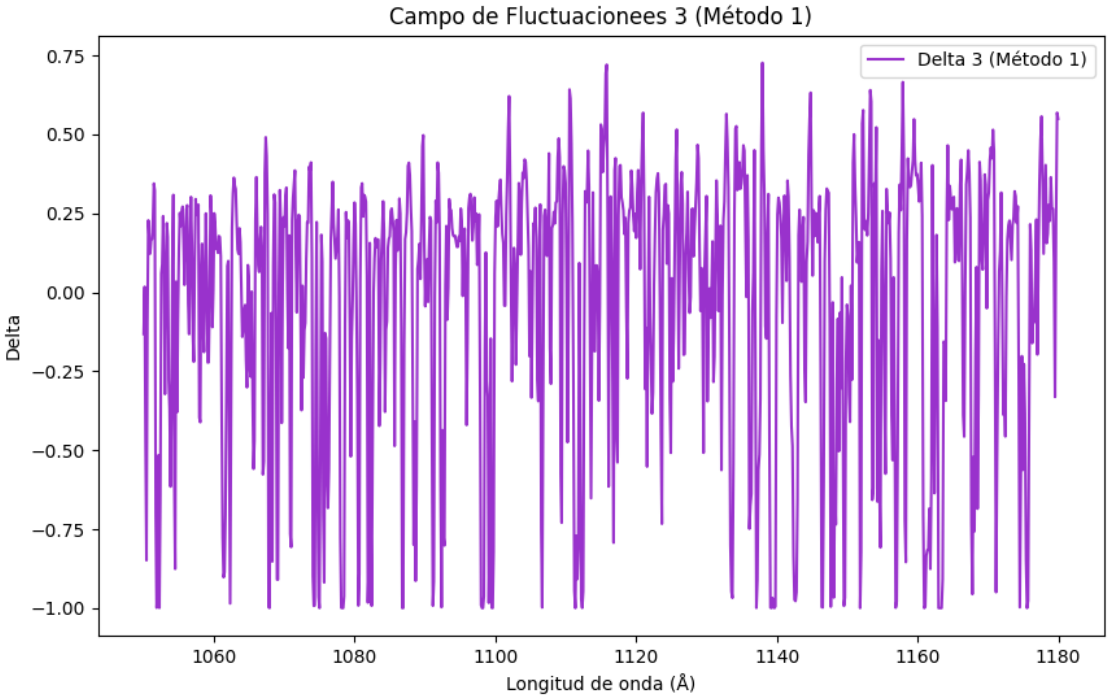
[ ]:
```
# Fórmula para Delta
def compute_delta(flujo, continuo):
    return flujo / continuo - 1
```

[ ]:
```
# Máscara para la región del bosque de Lyman Alpha (1050, 1180)
mask_forest = (longitudes_onda_mascara_sim >= 1050) &␣
 ↪(longitudes_onda_mascara_sim <= 1180)

# Aplicar la máscara a las longitudes de onda
longitudes_onda_bosque = longitudes_onda_mascara_sim[mask_forest]

# Aplicar la máscara a las deltas jeje
delta1_forest = compute_delta(flujos_normalizados1, continuo1)[:, mask_forest]
delta2_forest = compute_delta(flujos_normalizados2, continuo2)[:, mask_forest]
delta3_forest = compute_delta(flujos_normalizados3, continuo3)[:, mask_forest]

# Graficamos los cortes, wi
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_bosque, delta1_forest[i], label=f'Delta {i+1}␣
 ↪(Método 1)', c= 'darkorchid')
    plt.xlabel('Longitud de onda (Å)')
```

```
    plt.ylabel('Delta')
    plt.title(f'Campo de Fluctuacionees {i+1} (Método 1)')
    plt.legend()
    plt.show()

#Delta es un arreglo, tenemos 100, una para cada espectro, xfa q no se te␣
 ↪olvide
```




Campo de Fluctuacionees 2 (Método 1)
Delta 2 (Método 1)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Delta
1060
1080
1100
1120
1140
1160
1180
Longitud de onda (Å)


Campo de Fluctuacionees 3 (Método 1)
Delta 3 (Método 1)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Delta
1060
1080
1100
1120
1140
1160
1180
Longitud de onda (Å)

2.2 Funciones de correlación

[ ]:
```
prom = np.mean(delta1_forest, axis = 1)
prom[1]
```

[ ]:
```
-0.03433291369652697
```

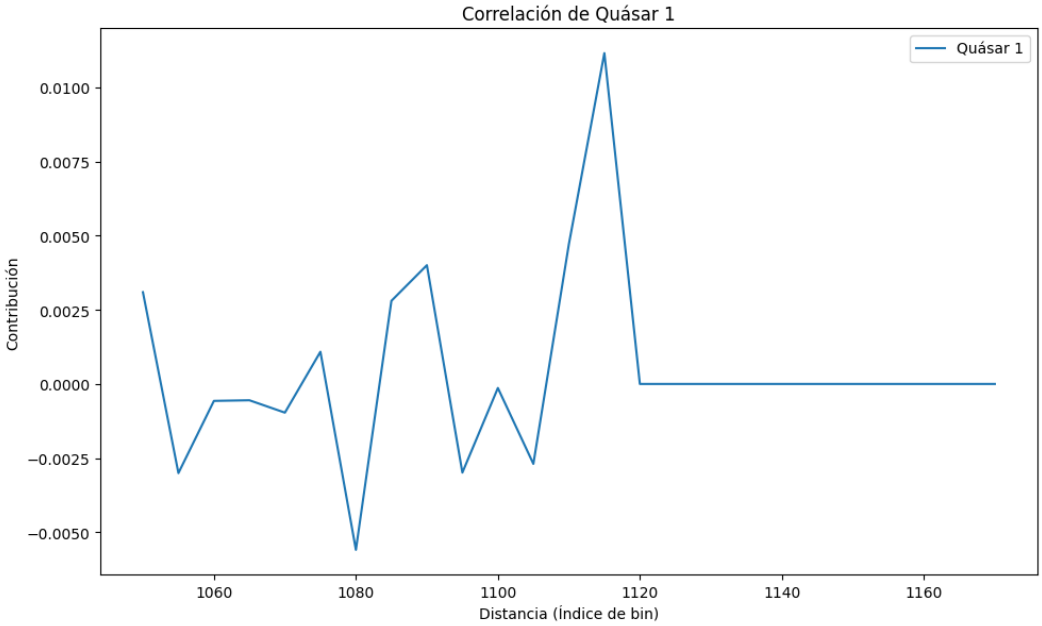
[ ]:
```
#Necesitamos que el promedio sea muy cercano a cero
promedio_delta1_forest = delta1_forest - np.mean(delta1_forest, axis = 1,␣
 ↪keepdims=True)
#delta2_forest_tras = delta2_forest - np.mean(delta2_forest, axis=1,␣
 ↪keepdims=True)
#delta3_forest_tras = delta3_forest - np.mean(delta3_forest, axis=1,␣
 ↪keepdims=True)
```

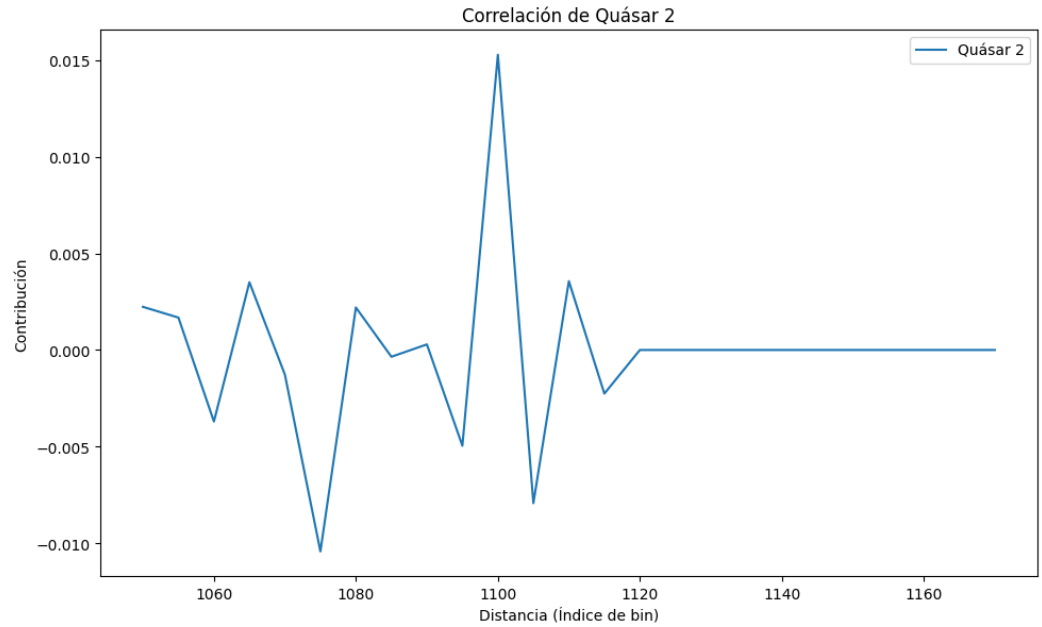
[ ]:
```
# un for para las deltas y otro for para otras deltas
#Hacer el bin a manito
"""  for i [0.933]
        for j [i, 933]
            dist = j-i
            guardar en el hist( int(dist/indice=93.4) + = producto delta i con␣
 ↪la delta j
            también guardar un hist2 que sea lo mimo pero con un +=1

Maybe al final hay que normalizar"""

#si delta es 1 es q datos y random son iguales

#Luego hacer la transformada de Fourier y va a dar algo complejp
""" delta_k = int dx e^{ikx} delta(x)
    p(k) | delta(k) |^2  - módulo
    El módulo es delta(k) delta*(k)
"""

"""arreglar en eje x
-Ahorita puede ser 2pi/lmabda
- ponerlo en amstrongs
velocidades
"""

#todo es por cuásar : Esto hay que hacerlo para cada cuásar

"""paso 1 quitar la media al campo de deltas
paso 2 e el espacio de indices a cuánto correpsonde nuestro tamaño de bin
paso 3:hacer for, distancia entre bines, guardar en el "histograma" que␣
 ↪realmente es chi
paso 4 ver coomo se ve la función
*puede q falte una normalización"""
```

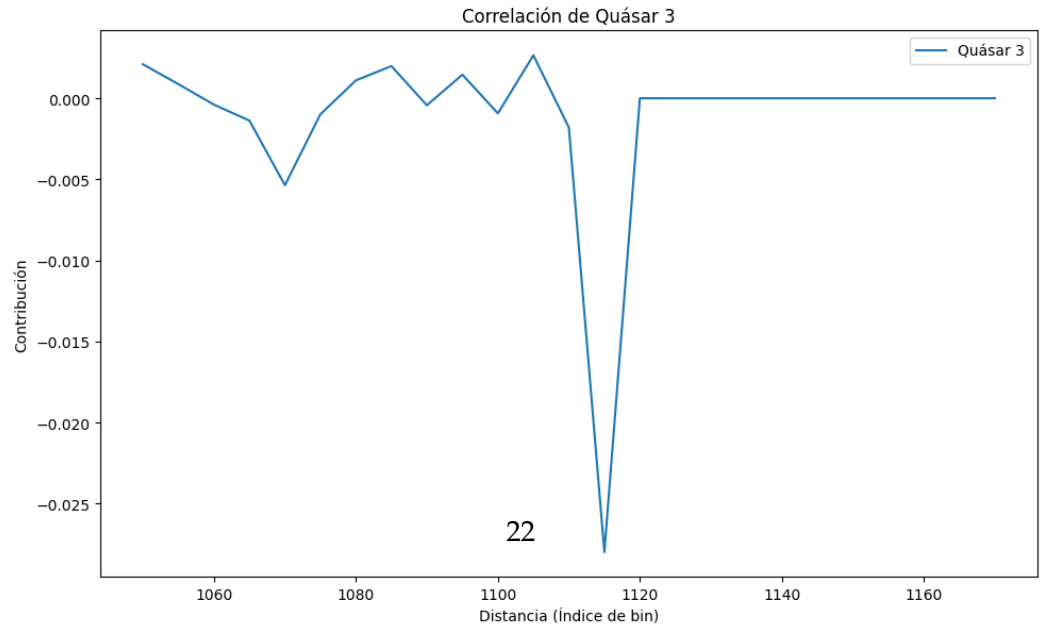
[ ]:
```
'paso 1 quitar la media al campo de deltas\npaso 2 e el espacio de indices a
cuánto correpsonde nuestro tamaño de bin\npaso 3:hacer for, distancia entre
bines, guardar en el "histograma" que realmente es chi\npaso 4 ver coomo se ve
la función\n*puede q falte una normalización'
```

[ ]:
```
# Tamaño del bin
delta_lambda = 5

# Definimos los bins en el rango de longitudes de onda del bosque de Lyman␣
 ↪Alpha
bins = np.arange(1050, 1180, delta_lambda)

# Función para calcular histogramas
def calcular_histogramas(deltas, longitudes_onda_sim, bins, delta_lambda):
    # Arrays de ceros
    num_qso, num_puntos = deltas.shape  # Número de qso y número de puntos en␣
 ↪cada uno
    hist_contador = np.zeros((num_qso, len(bins) - 1))
    hist_contribucion = np.zeros((num_qso, len(bins) - 1))
    hist_div = np.zeros((num_qso, len(bins) - 1))

    for q in range(num_qso):  # Pasamos por cada quásar
        for i in range(num_puntos):  # Pasamos por cada punto
            for j in range(i + 1, num_puntos):  # +1 para evitar la␣
 ↪autocorrelación
                dist = longitudes_onda_sim[j] - longitudes_onda_sim[i]  #␣
 ↪Distancia en longitudes de onda
                bin_idx = int(dist / delta_lambda)  # Índice del bin␣
 ↪correspondiente
                if bin_idx < len(hist_contador[q]):  # Verificamos que el␣
 ↪índice esté en el rango
                    hist_contador[q, bin_idx] += 1  # Contamos la ocurrencia
                    hist_contribucion[q, bin_idx] += deltas[q, i] * deltas[q,␣
 ↪j]  # Contribución de las deltas

        with np.errstate(divide='ignore', invalid='ignore'):
            hist_div[q] = np.where(hist_contador[q] != 0, hist_contribucion[q] /
 ↪ hist_contador[q], 0)

    return hist_contador, hist_contribucion, hist_div  # Regresamos los␣
 ↪histogramas

# Calcular histogramas para promedio_delta1_forest usando las longitudes de␣
 ↪onda
hist_contador, hist_contribucion, hist_div =␣
 ↪calcular_histogramas(promedio_delta1_forest, longitudes_onda_mascara_sim,␣
 ↪bins, delta_lambda)
```

[ ]:
```
# Graficar los histogramas para los primeros 3 quásares en gráficas separadas
fig, axes = plt.subplots(3, 1, figsize=(10, 18))
for i in range(3):
    axes[i].plot(bins[:-1], hist_div[i], label=f'Quásar {i+1}')
    axes[i].set_xlabel('Distancia (Índice de bin)')
    axes[i].set_ylabel('Contribución')
    axes[i].set_title(f'Correlación de Quásar {i+1}')
    axes[i].legend()

plt.tight_layout()
plt.show()
```


Correlación de Quásar 1
Quásar 1
Contribución
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
1060
1080
1100
1120
1140
1160
Distancia (Índice de bin)


Correlación de Quásar 2
Quásar 2
Contribución
0.015
0.010
0.005
0.000
−0.005
−0.010
1060
1080
1100
1120
1140
1160
Distancia (Índice de bin)


Correlación de Quásar 3
Quásar 3
Contribución
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
1060
1080
1100
1120
1140
1160
Distancia (Índice de bin)

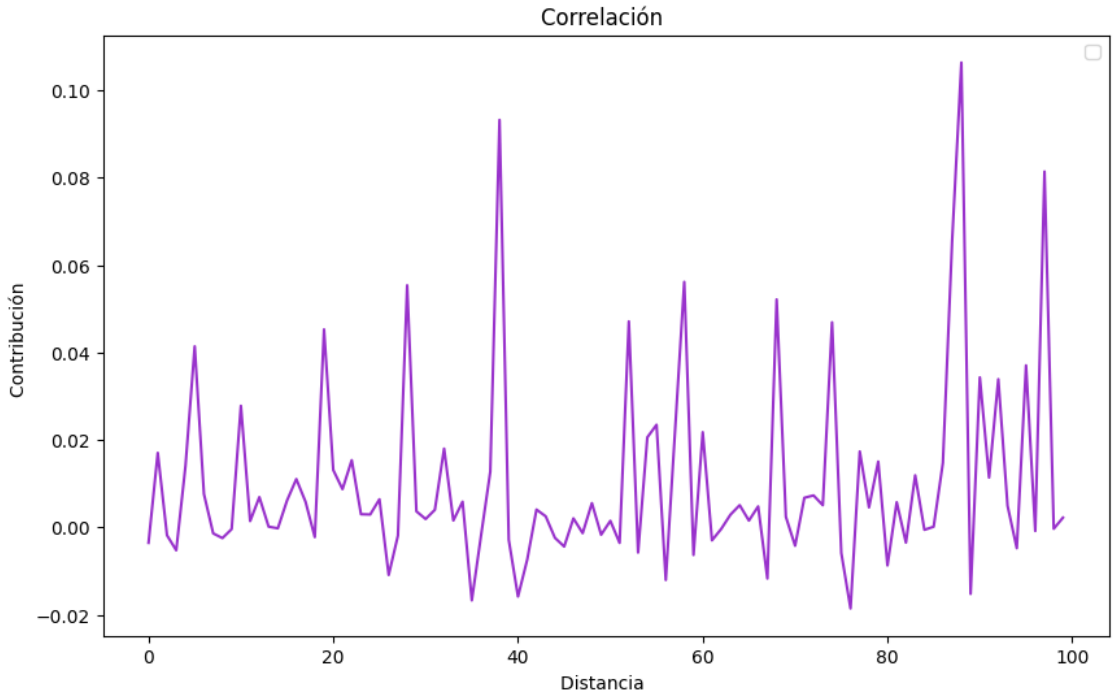
[ ]:
```
#Tamaño de intervalo
delta_lambda = 5

# En bins de posición
def calcular_histogramas_posicion(deltas, delta_lambda):
    num_qso, num_puntos = deltas.shape
    max_dist = num_puntos // delta_lambda


    # Los arrays de ceros ahora tienen que ser bidimensionales
    hist_contador_pos = np.zeros((num_qso, max_dist))
    hist_contribucion_pos = np.zeros((num_qso, max_dist))
    hist_div_pos = np.zeros((num_qso, max_dist))

    for q in range(num_qso):
        for i in range(num_puntos):
            for j in range(i + 1, num_puntos):
                dist = j - i
                bin_idx = int(dist / delta_lambda)
                if bin_idx < max_dist:
                    hist_contador_pos[q, bin_idx] += 1
                    hist_contribucion_pos[q, bin_idx] += deltas[q, i] *␣
 ↪deltas[q, j]

        with np.errstate(divide = 'ignore', invalid = 'ignore'):
          hist_div_pos[q] = np.where(hist_contador_pos[q] !=0,␣
 ↪hist_contribucion_pos[q] / hist_contador_pos[q], 0)

    return hist_contador_pos, hist_contribucion_pos, hist_div_pos

# Lo usamos
hist_contador_pos, hist_contribucion_pos, hist_div_pos =␣
 ↪calcular_histogramas_posicion(promedio_delta1_forest, delta_lambda)
```

[ ]:
```
#a = hist_contribucion_pos[0,:] / hist_contador_pos[0,:]
#a
```

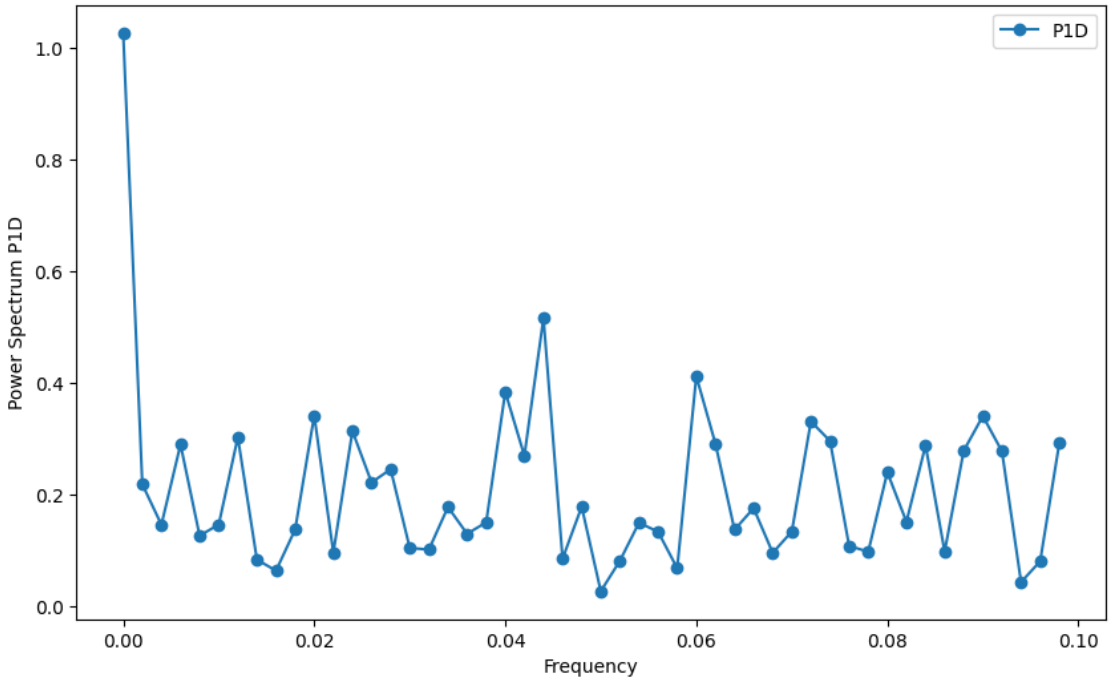
[ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(np.arange(len(hist_div_pos)), hist_div_pos[:,1], c='darkorchid')
plt.xlabel('Distancia ')
plt.ylabel('Contribución ')
plt.title('Correlación ')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
```

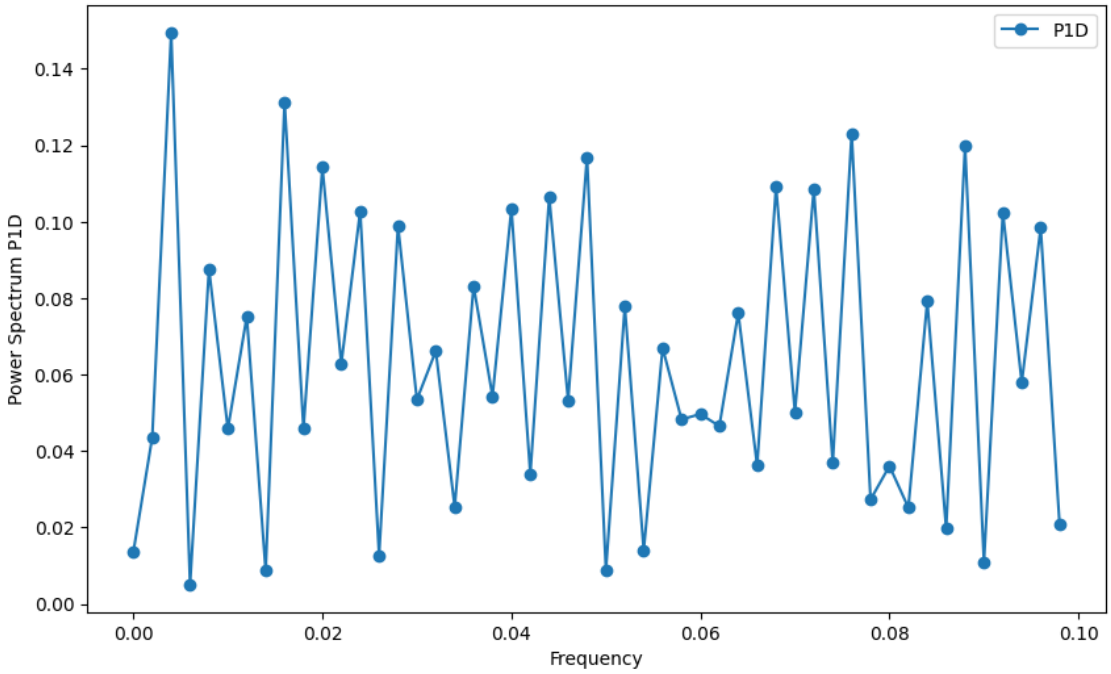
that artists whose label start with an underscore are ignored when legend() is called with no argument.



[ ]:
```
bin_size = 5  ### Tamaño de bins usado para la correlación
bin_edges = np.arange(len(hist_div_pos[:,1])) * bin_size

P1D = np.fft.fft(hist_div_pos[:,1])
freq = np.fft.fftfreq(len(hist_div_pos[:,1]), d=bin_size)

# Para solo tomar las frecuencias positivas y sus valores correspondientes del␣
 ↪espectro
positive_freq = freq[:len(freq)//2]
P1D_positive = np.abs(P1D[:len(P1D)//2])

plt.figure(figsize=(10, 6))
plt.plot(positive_freq, P1D_positive, marker='o', linestyle='-', label='P1D')
plt.xlabel('Frequency')
plt.ylabel('Power Spectrum P1D')
plt.legend()
plt.show()
```

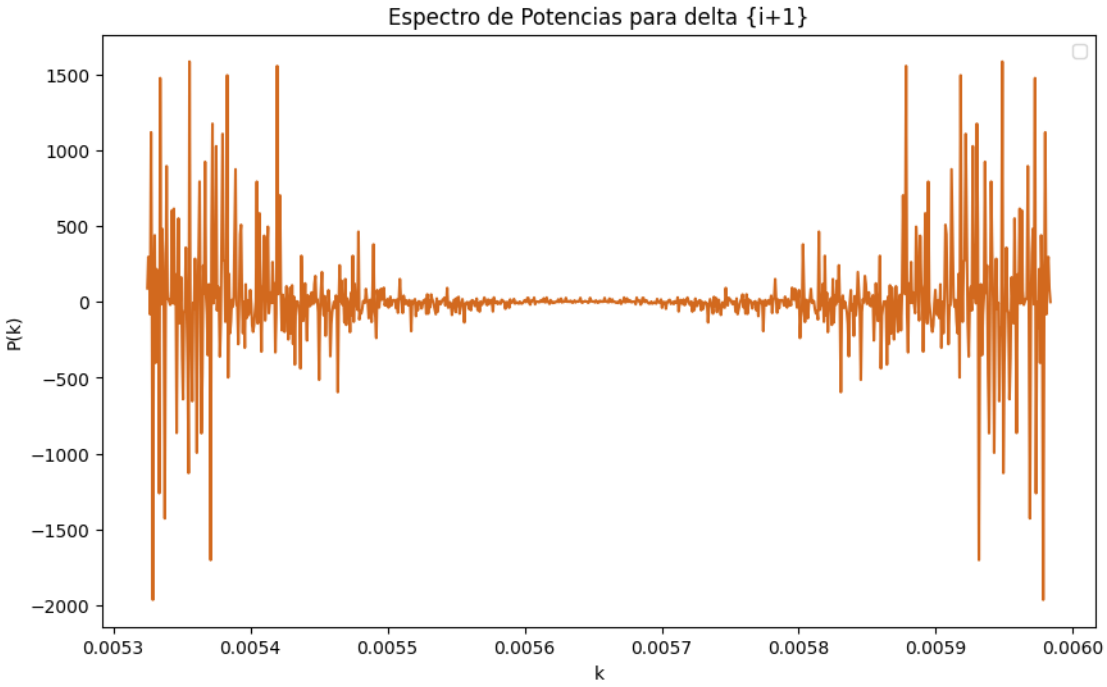


[ ]:
```
bin_size = 5  ### Tamaño de bins usado para la correlación
bin_edges = np.arange(len(hist_div[:,1])) * bin_size

P1D = np.fft.fft(hist_div[:,1])
freq = np.fft.fftfreq(len(hist_div[:,1]), d=bin_size)

# Para solo tomar las frecuencias positivas y sus valores correspondientes del␣
 ↪espectro
positive_freq = freq[:len(freq)//2]
P1D_positive = np.abs(P1D[:len(P1D)//2])

plt.figure(figsize=(10, 6))
plt.plot(positive_freq, P1D_positive, marker='o', linestyle='-', label='P1D')
plt.xlabel('Frequency')
plt.ylabel('Power Spectrum P1D')
plt.legend()
plt.show()
```



[ ]:
```
# Calcular valores de k correspondientes a nuestras lambda
lambda_mean = np.mean(longitudes_onda_bosque)
k_values = 2 * np.pi / longitudes_onda_bosque

# Fourier
def calcular_delta_k(deltas):
    return np.fft.fft(deltas, axis=1)

delta_k1 = calcular_delta_k(promedio_delta1_forest)

# Aquí hacemos el módulo
def calcular_potencia(delta_k):
    return np.abs(delta_k) * np.conjugate(delta_k)

potencia1 = calcular_potencia(delta_k1)

# Graficamos, yipi
for i in range(1):
  plt.figure(figsize=(10, 6))
  plt.plot(k_values, potencia1[i], c='chocolate')
  plt.xlabel('k')
  plt.ylabel('P(k)')
  plt.title('Espectro de Potencias para delta {i+1}')
  plt.legend()
  plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py:1335:
ComplexWarning: Casting complex values to real discards the imaginary part
  return np.asarray(x, float)
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
```



3 Datos EDR

Aquí vamos a comenzar a correr el código si estamos usando los datos reales. Primero vamos a elegir uno de nuestros archivos y leer sus metadatos, estos contienen detalles relevantes sobre la información contenida en el archivo, pero que no necesariamente conforma parte de los datos dentros, en este caso, por ejemplo: RA y DEC, que son asención recta y decliación, llamadas coordenadas celestes, esto nos dice donde está ubicado el cuásar, lo que no es esencial para nuestro análisis.

[ ]:
```
#Aquí vamos a explorar un poco la estructura de los archivos, en este caso los␣
 ↪metadatos del archivo, para entender un poco mejor su estructura

# Vamos a elegir un archivo en específico, esta es la ruta al archivo FITS,␣
 ↪puedes elegir el archivo que quieras:
path_metadata = '/content/drive/My Drive/Verano2024/2.2-2.4/
 ↪data_39633453527861912.fits'

# Abrir el archivo FITS y extraer los metadatos específicos
```

```python
with fits.open(path_metadata) as hdul:
    # Imprimir información básica sobre el archivo
    print(hdul.info())

    # Iterar sobre cada extensión (HDU) del archivo
    for hdu in hdul:
        # Imprimir metadata de cada extensión
        print(hdu.header)
        print("\n")
```

```
Filename: /content/drive/My Drive/Verano2024/2.2-2.4/data_39633453527861912.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  B_WAVELENGTH    1 PrimaryHDU      15   (2751,)   float64
  1  B_FLUX        1 ImageHDU         7   (2751,)   float32
  2  B_IVAR        1 ImageHDU         7   (2751,)   float32
None
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -64 / number of bits per data pixel
NAXIS   =                    1 / number of data axes
NAXIS1  =                 2751 / length of data axis 1
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in
'AstronomyCOMMENT   and Astrophysics', volume 376, page 359; bibcode:
2001A&A...376..359H EXTNAME = 'B_WAVELENGTH'
TARGETID=    39633453527861912 / Target ID
Z_VALUE =       2.215405029813 / Redshift (Z) value
RA      =     3.89484063193223 / Right Ascension
DEC     =     1.15222048171649 / Declination
SPECTYPE= 'QSO     '           / Spectral Type
SURVEY  = 'special '           / Survey Name
PROGRAM = 'dark    '           / Program Name
END


XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    1 / number of data axes
NAXIS1  =                 2751 / length of data axis 1
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
EXTNAME = 'B_FLUX  '
END


XTENSION= 'IMAGE   '           / IMAGE extension
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    1 / number of data axes
```

```
NAXIS1  =                 2751 / length of data axis 1
PCOUNT  =                    0 / required keyword; must = 0
GCOUNT  =                    1 / required keyword; must = 1
EXTNAME = 'B_IVAR  '
END
```

Lo siguiento que hacemos es extraer los datos que sabemos que nos serán útiles, en este caso es el valor del redshift (Z_VALUE), target ID (TARGETID, este es como un número de identifiación para cada cuásar), los valores de las longitudes de onda y flujo en el azul (B_WAVELENGTH Y B_FLUX). Esto puede tomar unos pocos minutos debido a la cantidad de cuásares que estamos procesando.

[ ]:
```
# Ahora vamos a extraer la información que necesitamos, para este caso, sólo va␣
 ↪a ser necesario utilizar el flujo y longitud de onda del azul

# Listar todos los archivos FITS en la carpeta
archivo_fits = [f for f in os.listdir(path1) if f.endswith('.fits')]

# Listas para almacenar los datos
z_values = []
b_fluxes = []
target_ids = []
b_wavelengths = []

# Procesar cada archivo FITS y extraer la información
for fits_file in archivo_fits:
    file_path = os.path.join(path1, fits_file)

    with fits.open(file_path) as hdul:
        header = hdul[0].header
        target_id = header.get('TARGETID')
        z_value = header.get('Z_VALUE')

        b_wavelength = hdul['B_WAVELENGTH'].data
        b_flux = hdul['B_FLUX'].data

        # Añadir los datos a las listas para poder hacer una tabla
        target_ids.append(target_id)
        z_values.append(z_value)
        b_wavelengths.append(b_wavelength)
        b_fluxes.append(b_flux)

# Crear la tabla Astropy
table = Table([target_ids, z_values, b_wavelengths, b_fluxes],␣
 ↪names=('Target_ID', 'Redshift', 'b_wavelength', 'b_flux'))
```

Para tener una idea de cómo luce la tabla que creamos y que valores contiene, imprimimos las

primeras cinco columnas de nuestra tabla. Esto puede ayudarnos a comenzar a entender cómo están estructurados nuestros datos.

[ ]:
```
print(table[0:5])
```

```
    Target_ID         Redshift            b_wavelength                 b_flux
---------------- ---------------- ------------------------
------------------------
39633328088811503 2.20347569102801 3600.0 .. 5800.0000000005  0.75633854 ..
0.19721018
39627878844859402  2.2459811582367 3600.0 .. 5800.0000000005  1.4971946 ..
-0.71702373
39633315514287680 2.20970919907885 3600.0 .. 5800.0000000005 -0.35957497 ..
0.40593535
39627652612497366 2.30414260205499 3600.0 .. 5800.0000000005  1.4533299 ..
-0.30566147
39632981853211830 2.29378180897824 3600.0 .. 5800.0000000005   1.5392184 ..
0.17059562
```

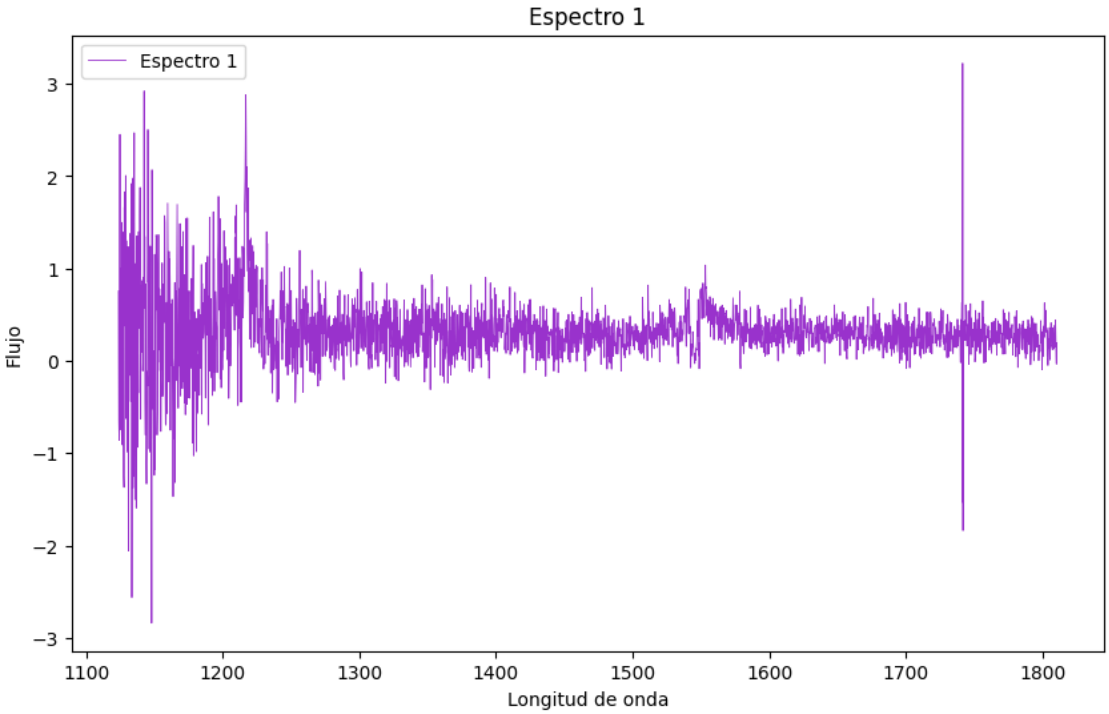
Como arriba ya teníamos un código estructurado para realizar un análisis de las simulaciones, nuevamente vamos a extraer datos necesarios de una forma que se adapte mejor al código que ya tenemos.

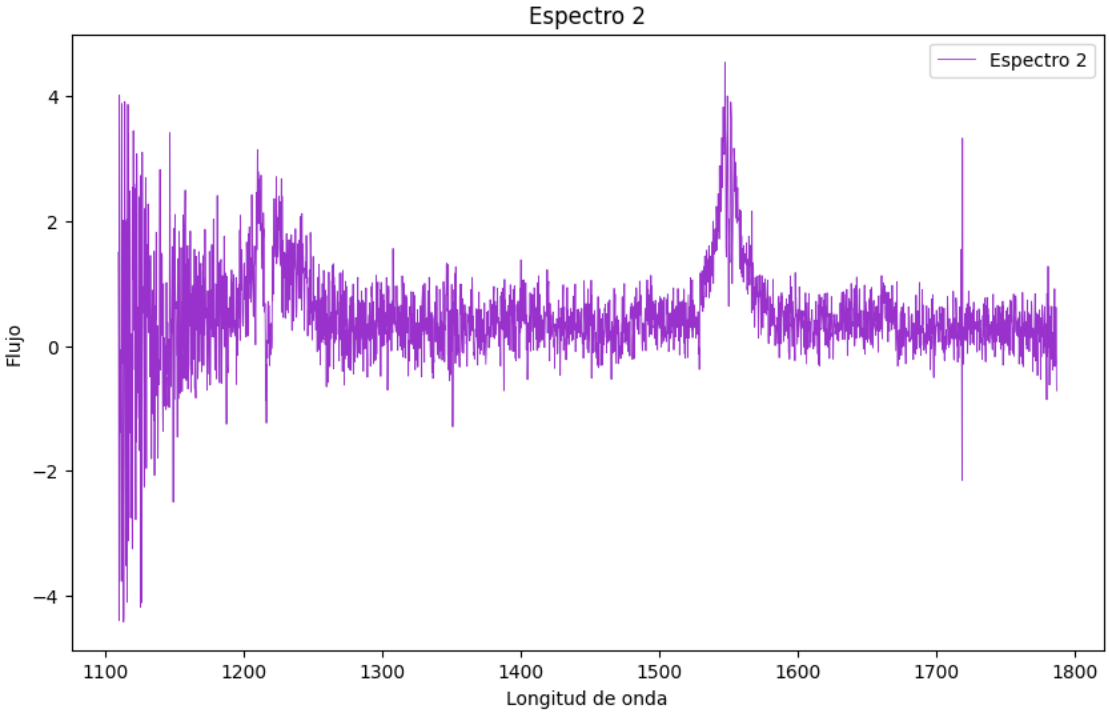
[ ]:
```
# Listas para almacenar los datos
target_ids = []
z_values = []
b_wavelengths = []
b_fluxes = []

# Procesar cada archivo FITS
for fits_file in archivo_fits:
    file_path = os.path.join(path1, fits_file)

    with fits.open(file_path) as hdul:
        header = hdul[0].header
        target_id = header.get('TARGETID')
        z_value = header.get('Z_VALUE')

        b_wavelength = hdul['B_WAVELENGTH'].data
        b_flux = hdul['B_FLUX'].data

        # Añadir los datos a las listas
        target_ids.append(target_id)
        z_values.append(z_value)
        b_wavelengths.append(b_wavelength)
        b_fluxes.append(b_flux)

# Crear arrays de arrays para b_flux y b_wavelength ajustados por el redshift
b_fluxes_array = np.array(b_fluxes)
```

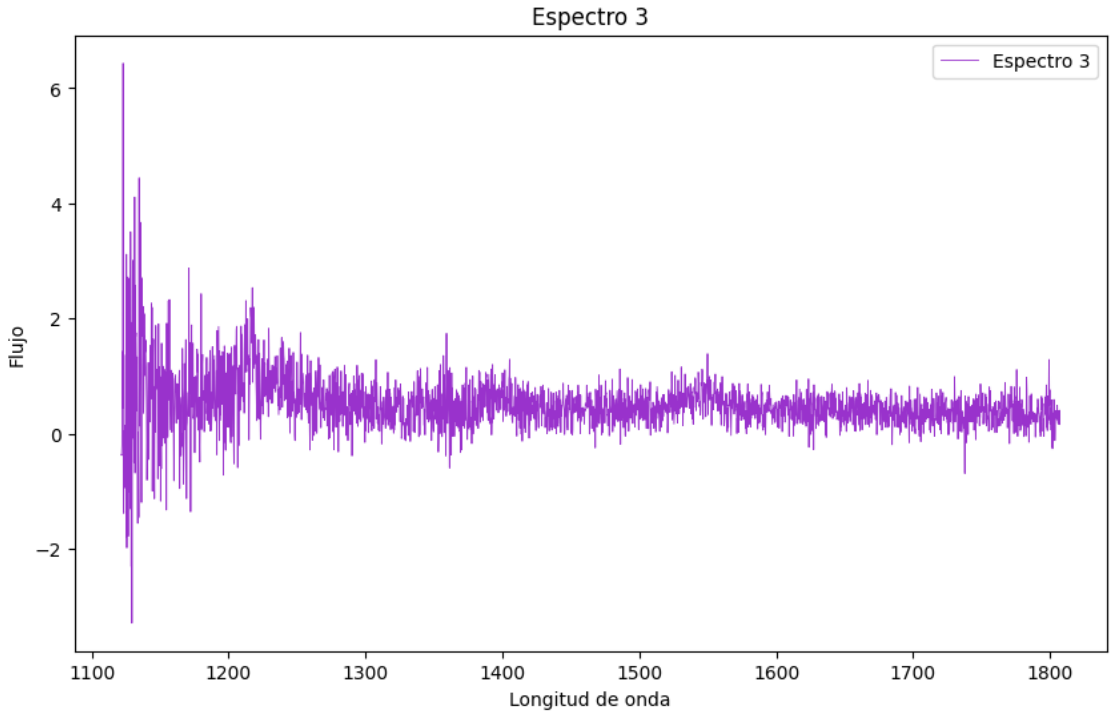
```
b_wavelengths_array = np.array([b_wavelengths[i] / (1 + z_values[i]) for i in␣
 ↪range(len(b_wavelengths))])

# Crear el array 'datos'
datos = [b_fluxes_array, b_wavelengths_array]
```

Vamos a visualizar los primeros tres espectros (puedes ver cuántos o los que tú quieras), para ver los espectros.

[ ]:
```
# Hacer la máscara para los datos que nos resultan más útiles
longitudes_onda = datos[1]
flujos_sin_normalizar = datos[0]

mask = (longitudes_onda >= 550) & (longitudes_onda <= 2000)

longitudes_onda_mascara = longitudes_onda[:, mask[0]]
flujos_mascara = flujos_sin_normalizar[:, mask[0]]

# Graficar los primeros 3 espectros
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara[i], flujos_mascara[i], label=f'Espectro␣
 ↪{i+1}', linewidth=0.6, c='darkorchid')
    plt.xlabel('Longitud de onda')
    plt.ylabel('Flujo')
    plt.title(f'Espectro {i+1}')
    plt.legend()
    plt.show()
```


Espectro 1
Espectro 1
Flujo
Longitud de onda
3
2
1
0
−1
−2
−3
1100
1200
1300
1400
1500
1600
1700
1800


Espectro 2
Espectro 2
Flujo
Longitud de onda
4
2
0
−2
−4
1100
1200
1300
1400
1500
1600
1700
1800



[ ]:
```
# Definimos las funciones

# Usando una altura media entre el máximo y mínimo de cada espectro
def normalizar_altura_media(flujo):
    flujo_min = np.min(flujo, axis=1)[:, np.newaxis]
    flujo_max = np.max(flujo, axis=1)[:, np.newaxis]

    # Handle potential division by zero
    denominator = flujo_max - flujo_min
    normalized_flujo = np.where(denominator != 0, (flujo - flujo_min) /␣
 ↪denominator, 1)

    return normalized_flujo # NOTA: más bien era máximo y el mínimo y quitarle␣
 ↪esa distancia al cuásar, algo como restar (maximo-min)/2

# Promedio de alturas para un pequeño rango en lambda
def normalizar_promedio_altura(flujo, wavelength, rango_min, rango_max):
    rango_mascara = (wavelength >= rango_min) & (wavelength <= rango_max)
    # Ensure rango_mascara is applied to the second axis of flujo
    prom_flujo = np.mean(flujo[:, rango_mascara[0]], axis=1)[:, np.newaxis]

    # Handle potential division by zero
    normalized_flujo = np.where(prom_flujo != 0, flujo / prom_flujo, 1)
```

```
    return normalized_flujo

# Promedio de todas
def normalizar_promedio_todas(flujo):
        mean_flujo = np.mean(flujo, axis=1)[:, np.newaxis]
        # Handle spectra with all zeros
        normalized_flujo = np.where((mean_flujo != 0) & (np.any(flujo != 0,␣
 ↪axis=1)[:, np.newaxis]),
                                    flujo / mean_flujo, np.nan)
        return normalized_flujo
```

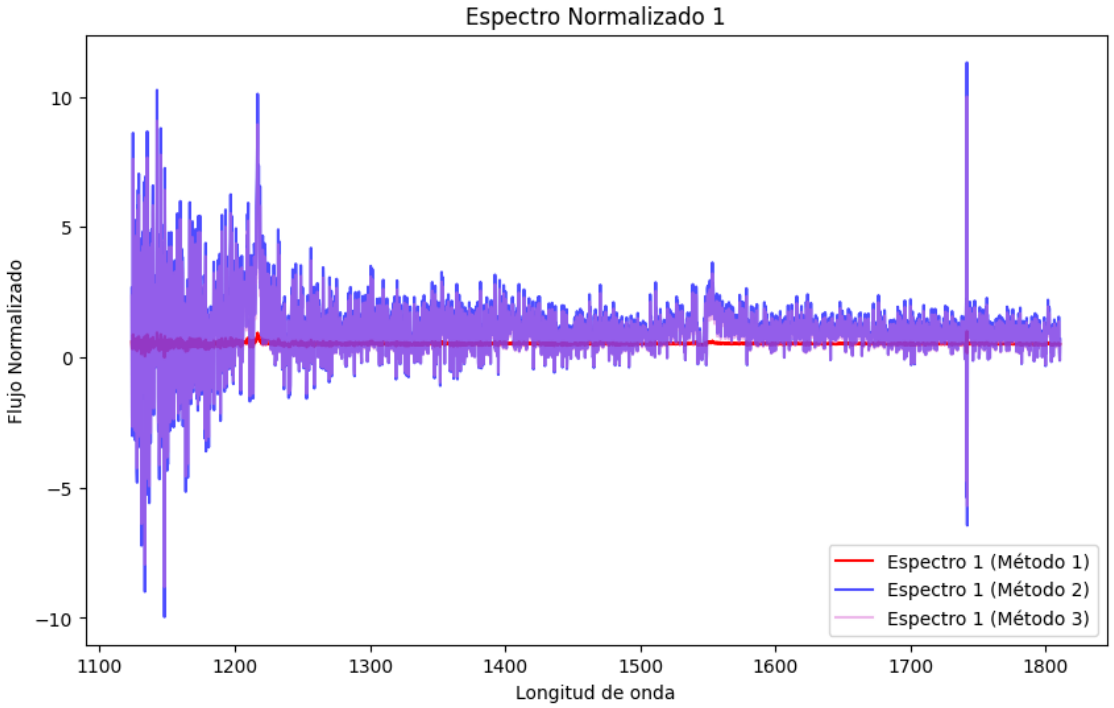
[ ]:
```
longitudes_onda_mascara.shape
```

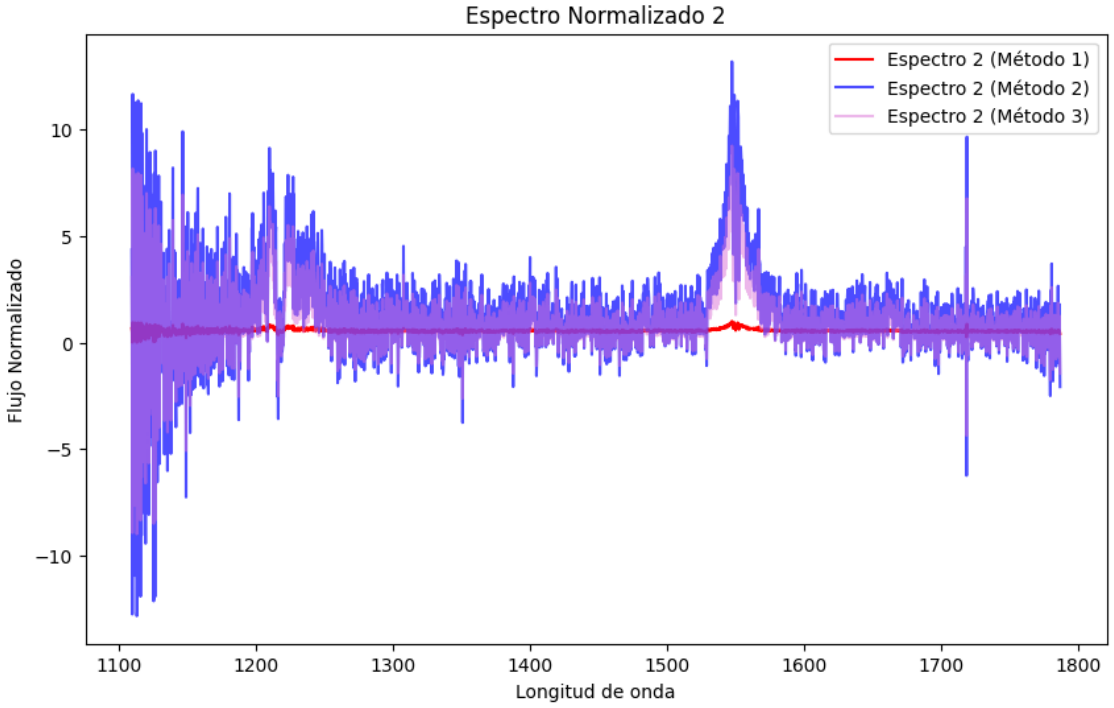
[ ]:
```
(6329, 2751)
```

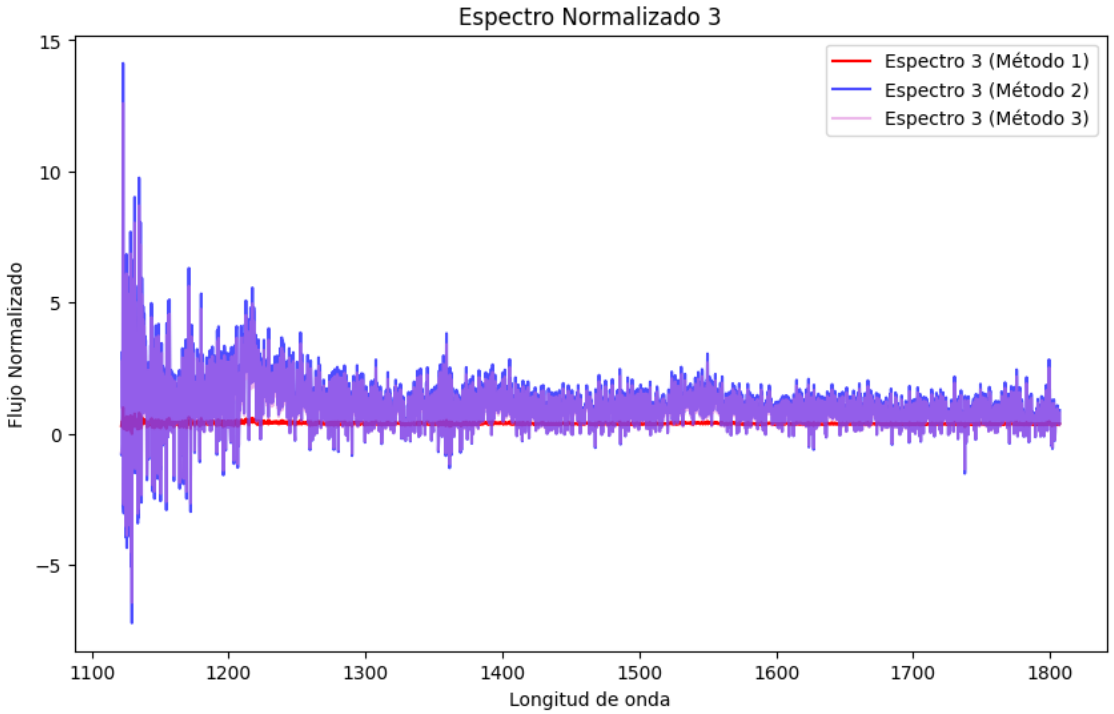
[ ]:
```
# Aplicamos las funciones de normalización a los datos enmascarados
flujos_normalizados1 = normalizar_altura_media(flujos_mascara)
flujos_normalizados2 = normalizar_promedio_altura(flujos_mascara,␣
 ↪longitudes_onda_mascara, 1400, 1500) # De preferencia más a la derecha
flujos_normalizados3 = normalizar_promedio_todas(flujos_mascara)
```

```
<ipython-input-64-d1ce71e06c7a>:10: RuntimeWarning: invalid value encountered in
divide
  normalized_flujo = np.where(denominator != 0, (flujo - flujo_min) /
denominator, 1)
<ipython-input-64-d1ce71e06c7a>:21: RuntimeWarning: invalid value encountered in
divide
  normalized_flujo = np.where(prom_flujo != 0, flujo / prom_flujo, 1)
<ipython-input-64-d1ce71e06c7a>:30: RuntimeWarning: invalid value encountered in
divide
  flujo / mean_flujo, np.nan)
```

[ ]:
```
# Graficamos los primeros 3 espectros juntos (el original y las tres␣
 ↪normalizaciones)
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara[i], flujos_normalizados1[i],␣
 ↪label=f'Espectro {i+1} (Método 1)', c='red')
    plt.plot(longitudes_onda_mascara[i], flujos_normalizados2[i],␣
 ↪label=f'Espectro {i+1} (Método 2)', c='blue', alpha=0.7)
    plt.plot(longitudes_onda_mascara[i], flujos_normalizados3[i],␣
 ↪label=f'Espectro {i+1} (Método 3)', c='orchid', alpha=0.5)
    plt.xlabel('Longitud de onda')
    plt.ylabel('Flujo Normalizado')
    plt.title(f'Espectro Normalizado {i+1}')
    plt.legend()
    plt.show()
```


Espectro Normalizado 1
Flujo Normalizado
Longitud de onda
Espectro 1 (Método 1)
Espectro 1 (Método 2)
Espectro 1 (Método 3)


Espectro Normalizado 2
Flujo Normalizado
Longitud de onda
Espectro 2 (Método 1)
Espectro 2 (Método 2)
Espectro 2 (Método 3)



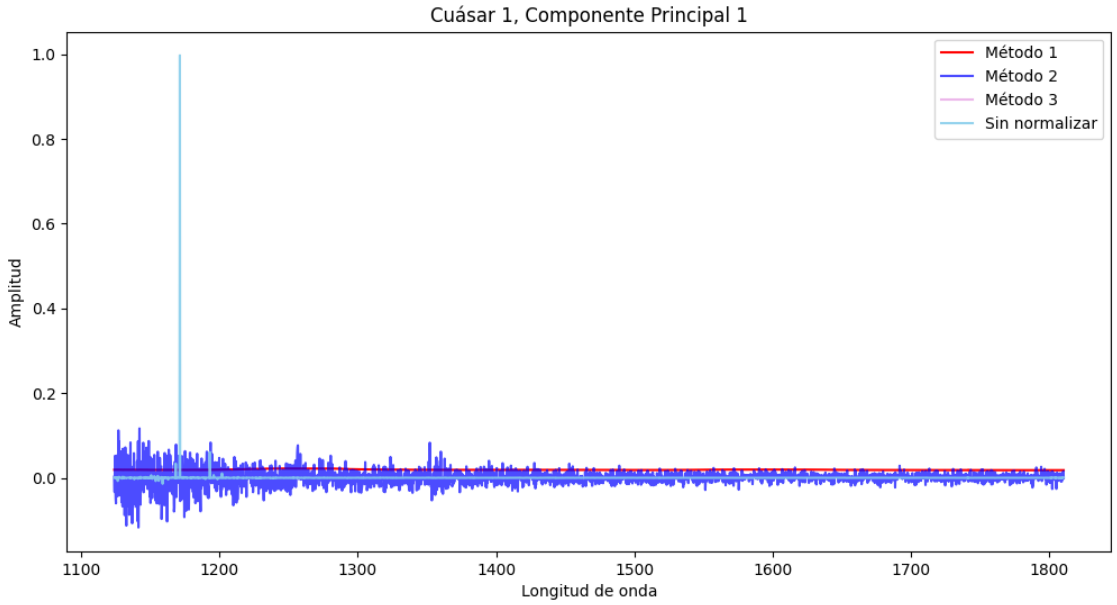
[ ]:
```
#Llamamos al código que importamos en collab
import empca
from empca import empca

#Definimos el número de eigenvectores que queremos y las iteraciones
nvec = 10
niter = 10

weights = np.ones_like(flujos_mascara)

# Aplicar EMPCA para cada método de normalización y sin normalizar
modelo = empca(flujos_mascara, weights, niter=niter, nvec=nvec)
modelo1 = empca(flujos_normalizados1, weights, niter=niter, nvec=nvec)
modelo2 = empca(flujos_normalizados2, weights, niter=niter, nvec=nvec)
modelo3 = empca(flujos_normalizados3, weights, niter=niter, nvec=nvec)
```

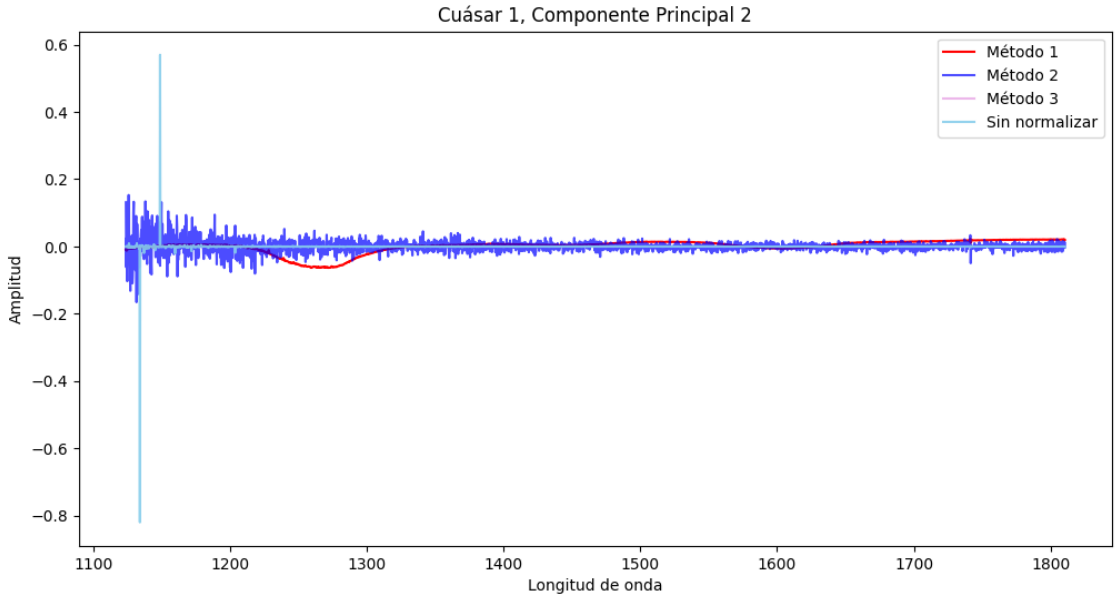
```
       iter         R2             rchi2
EMPCA  1/10       0.05602670    284.48500822
EMPCA  2/10       0.79950965     59.90539826
EMPCA  3/10       0.90107768     29.53030689
EMPCA  4/10       0.92403531     22.67190064
EMPCA  5/10       0.92699699     21.78397773
EMPCA  6/10       0.93002098     20.87454801
EMPCA  7/10       0.93105224     20.56562809
```

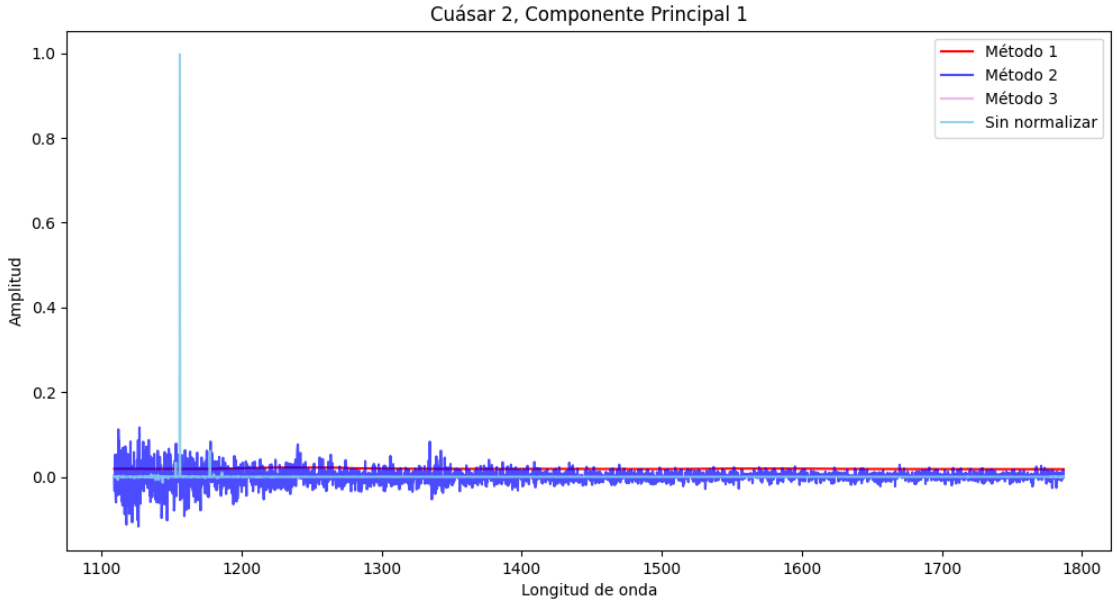
```
EMPCA  8/10       0.93227234     20.20163954
EMPCA  9/10       0.93436978     19.57603344
EMPCA 10/10       0.93633275     18.99052695
R2: 0.9419797538318626
       iter        R2             rchi2
EMPCA  1/10       0.00474055      0.19882524
EMPCA  2/10       0.82081307      0.00375815
EMPCA  3/10       0.84384931      0.00327500
EMPCA  4/10       0.84596407      0.00323064
EMPCA  5/10       0.84619674      0.00322576
EMPCA  6/10       0.84617646      0.00322619
EMPCA  7/10       0.84610499      0.00322769
EMPCA  8/10       0.84603579      0.00322914
EMPCA  9/10       0.84599930      0.00322990
EMPCA 10/10       0.84601214      0.00322964
R2: 0.8467196511848053
       iter        R2             rchi2
EMPCA  1/10       0.05124278   1986.98171037
EMPCA  2/10       0.78020058    461.40397451
EMPCA  3/10       0.90033998    209.98058972
EMPCA  4/10       0.92377377    160.94059162
EMPCA  5/10       0.92283395    162.90706533
EMPCA  6/10       0.94821151    109.78712010
EMPCA  7/10       0.96763870     69.12007427
EMPCA  8/10       0.97391219     55.98570724
EMPCA  9/10       0.97564910     52.34844082
EMPCA 10/10       0.97622709     51.13798426
R2: 0.9770243675876091
       iter        R2             rchi2
EMPCA  1/10             nan             nan
EMPCA  2/10             nan             nan
EMPCA  3/10             nan             nan
EMPCA  4/10             nan             nan
EMPCA  5/10             nan             nan
EMPCA  6/10             nan             nan
EMPCA  7/10             nan             nan
EMPCA  8/10             nan             nan
EMPCA  9/10             nan             nan
EMPCA 10/10             nan             nan
R2: nan
```

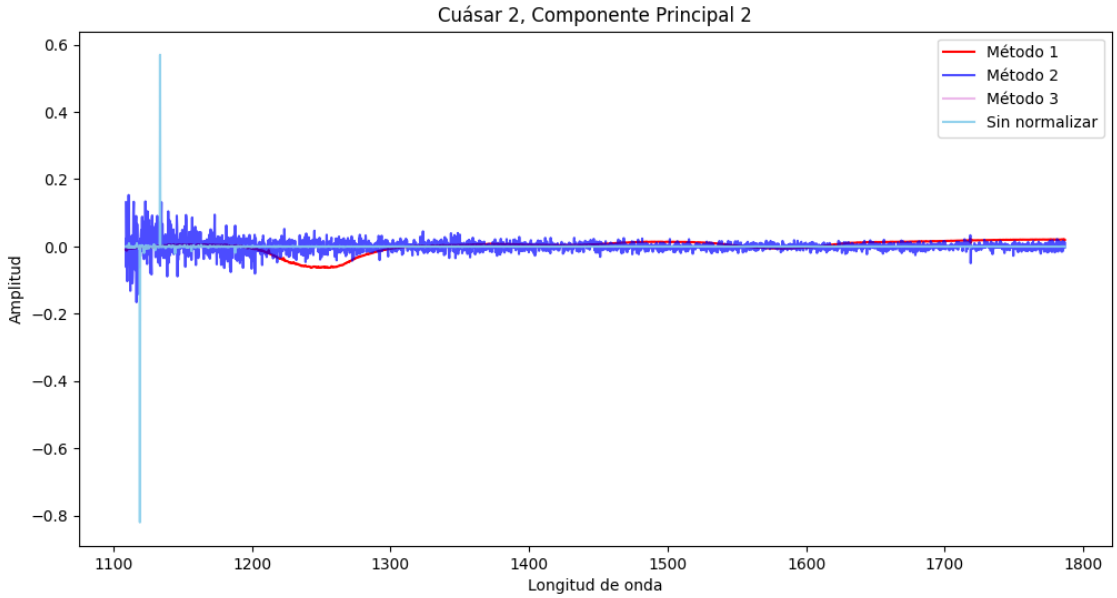
[ ]:
```python
# Iteramos sobre los primeros tres cuásares (i de 0 a 2)
for i in range(3):
    # Iteramos sobre las dos primeras componentes principales (j de 0 a 1)
    for j in range(2):
        plt.figure(figsize=(12, 6))
```

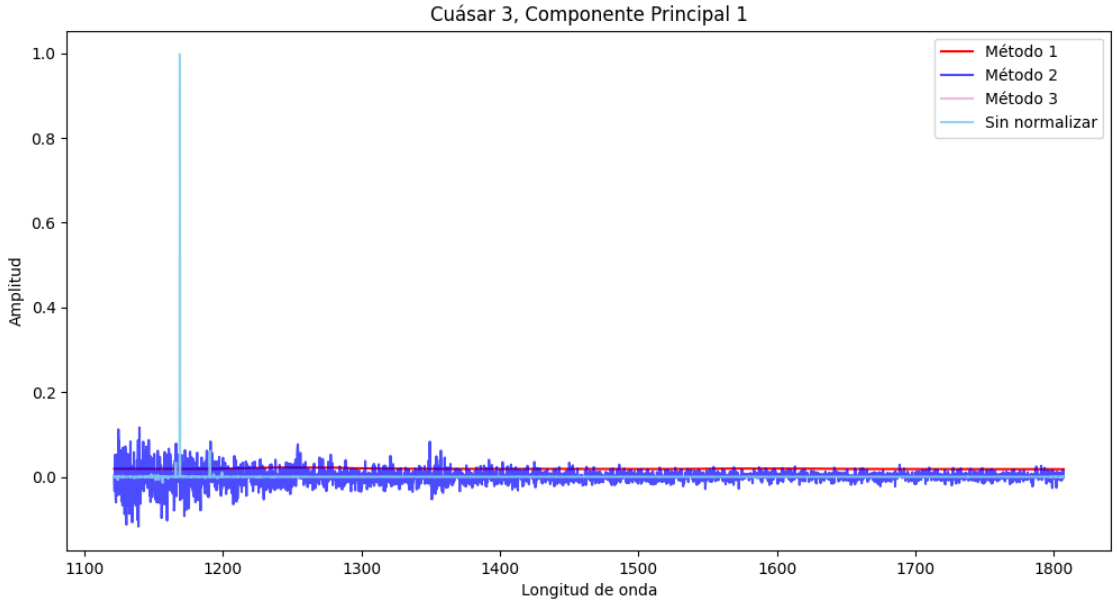
```
        plt.plot(longitudes_onda_mascara[i], modelo1.eigvec[j], label=f'Método␣
↪1', c='red')
        plt.plot(longitudes_onda_mascara[i], modelo2.eigvec[j], label=f'Método␣
↪2', c='blue', alpha = 0.7)
        plt.plot(longitudes_onda_mascara[i], modelo3.eigvec[j], label=f'Método␣
↪3', c='orchid', alpha = 0.5)
        plt.plot(longitudes_onda_mascara[i], modelo.eigvec[j], label=f'Sin␣
↪normalizar' , c='skyblue', alpha = 0.9)

        plt.xlabel('Longitud de onda')
        plt.ylabel('Amplitud')
        plt.title(f'Cuásar {i+1}, Componente Principal {j+1}')
        plt.legend()
        plt.show()
```

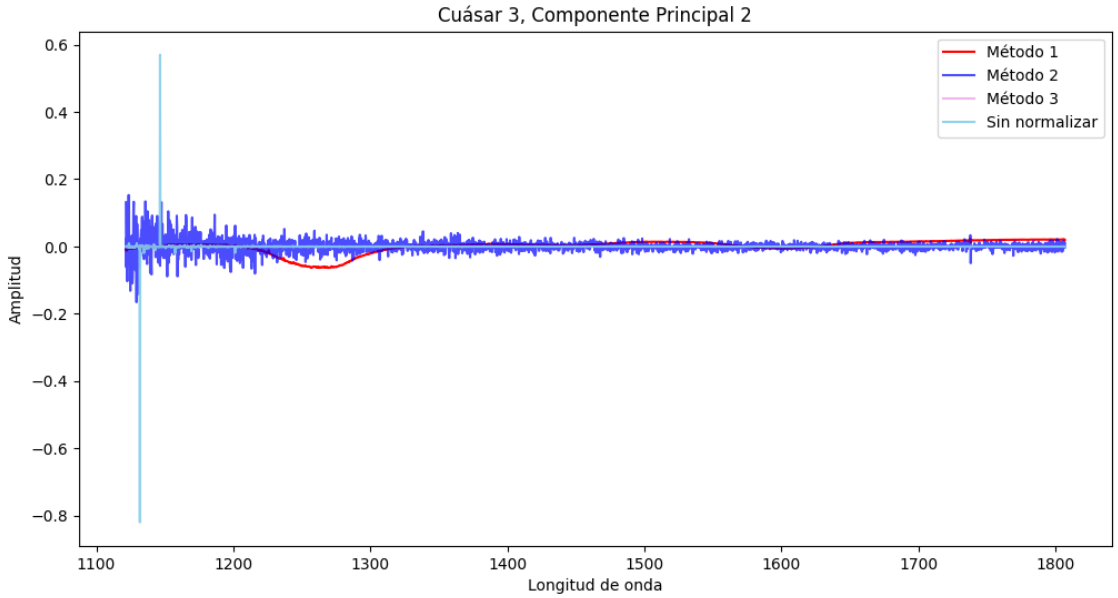



Cuásar 1, Componente Principal 2
Método 1
Método 2
Método 3
Sin normalizar
Amplitud
Longitud de onda
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1100
1200
1300
1400
1500
1600
1700
1800


Cuásar 2, Componente Principal 1
Método 1
Método 2
Método 3
Sin normalizar
Amplitud
Longitud de onda
1.0
0.8
0.6
0.4
0.2
0.0
1100
1200
1300
1400
1500
1600
1700
1800

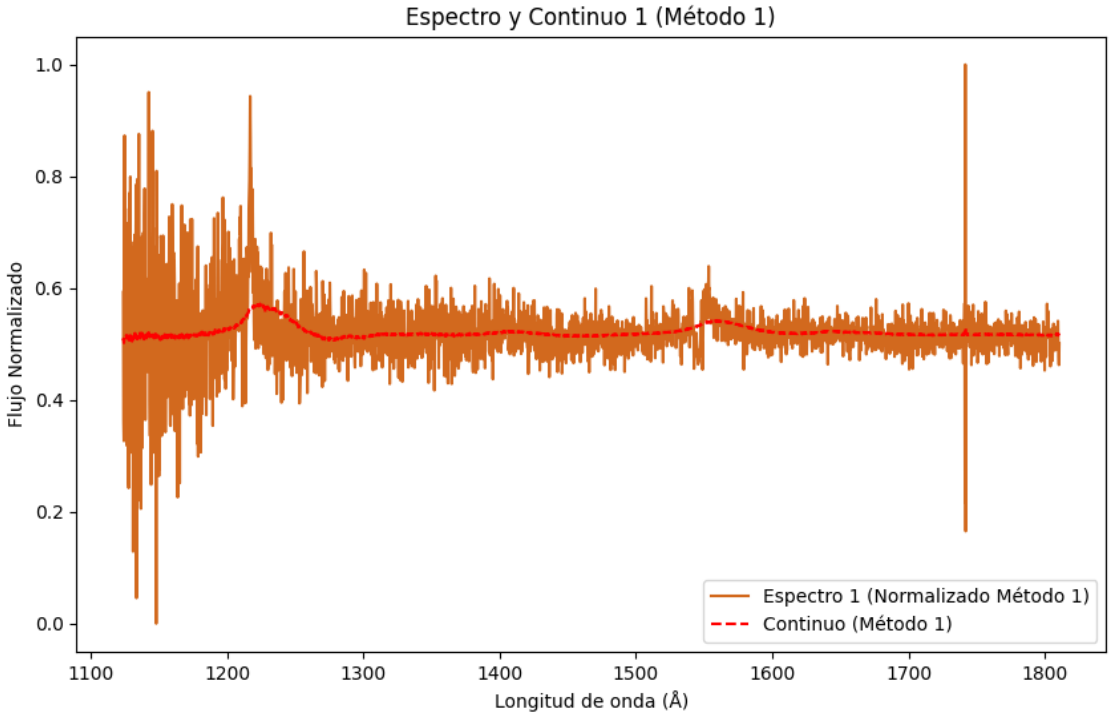

Cuásar 2, Componente Principal 2
Método 1
Método 2
Método 3
Sin normalizar
Amplitud
Longitud de onda
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
1100
1200
1300
1400
1500
1600
1700
1800

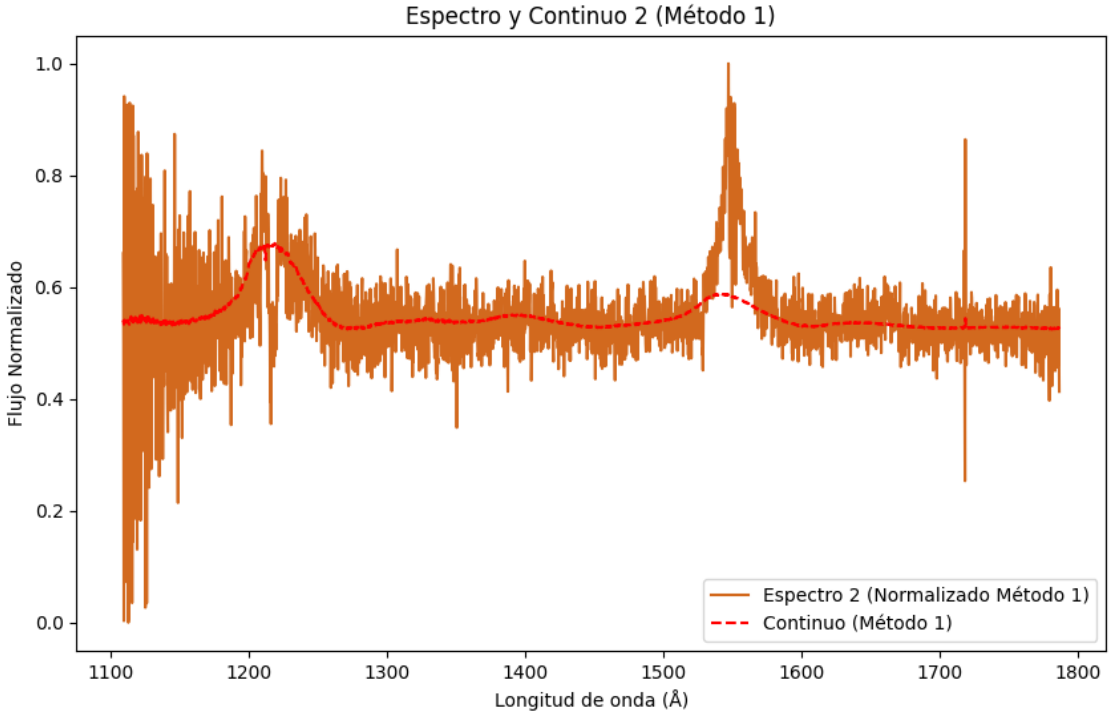

Cuásar 3, Componente Principal 1
Método 1
Método 2
Método 3
Sin normalizar
Amplitud
Longitud de onda
1.0
0.8
0.6
0.4
0.2
0.0
1100
1200
1300
1400
1500
1600
1700
1800

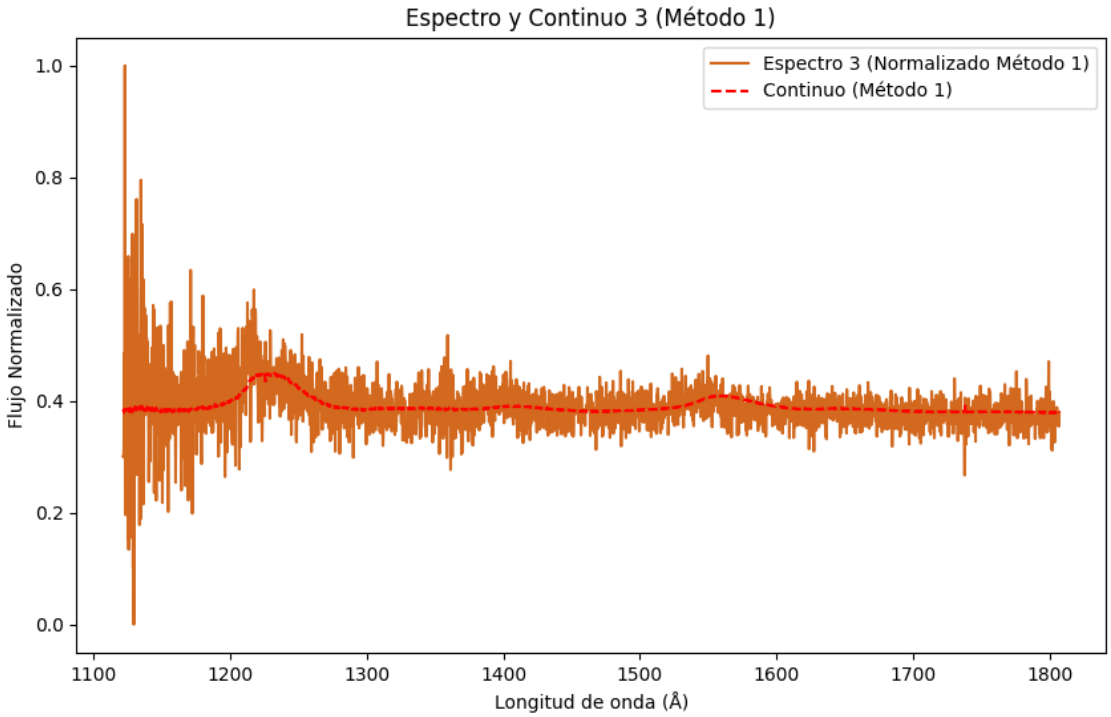


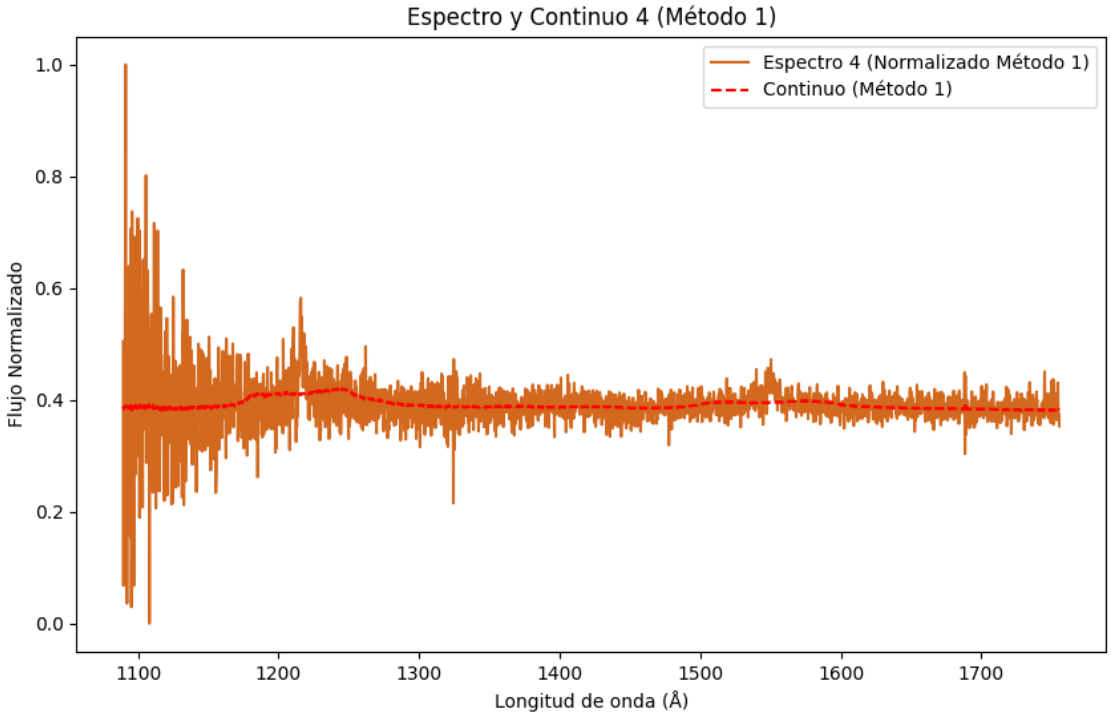
[ ]:
```
# Vamoa a rerear modeloos de continuo usando combinaciones lineales de componentes principales
def recrear_continuo(modelo, num_componentes, flujos):
    eigenvectores = modelo.eigvec[:num_componentes]
    proyecciones = np.dot(flujos, eigenvectores.T)
    continuo = np.dot(proyecciones, eigenvectores)
    return continuo

# Usamos el modeloo normalizado por el método 1 como ejemplo
num_componentes = 3
continuo1 = recrear_continuo(modelo1, num_componentes, flujos_normalizados1)
continuo2 = recrear_continuo(modelo2, num_componentes, flujos_normalizados2)
continuo3 = recrear_continuo(modelo3, num_componentes, flujos_normalizados3)

# Graficar algunos ejemplos de espectros normalizados junto con su modeloo de continuo
for i in range(5):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_mascara[i], flujos_normalizados1[i], label=f'Espectro {i+1} (Normalizado Método 1)', c='chocolate') #Estos colorcitos nada que ver
    plt.plot(longitudes_onda_mascara[i], continuo1[i], label=f'Continuo (Método 1)', c='red', linestyle='--')
    plt.xlabel('Longitud de onda (Å)')
    plt.ylabel('Flujo Normalizado')
    plt.title(f'Espectro y Continuo {i+1} (Método 1)')
    plt.legend()
```

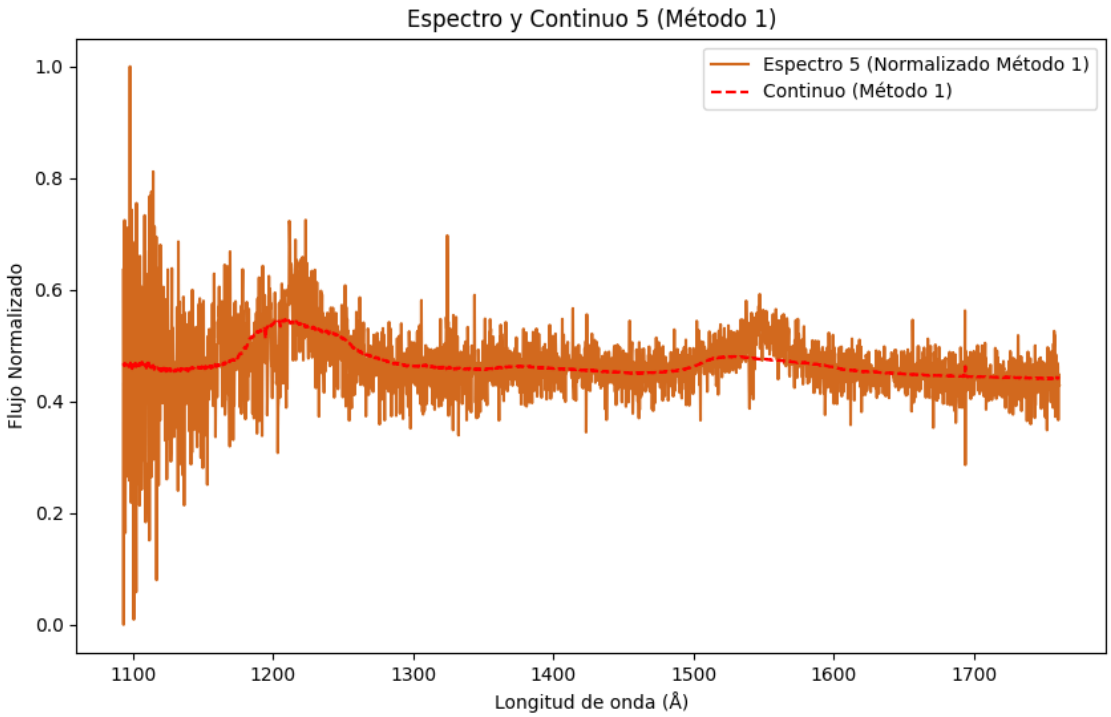
```
    plt.show()
```


Espectro y Continuo 1 (Método 1)
Flujo Normalizado
Longitud de onda (Å)
Espectro 1 (Normalizado Método 1)
Continuo (Método 1)


Espectro y Continuo 2 (Método 1)
Flujo Normalizado
Longitud de onda (Å)
Espectro 2 (Normalizado Método 1)
Continuo (Método 1)


Espectro y Continuo 3 (Método 1)
Espectro 3 (Normalizado Método 1)
Continuo (Método 1)
Flujo Normalizado
Longitud de onda (Å)
0.0
0.2
0.4
0.6
0.8
1.0
1100
1200
1300
1400
1500
1600
1700
1800

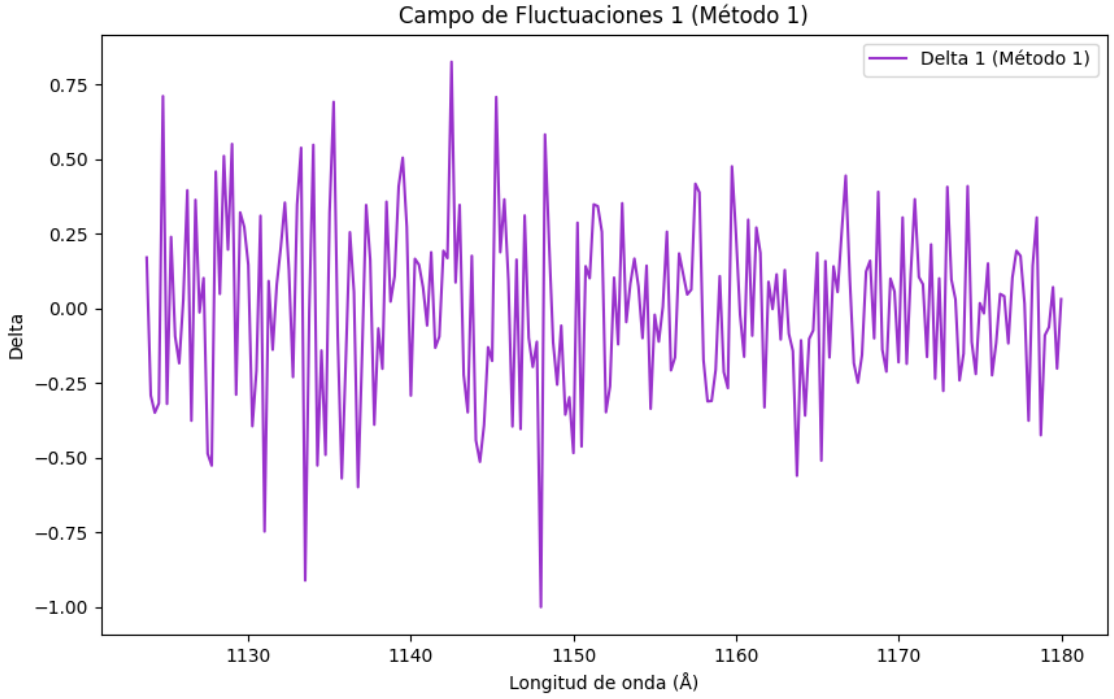

Espectro y Continuo 4 (Método 1)
Espectro 4 (Normalizado Método 1)
Continuo (Método 1)
Flujo Normalizado
Longitud de onda (Å)
0.0
0.2
0.4
0.6
0.8
1.0
1100
1200
1300
1400
1500
1600
1700

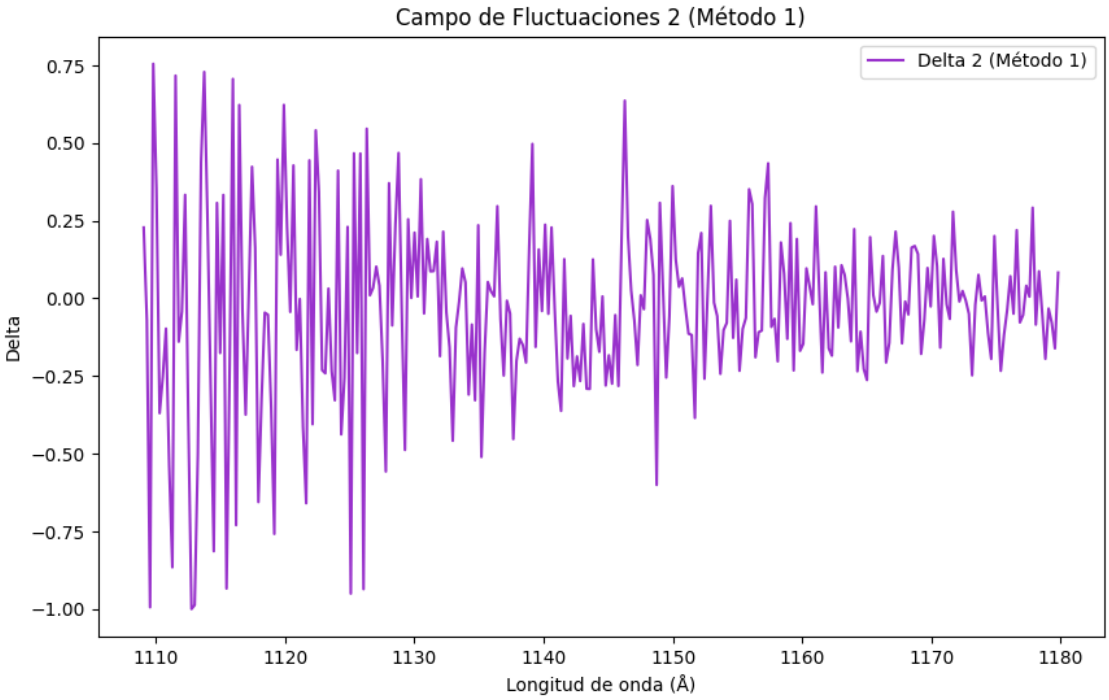


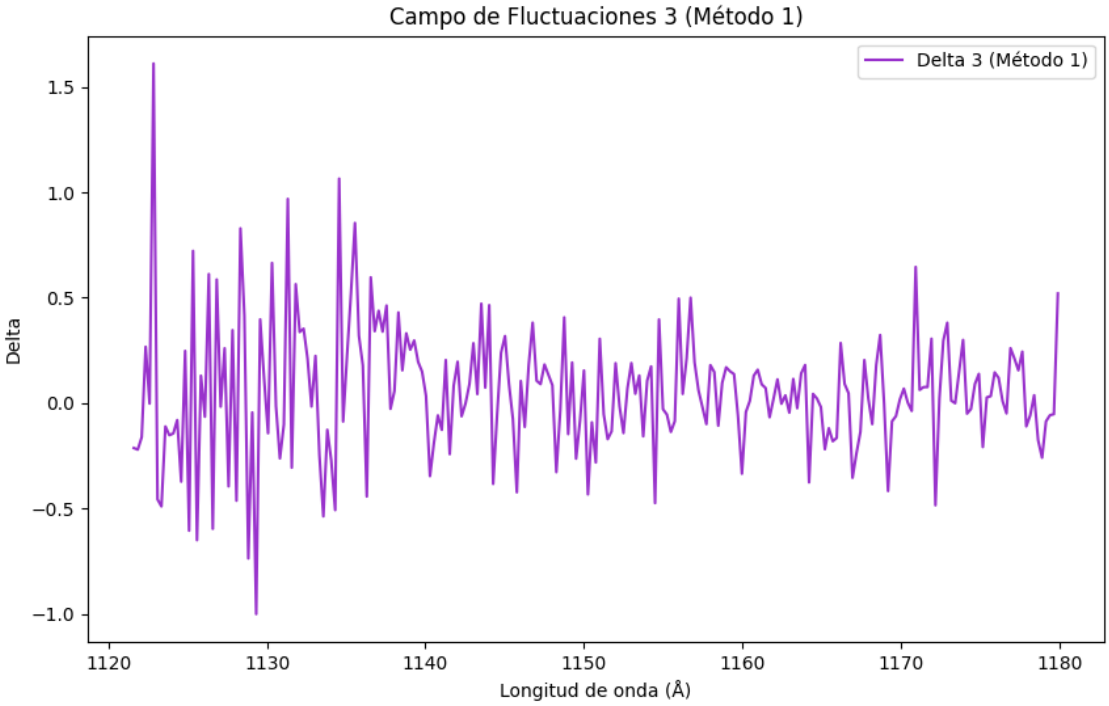
[ ]:
```
# Fórmula para Delta
def compute_delta(flujo, continuo):
    return flujo / continuo - 1
```

[ ]:
```
# Máscara para la región del bosque de Lyman Alpha (1050, 1180)
mask_forest = [(w >= 1050) & (w <= 1180) for w in longitudes_onda_mascara]

# Aplicar la máscara y calcular delta sin interpolar ni extrapolar
def aplicar_mascara_calcular_delta(flujos, continuos, masks,␣
 ↪longitudes_onda_mascara):
    deltas_forest = []
    longitudes_onda_validas = []

    for f, c, m, w in zip(flujos, continuos, masks, longitudes_onda_mascara):
        flujo_mascara = f[m]
        continuo_mascara = c[m]
        longitud_onda_mascara = w[m]

        delta = compute_delta(flujo_mascara, continuo_mascara)
        deltas_forest.append(delta)
        longitudes_onda_validas.append(longitud_onda_mascara)
```

```
    return deltas_forest, longitudes_onda_validas

# Aplicar la máscara y calcular delta
delta1_forest, longitudes_onda_validas1 =␣
 ↪aplicar_mascara_calcular_delta(flujos_normalizados1, continuo1, mask_forest,␣
 ↪longitudes_onda_mascara)
delta2_forest, longitudes_onda_validas2 =␣
 ↪aplicar_mascara_calcular_delta(flujos_normalizados2, continuo2, mask_forest,␣
 ↪longitudes_onda_mascara)
delta3_forest, longitudes_onda_validas3 =␣
 ↪aplicar_mascara_calcular_delta(flujos_normalizados3, continuo3, mask_forest,␣
 ↪longitudes_onda_mascara)

# Graficamos los cortes
for i in range(3):
    plt.figure(figsize=(10, 6))
    plt.plot(longitudes_onda_validas1[i], delta1_forest[i], label=f'Delta {i+1}␣
 ↪(Método 1)', c='darkorchid')
    plt.xlabel('Longitud de onda (Å)')
    plt.ylabel('Delta')
    plt.title(f'Campo de Fluctuaciones {i+1} (Método 1)')
    plt.legend()
    plt.show()
```




Campo de Fluctuaciones 2 (Método 1)
Delta 2 (Método 1)
Delta
Longitud de onda (Å)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
1110
1120
1130
1140
1150
1160
1170
1180


Campo de Fluctuaciones 3 (Método 1)
Delta 3 (Método 1)
Delta
Longitud de onda (Å)
1.5
1.0
0.5
0.0
−0.5
−1.0
1120
1130
1140
1150
1160
1170
1180

[ ]:
```
mean_deltas = [np.mean(delta) for delta in delta1_forest]

print(mean_deltas[1])
```

```
-0.03149491271682145
```

[ ]:
```
#Necesitamos que el promedio sea muy cercano a cero
promedio_delta1_forest = [delta - np.mean(delta) for delta in delta1_forest]
#delta2_forest_tras = delta2_forest - np.mean(delta2_forest, axis=1,␣
 ↪keepdims=True)
#delta3_forest_tras = delta3_forest - np.mean(delta3_forest, axis=1,␣
 ↪keepdims=True)
```

3.1 Histograma de posición

[ ]:
```
"""# Función para calcular histogramas
def calcular_histograma_lambda(deltas, longitudes_onda_sim, bins, delta_lambda):
    # Convertir listas de arreglos a matrices numpy
    deltas = np.array([np.pad(d, (0, max(map(len, deltas)) - len(d)),␣
 ↪'constant') for d in deltas])
    longitudes_onda_sim = np.array([np.pad(w, (0, max(map(len,␣
 ↪longitudes_onda_sim)) - len(w)), 'constant') for w in longitudes_onda_sim])

    # Arrays de ceros
    num_qso, num_puntos = deltas.shape  # Número de qso y número de puntos en␣
 ↪cada uno
    hist_contador = np.zeros((num_qso, len(bins) - 1))
    hist_contribucion = np.zeros((num_qso, len(bins) - 1))
    hist_div = np.zeros((num_qso, len(bins) - 1))

    for q in range(num_qso):  # Pasamos por cada quásar
        for i in range(num_puntos):  # Pasamos por cada punto
            for j in range(i + 1, num_puntos):  # +1 para evitar la␣
 ↪autocorrelación
                dist = longitudes_onda_sim[q, j] - longitudes_onda_sim[q, i]  #␣
 ↪Distancia en longitudes de onda
                bin_idx = int(dist / delta_lambda)  # Índice del bin␣
 ↪correspondiente
                if bin_idx < len(hist_contador[q]):  # Verificamos que el␣
 ↪índice esté en el rango
                    hist_contador[q, bin_idx] += 1  # Contamos la ocurrencia
                    hist_contribucion[q, bin_idx] += deltas[q, i] * deltas[q,␣
 ↪j]  # Contribución de las deltas

        with np.errstate(divide='ignore', invalid='ignore'):
```

```
            hist_div[q] = np.where(hist_contador[q] != 0, hist_contribucion[q] /
 ↪ hist_contador[q], 0)

    return hist_contador, hist_contribucion, hist_div  # Regresamos los␣
 ↪histogramas

# Convertir listas de arreglos a matrices numpy
promedio_delta1_forest = [np.array(delta) for delta in promedio_delta1_forest]
longitudes_onda_mascara_sim = [np.array(w) for w in longitudes_onda_mascara]

# Calcular histogramas para promedio_delta1_forest usando las longitudes de␣
 ↪onda
hist_contador, hist_contribucion, hist_div =␣
 ↪calcular_histograma_lambda(promedio_delta1_forest,␣
 ↪longitudes_onda_mascara_sim, bins, delta_lambda)"""
```

[ ]:
```
"# Función para calcular histogramas\ndef calcular_histograma_lambda(deltas,
longitudes_onda_sim, bins, delta_lambda):\n    # Convertir listas de arreglos a
matrices numpy\n    deltas = np.array([np.pad(d, (0, max(map(len, deltas)) -
len(d)), 'constant') for d in deltas])\n    longitudes_onda_sim =
np.array([np.pad(w, (0, max(map(len, longitudes_onda_sim)) - len(w)),
'constant') for w in longitudes_onda_sim])\n\n    # Arrays de ceros\n
num_qso, num_puntos = deltas.shape  # Número de qso y número de puntos en cada
uno\n    hist_contador = np.zeros((num_qso, len(bins) - 1))\n
hist_contribucion = np.zeros((num_qso, len(bins) - 1))\n    hist_div =
np.zeros((num_qso, len(bins) - 1))\n\n    for q in range(num_qso):  # Pasamos
por cada quásar\n        for i in range(num_puntos):  # Pasamos por cada punto\n
for j in range(i + 1, num_puntos):  # +1 para evitar la autocorrelación\n
dist = longitudes_onda_sim[q, j] - longitudes_onda_sim[q, i]  # Distancia en
longitudes de onda\n                bin_idx = int(dist / delta_lambda)  # Índice
del bin correspondiente\n                if bin_idx < len(hist_contador[q]):  #
Verificamos que el índice esté en el rango\n                    hist_contador[q,
bin_idx] += 1  # Contamos la ocurrencia\n
hist_contribucion[q, bin_idx] += deltas[q, i] * deltas[q, j]  # Contribución de
las deltas\n\n        with np.errstate(divide='ignore', invalid='ignore'):\n
hist_div[q] = np.where(hist_contador[q] != 0, hist_contribucion[q] /
hist_contador[q], 0)\n\n    return hist_contador, hist_contribucion, hist_div  #
Regresamos los histogramas\n\n# Convertir listas de arreglos a matrices
numpy\npromedio_delta1_forest = [np.array(delta) for delta in
promedio_delta1_forest]\nlongitudes_onda_mascara_sim = [np.array(w) for w in
longitudes_onda_mascara]\n\n# Calcular histogramas para promedio_delta1_forest
usando las longitudes de onda\nhist_contador, hist_contribucion, hist_div =
calcular_histograma_lambda(promedio_delta1_forest, longitudes_onda_mascara_sim,
bins, delta_lambda)"
```

[ ]:
```
"""# Verificar los resultados
print('Contador:', hist_contador[0])
print('CoNTRIBUCIÓN:',hist_contribucion[0])
```

```
print('División:', hist_div[0])"""
```

[ ]:
```
"# Verificar los resultados\nprint('Contador:', hist_contador[0])\nprint('CoNTRIBUCIÓN:',hist_contribucion[0])\nprint('División:', hist_div[0])"
```

[ ]:
```
"""divi = hist_contribucion[0] / hist_contador[0]
divi[0]"""
```

[ ]:
```
'divi = hist_contribucion[0] / hist_contador[0]\ndivi[0]'
```

[ ]:
```
"""# Graficar los histogramas para los primeros 3 quásares en gráficas separadas
fig, axes = plt.subplots(3, 1, figsize=(10, 18))
for i in range(3):
    axes[i].plot(bins[:-1], hist_div[i], label=f'Quásar {i+1}', c='darkorchid')
    axes[i].set_xlabel('Distancia (Índice de bin)')
    axes[i].set_ylabel('Contribución')
    axes[i].set_title(f'Correlación por Armstrongs {i+1}')
    axes[i].legend()

plt.tight_layout()
plt.show()"""
```

[ ]:
```
"# Graficar los histogramas para los primeros 3 quásares en gráficas separadas\nfig, axes = plt.subplots(3, 1, figsize=(10, 18))\nfor i in range(3):\n    axes[i].plot(bins[:-1], hist_div[i], label=f'Quásar {i+1}', c='darkorchid')\n    axes[i].set_xlabel('Distancia (Índice de bin)')\n    axes[i].set_ylabel('Contribución')\n    axes[i].set_title(f'Correlación por Armstrongs {i+1}')\n    axes[i].legend()\n\nplt.tight_layout()\nplt.show()"
```

[ ]:
```
# Definimos el tamaño del bin
delta_lambda = 5

# Función para calcular histogramas en bins de posición
def calcular_histogramas_posicion(deltas, delta_lambda):
    num_qso = len(deltas)

    # Obtención de las distancias máximas posibles para cada conjunto de datos
    max_distancias = [len(d) // delta_lambda for d in deltas]

    # Arrays de ceros bidimensionales
    hist_contador_pos = [np.zeros(max_dist) for max_dist in max_distancias]
    hist_contribucion_pos = [np.zeros(max_dist) for max_dist in max_distancias]
    hist_div_pos = [np.zeros(max_dist) for max_dist in max_distancias]

    for q in range(num_qso):
        deltas_qso = deltas[q]
        num_puntos = len(deltas_qso)
        max_dist = max_distancias[q]
```

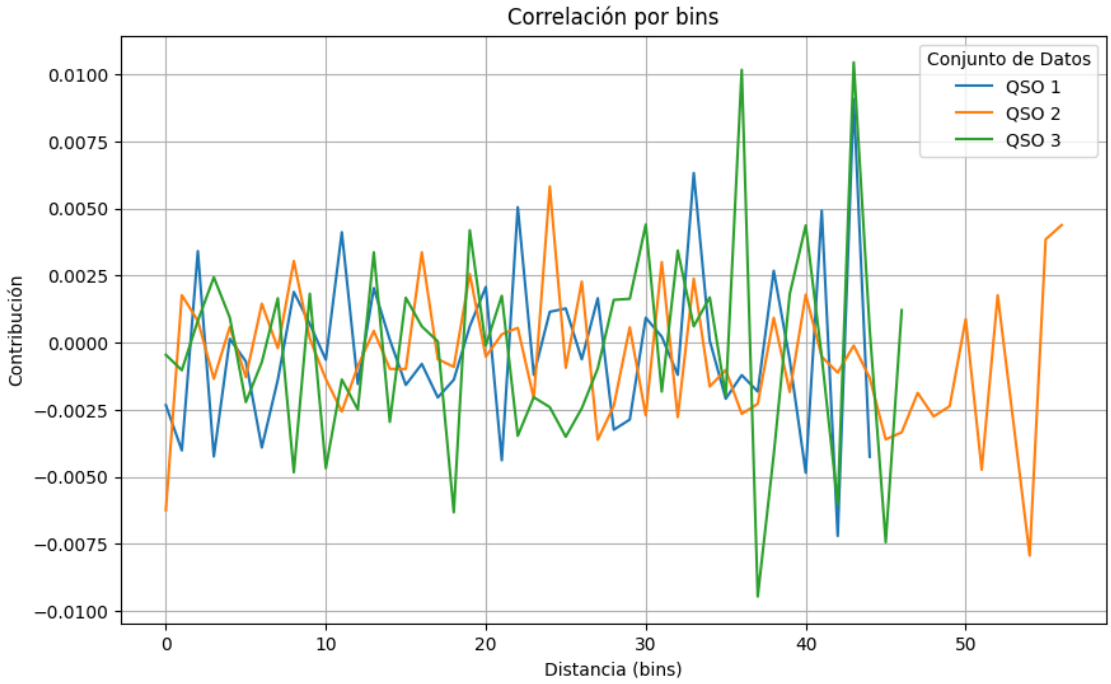
```
        for i in range(num_puntos):
            for j in range(i + 1, num_puntos):
                dist = j - i
                bin_idx = int(dist / delta_lambda)
                if bin_idx < max_dist:
                    hist_contador_pos[q][bin_idx] += 1
                    hist_contribucion_pos[q][bin_idx] += deltas_qso[i] *␣
 ↪deltas_qso[j]

        with np.errstate(divide='ignore', invalid='ignore'):
            hist_div_pos[q] = np.where(hist_contador_pos[q] != 0,␣
 ↪hist_contribucion_pos[q] / hist_contador_pos[q], 0)

    return hist_contador_pos, hist_contribucion_pos, hist_div_pos

# Convertimos las listas de arreglos en arreglos numpy
promedio_delta1_forest = [np.array(delta) for delta in promedio_delta1_forest]

# Calcular histogramas para promedio_delta1_forest en bins de posición
hist_contador_pos, hist_contribucion_pos, hist_div_pos =␣
 ↪calcular_histogramas_posicion(promedio_delta1_forest, delta_lambda)
```

[ ]:
```
# Verificar los resultados
print(hist_contador_pos[0])
print(hist_contribucion_pos[0])
print(hist_div_pos[0])
```

```
[ 894. 1095. 1070. 1045. 1020.  995.  970.  945.  920.  895.  870.  845.
  820.  795.  770.  745.  720.  695.  670.  645.  620.  595.  570.  545.
  520.  495.  470.  445.  420.  395.  370.  345.  320.  295.  270.  245.
  220.  195.  170.  145.  120.   95.   70.   45.   20.]
[-2.07949327 -4.39626013  3.6559506  -4.42860491  0.14818687 -0.6916883
 -3.78840663 -1.30112916  1.74501633  0.6226973  -0.55267744  3.48168447
 -1.25809782  1.61984057  0.08607256 -1.17191899 -0.5687878  -1.41799078
 -0.92456261  0.39805592  1.28785601 -2.60721417  2.8820765  -0.65370771
  0.59981032  0.63299185 -0.28979527  0.73931718 -1.36229376 -1.13161608
  0.34555343  0.0776936  -0.38279872  1.86720865  0.02189032 -0.51127681
 -0.26503424 -0.35516456  0.45612198 -0.09241911 -0.5815276   0.4681103
 -0.50417337  0.40876147 -0.08504162]
[-2.32605511e-03 -4.01484944e-03  3.41677626e-03 -4.23789943e-03
  1.45281248e-04 -6.95164124e-04 -3.90557384e-03 -1.37685625e-03
  1.89675688e-03  6.95751174e-04 -6.35261430e-04  4.12033665e-03
 -1.53426564e-03  2.03753531e-03  1.11782542e-04 -1.57304563e-03
 -7.89983055e-04 -2.04027450e-03 -1.37994420e-03  6.17140967e-04
  2.07718711e-03 -4.38187255e-03  5.05627456e-03 -1.19946368e-03
  1.15348139e-03  1.27877142e-03 -6.16585679e-04  1.66138692e-03
 -3.24355657e-03 -2.86485084e-03  9.33928181e-04  2.25198831e-04
 -1.19624601e-03  6.32952083e-03  8.10752677e-05 -2.08684411e-03
```

```
-1.20470107e-03 -1.82135674e-03  2.68307044e-03 -6.37373173e-04
-4.84606336e-03  4.92747683e-03 -7.20247665e-03  9.08358829e-03
-4.25208122e-03]
```

[ ]:
```
plt.figure(figsize=(10, 6))

for q in range(len(hist_div_pos[0:3])):
    # Graficar la contribución para cada QSO
    plt.plot(np.arange(len(hist_div_pos[q])), hist_div_pos[q], label=f'QSO␣
 ↪{q+1}')

plt.xlabel('Distancia (bins)')
plt.ylabel('Contribución')
plt.title('Correlación por bins')
plt.legend(title='Conjunto de Datos')
plt.grid(True)
plt.show()
```

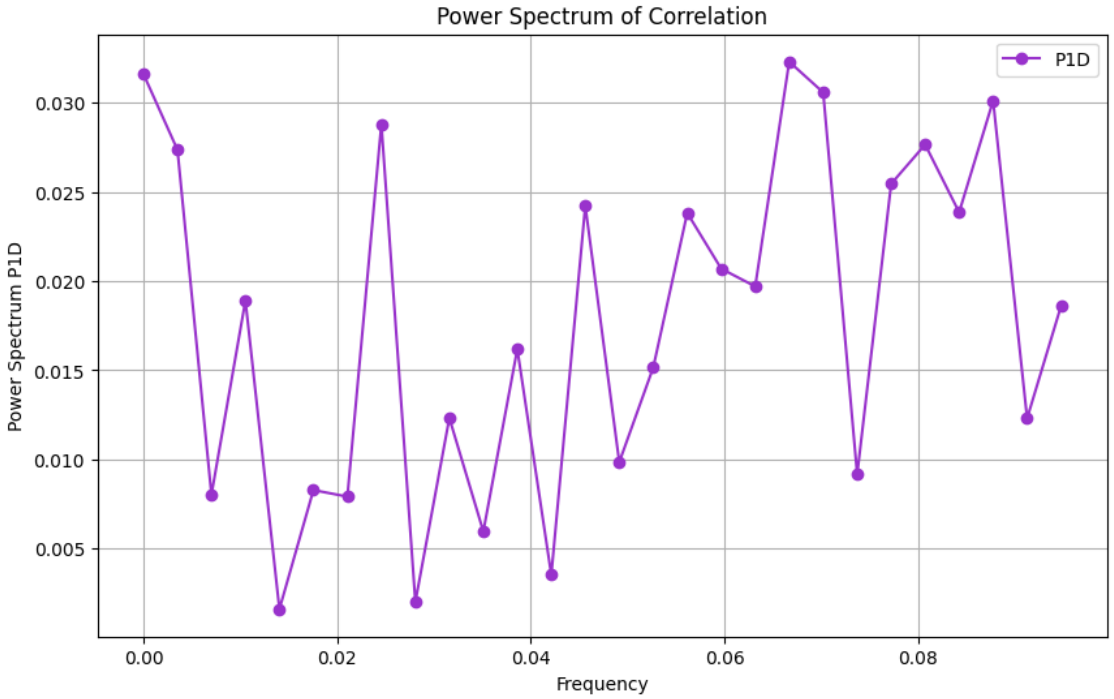


[ ]:
```
"""bin_size = 5  ### Tamaño de bins usado para la correlación
bin_edges = np.arange(len(hist_div[:,1])) * bin_size

P1D = np.fft.fft(hist_div[:,1])
freq = np.fft.fftfreq(len(hist_div[:,1]), d=bin_size)
```

```
# Para solo tomar las frecuencias positivas y sus valores correspondientes del␣
 ↪espectro
positive_freq = freq[:len(freq)//2]
P1D_positive = np.abs(P1D[:len(P1D)//2])

plt.figure(figsize=(10, 6))
plt.plot(positive_freq, P1D_positive, marker='o', linestyle='-', label='P1D',␣
 ↪c='darkorchid')
plt.xlabel('Frequency')
plt.ylabel('Power Spectrum P1D')
plt.legend()
plt.show()"""
```

[ ]:
```
"bin_size = 5  ### Tamaño de bins usado para la correlación\nbin_edges =
np.arange(len(hist_div[:,1])) * bin_size\n\nP1D =
np.fft.fft(hist_div[:,1])\nfreq = np.fft.fftfreq(len(hist_div[:,1]),
d=bin_size)\n\n# Para solo tomar las frecuencias positivas y sus valores
correspondientes del espectro\npositive_freq = freq[:len(freq)//2]\nP1D_positive
= np.abs(P1D[:len(P1D)//2])\n\nplt.figure(figsize=(10,
6))\nplt.plot(positive_freq, P1D_positive, marker='o', linestyle='-',
label='P1D', c='darkorchid')\nplt.xlabel('Frequency')\nplt.ylabel('Power
Spectrum P1D')\nplt.legend()\nplt.show()"
```

[ ]:
```
bin_size = 5  # Tamaño de bins usado para la correlación
num_bins = len(hist_div_pos[0])  # Asumimos que todos los conjuntos tienen el␣
 ↪mismo número de bins
bin_edges = np.arange(num_bins) * bin_size

hist_div_pos_selected = hist_div_pos[1]

P1D = np.fft.fft(hist_div_pos_selected)
freq = np.fft.fftfreq(len(hist_div_pos_selected), d=bin_size)

# Para solo tomar las frecuencias positivas y sus valores correspondientes del␣
 ↪espectro
positive_freq = freq[:len(freq)//2]
P1D_positive = np.abs(P1D[:len(P1D)//2])

plt.figure(figsize=(10, 6))
plt.plot(positive_freq, P1D_positive, marker='o', linestyle='-', label='P1D',␣
 ↪c='darkorchid')
plt.xlabel('Frequency')
plt.ylabel('Power Spectrum P1D')
plt.title('Power Spectrum of Correlation')
plt.legend()
plt.grid(True)
plt.show()
```


Power Spectrum of Correlation
P1D
Power Spectrum P1D
0.030
0.025
0.020
0.015
0.010
0.005
0.00
0.02
0.04
0.06
0.08
Frequency

[ ]: