modelo_gmm

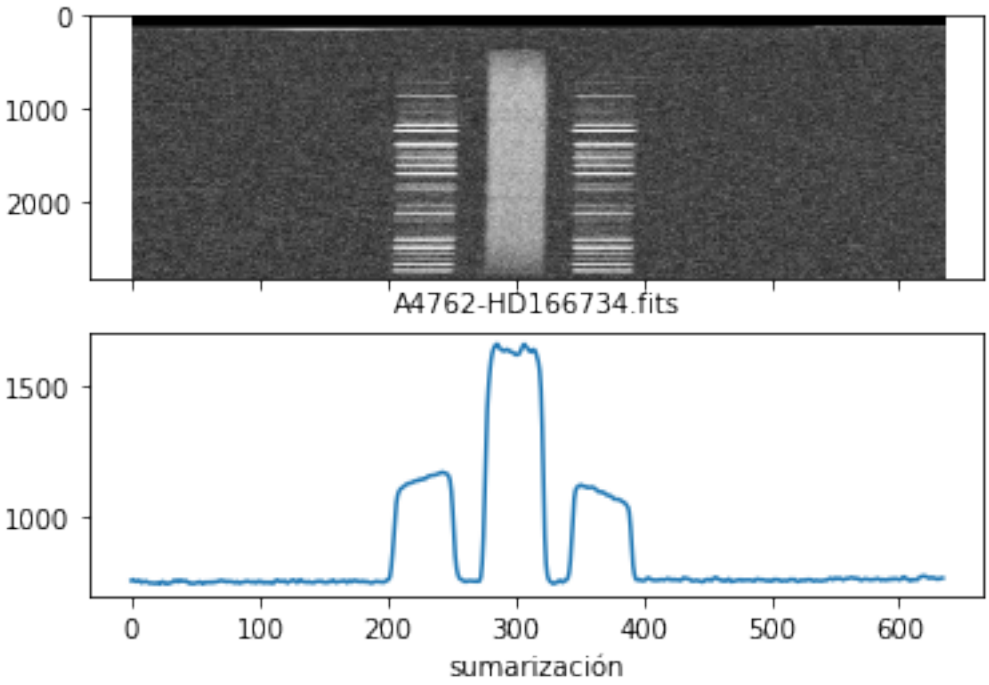
March 15, 2021

Se cargan las librerias necesarias y se extra los datos de la imagen. Este presenta 1 espectro de ciencia sin ningún ruido particular.

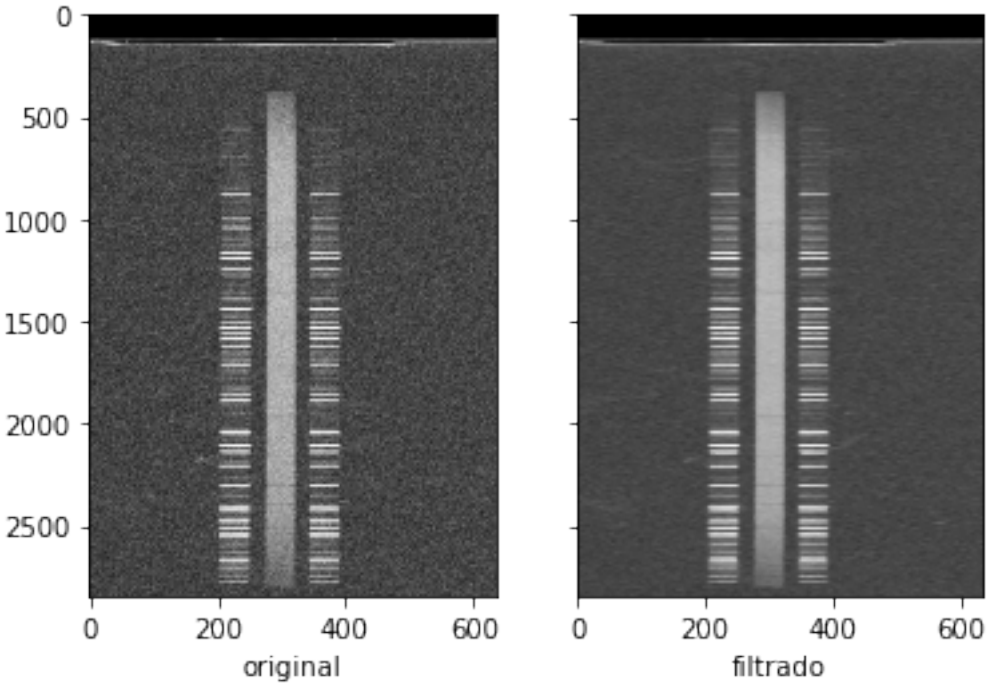
[57]:
```
#Tratamiento de filesystem
#import os
#import sys
#sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
#%load_ext autoreload
#%autoreload 2

import Recursos
%autoreload 1
#import Recursos
#Tratamientos de datos
import numpy as np
#Gráficos
import matplotlib.pyplot as plt

import scipy.stats as stats
#Modelos
from sklearn.mixture import GaussianMixture


NOMBRE_IMG1 = 'A4762-HD166734.fits' #posee un espectro celeste

datos, clusters = Recursos.getInfo(NOMBRE_IMG1)
datos = Recursos.normalize_MinMax_2d(datos)
Y = np.sum(datos, axis=0) #sumariza los pixels en el eje x
x = np.arange(Y.size)
```

[79]:
```
def graficarInfo(img,x,Y):
    fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, sharex=True)
    ax1.imshow(img, cmap='gray')
    ax1.set_aspect('auto')
    ax2.plot(x,Y)
    ax1.set_xlabel(NOMBRE_IMG1)
    ax2.set_xlabel("sumarización")
    plt.show()
```

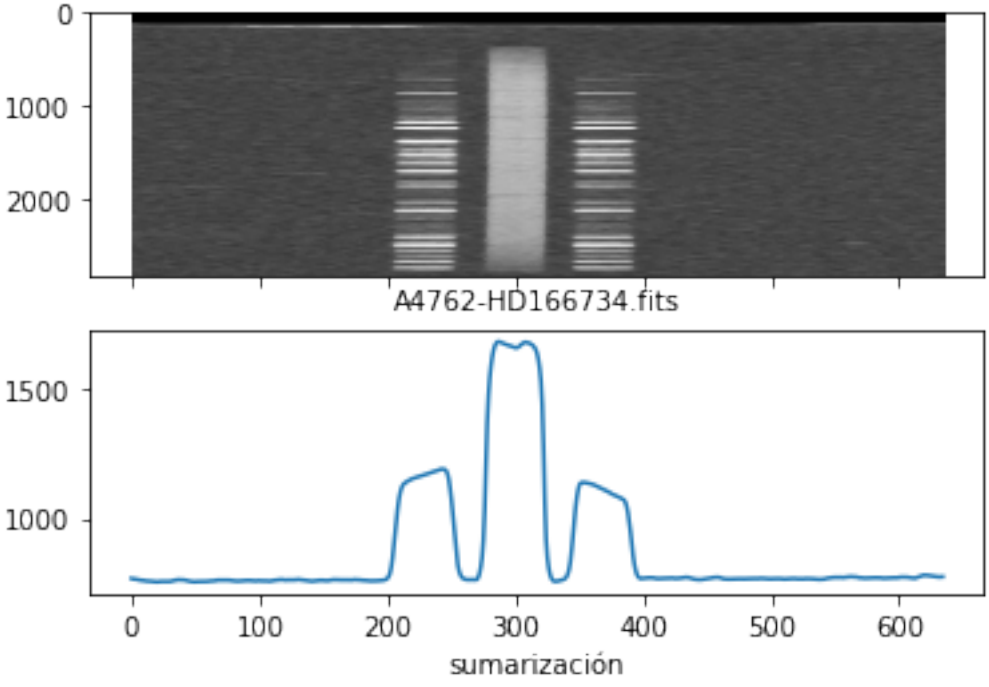
[80]:
```
graficarInfo(datos,x,Y)
```



[95]:
```
import cv2
import scipy.ndimage as ndimage

b = ndimage.filters.median_filter(datos,size=8)
graficar_imgs(datos,b)
```



[96]:
```
datos_fil = Recursos.normalize_MinMax_2d(b)
Y_fil = np.sum(datos_fil, axis=0) #sumariza los pixels en el eje x
graficarInfo(datos_fil,x,Y_fil)
```

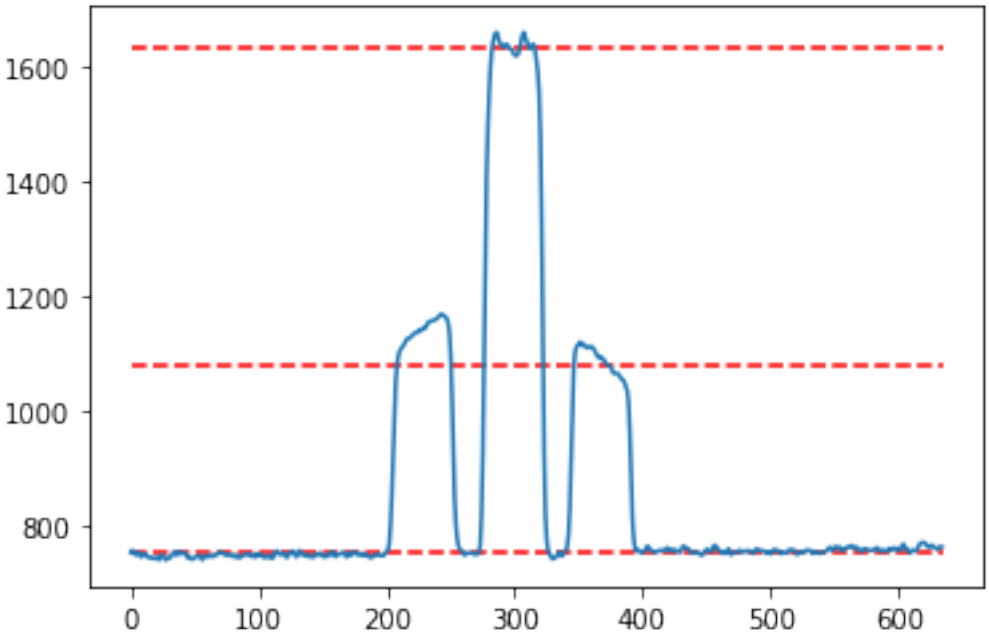


[97]:
```
def modelo_gmm(datos, clusters, nro_iters):
    #Se configura el modelo
    gmm = GaussianMixture(
            n_components=clusters,   #agregar cantidad total: *3
            covariance_type='full',
            n_init= nro_iters)
    #datosNormImg.shape = (datosNormImg.shape[0],1)

    #Se estima el modelo
    gmm.fit(datos)
    #Se predice el cluster para cada punto de la imagen
    #etiquetasClusters = gmm.predict(datosNormImg)
    print("Datos del modelo")
    print("centroides")
    print(gmm.means_)
    print("%")
    print(gmm.weights_)
    print("std")
    print(np.sqrt(gmm.covariances_))
    return gmm
```

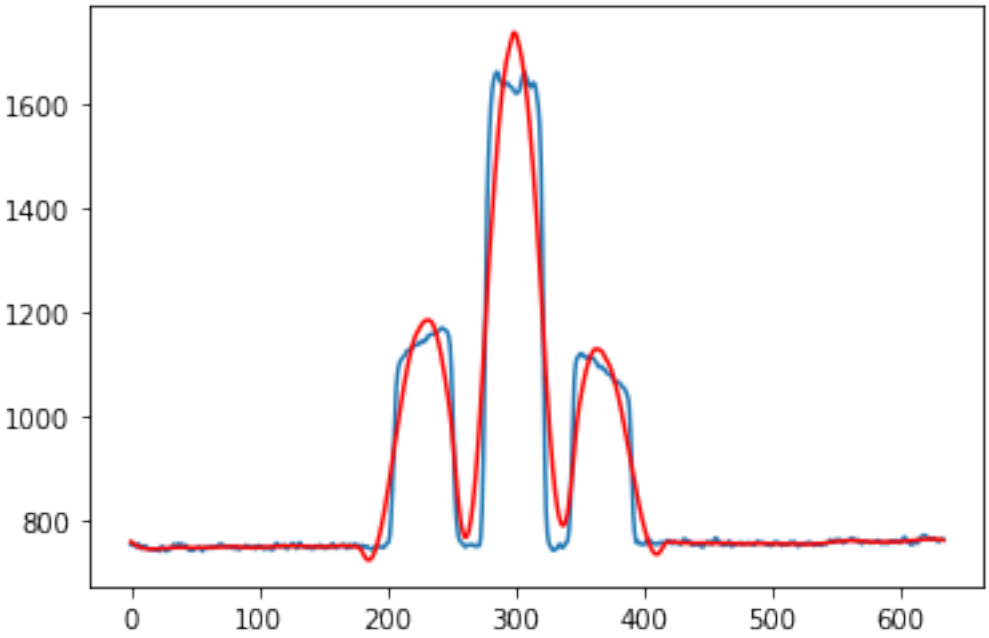
[99]:
```
print(Y.shape)
modelo = modelo_gmm(Y.reshape(-1,1), clusters*3, 500)
plt.plot(x,Y)
plt.hlines(modelo.means_,0, x.size,colors='r',linestyles='dashed')
plt.show
```

```
(635,)
Datos del modelo
centroides
[[1080.56840729]
 [ 751.59430906]
 [1632.6187355 ]]
%
[0.18360275 0.75377683 0.06262043]
std
[[[126.3131227 ]]

 [[  5.97013817]]

 [[ 22.0756855 ]]]
```

[99]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[100]:
```
import scipy.signal as signal
#x = np.linspace(0,2*np.pi,100)
#y = np.sin(x) + np.random.random(100) * 0.2
yhat = scipy.signal.savgol_filter(Y, 51, 3) # window size 51, polynomial order 3

plt.plot(x,Y)
plt.plot(x,yhat, color='red')
plt.show()
```

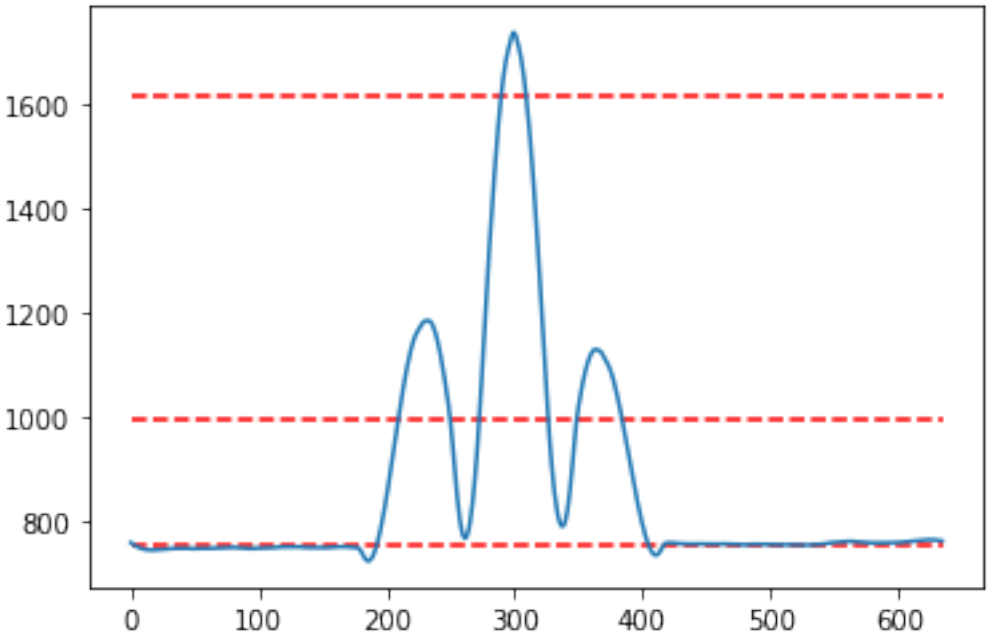


[102]:
```
modelo = modelo_gmm(yhat.reshape(-1,1), clusters*3, 1000)
plt.plot(x,yhat)
plt.hlines(modelo.means_,0, x.size,colors='r',linestyles='dashed')
plt.show

"""Datos del modelo con 500 iter.
centroides
[[ 751.34297356]
 [ 992.37093311]
 [1618.12685676]]
%
[0.65593151 0.29152158 0.05254691]
std
[[[  5.7417794 ]]

 [[161.77982787]]

 [[102.02408157]]]
"""
```

```
Datos del modelo
centroides
[[ 751.34297356]
 [ 992.37093311]
 [1618.12685676]]
%
```

```
[0.65593151 0.29152158 0.05254691]
std
[[[  5.7417794 ]]

 [[161.77982787]]

 [[102.02408157]]]
```

[102]:
```
'Datos del modelo con 500 iter.\ncentroides\n[[ 751.34297356]\n [
992.37093311]\n [1618.12685676]]\n%\n[0.65593151 0.29152158
0.05254691]\nstd\n[[[  5.7417794 ]]\n\n [[161.77982787]]\n\n
[[102.02408157]]]\n'
```

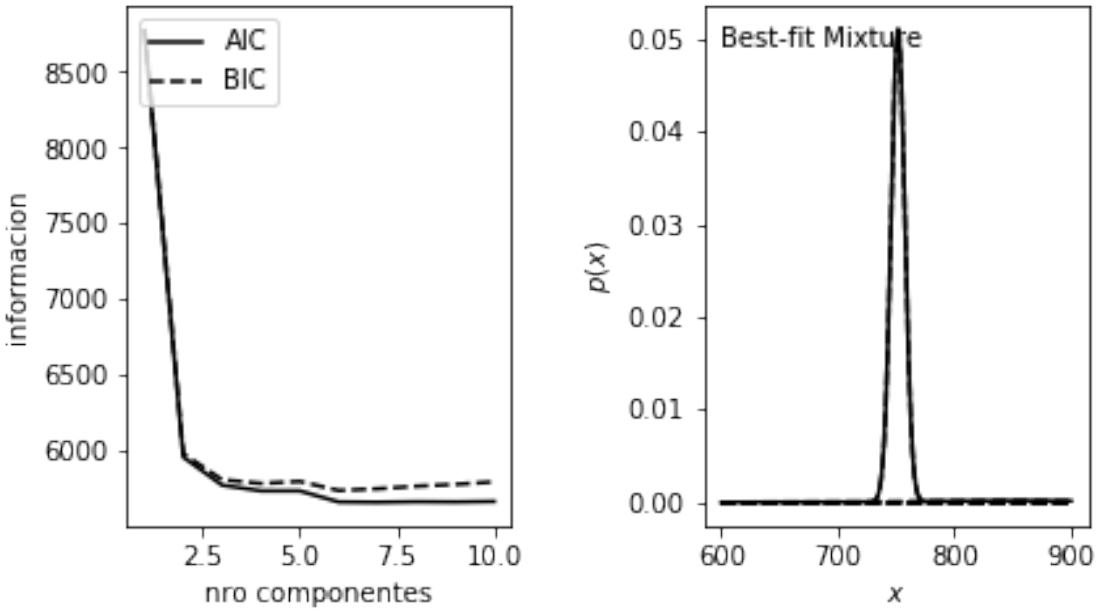


Siendo un modelo probabilístico se pueden aplicar métricas como el Akaike information criterion (AIC) o Bayesian information criterion (BIC) para identificar cómo se van ajustando los datos observados al modelo. Cabe aclarar que en ambas métricas, cuanto más bajo sea el valor es mejor. Por lo que se prueba para revisar el modelo con un rango de 1 a 10 como valores del nro de componentes.

[108]:
```
#se entrena modelos con 1-10 componentes
Y_copy = Y.copy()
Y_copy = Y_copy.reshape(-1,1)
N = np.arange(1, 11)
models = [None for i in range(len(N))]

for i in range(len(N)):
    models[i] = GaussianMixture(N[i]).fit(Y_copy)
```

```
#se calcula AIC y BIC
AIC = [m.aic(Y_copy) for m in models]
BIC = [m.bic(Y_copy) for m in models]
```

[118]:
```
#se grafica los resultados
#fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, sharex=True)

fig = plt.figure()
fig.subplots_adjust(left=0.12, right=0.97,
                    bottom=0.21, top=0.9, wspace=0.5)
#grafico 1
ax1 = fig.add_subplot(121)
ax1.plot(N, AIC, '-k', label='AIC')
ax1.plot(N, BIC, '--k', label='BIC')
ax1.set_xlabel('nro componentes')
ax1.set_ylabel('informacion')
ax1.legend(loc=2)

#grafico 2
# plot 1: data + best-fit mixture
ax2 = fig.add_subplot(122)
M_best = models[np.argmin(AIC)]

x_best = np.linspace(600,900, 1000)
logprob = M_best.score_samples(x_best.reshape(-1, 1))
responsibilities = M_best.predict_proba(x_best.reshape(-1, 1))
pdf = np.exp(logprob)
pdf_individual = responsibilities * pdf[:, np.newaxis]

#ax2.hist(Y_copy,bins=100, density=True, histtype='stepfilled', alpha=0.4)
#ax2.plot(x,Y_copy,color='r')
ax2.plot(x_best, pdf, '-k')
ax2.plot(x_best, pdf_individual, '--k')
ax2.text(0.04, 0.96, "Best-fit Mixture",
        ha='left', va='top', transform=ax2.transAxes)
ax2.set_xlabel('$x$')
ax2.set_ylabel('$p(x)$')

plt.show()
```



[119]:
```
print(M_best)
```

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=7, n_init=1, precisions_init=None,
                random_state=None, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
```