2hr_3s_DBSCAN

August 28, 2025

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
```

[3]:
```
import os
```

[4]:
```
import pandas as pd
```

[5]:
```
import psrchive
```

[6]:
```
from scipy.optimize import curve_fit
from scipy.stats import skew, kurtosis
```

[7]:
```
from scipy.stats import normaltest
```

[8]:
```
import corner
```

[9]:
```
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.io.output import output_notebook
import base64
from io import BytesIO
```

[10]:
```
from bokeh.palettes import Viridis256
from matplotlib.colors import Normalize, to_hex
from matplotlib import cm
```

1 Preparing 2hr psr data

[11]:
```
os.chdir("/idia/projects/pulsar-timing/users/athambiran/singlepulses/2hr_obs")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[11], line 1
----> 1␣
  ↪os.chdir("/idia/projects/pulsar-timing/users/athambiran/singlepulses/2hr_obs"
```

```
FileNotFoundError: [Errno 2] No such file or directory: '/idia/projects/
↪pulsar-timing/users/athambiran/singlepulses/2hr_obs'
```

[ ]:
```
ls
```

[ ]:
```
arch = psrchive.Archive_load("2hr.t3.f4.pazi")
```

[ ]:
```
arch.remove_baseline()
arch.dedisperse()
arch.convert_state("Stokes")
arch.get_state()
```

[ ]:
```
arch.fscrunch()
arch.bscrunch(4)
```

[ ]:
```
data_for_profiles = arch.get_data().squeeze()
np.shape(data_for_profiles) ## time, Stokes, bins (1024/4)
```

[ ]:
```
obs_duration = arch.integration_length()
num_chan = arch.get_nchan()
num_subint = arch.get_nsubint()
num_bins = arch.get_nbin()
num_pol = arch.get_npol()
pol_state = arch.get_state()

print(f'And has the following characteristics:\n')
print(f'Observation duration of \t\t\t {obs_duration:.2f} seconds')
print(f'Number of frequency channels\t\t\t {num_chan}')
print(f'Number of polarisations \t\t\t {num_pol}')
print(f'In polarisation state \t\t\t\t {pol_state}')
print(f'Number of subintegrations (time blocks)\t\t {num_subint}')
print(f'Each subintegration (time block) is made up of   {num_bins} data bins')
```

[ ]:
```
### Make one plot of subint X to see if we want to center peaks
subint_num = 2

plt.figure(figsize=(7,4))
plt.plot(data_for_profiles[subint_num,0,:], label="Profile", alpha = 0.3)
plt.hlines(0, 0, 256, color = 'black', alpha = 0.2)

roll_with = 150
data_for_profiles_rolled = np.roll(data_for_profiles,roll_with, axis=2) ##note␣
↪axis 2 is the bins axis
plt.plot(data_for_profiles_rolled[subint_num,0,:], label="Profile shifted by %d␣
↪bins" %roll_with)
```

```
plt.legend(fontsize=8, loc='best')
plt.show()
```

[ ]:
```
cd 3sSubints_csv/
```

[ ]:
```
##Create dataframes per subint containing Stokes profiles
for subint in range(num_subint):
    df = pd.DataFrame(data_for_profiles_rolled[subint,:,:].
 ↪T,columns=["I","Q","U","V"]) ##transpose to have I, Q, U, V as columns␣
 ↪(instead of rows)
    df.to_csv("Profile_stokes_%d.csv" %subint, index=False)
    #df.to_csv("Profile_stokes_%d.csv" % (subint + 1), index=False)
```

2 Fitting Gaussians

[ ]:
```
subints ={}
for i in range(0,num_subint):
    filename = f"Profile_stokes_{i}.csv"
    subints[i] = np.loadtxt(filename, delimiter=",", skiprows=1)
```

[ ]:
```
def Gauss(x, a, x0, sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
```

[ ]:
```
x_values = np.arange(1,257)
y_values = {}
popt_dict ={}
pcov_dict = {}
skewness = {}
kurt ={}
fontsize=8.5
sigma ={}
fit_success = {}  # Track which fits succeeded
```

[ ]:
```
for i in range(0, num_subint):
    try:
        y_values[i] = subints[i][:, 0]  # Stokes I for subintegration i

        #optional: skip very noisy or flat signals
        if np.max(y_values[i]) - np.min(y_values[i]) < 1e-3:
            print(f"Skipped {i} due to low signal variation")
            popt_dict[i] = np.zeros(3)  #3 parameters: amplitude, mean, std
            pcov_dict[i] = np.zeros((3, 3))  # covariance is 3x3 matrix
            fit_success[i] = False
            continue

        # Fit Gaussian
```

```
        popt_dict[i], pcov_dict[i] = curve_fit(Gauss, x_values, y_values[i],␣
↪p0=[max(y_values[i]), np.mean(x_values), np.std(x_values)], maxfev = 900)
        skewness[i] = skew(y_values[i])
        kurt[i] = kurtosis(y_values[i])
        fit_success[i] = True
        #print(f"Fit success: {i}")

    except Exception as e:
        #print(f"Fit failed at {i}: {e}")
        popt_dict[i] = np.zeros(3)
        pcov_dict[i] = np.zeros((3, 3))
        skewness[i] = 0
        kurt[i] = 0
        fit_success[i] = False
```

[ ]:
```
#Gaussian_data[1431]
```

[ ]:
```
#first column is subint
#amplitude = popt_dict[i][0]
#std deviation = popt_dict[i][2]
#skewness = skewness[i]
#kurtosis = kurt[i]
#mean = popt_dict[i][1]


Gaussian_data = np.empty((num_subint, 6))

for i in range(0, num_subint):
    Gaussian_data[i][0] = i
    Gaussian_data[i][1]= popt_dict[i][0]
    Gaussian_data[i][2]= popt_dict[i][2]
    Gaussian_data[i][3]= skewness[i]
    Gaussian_data[i][4]= kurt[i]
    Gaussian_data[i][5]= popt_dict[i][1]
```

[ ]:
```
amplitudes_G = Gaussian_data[:, 1]
std_devs_G = Gaussian_data[:, 2]
```

[ ]:
```
plt.scatter(amplitudes_G, std_devs_G, c='blue', marker='o',s=10)
plt.title('amp, std dev of 2hr data 3s subint w/o skew')
plt.xlabel('Amplitude')
plt.ylabel('Std Deviation')
plt.ylim(-0.0001, 70)
plt.xlim(-0.001, 0.1)
plt.show()
```

[ ]:
```
sizes = np.abs(Gaussian_data[:, 3]) * 2  # scale skewness for visibility
plt.scatter(amplitudes_G, std_devs_G, c='blue', marker='o',s=sizes)
plt.title('amp, std dev, skew of 2hr data 3s subint')
plt.xlabel('Amplitude')
plt.ylabel('Std Deviation')
plt.ylim(-0.0001, 70)
plt.xlim(-0.001, 0.1)
plt.show()

#skewness shown in size of marker
```

[ ]:
```
#Loop through each subint and apply the filters
for i in range(0, num_subint):
    amp = Gaussian_data[i][1]
    std = Gaussian_data[i][2]

    #if both amplitude and std dev are 0, reject
    if amp == 0 and std == 0:
        fit_success[i] = False
        continue

    #if amplitude or std dev out of desired range, reject
    if not (0 < amp < 0.025):
        fit_success[i] = False
        continue

    if not (0 < std < 45):
        fit_success[i] = False
        continue
    #GAUSSIANITY TEST
    data = data_for_profiles_rolled[i, 0, :]
    result=normaltest(data)

    if (result.pvalue >=0.05):
        fit_success[i] = False
        continue


    fit_success[i] = True

#building the filtered_data array using only the successful fits
filtered_data = np.array([Gaussian_data[i] for i in range(num_subint) if␣
 ↪fit_success.get(i, False)])
```

[ ]:
```
amplitudes_f = filtered_data[:, 1]
std_devs_f = filtered_data[:, 2]
skewness_f = filtered_data[:,3]
```

```
kurtosis_f = filtered_data[:,4]
mean_f = filtered_data[:,5]
```

[ ]:
```
# Stack features vertically and transpose to shape (n_samples, n_features)
data = np.vstack([amplitudes_f, std_devs_f, skewness_f, kurtosis_f, mean_f]).T

fig = corner.corner(
    data,
    labels=["Amplitude", "Std. Deviation", "Skewness", "Kurtosis", "Mean"],
    quantiles=[0.16, 0.5, 0.84],
    show_titles=True,
    title_kwargs={"fontsize": 10},
)

#figure.savefig("corner_plot.png", dpi=300, bbox_inches="tight")
```

[ ]:
```
fig, ax = plt.subplots(5,5, figsize=(20,20))

markersize=7

ax[0][0].scatter(amplitudes_f, amplitudes_f, s=markersize)
ax[1][0].scatter(amplitudes_f, std_devs_f, s=markersize)
ax[2][0].scatter(amplitudes_f, skewness_f, s=markersize)
ax[3][0].scatter(amplitudes_f, kurtosis_f, s=markersize)
ax[4][0].scatter(amplitudes_f, mean_f, s=markersize)
ax[0][0].set_title("Amplitude", size=10)
ax[0][0].set_ylabel("Ampitude", size=10)
ax[1][0].set_ylabel("Std. Deviation", size=10)
ax[2][0].set_ylabel("Skewness", size=10)
ax[3][0].set_ylabel("Kurtosis", size=10)
ax[4][0].set_ylabel("Mean", size=10)


ax[0][1].scatter(std_devs_f, amplitudes_f, s=markersize)
ax[0][1].set_title("Std. Deviation", size=10)
ax[1][1].scatter(std_devs_f, std_devs_f, s=markersize)
ax[2][1].scatter(std_devs_f, skewness_f, s=markersize)
ax[3][1].scatter(std_devs_f, kurtosis_f, s=markersize)
ax[4][1].scatter(std_devs_f, mean_f, s=markersize)


ax[0][2].scatter(skewness_f, amplitudes_f, s=markersize)
ax[0][2].set_title("Skewness", size=10)
ax[1][2].scatter(skewness_f, std_devs_f, s=markersize)
ax[2][2].scatter(skewness_f, skewness_f, s=markersize)
ax[3][2].scatter(skewness_f, kurtosis_f, s=markersize)
ax[4][2].scatter(skewness_f, mean_f, s=markersize)
```

```
ax[0][3].scatter(kurtosis_f, amplitudes_f, s=markersize)
ax[0][3].set_title("Kurtosis", size=10)
ax[1][3].scatter(kurtosis_f, std_devs_f, s=markersize)
ax[2][3].scatter(kurtosis_f, skewness_f, s=markersize)
ax[3][3].scatter(kurtosis_f, kurtosis_f, s=markersize)
ax[4][3].scatter(kurtosis_f, mean_f, s=markersize)


ax[0][4].scatter(mean_f, amplitudes_f, s=markersize)
ax[0][4].set_title("Mean", size=10)
ax[1][4].scatter(mean_f, std_devs_f, s=markersize)
ax[2][4].scatter(mean_f, skewness_f, s=markersize)
ax[3][4].scatter(mean_f, kurtosis_f, s=markersize)
ax[4][4].scatter(mean_f, mean_f, s=markersize)


for row in range(5):
    for col in range(1, 5):  #skip col=0
        ax[row][col].tick_params(labelleft=False)

plt.tight_layout(pad=1.0)

plt.savefig("parameters.png", dpi=300, bbox_inches="tight")
plt.show()
```

[ ]:
```
sizes = np.abs(filtered_data[:, 3]) * 2  # scaling skewness for visibility

plt.scatter(filtered_data[:, 1], filtered_data[:, 2], c='blue', marker='o',␣
 ↪s=sizes)
plt.title('amp, std dev, skew of 2hr data 3s subint')
plt.xlabel('Amplitude')
plt.ylabel('Std Deviation')
#plt.ylim(-1, 40)
#plt.xlim(-0.01, 0.045)
plt.show()
```

[ ]:
```
#selecting all columns except the first
data_for_dbscan = filtered_data[:, 1:]
dbscan = DBSCAN(eps=3, min_samples=10)

#Applying DBSCAN on this subset of data
labels = dbscan.fit_predict(data_for_dbscan)
```

[ ]:
```
#New after gaussianity test
```

```
sizes = np.abs(filtered_data[:, 3]) * 11  # scale skewness for visibility

#mask = labels != -1
#labels_clust = labels[mask]


plt.figure(figsize=(8, 6))
#plt.scatter(filtered_data[:, 1], filtered_data[:, 2],c=labels, cmap='viridis',␣
↪marker='o',s=sizes)

plt.scatter(filtered_data[:, 1], filtered_data[:, 2],c=labels, cmap='viridis',␣
↪marker='o',s=sizes)
plt.scatter(filtered_data[labels == -1, 1], filtered_data[labels == -1, 2],␣
↪c='red', s=10, marker='x', label='Noise')
# Mark noise points (label = -1)
#plt.scatter(filtered_data[labels == -1, 1], filtered_data[labels == -1, 2],␣
↪c='red', s=10, marker='x', label='Noise')
plt.title('amp, std dev, skew, of 2 hr data 3s subint with gaussianity test')
plt.xlabel('Amplitude')
plt.ylabel('Std Deviation')
#plt.ylim(-0.0001, 70)
#plt.xlim(-0.001, 0.1)

#amp, std dev, skew, kurt, mean are being used for clustering algorithm
```

[ ]:
```
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
print(f"Number of clusters found: {n_clusters}")
```

[ ]:
```
subints_list = filtered_data[:, 0].astype(int)
amplitudes = filtered_data[:, 1]
std_devs = filtered_data[:, 2]

images = []
```

[ ]:
```
#colormap from DBSCAN labels
unique_labels = np.unique(labels)
norm = Normalize(vmin=unique_labels.min(), vmax=unique_labels.max())
cmap = plt.colormaps.get_cmap('viridis')

colors = [to_hex(cmap(norm(label))) if label != -1 else '#ff0000'  #Red for␣
↪noise
          for label in labels]
```

[ ]:
```
#np.shape(filtered_data[:, 0].astype(int))
```

```
[ ]: for subint in subints_list:
    intensities = data_for_profiles_rolled[subint, 0, :]
    gaussians = Gauss(x_values, *popt_dict[subint])

    #matplotlib figure of the profile
    fig, ax = plt.subplots()
    ax.plot(intensities, color='blue',alpha=0.7)
    ax.plot(gaussians, color='black')
    ax.set_title(f"Subint {subint}")
    #ax.axis("off")  # Hide axes for cleaner display

    buf = BytesIO()
    plt.savefig(buf, format="png", bbox_inches='tight')
    plt.close(fig)

    encoded = base64.b64encode(buf.getvalue()).decode()
    images.append(f"<img src='data:image/png;base64,{encoded}' width='300'>")

output_notebook()

#Bokeh data source
source = ColumnDataSource(data=dict(
    amplitude=amplitudes,
    std_dev=std_devs,
    subint=subints_list.astype(str),
    image=images,
    color=colors
))

p = figure(title="Amplitude vs Std Dev", width=800, height=600,
           x_axis_label="Amplitude", y_axis_label="Std Deviation",
           tools="pan,wheel_zoom,reset")

p.scatter("amplitude", "std_dev", size=8, color="color", alpha=0.5,␣
 ↪source=source)

#image of profiles when hovering
hover = HoverTool(tooltips="""
    <div>
        <span style="font-size: 12px;">Subint: @subint</span><br>
        @image
    </div>
""")

#box = BoxZoomTool()
p.add_tools(hover)
```

```
show(p)
```

[ ]:
```
from bokeh.plotting import output_file
```

[ ]:
```
for subint in subints_list:
    intensities = data_for_profiles_rolled[subint, 0, :]
    gaussians = Gauss(x_values, *popt_dict[subint])

    #matplotlib figure of the profile
    fig, ax = plt.subplots()
    ax.plot(intensities, color='blue',alpha=0.7)
    ax.plot(gaussians, color='black')
    ax.set_title(f"Subint {subint}")
    #ax.axis("off")

    buf = BytesIO()
    plt.savefig(buf, format="png", bbox_inches='tight')
    plt.close(fig)

    encoded = base64.b64encode(buf.getvalue()).decode()
    images.append(f"<img src='data:image/png;base64,{encoded}' width='300'>")

output_file("/idia/projects/pulsar-timing/users/athambiran/singlepulses/2hr_obs/
↪bokeh_plot.html")

source = ColumnDataSource(data=dict(
    amplitude=amplitudes,
    std_dev=std_devs,
    subint=subints_list.astype(str),
    image=images,
    color=colors
))

p = figure(title="Amplitude vs Std Dev", width=800, height=600,
           x_axis_label="Amplitude", y_axis_label="Std Deviation",
           tools="pan,wheel_zoom,reset")

p.scatter("amplitude", "std_dev", size=8, color="color", alpha=0.5,␣
↪source=source)


hover = HoverTool(tooltips="""
    <div>
        <span style="font-size: 12px;">Subint: @subint</span><br>
        @image
    </div>
""")
```

```
#box = BoxZoomTool()
p.add_tools(hover)

show(p)
```

[ ]:
```
save(p,'/users/rev/abigail/2hr_3s.html')
```

[ ]:

[ ]:

[ ]:

[ ]: