20230612_background

July 25, 2023

1 Backgroud without microwave

And * no gas in gas cell (No Ps formation) * gas in gas cell (Ps formation)

ref: https://mathematica.stackexchange.com/a/124338

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from functools import lru_cache, wraps
import time
from typing import Callable, List, Tuple, Union, Optional, Any, Dict

from scipy.optimize import curve_fit
from scipy.special import voigt_profile, factorial
```

[ ]:
```python
CMP_RADIUS = 40 # mm

CMP_RADIUS_WARNING = False
```

1.1 Background

[ ]:
```python
def read_ps_beam_file_no_properties(file: Path, coordinate_correction=True) ->␣
↪list[np.ndarray, np.ndarray, list[np.ndarray, np.ndarray], np.ndarray]:
    """
    read the data file from the Ps spectroscopy

    :param
        file: the file path
        coordinate_correction: if True, the coordinate will be corrected␣
↪according to the setup
            which is
                [x'] = 0.93*[x] + 0.5
                [y'] = 0.88*[y] + 0.5
    :return:
        x: np.ndarray, the x coordinate
        y: np.ndarray, the y coordinate
```

```
        meshgrid: list[np.ndarray, np.ndarray], the meshgrid of the x and y␣
↪coordinate
        matrix: np.ndarray, the data matrix
        properties: (this is based on the file name)
            power: the power of the microwave, in dBm
            frequency: the frequency of the microwave, in GHz
            time: the time when the data is taken
    """
    file = Path(file)

    # read csv file
    data = np.loadtxt(file, delimiter=",")
    # the data file is provided in x,y,data format

    # transform to x,y,matrix format
    x = np.unique(data[:, 0])
    x.sort()
    y = np.unique(data[:, 1])
    y.sort()

    matrix = np.zeros((len(x), len(y)))

    # Find the indices of non-zero entries
    data_non_zero_entries = np.nonzero(data[:, 2])

    # Extract the non-zero data into a separate variable
    non_zero_data = data[data_non_zero_entries]

    # Get the corresponding x and y indices for the non-zero entries
    x_indices = np.searchsorted(x, non_zero_data[:, 0])
    y_indices = np.searchsorted(y, non_zero_data[:, 1])

    # Assign the non-zero values to the matrix using advanced indexing
    matrix[x_indices, y_indices] = non_zero_data[:, 2]

    # check if the code gives the same result
    # _matrix = np.zeros((len(x), len(y)))
    # # since the data is provided in x,y,data format, we need to transform it␣
↪to matrix format
    # _data_non_zero_entries = np.argwhere(data[:, 2] != 0)
    # for i in _data_non_zero_entries:
    #     _matrix[np.where(x == data[i, 0]), np.where(
    #         y == data[i, 1])] = data[i, 2]
    # print(np.all(matrix == _matrix))

    if coordinate_correction:  # should always be true
        x = 0.93*x + 0.5
```

```
        y = 0.88*y + 0.5

    meshgrid_x, meshgrid_y = np.meshgrid(x, y, indexing="ij", sparse=True)

    properties = {
        "filename": file.name,
    }
    return x, y, [meshgrid_x, meshgrid_y], matrix, properties
```

[ ]:
```
def hotspot_10th_largest(matrix: np.ndarray) -> np.ndarray:
    """
    Hotspot removal method
    The hotspots are defined as the data that is larger than twice of the 10th␣
 ↪largest data

    :param
        matrix: the data matrix
    :return:
        matrix_without_hotspots: the data matrix without hotspots
    """

    # the hotspots are defined as the data that is larger than twice of the␣
 ↪10th largest data
    # the hotspots are removed by setting the value to 0

    # get the 10th largest data
    largest_10th_data = np.sort(matrix.flatten())[-10]
    # set the data that is larger than twice of the 10th largest data to 0
    matrix_without_hotspots = np.where(matrix > 2*largest_10th_data, 0, matrix)

    return matrix_without_hotspots


def get_matrix_without_hotspots(file: Path, coordinate_correction: bool = True,␣
 ↪hotspot_func=hotspot_10th_largest):
    """
    read the data file from the Ps spectroscopy and get the raw matrix

    1. remove the data outside the circle with radius of CMP_RADIUS
    2. remove the hotspots via any of the following methods
        * remove the data that is larger than twice of the 10th largest data
    3. sum up the data

    :param
        file: the file path
        coordinate_correction: if True, the coordinate will be corrected␣
 ↪according to the setup
```

```
        hotspot_func: the function to remove the hotspots

    :return:
        matrix_without_hotspots: the data matrix without hotspots
        properties: (this is based on the file name)
            see read_ps_beam_file
    """
    x, y, [meshgrid_x, meshgrid_y], matrix, properties =␣
↪read_ps_beam_file_no_properties(
        file, coordinate_correction)

    # assertion check
    # The data in xy plane should be inside a circle with radius of CMP_RADIUS

    # generate a matrix of the distance to the center
    distance_matrix = np.sqrt((meshgrid_x)**2 + (meshgrid_y)**2)

    global outside_circle_boolean_matrix # for this notebook only
    # if the matrix is outside the circle, set it to True, otherwise False
    outside_circle_boolean_matrix = distance_matrix > CMP_RADIUS

    # check the data outside the circle is zero
    if CMP_RADIUS_WARNING:
        # set the data inside the circle to 0
        matrix_without_data_inside_circle = np.where(
            outside_circle_boolean_matrix, matrix, 0)
        if not np.all(matrix_without_data_inside_circle == 0):
            indexes = np.argwhere(matrix_without_data_inside_circle != 0)
            for index in indexes:
                print(
                    f"[WARN] ({x[index[0]]}, {y[index[1]]}) has data outside␣
↪CMP_RADIUS, the value is {matrix_without_data_inside_circle[index[0],␣
↪index[1]]}")
    # ends of assertion check

    matrix_data_inside_circle = np.where(
        outside_circle_boolean_matrix, -np.inf, matrix)

    # remove hotspots
    matrix_without_hotspots = hotspot_func(matrix_data_inside_circle)

    # sanity check
    # fig = plt.figure()
    # ax = fig.add_subplot(111, projection='3d')
    # ax.plot_surface(meshgrid_x, meshgrid_y, matrix_without_hotspots,␣
↪cmap='viridis')
    # ax.set_xlabel('x')
```

```
    # ax.set_ylabel('y')
    # ax.set_zlabel('counts')

    data = {
        "x": x,
        "y": y,
        "meshgrid_x": meshgrid_x,
        "meshgrid_y": meshgrid_y,
        "matrix": matrix,
        "matrix_without_hotspots": matrix_without_hotspots,
    }

    return matrix_without_hotspots, properties, data
```

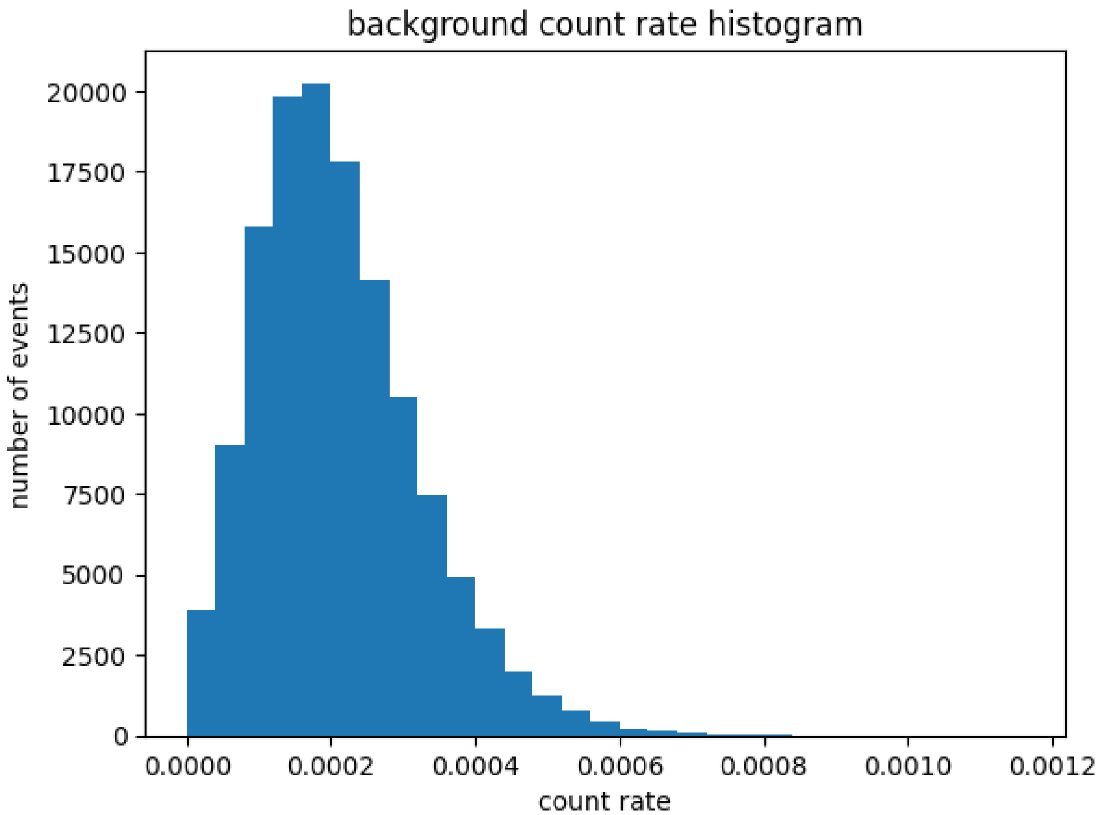
[ ]:
```
BACKGROUD_FILE = Path("./data/2023/06/07/background_LP_12V_RP_142V.txt")
BACKGROUND_FRAMETIME= 24140
```

[ ]:
```
background_matrix, background_properties, background_data =␣
 ↪get_matrix_without_hotspots(BACKGROUD_FILE)

background_matrix = background_matrix / BACKGROUND_FRAMETIME # rate
```

background histogram

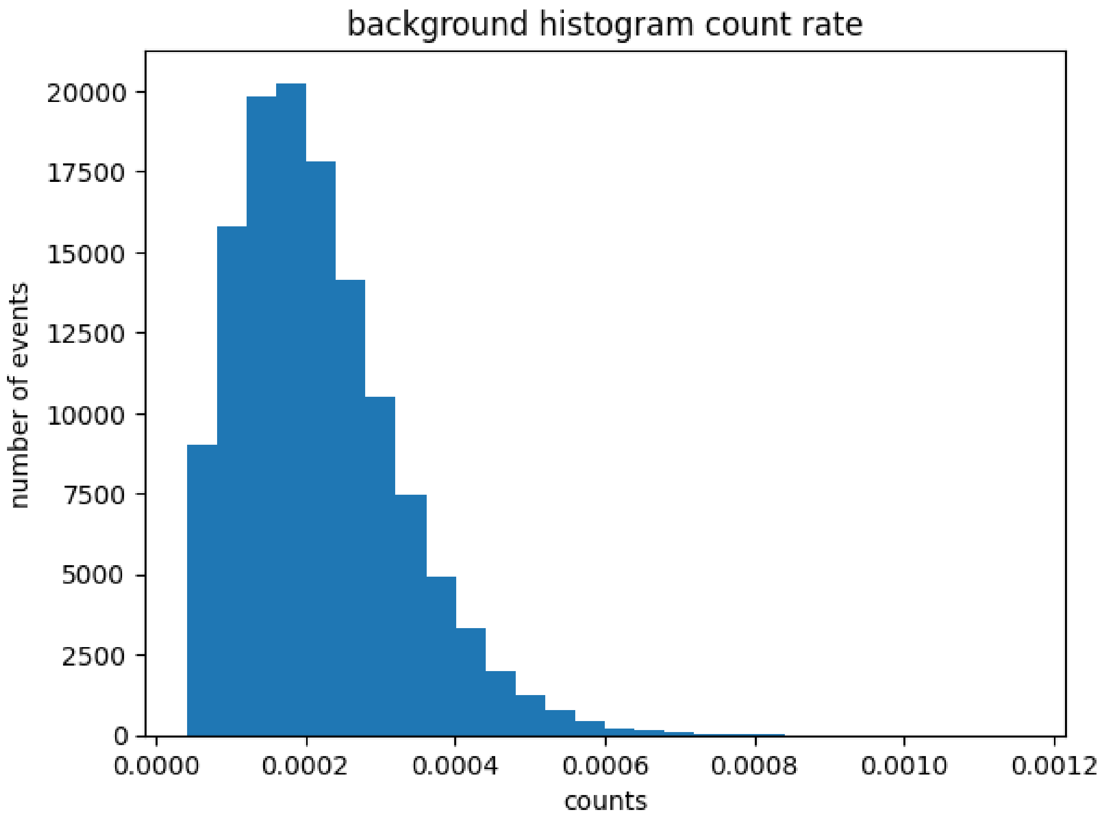
[ ]:
```
plt.figure()

plt.hist(background_matrix.flatten()[background_matrix.flatten()!=-np.inf],␣
 ↪bins=1 +
         int(np.max(background_matrix.flatten())*BACKGROUND_FRAMETIME))
plt.xlabel("count rate")
plt.ylabel("number of events")
# log scale
# plt.yscale("log")
plt.title("background count rate histogram")
```

[ ]:
```
Text(0.5, 1.0, 'background count rate histogram')
```



[ ]:
```
plt.figure()
background_matrix_without_inf = background_matrix.flatten()[background_matrix.
↪flatten()!=-np.inf]
plt.hist(background_matrix_without_inf[background_matrix_without_inf!=0],␣
↪bins=int(np.max(background_matrix.flatten())*BACKGROUND_FRAMETIME))
plt.xlabel("counts")
plt.ylabel("number of events")
# log scale
# plt.yscale("log")
plt.title("background histogram count rate")
```

[ ]: Text(0.5, 1.0, 'background histogram count rate')




```
[ ]: def r_squared(y, y_fit):
         return 1 - np.sum((y - y_fit)**2) / np.sum((y - np.mean(y))**2)


     def gaussian(x, mu, sigma, A):
         return A * np.exp(-(x - mu)**2 / (2 * sigma**2))


     def gaussian_fit(rate, hist):
         popt, pcov = curve_fit(gaussian, rate, hist, p0=[
                                rate[np.argmax(hist)], 0.0001, np.max(hist)],␣
      ↪bounds=(np.zeros(3), np.inf * np.ones(3)), maxfev=10000)

         # r-squared
         _r_squared = r_squared(hist, gaussian(rate, *popt))
         return popt, pcov, _r_squared


     def double_gaussian(x, mu1, sigma1, A1, mu2, sigma2, A2):
         return gaussian(x, mu1, sigma1, A1) + gaussian(x, mu2, sigma2, A2)
```

```python
def double_gaussian_fit(rate, hist):
    initial_guess_1, _, _ = gaussian_fit(rate, hist)

    # use the difference between data and the fit for one gaussian as the initial guess for the second gaussian
    difference = hist - gaussian(rate, *initial_guess_1)
    initial_guess_2, _, _ = gaussian_fit(rate, difference)

    initial_guess = np.concatenate((initial_guess_1, initial_guess_2))

    popt, pcov = curve_fit(
        double_gaussian, rate, hist, p0=initial_guess, bounds=(np.zeros(6), np.inf * np.ones(6)), maxfev=10000)

    if popt[0] < popt[3]:
        pass
    else:
        popt = np.array([popt[3], popt[4], popt[5], popt[0], popt[1], popt[2]])

    _r_squared = r_squared(hist, double_gaussian(rate, *popt))

    return popt, pcov, _r_squared


def triple_gaussian(x, mu1, sigma1, A1, mu2, sigma2, A2, mu3, sigma3, A3):
    return gaussian(x, mu1, sigma1, A1) + gaussian(x, mu2, sigma2, A2) + gaussian(x, mu3, sigma3, A3)


def triple_gaussian_fit(rate, hist):
    initial_guess_12, _, _ = double_gaussian_fit(rate, hist)

    difference = hist - double_gaussian(rate, *initial_guess_12)
    initial_guess_3, _, _ = gaussian_fit(rate, difference)

    initial_guess = np.concatenate((initial_guess_12, initial_guess_3))

    popt, pcov = curve_fit(
        triple_gaussian, rate, hist, p0=initial_guess, bounds=(np.zeros(9), np.inf * np.ones(9)), maxfev=10000)

    _r_squared = r_squared(hist, triple_gaussian(rate, *popt))

    return popt, pcov, _r_squared
```

```
def quadruple_gaussian(x, mu1, sigma1, A1, mu2, sigma2, A2, mu3, sigma3, A3,␣
↪mu4, sigma4, A4):
    return gaussian(x, mu1, sigma1, A1) + gaussian(x, mu2, sigma2, A2) +␣
↪gaussian(x, mu3, sigma3, A3) + gaussian(x, mu4, sigma4, A4)


def quadruple_gaussian_fit(rate, hist):
    initial_guess_123, _, _ = triple_gaussian_fit(rate, hist)

    difference = hist - triple_gaussian(rate, *initial_guess_123)
    initial_guess_4, _, _ = gaussian_fit(rate, difference)

    initial_guess = np.concatenate((initial_guess_123, initial_guess_4))

    popt, pcov = curve_fit(
        quadruple_gaussian, rate, hist, p0=initial_guess, bounds=(np.zeros(12),␣
↪np.inf * np.ones(12)), maxfev=10000)

    _r_squared = r_squared(hist, quadruple_gaussian(rate, *popt))

    return popt, pcov, _r_squared
```

[ ]:
```
backgroud_bins = int(np.max(background_matrix)*BACKGROUND_FRAMETIME) +1
background_hist = np.histogram(
    background_matrix[background_matrix!=-np.inf],
    bins=backgroud_bins, range=(0, np.max(background_matrix)))[0]

background_rates = np.arange(0, backgroud_bins)/BACKGROUND_FRAMETIME
```

1.1.1 single - quadruple gaussian fit for background

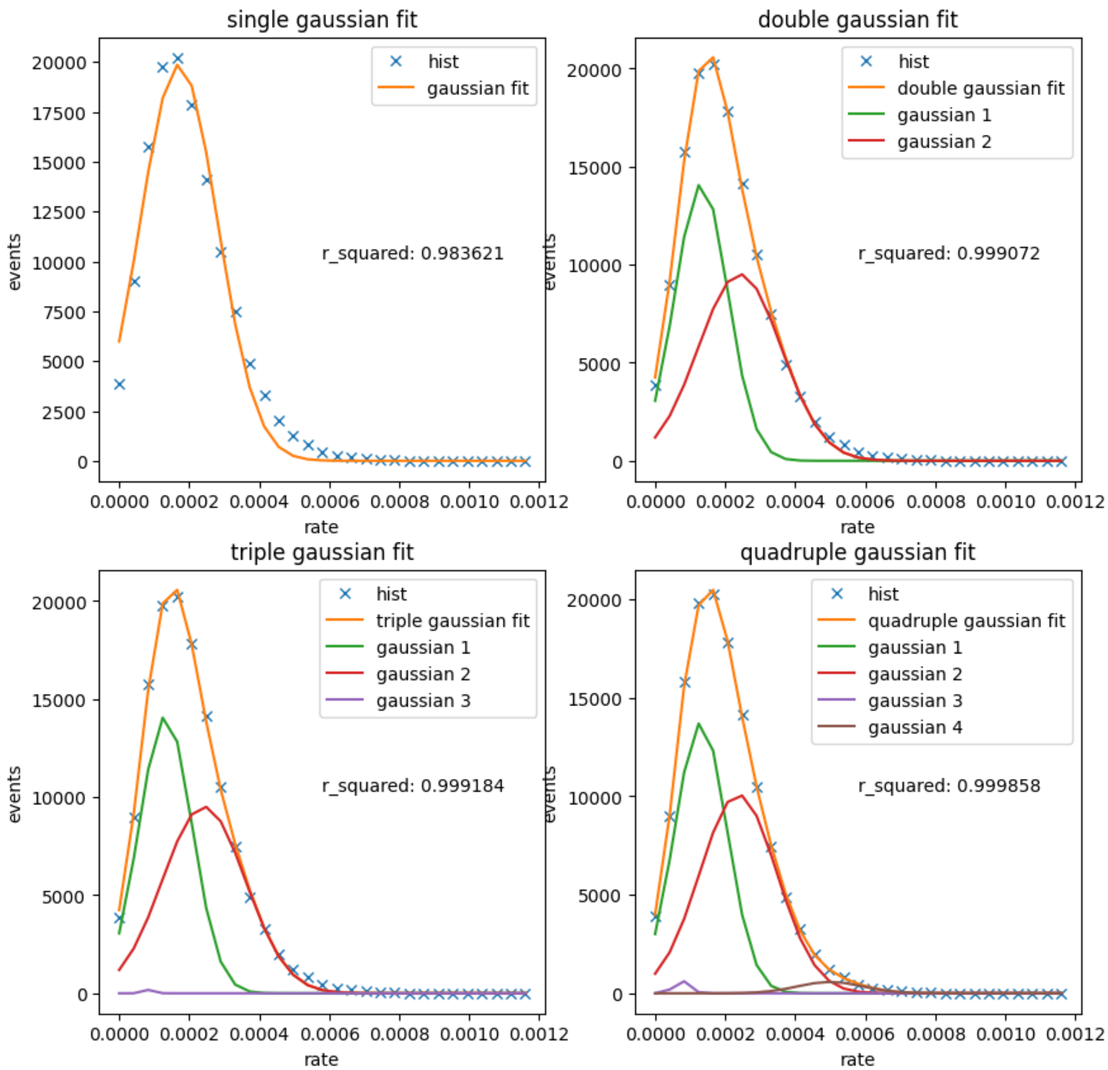
[ ]:
```
# plot single - quadruple gaussian fit in one plot
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs = axs.flatten()

popt,_ ,_r_squared= gaussian_fit(background_rates,background_hist)
axs[0].plot(background_rates, background_hist, "x", label="hist")
axs[0].plot(background_rates, gaussian(background_rates, *popt),␣
↪label="gaussian fit")
axs[0].legend()
axs[0].set_title("single gaussian fit")
axs[0].set_xlabel("rate")
axs[0].set_ylabel("events")
axs[0].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[0].
↪transAxes)
print("single gaussian fit")
```

```
print("mu: ", popt[0]/BACKGROUND_FRAMETIME)
print("sigma: ", popt[1]/BACKGROUND_FRAMETIME)
print("A: ", popt[2])
print("r_squared: ", _r_squared)


popt, _, _r_squared = double_gaussian_fit(background_rates,background_hist)
axs[1].plot(background_rates, background_hist, "x", label="hist")
axs[1].plot(background_rates, double_gaussian(background_rates, *popt),␣
 ↪label="double gaussian fit")
axs[1].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
 ↪popt[2]), label="gaussian 1")
axs[1].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
 ↪popt[5]), label="gaussian 2")
axs[1].legend()
axs[1].set_title("double gaussian fit")
axs[1].set_xlabel("rate")
axs[1].set_ylabel("events")
axs[1].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[1].
 ↪transAxes)
print("double gaussian fit")
print(f"-----|--1--|--2--|")
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = triple_gaussian_fit(background_rates,background_hist)
axs[2].plot(background_rates, background_hist, "x", label="hist")
axs[2].plot(background_rates, triple_gaussian(background_rates, *popt),␣
 ↪label="triple gaussian fit")
axs[2].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
 ↪popt[2]), label="gaussian 1")
axs[2].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
 ↪popt[5]), label="gaussian 2")
axs[2].plot(background_rates, gaussian(background_rates, popt[6], popt[7],␣
 ↪popt[8]), label="gaussian 3")
axs[2].legend()
axs[2].set_title("triple gaussian fit")
axs[2].set_xlabel("rate")
axs[2].set_ylabel("events")
axs[2].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[2].
 ↪transAxes)

print("triple gaussian fit")
print(f"-----|--1--|--2--|--3--|")
```

```
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = quadruple_gaussian_fit(background_rates,background_hist)
axs[3].plot(background_rates, background_hist, "x", label="hist")
axs[3].plot(background_rates, quadruple_gaussian(background_rates, *popt),␣
 ↪label="quadruple gaussian fit")
axs[3].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
 ↪popt[2]), label="gaussian 1")
axs[3].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
 ↪popt[5]), label="gaussian 2")
axs[3].plot(background_rates, gaussian(background_rates, popt[6], popt[7],␣
 ↪popt[8]), label="gaussian 3")
axs[3].plot(background_rates, gaussian(background_rates, popt[9], popt[10],␣
 ↪popt[11]), label="gaussian 4")
axs[3].legend()
axs[3].set_title("quadruple gaussian fit")
axs[3].set_xlabel("rate")
axs[3].set_ylabel("events")
axs[3].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[3].
 ↪transAxes)
print("quadruple gaussian fit")
print(f"-----|--1--|--2--|--3--|--4--|")
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|{popt[9]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|{popt[10]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|{popt[11]:.4g}|")
print("r_squared: ", _r_squared)
```

```
single gaussian fit
mu:  7.066155194573954e-09
sigma:  4.562849236879179e-09
A:  19883.555503818123
r_squared:  0.9836211087909913
double gaussian fit
-----|--1--|--2--|
-mu--|0.0001324|0.0002419|
-sig-|7.566e-05|0.0001186|
-A---|1.413e+04|9519|
r_squared:  0.9990723232186636
triple gaussian fit
-----|--1--|--2--|--3--|
-mu--|0.0001324|0.0002419|8.487e-05|
-sig-|7.566e-05|0.0001186|1.494e-05|
-A---|1.413e+04|9519|172|
r_squared:  0.9991842827556141
```

```
quadruple gaussian fit
-----|--1--|--2--|--3--|--4--|
-mu--|0.0001306|0.0002375|7.574e-05|0.0005001|
-sig-|7.494e-05|0.0001103|2.196e-05|9.329e-05|
-A---|1.374e+04|1.009e+04|641.8|576.8|
r_squared:  0.9998582376859723
```

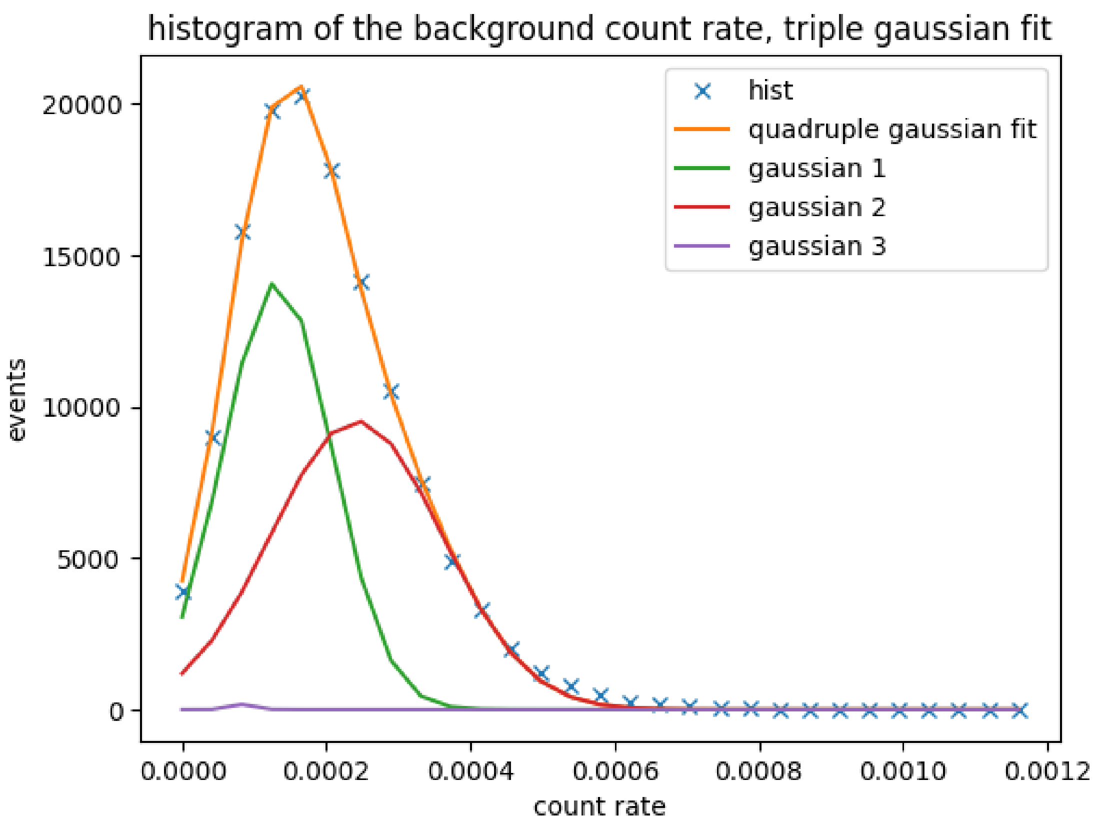


triple guassian fit for background

[ ]:
```python
plt.figure()

popt, _, _r_squared = triple_gaussian_fit(background_rates,background_hist)
```

```
plt.plot(background_rates, background_hist, "x", label="hist")
plt.plot(background_rates, triple_gaussian(background_rates, *popt),␣
 ↪label="quadruple gaussian fit")

plt.plot(background_rates, gaussian(background_rates, *popt[:3]),␣
 ↪label="gaussian 1")
plt.plot(background_rates, gaussian(background_rates, *popt[3:6]),␣
 ↪label="gaussian 2")
plt.plot(background_rates, gaussian(background_rates, *popt[6:9]),␣
 ↪label="gaussian 3")


plt.xlabel("count rate")
plt.ylabel("events")
plt.legend()
plt.title("histogram of the background count rate, triple gaussian fit")


print(f"-----|--1--|--2--|--3--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|")


# r squared
print(f"r squared: {_r_squared}")
```

```
-----|--1--|--2--|--3--|
--mu-|0.0001324|0.0002419|8.487e-05|
sigma|7.566e-05|0.0001186|1.494e-05|
--A--|1.413e+04|9519|172|
r squared: 0.9991842827556141
```



1.2 Ps Beam

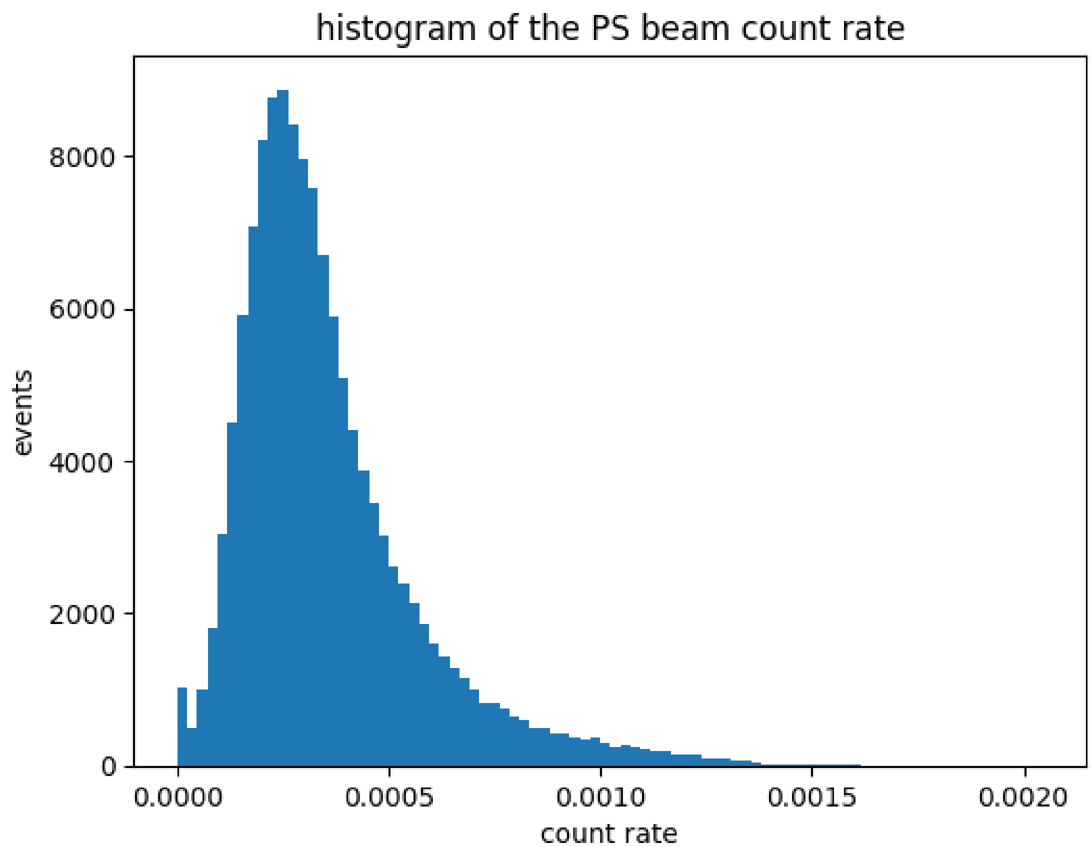
[ ]:
```
PSBEAM_FILE = Path("./data/2023/06/07/psbeam_LP_12V_RP_142V.txt")

PSBEAM_FRAMETIME= 41530
```

[ ]:
```
psbeam_matrix, psbeam_properties, psbeam_data =␣
  ↪get_matrix_without_hotspots(PSBEAM_FILE)

psbeam_matrix = psbeam_matrix/PSBEAM_FRAMETIME # rate
```

1.2.1 psbeam histogram

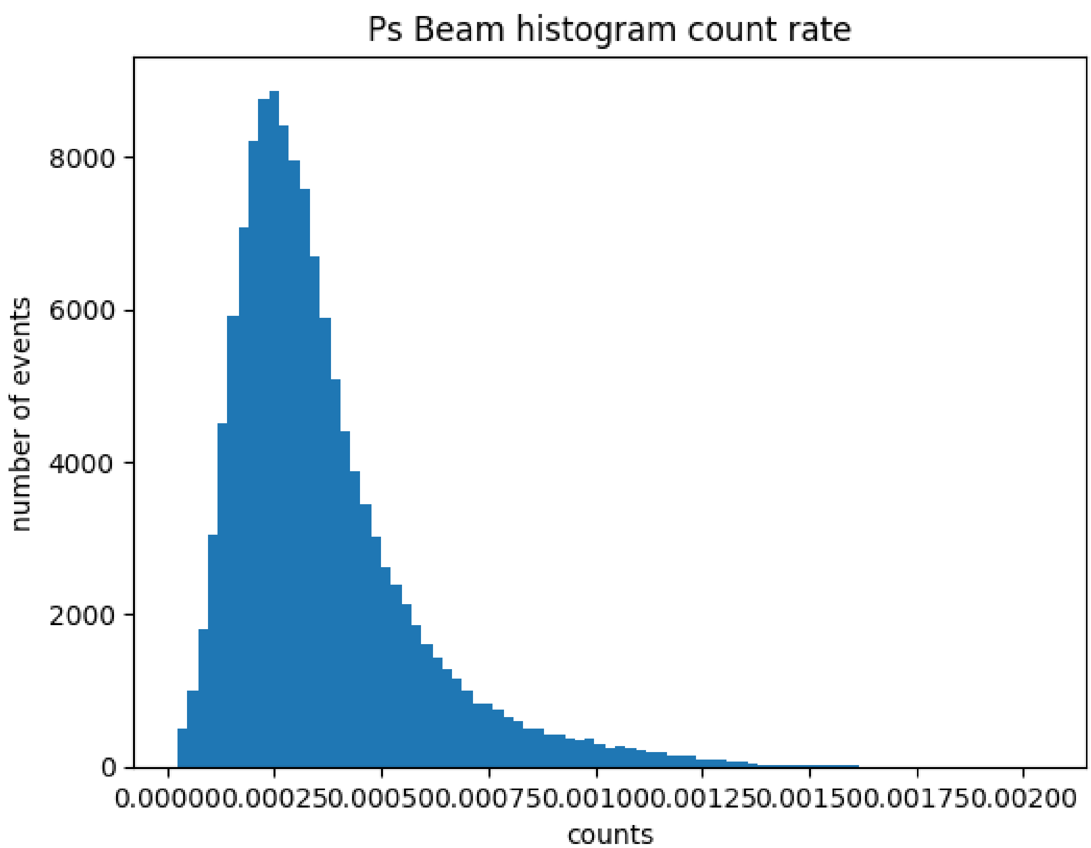
[ ]:
```
plt.figure()

plt.hist(psbeam_matrix.flatten()[background_matrix.flatten()!=-np.
  ↪inf],bins=1+int(np.max(psbeam_matrix.flatten())*PSBEAM_FRAMETIME))
plt.xlabel("count rate")
plt.ylabel("events")
plt.title("histogram of the PS beam count rate")
```

[ ]: Text(0.5, 1.0, 'histogram of the PS beam count rate')



[ ]:
```
plt.figure()

psbeam_matrix_without_inf = psbeam_matrix.flatten()[psbeam_matrix.flatten()!
 ↪=-np.inf]
plt.hist(psbeam_matrix_without_inf.flatten()[psbeam_matrix_without_inf.
 ↪flatten()!=0], bins=int(np.max(psbeam_matrix.flatten())*PSBEAM_FRAMETIME))
plt.xlabel("counts")
plt.ylabel("number of events")
# log scale
# plt.yscale("log")
plt.title("Ps Beam histogram count rate")
```

[ ]: Text(0.5, 1.0, 'Ps Beam histogram count rate')



[ ]:
```
psbeam_bins = int(np.max(psbeam_matrix.flatten())*PSBEAM_FRAMETIME) +1
psbeam_hist = np.histogram(psbeam_matrix[psbeam_matrix!=-np.inf],␣
 ↪bins=psbeam_bins,range=(0, np.max(psbeam_matrix)))[0]

psbeam_rates = np.arange(0, psbeam_bins)/PSBEAM_FRAMETIME
```

1.2.2 single - quadruple gaussian fit for Ps beam

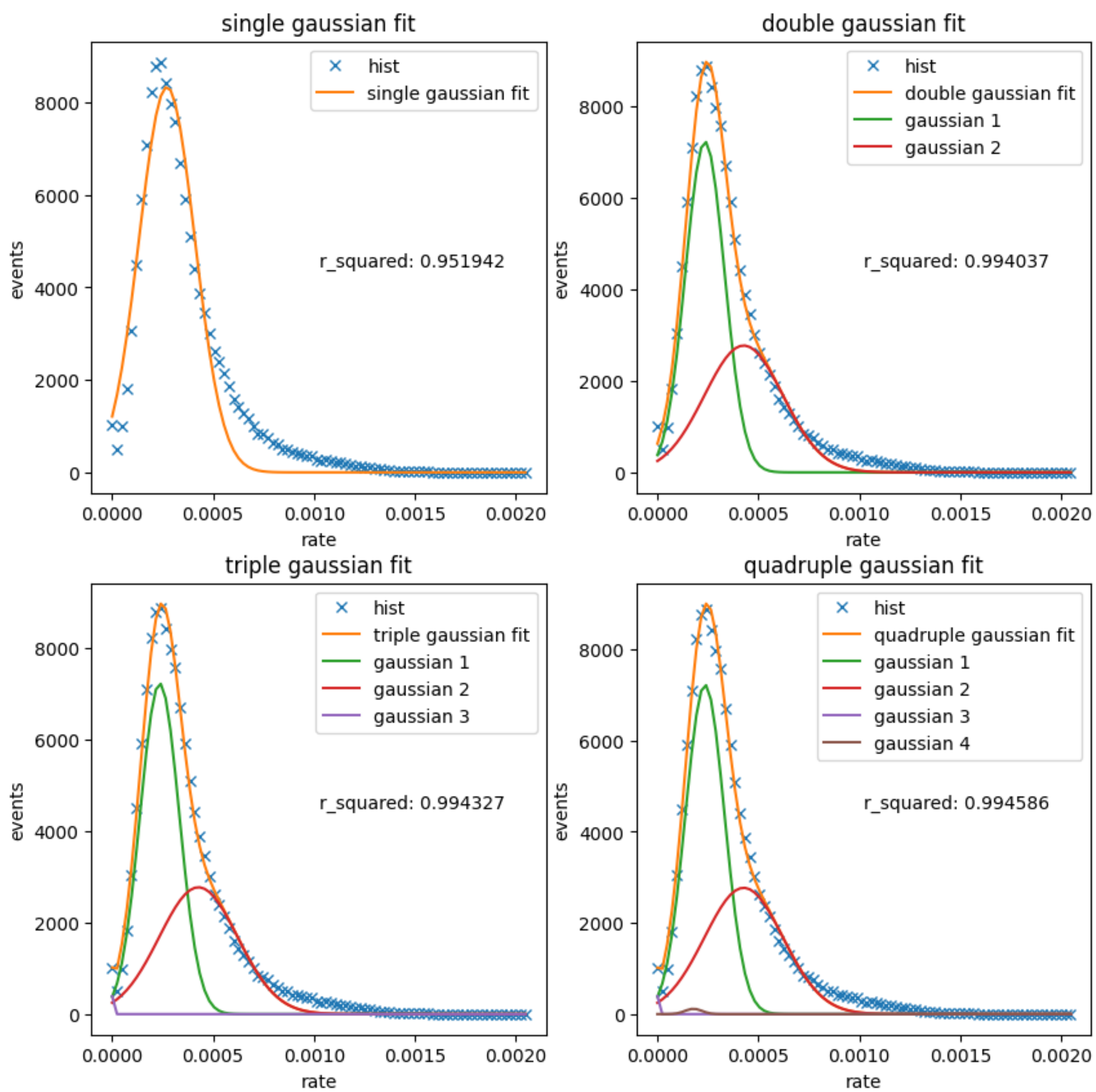
[ ]:
```
# plot single - quadruple gaussian fits in one plot

fig, axs = plt.subplots(2,2, figsize=(10,10))
axs=axs.flatten()

popt, _, _r_squared = gaussian_fit(psbeam_rates,psbeam_hist)
axs[0].plot(psbeam_rates, psbeam_hist, "x", label="hist")
axs[0].plot(psbeam_rates, gaussian(psbeam_rates, *popt), label="single gaussian␣
 ↪fit")
axs[0].legend()
axs[0].set_title("single gaussian fit")
```

```
axs[0].set_xlabel("rate")
axs[0].set_ylabel("events")
axs[0].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[0].
 ↪transAxes)
print("single gaussian fit")
print(f"-----|--1--|")
print(f"-mu--|{popt[0]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|")
print(f"-A---|{popt[2]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = double_gaussian_fit(psbeam_rates,psbeam_hist)
axs[1].plot(psbeam_rates, psbeam_hist, "x", label="hist")
axs[1].plot(psbeam_rates, double_gaussian(psbeam_rates, *popt), label="double␣
 ↪gaussian fit")
axs[1].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]),␣
 ↪label="gaussian 1")
axs[1].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]),␣
 ↪label="gaussian 2")
axs[1].legend()
axs[1].set_title("double gaussian fit")
axs[1].set_xlabel("rate")
axs[1].set_ylabel("events")
axs[1].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[1].
 ↪transAxes)
print("double gaussian fit")
print(f"-----|--1--|--2--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = triple_gaussian_fit(psbeam_rates,psbeam_hist)
axs[2].plot(psbeam_rates, psbeam_hist, "x", label="hist")
axs[2].plot(psbeam_rates, triple_gaussian(psbeam_rates, *popt), label="triple␣
 ↪gaussian fit")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]),␣
 ↪label="gaussian 1")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]),␣
 ↪label="gaussian 2")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[6], popt[7], popt[8]),␣
 ↪label="gaussian 3")
axs[2].legend()
axs[2].set_title("triple gaussian fit")
axs[2].set_xlabel("rate")
axs[2].set_ylabel("events")
```

```
axs[2].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[2].
  ↪transAxes)
print("triple gaussian fit")
print(f"-----|--1--|--2--|--3--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates,psbeam_hist)
axs[3].plot(psbeam_rates, psbeam_hist, "x", label="hist")
axs[3].plot(psbeam_rates, quadruple_gaussian(psbeam_rates, *popt),␣
  ↪label="quadruple gaussian fit")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]),␣
  ↪label="gaussian 1")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]),␣
  ↪label="gaussian 2")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[6], popt[7], popt[8]),␣
  ↪label="gaussian 3")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[9], popt[10], popt[11]),␣
  ↪label="gaussian 4")
axs[3].legend()
axs[3].set_title("quadruple gaussian fit")
axs[3].set_xlabel("rate")
axs[3].set_ylabel("events")
axs[3].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[3].
  ↪transAxes)
print("quadruple gaussian fit")
print(f"-----|--1--|--2--|--3--|--4--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|{popt[9]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|{popt[10]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|{popt[11]:.4g}|")
print("r_squared: ", _r_squared)
```

```
single gaussian fit
-----|--1--|
-mu--|0.0002713|
-sig-|0.0001381|
-A---|8336|
r_squared:  0.9519415849287788
double gaussian fit
-----|--1--|--2--|
--mu-|0.0002354|0.0004273|
sigma|9.692e-05|0.0001946|
--A--|7225|2771|
r_squared:  0.9940374944307048
triple gaussian fit
```

```
-----|--1--|--2--|--3--|
--mu-|0.0002354|0.0004273|1e-10|
sigma|9.692e-05|0.0001946|4.356e-07|
--A--|7225|2771|391.2|
r_squared:  0.9943272966061032
quadruple gaussian fit
-----|--1--|--2--|--3--|--4--|
--mu-|0.0002354|0.0004273|1e-10|0.0001785|
sigma|9.692e-05|0.0001946|4.356e-07|4.173e-05|
--A--|7225|2771|391.2|116.7|
r_squared:  0.9945856724170575
```

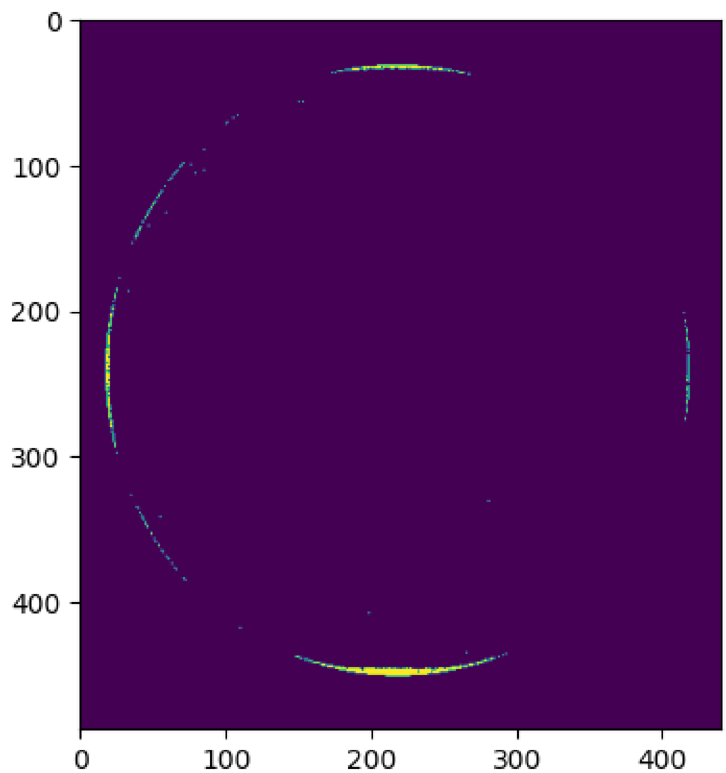


1.3 Always zero points

When using imshow, use transpose due to matrix

[ ]:
```
always_zero_points = np.logical_and(np.where(psbeam_matrix == 0,True,False) ,
    np.where(background_matrix == 0,True,False))

plt.imshow(always_zero_points, origin="upper")
```

[ ]: <matplotlib.image.AxesImage at 0x1490a7de470>



1.3.1 removing zero in fitting as most of them are on the edge

1.3.2 background zero removed

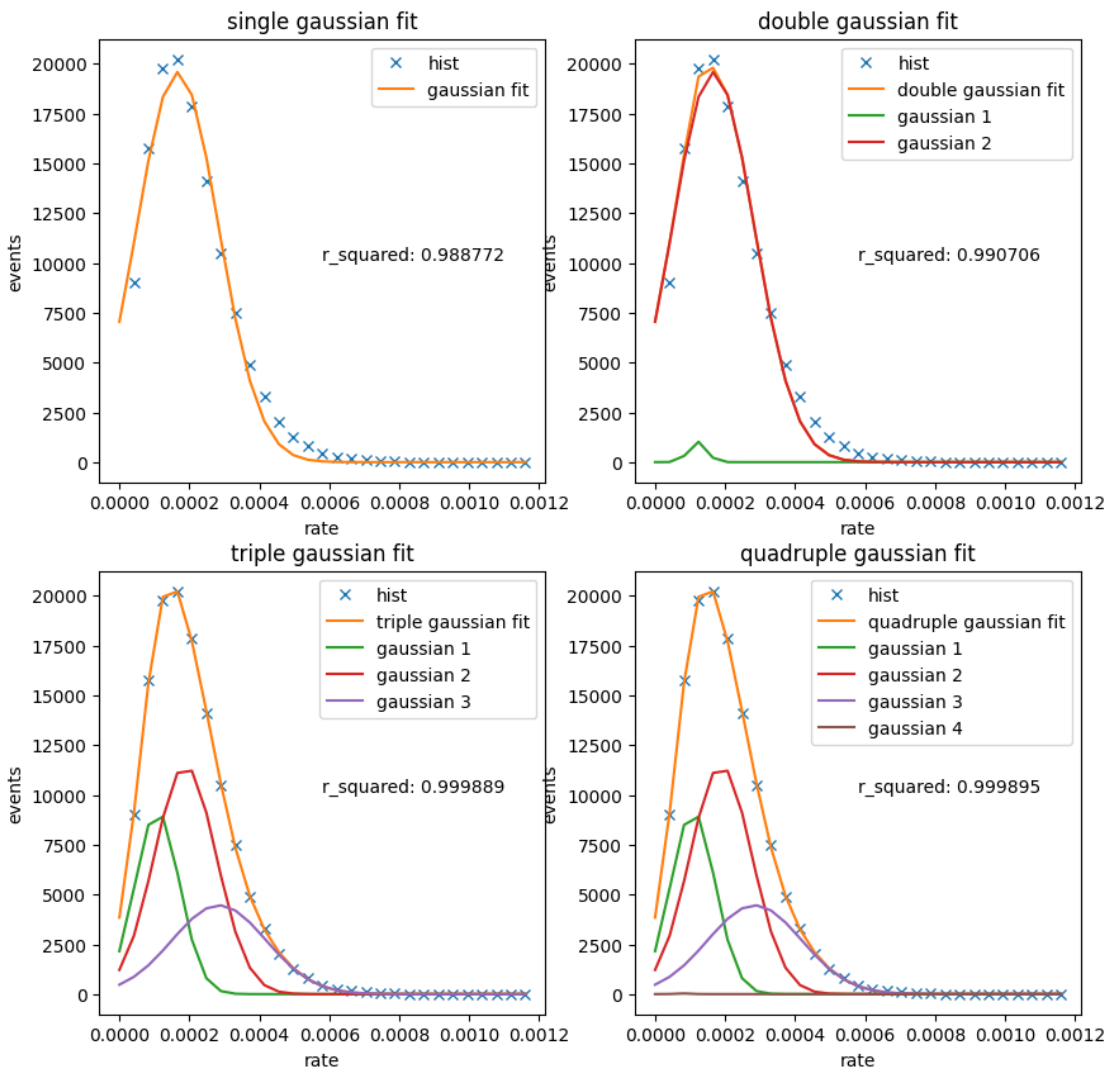
[ ]:
```
# plot single - quadruple gaussian fit in one plot
fig, axs = plt.subplots(2, 2, figsize=(10, 10))
axs = axs.flatten()

popt,_ ,_r_squared= gaussian_fit(background_rates[1:],background_hist[1:])
axs[0].plot(background_rates[1:], background_hist[1:], "x", label="hist")
axs[0].plot(background_rates, gaussian(background_rates, *popt),
    label="gaussian fit")
```

```
axs[0].legend()
axs[0].set_title("single gaussian fit")
axs[0].set_xlabel("rate")
axs[0].set_ylabel("events")
axs[0].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[0].
    ↪transAxes)
print("single gaussian fit")
print("mu: ", popt[0]/BACKGROUND_FRAMETIME)
print("sigma: ", popt[1]/BACKGROUND_FRAMETIME)
print("A: ", popt[2])
print("r_squared: ", _r_squared)


popt, _, _r_squared = double_gaussian_fit(background_rates[1:
    ↪],background_hist[1:])
axs[1].plot(background_rates[1:], background_hist[1:], "x", label="hist")
axs[1].plot(background_rates, double_gaussian(background_rates, *popt),␣
    ↪label="double gaussian fit")
axs[1].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
    ↪popt[2]), label="gaussian 1")
axs[1].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
    ↪popt[5]), label="gaussian 2")
axs[1].legend()
axs[1].set_title("double gaussian fit")
axs[1].set_xlabel("rate")
axs[1].set_ylabel("events")
axs[1].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[1].
    ↪transAxes)
print("double gaussian fit")
print(f"-----|--1--|--2--|")
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = triple_gaussian_fit(background_rates[1:
    ↪],background_hist[1:])
axs[2].plot(background_rates[1:], background_hist[1:], "x", label="hist")
axs[2].plot(background_rates, triple_gaussian(background_rates, *popt),␣
    ↪label="triple gaussian fit")
axs[2].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
    ↪popt[2]), label="gaussian 1")
axs[2].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
    ↪popt[5]), label="gaussian 2")
axs[2].plot(background_rates, gaussian(background_rates, popt[6], popt[7],␣
    ↪popt[8]), label="gaussian 3")
```

```
axs[2].legend()
axs[2].set_title("triple gaussian fit")
axs[2].set_xlabel("rate")
axs[2].set_ylabel("events")
axs[2].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[2].
 ↪transAxes)

print("triple gaussian fit")
print(f"-----|--1--|--2--|--3--|")
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = quadruple_gaussian_fit(background_rates[1:
 ↪],background_hist[1:])
axs[3].plot(background_rates[1:], background_hist[1:], "x", label="hist")
axs[3].plot(background_rates, quadruple_gaussian(background_rates, *popt),␣
 ↪label="quadruple gaussian fit")
axs[3].plot(background_rates, gaussian(background_rates, popt[0], popt[1],␣
 ↪popt[2]), label="gaussian 1")
axs[3].plot(background_rates, gaussian(background_rates, popt[3], popt[4],␣
 ↪popt[5]), label="gaussian 2")
axs[3].plot(background_rates, gaussian(background_rates, popt[6], popt[7],␣
 ↪popt[8]), label="gaussian 3")
axs[3].plot(background_rates, gaussian(background_rates, popt[9], popt[10],␣
 ↪popt[11]), label="gaussian 4")
axs[3].legend()
axs[3].set_title("quadruple gaussian fit")
axs[3].set_xlabel("rate")
axs[3].set_ylabel("events")
axs[3].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[3].
 ↪transAxes)
print("quadruple gaussian fit")
print(f"-----|--1--|--2--|--3--|--4--|")
print(f"-mu--|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|{popt[9]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|{popt[10]:.4g}|")
print(f"-A---|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|{popt[11]:.4g}|")
print("r_squared: ", _r_squared)
```

```
single gaussian fit
mu:  6.900966237650693e-09
sigma:  4.827067240890063e-09
A:  19587.922473229137
r_squared:  0.9887720968985682
double gaussian fit
-----|--1--|--2--|
```

```
-mu--|0.0001216|0.0001666|
-sig-|2.499e-05|0.0001165|
-A---|1027|1.959e+04|
r_squared:  0.9907056996586165
triple gaussian fit
-----|--1--|--2--|--3--|
-mu--|0.0001081|0.0001883|0.0002846|
-sig-|6.35e-05|8.883e-05|0.0001343|
-A---|9199|1.147e+04|4462|
r_squared:  0.9998889497285287
quadruple gaussian fit
-----|--1--|--2--|--3--|--4--|
-mu--|0.0001081|0.0001883|0.0002846|8.141e-05|
-sig-|6.35e-05|8.883e-05|0.0001343|2.231e-05|
-A---|9199|1.147e+04|4462|34.01|
r_squared:  0.9998951149427556
```



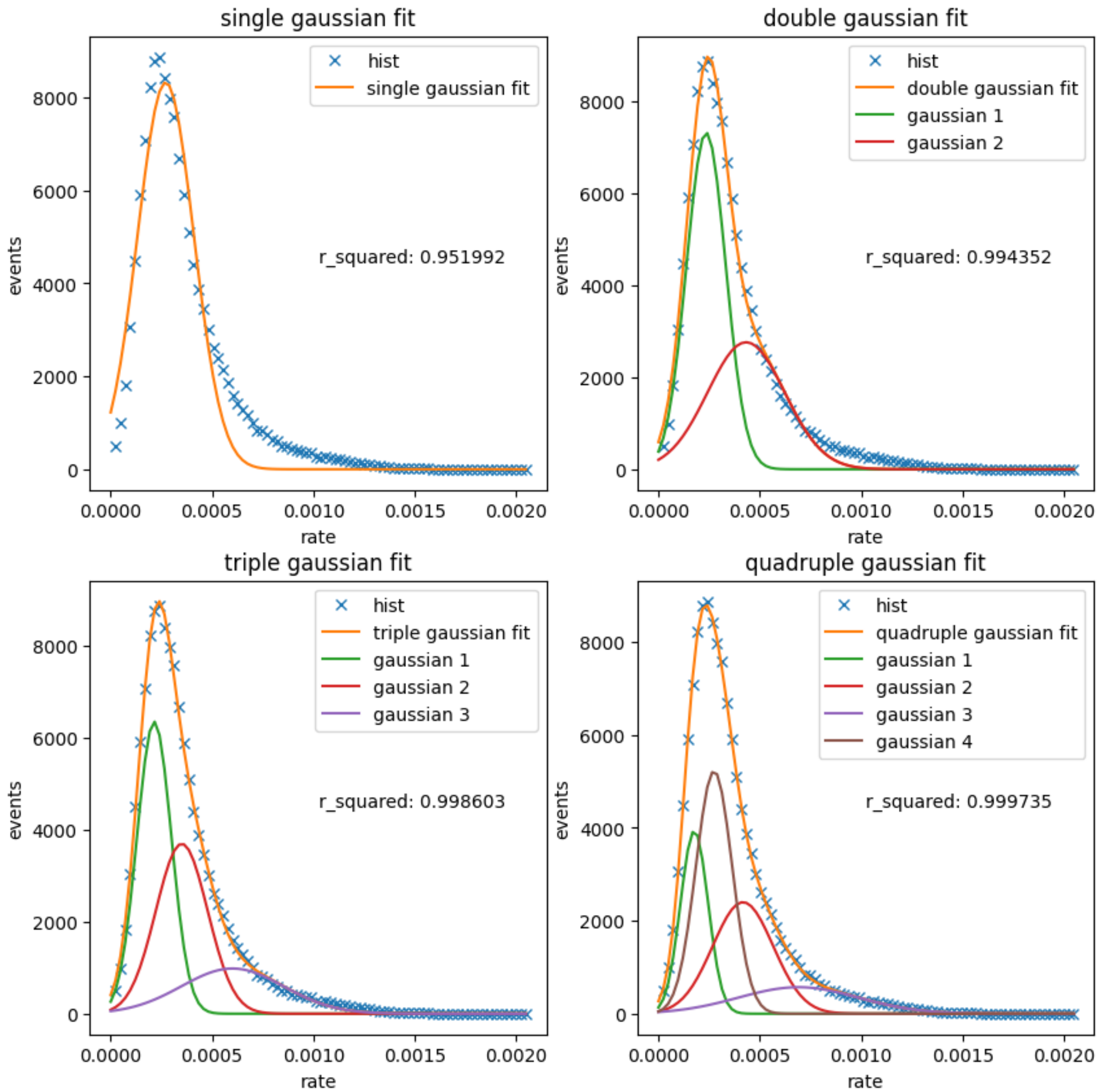
1.3.3 psbeam zero removed

```
[ ]: # plot single - quadruple gaussian fits in one plot

fig, axs = plt.subplots(2,2, figsize=(10,10))
axs=axs.flatten()

popt, _, _r_squared = gaussian_fit(psbeam_rates[1:],psbeam_hist[1:])
axs[0].plot(psbeam_rates[1:], psbeam_hist[1:], "x", label="hist")
axs[0].plot(psbeam_rates, gaussian(psbeam_rates, *popt), label="single gaussian␣
 ↪fit")
axs[0].legend()
axs[0].set_title("single gaussian fit")
axs[0].set_xlabel("rate")
axs[0].set_ylabel("events")
axs[0].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[0].
 ↪transAxes)
print("single gaussian fit")
print(f"-----|--1--|")
print(f"-mu--|{popt[0]:.4g}|")
print(f"-sig-|{popt[1]:.4g}|")
print(f"-A---|{popt[2]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = double_gaussian_fit(psbeam_rates[1:],psbeam_hist[1:])
axs[1].plot(psbeam_rates[1:], psbeam_hist[1:], "x", label="hist")
axs[1].plot(psbeam_rates, double_gaussian(psbeam_rates, *popt), label="double␣
 ↪gaussian fit")
axs[1].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]),␣
 ↪label="gaussian 1")
axs[1].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]),␣
 ↪label="gaussian 2")
axs[1].legend()
axs[1].set_title("double gaussian fit")
axs[1].set_xlabel("rate")
axs[1].set_ylabel("events")
axs[1].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[1].
 ↪transAxes)
print("double gaussian fit")
print(f"-----|--1--|--2--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|")
print("r_squared: ", _r_squared)
```

```
popt, _, _r_squared = triple_gaussian_fit(psbeam_rates[1:],psbeam_hist[1:])
axs[2].plot(psbeam_rates[1:], psbeam_hist[1:], "x", label="hist")
axs[2].plot(psbeam_rates, triple_gaussian(psbeam_rates, *popt), label="triple gaussian fit")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]), label="gaussian 1")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]), label="gaussian 2")
axs[2].plot(psbeam_rates, gaussian(psbeam_rates, popt[6], popt[7], popt[8]), label="gaussian 3")
axs[2].legend()
axs[2].set_title("triple gaussian fit")
axs[2].set_xlabel("rate")
axs[2].set_ylabel("events")
axs[2].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[2].transAxes)
print("triple gaussian fit")
print(f"-----|--1--|--2--|--3--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|")
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|")
print("r_squared: ", _r_squared)

popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates[1:],psbeam_hist[1:])
axs[3].plot(psbeam_rates[1:], psbeam_hist[1:], "x", label="hist")
axs[3].plot(psbeam_rates, quadruple_gaussian(psbeam_rates, *popt), label="quadruple gaussian fit")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]), label="gaussian 1")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]), label="gaussian 2")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[6], popt[7], popt[8]), label="gaussian 3")
axs[3].plot(psbeam_rates, gaussian(psbeam_rates, popt[9], popt[10], popt[11]), label="gaussian 4")
axs[3].legend()
axs[3].set_title("quadruple gaussian fit")
axs[3].set_xlabel("rate")
axs[3].set_ylabel("events")
axs[3].text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=axs[3].transAxes)
print("quadruple gaussian fit")
print(f"-----|--1--|--2--|--3--|--4--|")
print(f"--mu-|{popt[0]:.4g}|{popt[3]:.4g}|{popt[6]:.4g}|{popt[9]:.4g}|")
print(f"sigma|{popt[1]:.4g}|{popt[4]:.4g}|{popt[7]:.4g}|{popt[10]:.4g}|")
```

```python
print(f"--A--|{popt[2]:.4g}|{popt[5]:.4g}|{popt[8]:.4g}|{popt[11]:.4g}|")
print("r_squared: ", _r_squared)
```

```
single gaussian fit
-----|--1--|
-mu--|0.0002712|
-sig-|0.0001385|
-A---|8326|
r_squared:  0.9519915706558384
double gaussian fit
-----|--1--|--2--|
--mu-|0.0002351|0.0004323|
sigma|9.683e-05|0.0001899|
--A--|7327|2765|
r_squared:  0.9943516619893307
triple gaussian fit
-----|--1--|--2--|--3--|
--mu-|0.0002146|0.0003504|0.0006005|
sigma|8.501e-05|0.0001278|0.0002521|
--A--|6356|3699|984|
r_squared:  0.9986027589130935
quadruple gaussian fit
-----|--1--|--2--|--3--|--4--|
--mu-|0.0001769|0.0004168|0.0006905|0.0002742|
sigma|6.805e-05|0.0001492|0.0002967|8.962e-05|
--A--|3937|2400|570.5|5224|
r_squared:  0.9997346081179947
```



2 Bayes theorem

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

$P(noise|data) = \frac{P(data|noise)P(noise)}{P(data)}$

$P(noise|data)$ is the probability of noise at a given data

$P(data|noise)$ is the probability of data at a given noise $= \frac{\#data}{\#noise}$

$P(noise)$ is the probability of noise

$P(data)$ is the probability of rate $= P(data|noise)P(noise) + P(data|signal)P(signal)$

$P(rate|noise)P(noise)$ comes from the background.

3 Normalize for background to obtain a probability distribution wrt rate

3.0.1 extract noise from psbeam

[ ]:
```
# integral from zero to infinity
from scipy.stats import norm

def get_gaussian_fit_integral(popt):

    assert len(popt) %3 == 0
    n_gaussians = len(popt)//3

    integral = 0
    for i in range(n_gaussians):
        integral += popt[3*i+2]*norm.sf(0, loc=popt[3*i], scale=popt[3*i+1])

    return integral
```

To help understand Bayes theorem, we use a quadruple gaussian fit and suppose the noise is the first two guassians.

And as a function of rate

notation: $P[noise|rate](rate)$

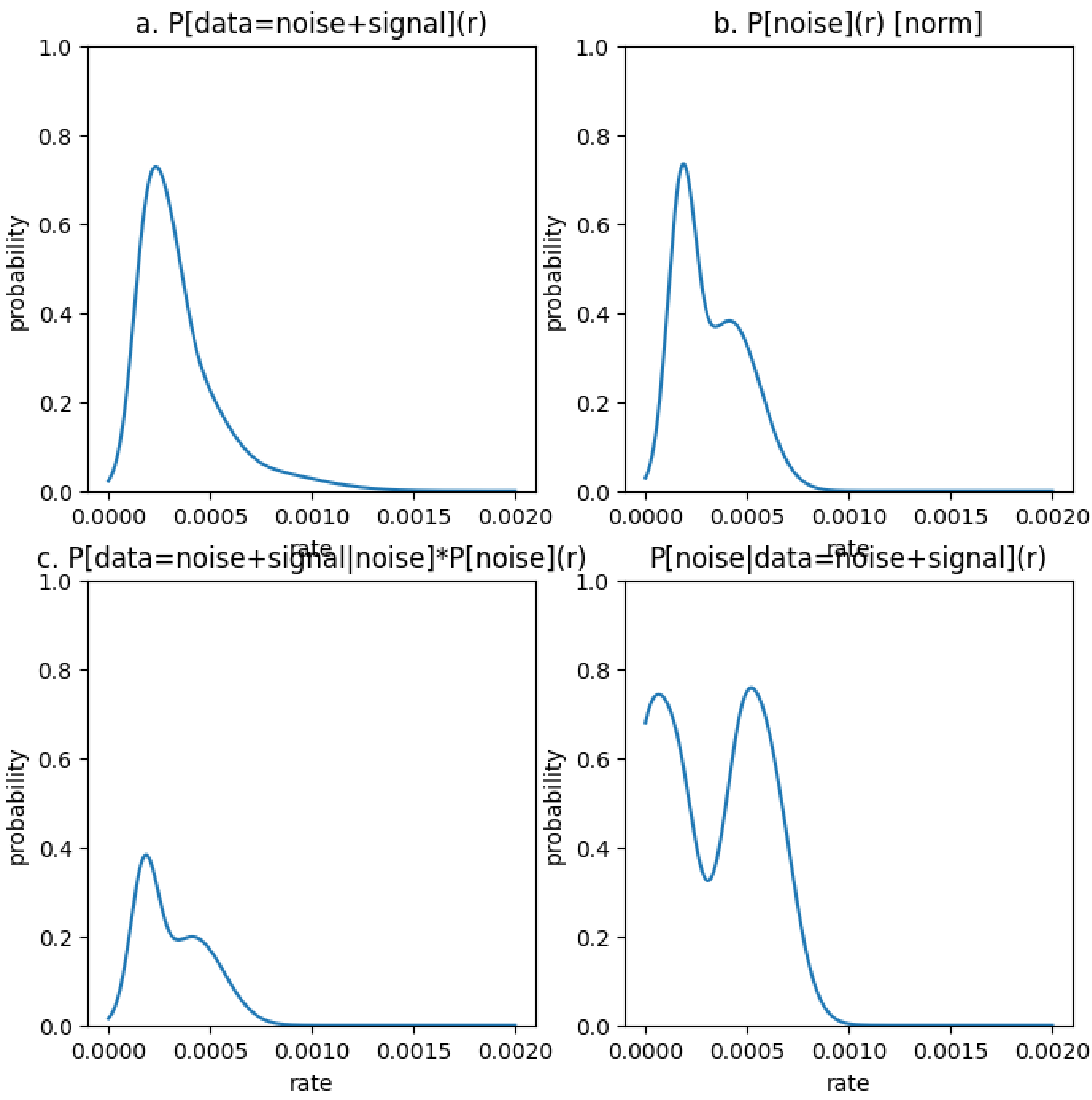
[ ]:
```
# FIXME maybe 5-6 guassians are better



psbeam_quadruple_popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates[1:
 ↪],psbeam_hist[1:])

psbeam_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt)
print(psbeam_integral_via_fit)
normalized_psbeam_probability = lambda rate: quadruple_gaussian(rate,␣
 ↪*psbeam_quadruple_popt)/psbeam_integral_via_fit

noise_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt[:6])
print(noise_integral_via_fit)
normlized_noise_probability = lambda rate: double_gaussian(rate,␣
 ↪*psbeam_quadruple_popt[:6])/noise_integral_via_fit # this must be psbeam␣
 ↪integral, i.e. this is not normalized


noise_given_psbeam_probability = noise_integral_via_fit/psbeam_integral_via_fit
```

```
noise_probability = lambda rate: normlized_noise_probability(rate)/
 ↪normalized_psbeam_probability(rate)*noise_given_psbeam_probability
```

```
12095.828911931076
6312.47334308469
```

[ ]:
```
_continuous_rate = np.linspace(0,0.002,1000)



fig, axs = plt.subplots(2,2, figsize=(8,8))
axs = axs.flatten()
for ax in axs:
    ax.set_xlabel("rate")
    ax.set_ylabel("probability")
    ax.set_ylim(0, 1.0)

axs[0].plot(_continuous_rate, normalized_psbeam_probability(_continuous_rate),␣
 ↪label="psbeam")
axs[0].set_title("a. P[data=noise+signal](r)")
axs[1].plot(_continuous_rate, normlized_noise_probability(_continuous_rate),␣
 ↪label="noise")
axs[1].set_title("b. P[noise](r) [norm]")
axs[2].plot(_continuous_rate,␣
 ↪normlized_noise_probability(_continuous_rate)*noise_given_psbeam_probability,␣
 ↪label="noise")
axs[2].set_title("c. P[data=noise+signal|noise]*P[noise](r)")

axs[3].plot(_continuous_rate, noise_probability(_continuous_rate),␣
 ↪label="noise")
axs[3].set_title("P[noise|data=noise+signal](r)")
```

[ ]:
```
Text(0.5, 1.0, 'P[noise|data=noise+signal](r)')
```



a. is based on the histogram of the data. It shows the probability of a rate, i.e. how likely the rate is inside the image.
b. as we use the first two gaussian fits to represent the noise. this is the normalized probability of being a noise.
c. P[data=noise+signal|noise]*P[noise]
d. is the probability of noise given data.

[ ]:

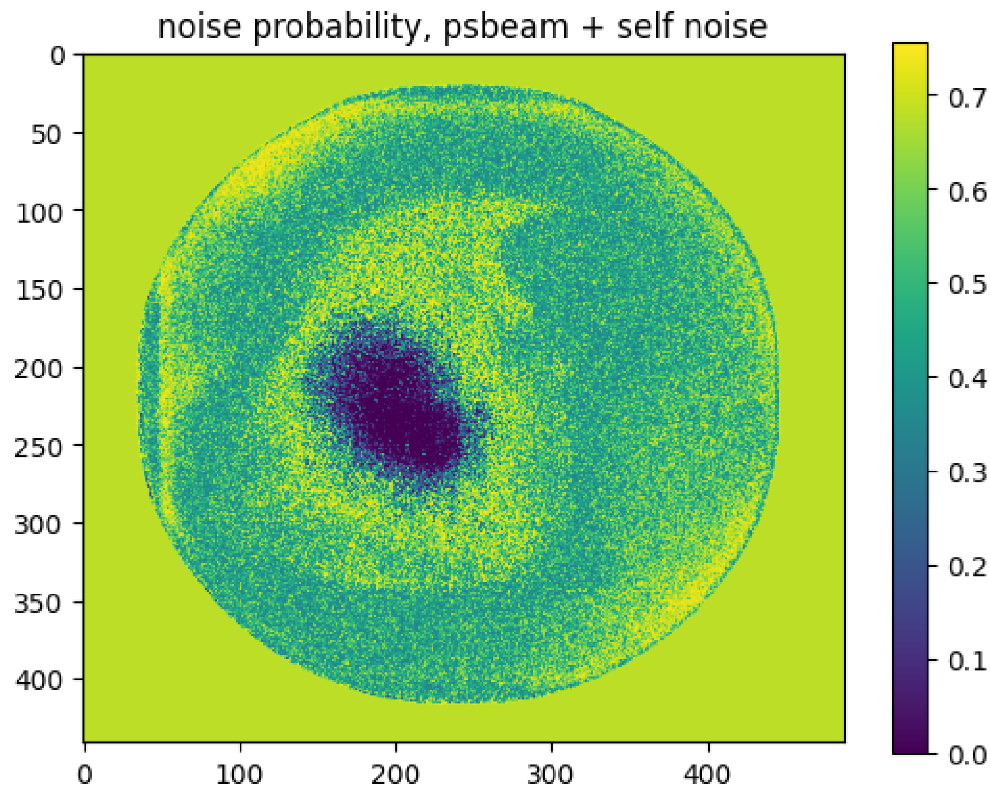
```

```

[ ]:

```
psbeam_matrix_inf_as_zero = np.where(psbeam_matrix == -np.inf, 0, psbeam_matrix)
noise_probability_matrix = noise_probability(psbeam_matrix_inf_as_zero)
plt.imshow(noise_probability_matrix.T, origin="upper",aspect="equal")
```

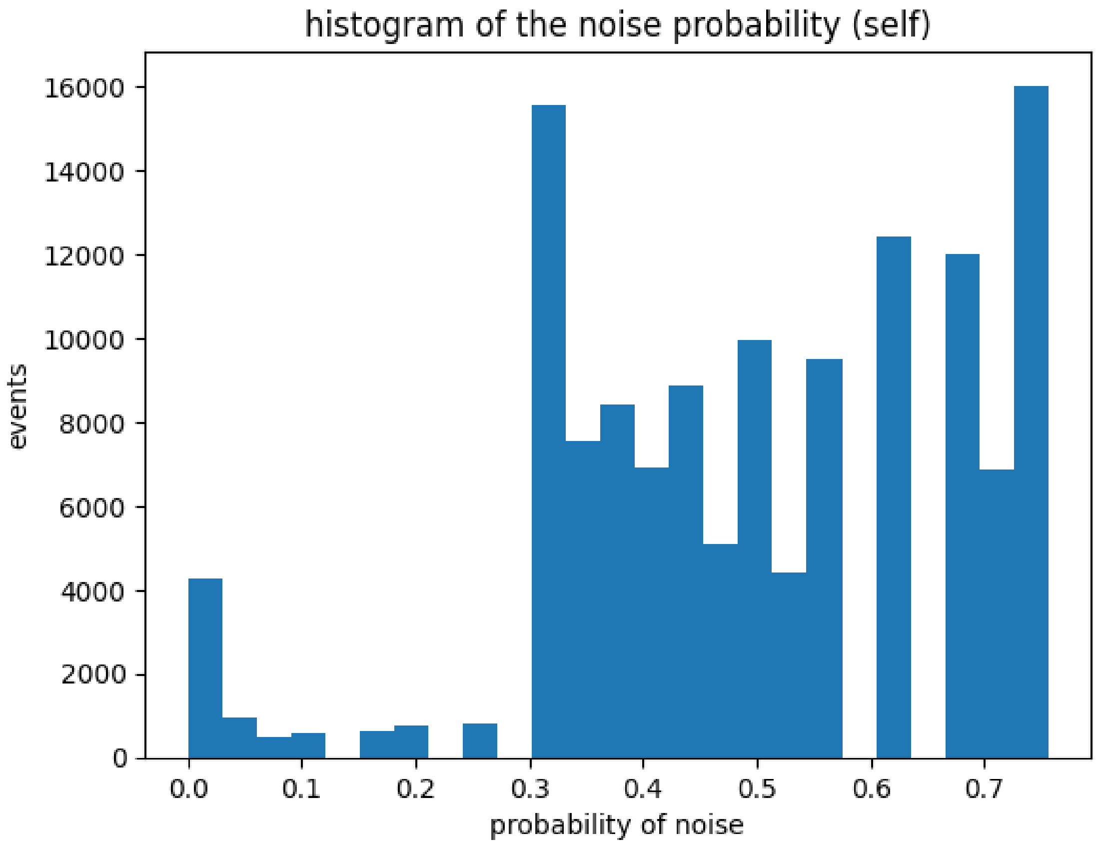
```
plt.colorbar()
plt.title("noise probability, psbeam + self noise")
```

[ ]: Text(0.5, 1.0, 'noise probability, psbeam + self noise')



[ ]:
```
plt.figure()

noise_probability_matrix_neg_inf=np.where(
        outside_circle_boolean_matrix, -np.inf, noise_probability_matrix)
plt.hist(noise_probability_matrix_neg_inf.
 ↪flatten()[noise_probability_matrix_neg_inf.flatten()!=-np.inf],bins=25)
plt.xlabel("probability of noise")
plt.ylabel("events")
plt.title("histogram of the noise probability (self)")
```

[ ]: Text(0.5, 1.0, 'histogram of the noise probability (self)')



[ ]:
```

```

[ ]:
```
# normalize for three gaussian fit
background_triple_popt, _, _r_squared = triple_gaussian_fit(background_rates[1:
 ↪],background_hist[1:])


background_integral_via_fit = get_gaussian_fit_integral(background_triple_popt)

print("background_integral_via_fit: ", background_integral_via_fit)
normlized_background_probability = lambda rate: triple_gaussian(rate,␣
 ↪*background_triple_popt)/background_integral_via_fit
```

```
background_integral_via_fit:  24453.279892751223
```

[ ]:
```
# FIXME maybe 5-6 guassians are better


psbeam_quadruple_popt, _, _r_squared = quadruple_gaussian_fit(
    psbeam_rates[1:], psbeam_hist[1:])
```

```
psbeam_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt)
print(psbeam_integral_via_fit)


def normlized_psbeam_probability(rate): return quadruple_gaussian(
    rate, *psbeam_quadruple_popt)/psbeam_integral_via_fit


noise_given_psbeam = psbeam_integral_via_fit/background_integral_via_fit
print(psbeam_integral_via_fit, background_integral_via_fit, noise_given_psbeam)
```

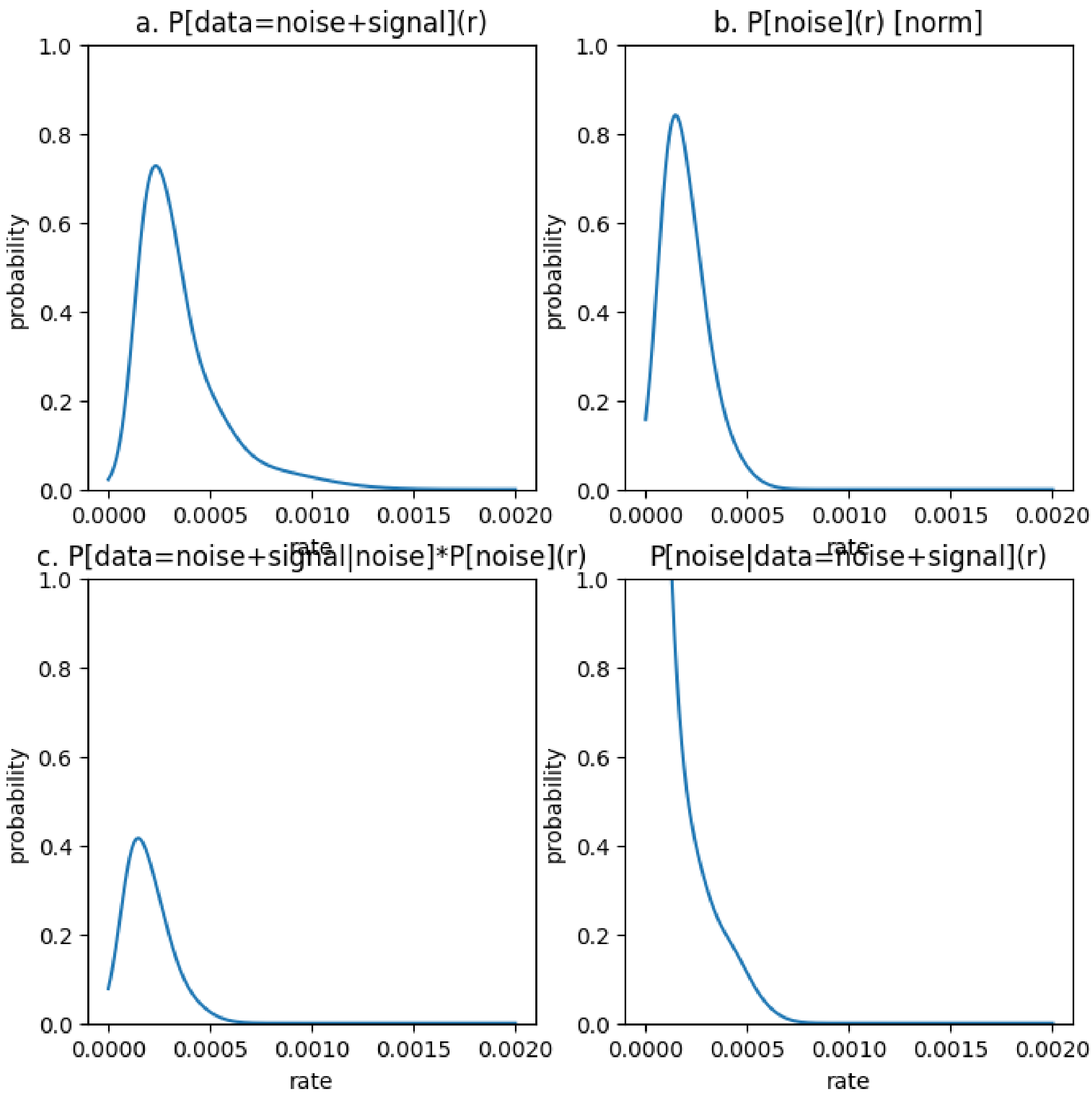
```
12095.828911931076
12095.828911931076 24453.279892751223 0.49465057305121213
```

[ ]:
```
noise_probability = lambda rate:␣
 ↪normlized_background_probability(rate)*noise_given_psbeam/
 ↪normlized_psbeam_probability(rate)
```

[ ]:
```
psbeam_matrix_inf_as_zero = np.where(psbeam_matrix == -np.inf, 0, psbeam_matrix)
noise_probability_matrix = noise_probability(psbeam_matrix_inf_as_zero)
```

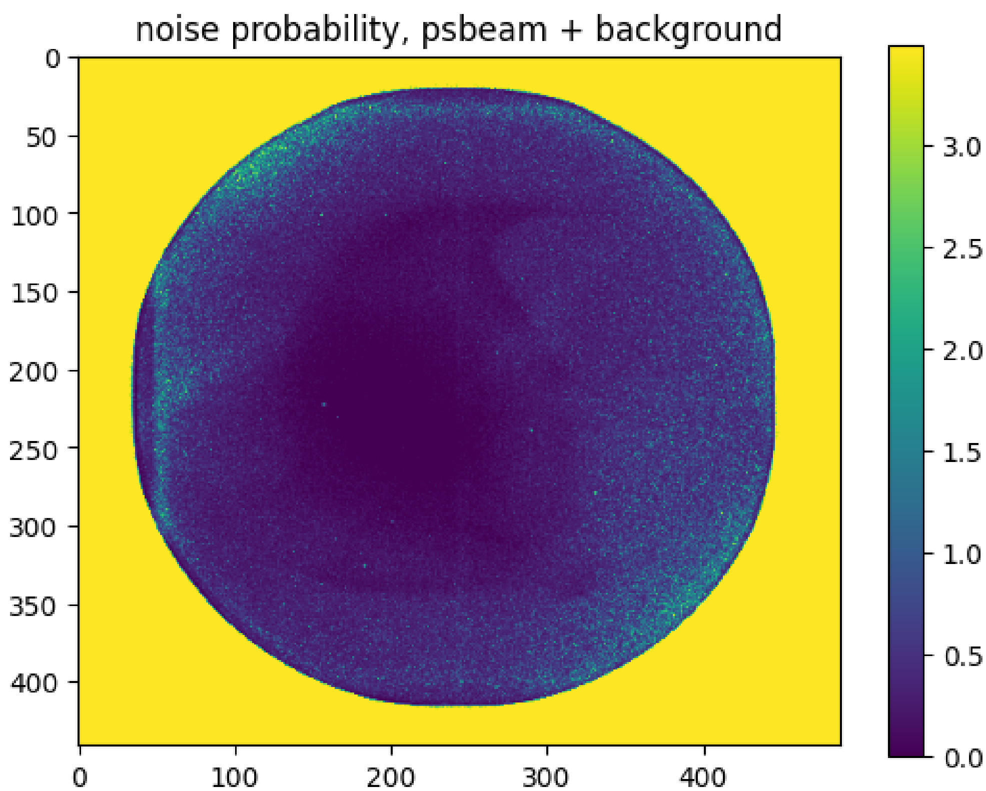
[ ]:
```
_continuous_rate = np.linspace(0,0.002,1000)


fig, axs = plt.subplots(2,2, figsize=(8,8))
axs = axs.flatten()
for ax in axs:
    ax.set_xlabel("rate")
    ax.set_ylabel("probability")
    ax.set_ylim(0, 1.0)

axs[0].plot(_continuous_rate, normalized_psbeam_probability(_continuous_rate),␣
 ↪label="psbeam")
axs[0].set_title("a. P[data=noise+signal](r)")
axs[1].plot(_continuous_rate,␣
 ↪normlized_background_probability(_continuous_rate), label="noise")
axs[1].set_title("b. P[noise](r) [norm]")
axs[2].plot(_continuous_rate,␣
 ↪normlized_background_probability(_continuous_rate)*noise_given_psbeam,␣
 ↪label="noise")
axs[2].set_title("c. P[data=noise+signal|noise]*P[noise](r)")

axs[3].plot(_continuous_rate, noise_probability(_continuous_rate),␣
 ↪label="noise")
axs[3].set_title("P[noise|data=noise+signal](r)")
```

[ ]: Text(0.5, 1.0, 'P[noise|data=noise+signal](r)')



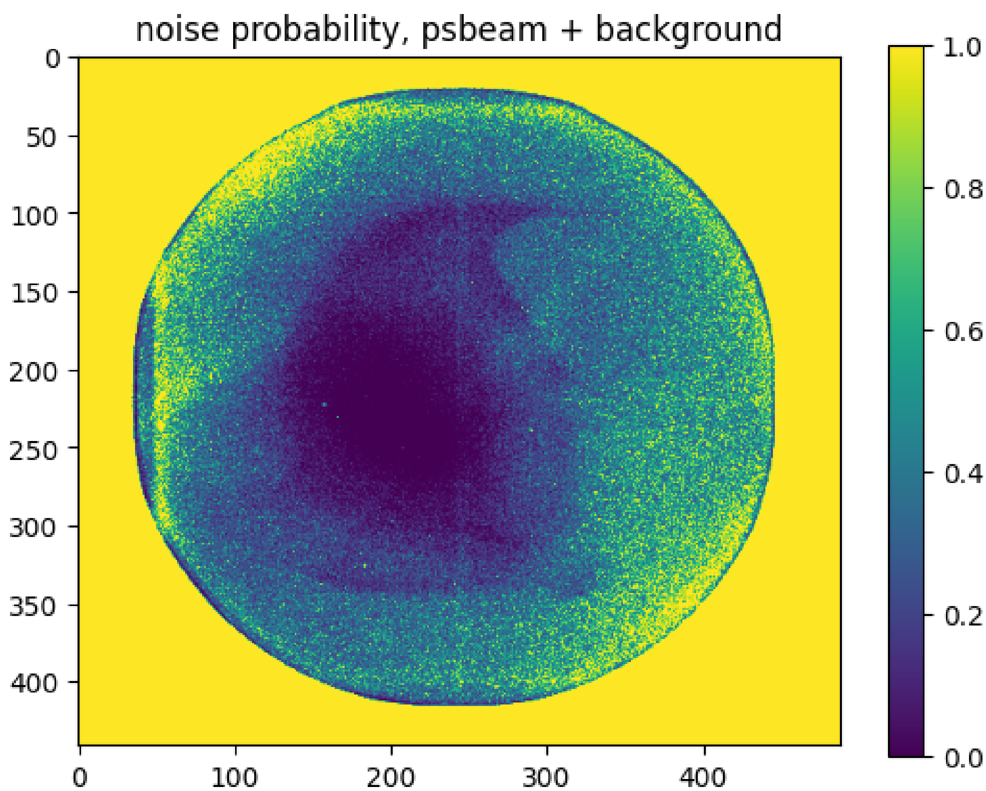
[ ]:
```
plt.imshow(noise_probability_matrix.T, origin="upper",aspect="equal")
plt.colorbar()
plt.title("noise probability, psbeam + background")
```

[ ]: Text(0.5, 1.0, 'noise probability, psbeam + background')



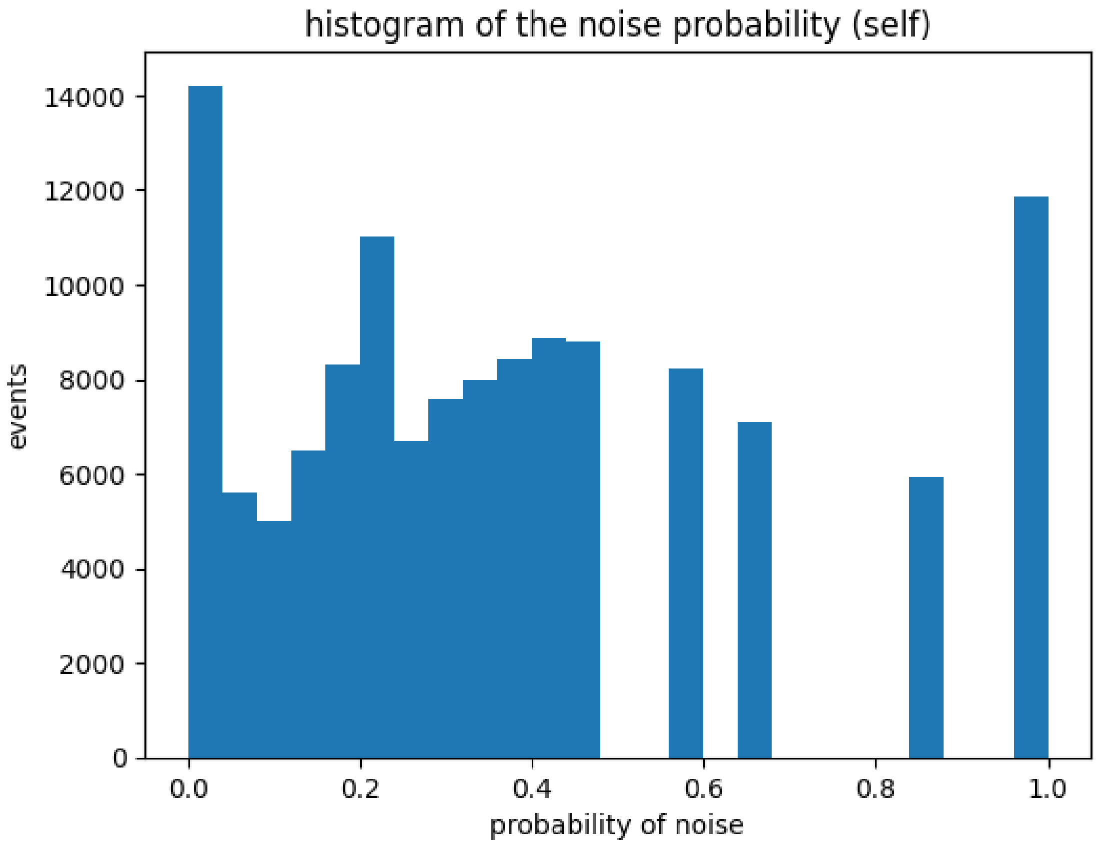
[ ]:
```
plt.imshow(np.where(noise_probability_matrix>1, 1, noise_probability_matrix).T,
    origin="upper",aspect="equal")
plt.colorbar()
plt.title("noise probability, psbeam + background")
```

[ ]: Text(0.5, 1.0, 'noise probability, psbeam + background')



[ ]:
```
plt.figure()

noise_probability_matrix_neg_inf=np.where(
        outside_circle_boolean_matrix, -np.inf, noise_probability_matrix)
noise_probability_matrix_neg_inf = np.where(noise_probability_matrix_neg_inf>1,␣
 ↪1, noise_probability_matrix_neg_inf)
plt.hist(noise_probability_matrix_neg_inf.
 ↪flatten()[noise_probability_matrix_neg_inf.flatten()!=-np.inf],bins=25)
plt.xlabel("probability of noise")
plt.ylabel("events")
plt.title("histogram of the noise probability (self)")
```

[ ]: Text(0.5, 1.0, 'histogram of the noise probability (self)')



[ ]:

```

```

3.0.2 Goodness of fit (to determine the number of gaussian)

[ ]:

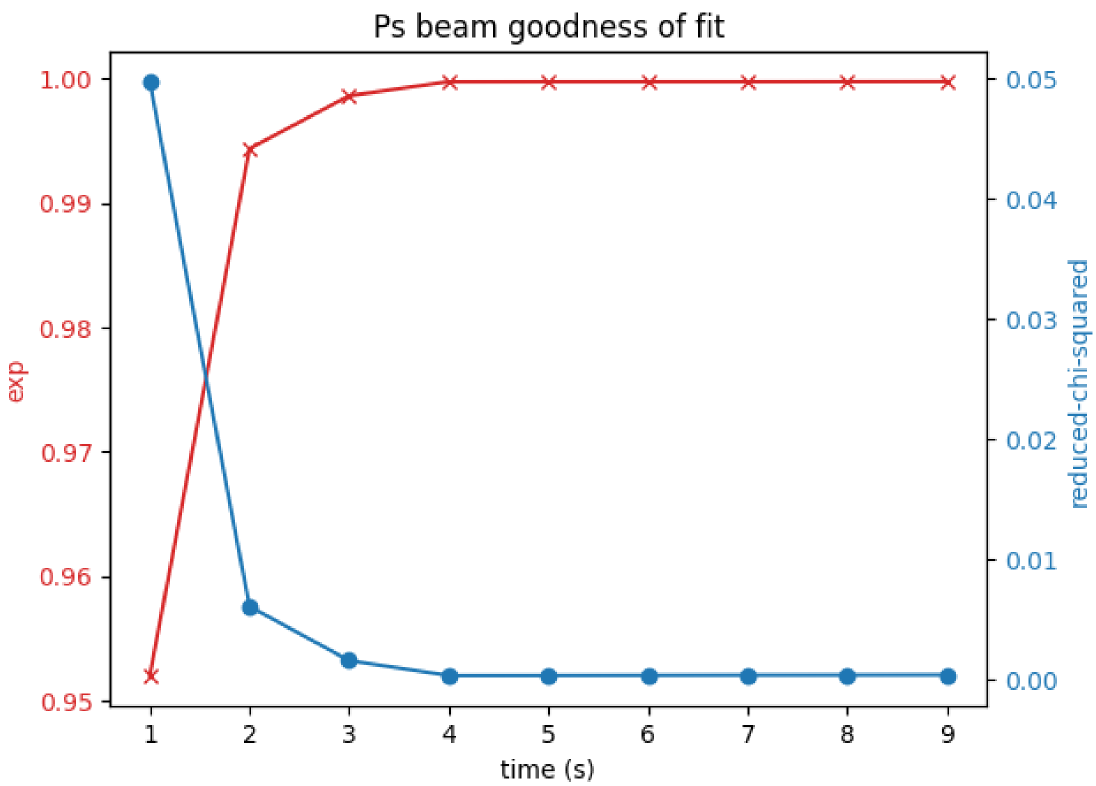
```
def n_gaussian(rate, *args):
    """
    input:
        rate: np.array
        *args: [mu, sigma, A, ...]
    """

    assert len(args) % 3 == 0
    n = len(args)//3

    result = np.zeros_like(rate)
    for i in range(n):
        mu = args[3*i]
        sigma = args[3*i+1]
        A = args[3*i+2]
        result += A*np.exp(-(rate-mu)**2/(2*sigma**2))
```

```python
    return result


def reduced_chi_squared(data, fit, n, p):
    """
    input:
        data: np.array
        fit: np.array
        n: int, number of parameters
        p: int, number of data points
    """
    return np.sum((data-fit)**2)/((p-n)*np.std(data)**2)


def gaussian_fit_1(rate, hist):
    popt, pcov = curve_fit(gaussian, rate, hist, p0=[rate[np.argmax(hist)], 0.
↪0001, np.max(
        hist)], bounds=(np.zeros(3), np.inf * np.ones(3)), maxfev=10000)
    return popt, pcov, r_squared(hist, gaussian(rate, *popt)),␣
↪reduced_chi_squared(hist, gaussian(rate, *popt), 3, len(hist))


def n_gaussian_fit(rate, hist, n: int):
    if n == 1:
        return gaussian_fit_1(rate, hist)

    initial_guess_n_minus_1, _, _, _ = n_gaussian_fit(rate, hist, n-1)

    # use the difference between data and the fit for one gaussian as the␣
↪initial guess for the second gaussian
    difference = hist - n_gaussian(rate, *initial_guess_n_minus_1)
    initial_guess_n, _, _ = gaussian_fit(rate, difference)

    initial_guess = np.concatenate((initial_guess_n_minus_1, initial_guess_n))

    popt, pcov = curve_fit(
        n_gaussian, rate, hist, p0=initial_guess, bounds=(np.zeros(n*3), np.inf␣
↪* np.ones(n*3)), maxfev=10000)

    _r_squared = r_squared(hist, n_gaussian(rate, *popt))

    assert len(popt) == n * 3

    _chi_squared = reduced_chi_squared(
        hist, n_gaussian(rate, *popt), 3*n, len(hist))
    return popt, pcov, _r_squared, _chi_squared
```

[ ]:
```python
N = 10
_r_squareds = []
_chi_squareds = []
print("N r_squared reduced_chi_squared")
for i in range(1, N):
    _, _, _r_squared, _chi_squared = n_gaussian_fit(
        psbeam_rates[1:], psbeam_hist[1:], i)
    print(i, _r_squared, _chi_squared)
    _r_squareds.append(_r_squared)
    _chi_squareds.append(_chi_squared)


fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('time (s)')
ax1.set_ylabel('exp', color=color)
ax1.plot(np.arange(1, N), _r_squareds, "x-", color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
# we already handled the x-label with ax1
ax2.set_ylabel('reduced-chi-squared', color=color)
ax2.plot(np.arange(1, N), _chi_squareds, "o-", color=color)
ax2.tick_params(axis='y', labelcolor=color)


ax1.set_title("Ps beam goodness of fit")
```

```
N r_squared reduced_chi_squared
1 0.9519915706558384 0.049764835295777732
2 0.9943516619893307 0.006077325707682224
3 0.9986027589130935 0.0015627038471981351
4 0.9997346081179947 0.00030901794480065684
5 0.9997387939848299 0.00031717873270651
6 0.9997396294937346 0.00033032079153067173
7 0.9997402372951074 0.0003449973424353936
8 0.9997471620158134 0.0003523152238666589
9 0.9997517822997185 0.00036376731937797386
```

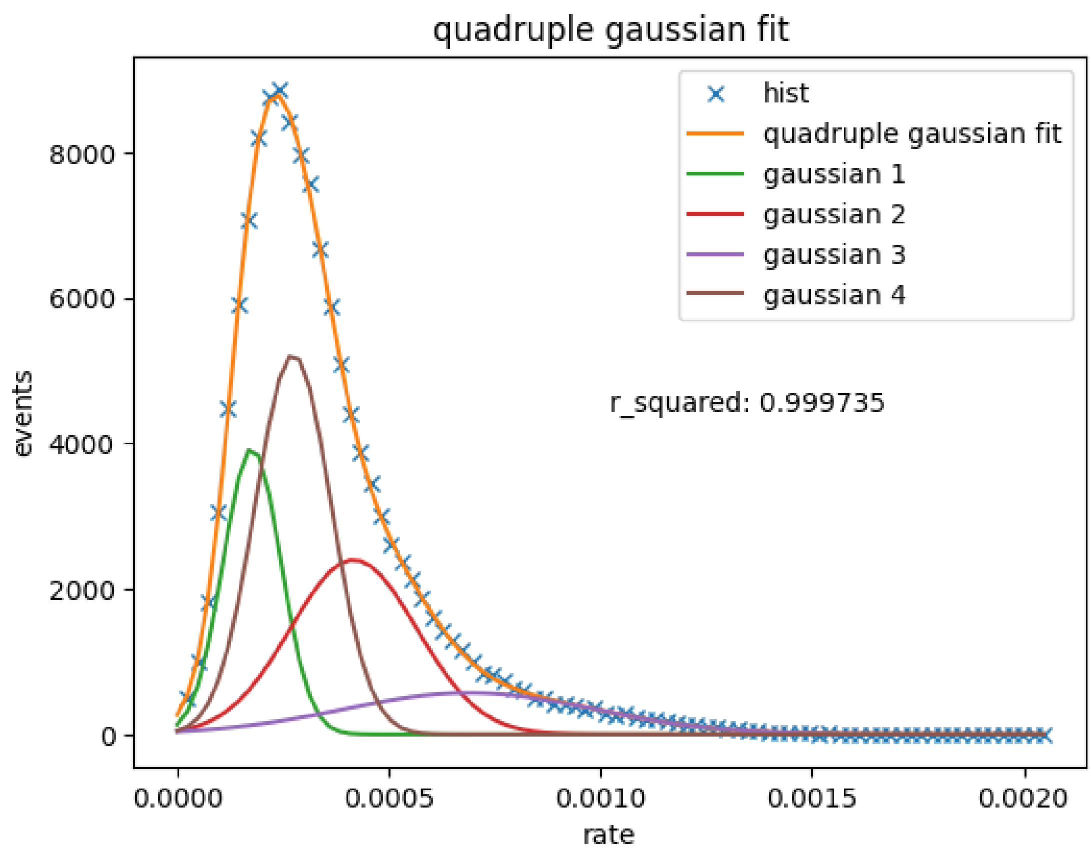
[ ]:
```
Text(0.5, 1.0, 'Ps beam goodness of fit')
```



4 gaussian fit is good because of minimum reduced chi square.

```
[ ]: popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates[1:],psbeam_hist[1:])
fig, ax = plt.subplots()
plt.plot(psbeam_rates[1:], psbeam_hist[1:], "x", label="hist")
plt.plot(psbeam_rates, quadruple_gaussian(psbeam_rates, *popt),␣
 ↪label="quadruple gaussian fit")
plt.plot(psbeam_rates, gaussian(psbeam_rates, popt[0], popt[1], popt[2]),␣
 ↪label="gaussian 1")
plt.plot(psbeam_rates, gaussian(psbeam_rates, popt[3], popt[4], popt[5]),␣
 ↪label="gaussian 2")
plt.plot(psbeam_rates, gaussian(psbeam_rates, popt[6], popt[7], popt[8]),␣
 ↪label="gaussian 3")
plt.plot(psbeam_rates, gaussian(psbeam_rates, popt[9], popt[10], popt[11]),␣
 ↪label="gaussian 4")
plt.legend()
plt.title("quadruple gaussian fit")
plt.xlabel("rate")
plt.ylabel("events")
plt.text(0.5, 0.5, f"r_squared: {_r_squared:.6g}", transform=ax.transAxes)
```

```
[ ]: Text(0.5, 0.5, 'r_squared: 0.999735')
```



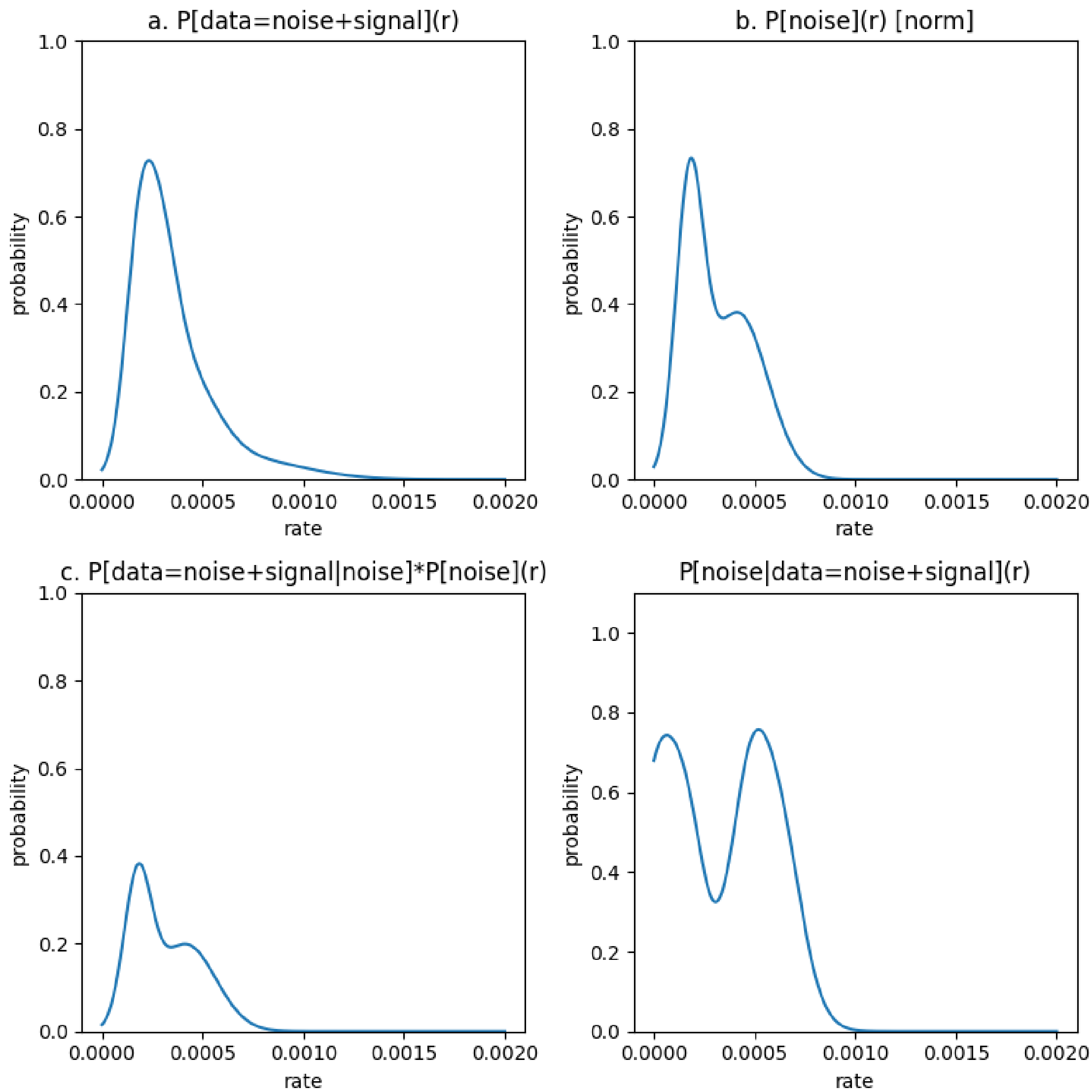
[ ]:
```
psbeam_quadruple_popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates[1:
 ↪],psbeam_hist[1:])

psbeam_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt)
print(psbeam_integral_via_fit)
normalized_psbeam_probability = lambda rate: quadruple_gaussian(rate,␣
 ↪*psbeam_quadruple_popt)/psbeam_integral_via_fit

# index of gaussians
x = [0,1]

# index of gaussian variables
_x = []
for i in x:
    _x.append(3*i)
    _x.append(3*i+1)
    _x.append(3*i+2)
```

```
noise_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt[_x])
print(noise_integral_via_fit)
normlized_noise_probability = lambda rate: n_gaussian(rate,␣
 ↪*psbeam_quadruple_popt[_x])/noise_integral_via_fit # this must be psbeam␣
 ↪integral, i.e. this is not normalized


noise_given_psbeam_probability = noise_integral_via_fit/psbeam_integral_via_fit
noise_probability = lambda rate: normlized_noise_probability(rate)/
 ↪normalized_psbeam_probability(rate)*noise_given_psbeam_probability
```

```
12095.828911931076
6312.47334308469
```

[ ]:

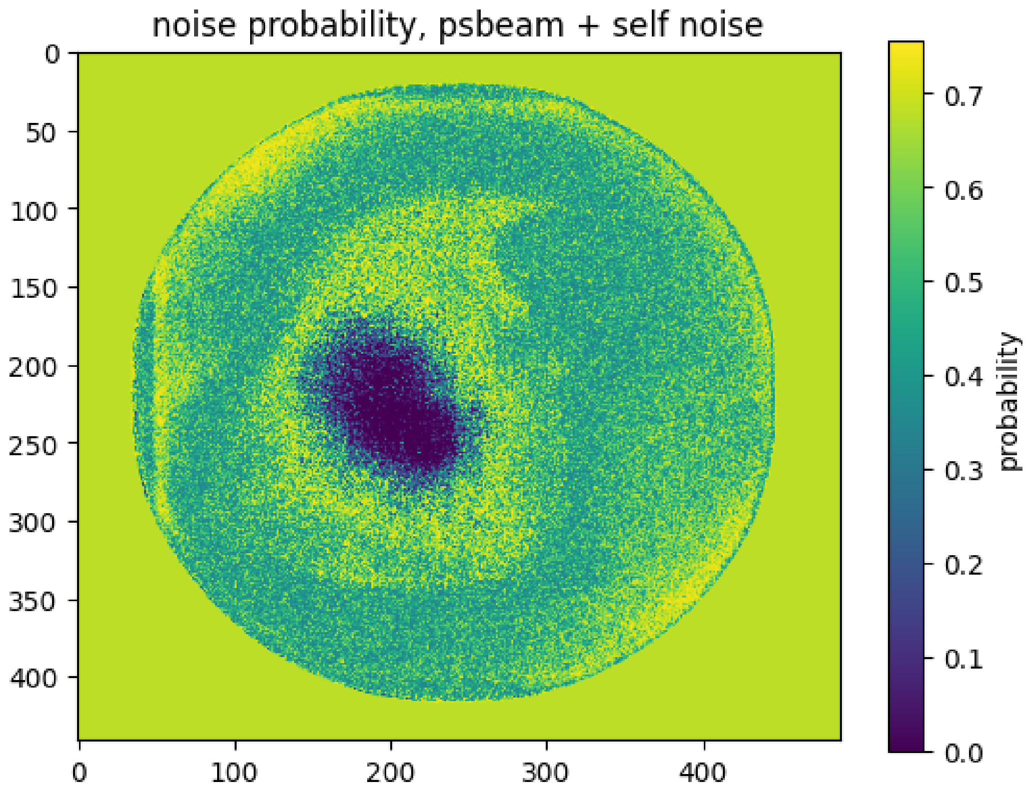
```
_continuous_rate = np.linspace(0,0.002,1000)


fig, axs = plt.subplots(2,2, figsize=(8,8))
axs = axs.flatten()
for ax in axs:
    ax.set_xlabel("rate")
    ax.set_ylabel("probability")
    ax.set_ylim(0, 1.0)

axs[0].plot(_continuous_rate, normalized_psbeam_probability(_continuous_rate),␣
 ↪label="psbeam")
axs[0].set_title("a. P[data=noise+signal](r)")
axs[1].plot(_continuous_rate, normlized_noise_probability(_continuous_rate),␣
 ↪label="noise")
axs[1].set_title("b. P[noise](r) [norm]")
axs[2].plot(_continuous_rate,␣
 ↪normlized_noise_probability(_continuous_rate)*noise_given_psbeam_probability,␣
 ↪label="noise")
axs[2].set_title("c. P[data=noise+signal|noise]*P[noise](r)")

axs[3].plot(_continuous_rate, noise_probability(_continuous_rate),␣
 ↪label="noise")
axs[3].set_title("P[noise|data=noise+signal](r)")
axs[3].set_ylim(0, 1.1)

fig.tight_layout()
```



[ ]:
```
psbeam_matrix_inf_as_zero = np.where(psbeam_matrix == -np.inf, 0, psbeam_matrix)
noise_probability_matrix = noise_probability(psbeam_matrix_inf_as_zero)
```

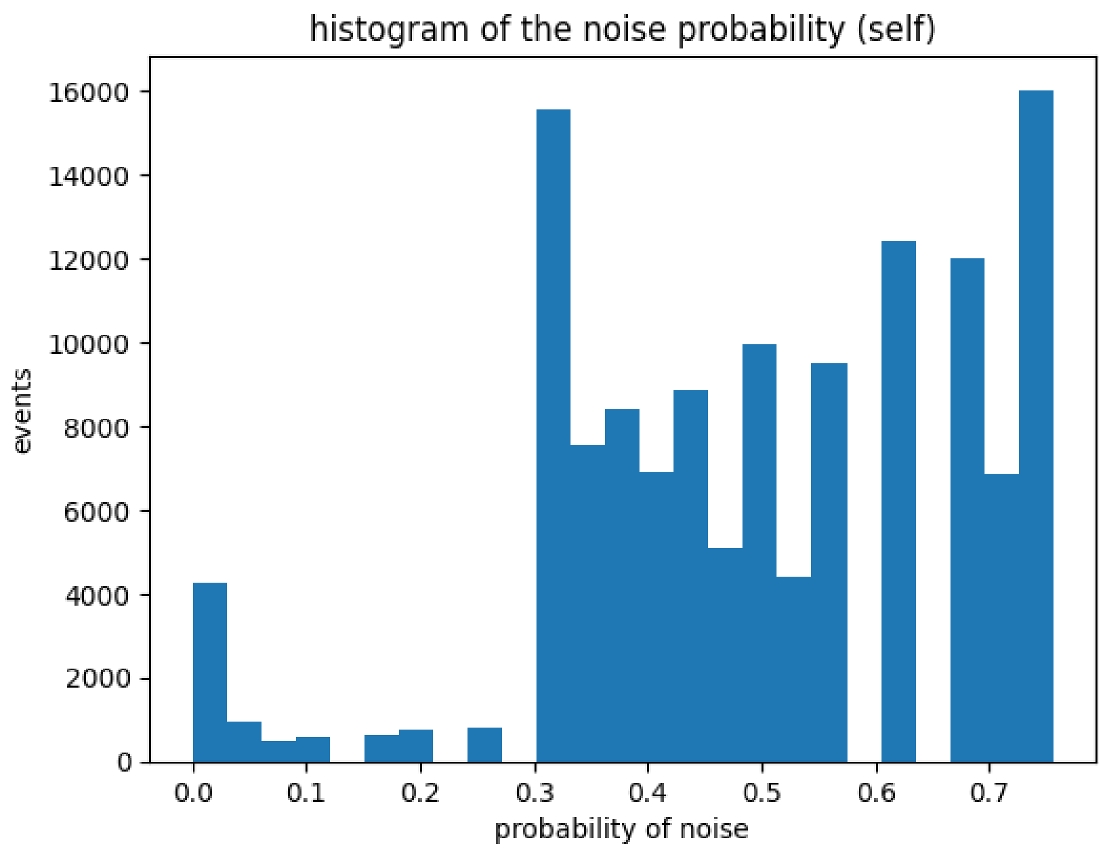
[ ]:
```
plt.imshow(noise_probability_matrix.T, origin="upper",aspect="equal")
plt.colorbar().set_label("probability")
plt.title("noise probability, psbeam + self noise")
```

[ ]:
```
Text(0.5, 1.0, 'noise probability, psbeam + self noise')
```



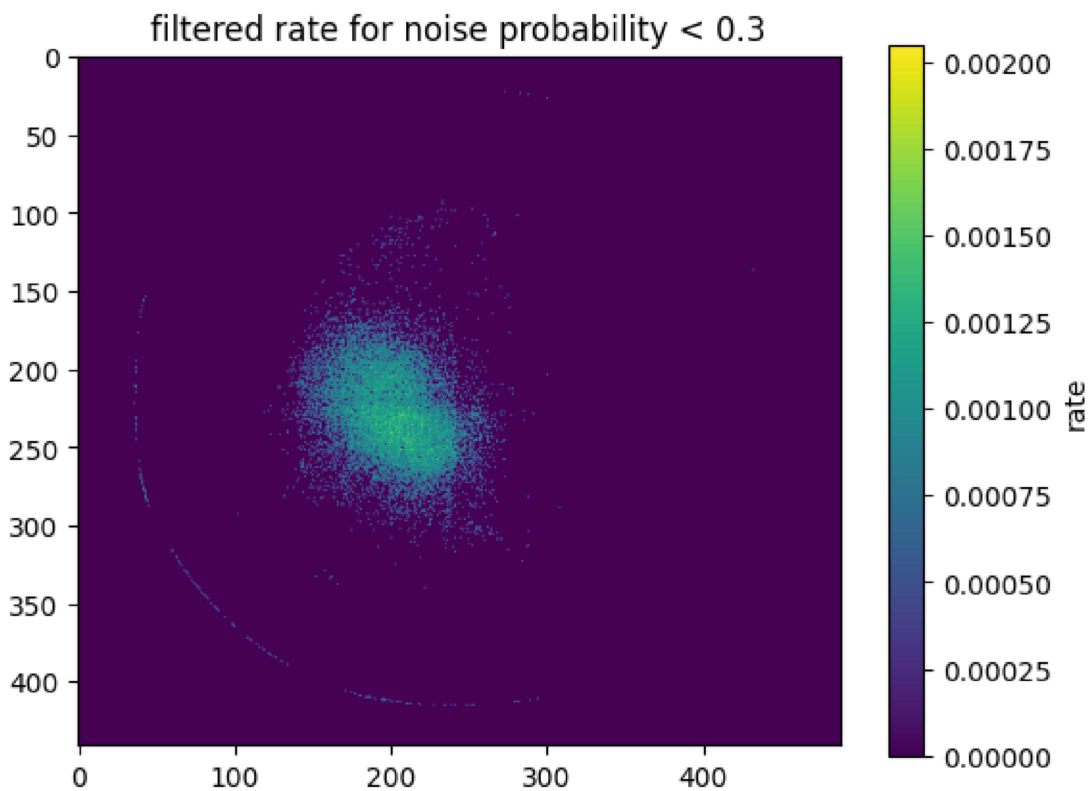
[ ]:
```
plt.figure()

noise_probability_matrix_neg_inf=np.where(
        outside_circle_boolean_matrix, -np.inf, noise_probability_matrix)
plt.hist(noise_probability_matrix_neg_inf.
 ↪flatten()[noise_probability_matrix_neg_inf.flatten()!=-np.inf],bins=25)
plt.xlabel("probability of noise")
plt.ylabel("events")
plt.title("histogram of the noise probability (self)")
```

[ ]:
```
Text(0.5, 1.0, 'histogram of the noise probability (self)')
```



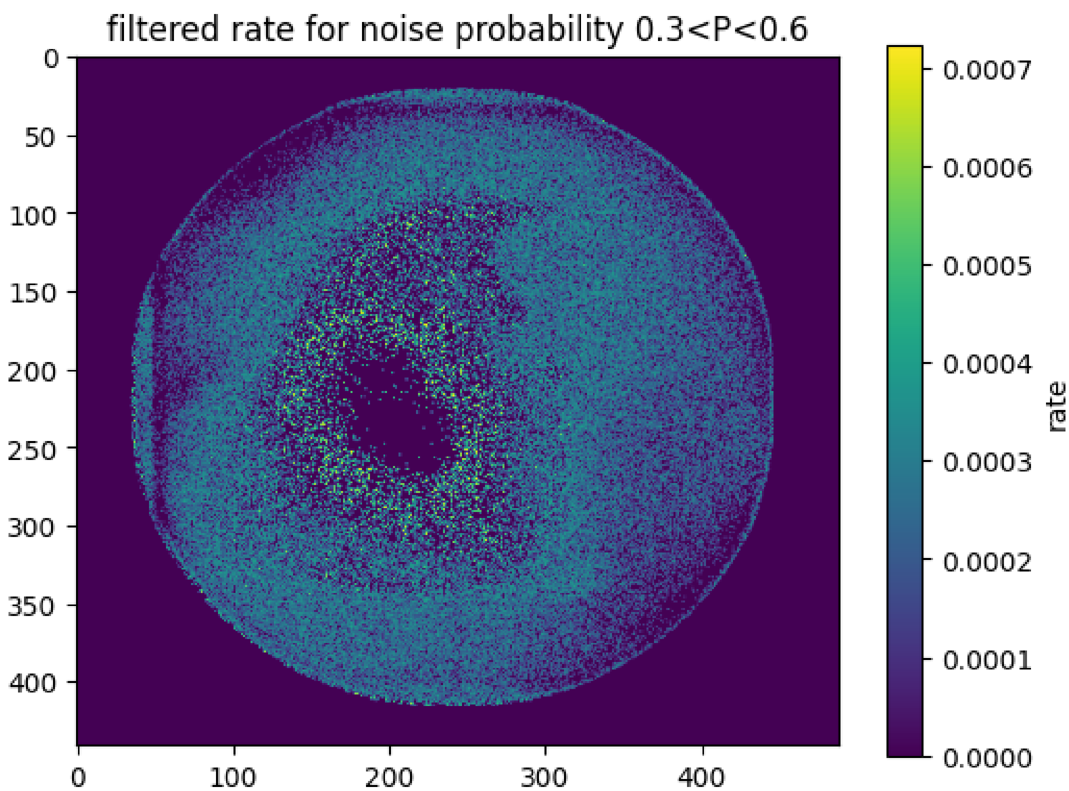
[ ]:
```
filtered_noise_probability_matrix = np.
    ↪logical_and(noise_probability_matrix_neg_inf>0,noise_probability_matrix_neg_inf<0.
    ↪3)

filtered_rate = np.where(filtered_noise_probability_matrix, psbeam_matrix, 0)

plt.figure()
plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
plt.colorbar().set_label("rate")
plt.title("filtered rate for noise probability < 0.3")


print(f"number of pixels with noise probability < 0.3: {np.
    ↪sum(filtered_noise_probability_matrix)}")
print(f"integral rate for noise probability < 0.3: {np.sum(filtered_rate)}")
```

```
number of pixels with noise probability < 0.3: 8491
integral rate for noise probability < 0.3: 8.040308210931855
```



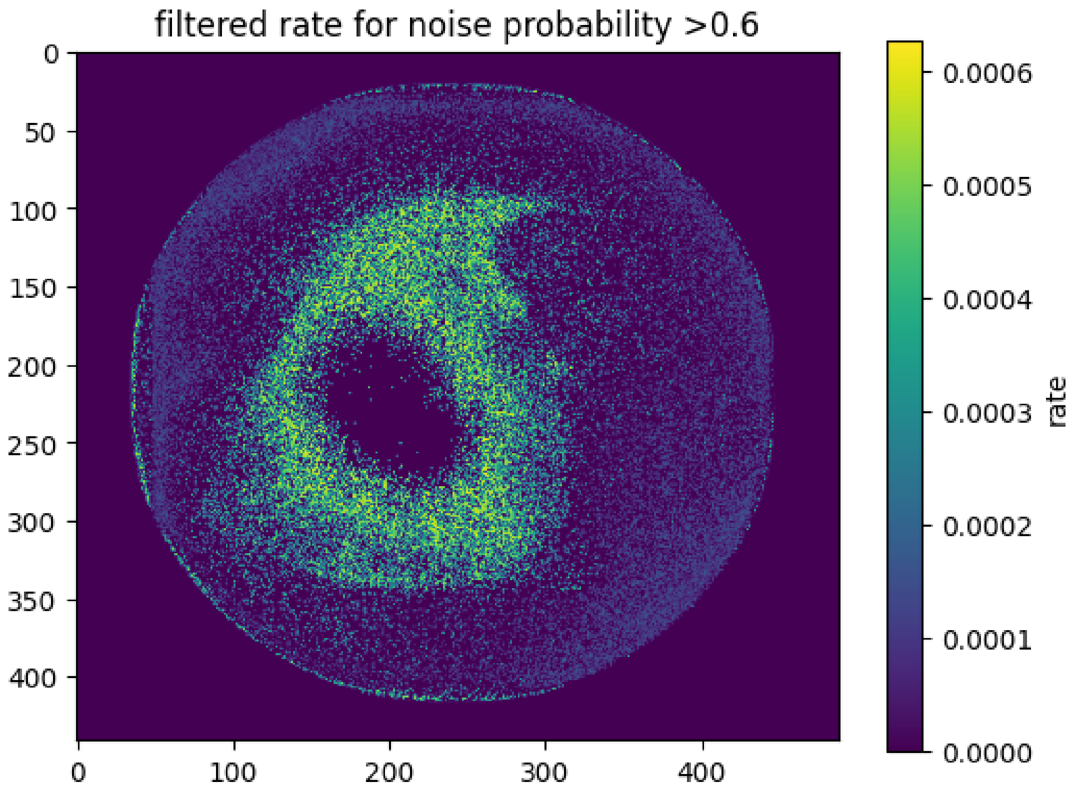

```
[ ]: filtered_noise_probability_matrix = np.
      ↪logical_and(noise_probability_matrix_neg_inf>0.
      ↪3,noise_probability_matrix_neg_inf<0.6)

     filtered_rate = np.where(filtered_noise_probability_matrix, psbeam_matrix, 0)

     plt.figure()
     plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
     plt.colorbar().set_label("rate")
     plt.title("filtered rate for noise probability 0.3<P<0.6")


     print(f"number of pixels with noise probability 0.3<P<0.6: {np.
      ↪sum(filtered_noise_probability_matrix)}")
     print(f"integral rate for noise probability 0.3<P<0.6: {np.sum(filtered_rate)}")
```

```
number of pixels with noise probability 0.3<P<0.6: 76214
integral rate for noise probability 0.3<P<0.6: 23.60146881772213
```



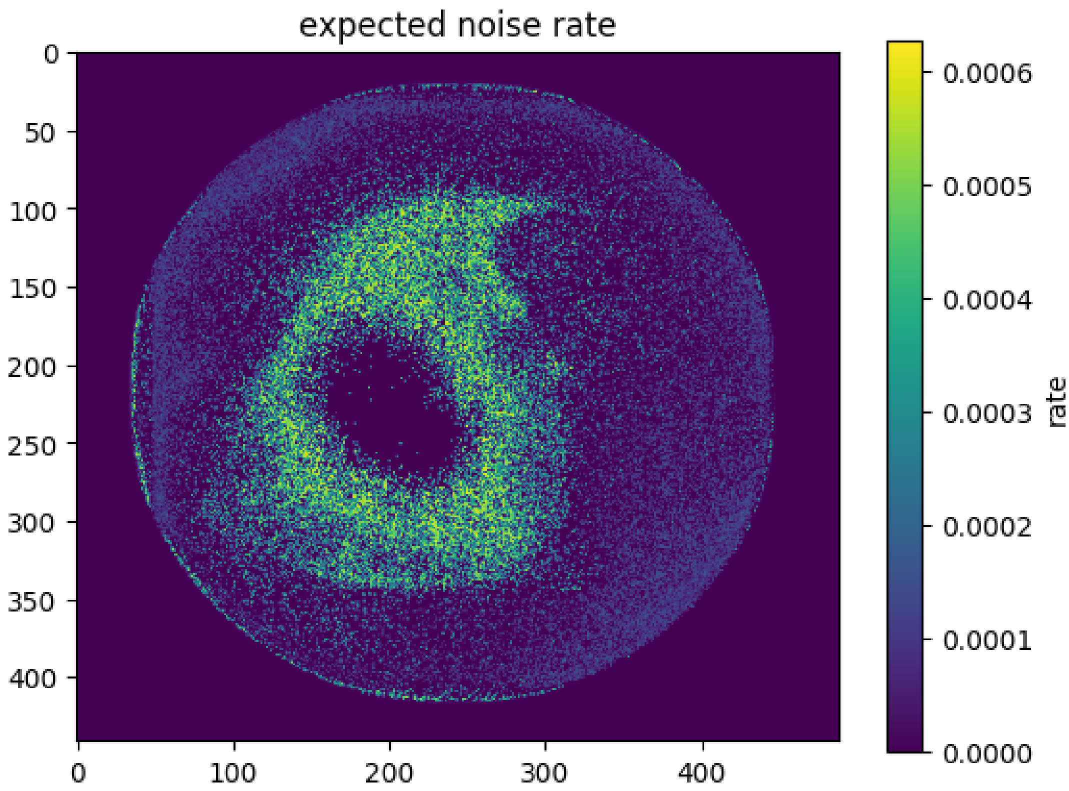
[ ]:
```
filtered_noise_probability_matrix = np.
    ↪logical_and(noise_probability_matrix_neg_inf>0.
    ↪6,noise_probability_matrix_neg_inf<=1)

filtered_rate = np.where(filtered_noise_probability_matrix, psbeam_matrix, 0)

plt.figure()
plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
plt.colorbar().set_label("rate")
plt.title("filtered rate for noise probability >0.6")


print(f"number of pixels with noise probability >0.6: {np.
    ↪sum(filtered_noise_probability_matrix)}")
print(f"integral rate for noise probability >0.6: {np.sum(filtered_rate)}")
```

```
number of pixels with noise probability >0.6: 47268
integral rate for noise probability >0.6: 14.500890922224896
```



[ ]:
```
expected_noise = np.multiply(noise_probability_matrix_neg_inf, psbeam_matrix)
plt.figure()
plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
plt.colorbar().set_label("rate")
plt.title("expected noise rate")
```

[ ]:
```
Text(0.5, 1.0, 'expected noise rate')
```



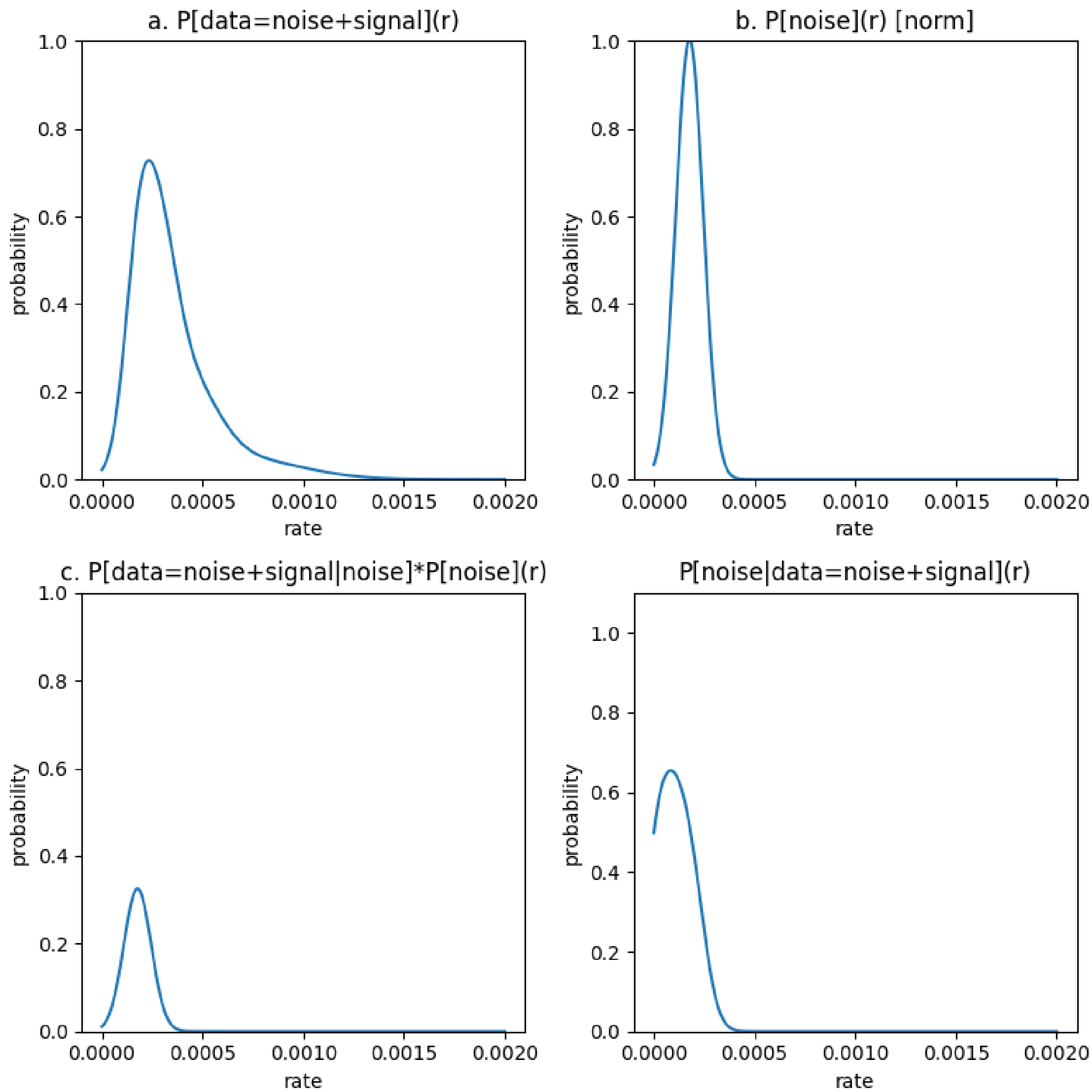
4 change number of gaussians for noise

[ ]:
```
psbeam_quadruple_popt, _, _r_squared = quadruple_gaussian_fit(psbeam_rates[1:
↪],psbeam_hist[1:])

psbeam_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt)
print(psbeam_integral_via_fit)
normalized_psbeam_probability = lambda rate: quadruple_gaussian(rate,␣
↪*psbeam_quadruple_popt)/psbeam_integral_via_fit

# index of gaussians
x = [0]

# index of gaussian variables
_x = []
for i in x:
    _x.append(3*i)
    _x.append(3*i+1)
    _x.append(3*i+2)
```

```
noise_integral_via_fit = get_gaussian_fit_integral(psbeam_quadruple_popt[_x])
print(noise_integral_via_fit)
normlized_noise_probability = lambda rate: n_gaussian(rate,␣
↪*psbeam_quadruple_popt[_x])/noise_integral_via_fit # this must be psbeam␣
↪integral, i.e. this is not normalized


noise_given_psbeam_probability = noise_integral_via_fit/psbeam_integral_via_fit
noise_probability = lambda rate: normlized_noise_probability(rate)/
↪normalized_psbeam_probability(rate)*noise_given_psbeam_probability
```

```
12095.828911931076
3918.911465947577
```

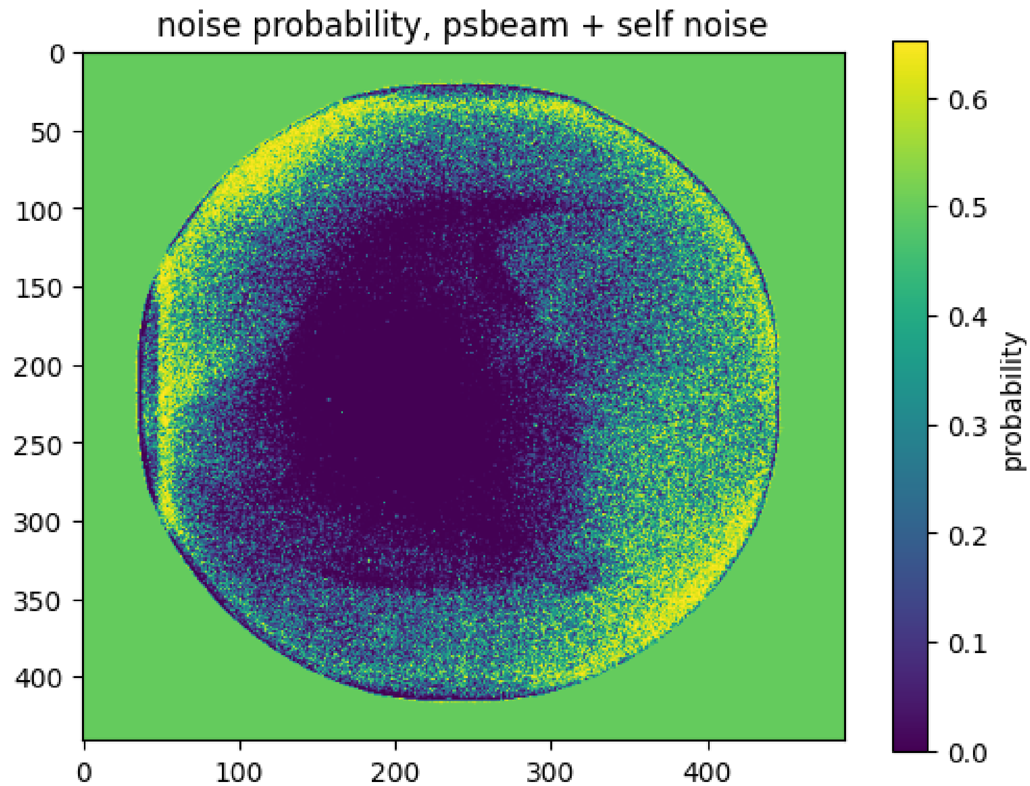
[ ]:
```
_continuous_rate = np.linspace(0,0.002,1000)


fig, axs = plt.subplots(2,2, figsize=(8,8))
axs = axs.flatten()
for ax in axs:
    ax.set_xlabel("rate")
    ax.set_ylabel("probability")
    ax.set_ylim(0, 1.0)

axs[0].plot(_continuous_rate, normalized_psbeam_probability(_continuous_rate),␣
↪label="psbeam")
axs[0].set_title("a. P[data=noise+signal](r)")
axs[1].plot(_continuous_rate, normlized_noise_probability(_continuous_rate),␣
↪label="noise")
axs[1].set_title("b. P[noise](r) [norm]")
axs[2].plot(_continuous_rate,␣
↪normlized_noise_probability(_continuous_rate)*noise_given_psbeam_probability,␣
↪label="noise")
axs[2].set_title("c. P[data=noise+signal|noise]*P[noise](r)")

axs[3].plot(_continuous_rate, noise_probability(_continuous_rate),␣
↪label="noise")
axs[3].set_title("P[noise|data=noise+signal](r)")
axs[3].set_ylim(0, 1.1)

fig.tight_layout()
```



[ ]:
```
psbeam_matrix_inf_as_zero = np.where(psbeam_matrix == -np.inf, 0, psbeam_matrix)
noise_probability_matrix = noise_probability(psbeam_matrix_inf_as_zero)
```

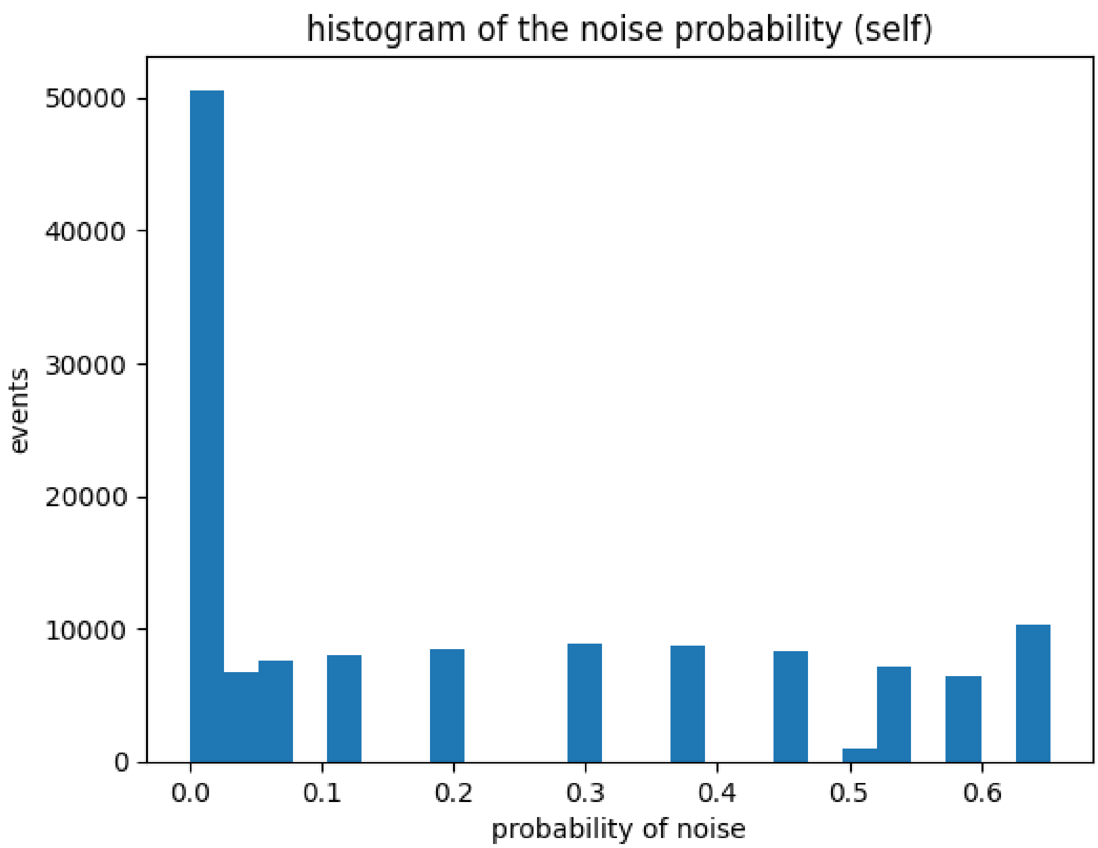
[ ]:
```
plt.imshow(noise_probability_matrix.T, origin="upper",aspect="equal")
plt.colorbar().set_label("probability")
plt.title("noise probability, psbeam + self noise")
```

[ ]:
```
Text(0.5, 1.0, 'noise probability, psbeam + self noise')
```



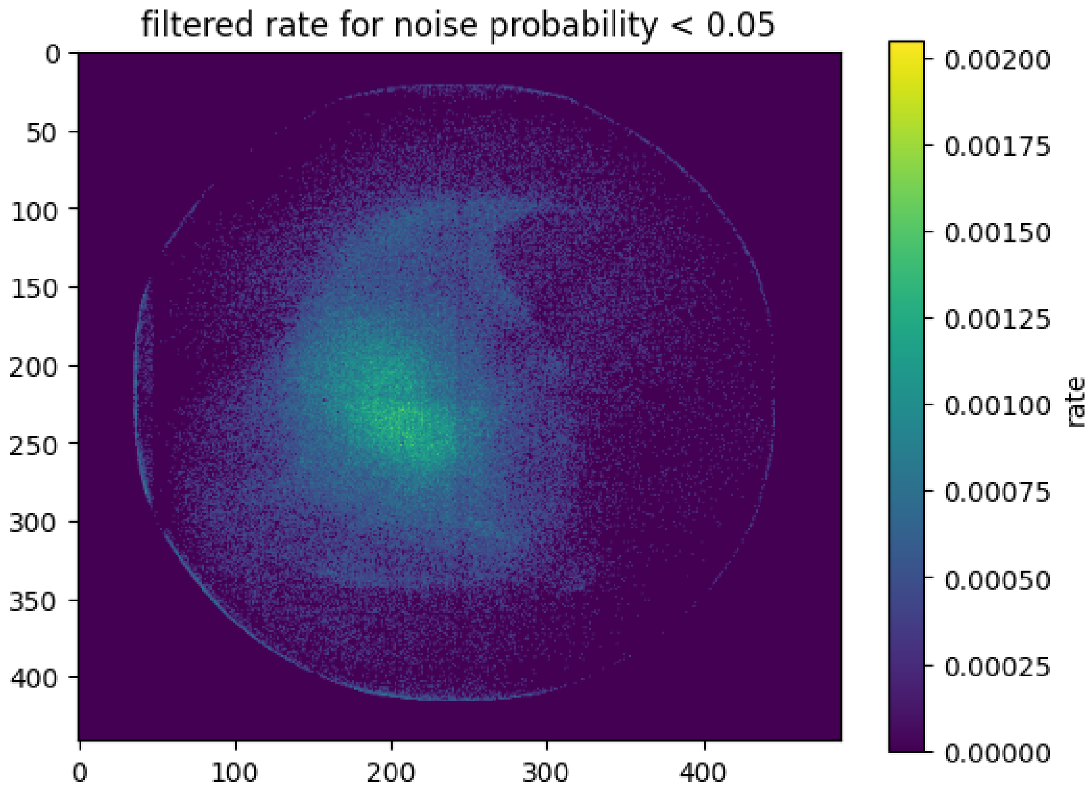
[ ]:
```
plt.figure()

noise_probability_matrix_neg_inf=np.where(
        outside_circle_boolean_matrix, -np.inf, noise_probability_matrix)
plt.hist(noise_probability_matrix_neg_inf.
 ↪flatten()[noise_probability_matrix_neg_inf.flatten()!=-np.inf],bins=25)
plt.xlabel("probability of noise")
plt.ylabel("events")
plt.title("histogram of the noise probability (self)")
```

[ ]: Text(0.5, 1.0, 'histogram of the noise probability (self)')



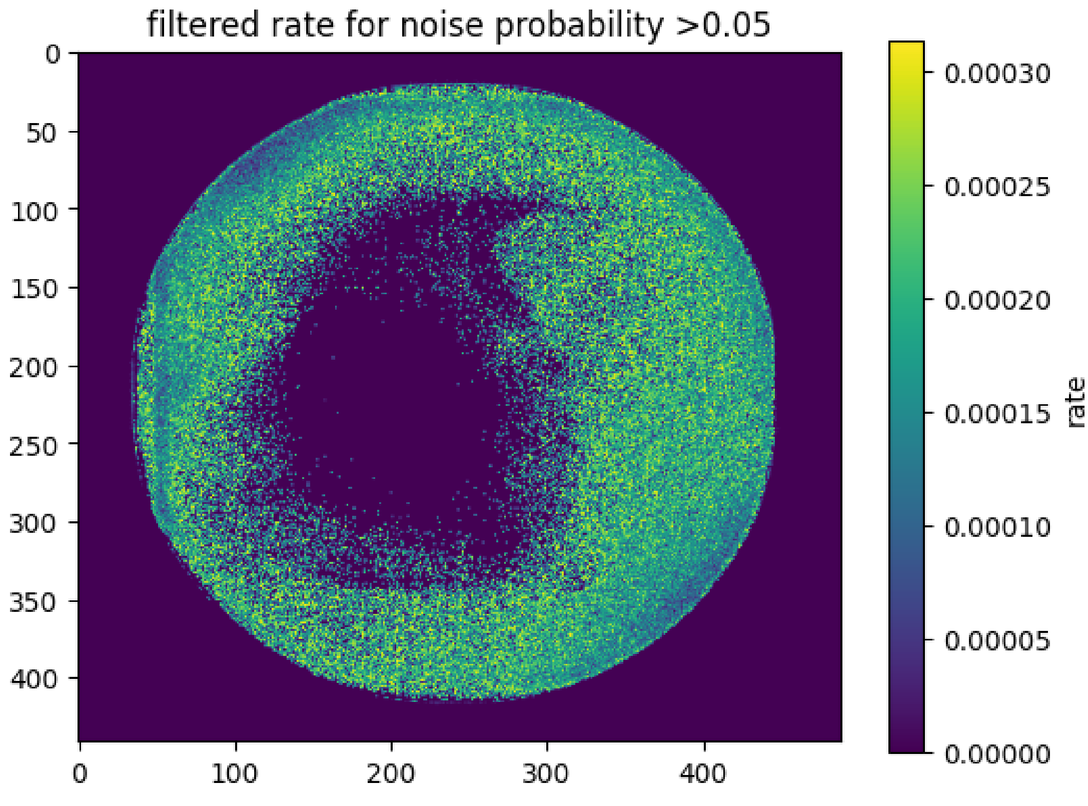
[ ]:
```
filtered_noise_probability_matrix = np.
  ↪logical_and(noise_probability_matrix_neg_inf>0,noise_probability_matrix_neg_inf<0.
  ↪05)

filtered_rate = np.where(filtered_noise_probability_matrix, psbeam_matrix, 0)

plt.figure()
plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
plt.colorbar().set_label("rate")
plt.title("filtered rate for noise probability < 0.05")


print(f"number of pixels with noise probability < 0.05: {np.
  ↪sum(filtered_noise_probability_matrix)}")
print(f"integral rate for noise probability < 0.05: {np.sum(filtered_rate)}")
```

```
number of pixels with noise probability < 0.05: 57256
integral rate for noise probability < 0.05: 30.539345051769807
```



[ ]:
```
filtered_noise_probability_matrix = np.
    ↪logical_and(noise_probability_matrix_neg_inf>0.
    ↪05,noise_probability_matrix_neg_inf<1.01)

filtered_rate = np.where(filtered_noise_probability_matrix, psbeam_matrix, 0)

plt.figure()
plt.imshow(filtered_rate.T, origin="upper",aspect="equal")
plt.colorbar().set_label("rate")
plt.title("filtered rate for noise probability >0.05")


print(f"number of pixels with noise probability >0.05: {np.
    ↪sum(filtered_noise_probability_matrix)}")
print(f"integral rate for noise probability >0.05: {np.sum(filtered_rate)}")
```

```
number of pixels with noise probability >0.05: 74717
integral rate for noise probability >0.05: 15.603322899109076
```


filtered rate for noise probability >0.05
0
50
100
150
200
250
300
350
400
0
100
200
300
400
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
rate

```
[ ]:
```