Lab Session #4

Computational Neurophysiology [E010620A]

Dept of Electronics and Informatics (VUB) and Dept of Information Technology (UGent)

Jorne Laton, Lloyd Plumart, Talis Vertriest, Jeroen Van Schependom, Sarah Verhulst

Student names and IDs: Robbe De Beck [01902805], Robbe De Muynck [01908861]
Academic Year: 2022-2023

Spiking Stochastics - Decoder and Encoder models

This exercise is adapted from the examples provided in the textbook "Case Studies in Neural Data Analysis", by Mark Kramer and Uri Eden (2016, MIT Press) and the 2020 eLife-LABs publication by Emily Schlafly, Anthea Cheung, Samantha Michalka, Paul Lipton, Caroline Moore-Kochlacs, Jason Bohland, Uri Eden, Mark Kramer. The outline of this exercise follows the theory presented in chapters 7 & 9 of "Neuronal Dynamics" by Gerstner, Kistler, Naud, Paninski (2014, Cambridge University Press). It also uses the principles outlined in Jackson and Carney, 2005 (publication provided with this exercise).

Aim

Here, we go deeper into understanding the stochastics behind neuronal action potential (spiking) generation. We will study the Poisson and Gaussian models and explore ways in which we can fit these point-process models to recorded datasets. In the second part of the exercise, we will consider encoder models that are based on leaky integrate-and-fire models with noisy inputs (8.1 in Gerstner et al.) and apply these models to test a specific research hypothesis related to the spontanous firing rate observed in cat auditory-nerve fibers.

Part 1 (Q1-Q4)

Data: Spontaneous spiking activity from a retinal neuron in culture, exposed to low-light and high-light environments. We continue with the dataset from the previous notebook.

Goal: Build simple models of interspike interval distributions as a function of the ambient light level.

Tools: Interspike interval (ISI) histograms, firing rate, maximum likelihood estimation, Kolmogorov-Smirnov plots, Poisson and Gaussian distributions.

Part 2 (Q5-Q7)

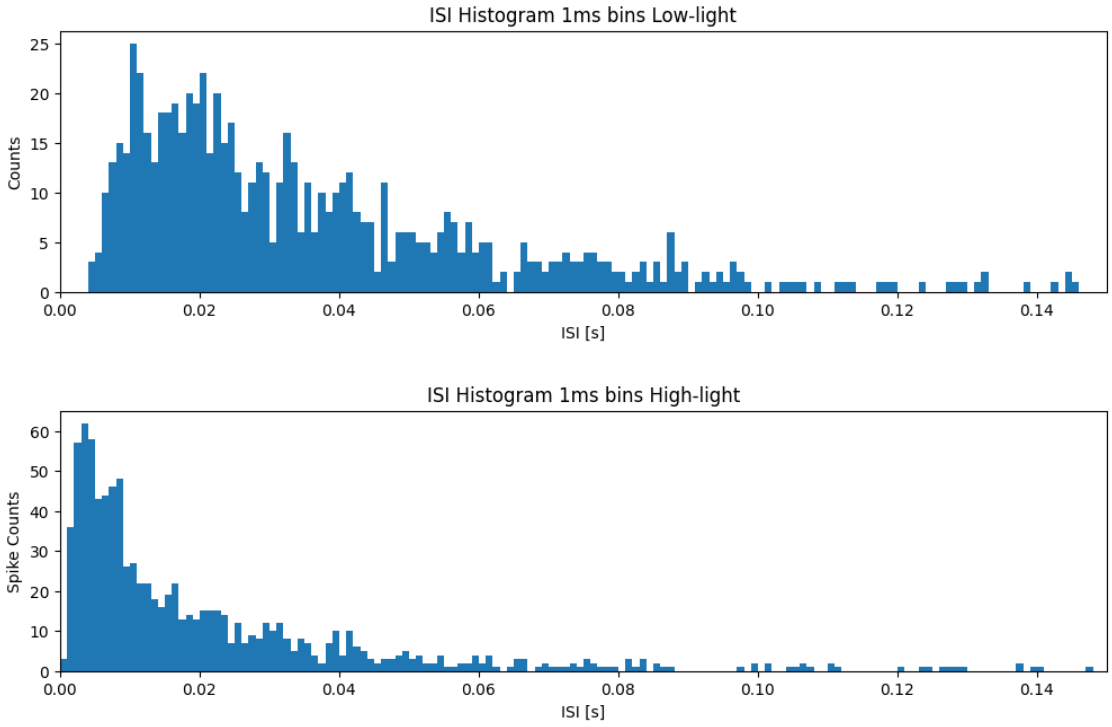
Data: Auditory-nerve spike patterns recorded in the absence of external stimulation from two different fiber types; a low-spontaneous rate fiber, and a high-spontaneous rate fiber.

Goal: Build decoder models with the same stochastics as observed experimentally.

Tools: Encoder models, ISI histograms, autocorrelation functions, integrate-and-fire models.

```
#Run this cell to make sure to have the preambles/data loaded
from pylab import *
import scipy.io as sio
import numpy as np
import random
import warnings
from ffGn_module import ffGn, inhomPP
%matplotlib inline
rcParams['figure.figsize']=(12,3)  # Change the default figure size

warnings.simplefilter(action='ignore', category=FutureWarning)

data = sio.loadmat('matfiles/08_spikes-1.mat')  # Load the spike train
data
print(data.keys())

spikes_low = data['SpikesLow'][0]
spikes_high = data['SpikesHigh'][0]

ISI_low = np.diff(spikes_low)
ISI_high = np.diff(spikes_high)

bin_edges = arange(0, 0.501, 0.001)  # Define the bins for the
histogram

plt.hist(ISI_low, bin_edges)        # Plot the histogram of the low-
light ISI data
xlim([0, 0.15])            # ... focus on ISIs from 0 to 150 ms
ylabel('Counts')
xlabel('ISI [s]')# ... label the y-axis
title('ISI Histogram 1ms bins Low-light')            # ... give the
plot a title
plt.show()

plt.hist(ISI_high, bin_edges)       # Plot the histogram of the high-
light ISI data
xlim([0, 0.15])            # ... focus on ISIs from 0 to 150 ms
xlabel('ISI [s]')              # ... label the x-axis
ylabel('Spike Counts')               # ... and the y-axis
title('ISI Histogram 1ms bins High-light')           # ... give the
plot a title
plt.show()
```

```
dict_keys(['__header__', '__version__', '__globals__', 'SpikesLow',
'SpikesHigh'])
```



PART 1

Q1: Statistical Models

We now consider another powerful technique to understand spiking data, by constructing a ***statistical model*** of the data. This model will capture important features of the data but does not consist of explicit biophysical components, such as (for example) in the Hodgkin-Huxley equations. Statistical models are thus ***phenomenological*** in nature and not biophysically realistic.

To construct a statistical model, we typically assume that the ISIs are indendent samples from an unknown distribution. We then consider a class of statistical distributions that might fit the original data and identify which of these distributions (and its parameters) that provide the best fit to the observed data.

The most basic model we could fit is the Poisson distribution (assumption: the number of spikes in any bin is independent of all previous and future spikes) with an unknown but constant firing rate λ (ν in the lecture slides). The probability P of observing k spikes in any time bin is given by the Poisson distribution,

$$P(k)=\frac{\lambda^k e^{-\lambda}}{k!},$$

where $k!$ is the factorial of k. Under this model, the distribution for the number of spikes in a bin is Poisson. An important question for our data-set is what the distribution of the waiting times between spikes (i.e., ISIs) looks like? It can be shown mathematically that for any Poisson process with a constant firing rate, the ISIs have an exponential distribution [Kass, Eden & Brown, 2014]. Mathematically, the probability density function for any ISI taking on a value x is

$$f(x) = \lambda \exp(-\lambda x),$$

where λ is the rate parameter for the Poisson process.

Our goal is to find a good value of λ so that our statistical model

$$f(x) = \lambda \exp(-\lambda x),$$

matches the observed ISI distributions.

Alert 2! The histogram ISI values are in the scale of frequencies, not probabilities, and moreover also binned,thus not taking into account its density. This means that we have to scale the histogram data twice. Once from frequencies to probability and once more to scale it from binned values to preveal the density values.

- Q1: First make a histogram of the actual data, bin the ISI's in 1ms bins. Then scale your data as described in alert 2 to be able to fit a probability density function. (Control that the total area of your histogram approaches the value of 1). Plot the scaled data together with the exponential probability density function, where you guess a value of λ . Now try different values of λ and guess which λ provides the best visual match. Do these steps for both the low and high light conditions.

- Fill in answer here

Q2: Likelihood functions

The process of guessing values of λ and comparing the model to the empirical ISI distribution can be automated by determining the ***likelihood*** function of the joint probability distribution of the data. The goal is to find the value of λ that maximizes the likelihood of the data given the statistical model; this value of λ will be the best fit of the model to the data.

To implement this procedure, first consider the probability density of observing a sequence of ISIs, $x_1, x_2, \ldots, x_n$. If we assume that the ISIs are independent, then the probability density is:

$$f(x_1, x_2, \ldots, x_n) \quad ¿ \quad f(x_1) f(x_2) \ldots f(x_n) \quad ¿ \quad \lambda^n \exp\left(-\lambda \sum_{i=1}^{n} x_i\right). ¿$$

We call this expression the joint probability distribution of the observed data. In the first equality, we separate the joint probability distribution $f(x_1, x_2, \ldots, x_n)$ into a product of probability distributions of each event (i.e., $f(x_1)$, the probability of the first ISI equaling x_1;

multiplied by $f(x_2)$, the probability of the second ISI equaling x_2; multiplied by $f(x_3)$, the probability of the third ISI equaling x_3; and so on). This partitioning of the joint probability is valid here because we assume the ISIs are independent. In the second equality, we replace each probability distribution with the exponential distribution we expect for the ISIs of a Poisson process. In the last equality, we rewrite the expression as a single exponential. Notice that this last expression is a function of the unknown rate parameter, λ.

When considered as a function of the unknown parameters, the joint distribution of the data is also called the *likelihood*. In this case, we write

$$L(\lambda) = \lambda^n e^{-\lambda(x_1+x_2+\ldots+x_n)},$$

to indicate that the likelihood L is a function of λ.

- Q2.1: Plot the likelihood function $L(\lambda)$ for each light condition and consider a range of λ values. Evaluate the maxima carefully, does the result make sense when considering the equation? If the result doesn't make sense at first, do not panic. It is a feature, not a bug.
- Q2.2: Plot the log of the likelihood instead, and identify which $L(\lambda)$s are a good fit to the low and high-light conditions. The λ, where the log likelihood is maximized, is called the **maximum likelihood estimate** of λ: $\hat{\lambda}$. The log-likelihood function is:

$$l(\lambda) = n\ln(\lambda) - \lambda(x_1+x_2+\ldots+x_n),$$

- Q2.3: Is the difference in the Poisson rate parameter between the low-and high-light conditions statistically significant? To address this last question, we can use a bootstrap analysis. Combine all the ISIs from both conditions into one pool, sample many (1000) new datasets with values randomly selected from that pool. Plot the distribution of the differences accross the 1000 samples as a histogram, together with the actual difference in both original rate parameters. Compare the results, what is your conclusion?
- Fill in answer here

Q3: Goodness of Fit

We assumed that the spikes were generated according to a poisson process, but is the exponential pdf that we created a good fit for the ISI?

- Q3.1: To answer this, first compare the model fits and the scaled data visually. Therefore plot again the exponential pdf (with the optimal λ) together with the scaled histogram values. Compare the expected proportion of ISIs for a Poisson process to the ISI histograms we actually observe in each condition. What is your conclusion?

To go beyond visual inspection and quantify the goodness of fit, we compare the cumulative distributions computed from the scaled data and the model. The ***cumulative distribution function (CDF)***, $F(x)$, is the probability that a random variable will take on a

value less than or equal to x. For the exponential ISI model with rate parameter λ, the model CDF is

$$F_{mod}(x) \quad ¿ \quad Pr(\text{ISI} \le x) \quad 1 - e^{-\lambda x} . ¿$$

We compare this to the empirical CDF of the data, $F_{emp}(x)$, which is defined as the proportion of observations less than or equal to x. In other words, this is the cumulative sum of all previous areas of the histogram, not just cumulative sum of the scaled probability density values. Scale your empirical data correctly, the last value should approach 1.

- Q3.2: Compute and plot the CDF function together with the empirical values. Evaluate how well the model fits the empirical data. Do this for both conditions and compare the CDF models according to its light condition.

- Q3.3: Another common way to visualize the difference between the model and empirical distributions is a ***Kolmogorov-Smirnov(KS)*** plot. This is a plot of the empirical CDF against the model CDF directly. Since the KS plot compares CDFs, both the x-axis and y-axis range from 0 to 1. A perfect fit between the model and empirical CDFs would look like a straight, 45-degree line between the points (0,0) and (1,1). Any deviation from this line represents deviation between the observed and model distributions. One nice result for comparing CDFs is that with enough data, the maximum difference between the model and empirical CDFs has a known asymptotic distribution, which can be used to put confidence bounds about the KS plot [Kass, Eden & Brown, 2014]. For 95% confidence bounds, a well-fit model should stay within $\pm 1.36/\sqrt{N}$ of the 45-degree line, where N is the number of ISIs observed. Let's place these confidence bounds on the KS plot and determine whether your model fits the data well.

- Fill in answer here

Q4: More advanced, inverse Gaussian probability models.

So far, we investigated how well one class of models, the exponential distribution function, fits the observed ISI distributions. However, this type of statistical model is not always sufficient to mimic the observed data. There are many other choices for statistical models; and here we'll try one other class of models: the inverse Gaussian probability model. This model has previously been used successfully to describe ISI structures [Iyengar & Liao, 1997]) and the mathematical expression for the inverse Gaussian probability density is,

$$f(x) = \sqrt{\frac{\lambda}{2\pi x^3}} \exp\left(\frac{-\lambda(x-\mu)^2}{2x\mu^2}\right).$$

The inverse Gaussian distribution has two parameters that determine its shape: μ, which determines the mean of the distribution; and λ, which is called the shape parameter. At x = 0, the inverse Gaussian has a probability density equal to zero, which suggests it could capture some of the refractoriness seen in the data.

If we again assume that the ISIs are independent of each other, then the likelihood of observing the sequence of ISIs, $x_1, x_2, \ldots, x_n$, is the product of the probabilities of each ISI,

$$L(\mu, \lambda) = f(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{N} \sqrt{\frac{\lambda}{2\pi x_i^3}} \exp\left(\frac{-\lambda(x_i - \mu)^2}{2 x_i \mu^2}\right)$$

The log likelihood is then

$$\log(L(\mu, \lambda)) = \frac{N}{2} \log\frac{\lambda}{2\pi} - \frac{3}{2}\sum_{i=1}^{N} \log x_i - \sum_{i=1}^{N} \frac{\lambda(x_i - \mu)^2}{2 x_i \mu^2}$$

Since this distribution has two parameters, the maximum likelihood solution for this model is the pair of parameter estimates $\hat{\mu}$, $\hat{\lambda}$ that maximizes the likelihood of the data. We can solve for the maximum likelihood estimate analytically by taking the derivative with respect to both parameters, setting these equal to zero, and solving the resulting set of equations. In this case, the maximum likelihood estimators are

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

and

$$\hat{\lambda} = \left(\frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{x_i} - \frac{1}{\acute{\mu}}\right)\right)^{-1}$$

- Q4.1 : Use this expression to fit an inverse Gaussian model to the scaled data in each condition. Again, make sure you scale the empirical data correctly to the model and plot them together.
- Q4.2: Consider the goodness-of-fit of the model in the two conditions. Does the inverse Gaussian model provide a good fit to the ISIs, and motivate why (or why not) this model is better fit? (calculate the CDF of both the inverse gaussian and the empirical data. Again, use the Ks plot to evaluate.)
- Q4.3: Compare the μ and λ estimates between the two conditions. What can you conclude about the differences between the low- and high-light conditions?
- Fill in answer here

PART 2

Q5: From Decoding to Encoding models

Identifying the best fitting model and its parameters helps us to understand and describe spiking statistics of a variety of neurons, and this process is called ***Decoding***. However, when developing neuronal models or mimicking physiological processes in machines (e.g. robots) one would like to follow an ***Encoding*** approach in which we provide a stimulus to a

model of the neuron and predict a spiking pattern that mimics that of the biophysical system. Well-designed encoding models can generate spiking patterns to any stimulus (e.g. Hodgkin-Huxkley type of neuronal models with escape noise), but for simplicity, we will focuss on simulating the spiking statistics of auditory-nerve models when spiking at their spontaneous rate (i.e. without external simululation applied). Indeed, spiking can occur in the absence of stimuluation because there is always some sort of neuronal/background noise present in biological systems that can drive the neuron temporarily above its firing threshold and generate sporadic spikes. Neurons or nerve fibers are therefore often characterised by their ***Spontaneous Rate (SR)*** i.e. the number of spikes/s (firing rate) generated in the absence of external stimulation.

Much of what we know about the auditory-nerve fibers (i.e. connection between the sensory cochlear inner-hair-cells and the spiral ganglion cells and auditory nerve bundle) stems from the characterisation of single-unit neuronal recordings in cats made by [Nelson Kiang, 1965] and [Charles M Liberman, 1978]. The below picture provides a good overview of the distribution of different spontaneous rates that were found from a large number fo single-unit of recordings (in the absence of external stimulation). Taken from [Jackson and Carney, 2005]

The authors of the last paper pose that the distribution of observed SRs fibers can be generated using ***Poisson-equivalent integrate-and-fire models*** for two or three fixed SR values. In other words, the stochastics of the underlying spike generation process is such that the observed SR distribution can be observed even when only two/three types of SR-AN fibers exist. Here, you will implement the proposed encoder model and compare it to a standard poisson-based encoder model.

- Q5.1 : First, determine (decode) the model parameters given recordings of two different AN fibers. Load in the data, and determine the $\hat{\lambda}$ and $\hat{\mu}$ parameters for both low and high-spontaneous rate AN fibers using an inverse Gaussian model. Plot again as done in Q4
- Fill in answer here

Q6: Inhomogeneous integrate-and-fire encoder model

Next, you can implement the encoding principle as outlined in [Jackson and Carney, 2005].

The authors suggest that a poisson process (i.e., each spike is generated independently from the previous spike times) cannot account for the observed behavior, instead the encoder model should include a ***long-range temporal dependence (LRD)*** which is characterised by the ***Hurst exponent (H)***. The hurst index is a measure of the long-term memory of a time series and relates to the rate at which the autocorrelation of a time series decreases as the lag between pairs of spike times increases. It quantifies the relative tendency of a time series to either regress strongly to the mean or to cluster in a specific direction.

- A value of H=0.5 reflects an exponential decay and describes a completely uncorrelated series (i.e. Poisson process). For this series, the absolute values of the autocorrelation function decays exponentially quickly to zero.
- Differently, for values of H: 0.5 - 1, there is LRD and the time serie has a long-term positive autocorrelation, i.e. the autocorrelations decay so slowly that their sum diverges to infinity. This means that a late spike in the series will probably be followed by a spike after a long ISI, and that the late spikes will all be characterised by long ISIs.

The authors suggest that the spontaneous rate histogram can be created by a ***fractal-Gaussian-noise-driven inhomogeneous poisson process (fGnDP)***, a type of escape noise model (chapter 9.4.1 in Gerstner) that consists of a doubly stochastic Poisson process, wherein the stochastic process that serves as an input to a poisson-equivalent integrate-fire model is a fractal-Gaussian noise (fGn) that can be made to contain LRD by modifying the Hurst index.

- Q6.1: Before we generate the spikes, let's first generate the escape noise (fGn). Generate a noisy output signal of 30-s length (FS = 1000 Hz) that has a mean rate equal to the SR of the fiber and that either follows a poisson process (H=0.5) or has LRD (H=0.95). Consider the SR 80 spikes/s. Note that even if H=0.5, the underlying SR distribution follows an inhomogeneous poisson process, where the rate varies due to pure white noise which is gaussian distributed around the mean SR. In the case of LRD (H=0.95), the distribution of the SR still follows an inhomogeneous poisson process, but the rate now varies due to white noise that also contains the effect of LRD. With the `ffGn(N, dt, H, SR)` function you can create both SR signals over time, one time for H=0.5 and one time for H=0.95. Note that in both cases (H=0.5 and H=0.95) the SR signal returned, has a mean of 0, to which you need to add the SR. Plot the time series of both noise signals, do you notice differences? This signal is equivalent to the $\Lambda(t)$ signal in the paper. Next, plot the autocorrelation function of the two noises (ACF, see previous exercise) and describe + interpret the differences between the two curves.
- Q6.2: In the second phase, feed the noise signal returned by the ffGn function as `in` to an inhomogenous integrate-and-fire neuron, which will output a list of spike times, given the noisy SR time-domain signal. The function works as follows: `spiketimes, PSref = inhomPP(in,dt)`. `PSref` is a control value that gives the poisson spike times that were on the basis of the inhomogeneous integrate-and-fire calculation. Compare the characteristics (ISI's) of the following models: a) inhomPP with H=0.5 input, b) inhomPP with H=0.95 input and c) the PSref basis spike times (those take a rate of 1 spike/s). Plot the three model's ISI's through a histogram with 1ms bins. Do your results align with expectations, why/why not?
- Fill in answer here

Q7: Investigate the research hypothesis

In the last step of this exercise, you should repeat the numerical experiment that Jackson and Carney performed to conclude that an inhomogeneous poisson process with 2/3 SR and includes LRD, is able to replicate the histogram characteristics (ISI's) of the reference AN SR distribution. You should also compare your outcomes with the figure when using Poisson-based (H=0.5) fGn as input to the inhomogeneous integrate-and-fire model.

- Q7.1: To obtain the histogram, you should compute the mean SR (spikes/s) in a repeated experiment, where you simulate the spike times in 30-s windows (using your encoder of choise) and repeat this experiment 150 times. You will need to use a for-loop for this. Consider the SRs of -20, 10 and 80 spikes/s and use FS = 1000 Hz. Compare your histograms with the reference figure and between Poisson, LRD models. Was their hypothesis sensible?

- Fill in answer here

Answers

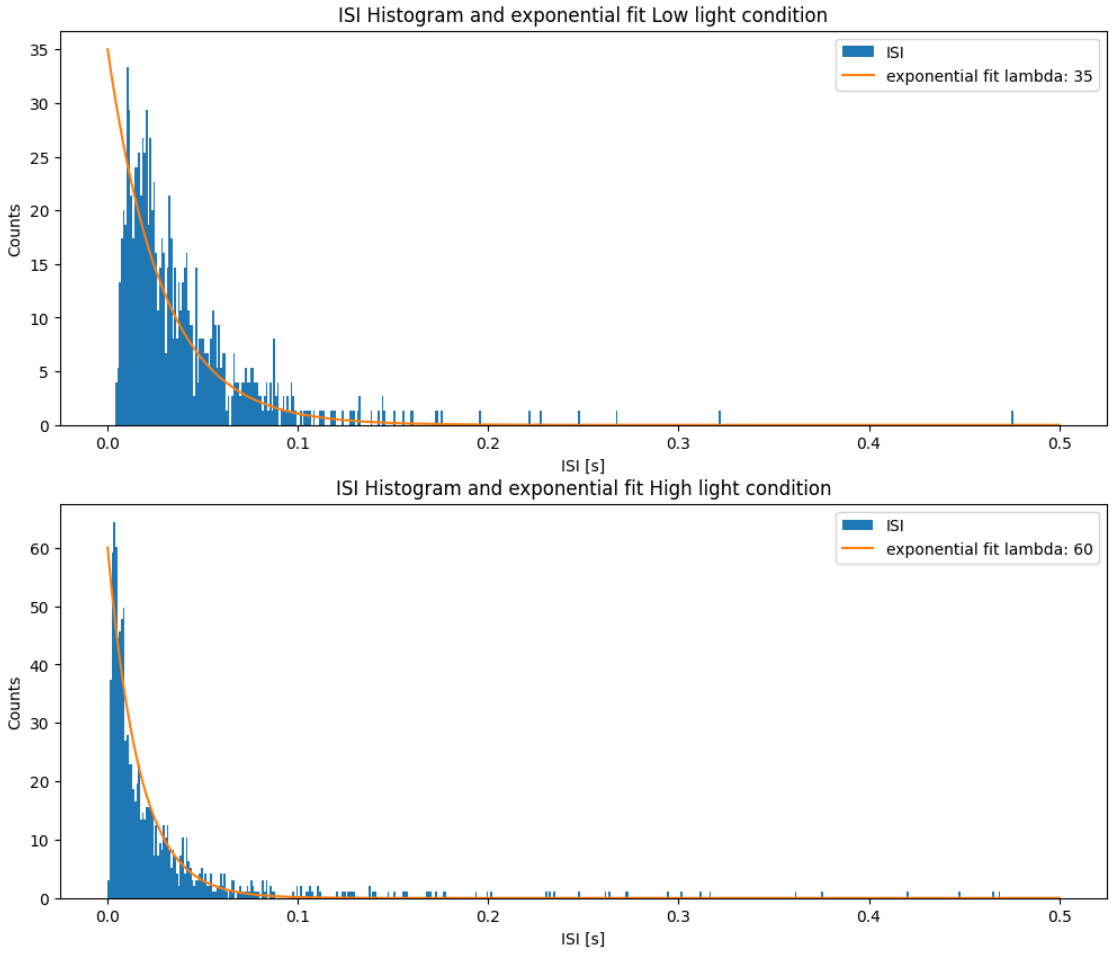
A1: Statistical Models

- Go back to Q1

```
# use the following variables:
# ISI_low (ISI's for low light)
# ISI_high (ISI's for high light)
# prob_density_low (y axis values of the scaled histogram of ISI in
1ms bins, low light condition)
# prob_density_high (y axis values of the scaled histogram of ISI in
1ms bins, high light condition)
# bin_edges
# bin_mids
# N_low (total number of ISI's, low light)
# N_high (total number of ISI's, high light)
# exp_pdf_low (exponential pdf, low light)
# exp_pdf_high (exponential pdf, high light)

######################
##   Q.1 solution   ##
######################

ISI_low = np.diff(spikes_low)
ISI_high = np.diff(spikes_high)

bin_edges = arange(0, 0.501, 0.001)  # Define the bins for the
histogram

L = 35
fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,10))
prob_density_low, bins_low, _ = ax1.hist(ISI_low, bin_edges,
density=True, label='ISI')
```

```
print('total area of historgram low light condition: {}
ms'.format(np.sum(prob_density_low)))
ax1.plot(bin_edges, L*np.exp(-L*bin_edges), label='exponential fit
lambda: {}'.format(L))
ax1.set_xlabel('ISI [s]')
ax1.set_ylabel('Counts')
ax1.set_title('ISI Histogram and exponential fit Low light condition')
prob_density_high, bins_high, _ = ax2.hist(ISI_high, bin_edges,
density=True, label='ISI')
print('total area of historgram high light condition: {}
ms'.format(np.sum(prob_density_high)))
L = 60
ax2.plot(bin_edges, L*np.exp(-L*bin_edges), label='exponential fit
lambda: {}'.format(L))
ax2.set_xlabel('ISI [s]')
ax2.set_ylabel('Counts')
ax2.set_title('ISI Histogram and exponential fit High light
condition')
ax1.legend()
ax2.legend()
plt.show()
```

```
total area of historgram low light condition: 1000.0000000000003 ms
total area of historgram high light condition: 1000.0 ms
```



A1 conclusion

The probability density function has units: 1/s. The sum amounts to 1000 ms. This gives a total probability of 1 (after multiplying with the binsize).

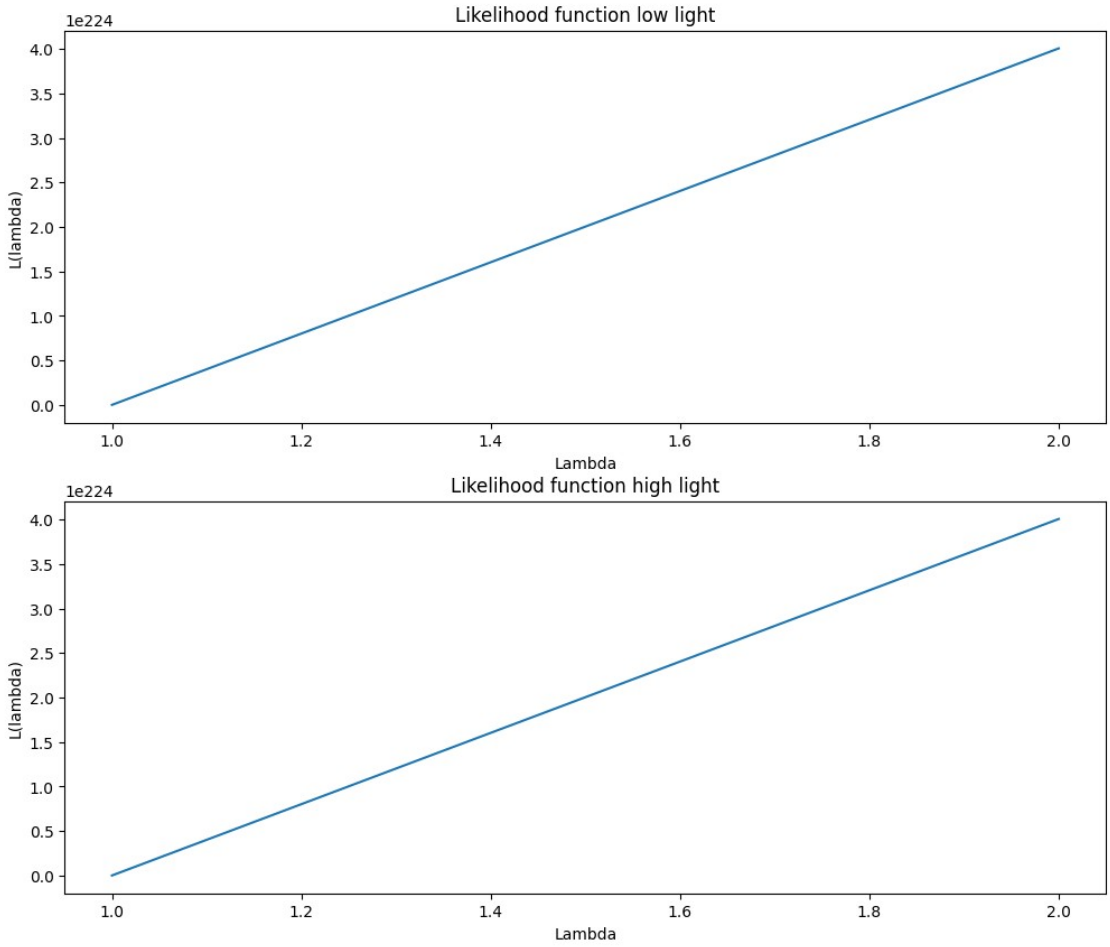
A2: Likelihood functions

- Go back to Q2

```
# hint, exclude lambda = 0 in your range of lambdas.

# Q2.1: use the following variables:
# likelihood_low      (for your likelihood model, low light)
# likelihood_high      (for your likelihood model, high light)

######################
##  Q2.1 solution   ##
######################
N_low = len(ISI_low)
Lambdas = np.linspace(1, 40, 40)
prob_low_sum = np.sum(prob_density_low)*0.001
```

```
likelihood_low = np.exp(-Lambdas*prob_low_sum) * Lambdas**N_low
# print(likelihood_low)

N_high = len(ISI_low)
prob_high_sum = np.sum(prob_density_low)*0.001
likelihood_high = np.exp(-Lambdas*prob_low_sum) * Lambdas**N_high
# print(likelihood_low)

fig, (ax1, ax2) = plt.subplots(2,1,figsize=(12,10))
ax1.plot(Lambdas, likelihood_low)
ax1.set_title('Likelihood function low light')
ax1.set_xlabel('Lambda')
ax1.set_ylabel('L(lambda)')

ax2.plot(Lambdas, likelihood_high)
ax2.set_title('Likelihood function high light')
ax2.set_xlabel('Lambda')
ax2.set_ylabel('L(lambda)')
plt.show()
```

```
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2591738367.py:13:
RuntimeWarning: overflow encountered in power
  likelihood_low = np.exp(-Lambdas*prob_low_sum) * Lambdas**N_low
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2591738367.py:18:
RuntimeWarning: overflow encountered in power
  likelihood_high = np.exp(-Lambdas*prob_low_sum) * Lambdas**N_high
```

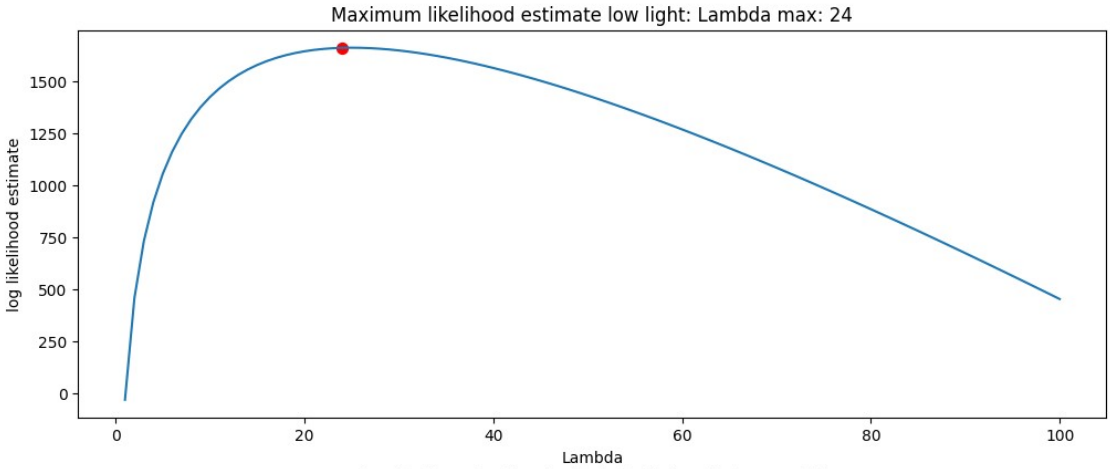


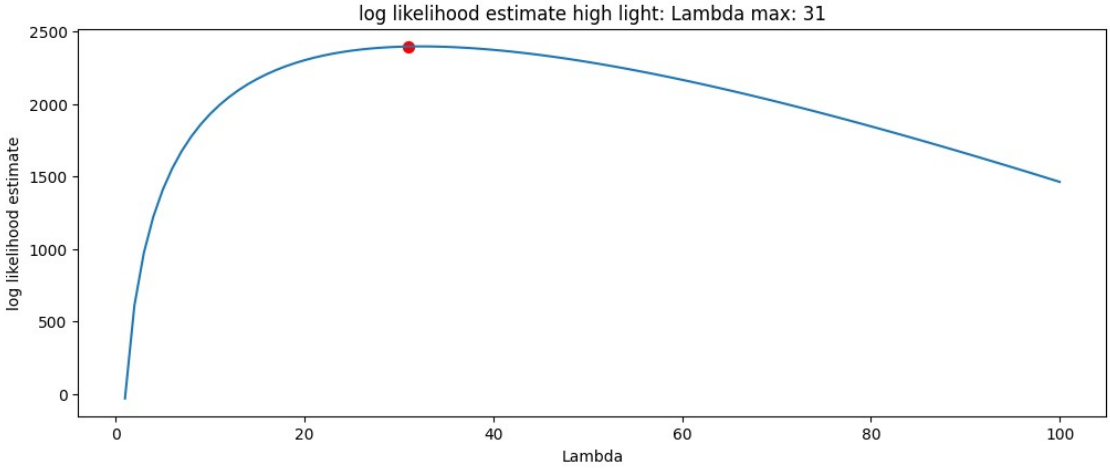
A2.1 conclusion

This result does not make sense. In both cases the likelihood tends to infinity. From the equation of the likelihood it is expected that there would be a maximum. This is not the case here. This is the result of an overflow error of Python. It is trying to calculate very large (lambda ** N) and very small (exp(-lambda*sum(x))) numbers.

```
# Q2.2: use the following variables:

# log_likelihood_low (for your log_likelihood model, low light)
# log_likelihood_high (for your log_likelihood model, high light)

# max_likelihood_lambda_low (for the optimal lambda value, low light)
# max_likelihood_lambda_high (for the optimal lambda value, high
light)

######################
##  Q2.2 solution   ##
```

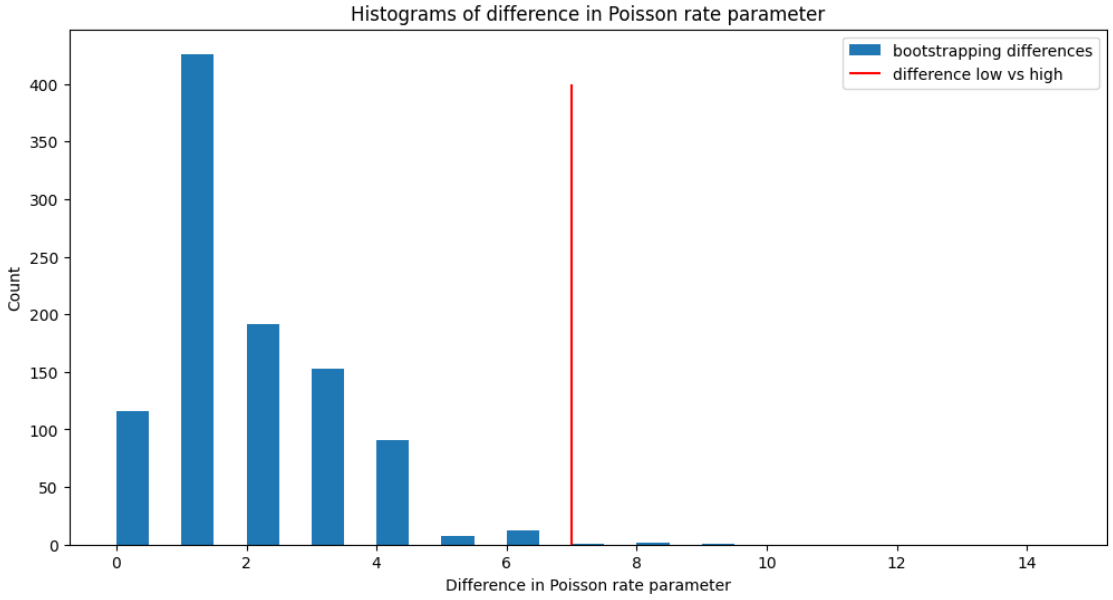
```
######################

def calc_log_likelihood(ISI):

    N = len(ISI)

    ISI_sum = np.sum(ISI)
    log_likelihood = [N*np.log(Lambda) - Lambda*ISI_sum for Lambda in
Lambdas]
    max_likelihood_lambda = np.argmax(log_likelihood)



    return log_likelihood, max_likelihood_lambda


Lambdas = np.linspace(1, 100, 100)

log_likelihood_low, max_likelihood_lambda_low =
calc_log_likelihood(ISI_low)
log_likelihood_high, max_likelihood_lambda_high =
calc_log_likelihood(ISI_high)
print('lambda_max = {}'.format(max_likelihood_lambda_low))

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,10))
ax1.plot(Lambdas, log_likelihood_low)
ax1.scatter(max_likelihood_lambda_low,
log_likelihood_low[max_likelihood_lambda_low],s=50,color='r')
ax1.set_xlabel('Lambda')
ax1.set_ylabel('log likelihood estimate')
ax1.set_title('Maximum likelihood estimate low light: Lambda max:
{}'.format(max_likelihood_lambda_low))
print('lambda_max = {}'.format(max_likelihood_lambda_high))
ax2.plot(Lambdas, log_likelihood_high)
ax2.scatter(max_likelihood_lambda_high,
log_likelihood_high[max_likelihood_lambda_high],s=50,color='r')
ax2.set_xlabel('Lambda')
ax2.set_ylabel('log likelihood estimate')
ax2.set_title('log likelihood estimate high light: Lambda max:
{}'.format(max_likelihood_lambda_high))

plt.show()

lambda_max = 24
lambda_max = 31
```






```
# use the following parameters:
# N_all (N_low + N_high)
# mean_lambda_diff (1 digit value)
# sample_diff (array with 1000 difference values)

######################
##  Q2.3 solution   ##
######################

ISIS = np.concatenate((ISI_high,ISI_low))
N_all = len(ISIS)

Lambda_diff = []

for d in range(1000):

    random.shuffle(ISIS)
    ds1 = ISIS[:N_all//2]
    ds2 = ISIS[N_all//2:]

    _, max_likelihood_lambda_1 = calc_log_likelihood(ds1)
```

```
    _, max_likelihood_lambda_2 = calc_log_likelihood(ds2)

    Lambda_diff.append(abs(max_likelihood_lambda_2-
max_likelihood_lambda_1))

bin_edges = arange(0, 15, 0.5)
fig, ax = plt.subplots(1,1,figsize=(12,6))
ax.hist(Lambda_diff, bin_edges, label='bootstrapping differences')
ax.vlines(abs(max_likelihood_lambda_high-max_likelihood_lambda_low),
0, 400, 'r', label='difference low vs high')
ax.set_title('Histograms of difference in Poisson rate parameter')
ax.set_xlabel('Difference in Poisson rate parameter')
ax.set_ylabel('Count')
plt.legend()
plt.show()
```



A2.3 conclusion

There is a significant difference between the lambda of the high light condition compared to the low light condition. This can be concluded from the plot above. The difference of the original low vs high intensisy (indicated by the red line) lies at the very end of the distribution of the boostrapped analysis. From this it can be concluded that the original difference is significantly different from the randomly sampled bootstrapped analysis.

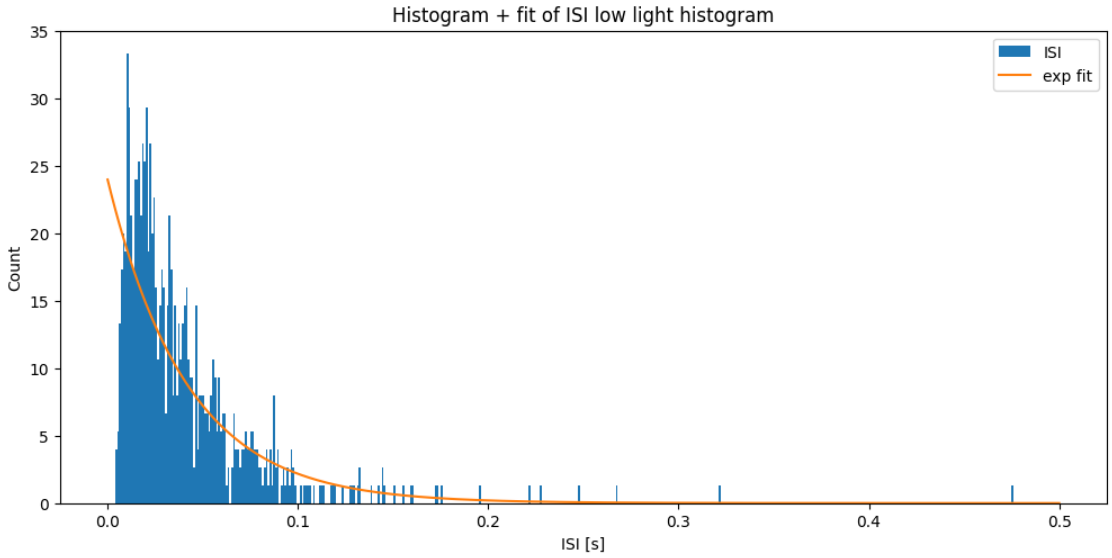
A3: Goodness of Fit

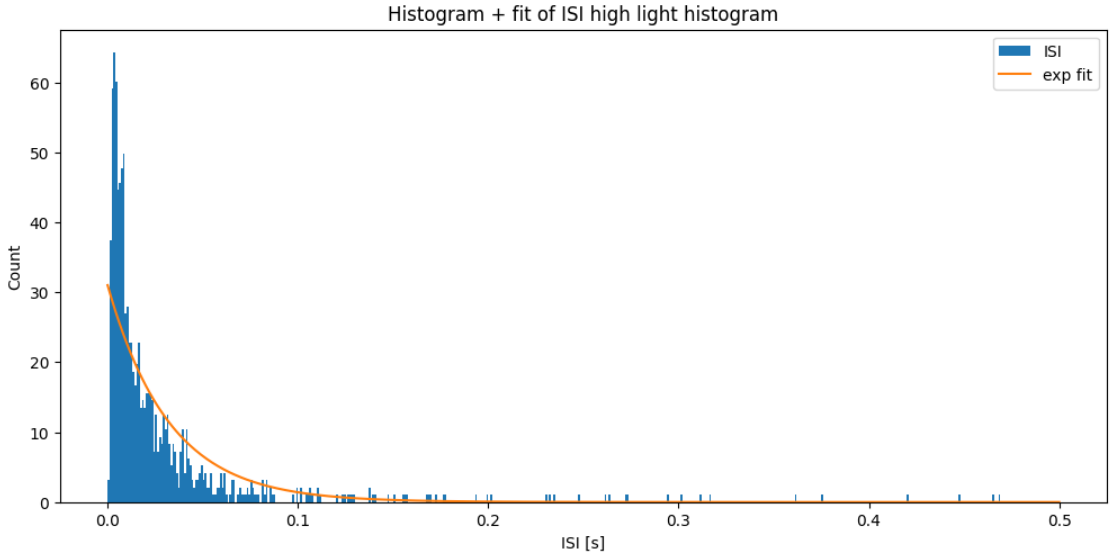
- Go back to Q3

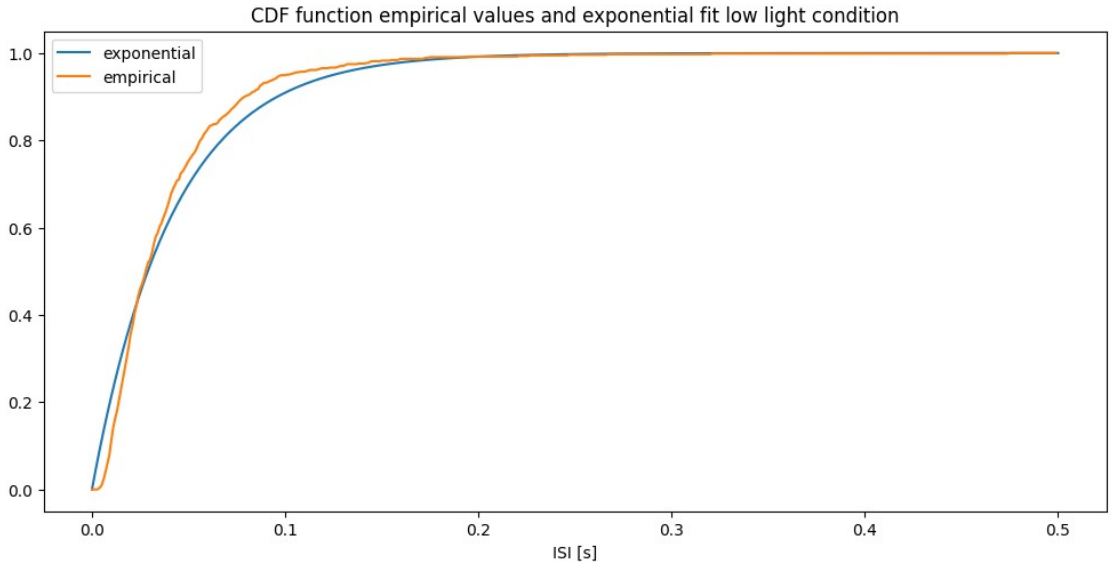
```
# use the following parameters
# ISI_low (ISI's for low light)
# ISI_high (ISI's for high light)
# prob_density_low (y axis values of the scaled histogram of ISI in
1ms bins, low light condition)
# prob_density_high (y axis values of the scaled histogram of ISI in
1ms bins, high light condition)
# bin_edges
# bin_mids
# N_low (total number of ISI's, low light)
# N_high (total number of ISI's, high light)
# exp_pdf_low (exponential pdf, low light)
# exp_pdf_high (exponential pdf, high light)
# CDF_exp_low (the CDF function of the exponential pdf, low light)
# CDF_exp_high (the CDF function of the exponential pdf, high light)
# CDF_emp_low (empirical values, low light)
# CDF_emp_high (empirical values, high light)

########################
##   Q3.1 solution   ##
########################

# Your Code Goes Here
# Check the integral (sum of all y-values *dt=1)

ISI_low = np.diff(spikes_low)
ISI_high = np.diff(spikes_high)

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
bin_edges = arange(0, 0.501, 0.001)  # Define the bins for the
histogram
prob_density_low, bins_low, _ = ax1.hist(ISI_low, bin_edges,
density=True, label='ISI')
Lambdas_low = 24
exp_pdf_low = Lambdas_low*np.exp(-Lambdas_low*bin_edges)
ax1.plot(bin_edges, exp_pdf_low, label='exp fit')
ax1.set_title('Histogram + fit of ISI low light histogram')
ax1.set_xlabel('ISI [s]')
ax1.set_ylabel('Count')
prob_density_high, bins_high, _ = ax2.hist(ISI_high, bin_edges,
density=True, label='ISI')
Lambda_high = 31
exp_pdf_high = Lambda_high*np.exp(-Lambda_high*bin_edges)
ax2.plot(bin_edges, exp_pdf_high, label='exp fit')
ax2.set_title('Histogram + fit of ISI high light histogram')
ax2.set_xlabel('ISI [s]')
ax2.set_ylabel('Count')
ax1.legend()
ax2.legend()
plt.show()
```

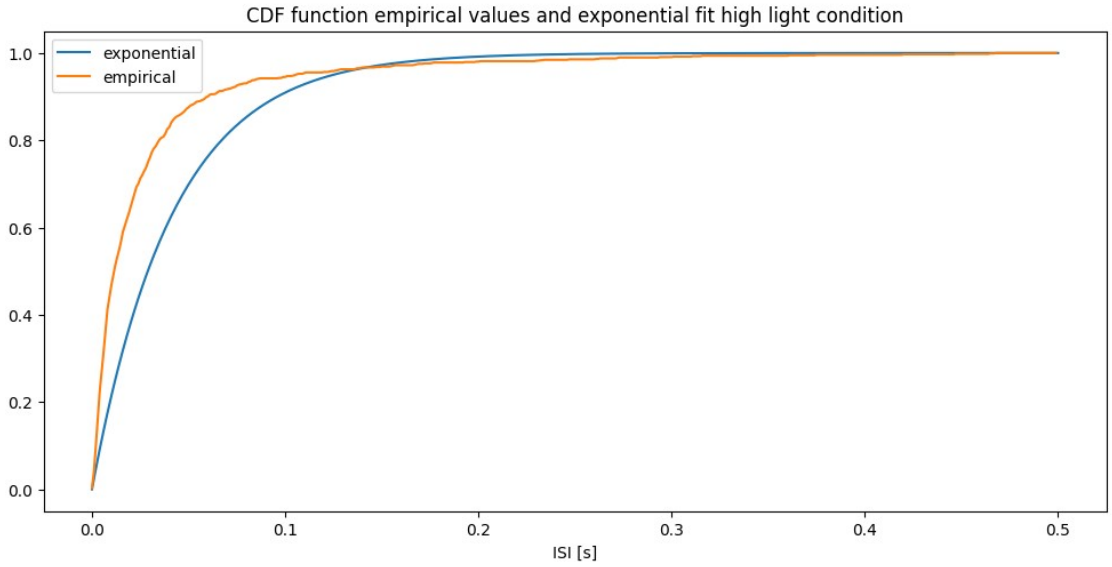
```
######################
##  Q3.2 solution   ##
######################

# for low light condition
Lambda_low = 24
CDF_exp_low = 1 - np.exp(-Lambda_low*bin_edges)

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
ax1.plot(bin_edges, CDF_exp_low, label='exponential')
ax1.set_title('CDF function empirical values and exponential fit low
light condition')
ax1.set_xlabel('ISI [s]')

CDF_emp_low = np.cumsum(prob_density_low)*0.001 # multiply with
binwidth, 1ms
ax1.plot(bin_edges[:-1], CDF_emp_low, label='empirical')
ax1.legend()


# for high light condition
Lambda_high = 24
CDF_exp_high = 1 - np.exp(-Lambda_high*bin_edges)
ax2.plot(bin_edges, CDF_exp_high, label='exponential')
CDF_emp_high = np.cumsum(prob_density_high)*0.001 # multiply with
binwidth, 1ms
ax2.plot(bin_edges[:-1], CDF_emp_high, label='empirical')
ax2.set_title('CDF function empirical values and exponential fit high
light condition')
ax2.set_xlabel('ISI [s]')
ax2.legend()
plt.show()


######################
##  Q3.3 solution   ##
######################
N_low = len(CDF_emp_low)
fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
ax1.plot(CDF_exp_low[:-1], CDF_emp_low)
ax1.plot(CDF_emp_low,CDF_emp_low, 'k')
ax1.fill_between(CDF_emp_low, CDF_emp_low+1.36/np.sqrt(N_low),
CDF_emp_low-1.36/np.sqrt(N_low), alpha=0.5, color='r')
ax1.set_title('KS plot low light condition')
ax1.set_ylabel('CDF empirical')
ax1.set_xlabel('CDF exponential')

N_high = len(CDF_emp_high)
```

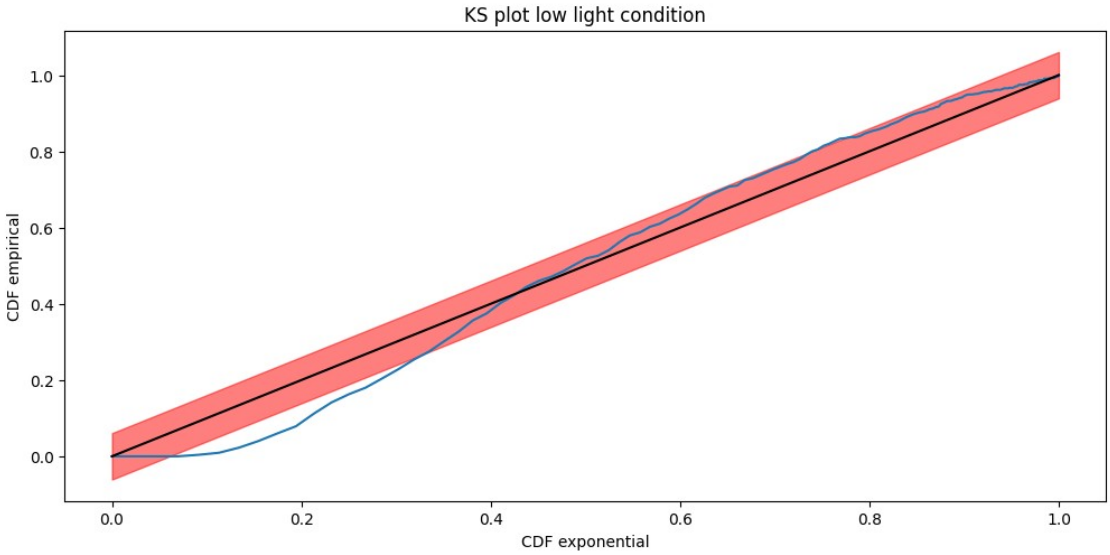
```
ax2.plot(CDF_exp_high[:-1], CDF_emp_high)
ax2.plot(CDF_emp_high,CDF_emp_high, 'k')
ax2.fill_between(CDF_emp_high, CDF_emp_high+1.36/np.sqrt(N_high),
CDF_emp_high-1.36/np.sqrt(N_high), alpha=0.5, color='r')
ax2.set_title('KS plot high light condition')
ax2.set_ylabel('CDF empirical')
ax2.set_xlabel('CDF exponential')

plt.show()
```

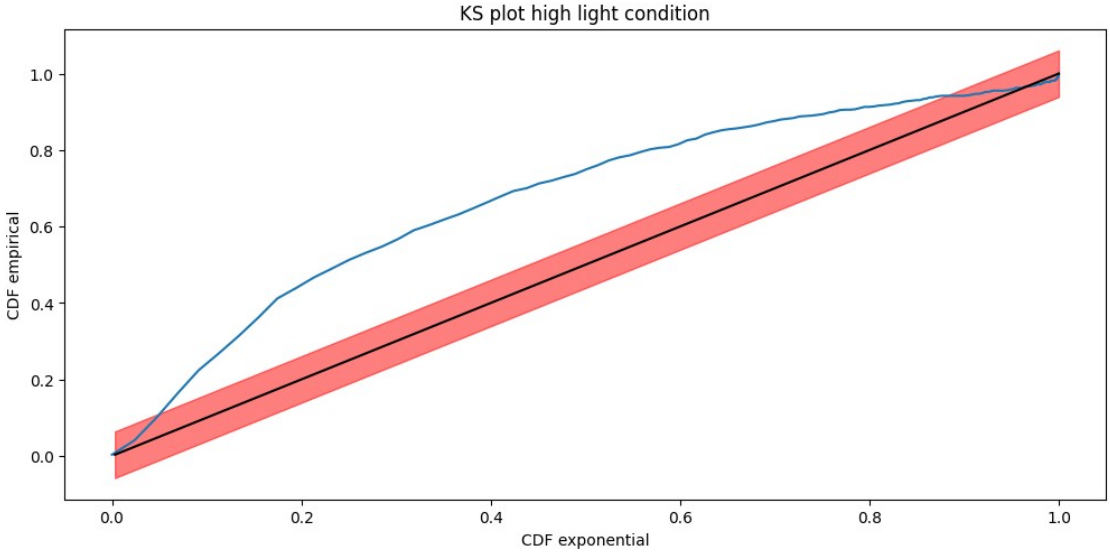





CDF function empirical values and exponential fit low light condition
exponential
empirical
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
ISI [s]


CDF function empirical values and exponential fit high light condition
exponential
empirical
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
ISI [s]





A3.1 conclusion

The fit is not perfect for the ISI histogram. The exponential pdf has a exponentially decreasing trend. This is not the case for the ISI histogram. The lowest ISI values are overestimated (the refractory period is not taken into account with the exponential fit). For higher values of ISI's the relationship holds up fairly well. For lower values of ISI's however the poisson process underestimates the amount of ISI's with a lower value (bursting is not taken into account). The refractory period and bursting are not taken into account because for a Poisson process values are assumed to be independant.

A3.2 conclusion

The conclusion from this plot confirms our visual inspection from above. There is a good fit for higher values but worse for lower values. However, the low light intensity has a visible better fit than the high light intensity which has a consistant higher CDF value.

A3.3 conclusion

From the plots above we can see that in both cases, low and high light intensity, the fit is not good. The KS line does not lie within the confidence bounds. However it is noteworthy that in the low light condion the error lies a lot closer to the confidence boundary.

A4: More advanced, inverse Gaussian probability models

- Go back to Q4

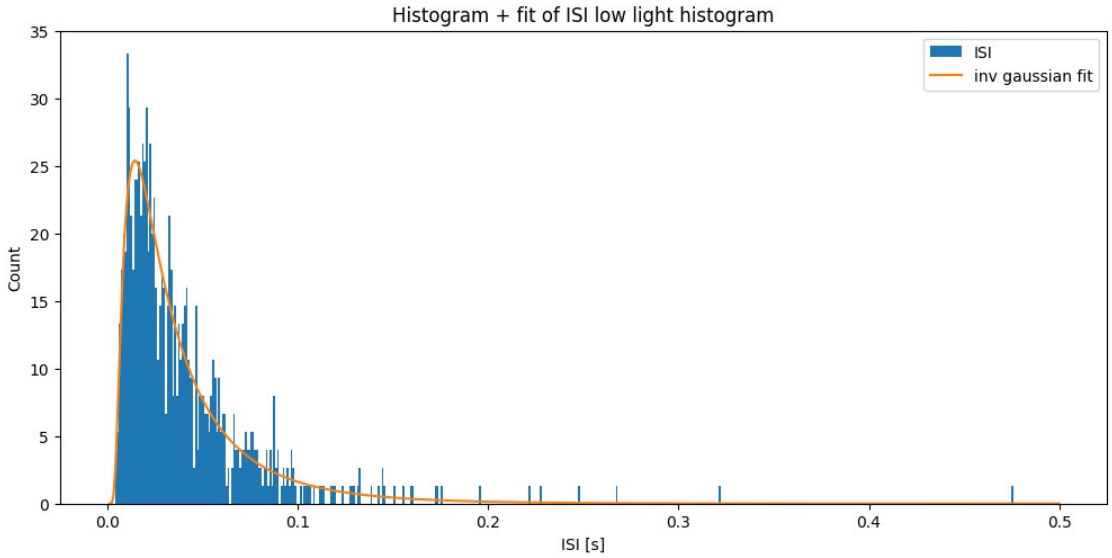
```
# use the following parameters:
# mu_low (mean of the ISI's... the mu of the inverse gaussian, low
light)
# mu_high (mean of the ISI's... the mu of the inverse gaussian, high
light)
# lambda_low (shape parameter of the inverse gaussion, low light)
# lambda_high (shape parameter of the inverse gaussion, low light)
# inv_gaussian_low (inverse gaussion model, low light)
# inv_gaussian_high (inverse gaussion model, high light)
# CDF_invgauss_low (CDF of inverse gaussian function, low light)
# CDF_invgauss_high (CDF of inverse gaussian function, high light)

########################
##   Q4.1 solution   ##
########################

# Your Code Goes Here
ISI_low = np.diff(spikes_low)
ISI_high = np.diff(spikes_high)

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
bin_edges = arange(0, 0.501, 0.001)  # Define the bins for the
histogram
prob_density_low, bins_low, _ = ax1.hist(ISI_low, bin_edges,
density=True, label='ISI')

mu_low = np.mean(ISI_low)
Lambda_low = (np.mean(1/ISI_low - 1/mu_low))**(-1)

inv_gaussian_low = np.sqrt(Lambda_low/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_low*(bin_edges-mu_low)**2/(2*bin_edges*mu_low**2))
```

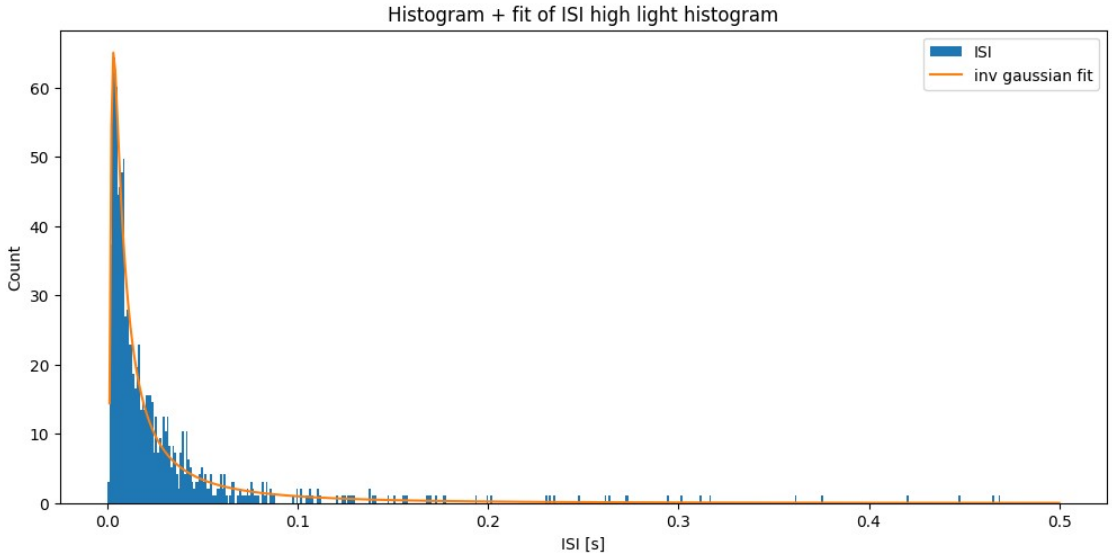
```
ax1.plot(bin_edges, inv_gaussian_low, label='inv gaussian fit')
ax1.set_title('Histogram + fit of ISI low light histogram')
ax1.set_xlabel('ISI [s]')
ax1.set_ylabel('Count')
ax1.legend()
prob_density_high, bins_high, _ = ax2.hist(ISI_high, bin_edges,
density=True, label='ISI')
mu_high = np.mean(ISI_high)
Lambda_high = (np.mean(1/ISI_high - 1/mu_high))**(-1)

inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_high*(bin_edges-mu_high)**2/(2*bin_edges*mu_high**2))
ax2.plot(bin_edges, inv_gaussian_high, label='inv gaussian fit')
ax2.set_title('Histogram + fit of ISI high light histogram')
ax2.set_xlabel('ISI [s]')
ax2.set_ylabel('Count')
ax2.legend()
plt.show()

########################
##   Q4.2 solution   ##
########################
fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
CDF_invgauss_low = np.cumsum(inv_gaussian_low[1:]) *0.001 # multiply
with binwidth, 1ms
ax1.plot(bin_edges[1:], CDF_invgauss_low, label='inv gaussian fit')
CDF_emp_low = np.cumsum(prob_density_low)*0.001 # multiply with
binwidth, 1ms
ax1.plot(bin_edges[:-1], CDF_emp_low, label='empirical')
ax1.set_title('CDF function empirical values and inv gaussian fit low
light condition')
ax1.set_xlabel('ISI [s]')
ax1.legend()
CDF_invgauss_high = np.cumsum(inv_gaussian_high[1:])*0.001 # multiply
with binwidth, 1ms
ax2.plot(bin_edges[1:], CDF_invgauss_high, label='inv gaussian fit')
CDF_emp_high = np.cumsum(prob_density_high)*0.001 # multiply with
binwidth, 1ms
ax2.plot(bin_edges[:-1], CDF_emp_high, label='empirical')
ax2.set_title('CDF function empirical values and inv gaussian fit fit
high light condition')
ax2.set_xlabel('ISI [s]')
ax2.legend()
plt.show()

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
N_low = len(CDF_emp_low)
ax1.plot(CDF_invgauss_low, CDF_emp_low)
ax1.plot(CDF_emp_low,CDF_emp_low, 'k')
```

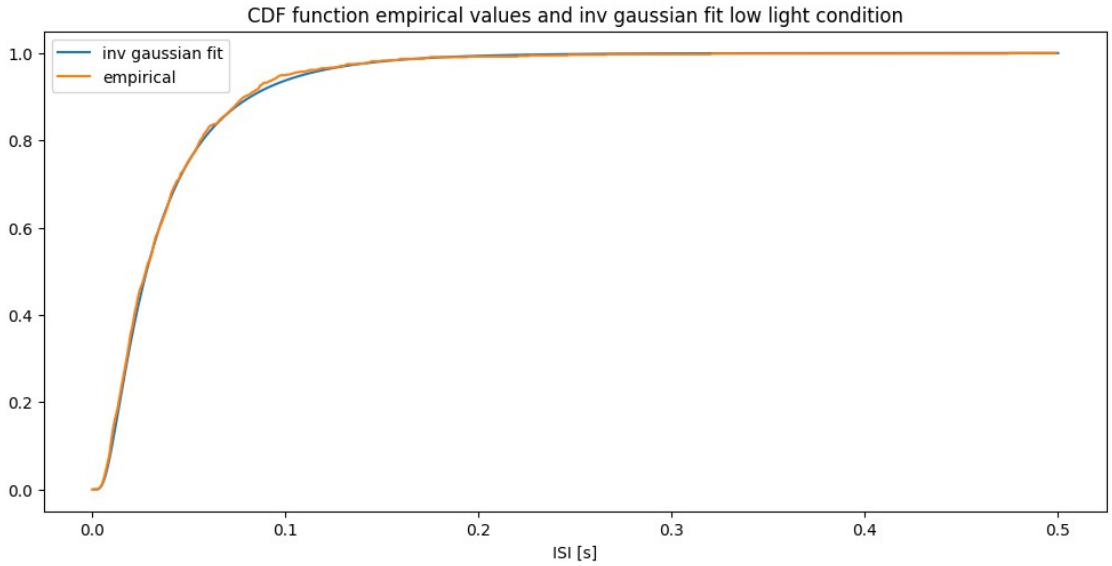
```
ax1.fill_between(CDF_emp_low, CDF_emp_low+1.36/np.sqrt(N_low),
CDF_emp_low-1.36/np.sqrt(N_low), alpha=0.5, color='r')
ax1.set_xlabel('inv gaussian')
ax1.set_ylabel('empirical')
ax1.set_title('KS plot low light condition')
N_high = len(CDF_emp_high)
ax2.plot(CDF_invgauss_high, CDF_emp_high)
ax2.plot(CDF_emp_high,CDF_emp_high, 'k')
ax2.fill_between(CDF_emp_high, CDF_emp_high+1.36/np.sqrt(N_high),
CDF_emp_high-1.36/np.sqrt(N_high), alpha=0.5, color='r')
ax2.set_xlabel('inv gaussian')
ax2.set_ylabel('empirical')
ax2.set_title('KS plot high light condition')
plt.show()
#######################
##   Q4.3 solution   ##
#######################
print('Low light condition: mu: {}, lambda: {}'.format(mu_low,
Lambda_low))
print('High light condition: mu: {}, lambda: {}'.format(mu_high,
Lambda_high))
print('Difference: mu: {}, lambda: {}'.format(abs(mu_high-
mu_low),abs(Lambda_high-Lambda_low)))

ISIS = np.concatenate((ISI_high,ISI_low))
N_all = len(ISIS)

Lambda_diff = []
mu_diff = []

for d in range(1000):

    random.shuffle(ISIS)
    ds1 = ISIS[:N_all//2]
    ds2 = ISIS[N_all//2:]

    mu_1 = np.mean(ds1)
    Lambda_1 = (np.mean(1/ds1 - 1/mu_1))**(-1)

    mu_2 = np.mean(ds2)
    Lambda_2 = (np.mean(1/ds2 - 1/mu_2))**(-1)

    Lambda_diff.append(abs(Lambda_2-Lambda_1))
    mu_diff.append(abs(mu_2-mu_1))

bin_edges = arange(0, 0.06, 0.0005)
fig, (ax1, ax2) = plt.subplots(2,1,figsize=(12,10))
ax1.hist(Lambda_diff, bin_edges, label='bootstrapping differences')
ax1.vlines(abs(Lambda_high-Lambda_low), 0, 400, 'r', label='difference
low vs high')
```

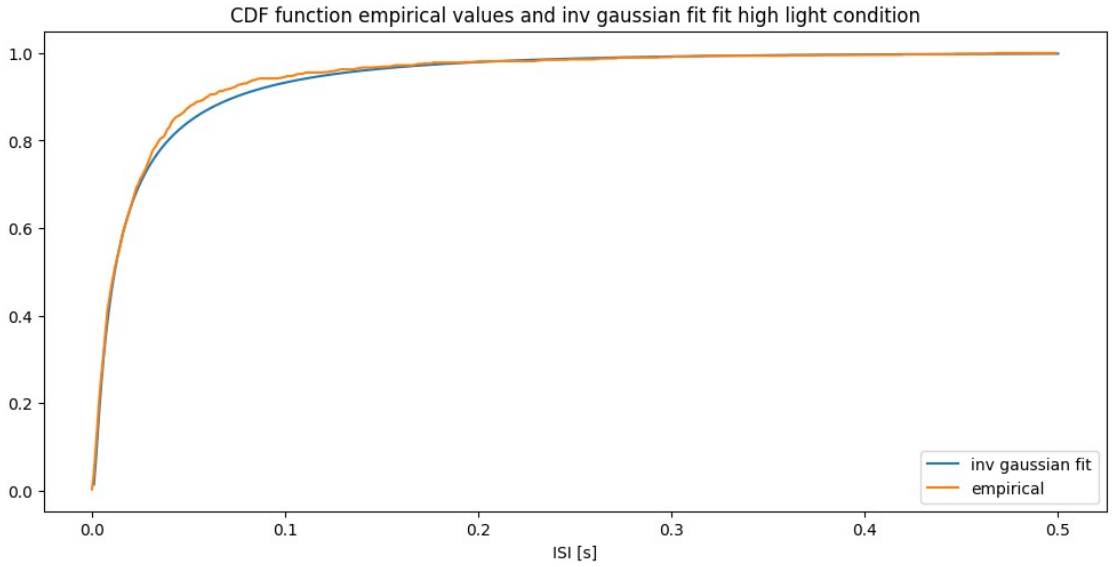
```
ax1.set_title('Histograms of difference in lambda parameter')
ax1.set_xlabel('Difference in lambda parameter')
ax1.set_ylabel('Count')
ax1.legend()

bin_edges = arange(0, 0.02, 0.0005)
ax2.hist(mu_diff, bin_edges, label='bootstrapping differences')
ax2.vlines(abs(mu_high-mu_low), 0, 400, 'r', label='difference low vs
high')
ax2.set_title('Histograms of difference in mu parameter')
ax2.set_xlabel('Difference in mu parameter')
ax2.set_ylabel('Count')
ax2.legend()
plt.show()
```

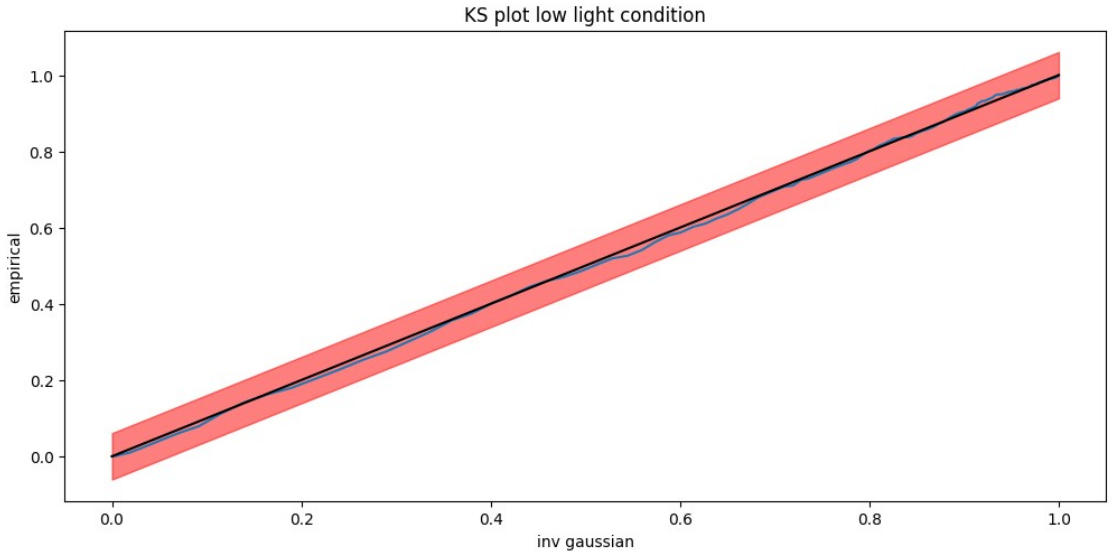
```
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2433680903.py:26:
RuntimeWarning: divide by zero encountered in divide
  inv_gaussian_low =
np.sqrt(Lambda_low/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_low*(bin_edges-mu_low)**2/(2*bin_edges*mu_low**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2433680903.py:26:
RuntimeWarning: invalid value encountered in multiply
  inv_gaussian_low =
np.sqrt(Lambda_low/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_low*(bin_edges-mu_low)**2/(2*bin_edges*mu_low**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2433680903.py:37:
RuntimeWarning: divide by zero encountered in divide
  inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_high*(bin_edges-mu_high)**2/(2*bin_edges*mu_high**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2433680903.py:37:
RuntimeWarning: invalid value encountered in multiply
  inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges**3))*np.exp(-
Lambda_high*(bin_edges-mu_high)**2/(2*bin_edges*mu_high**2))
```

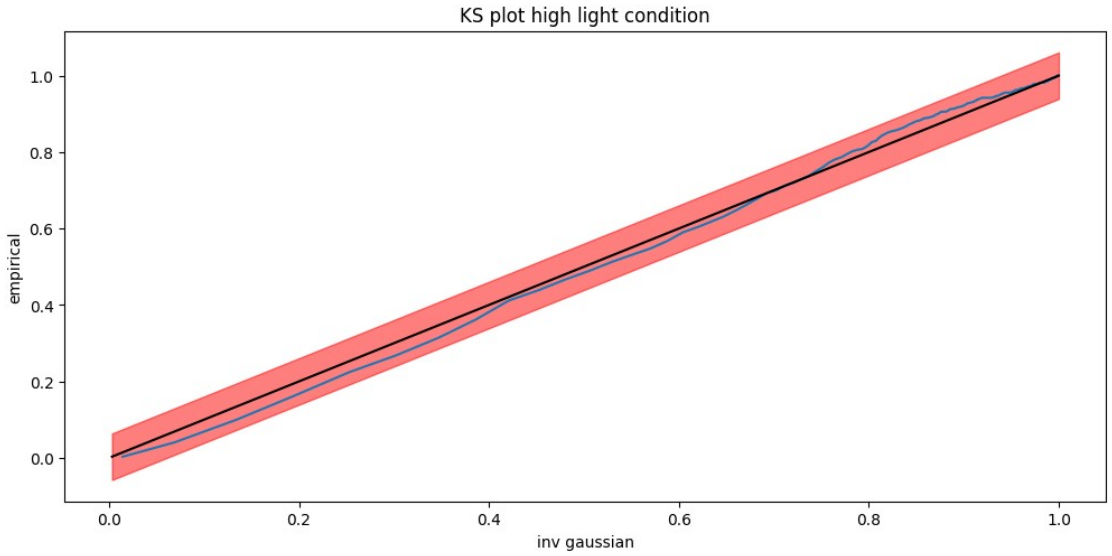

Histogram + fit of ISI low light histogram
ISI
inv gaussian fit
Count
ISI [s]
0
5
10
15
20
25
30
35
0.0
0.1
0.2
0.3
0.4
0.5


Histogram + fit of ISI high light histogram
ISI
inv gaussian fit
Count
ISI [s]
0
10
20
30
40
50
60
0.0
0.1
0.2
0.3
0.4
0.5

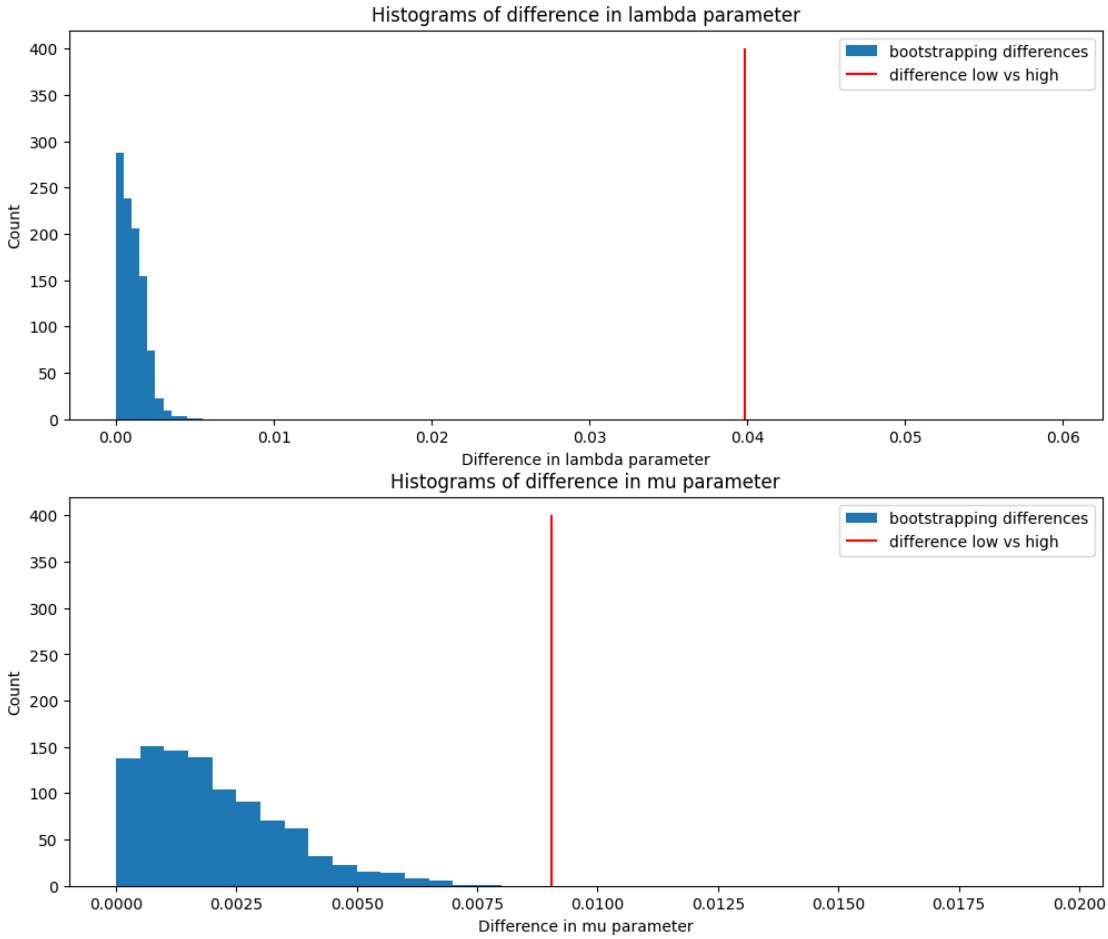

CDF function empirical values and inv gaussian fit low light condition
inv gaussian fit
empirical
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
ISI [s]


CDF function empirical values and inv gaussian fit fit high light condition
1.0
0.8
0.6
0.4
0.2
0.0
inv gaussian fit
empirical
0.0
0.1
0.2
0.3
0.4
0.5
ISI [s]





Low light condition: mu: 0.039988397284383186, lambda: 0.04931816769253932
High light condition: mu: 0.030941974963219623, lambda: 0.009498135387175857
Difference: mu: 0.009046422321163563, lambda: 0.03982003230536346



A4 conclusion

With the inverse Gaussian fit there is a better fit for both the low and high light intensities. The inverse Gaussian fit models the refractory period and bursting a lot better than the exponential fit. Upon inspecting the KS plots, it is clear that the fit for both the low and high intensity light are significantly good. Both conditions lie inside the confidence interval of the KS plot.

There is a significant difference in the lambda values; the high ligh condition has a lower value compared to the low light condition. This is because there is a lot more bursting in the high light condition. Also for the mu parameter low light condition has a significant higher value compared to the high light condition. The low light condition has a higher average probability than the high light condition because the ISIs are more spread out.

A5: From Decoding to Encoding models

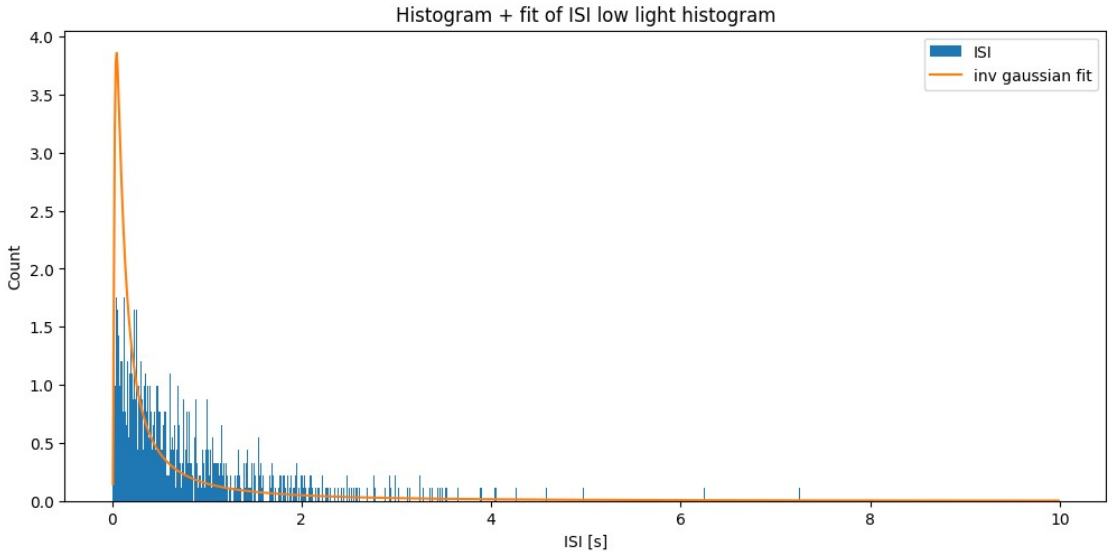
- Go back to Q5

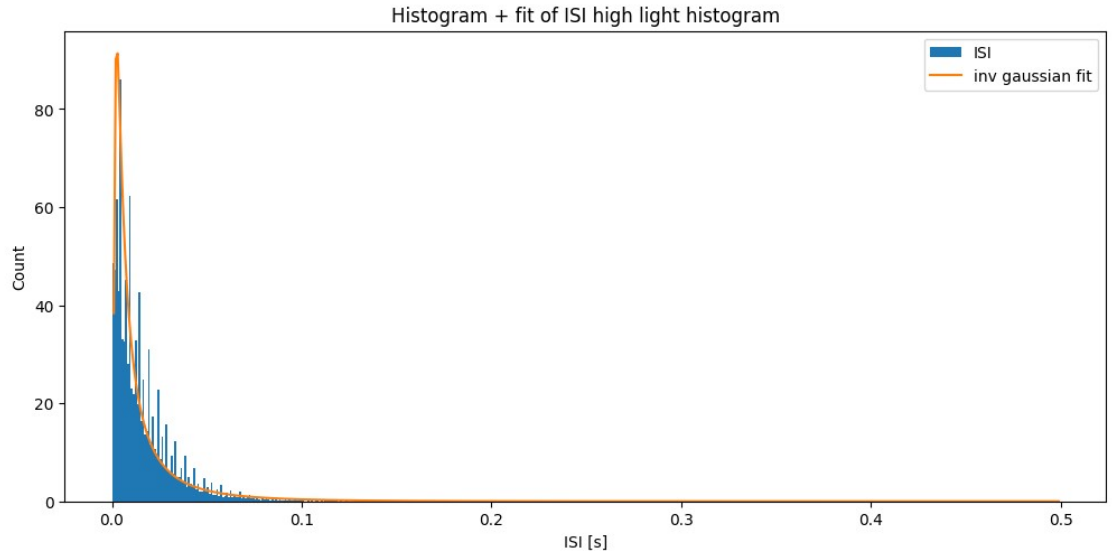
```
# Your Code Goes Here
data = sio.loadmat('matfiles/ANspikes.mat')  # Load the spike train
data
print(data.keys())

fibers_low = data['LS'][0]
fibers_high = data['HS'][0]


dict_keys(['__header__', '__version__', '__globals__', 'LS', 'LS_P',
'HS', 'HS_P'])

# use the same parameters as in previous questions.

######################
##   Q5 solution   ##
######################

ISI_low = np.diff(fibers_low)
ISI_high = np.diff(fibers_high)
fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
bin_edges_low = arange(0, 10, 0.01)  # Define the bins for the
histogram
prob_density_low, bins_low, _ = ax1.hist(ISI_low, bin_edges_low,
density=True, label='ISI')

mu_low = np.mean(ISI_low)
Lambda_low = (np.mean(1/ISI_low - 1/mu_low))**(-1)

inv_gaussian_low =
np.sqrt(Lambda_low/(2*np.pi*bin_edges_low**3))*np.exp(-
Lambda_low*(bin_edges_low-mu_low)**2/(2*bin_edges_low*mu_low**2))
ax1.plot(bin_edges_low, inv_gaussian_low, label='inv gaussian fit')
ax1.set_title('Histogram + fit of ISI low light histogram')
ax1.set_xlabel('ISI [s]')
ax1.set_ylabel('Count')
ax1.legend()

bin_edges_high = arange(0, 0.5, 0.001)
prob_density_high, bins_high, _ = ax2.hist(ISI_high, bin_edges_high,
density=True, label='ISI')
mu_high = np.mean(ISI_high)
Lambda_high = (np.mean(1/ISI_high - 1/mu_high))**(-1)

inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges_high**3))*np.exp(-
Lambda_high*(bin_edges_high-mu_high)**2/(2*bin_edges_high*mu_high**2))
ax2.plot(bin_edges_high, inv_gaussian_high, label='inv gaussian fit')
ax2.set_title('Histogram + fit of ISI high light histogram')
ax2.set_xlabel('ISI [s]')
```

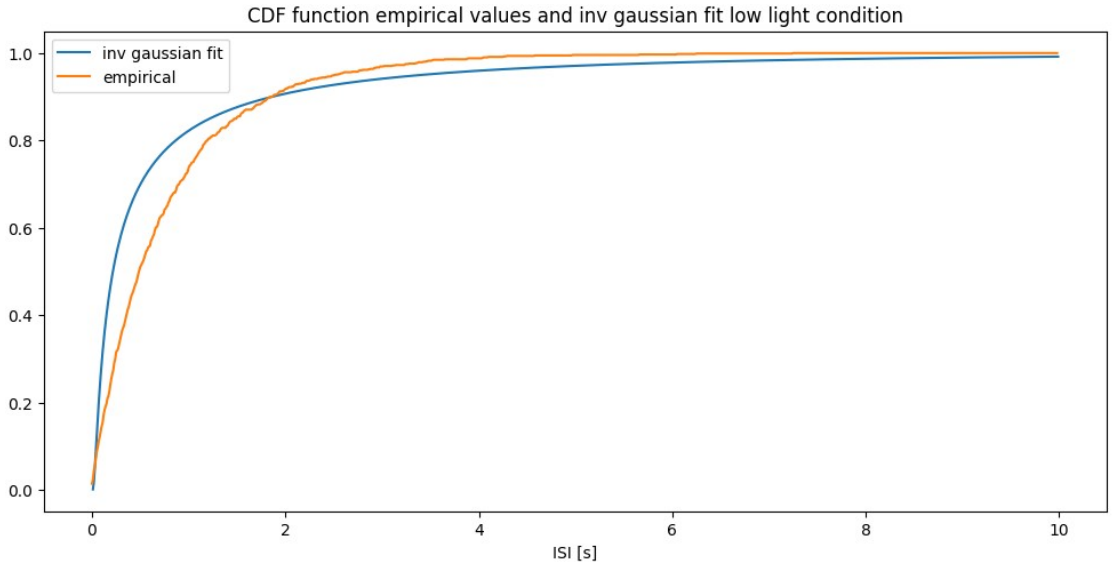
```
ax2.set_ylabel('Count')
ax2.legend()
plt.show()

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
CDF_invgauss_low = np.cumsum(inv_gaussian_low[1:]) *0.01 # multiply
with binwidth, 1ms
ax1.plot(bin_edges_low[1:], CDF_invgauss_low, label='inv gaussian
fit')

CDF_emp_low = np.cumsum(prob_density_low)*0.01 # multiply with
binwidth, 1ms
ax1.plot(bin_edges_low[:-1], CDF_emp_low, label='empirical')
ax1.set_title('CDF function empirical values and inv gaussian fit low
light condition')
ax1.set_xlabel('ISI [s]')
ax1.legend()


CDF_invgauss_high = np.cumsum(inv_gaussian_high[1:])*0.001 # multiply
with binwidth, 1ms
ax2.plot(bin_edges_high[1:], CDF_invgauss_high,label='inv gaussian
fit')
CDF_emp_high = np.cumsum(prob_density_high)*0.001 # multiply with
binwidth, 1ms
ax2.plot(bin_edges_high[:-1], CDF_emp_high, label='empirical')
ax2.set_title('CDF function empirical values and inv gaussian fit high
light condition')
ax2.set_xlabel('ISI [s]')
ax2.legend()
plt.show()

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
N_low = len(CDF_emp_low)
ax1.plot(CDF_invgauss_low, CDF_emp_low)
ax1.plot(CDF_emp_low,CDF_emp_low, 'k')
ax1.fill_between(CDF_emp_low, CDF_emp_low+1.36/np.sqrt(N_low),
CDF_emp_low-1.36/np.sqrt(N_low), alpha=0.5, color='r')
ax1.set_xlabel('inv gaussian')
ax1.set_ylabel('empirical')
ax1.set_title('KS plot low light condition')

N_high = len(CDF_emp_high)
ax2.plot(CDF_invgauss_high, CDF_emp_high)
ax2.plot(CDF_emp_high,CDF_emp_high, 'k')
ax2.fill_between(CDF_emp_high, CDF_emp_high+1.36/np.sqrt(N_high),
CDF_emp_high-1.36/np.sqrt(N_high), alpha=0.5, color='r')
ax2.set_xlabel('inv gaussian')
ax2.set_ylabel('empirical')
ax2.set_title('KS plot low light condition')
```

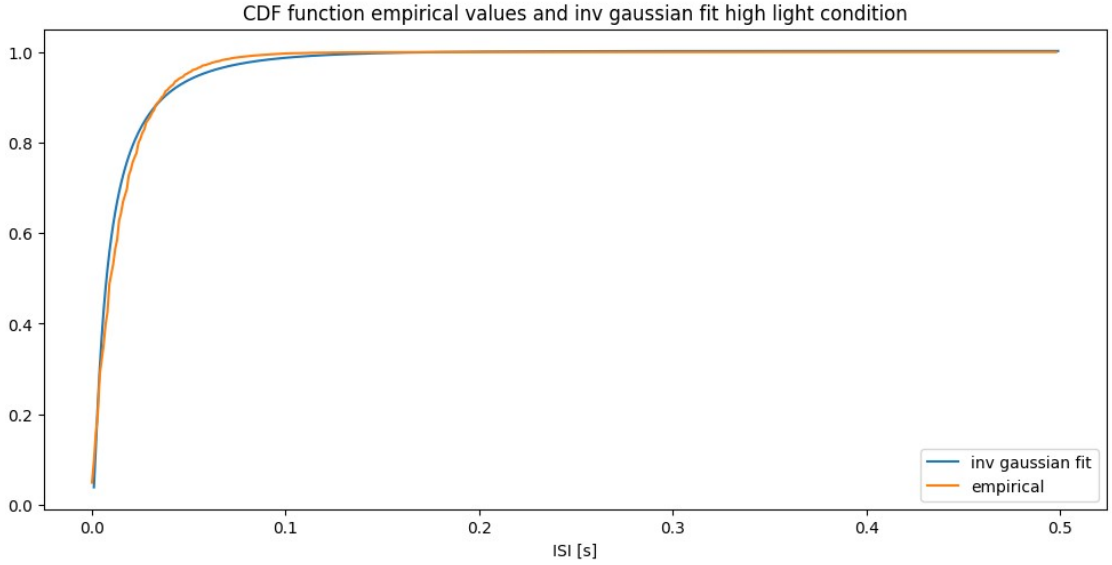
```
plt.show()

print('Low condition: mu: {}, lambda: {}'.format(mu_low, Lambda_low))
print('High condition: mu: {}, lambda: {}'.format(mu_high,
Lambda_high))
```

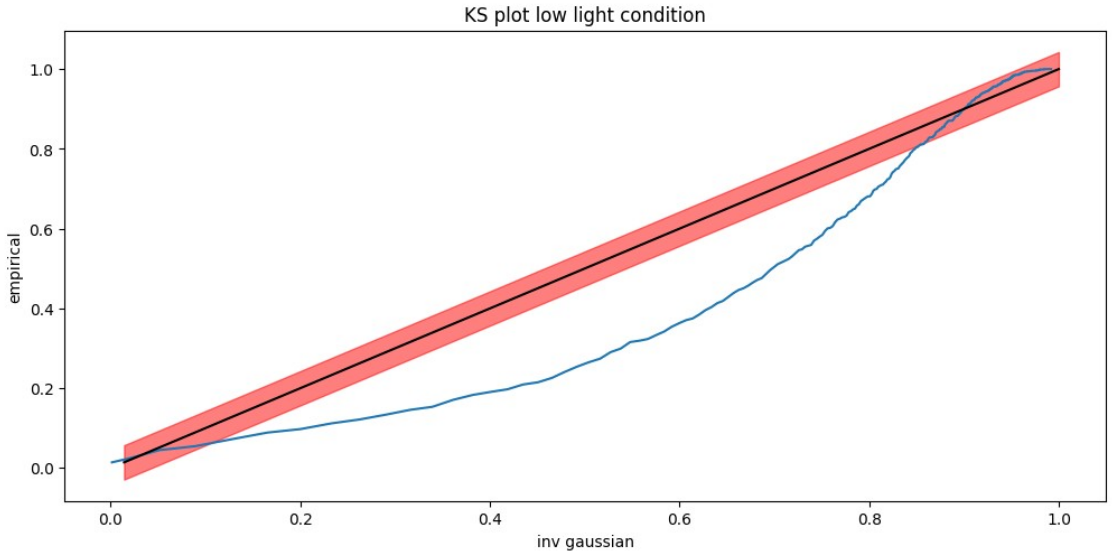
```
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2371617083.py:16:
RuntimeWarning: divide by zero encountered in divide
  inv_gaussian_low =
np.sqrt(Lambda_low/(2*np.pi*bin_edges_low**3))*np.exp(-
Lambda_low*(bin_edges_low-mu_low)**2/(2*bin_edges_low*mu_low**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2371617083.py:16:
RuntimeWarning: invalid value encountered in multiply
  inv_gaussian_low =
np.sqrt(Lambda_low/(2*np.pi*bin_edges_low**3))*np.exp(-
Lambda_low*(bin_edges_low-mu_low)**2/(2*bin_edges_low*mu_low**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2371617083.py:28:
RuntimeWarning: divide by zero encountered in divide
  inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges_high**3))*np.exp(-
Lambda_high*(bin_edges_high-mu_high)**2/(2*bin_edges_high*mu_high**2))
C:\Users\robbe\AppData\Local\Temp\ipykernel_14340\2371617083.py:28:
RuntimeWarning: invalid value encountered in multiply
  inv_gaussian_high =
np.sqrt(Lambda_high/(2*np.pi*bin_edges_high**3))*np.exp(-
Lambda_high*(bin_edges_high-mu_high)**2/(2*bin_edges_high*mu_high**2))
```

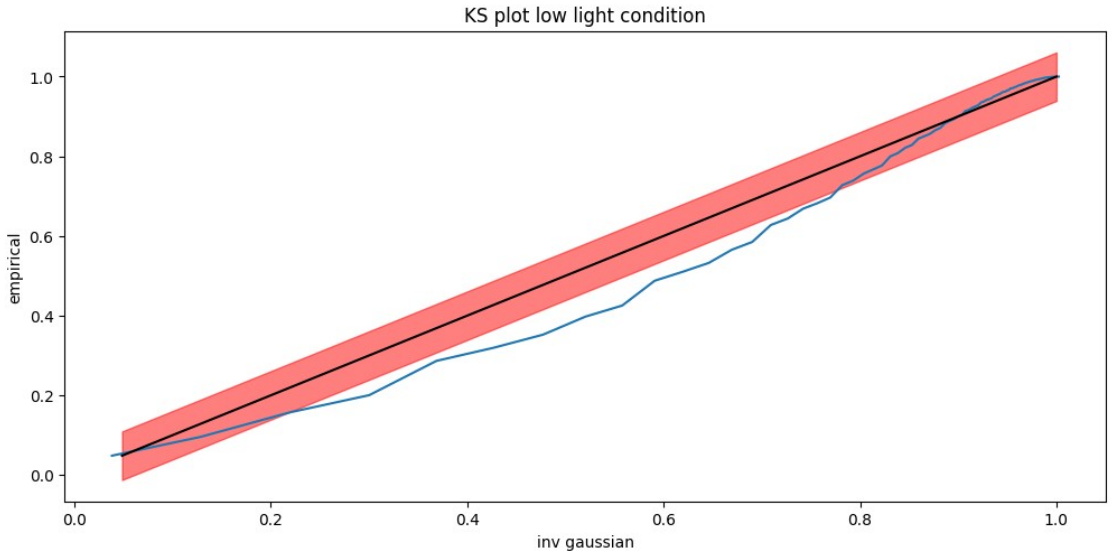

Histogram + fit of ISI low light histogram
ISI
inv gaussian fit
Count
ISI [s]
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
2
4
6
8
10


Histogram + fit of ISI high light histogram
ISI
inv gaussian fit
Count
ISI [s]
0
20
40
60
80
0.0
0.1
0.2
0.3
0.4
0.5


CDF function empirical values and inv gaussian fit low light condition
inv gaussian fit
empirical
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
ISI [s]


CDF function empirical values and inv gaussian fit high light condition
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
ISI [s]
inv gaussian fit
empirical






```
Low condition: mu: 0.78685197368419, lambda: 0.142523597464061
High condition: mu: 0.016108533202073717, lambda: 0.007633231499226967
```

A5 conclusion

At first glance of the fits on the histograms appear to be good. For the low light condition there is an overestimation of the low ISI values.

After creating the CDF and plotting the KS curves it is clear that for both cases the fit is not good enough. It does come close for the high light condition.

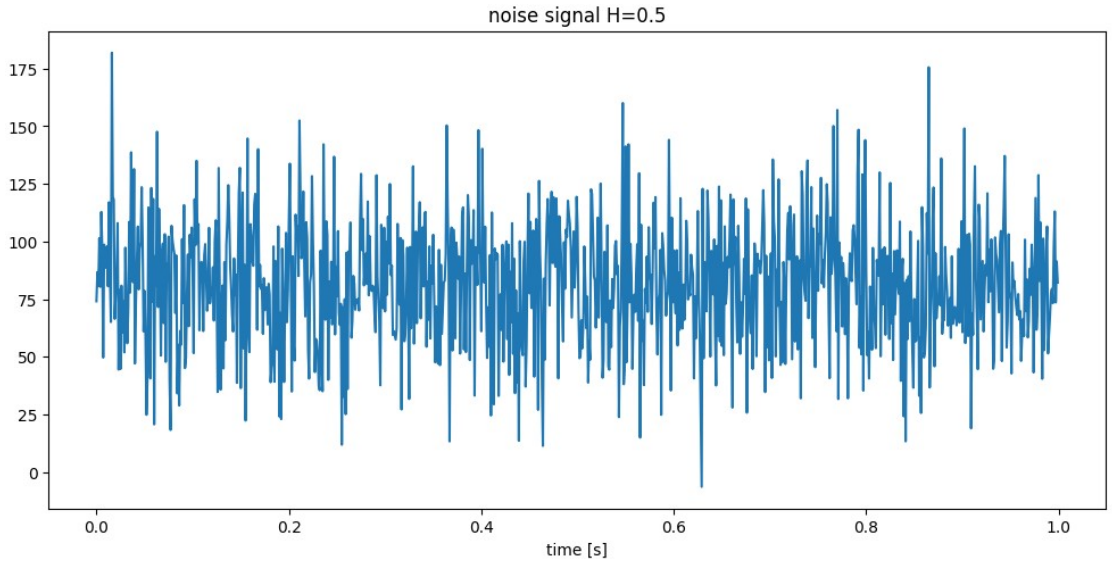
A6: Inhomogenous integrate-and-fire encoder model

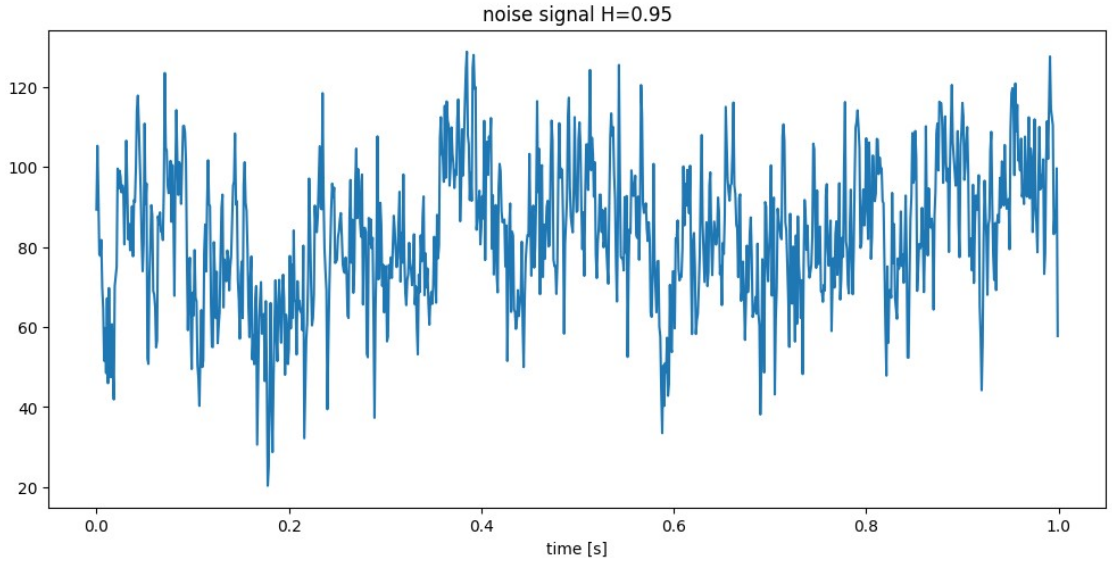
- Go back to Q6

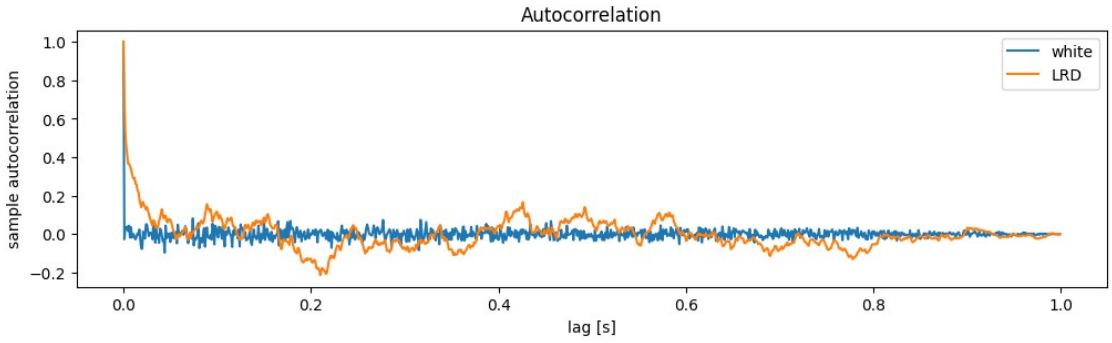
```
# use the following parameters for Q6.1:

# rate_white (your rate over time signal with H=0.5 with an average of
80 spikes/s)
# rate_LRD (your rate over time signal with H=0.95 with an average of
80 spikes/s)
# ACF_white (autocorrelation of rate_white for the first 1000 data
points)
# ACF_LRD (autocorrelation of rate_LRD for the first 1000 data points)

# plot the first 1000 data points, not all 30000
# similar for the autocorrelation, plot both ACF's in one figure

# use the following parameters for Q6.2:
# spiketimes_white (the first output parameter of the inhomPP
function, H=0.5)
# spiketimes_LRD (the first output parameter of the inhomPP function,
H=0.95)

########################
##   Q6.1 solution   ##
########################
noise_time = np.arange(0, 30, 0.001)
SR = 80 # spikes / s
Fs = 1000
H_white = 0.5
H_LRD = 0.95
rate_white = ffGn(30*Fs, 1/Fs, H_white, SR) + SR
rate_LRD = ffGn(30*Fs, 1/Fs, H_LRD, SR) + SR

fig, (ax1,ax2) = plt.subplots(2,1,figsize=(12,12))
ax1.plot(noise_time[:1000], rate_white[:1000])
ax1.set_title('noise signal H=0.5')
ax1.set_xlabel('time [s]')
ax2.plot(noise_time[:1000], rate_LRD[:1000])
ax2.set_title('noise signal H=0.95')
ax2.set_xlabel('time [s]')

plt.show()

def autocorr(spike_data, lags=1000):
    x_corr = np.correlate(spike_data-np.mean(spike_data), spike_data-
np.mean(spike_data), 'full')
    x_corr = x_corr[x_corr.size//2:] / np.max(x_corr)
    return x_corr[:lags+1]

ACF_white = autocorr(rate_white[:1000])
ACF_LRD = autocorr(rate_LRD[:1000])
```

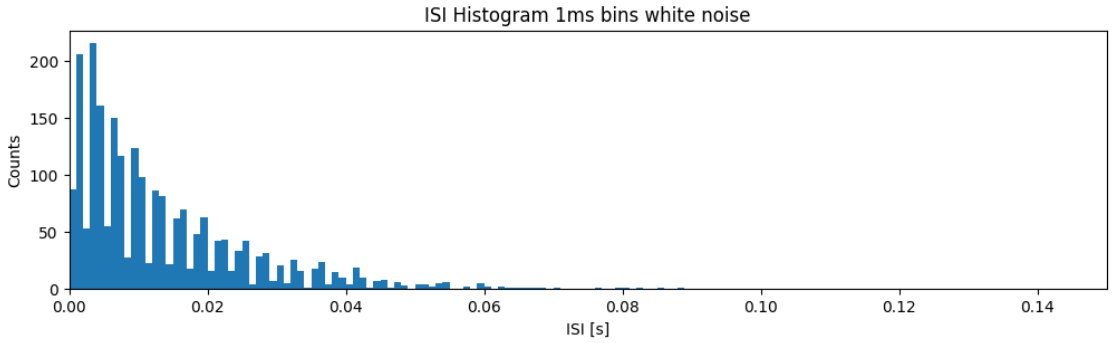
```
fig, ax = plt.subplots(1, 1)
x = np.arange(0, 1, 0.001)
ax.plot(x, ACF_white, label='white')
ax.plot(x, ACF_LRD, label='LRD')
# ax.fill_between(x, -CI_bound_1, CI_bound_1, color='grey', alpha=0.3,
                # label='CI: still likely to be generated by chance')
# ax.set_title('Q6.3 Sample autocorrelation as a function of lag (1ms
bins)')
ax.set_xlabel('lag [s]')
ax.set_ylabel('sample autocorrelation')
ax.set_title('Autocorrelation')
ax.legend()
# ax.set_ybound(upper=0.05)
plt.show()

########################
##   Q6.2 solution   ##
########################

spiketimes_white, PSref_white = inhomPP(rate_white,1/Fs)
spiketimes_LRD, PSref_LRD = inhomPP(rate_LRD,1/Fs)

ISI_white = np.diff(spiketimes_white)
ISI_LRD = np.diff(spiketimes_LRD)
ISI_white_ref = np.diff(sorted(PSref_white))
ISI_LRD_ref = np.diff(sorted(PSref_LRD))

bin_edges = arange(0, 0.501, 0.001)  # Define the bins for the
histogram

plt.hist(ISI_white, bin_edges)        # Plot the histogram of the low-
light ISI data
xlim([0, 0.15])            # ... focus on ISIs from 0 to 150 ms
ylabel('Counts')
xlabel('ISI [s]')# ... label the y-axis
title('ISI Histogram 1ms bins white noise')             # ... give the
plot a title
plt.show()

bin_edges = arange(0, 0.501, 0.001)
plt.hist(ISI_white_ref, bin_edges)        # Plot the histogram of the
low-light ISI data
ylabel('Counts')
xlim([0, 0.05])
xlabel('ISI [s]')# ... label the y-axis
title('ISI Histogram 1ms bins PD_ref')             # ... give the plot
a title
plt.show()
```

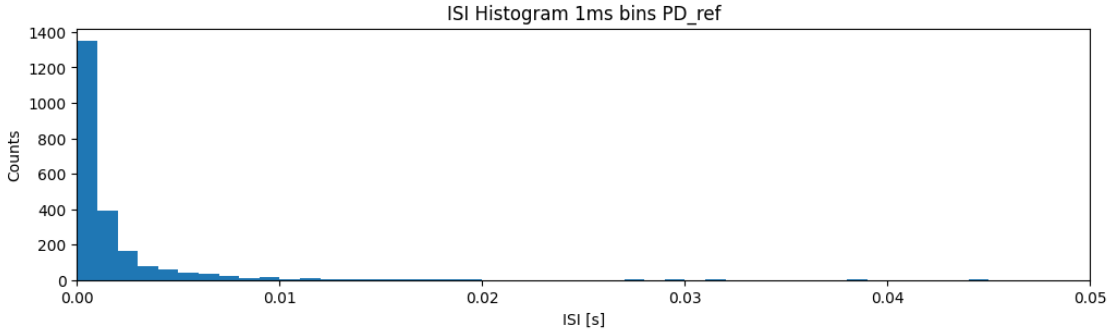
```
plt.hist(ISI_LRD, bin_edges)        # Plot the histogram of the low-
light ISI data
xlim([0, 0.15])            # ... focus on ISIs from 0 to 150 ms
ylabel('Counts')
xlabel('ISI [s]')# ... label the y-axis
title('ISI Histogram 1ms bins LRD noise')             # ... give the
plot a title
plt.show()

plt.hist(ISI_LRD_ref, bin_edges)        # Plot the histogram of the
low-light ISI data
ylabel('Counts')
xlim([0, 0.05])
xlabel('ISI [s]')# ... label the y-axis
title('ISI Histogram 1ms bins PD_ref')             # ... give the plot
a title
plt.show()
```

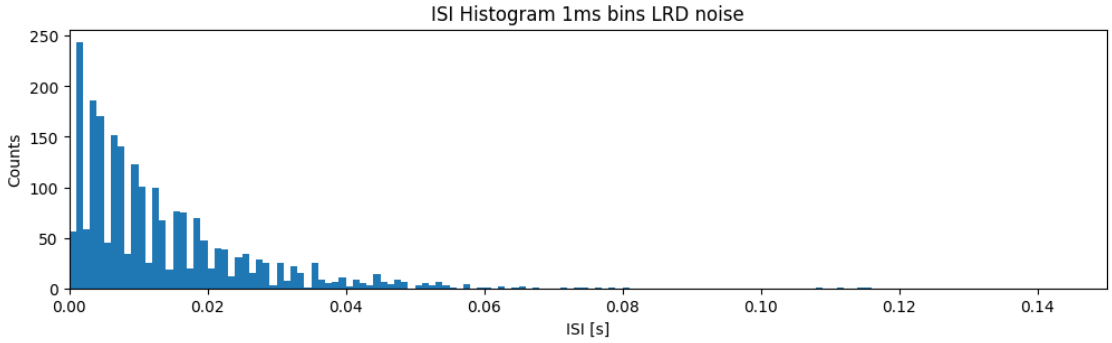

noise signal H=0.5
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
time [s]

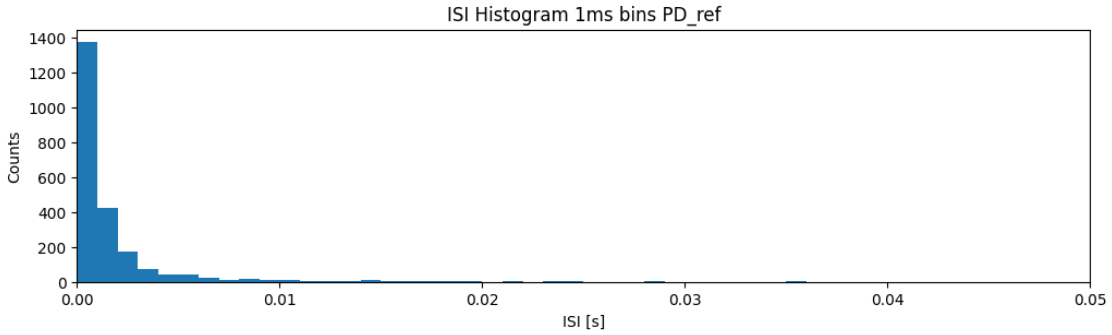

noise signal H=0.95
120
100
80
60
40
20
0.0
0.2
0.4
0.6
0.8
1.0
time [s]


Autocorrelation
white
LRD
sample autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
lag [s]









A6.1 conclusion

It is clear from the plots of the simulated signals that the noise signal with H=0.5 (white noise) is completely random. On the plot of the noise signal with H=0.95 there is some pattern visible. Some low frequency oscilatory behaviour is present.

The autocorrelation of the white noise remains zero. This means that there is no correlation at any lag value. The LRD noise signal has eleveted autocorrelation values for low lag values. For the remaining lag values there is some damped oscillatory behaviour. This shows that there is a large probability of finding another spike shortly after a spike has occured.

A6.2 conclusion

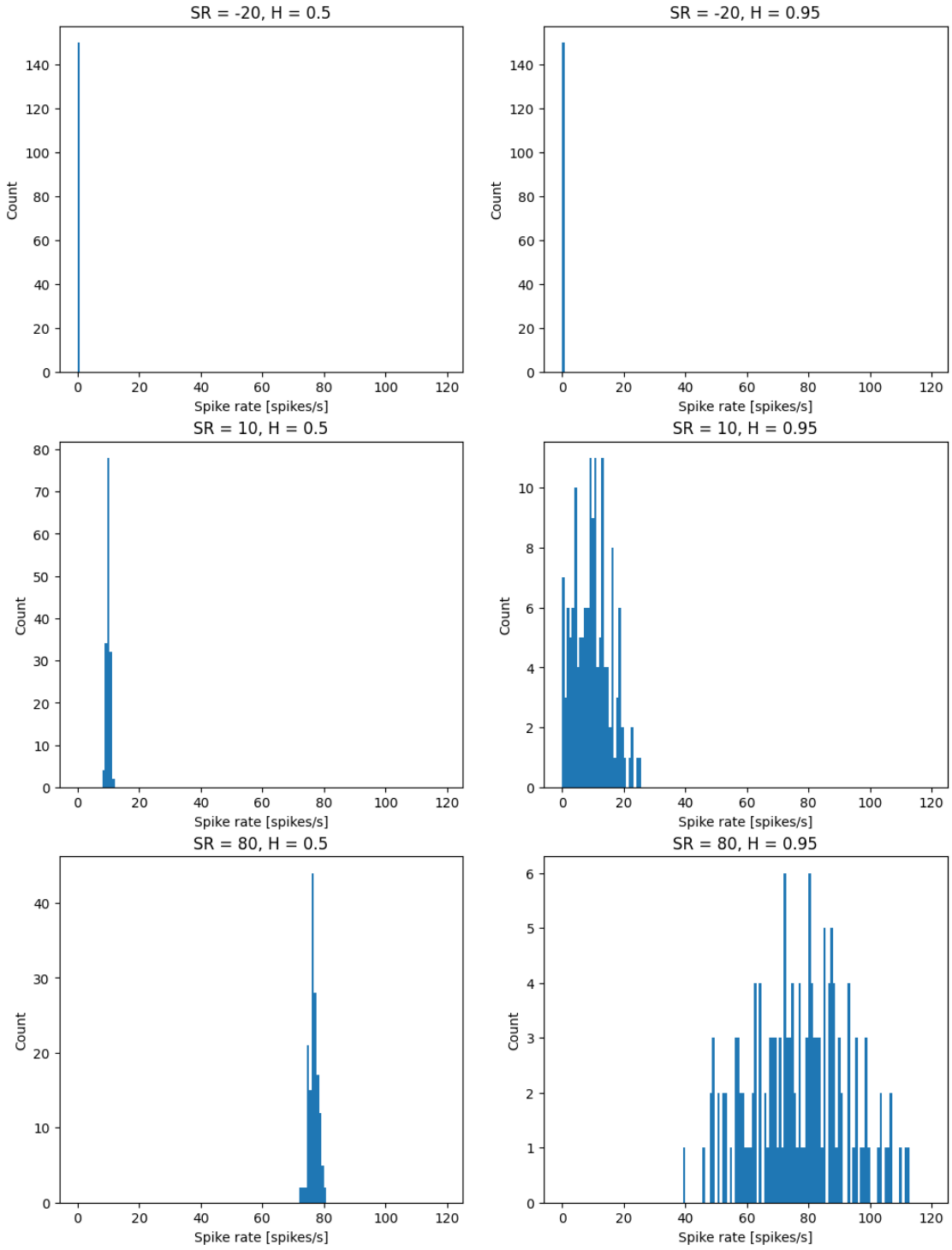
At first glance, the ISI histograms of H=0.5 and H=0.95 have a similar shape. It is noticable that the historgram of H=0.95 shows larger counts for larger ISI values. (This elevation is however very small). This matches with the explanation from the description in the question. For higher H exponents higher counts for high ISI values are expected. The control value PSref only has high counts for small ISI values.

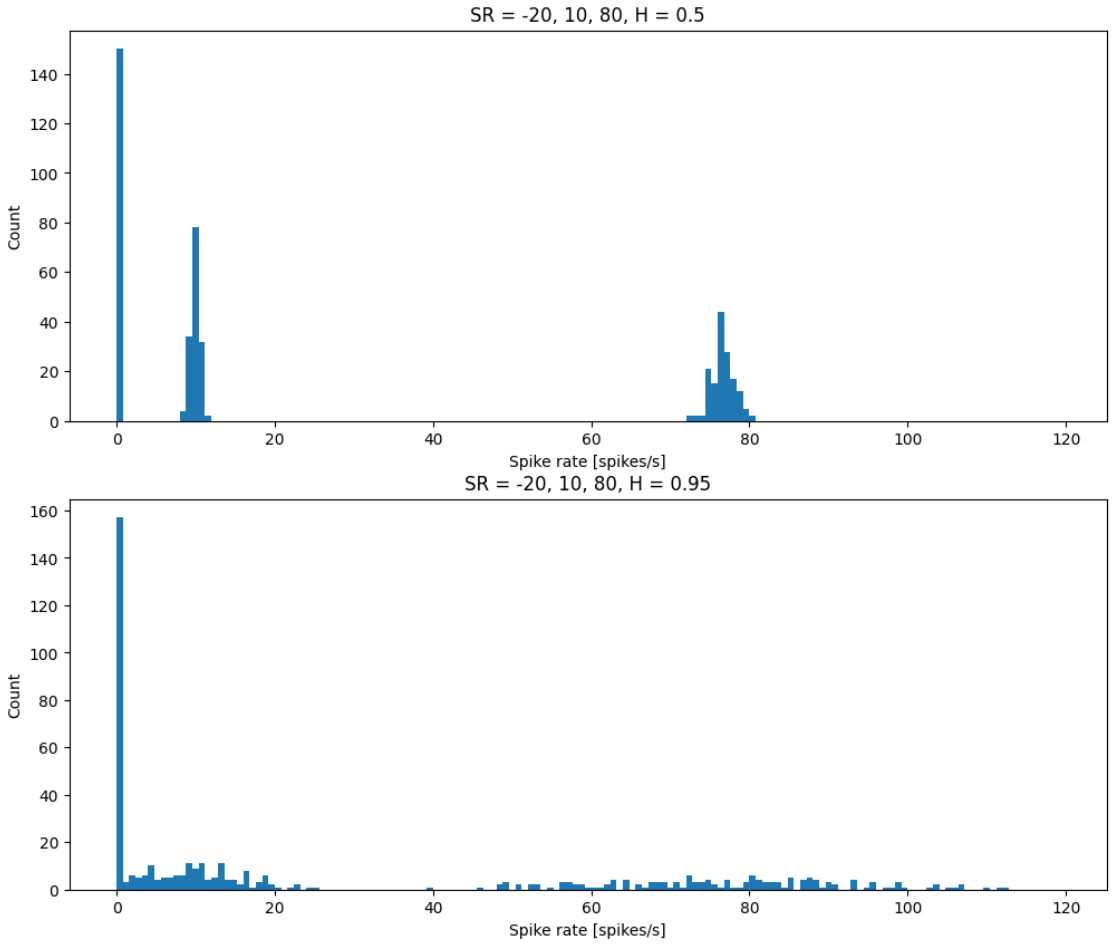
A7: Investigate the research hypothesis

- Go back to Q7

```
# Your Code Goes Here
SR = 80

def jackson(SR, H):
    mean_SR = []
    for exp in range(150):
        rate = ffGn(30*Fs, 1/Fs, H, SR) + SR
        spiketimes , PSref = inhomPP(rate,1/Fs)
        mean_SR.append(len(spiketimes)/30)

    return mean_SR


# Make the figures here
fig, ax = plt.subplots(3, 2, figsize=(12,16))
mean_SR_20_LDR = jackson(-20, 0.95)
bin_edges = arange(0, 120, .01*SR)
ax[0,1].hist(mean_SR_20_LDR, bin_edges)
ax[0,1].set_title('SR = -20, H = 0.95')
ax[0,1].set_xlabel('Spike rate [spikes/s]')
ax[0,1].set_ylabel('Count')

mean_SR_20_white = jackson(-20, 0.5)
bin_edges = arange(0, 120, .01*SR)
ax[0,0].hist(mean_SR_20_white, bin_edges)
ax[0,0].set_title('SR = -20, H = 0.5')
ax[0,0].set_xlabel('Spike rate [spikes/s]')
ax[0,0].set_ylabel('Count')
```

```
mean_SR_10_LDR = jackson(10, 0.95)
bin_edges = arange(0, 120, .01*SR)
ax[1,1].hist(mean_SR_10_LDR, bin_edges)
ax[1,1].set_title('SR = 10, H = 0.95')
ax[1,1].set_xlabel('Spike rate [spikes/s]')
ax[1,1].set_ylabel('Count')

mean_SR_10_white = jackson(10, 0.5)
bin_edges = arange(0, 120, .01*SR)
ax[1,0].hist(mean_SR_10_white, bin_edges)
ax[1,0].set_title('SR = 10, H = 0.5')
ax[1,0].set_xlabel('Spike rate [spikes/s]')
ax[1,0].set_ylabel('Count')


mean_SR_80_LDR = jackson(80, 0.95)
bin_edges = arange(0, 120, .01*SR)
ax[2,1].hist(mean_SR_80_LDR, bin_edges)
ax[2,1].set_title('SR = 80, H = 0.95')
ax[2,1].set_xlabel('Spike rate [spikes/s]')
ax[2,1].set_ylabel('Count')

mean_SR_80_white = jackson(80, 0.5)
bin_edges = arange(0, 120, .01*SR)
ax[2,0].hist(mean_SR_80_white, bin_edges)
ax[2,0].set_title('SR = 80, H = 0.5')
ax[2,0].set_xlabel('Spike rate [spikes/s]')
ax[2,0].set_ylabel('Count')

plt.show()

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(12,10))
ax1.hist(np.concatenate((mean_SR_20_white,mean_SR_10_white,mean_SR_80_
white)), bin_edges)
ax1.set_title('SR = -20, 10, 80, H = 0.5')
ax1.set_xlabel('Spike rate [spikes/s]')
ax1.set_ylabel('Count')

ax2.hist(np.concatenate((mean_SR_20_LDR,mean_SR_10_LDR,mean_SR_80_LDR)
), bin_edges)
ax2.set_title('SR = -20, 10, 80, H = 0.95')
ax2.set_xlabel('Spike rate [spikes/s]')
ax2.set_ylabel('Count')
plt.show()
```


SR = -20, H = 0.5
SR = -20, H = 0.95
SR = 10, H = 0.5
SR = 10, H = 0.95
SR = 80, H = 0.5
SR = 80, H = 0.95
Count
Spike rate [spikes/s]



A7 conclusion

The generated images resemble the same effect as shown in the paper. The white noise simulations have a narrow SR distribution, since there is no temporal correlation. The LRD histograms have a broader distribution. The negative SR seems non physiological. It represents a spiking system where the mean of the process driving the action potential generator is below a threshold.

Using the inhomogeneous poisson process with 3 different SR and including LRD gives the same result as in the paper. Thus their hypothesis that the behaviour of AN fibers can be caputred this way is sensible. The use of LRD is needed because this includes the memory and long range dependency.