# Lab Session #3

## 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

# Basic Analysis of Spike Train Data

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 chapter 7 of "Neuronal Dynamics" by Gerstner, Kistler, Naud, Paninski (2014, Cambridge University Press). It uses two different datasets to introduce and develop some the concepts.

## Aim

Typically, the first step in any data analysis involves visualizing and using simple descriptive statistics to characterize pertinent features of the data. For time series data that take on a continuous value at each time point, like the field potentials analyzed in earlier notebooks, we typically start by simply plotting each data value as a function of time. For spike train data, things can become a bit more complicated. One reason for this is that there are multiple equivalent ways to describe the same spike train data. The data could be stored as a sequence of spike times; as a sequence of waiting times between spikes (interspike intervals); or as a discrete time series indicating the number of spikes in discrete time bins. Knowing how to manipulate and visualize spike train data using all these different representations is the first step understanding the structure of the data and is the primary goal of this notebook.

## Case Study 1: Retina Neuron

Neurons in the retina typically respond to patterns of light displayed over small sections of the visual field. However, when retinal neurons are grown in culture and held under constant light and environmental conditions, they will still spontaneously fire action potentials. In a fully functioning retina, this spontaneous activity is sometimes described as background firing activity, which is modulated as a function of visual stimuli. It is useful to understand the properties of this background activity in order to determine in future experiments how these firing properties are affected by specific stimuli.

### Data

A researcher is examining the background firing properties of one of these neurons. He records the spiking activity in one of two states, with the room lights off (low ambient light levels) or with the room lights on (high ambient light levels) and would like to determine whether there is a difference in background firing between these two conditions, and whether one environment is more conducive to future experimental analyses. He records the spiking activity for 30 seconds in each condition.
**Data:** Spontaneous spiking activity from a retinal neuron in culture, exposed to low-light and high-light environments.

**Goal:** Visualize spike trains, compute and interpret descriptive statistics.

[illegible]

### Before you start

To prepare your environment for this exercise, you'll have to install the nitime package e.g. type `pip install nitime` in the command line after opening conda or your conda environment

In [1]:
```python
# Prepare the standard modules and load in the data
from pylab import *
import scipy.io as sio
import warnings
%matplotlib inline
rcParams['figure.figsize']=(12,3)  # Change the default figure size
import matplotlib.pyplot as plt

# and notebook specific modules
from scipy.stats import gamma      # Import the gamma object from the SciPy stats toolbox
warnings.simplefilter(action='ignore', category=FutureWarning)
data = sio.loadmat('matfiles/08_spikes-1.mat')  # Load the spike train data
print(data.keys())

#You should find two variables in the `data` dictionary:
#`SpikesLow`: spike times over 30 s in the low ambient light condition,
#`SpikesHigh`: spike times over 30 s in the high ambient light condition.
#We take these two variables from `data` so we can work with them directly.
spikes_low = data['SpikesLow'][0]
spikes_high = data['SpikesHigh'][0]
#Each variable is a single (1-D) vector that gives a set of increasing spike times for the associated conditio
#Each row holds a single spike time.
#The two vectors are of different sizes (SpikesLow.shape) because the neuron fired a different number of spike
import pandas as pd
```

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

### Q1: Compute the firing rate (in Hz) of the neuron in the low- and high-light condition

Firing rate ($f$) is mathematically defined as

$$f = \frac{n}{T}, \tag{1}$$

where $n$ is the number of spikes over the time interval $T$.

- Fill in answer here

### Q2: Visualise the spike train data

- Q2.1: Plot the spike data and identify which variables are plotted on the x- and y-axis.
- Q2.2: The standard way of plotting does not yield dot-raster plots (spikes over time), you can generate such a plot by using the `ones_like()` function. This function produces an array of 1s (ones) that has the same dimensions as the input. When plotting your spikes, use the `'.'` marker and remove the $y$-axis tick marks since they don't carry any real information, in this case `yticks([])`. Compare the low and high-light spike traces in the same figure (and label them), you can do so by multiplying the `ones_like()` function by two to yield separate traces. Compare the traces within 5 s intervals, as well as across the 30s, can you identify marked visual differences between the low and high-light conditions?
- Fill in answer here

### Q3: Spike Counts

A common approach analyzing spiking data is to discretize time into bins of fixed width and count the number of events(spikes) in each time bin. The length of the time window is set by the experimenter and depends on the type of neuron and stimulus. Bin sizes are often chosen such that multiple spikes occur in a single bin to yield a meaningfull analysis of the spike train and its variability across time bins. To build statistical models of spike trains, the time bins are taken sufficiently small, say, 1 ms for a typical spike train, such that the resulting binning process (also called increments) is just a sequence of zeros and ones. In this case, the probability that more than one spike occurs in each bin is zero or negligibly small (and similar to actual spiking statistics with refractoriness). Each tiny time bin then contains a spike (assign that bin value 1) or does not (assign that bin value 0). In our case, we use multiple bin sizes to characterize the variability of the spiking data and to examine temporal dependencies between spikes.

- Q3.1: Start by binning the spike train data of low and high-light conditions into time bins of 50 ms. To do so, use the functions `np.histogram()` and `np.arange()` . How many bins have 4 spikes in them, and how many bins have no spikes in them (for both conditions)?
- Q3.2: A question that often arises is how variable the binned counts are. To illustrate this variability, let's consider two scenarios. In the first, consider a neuron that fires perfectly regularly, like a metronome. In this case, we expect the number of spikes in each time bin to be nearly identical. On the other hand, consider the scenario of a neuron that fires in irregular bursts. In this case, we expect much more variability in the number of spikes in each time bin, depending on whether a time bin contained a burst of spikes or a quiet period. To characterize this variability, a standard measure to compute is the sample ***Fano factor (FF)***. It's easy to calculate the Fano factor: the sample variance of the binned counts (increments) divided by the sample mean of the binned counts (increments). Use the built in python methods `.mean()` and `.var()` to calculate the FF.

### Q4: Fano Factors

To understand what the resulting FF means, we can compare the variability of the increments of a spike train (i.e. the FF) to the variability of a Poisson process. When considering spiking activity as a Poisson Process, it is assumed that each spike occurrence is independent of every other spike occurrence. Poisson processes are not necessarily accurate models for spike train data as our biological knowledge reveals that the occurrence of a spike does depend on the occurrence of previous spikes (e.g., because of the refractory period of a neuron, we do not expect a spike to occur immediately after another spike). However, because of the mathematical properties of the Poisson probability distribution, its theoretical variance and mean are equal, yielding a FF of 1, and a good model compare our results to. If the **FF is well below 1** for a particular set of increments, this suggests that the spiking is more regular than a Poisson process for the time scale at which the increments were binned. In this case, spiking activity in the past is influencing the neuron to spike in a more predictable manner in subsequent bins. If the **FF is well above 1**, this suggests that the spiking is more variable than a Poisson process for the time scale at which the increments were binned.

- Q4.1: How do you interpret the FFs you obtained for the low and high-light conditions?

Next, we'd like to get an idea of how far above or below the value of 1 the calculated FF has to be before we can be confident that there is a statistically significant difference in the variability from a Poisson process. After all, even if we had spiking from a true Poisson process, from one experiment to the next we would expect to find different values for the increments, and FF values that fluctuate slightly above and below 1. It can be shown theoretically that the distribution of FFs that we might compute from a Poisson process follows a gamma distribution with shape parameter $(N-1)/2$ and scale parameter $2/(N-1)$, where $N$ is the number of time bins used in the FF calculation [Eden & Kramer, 2010 (https://doi.org/10.1016/j.jneumeth.2010.04.012)].

- Q4.2: Use the `gamma.pdf()` function from scipy to generate a the probability density function that goes with the Poisson process and calculate the confidence interval bounds (2.5 and 97.5% points) outside which FFs are significantly different from the FF belonging to a Poisson process to answer the question as to whether your low- or high-light spiking rates are significantly different from a Poisson process (Motivate your answer).Use the `gamma.ppf()` function from scipy.

- Q4.3: How do the results for the Fano factor change in each condition with different choices for the bin size of the increment process (e.g., 25 ms, 100 ms, 400 ms)?
- Fill in answer here

**Q5: Interspike interval distributions**

A common method to study neural variability given a certain stationnary input, is by means of the interspike-interval (ISI) distributions. This method can quantify bursts of spikes that cluster near each other in time, or identify periods that contain less spikes. If you zoom in on your dot-raster plots (e.g. in a 0.4s window), you can observe qualitative differences in the spiking structure between the low- and high-light conditions.

Q5.1: Use the `np.diff()` function to study the ISI for each condition, and afterwards use a histogram to visualize and compare the ISIs in 20 ms and 2ms bins. Describe the features of the two histograms. What features of the ISI distributions are similar for the two conditions? What features are most strikingly different? Quantitatively describe how the neurons spike in response to the two conditions (and on which time scales).

- Fill in answer here

**Q6: Autocorrelation functions (ACF)**

For a stationary input scenario where a neuron fires a first spike at time t, the ISI distribution describes the probability that the next spike occurs at t+s. Instead, the autocorrelation function

$$C_{xx}[s]$$

focuses on the probability of finding **another** spike at time t+s, independent of whether this is the next spike of the neuron or not. Mathematically, the formula for the sample autocorrelation at a lag $s$ is

$$C_{xx}[s] = \frac{\sum_{n=1}^{N-s}(x_n - \bar{x})(x_{n+s} - \bar{x})}{\sum_{n=1}^{N}(x_n - \bar{x})^2}$$

where $x_n$ is the $n^{th}$ data point, and $\bar{x}$ is the sample mean of the data over index $n$.

- Q6.1: When considering the ISI histograms of Q5, we might expect relations between spiking events to extend up to 200 ms. First, compute the full autocorrelation `correlate(,'full')` for the low and high-light condition using increments of 50 ms (50ms binned histogram outputs) for lag 0 and 3 more lags (lag0=0ms-50ms, lag1=50ms-100ms, lag2=100ms-150ms, lag3=150ms-200ms). What is the correlation coefficient at lag 0 and did you expect this value? How can we interpret the values of the correlation values at the other considered lags?

- Q6.2: To assess whether the correlation coefficients are significant, you can approximate the confidence interval based on a standard normal distribution with standard deviation
$$1/\sqrt(N)$$
, for which any correlation value exceeding
$$\pm 2/\sqrt(N)$$
is unlikely to be generated by chance. Calculate the significance boundary and compare your found correlation factors to them to answer whether they reflect significantly correlated relationships. Plot the autocorrelation values as a function of lag, and also plot two lines that indicate the two confidence boundaries.
- Q6.3: To get a more fine-grained view on the autocorrelation function, generate a new histogram with 1-ms bins, and calculate the autocorrelation functions for lags corresponding to 100-ms (100 lags). Plot the autocorrelation values as in Q6.2. For which time-lags do the values in the low and high-light condition exceed the confidence boundaries, and what does this mean for the spiking process?
- Q6.4: To answer the question as to whether the differences in the autocorrelations between these two conditions are real, you can compute the difference in the autocorrelation functions between the low- and high-light conditions at every lag (for the histogram with 1ms bins). If we assume that the firing in each condition is independent, the significance bounds for this difference can be computed by the square root of the sum of the variances of the autocorrelation from each condition. The standard deviation of the autocorrelation for the low-light condition is $1/\sqrt{N_1}$, so the variance of the auto- correlation for the low-light condition is $1/N_1$. For the high-light condition, the variance of the autocorrelation is $1/N_2$. Plot the differenced autocorrelations and the significance bounds, to answer whether there are significantly different autocorrelations in the two conditions.

## Case Study 2: The subthalamic nucleus

### Q7: Peri-stimulus-time histograms (PSTH)

Before we delve deeper into the statistical models underlying neuronal spiking, we'll introduce one more common experimental neuroscience technique that is based on repeating the stimulation sequence several times. The neuronal responses are reported in a peri-stimulus time histogram (PSTH) with bin width $\Delta t$ (in the order of $ms$) and $t$ measured with respect to the start of the stimulation sequence. We'll apply this concept on an experiment in which a clinical neurosurgeon is analyzing spiking data from patients with Parkinson’s disease during surgery to implant DBS electrodes. To assess whether the electrode is localized properly in the STN (subthalamic nucleus), and to study STN activity during the planning and execution of a movement. Each patient as asked multiple times to perform a simple hand movement task during the electrode implantation procedure, while the resulting neural activity is recorded.

The data of the hand movement task consists of single unit spiking activity from one STN neuron recorded over 50 trials. In each trial, a patient held a joystick and watched cues appear on a screen. The first cue indicated whether the patient should move the joystick to the left or right. The second cue, the GO cue, indicated the time to start moving the joystick. The dataset contains 2s of activity for each trial, 1s before the GO cue and 1s after. We label the period before the GO cue as the planning period, and the period after the GO cue as the movement period.

**GOAL:** Some STN neurons in patients with Parkinson's disease shows rhythmic properties in the beta frequency range, 11–30 Hz ([Jasper & Andrews, 1938] (https://doi.org/10.1001/archneurpsyc.1938.02270010106010), [Brittain & Brown, 2014] (https://doi.org/10.1016/j.neuroimage.2013.05.084)), hence the neurosurgeon hypothesizes that information related to the movement task, such as a) the planning versus the movement period and b) whether the movement is to the left or right, will influence this rhythmic spiking

In [2]:

```python
# Load in the modules for this notebook and the data itself
from nitime.algorithms import multi_taper_psd
from nitime.utils import dpss_windows
import statsmodels.api as sm
from statsmodels.genmod.families import Poisson
import statsmodels.formula.api as smf
from pandas import DataFrame as df
from pandas import concat
from statsmodels.distributions.empirical_distribution import ECDF
from scipy.stats import chi2, norm
rcParams['figure.figsize']=(12,3)                # Change the default figure size

data2 = sio.loadmat('matfiles/10_spikes-1.mat') # Load the spike train data.
t = data2['t'][0]                          # Extract the t variable.
direction = data2['direction'].flatten()  # Extract the direction variable.
train = data2['train']                     # Extract the train variable
```

The dataset has three variables:

- `direction` : A 50×1 vector of indicators for the movement direction for each of 50 trials. A value of 0 indicates movement to the left, and a value of 1 indicates movement to the right.
- `t` : A 1×2000 vector of time stamps in milliseconds indicating the time into the trial. Time 0 indicates the GO cue.
- `train` : A 50×2000 matrix of spike values (1 if spike, 0 if not) for each ms and for each trial. The rows of the matrix correspond to trials, while the columns correspond to time(ms).

- Q7.1: Start by visualizing your dataset in a dot-raster plot. You can use the `plt.imshow()` function to visualize Matrixes with 0's and 1's. Group the left and right trial responses, do you notice qualitative differences between them, and is rythmic spiking apparent?
- Q7.2: PSTHs are useful for visualizing the relation between spike rates per bin. To compute the PSTH, the spikes times are partitioned into bins, and the number of spikes that occur within each bin are added across trials, and then divided by the number of trials and the length of the bin. In other words, this is a normal histogram but now averaged over all trials. Scale your y-axis to spike rate per second per bin. Use a 1ms, and later 10ms bin-size and functions `np.where()` , `np.histogram()` , and the `plt.bar()` plotting function to evaluate whether spiking is different in the planning and movement periods. What effect does the choice of bin-size have on your results interpretation, and what is a good bin size in your opinion?
- Q7.3: Next, quantify the average firing rate in planning and movement periods by averaging the spiking data across trials and time in each period. Are these consistent with your visual inspections? Afterwards, break down the rates into those belonging to the left and right movement tasks, are there differences?
- Q7.4: Using a boxplot, you can also visualise the trial-to-trial variability distribution in each of the 4 conditions (plan-left, movement-left, plan-right, movement-right). Use 1s time-windows for this analysis and use the `plt.boxplot()` function for this analysis. What is the S.I. unit corresponding to the variables in your boxplot, and which conditions yield significantly different results (based on the graphical boxplot results)
- Fill in answer here

**Q8: Rhythmic structure**

Boxplots were helpful to visualise the long-term spiking structure over periods of about 1 s, but do not provided much information about short-term rhythmic structure. At the same time, it was our goal to identify whether there was rhythmic structure in the spiking activitiy.

- Q8.1: To this end, you can first compute the ISI histograms of the planning and movement case. Afterwards, average the ISI histograms accross trials to compare whether there are visual differences between the ISI's of the planning and movement case.
- Q8.2: Another function to visualize history-dependent spike properties, is the sample **autocorrelation function** (ACF). Recall that the ACF shows the correlation between a signal at two points in time separated by a fixed lag. Different lags are evaluated. For these spike train data, let’s use the increments (i.e., the number of spikes in each time bin) as the signal, compute the ACF for each trial, and average the results over all the trials.
- Q8.3: So far, we have focused on statistical characterizations of spike trains in the time domain. Another approach to visualizing and describing structure in point processes focuses on estimating rhythmic (or harmonic) features of the data in the frequency domain. The autocorrelation function is closely linked to the power spectrum of a neuronal spike train, in fact, the power spectrum of a spike train equals the Fourier transform of its autocorrelation function (Wiener-Khinchin theorem - see lecture). Look up and use the `multi_taper_psd()` function (Nitime package) to compute the power spectral density for each trial (planning and movement periods separately) and average the results across trials.

The planning and movement periods for each trial are 1 s in duration, so we can achieve a 4 Hz frequency resolution by setting the time-bandwidth product to 4. This allows us to use seven tapers, each providing independent and informative estimates. Tapering is a common method to avoid artifacts from discontinuities created by non-periodic data near the window boundaries (periodicity is a basic assumption in FFT). More info on multi-tapers: https://en.wikipedia.org/wiki/Multitaper (https://en.wikipedia.org/wiki/Multitaper) You can also calculate the spectrum by hand. The main reason for this is that we are working with point process data and most packages assume that a signal is continuous. Define a function mt_specpb to calculate the spectrum manually.
In this case, because the mean firing rate is so low, the difference is trivial. The multi_taper_psd() function from the Nitime package thus produces nearly the same result as mt_specpb, though it must be rescaled to see this. which line of the function shows that it treats the data as data points instead of continuous data?

- Q8.4: While computing the spectrums, we inherently assumed that the data of the planning period are stationary. In other words, that the mean and autocovariance structure do not change in time. We then concluded a sudden change between the planning and the movement periods and again that the data are stationary again during the movement period. To test whether this is a reasonable assumption, you can compute a spectrogram, using your `mt_specpb` function and `plt.contourf`. Use a moving window of 500 ms duration and a step size of 50 ms. Was the stationarity assumption reasonable for this dataset?

- Fill in answer here

**A1: Compute the firing rate (in Hz) of the neuron in the high- and low-light condition**

- Go back to Q1

In [3]:
```python
# Use the following variables:
#f_low
#f_high

#####################
##    Q1 solution   ##
#####################

T = 30 # [s]
f_low = len(spikes_low)/T # [Hz]
f_high = len(spikes_high)/T # [Hz]

print('f_low  = ', f_low, 'Hz')
print('f_high = ', f_high, "Hz")
```

```
f_low  =  25.0 Hz
f_high =  32.3 Hz
```

**A1 conclusion**

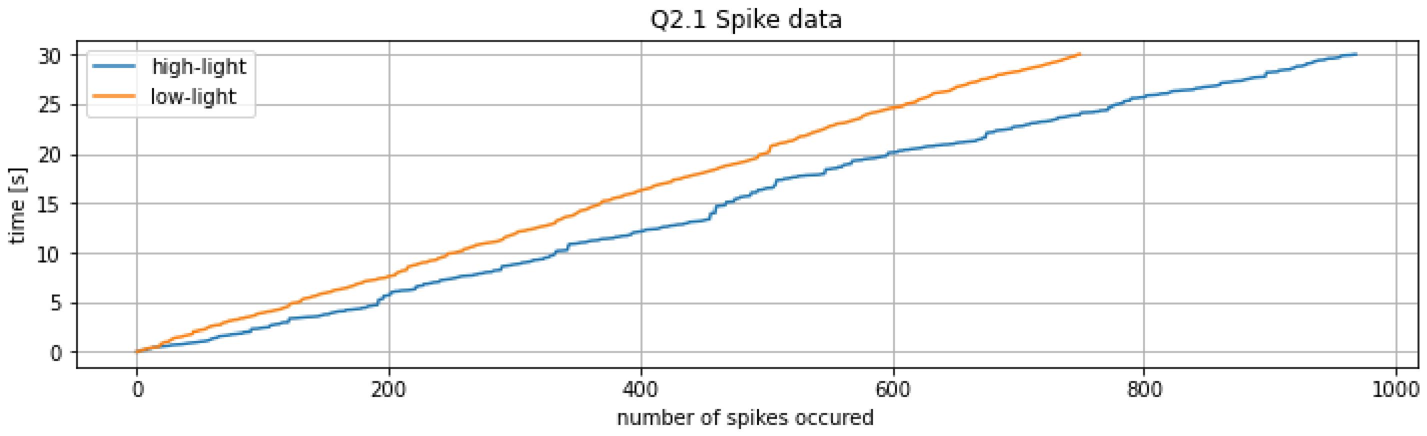
A higher ambient light condition results in a higher firing rate (32.3 Hz) than the low light condition (25.0 Hz).

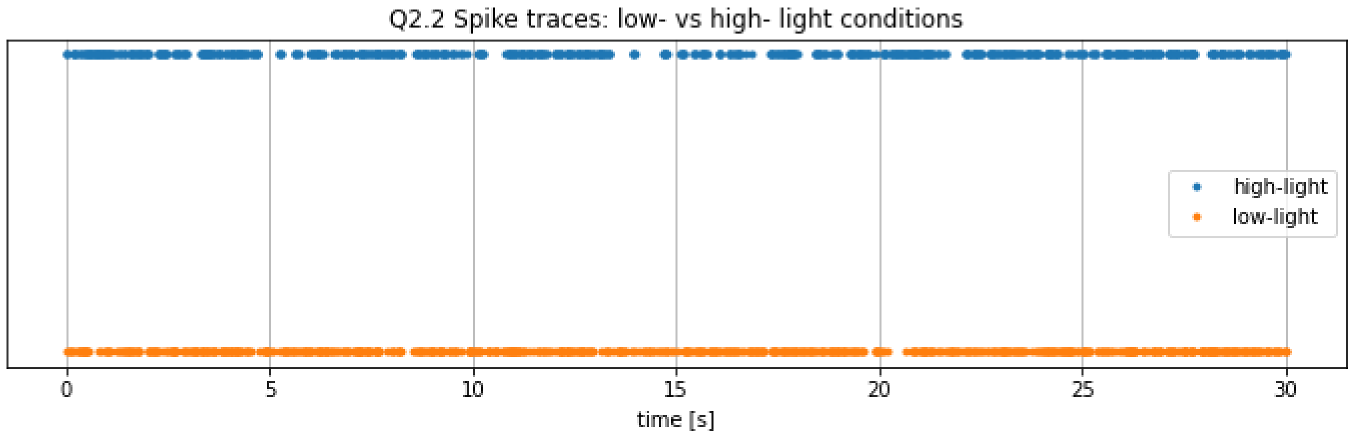
### A2: Visualise the spike train data

- Go back to Q2

In [4]:

```python
# Your start data is spikes_low and spikes_high

#######################
##   Q2.1 solution   ##
#######################

fig, ax = plt.subplots(1, 1, figsize=(12, 3))

ax.plot(spikes_high, label='high-light')
ax.plot(spikes_low, label='low-light')

ax.set_title('Q2.1 Spike data')
ax.set_xlabel('number of spikes occured')
ax.set_ylabel('time [s]')
ax.grid()
ax.legend()

plt.show()

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

fig_2, ax = plt.subplots(1, 1, figsize=(12, 3))

ax.plot(spikes_high, ones_like(spikes_high)*2,'.', label='high-light')
ax.plot(spikes_low, ones_like(spikes_low),'.', label='low-light')

ax.set_title('Q2.2 Spike traces: low- vs high- light conditions')
ax.set_xlabel('time [s]')
ax.set_yticks([])
ax.grid()
ax.legend()

plt.show()
```


Q2.1 Spike data
high-light
low-light
time [s]
30
25
20
15
10
5
0
0
200
400
600
800
1000
number of spikes occured


Q2.2 Spike traces: low- vs high- light conditions
high-light
low-light
0
5
10
15
20
25
30
time [s]

**A2 conclusion**

- If one plots the spike data, one can appreciate that both the *spikes_low* and *spikes_high* arrays result in: *number of spikes occured* and *elapsed time* on the x- and y-axis respectively.
- A more conventional way of plotting spike data is the use of a dot-raster plot (Q2.2): *elapsed time* on the x-axis and the discrete events (*spikes*) indicated as dots along this axis. One can see that along the complete 30s recording, the spikes occur more frequently in the high-light condition than in the low-light condition. When comparing the 5s intervals, there also seems to be more variability in spike frequency for the high-light condition than the low-light condition (i.e. longer periods of no spiking, followed by more rapid spiking).

### A3: Spike Rates

- Go back to Q3

In [5]:

```python
# Hint: if you use np.arange (as suggested), make sure that also your end time is included in the range.
# Hint: you can also use np.linspace, make sure that you want the number of time_edges of the bins, not the num
# Hint: you can use plt.hist() to plot but dont use plt.hist() to calculate the histogram values because it ret

# Use the following variables:
#increments_low_50
#increments_high_50
#time_bin_edges
#bins_low_4spikes
#bins_low_0spikes
#bins_high_4spikes
#bins_high_0spikes

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

binsize = 0.050 # [s]
time_bin_edges = np.arange(0, T+binsize, binsize)

increments_low_50, _ = np.histogram(spikes_low, bins=time_bin_edges)
increments_high_50, _ = np.histogram(spikes_high, bins=time_bin_edges)

bins_low_4spikes = (increments_low_50 == 4)*1
bins_low_0spikes = (increments_low_50 == 0)*1
bins_high_4spikes = (increments_high_50 == 4)*1
bins_high_0spikes = (increments_high_50 == 0)*1

# plt.hist(spikes_low, bins=time_bin_edges)
# plt.hist(spikes_high, bins=time_bin_edges)

# from tabulate import tabulate
# print(tabulate(df, headers = 'keys', tablefmt = 'latex_raw'))
```

In [6]:
```
# Use the following variables:
#FF_low_50
#FF_high_50

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

FF_low_50 = var(increments_low_50)/mean(increments_low_50)
FF_high_50 = var(increments_high_50)/mean(increments_high_50)

dict = {
    "#bins with 4 spikes": [np.sum(bins_low_4spikes), np.sum(bins_high_4spikes)],
    "#bins with 0 spikes": [np.sum(bins_low_0spikes), np.sum(bins_high_0spikes)],
    "Fano factor (FF)": [np.round(FF_low_50, 3), np.round(FF_high_50, 3)]
    }
df = pd.DataFrame(dict, index=['low-light', 'high-light'])
display(df)

# from tabulate import tabulate
# print(tabulate(df, headers = 'keys', tablefmt = 'github'))
```

| | #bins with 4 spikes | #bins with 0 spikes | Fano factor (FF) |
|---|---|---|---|
| **low-light** | 6 | 136 | 0.715 |
| **high-light** | 44 | 226 | 1.775 |

### A3 conclusion

*The results of binning the spike train data in 50ms bins can be found in the table below:*

| | #bins with 4 spikes | #bins with 0 spikes | Fano factor (FF) |
|---|---|---|---|
| low-light | 6 | 136 | 0.72 |
| high-light | 44 | 226 | 1.78 |

> The high-light condition has more bins with 4 spikes and more bins with 0 spikes (with respect to the low-light condtion). The Fano factor (FF) is higher for the high-light condition than the low-light.
>
> From these results, one can conclude that the high-light condition has more variability in the binned counts than the low-light condition.

### A4: Fano Factor

- Go back to Q4

In [7]:

```python
from scipy.stats import gamma   # Import the gamma object from the SciPy stats toolbox

# Use the following variables:
#ppf_low_50 (an array with 2 values: the lower and upper percentile)
#ppf_high_50
#ppf_low_25
#ppf_high_25
#ppf_low_100
#ppf_high_100
#ppf_low_500
#ppf_high_500

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

#see anwer below

#######################
##   Q4.2 solution   ##
#######################

def gamma_function(spike_data, binsize):
    # bin spike_data depending on specified binsize [s]
    time_bin_edges = np.arange(0, T+binsize, binsize)
    increments, _ = np.histogram(spike_data, bins=time_bin_edges)
    FF = var(increments)/mean(increments)

    # construct gamma distribution
    N = len(time_bin_edges)-1
    a = (N-1)/2       # shape parameter
    scale = 2/(N-1)   # scale parameter
    FF_distr = gamma(a, scale=scale)

    # calculate CI bounds
    ppf = FF_distr.ppf([0.025, 0.975])

    return ppf, FF

#######################
##   Q4.3 solution   ##
```

```python
#######################

# Calculate CI boundaries and FF for each binsize (25ms, 50ms, 100ms and 500ms)
ppf_low_25, FF_low_25 = gamma_function(spikes_low, binsize=0.025)
ppf_low_50, FF_low_50 = gamma_function(spikes_low, binsize=0.050)
ppf_low_100, FF_low_100 = gamma_function(spikes_low, binsize=0.100)
ppf_low_500, FF_low_500 = gamma_function(spikes_low, binsize=0.500)

ppf_high_25, FF_high_25 = gamma_function(spikes_high, binsize=0.025)
ppf_high_50, FF_high_50 = gamma_function(spikes_high, binsize=0.050)
ppf_high_100, FF_high_100 = gamma_function(spikes_high, binsize=0.100)
ppf_high_500, FF_high_500 = gamma_function(spikes_high, binsize=0.500)

# Return clear table
bounds = np.array([ppf_low_25, ppf_low_50, ppf_low_100, ppf_low_500,
    ppf_high_25, ppf_high_50, ppf_high_100, ppf_high_500])
FF = np.array([FF_low_25, FF_low_50, FF_low_100, FF_low_500,
    FF_high_25, FF_high_50, FF_high_100, FF_high_500])

dict = {
    "2.5 percentile": bounds[:,0].round(3),
    "97.5 percentile": bounds[:,1].round(3),
    "Fano factor (FF)": FF.round(3),
    "Sign. different from Poisson": (FF < bounds[:,0]) | (bounds[:,1] < FF)
    }
index = [
    'low-light 25ms',
    'low-light 50ms',
    'low-light 100ms',
    'low-light 500ms',
    'high-light 25ms',
    'high-light 50ms',
    'high-light 100ms',
    'high-light 500ms'
    ]
df = pd.DataFrame(dict, index=index)
display(df)

# print(tabulate(df, headers = 'keys', tablefmt = 'github'))
```

| | 2.5 percentile | 97.5 percentile | Fano factor (FF) | Sign. different from Poisson |
|---|---|---|---|---|
| **low-light 25ms** | 0.922 | 1.082 | 0.730 | True |
| **low-light 50ms** | 0.890 | 1.116 | 0.715 | True |
| **low-light 100ms** | 0.846 | 1.167 | 0.705 | True |
| **low-light 500ms** | 0.672 | 1.392 | 0.831 | False |
| **high-light 25ms** | 0.922 | 1.082 | 1.445 | True |
| **high-light 50ms** | 0.890 | 1.116 | 1.775 | True |
| **high-light 100ms** | 0.846 | 1.167 | 2.203 | True |
| **high-light 500ms** | 0.672 | 1.392 | 3.213 | True |

### A4.1 answer

From the FFs calculated in A3: the FF is higher ($> 1 \implies$ higher-than-Poisson variability) for the high-light condition than the low-light condition ($< 1 \implies$ sub-Poisson variability).

One could conclude that for the high-light condition, the spiking behaviour was more variable than a Poisson process and for the low-light condition, the spiking behaviour was more regular than a Poisson process.

### A4 conclusion

The results for question Q4.2 and Q4.3 are presented in the table below:

| | 2.5 percentile | 97.5 percentile | Fano factor (FF) | Significantly different from Poisson |
|---|---|---|---|---|
| low-light 25ms | 0.922 | 1.082 | 0.73 | Yes |
| low-light 50ms | 0.89 | 1.116 | 0.715 | Yes |
| low-light 100ms | 0.846 | 1.167 | 0.705 | Yes |

| | 2.5 percentile | 97.5 percentile | Fano factor (FF) | Significantly different from Poisson |
|---|---|---|---|---|
| low-light 500ms | 0.672 | 1.392 | 0.831 | No |
| high-light 25ms | 0.922 | 1.082 | 1.445 | Yes |
| high-light 50ms | 0.89 | 1.116 | 1.775 | Yes |
| high-light 100ms | 0.846 | 1.167 | 2.203 | Yes |
| high-light 500ms | 0.672 | 1.392 | 3.213 | Yes |

By comparing the calculated Fano factor (FF) to the two-sided 95% CI for the Poisson process' FF, the last column of the above table is generated. One can conclude that for all the investigated bin sizes (25ms, 50ms, 100ms & 500ms), the high-light condition shows a significantly higher FF than that of a Poisson process, indicating that the spiking is more variable than a Poisson process at the mentioned time scales.

For the low-light condition, the first three bin sizes (25ms, 50ms & 100ms) result in a significantly lower FF than that of a Poisson process, indicating that the spiking is more regular than a Poisson process at the mentioned time scales. Only for the 500ms bin size, the low-light FF is not significantly different from the Poisson FF; the binning is too coarse to draw meaningful conclusions about the

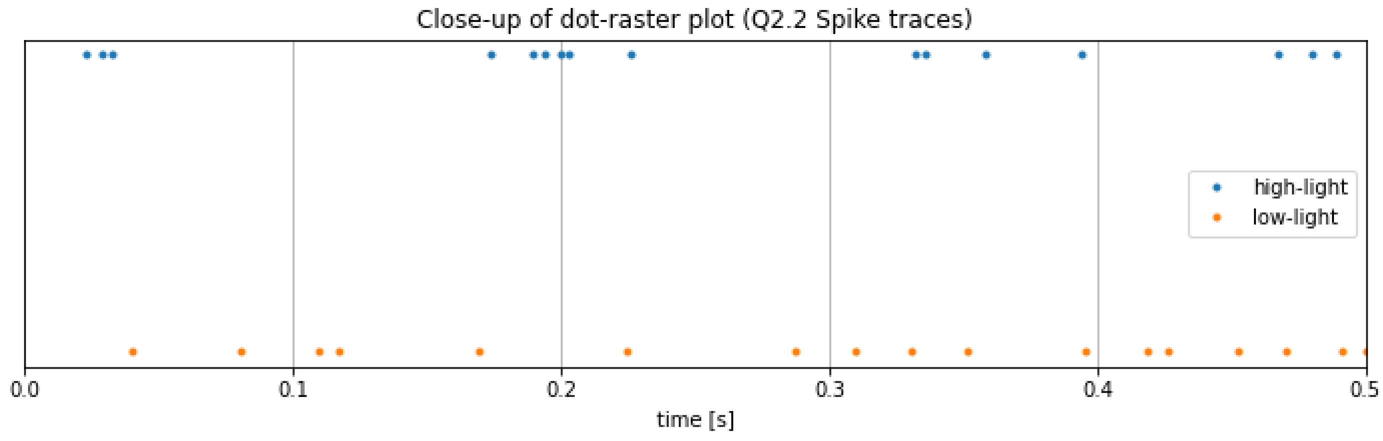
### A5: Interspike Intervals

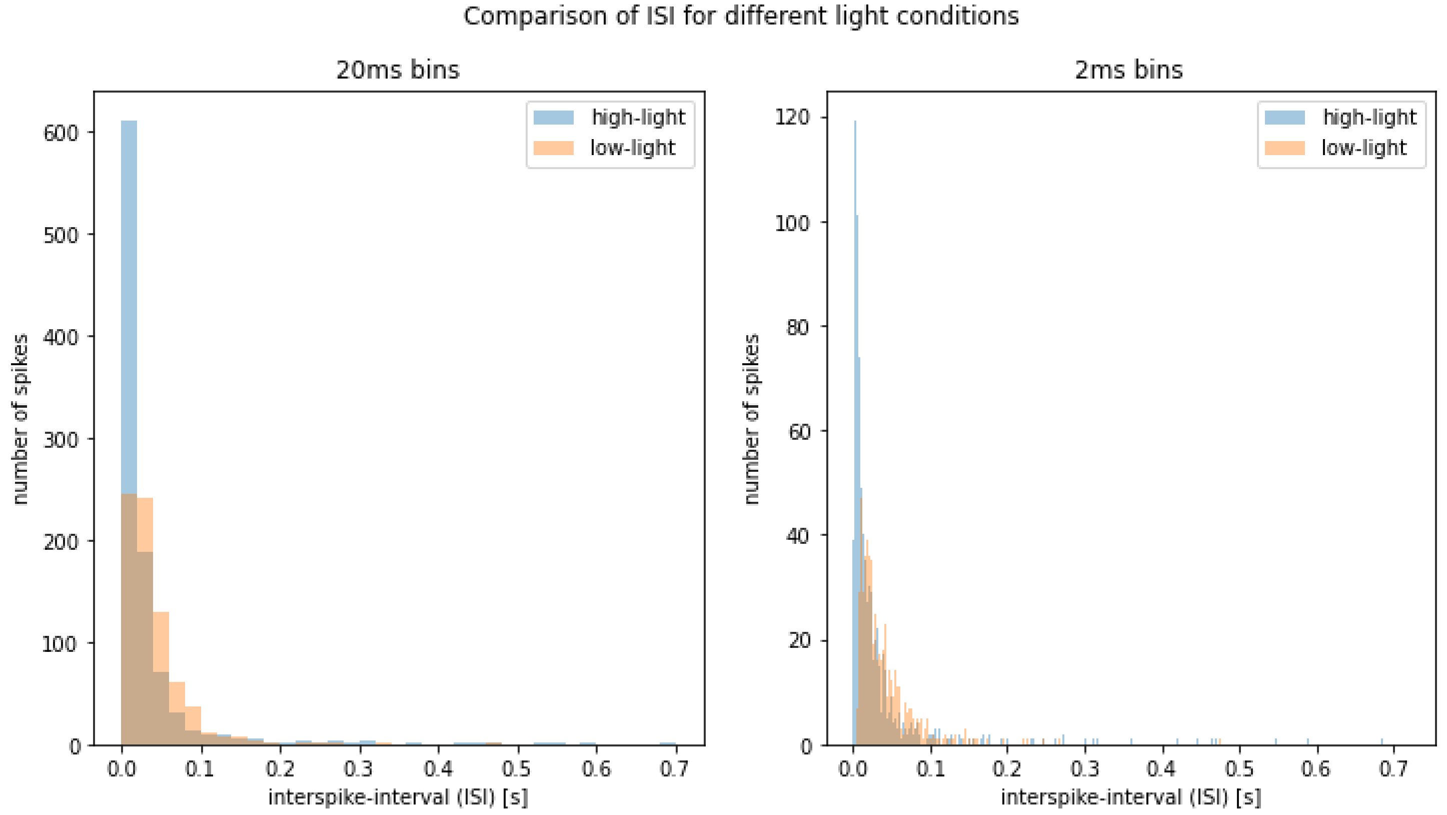
- Go back to Q5

In [8]:

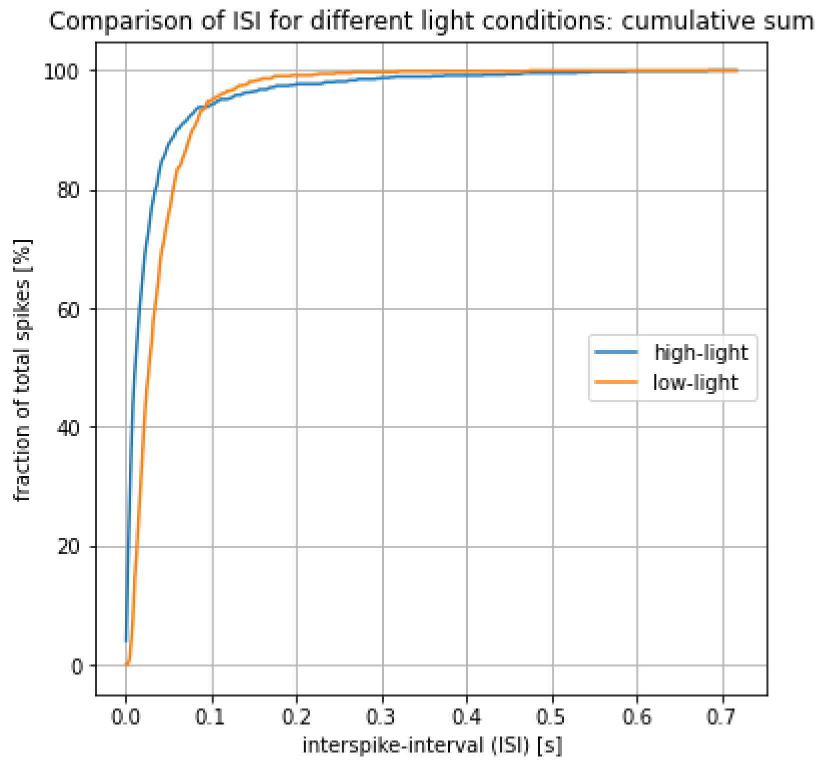
```python
# Use the following variables:

#ISI_low
#ISI_high
#ISI_hist_low_10
#ISI_hist_high_10
#ISI_hist_low_1
#ISI_hist_high_1

#######################
##   Q5.1 solution   ##
#######################

# Close-up of dot-raster plot from Q2.2
plt.figure(fig_2)
plt.title('Close-up of dot-raster plot (Q2.2 Spike traces)')
plt.xlim([0, 0.50])
plt.show()

# Calculation of ISI distributions
ISI_low = np.diff(spikes_low)
ISI_high = np.diff(spikes_high)

# print(max(ISI_high), max(ISI_low))
max_time = 0.72
time_bins_20 = np.arange(0, max_time, 0.020)
time_bins_2 = np.arange(0, max_time, 0.002)

ISI_hist_low_20, _ = np.histogram(ISI_low, time_bins_20)
ISI_hist_high_20, _ = np.histogram(ISI_high, time_bins_20)
ISI_hist_low_2, _ = np.histogram(ISI_low, time_bins_2)
ISI_hist_high_2, _ = np.histogram(ISI_high, time_bins_2)

# Construct plots for comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

ax1.hist(ISI_high, time_bins_20, alpha=0.4, label='high-light')
ax1.hist(ISI_low, time_bins_20, alpha=0.4, label='low-light')
ax2.hist(ISI_high, time_bins_2, alpha=0.4, label='high-light')
ax2.hist(ISI_low, time_bins_2, alpha=0.4, label='low-light')

```

```python
ax1.set_title('20ms bins')
ax1.set_xlabel('interspike-interval (ISI) [s]')
ax1.set_ylabel('number of spikes')
ax1.legend()
ax2.set_title('2ms bins')
ax2.set_xlabel('interspike-interval (ISI) [s]')
ax2.set_ylabel('number of spikes')
ax2.legend()
# ax1.set_xlim([0, 0.2])
# ax2.set_xlim([0, 0.2])
plt.suptitle('Comparison of ISI for different light conditions')
plt.show()

# Calculate most occuring ISI for each condition (2ms bins)
centers_bins_20 = (time_bins_20[:-1]+time_bins_20[1:])/2
centers_bins_2 = (time_bins_2[:-1]+time_bins_2[1:])/2
argmax_low_2 = np.argmax(ISI_hist_low_2)
argmax_high_2 = np.argmax(ISI_hist_high_2)

print(f"""
ISI most occuring for high-light condition: {centers_bins_2[argmax_high_2]} s
ISI most occuring for low-light condition:  {centers_bins_2[argmax_low_2]} s
""")
N_high, N_low = np.sum(ISI_hist_high_2), np.sum(ISI_hist_low_2)

# Extra plot to get a more clear view of the ISI distributions
fig, ax = plt.subplots(1, 1, figsize=(6, 6))

ax.plot(centers_bins_2, 100*ISI_hist_high_2.cumsum()/N_high, label='high-light')
ax.plot(centers_bins_2, 100*ISI_hist_low_2.cumsum()/N_low, label='low-light')
ax.set_title('Comparison of ISI for different light conditions: cumulative sum')
ax.set_xlabel('interspike-interval (ISI) [s]')
ax.set_ylabel('fraction of total spikes [%]')
ax.legend(loc=5)
ax.grid()

plt.show()
```


Close-up of dot-raster plot (Q2.2 Spike traces)
high-light
low-light
time [s]
0.0
0.1
0.2
0.3
0.4
0.5


Comparison of ISI for different light conditions
20ms bins
2ms bins
high-light
low-light
number of spikes
interspike-interval (ISI) [s]
0
100
200
300
400
500
600
20
40
60
80
100
120
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7

```
ISI most occuring for high-light condition: 0.003 s
ISI most occuring for low-light condition:  0.011 s
```



**A5 conclusion**

Both the high-light and low-light conditions result in ISI value distributions which are largely skewed to the right. The most counts for ISI values are situated at lower ISI values for the high-light condition (around 3ms) than for the low-light condition (around 11ms). There are more spike counts (in absolute number) in the high-light case versus the low-light case.

By combining this knowledge with the *cumulative sum comparison plot*, one can appreciate that the high-light condition is characterized by more rapid firing periods, followed by longer waiting periods in between 'bursts' (as the high-light and low-light curves cross-over around an ISI of 95ms). The low-light condition behaves more 'uniform' over time than the high-light condition. This conclusion on the behaviour difference between the high-light and low-light condition is supported by the close-up of the dot-raster plot above.

## A6: Autocorrelation functions (ACF)

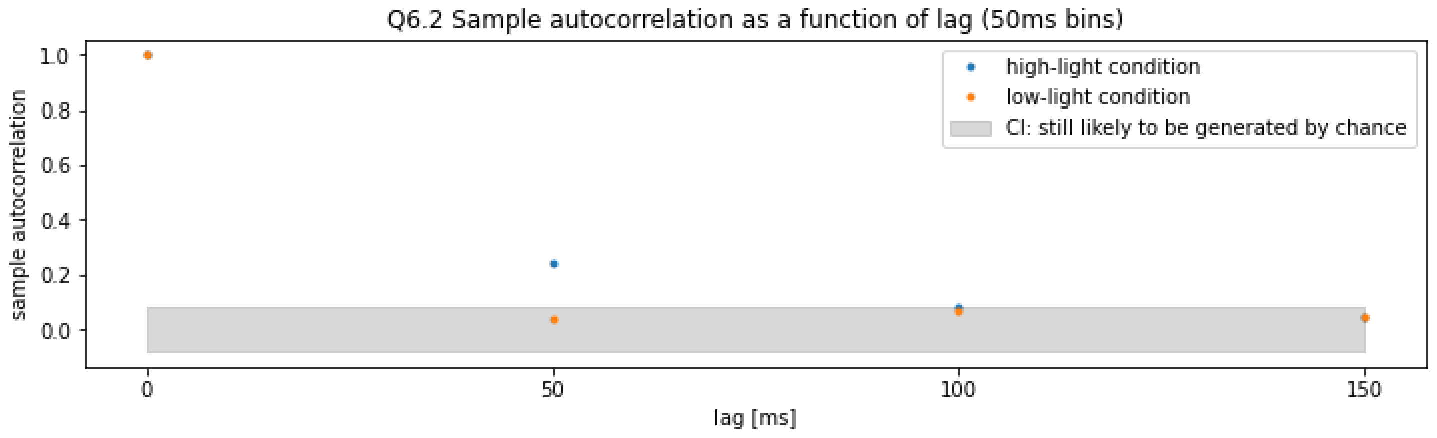
- Go back to Q6

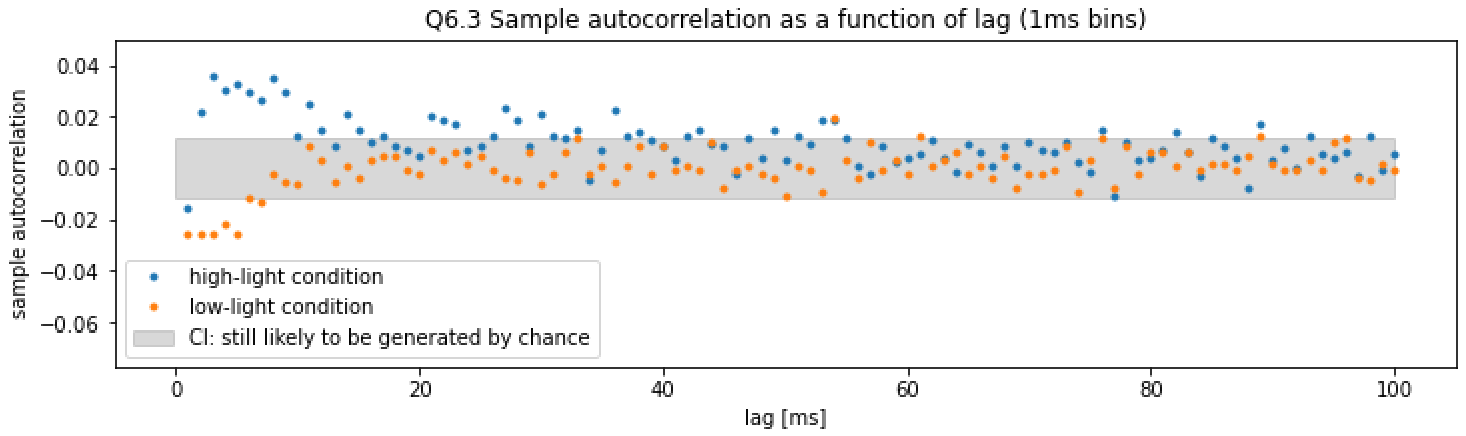
In [9]:

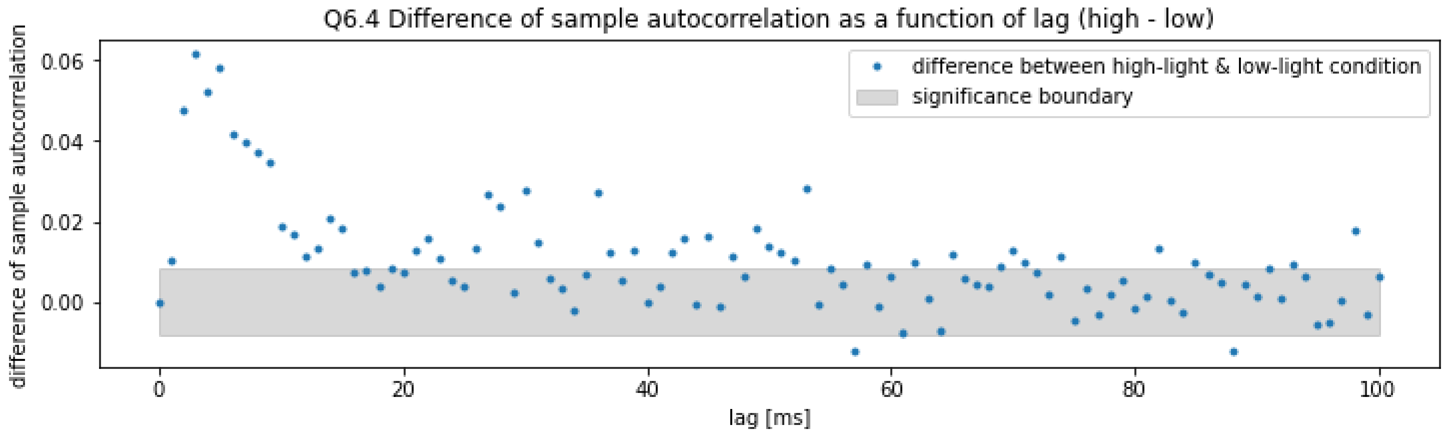
```python
# Hint: when using the np.correlate(,'full') function, the location of your first lag at lag 0 (t=0) is at ind

#use the following variables:
#ACF_low_50
#ACF_high_50
#ACF_low_1
#ACF_high_1

#######################
##   Q6.1 solution   ##
#######################

def autocorr(spike_data, binsize=0.050, lags=3):
    # Bin the spike data depending on binsize [s]
    T = 30 # [s]
    time_bin_edges = np.arange(0, T+binsize, binsize)
    increments, _ = np.histogram(spike_data, bins=time_bin_edges)

    # Calculate autocorrelation for defined lags
    mean = np.mean(increments)
    var = np.var(increments)
    ndata = increments - mean
    x_corr = np.correlate(ndata, ndata, 'full')
    x_corr = x_corr[x_corr.size//2:]
    x_corr = x_corr / var / len(ndata)

    # Calculate CI boundary
    N = len(increments)
    CI_bound = 2/np.sqrt(N)

    # Return only requested lags
    return x_corr[:lags+1], CI_bound, N


ACF_high_50, CI_bound_50, _ = autocorr(spikes_high)
ACF_low_50, _, _ = autocorr(spikes_low)

print(f"""
Q6.1
----
ACF_high_50: {np.round(ACF_high_50, 4)}
ACF_low_50:  {np.round(ACF_low_50, 4)}
CI boundary: {CI_bound_50:.4f}
""")

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

fig, ax = plt.subplots(1, 1)
x = np.arange(0, len(ACF_high_50))*50
ax.plot(x, ACF_high_50, '.', label='high-light condition')
ax.plot(x, ACF_low_50, '.', label='low-light condition')
ax.fill_between(x, -CI_bound_50, CI_bound_50, color='grey', alpha=0.3,
                label='CI: still likely to be generated by chance')
ax.set_title('Q6.2 Sample autocorrelation as a function of lag (50ms bins)')
ax.set_xlabel('lag [ms]')
ax.set_ylabel('sample autocorrelation')
ax.set_xticks(x)
ax.legend()
plt.show()

#######################
##   Q6.3 solution   ##
#######################

ACF_high_1, CI_bound_1,  N_high = autocorr(spikes_high, binsize=0.001, lags=100)
ACF_low_1, _, N_low = autocorr(spikes_low, binsize=0.001, lags=100)

fig, ax = plt.subplots(1, 1)
x = np.arange(0, len(ACF_high_1))*1
ax.plot(x, ACF_high_1, '.', label='high-light condition')
ax.plot(x, ACF_low_1, '.', label='low-light condition')
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 [ms]')
ax.set_ylabel('sample autocorrelation')
ax.legend()
ax.set_ybound(upper=0.05)
plt.show()

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

```python
##   Q6.4 solution   ##
#######################

diff = ACF_high_1-ACF_low_1
sign_bound = (1/N_high + 1/N_low)**0.5

fig, ax = plt.subplots(1, 1)
x = np.arange(0, len(ACF_high_1))*1
ax.plot(x, diff, '.', label='difference between high-light & low-light condition')
ax.fill_between(x, -sign_bound, sign_bound, color='grey', alpha=0.3,
                label='significance boundary')
ax.set_title('Q6.4 Difference of sample autocorrelation as a function of lag (high - low)')
ax.set_xlabel('lag [ms]')
ax.set_ylabel('difference of sample autocorrelation')
ax.legend()
plt.show()

```

```
Q6.1
----
ACF_high_50: [1.     0.2425 0.082  0.0464]
ACF_low_50:  [1.     0.0386 0.0701 0.0425]
CI boundary: 0.0816
```







**A6.1 answer**

The temporal structure of the spike train can be characterized using several measures, e.g. with ISI or with an autocorrelation coefficient.

The autocorrelation coefficient at *lag 0* is 1, which was expected from the mathematical formula for the sample autocorrelation (as the numerator and the denominator cancel out). The other correlation values (at the other considered lags) describe the probability that another spike is generated during the defined lag window after the first spike.

**A6.2 answer**

The figure (50ms bins) indicates that for the high-light condition we see significant values for the autocorrelation values up to 50ms lag. For the low-light condition, there are significant autocorrelation values to be observed up to around 7ms lag.

**A6.3 answer**

The figure (1ms bins) indicates that for the high-light condition we see significant values for the autocorrelation values up to around 54ms lag. The significant (positive) autocorrelation values have a slightly decreasing trend for increasing lags. The low-light condition shows significant (negative) autocorrelation values up to around 7ms lag.

The significant positive autocorrelation values indicate that there is a significant probability that another spike is generated within the 54ms lag window after a first spike. Contrarily, the significant negative autocorrelation values indicate that within the 7ms after a first spike, it is less likely to observe another spike.

**A6.4 answer**

For low lag values (up to 15ms), we observe a significant autocorrelation difference between the high-light and low-light conditions: another spike in this time window is more likely to be generated in the high-light condition than in the low-light condition. Up to 55ms lag, significant autocorrelation differences can still be observed, but with a smaller effect size and with increasing lag, more insignifcant difference values occur.

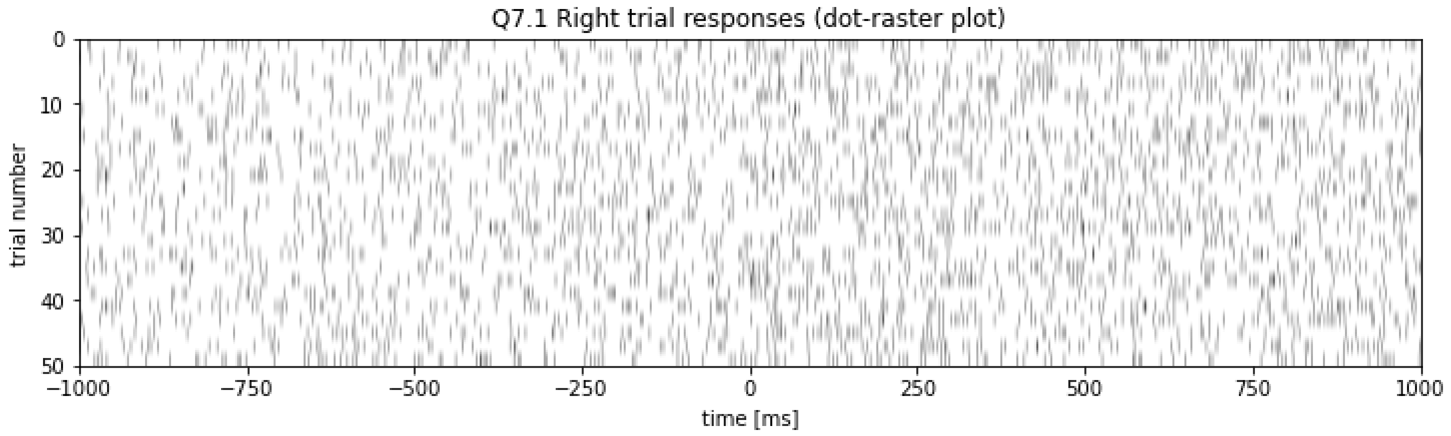
Lags higher than 55ms, result in almost no significant autocorrelation differences.

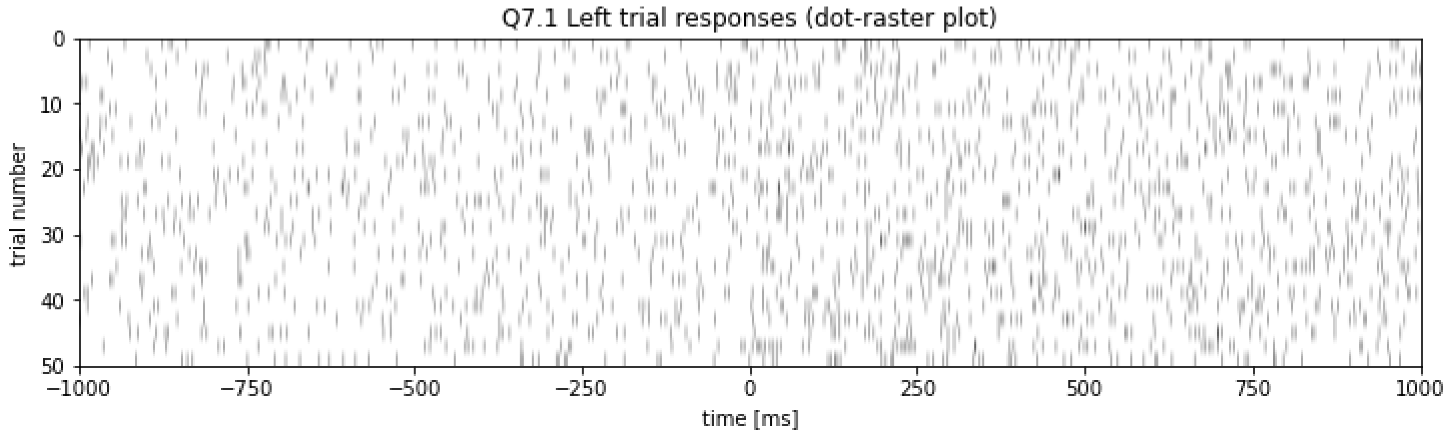
### A7: Peri-stimulus-time histogram (PSTH)

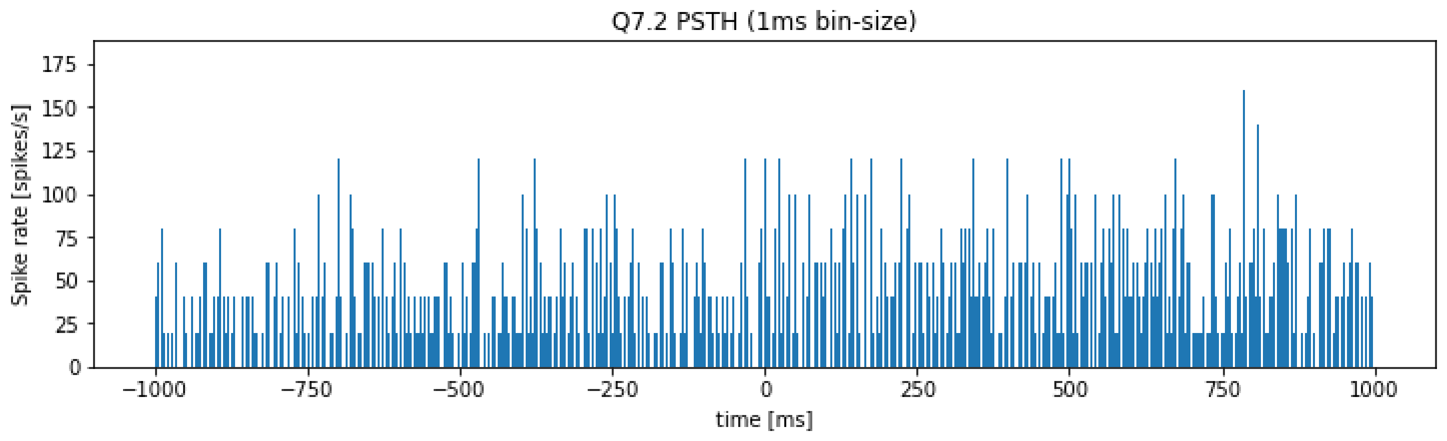
- Go back to Q7

In [10]:
```
########################
##   Q7.1 solution   ##
########################

# The direction vector should contain 0 for a left trial and 1 for a right trial.
# The sum function provides an efficient method to do so: sum(direction) which returns a value of 25.
# We conclude that half of the 50 trials involve right movement, and half involve left movement.

idx_R = np.where(direction)[0]
idx_L = np.where(direction==0)[0]
train_R = train[idx_R,:]
train_L = train[idx_L,:]

plt.imshow(train_L, aspect='auto', cmap='gray_r', extent = (-1000,1000,50,0))
plt.title('Q7.1 Right trial responses (dot-raster plot)')
plt.ylabel('trial number')
plt.xlabel('time [ms]')
plt.show()

plt.imshow(train_R, aspect='auto', cmap='gray_r', extent = (-1000,1000,50,0))
plt.title('Q7.1 Left trial responses (dot-raster plot)')
plt.ylabel('trial number')
plt.xlabel('time [ms]')
plt.show()

########################
##   Q7.2 solution   ##
########################

# Calculate PSTH (1ms bins)
PSTH_1 = np.mean(train, axis=0)/1e-3    # [spike rate per second per bin]

fig, ax = plt.subplots(1, 1)
ax.bar(t, PSTH_1)
ax.set_title('Q7.2 PSTH (1ms bin-size)')
ax.set_ylabel('Spike rate [spikes/s]')
ax.set_xlabel('time [ms]')
plt.show()

# Calculate PSTH (10ms bins)
time_end, binsize, N = 1000, 10, len(t) # [ms], [ms], [-]
```

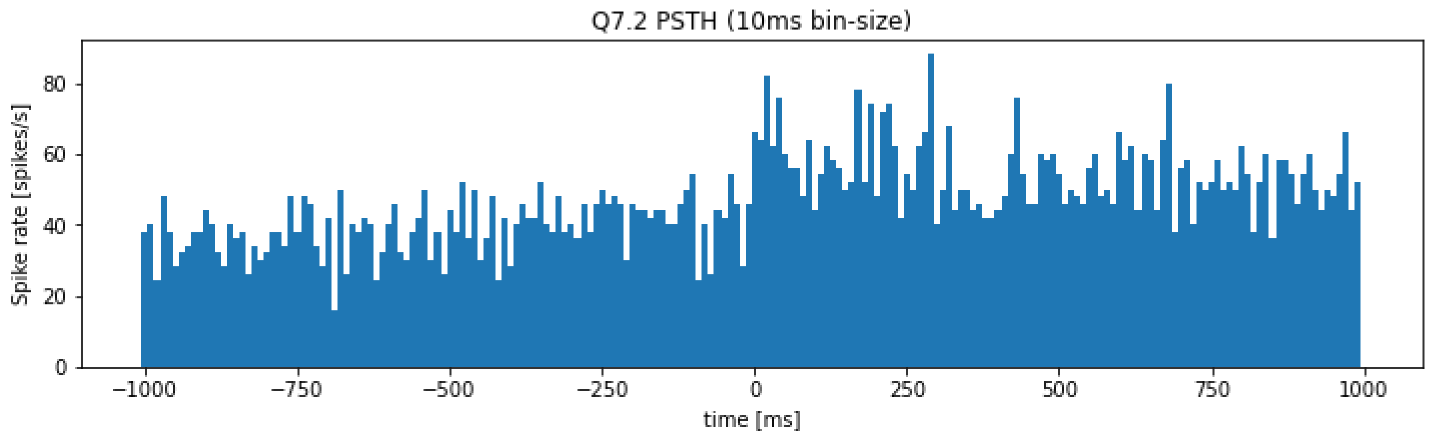
```
time_bins_10 = np.arange(-time_end, time_end, binsize)
PSTH_10 = np.mean(train, axis=0).reshape((N//binsize, binsize)).mean(axis=1)*1e3    # [spike rate per second pe

fig, ax = plt.subplots(1, 1)
ax.bar(time_bins_10, PSTH_10, width=10)
ax.set_title('Q7.2 PSTH (10ms bin-size)')
ax.set_ylabel('Spike rate [spikes/s]')
ax.set_xlabel('time [ms]')
plt.show()
```







Q7.2 PSTH (1ms bin-size)
Spike rate [spikes/s]
175
150
125
100
75
50
25
0
−1000
−750
−500
−250
0
250
500
750
1000
time [ms]

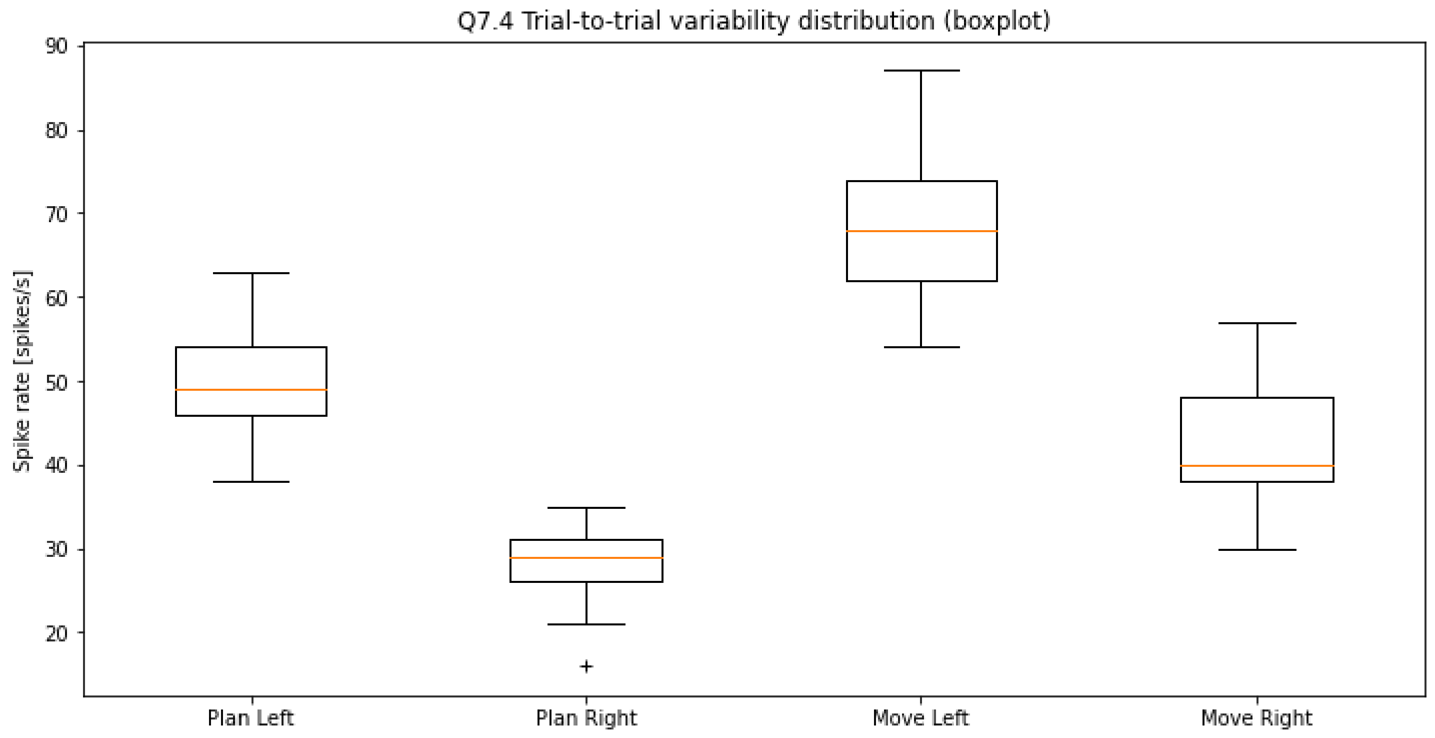


Q7.2 PSTH (10ms bin-size)
Spike rate [spikes/s]
80
60
40
20
0
−1000
−750
−500
−250
0
250
500
750
1000
time [ms]

In [11]:
```python
########################
##   Q7.3 solution   ##
########################

spiking_rate_plan = np.mean(train[:,:N//2])*1e3 # [spikes/s]
spiking_rate_move = np.mean(train[:,N//2:])*1e3

spiking_rate_plan_R = np.mean(train_R[:,:N//2])*1e3
spiking_rate_move_R = np.mean(train_R[:,N//2:])*1e3

spiking_rate_plan_L = np.mean(train_L[:,:N//2])*1e3
spiking_rate_move_L = np.mean(train_L[:,N//2:])*1e3

print(f"""Q7.3
-------
spiking_rate_plan: {spiking_rate_plan:.2f} spikes/s
spiking_rate_move: {spiking_rate_move:.2f} spikes/s

spiking_rate_plan_R: {spiking_rate_plan_R:.2f} spikes/s
spiking_rate_move_R: {spiking_rate_move_R:.2f} spikes/s
spiking_rate_avg_R:  {np.round((spiking_rate_plan_R + spiking_rate_move_R)/2, 2)} spikes/s

spiking_rate_plan_L: {spiking_rate_plan_L:.2f} spikes/s
spiking_rate_move_L: {spiking_rate_move_L:.2f} spikes/s
spiking_rate_avg_L:  {np.round((spiking_rate_plan_L +  spiking_rate_move_L)/2, 2)} spikes/s
""")

########################
##   Q7.4 solution   ##
########################

PlanL = sum(train_L[:,:N//2], axis = 1)   # Firing rate L, planning.
PlanR = sum(train_R[:,:N//2], axis = 1)   # Firing rate R, planning.
MoveL = sum(train_L[:,N//2:], axis = 1)   # Firing rate L, movement.
MoveR = sum(train_R[:,N//2:], axis = 1)   # Firing rate R, movement.

fig, ax = plt.subplots(1, 1, figsize=(12, 6))
ax.boxplot([PlanL, PlanR, MoveL, MoveR], sym='+',
        labels = ['Plan Left', 'Plan Right', 'Move Left', 'Move Right'])
ax.set_title('Q7.4 Trial-to-trial variability distribution (boxplot)')
ax.set_ylabel('Spike rate [spikes/s]')
```

```
plt.show()
```

```
Q7.3
-------
spiking_rate_plan: 38.96 spikes/s
spiking_rate_move: 54.96 spikes/s

spiking_rate_plan_R: 28.24 spikes/s
spiking_rate_move_R: 42.28 spikes/s
spiking_rate_avg_R:  35.26 spikes/s

spiking_rate_plan_L: 49.68 spikes/s
spiking_rate_move_L: 67.64 spikes/s
spiking_rate_avg_L:  58.66 spikes/s
```



**A7.1 conclusion**

Upon analyzing the separate raster plots, it becomes apparent that there is a marginally higher level of spiking activity during left trials compared to right trials, both before (i.e. the planning phase) and after (i.e. the movement phase) the 0-second mark. However, no conclusive evidence of a rhythmic spiking pattern can be deduced from the raster plot. While there are certain areas where one might argue for the presence of rhythmic spiking, it remains uncertain as to whether this pattern is genuine or simply a random occurrence.

**A7.2 conclusion**

The choice of bin size can have a significant impact on the interpretation of the results. The initial PSTH with very small bin widths suggested a higher firing rate during both the planning and movement phases than the firing rate found with larger bin sizes. This is because the small bin size (1ms) made it difficult to discern bins with low PSTH values, which were masked by neighboring bins with higher PSTH values. The later analysis with 10ms bin size revealed a more accurate average firing rate of 35 spikes/s during planning and 55 spikes/s during movement.

A good bin size would depend on the nature of the data being analyzed and the research question being addressed. In general, larger bin sizes may be preferable for data with lower spike rates or when the aim is to observe broad trends or patterns over time. Smaller bin sizes may be more appropriate when analyzing data with high spike rates or when the aim is to observe fine-scale temporal dynamics. However, it is important to be cautious when selecting bin sizes and to check the results obtained with different bin sizes to ensure that they are reliable and robust.

**A7.3 conclusion**

As printed in the code cell above, a spiking rate of 38.96 spikes/s and 54.96 spikes/s are found for the planning and movement phase respectively. This result is similar to the estimation of the average firing rates, obtained from the histogram with 10ms bin size.

By breaking down the rates into the left and right movement tasks, one can find a significant difference: 35.25 spikes/s versus 58.66 spikes/s for the right and left spiking rates respectively. One can conclude that spiking rates for planning and movement in left trials are higher than the rates for right trials.

### A7.4 conclusion

The S.I. unit for spikes/second is Hertz (Hz). From the boxplots above, one concludes that the firing rate is significantly higher in left trials and in the movement phase of each trial.

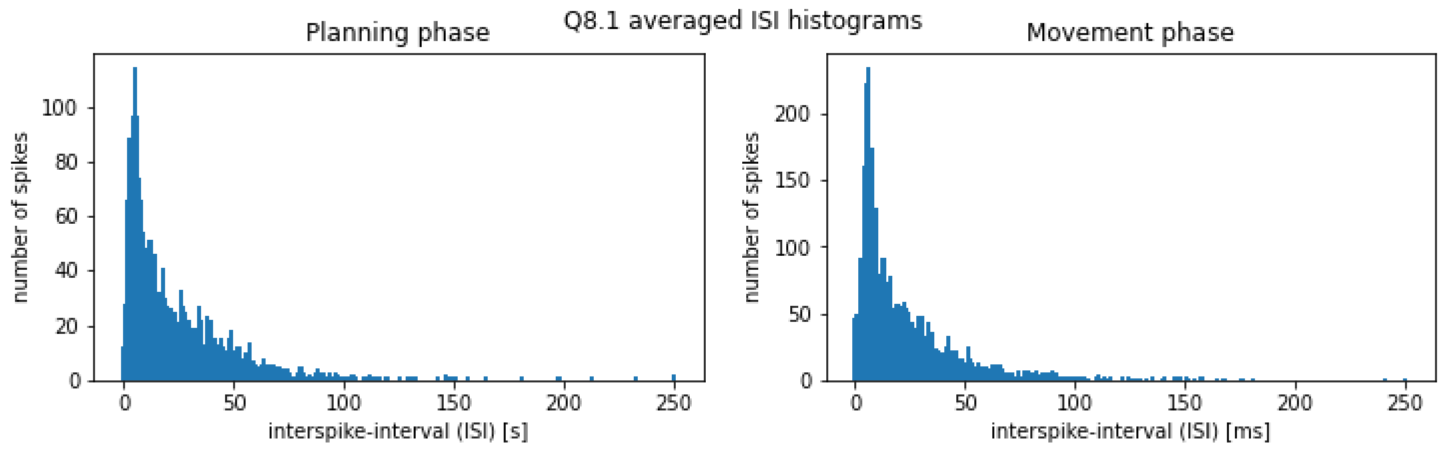
## A8: Rhythmic structure

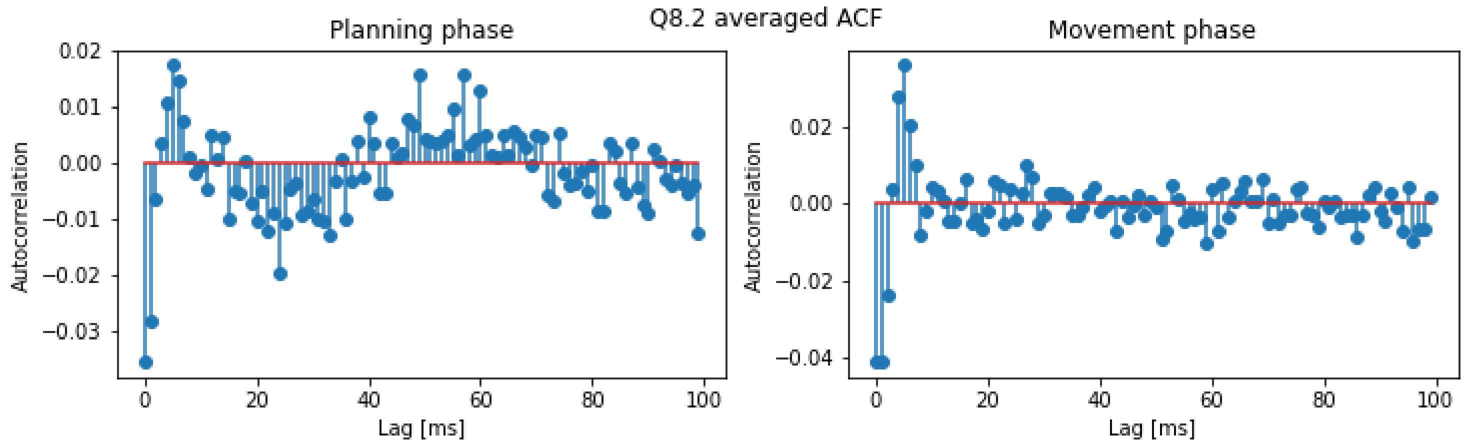
- Go back to Q8

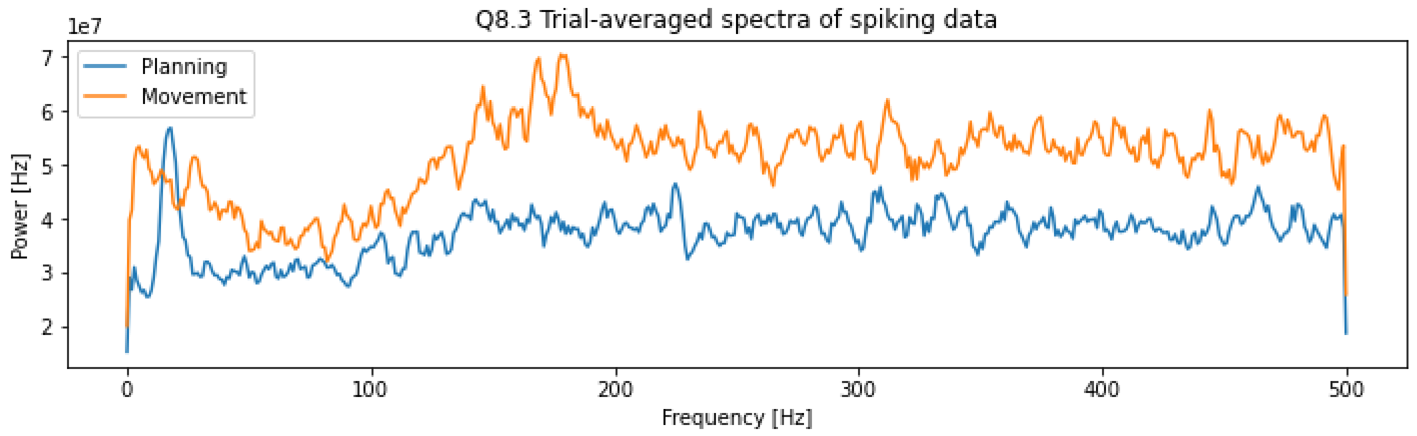
In [12]:
```python
# Q8.1:use the following variables:
#plan_train
#move_train
#plan_average_ISI_hist  ##the average ISI histogram accross all trials for the planning activity, bins of 1ms
#move_average_ISI_hist  ##the average ISI histogram accross all trials for the moving activity, bins of 1 ms

# Q8.2:use the following variables:
#average_ACF_plan
#average_ACF_move

# Note for Q8.2:
# The 'full' correlation function repeats equal values twice,
# Because going back in time is the same ACF value as going further in time.
# Start your correlate array halfway of the values returned by np.correlate('full').

# Note for Q8.3:
# Power spectral density (psd)
# Below you find the package input needs.
# nitime.algorithms.spectral.multi_taper_psd(s, Fs=6.283185307179586, NW=None, BW=None, adaptive=False, jackkn
# Fs is the sampling rate per second, here 1000
# s is sample, so the plan or move train
# NW stands for the time - bandwidth. This gives 2*NW-1 (7)tapers
# to have a better understanding of the multitaper method, you can watch the following video:
# https://www.youtube.com/watch?v=OBwnmiVT9TE

# Try to create the mt_specpb function with the help of:
# https://mark-kramer.github.io/Case-Studies-Python/10.html

# Note for  Q8.4:
# The plt.contour function needs the following input.
# plt.contourf([X, Y,] Z, [levels], **kwargs) #Z is optional

#######################
##   Q8.1 solution   ##
#######################

# Planning period
i_plan = where(t < 0)[0]
i_move = where(t >= 0)[0]
spiketrialsPlan, plan_train = where(train[:,i_plan] > 0)
plan_average_ISI = np.diff(plan_train)
plan_average_ISI = plan_average_ISI[where(plan_average_ISI > 0)]
plan_average_ISI_hist = histogram(plan_average_ISI, 250)[0]

plt.subplot(121)
plt.bar(linspace(0, 250, 250), plan_average_ISI_hist, width = 2)
plt.xlabel('interspike-interval (ISI) [s]')
plt.ylabel('number of spikes')
plt.title('Planning phase')

# Movement period
spiketrialsMove, move_train = where(train[:, i_move] > 0)
move_average_ISI = np.diff(move_train)
move_average_ISI = move_average_ISI[where(move_average_ISI>0)]
move_average_ISI_hist = histogram(move_average_ISI, 250)[0]

plt.subplot(122)
plt.bar(linspace(0, 250, 250), move_average_ISI_hist, width = 2)
plt.xlabel('interspike-interval (ISI) [ms]')
plt.ylabel('number of spikes')
plt.title('Movement phase')
plt.suptitle('Q8.1 averaged ISI histograms')
plt.show()

#######################
##   Q8.2 solution   ##
#######################

def ACFloop():
    # Initialize ACF arrays
    average_ACF_plan = zeros((50, 1999))
    average_ACF_move = zeros((50, 1999))

    # Loop over every trial
    for k in range(50):
        plan = train[k, i_plan]
        move = train[k, i_move]

        # Determine ACF
        corr1 = correlate(plan - mean(plan), plan - mean(plan), 'full')
        average_ACF_plan[k] = corr1 / linalg.norm(plan - mean(plan))**2
        corr2 = correlate(move - mean(move), move - mean(move), 'full')
        average_ACF_move[k] = corr2 / linalg.norm(move - mean(move))**2
```

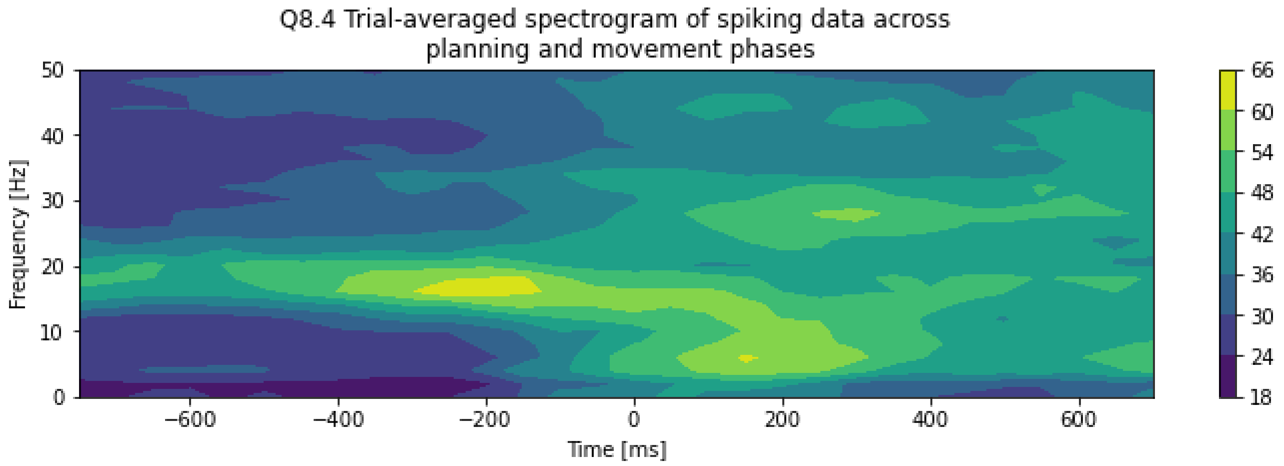
```python

    return average_ACF_plan, average_ACF_move

average_ACF_plan, average_ACF_move = ACFloop()

# Construct ACF plot
plt.figure(figsize=(12, 3))
plt.subplot(121)
plt.stem(mean(average_ACF_plan[:,1000:1100], axis = 0))
plt.xlabel('Lag [ms]')
plt.ylabel('Autocorrelation')
plt.title('Planning phase')

plt.subplot(122)
plt.stem(mean(average_ACF_move[:,1000:1100], axis = 0))
plt.xlabel('Lag [ms]')
plt.ylabel('Autocorrelation')
plt.title('Movement phase')
plt.suptitle('Q8.2 averaged ACF')
plt.show()

#######################
##   Q8.3 solution   ##
#######################

Fs = 1000
f, SPlan, var = multi_taper_psd(train[:, i_plan] * Fs, Fs=1000, NW=3)
f, SMove, var = multi_taper_psd(train[:, i_move] * Fs, Fs=1000, NW=3)
SPlan = np.mean(SPlan,axis=0) * Fs ** 2 / 2
SMove = np.mean(SMove, axis=0) * Fs ** 2 / 2

plt.plot(f, SPlan, label="Planning")
plt.plot(f, SMove, label="Movement")
plt.xlabel('Frequency [Hz]')
plt.ylabel('Power [Hz]')
plt.legend()
plt.title('Q8.3 Trial-averaged spectra of spiking data')
plt.show()

#######################
##   Q8.4 solution   ##
#######################
```

```python

def mt_specpb(data, Fs=1000, NW=3, trial_ave=True):
    # Compute & scale tapers
    tapers, _ = dpss_windows(data.shape[-1], NW, 2*NW-1)
    tapers *= sqrt(Fs)

    dataT = [[trial * t for t in tapers] for trial in data]
    T = rfft(tapers)
    J = rfft(dataT)

    # Compute spectrum
    J -= [T * trial.mean() for trial in data]
    J *= J.conj()
    S = J.mean(1).real
    f = rfftfreq(data.shape[-1], 1 / Fs)
    if trial_ave: S = S.mean(0)

    return f, S


window, step = .5, .05
fpass = [0, 50]
Fs = 1000

window, step = [int(Fs * x) for x in [window, step]]
starts = range(0, train.shape[-1] - window, step)
# Get frequencies
f = mt_specpb(train[:, range(window)], NW=2)[0]
findx = (f >= fpass[0]) & (f <= fpass[1])
f = f[findx]
# Compute and center the spectrum on each 500 ms window.
spectrogram = [mt_specpb(train[:, range(s, s + window)], NW=2)[1][findx] for s in starts]
T = t[starts] + window / 2

# Construct plot
plt.contourf(T, f, array(spectrogram).T)
plt.xlabel('Time [ms]')
plt.ylabel('Frequency [Hz]')
plt.title('Q8.4 Trial-averaged spectrogram of spiking data across\n planning and movement phases')
cbar = plt.colorbar()
plt.show()

```


Q8.1 averaged ISI histograms
Planning phase
Movement phase
number of spikes
interspike-interval (ISI) [s]
interspike-interval (ISI) [ms]
0
20
40
60
80
100
150
200
250
50


Q8.2 averaged ACF
Planning phase
Movement phase
Autocorrelation
Lag [ms]
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
0
20
40
60
80
100





**A8.1 conclusion**

During the movement period, one could observe a greater total number of inter-spike intervals (ISIs), which supports the previous findings of a higher firing rate during this period. Additionally, there is a more distinct peak in the ISI distribution around 6 ms during movement, indicating a greater inclination for burst firing. There are some dissimilarities in the ISI distribution's tail structure for ISIs over 20 ms, but it was challenging to link this to variations in rhythmic spiking structure.

**A8.2 conclusion**

In both periods the figures show a negative correlation for lags of 1–3 ms, followed by positive correlation at lags of 1–8 ms, with a peak at a lag of 6 ms. During the planning period, the figures show a clear structure continuing at higher lags, with an extended period of negative correlations at lags of about 15–35 ms and positive correlations at lags of about 50–70 ms. For the movement period, this structure at lags longer than about 10 ms seems reduced or even absent. These results suggest a change in the rhythmic firing properties between the planning and movement periods. The second peak in the autocorrelation plot for the planning period around 60 ms suggests a 1/60 ms (17 Hz) rhythm that disappears during movement

### A8.3 conclusion

Several distinct features can be observed in the spectral estimates for the planning and movement periods. The spectral density for the planning period asymptotes to approximately 35 Hz at higher frequencies, while for the movement period, it asymptotes closer to 55 Hz. These results support previous findings based on the observed mean firing rates. A decrease in spectral estimates can be observed at frequencies below 200 Hz, indicating an approximate refractory period of 5 ms. However, during the planning period, a significant peak can be observed at around 18 Hz, which is absent during the movement period. This suggests that STN neuron spikes occur rhythmically during the planning period and that this rhythm is attenuated during movement.

### A8.4 conclusion

One can observe several distinct features in the spectral estimates for the planning and movement periods. At higher frequencies, the spectral density for the planning period asymptotes to about 35 Hz, while for the movement period, it asymptotes closer to 55 Hz. This finding corroborates previous results based on the observed mean firing rates. Both spectral estimates exhibit a decrease at frequencies below 200 Hz, suggesting an approximately 5 ms refractory period. However, during the planning period, a large peak at about 18 Hz can be observed that does not appear during the movement period. This suggests that the STN neuron spikes rhythmically during the planning period and that this rhythm is attenuated during movement.