CodingLab4_leandro

May 16, 2023

Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2023

Name: FILL IN YOUR NAMES HERE

1 Coding Lab 4

If needed, download the data files `nds_cl_4_*.csv` from ILIAS and save it in the subfolder `../data/`. Use a subset of the data for testing and debugging, ideally focus on a single cell (e.g. cell number x). The spike times and stimulus conditions are read in as pandas data frames. You can solve the exercise by making heavy use of that, allowing for many quite compact computationis. If you need help on that, there is lots of documentation and several good tutorials are available online. Of course, converting the data into classical numpy arrays is also valid.

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.ticker as ticker
import numpy as np
import scipy.optimize as opt

from scipy import signal as signal

import itertools

%matplotlib inline

%load_ext jupyter_black

%load_ext watermark
%watermark --time --date --timezone --updated --python --iversions --watermark␣
 ↪-p sklearn
```

```
<IPython.core.display.HTML object>
```

```
Last updated: 2023-05-16 23:56:49CEST

Python implementation: CPython
Python version       : 3.10.0
IPython version      : 8.12.0

sklearn: 1.2.2

scipy     : 1.10.1
seaborn   : 0.12.2
numpy     : 1.24.2
pandas    : 2.0.0
matplotlib: 3.7.1

Watermark: 2.3.1
```

[ ]:
```
plt.style.use("../matplotlib_style.txt")
```

1.1 Load data

[ ]:
```
spikes = pd.read_csv("../data/nds_cl_4_spiketimes.csv")  # neuron id, spike time
stims = pd.read_csv("../data/nds_cl_4_stimulus.csv")  # stimulus onset in ms,␣
 ↪direction

stimDur = 2000.0  # in ms
nTrials = 11  # number of trials per condition
nDirs = 16  # number of conditions
deltaDir = 22.5  # difference between conditions

stims["StimOffset"] = stims["StimOnset"] + stimDur
```

We require some more information about the spikes for the plots and analyses we intend to make later. With a solution based on dataframes, it is natural to compute this information here and add it as additional columns to the **spikes** dataframe by combining it with the **stims** dataframe. We later need to know which condition (**Dir**) and trial (**Trial**) a spike was recorded in, the relative spike times compared to stimulus onset of the stimulus it was recorded in (**relTime**) and whether a spike was during the stimulation period (**stimPeriod**). But there are many options how to solve this exercise and you are free to choose any of them.

[ ]:
```
# you may add computations as specified above
spikes["Dir"] = np.nan
spikes["relTime"] = np.nan
spikes["Trial"] = np.nan
spikes["stimPeriod"] = np.nan

dirs = np.unique(stims["Dir"])
```

```
trialcounter = np.zeros_like(dirs)

for i, row in stims.iterrows():
    trialcounter[dirs == row["Dir"]] += 1

    i0 = spikes["SpikeTimes"] > row["StimOnset"]
    i1 = spikes["SpikeTimes"] < row["StimOffset"]

    select = i0.values & i1.values

    spikes.loc[select, "Dir"] = row["Dir"]
    spikes.loc[select, "Trial"] = trialcounter[dirs == row["Dir"]][0]
    spikes.loc[select, "relTime"] = spikes.loc[select, "SpikeTimes"] -␣
 ↪row["StimOnset"]
    spikes.loc[select, "stimPeriod"] = True

spikes = spikes.dropna()
```

[ ]:
```
spikes
```

[ ]:

	Neuron	SpikeTimes	Dir	relTime	Trial	stimPeriod
514	1	15739.000000	270.0	169.000000	1.0	True
515	1	15776.566667	270.0	206.566667	1.0	True
516	1	15808.466667	270.0	238.466667	1.0	True
517	1	15821.900000	270.0	251.900000	1.0	True
518	1	15842.966667	270.0	272.966667	1.0	True
…	…	…	…	…	…	…
223463	41	599045.166667	202.5	1868.166667	11.0	True
223464	41	599063.233333	202.5	1886.233333	11.0	True
223465	41	599068.166667	202.5	1891.166667	11.0	True
223466	41	599080.200000	202.5	1903.200000	11.0	True
223467	41	599144.366667	202.5	1967.366667	11.0	True

[129738 rows x 6 columns]

[ ]:
```
spikes["Dir"].unique()
```

[ ]:
```
array([270. ,  45. , 112.5, 225. , 180. , 157.5,  67.5, 202.5,   0. ,
       315. , 292.5, 337.5, 247.5,  90. ,  22.5, 135. ])
```

1.2 Task 1: Plot spike rasters

In a raster plot, each spike is shown by a small tick at the time it occurs relative to stimulus onset. Implement a function `plotRaster()` that plots the spikes of one cell as one trial per row, sorted by conditions (similar to what you saw in the lecture). Why are there no spikes in some conditions and many in others?

If you opt for a solution without a dataframe, you need to change the interface of the function.

Grading: 2 pts

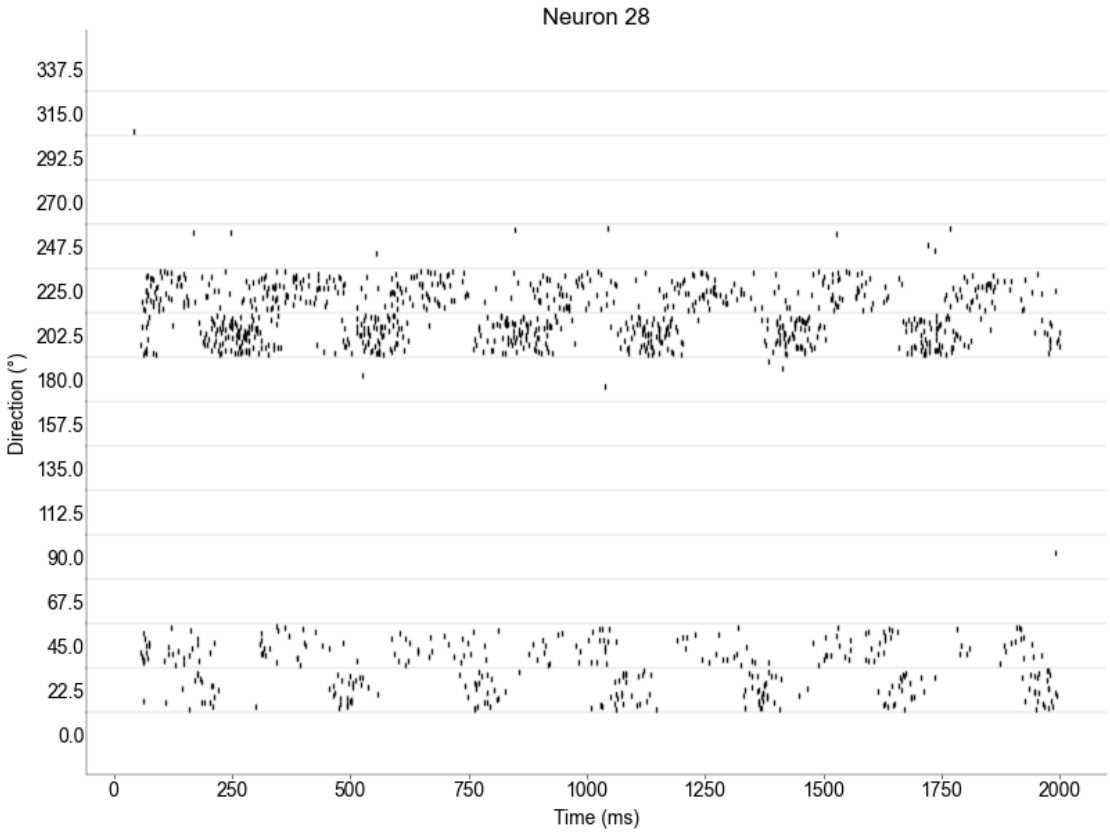
Answer: Some neurons only react to stimuly from certain directions. This shows in the rasterplots, when there are (almost) no spikes in a 'row'

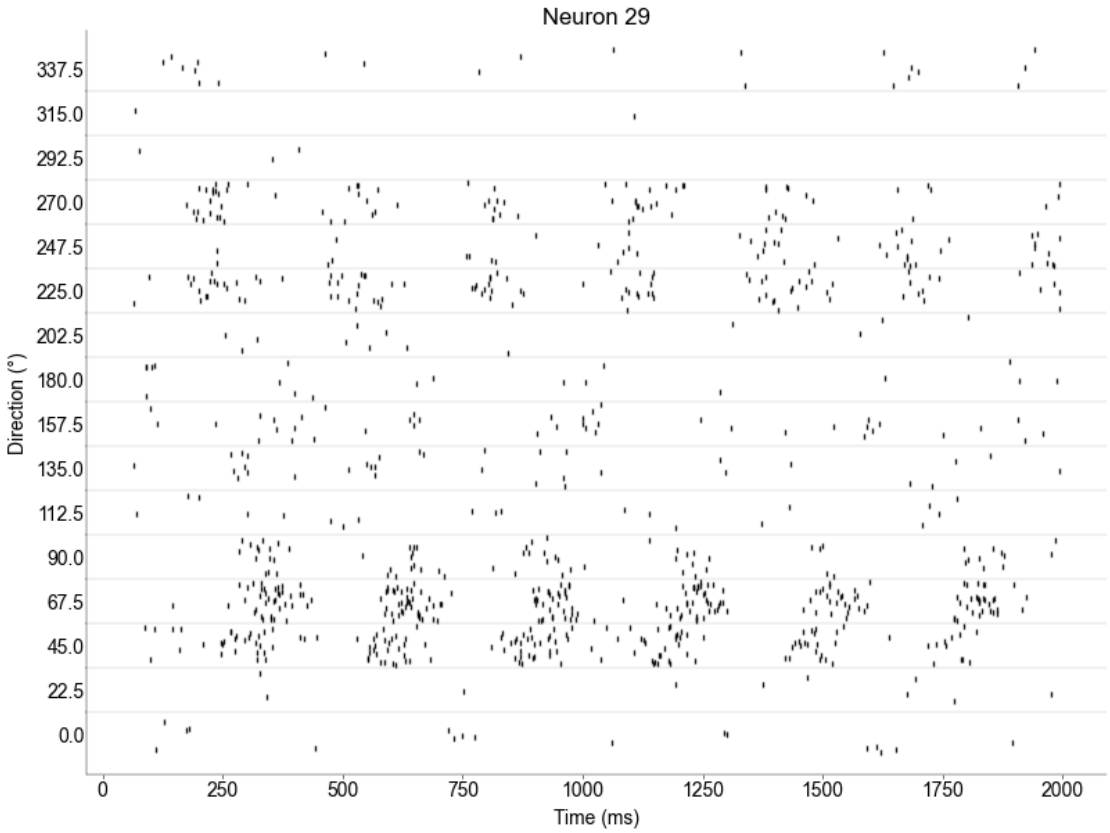
[ ]:
```python
def plotRaster(spikes: pd.DataFrame, neuron: int):
    """plot spike rasters for a single neuron sorted by condition

    Parameters
    ----------

    spikes: pd.DataFrame
        Pandas DataFrame with columns
            Neuron | SpikeTimes | Dir | relTime | Trial | stimPeriod

    neuron: int
        Neuron ID


    Note
    ----

    this function does not return anything, it just creates a plot!
    """

    fig, ax = plt.subplots(figsize=(8, 6))

    # ------------------------------------------------
    # Write a raster plot function for the data (2 pts)
    # ------------------------------------------------
    dirs = spikes["Dir"].unique()
    dirs.sort()
    neuron_spikes = spikes[spikes["Neuron"] == neuron]
    for dir in dirs:
        directional_data = neuron_spikes[neuron_spikes["Dir"] == dir]
        ax.scatter(
            directional_data["relTime"],
            directional_data["Dir"] + np.random.uniform(-10, 10,␣
↪len(directional_data)),
            marker="|",  # type: ignore
            label=dir,
            s=5,
            alpha=0.9,
            c="k",
        )

    ax.set_title(f"Neuron {neuron}")
    ax.set_xlabel("Time (ms)")
```

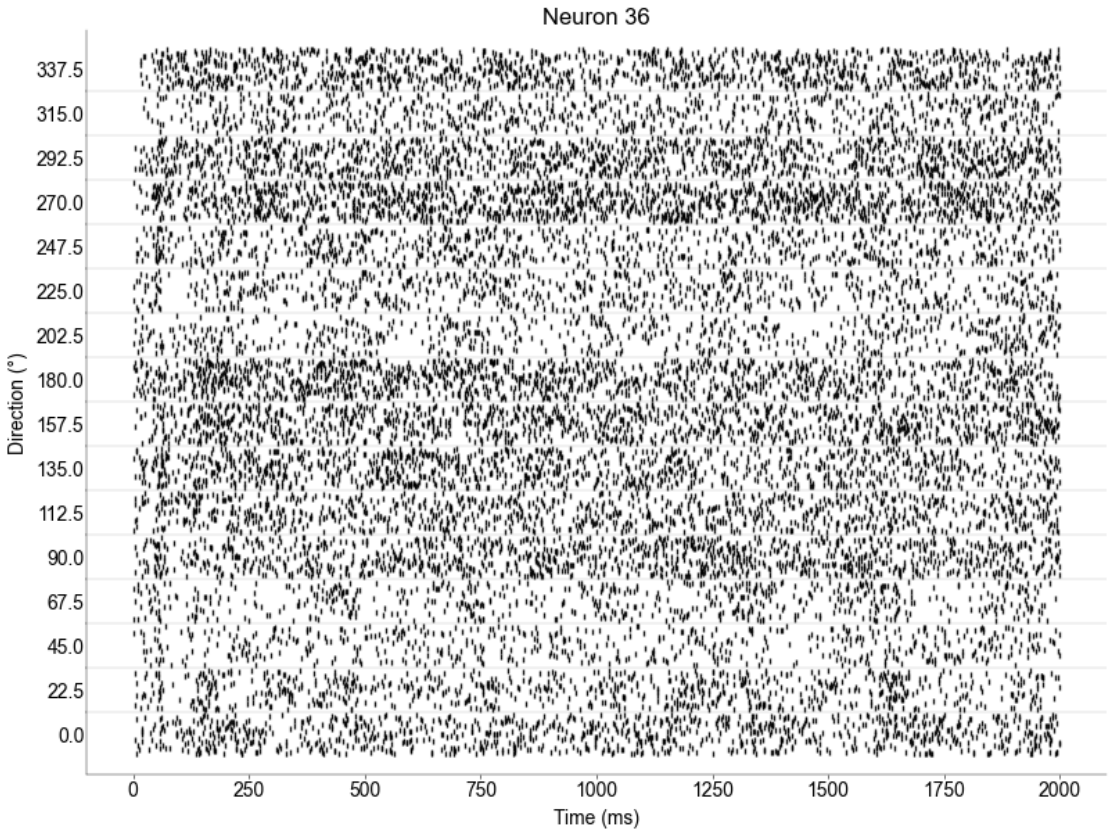
```
    ax.set_ylabel("Direction (°)")
    ax.set_ylim(np.min(dirs) - 20, np.max(dirs) + 20)
    ax.tick_params(axis="y", which="major", length=0)

    ax.set_yticks(dirs, minor=False)
    ax.set_yticks(dirs[:-1] + np.diff(dirs) / 2, minor=True)
    ax.grid(
        which="minor", axis="y", color="gray", linestyle="-", linewidth=0.3,␣
↪alpha=0.5
    )
    # insert your code here
    # stim direction should be on the y-axis and time on the x-axis
    # you can use plt.scatter or plt.plot to plot the responses to each stim
```

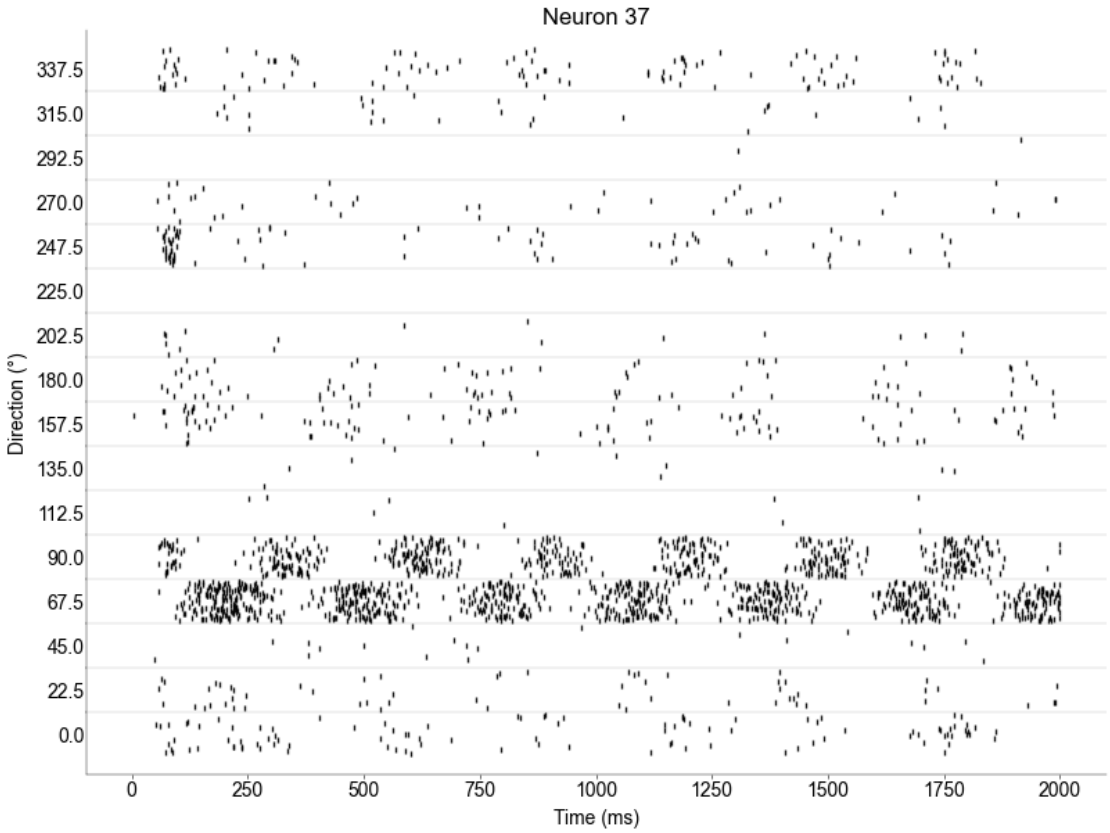
Show examples of different neurons. Good candidates to check are 28, 29, 36 or 37.

[ ]:
```
plotRaster(spikes, 28)
plotRaster(spikes, 29)
plotRaster(spikes, 36)
plotRaster(spikes, 37)
```




Neuron 29
337.5
315.0
292.5
270.0
247.5
225.0
202.5
180.0
157.5
135.0
112.5
90.0
67.5
45.0
22.5
0.0
Direction (°)
0
250
500
750
1000
1250
1500
1750
2000
Time (ms)


Neuron 36
337.5
315.0
292.5
270.0
247.5
225.0
202.5
180.0
157.5
135.0
112.5
90.0
67.5
45.0
22.5
0.0
Direction (°)
0
250
500
750
1000
1250
1500
1750
2000
Time (ms)



1.3 Task 2: Plot spike density functions

Compute an estimate of the spike rate against time relative to stimulus onset. There are two ways: * Discretize time: Decide on a bin size, count the spikes in each bin and average across trials. * Directly estimate the probability of spiking using a density estimator with specified kernel width.

Implement one of them in the function `plotPsth()`. If you use a dataframe you may need to change the interface of the function.

NOTE: we will be using method 1: Discretize time

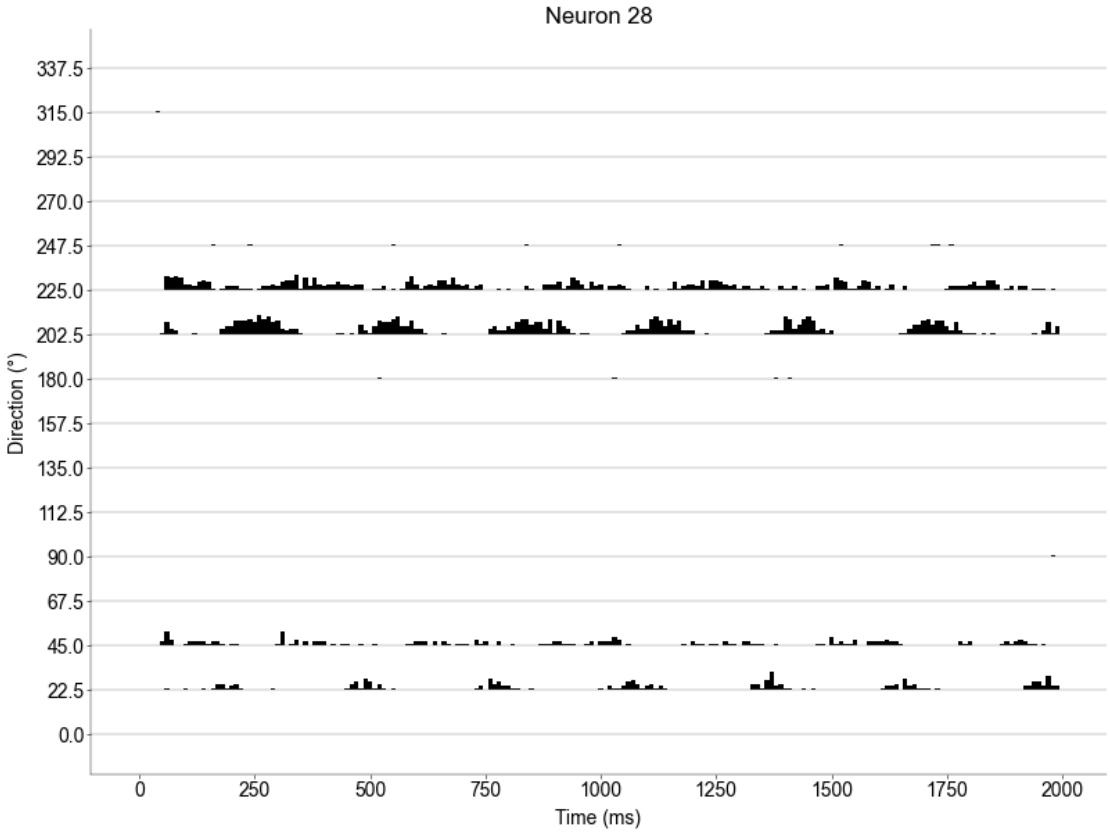
Grading: 2 pts

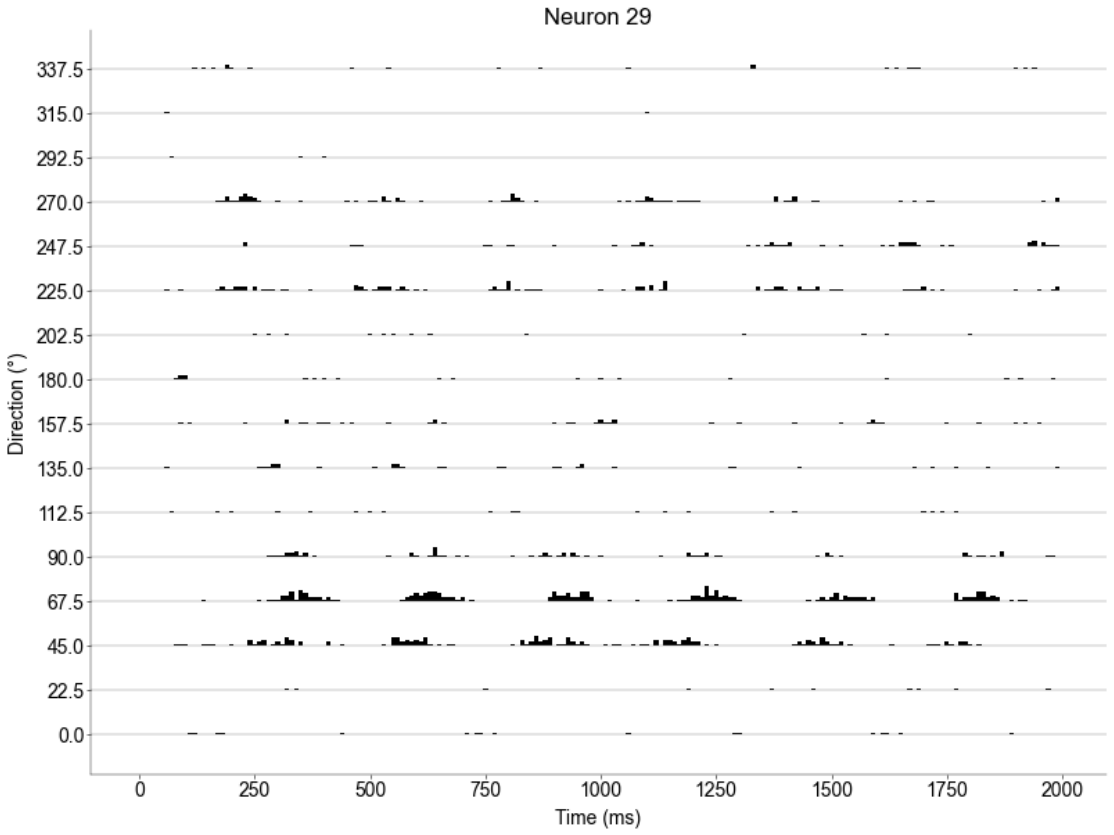
[ ]:
```
def plotPSTH(
    spikes: pd.DataFrame, neuron: int, binwidth: float = 10.0, plot_tpye="bar"
):
    """Plot PSTH for a single neuron sorted by condition

    Parameters
    ----------

    spikes: pd.DataFrame
```

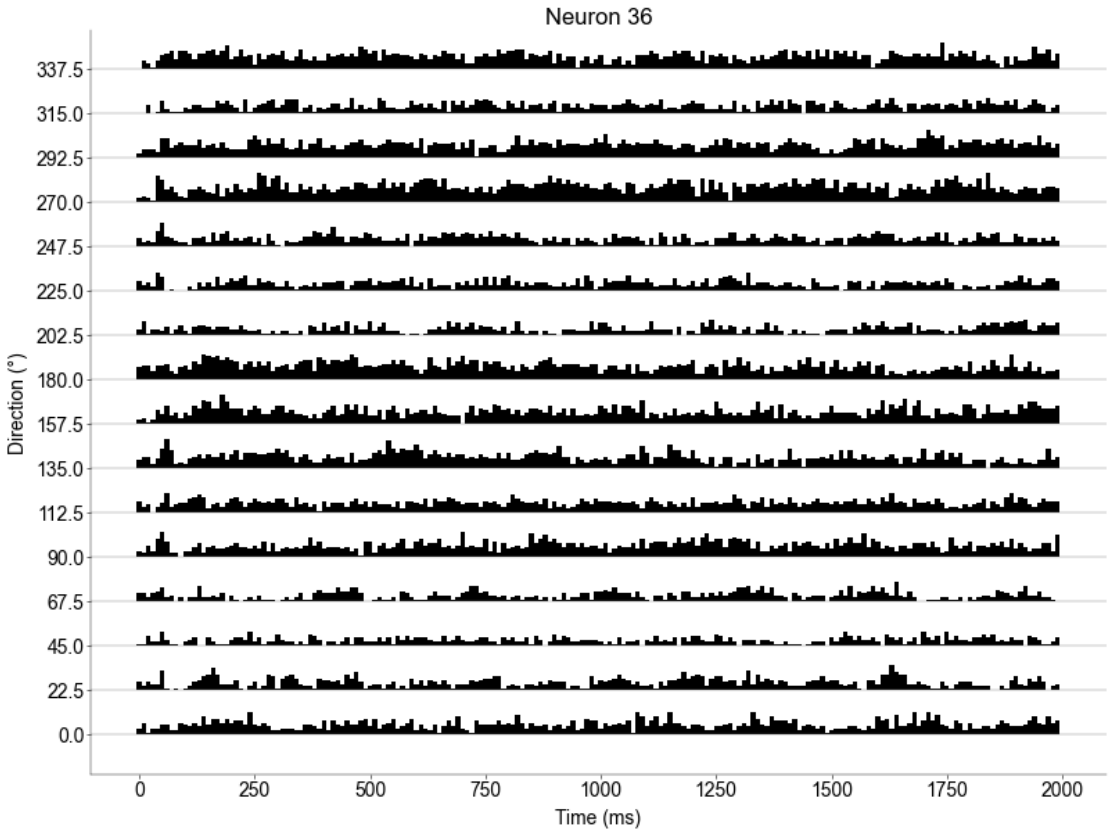
```
    Pandas DataFrame with columns
        Neuron | SpikeTimes | Dir | relTime | Trial | stimPeriod

neuron: int
    Neuron ID

binwidth: float
    width of the histogram bins in ms

Note
----

this function does not return anything, it just creates a plot!
"""

# insert your code here

# ------------------------------------------------
# Implement one of the spike rate estimates (1 pt)
# ------------------------------------------------

dirs = spikes["Dir"].unique()
dirs.sort()
neuron_spikes = spikes[spikes["Neuron"] == neuron]
bins = np.arange(0, spikes.relTime.max() + 1, binwidth)
psth = np.zeros((len(dirs), len(bins) - 1))
for i, dir in enumerate(dirs):
    directional_data = neuron_spikes[neuron_spikes["Dir"] == dir]
    psth[i], _ = np.histogram(directional_data["relTime"], bins=bins)

# ------------------------------------------------
# Plot the obtained spike rate estimates (1 pt)
# ------------------------------------------------
fig, ax = plt.subplots(figsize=(8, 6))

for i, dir in enumerate(dirs):
    if plot_tpye == "bar":
        ax.bar(
            bins[:-1],
            height=psth[i, :],
            width=binwidth,
            bottom=i * 22.5,
            color="black",
        )
    else:
        ax.plot(bins[:-1], psth[i, :] + i * 22.5, c="k", lw=1)
```

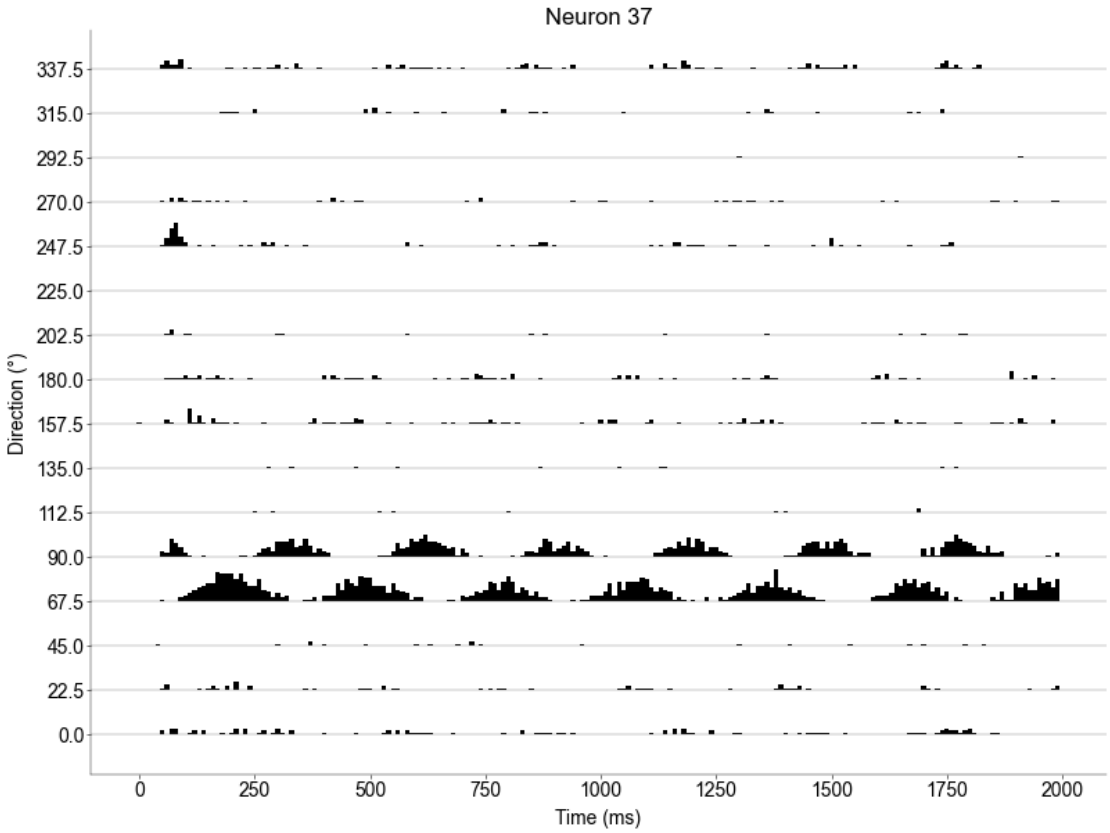
```
    ax.set_title(f"Neuron {neuron}")
    ax.set_xlabel("Time (ms)")
    ax.set_ylabel("Direction (°)")
    ax.set_ylim(np.min(dirs) - 20, np.max(dirs) + 20)
    ax.set_yticks(dirs, minor=False)

    ax.grid(
        which="major", axis="y", color="gray", linestyle="-", linewidth=0.3,␣
↪alpha=1
    )

    # plot should look similar to `plotRaster`
    # you can plot use plt.hist for each direction, but much cleaner
    # is to only plot bin centers vs bin heights using plt.plot
```

[ ]:
```
plotPSTH(spikes, 28)
plotPSTH(spikes, 29)
plotPSTH(spikes, 36)
plotPSTH(spikes, 37)
```



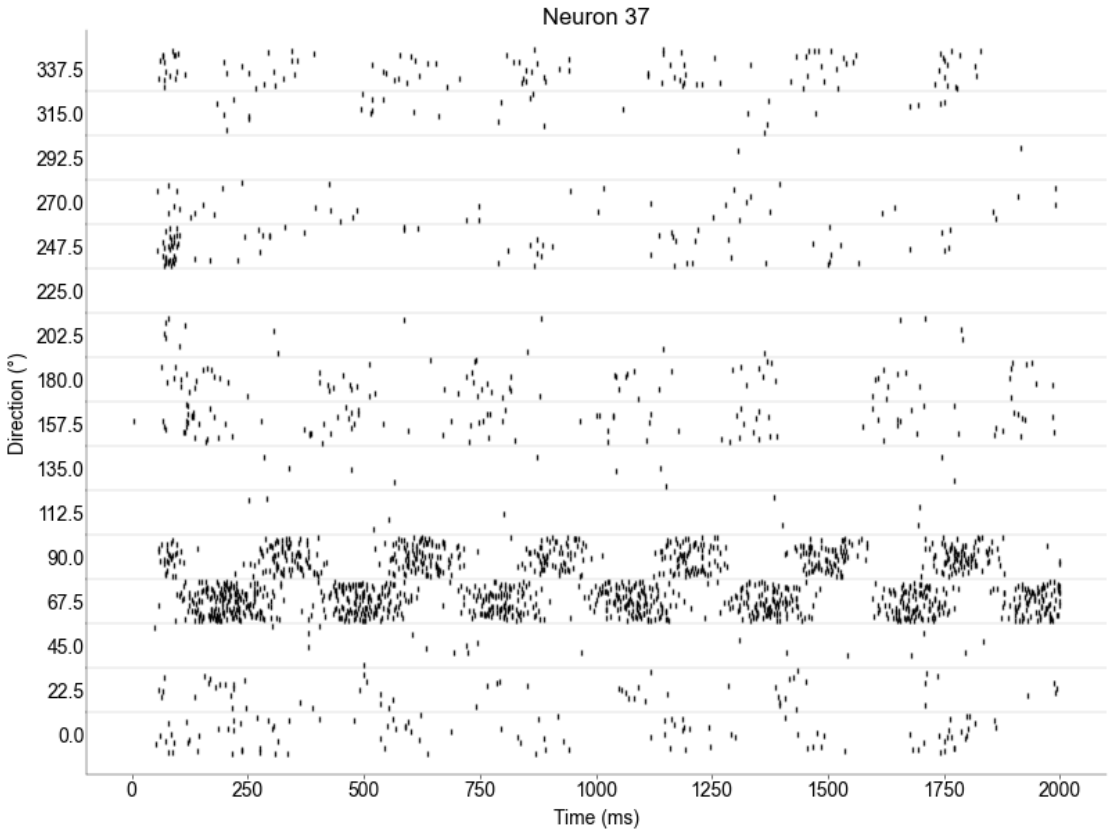

Neuron 29
337.5
315.0
292.5
270.0
247.5
225.0
202.5
180.0
157.5
135.0
112.5
90.0
67.5
45.0
22.5
0.0
Direction (°)
0
250
500
750
1000
1250
1500
1750
2000
Time (ms)

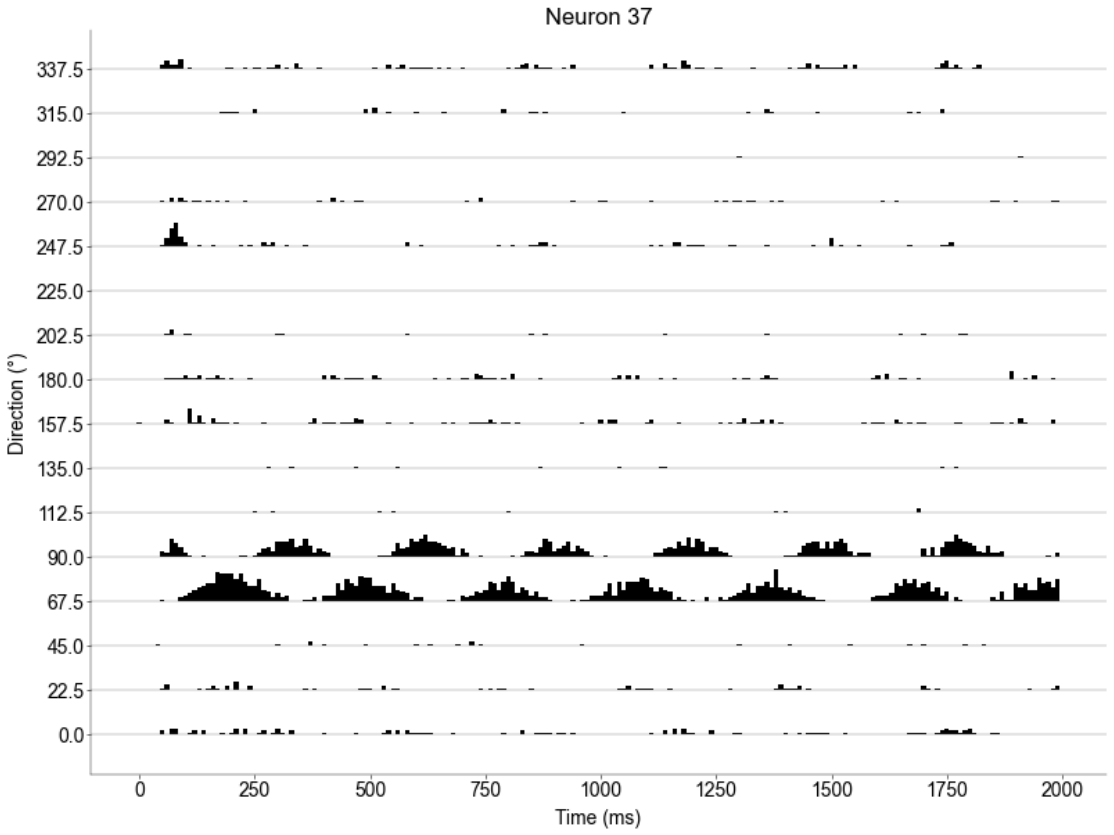

Neuron 36
337.5
315.0
292.5
270.0
247.5
225.0
202.5
180.0
157.5
135.0
112.5
90.0
67.5
45.0
22.5
0.0
Direction (°)
0
250
500
750
1000
1250
1500
1750
2000
Time (ms)



[ ]:
```
# plotPSTH(spikes, 28, plot_tpye="line")
# plotPSTH(spikes, 29, plot_tpye="line")
# plotPSTH(spikes, 36, plot_tpye="line")
# plotPSTH(spikes, 37, plot_tpye="line")
```

[ ]:
```
plotRaster(spikes, 37)
plotPSTH(spikes, 37)
```


Neuron 37
337.5
315.0
292.5
270.0
247.5
225.0
202.5
180.0
157.5
135.0
112.5
90.0
67.5
45.0
22.5
0.0
Direction (°)
0
250
500
750
1000
1250
1500
1750
2000
Time (ms)



1.4 Task 3: Fit and plot tuning functions

The goal is to visualize the activity of each neuron as a function of stimulus direction. First, compute the spike counts of each neuron for each direction of motion and trial. The result should be a matrix `x`, where x_{jk} represents the spike count of the j-th response to the k-th direction of motion (i.e. each column contains the spike counts for all trials with one direction of motion). If you used dataframes above, the `groupby()` function allows to implement this very compactly. Make sure you don't loose trials with zero spikes though. Again, other implementations are completely fine.

Fit the tuning curve, i.e. the average spike count per direction, using a von Mises model. To capture the non-linearity and direction selectivity of the neurons, we will fit a modified von Mises function:

$$f(\theta) = \exp(\alpha + \kappa(\cos(2 * (\theta - \phi)) - 1) + \nu(\cos(\theta - \phi) - 1))$$

Here, θ is the stimulus direction. Implement the von Mises function in `vonMises()` and plot it to understand how to interpret its parameters ϕ, κ, ν, α. Perform a non-linear least squares fit using a package/function of your choice. Implement the fitting in `tuningCurve()`.

Plot the average number of spikes per direction, the spike counts from individual trials as well as your optimal fit.

Select two cells that show nice tuning to test you code. (28, 37)

Grading: 3 pts

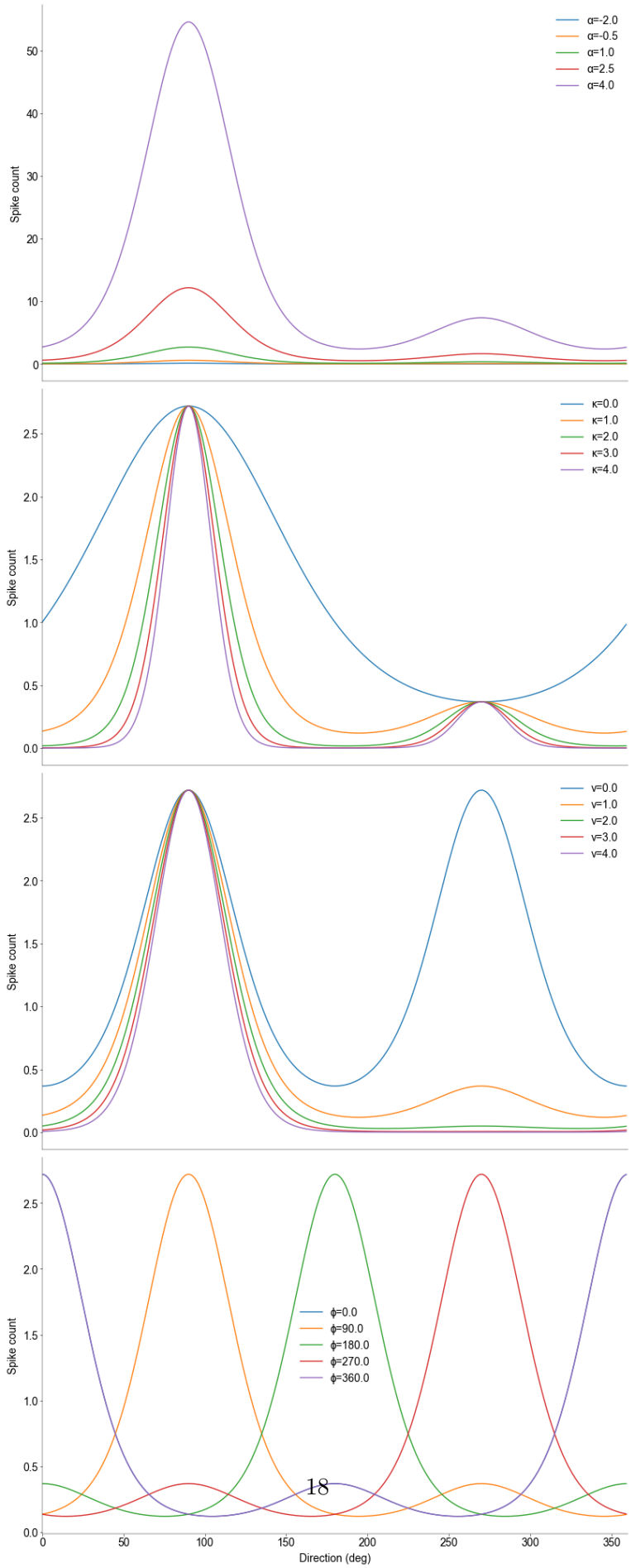
Answer: - α controls the height of the peak - κ controls the 'narrowness' of the peak (similar to inverse variance in Gaussian pdf) - ν controls the height of a second peak, 180° apart from the first one (at $\nu = 0$ the peak is the same height as the first one - at higher values of ν it is lower) - ϕ directly controls the position of the first peak (and indirectly the position of the second peak)

[ ]:
```
def vonMises( ,  ,  ,  ,  ):
    """Evaluate the parametric von Mises tuning curve with parameters p at␣
 ↪locations theta.

    Parameters
    ----------

     : np.array, shape=(N, )
        Locations. The input unit is degree.

     ,  ,  ,   : float
        Function parameters

    Return
    ------
    f: np.array, shape=(N, )
        Tuning curve.
    """

    # insert your code here

    # ------------------------------------
    # Implement the Mises model (0.5 pts)
    # ------------------------------------
     _  = np.deg2rad( ) - np.deg2rad( )
    f = np.exp(  +   * (np.cos(2 *  _ ) - 1) +   * (np.cos( _ ) - 1))

    return f
```

Plot the von Mises function while varying the parameters systematically.

[ ]:
```
#␣
 ↪-------------------------------------------------------------------------------
# plot von Mises curves with varying parameters and explain what they do (0.5␣
 ↪pts)
#␣
 ↪-------------------------------------------------------------------------------

fig, axs = plt.subplots(4, 1, figsize=(8, 20))
```

```
for   in np.linspace(-2, 4, 5):
    angles = np.arange(0, 360, 1)
    axs[0].plot(angles, vonMises(angles,  , 1, 1, 90), label=f"{ =}")

for   in np.linspace(0, 4, 5):
    angles = np.arange(0, 360, 1)
    axs[1].plot(angles, vonMises(angles, 1,  , 1, 90), label=f"{ =}")

for   in np.linspace(0, 4, 5):
    angles = np.arange(0, 360, 1)
    axs[2].plot(angles, vonMises(angles, 1, 1,  , 90), label=f"{ =}")

for   in np.linspace(0, 360, 5):
    angles = np.arange(0, 360, 1)
    axs[3].plot(angles, vonMises(angles, 1, 1, 1,  ), label=f"{ =}")

plt.legend()
for ax in axs:
    ax.legend()
    ax.set_xlim(0, 360)
    ax.set_ylabel("Spike count")
    if ax == axs[-1]:
        ax.set_xlabel("Direction (deg)")
    else:
        ax.set_xticks([])
```


α=-2.0
α=-0.5
α=1.0
α=2.5
α=4.0
Spike count
κ=0.0
κ=1.0
κ=2.0
κ=3.0
κ=4.0
Spike count
ν=0.0
ν=1.0
ν=2.0
ν=3.0
ν=4.0
Spike count
ϕ=0.0
ϕ=90.0
ϕ=180.0
ϕ=270.0
ϕ=360.0
Spike count
Direction (deg)

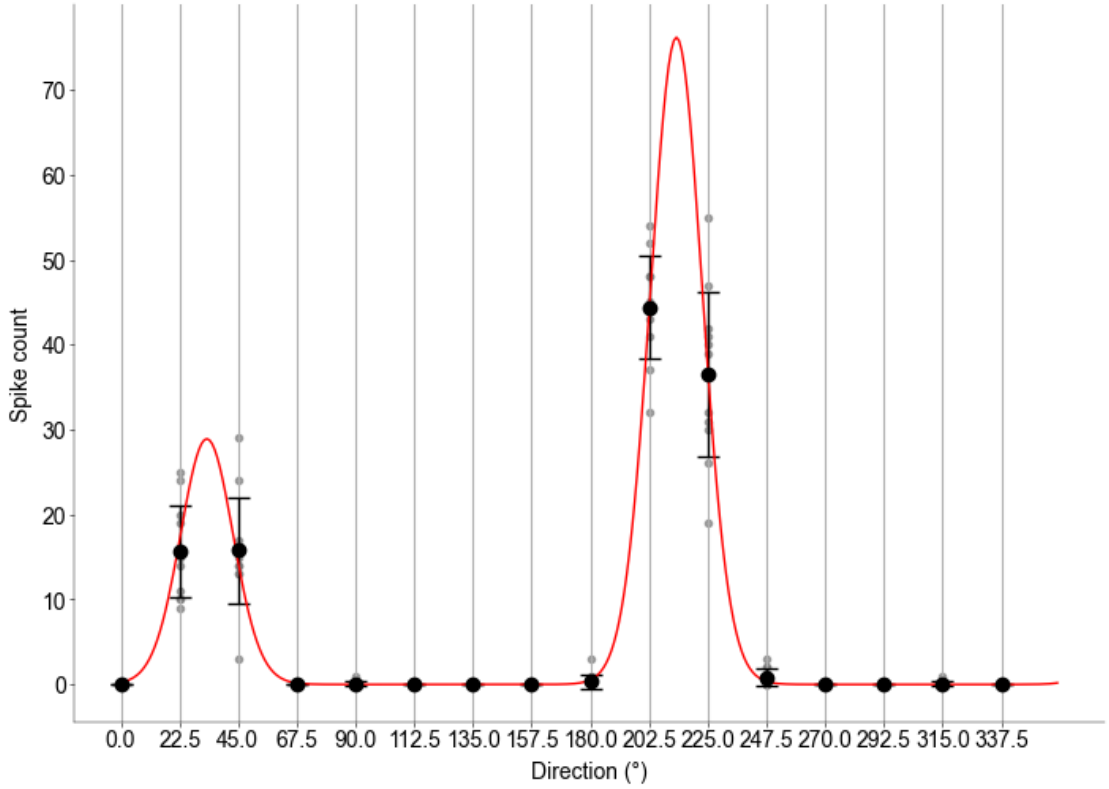
```
[ ]: def tuningCurve(counts, dirs, show=True):
        """Fit a von Mises tuning curve to the spike counts in count with direction␣
     ↪dir using a least-squares fit.

        Parameters
        ----------

        counts: np.array, shape=(total_n_trials, )
            the spike count during the stimulation period

        dirs: np.array, shape=(total_n_trials, )
            the stimulus direction in degrees

        show: bool, default=True
            Plot or not.


        Return
        ------
        p: np.array or list, (4,)
            parameter vector of tuning curve function
        """

        # insert your code here

        # ----------------------------------------
        # Compute the spike count matrix (0.5 pts)
        # ----------------------------------------
        sorted_directions = np.sort(np.unique(dirs))
        X = np.array([counts[dirs == dir] for dir in sorted_directions]).T
        assert X.shape == (dirs.shape[0] // len(sorted_directions),␣
     ↪len(sorted_directions))

        # ------------------------------------------------------------
        # fit the von Mises tuning curve to the spike counts (0.5 pts)
        # ------------------------------------------------------------
        # because there is easy access to local optima if
        #   is initialized incorrectly, we test multiple different inits for
        min_error = np.inf
        p = None
        for inits in [
            [0, 0, 0, 0],
            [0, 0, 0, 90],
            [0, 0, 0, 180],
```

```
        [0, 0, 0, 270],
        [0, 0, 0, 360],
    ]:
        _p, _ = opt.curve_fit(
            vonMises,
            dirs,
            counts,
            p0=inits,
            bounds=([0, 0, 0, 0], [np.inf, np.inf, np.inf, 360]),
        )
        error = np.sum((vonMises(dirs, *_p) - counts) ** 2)
        min_error = min(min_error, error)
        p = _p if min_error == error else p

    assert p is not None

    if show == True:
        # ------------------------------------------------
        # plot the data and fitted tuning curve (0.5 pts)
        # ------------------------------------------------

        fig, ax = plt.subplots(figsize=(7, 5))

        # the plot should contain both the data and the fitted curve
        # using seaborn makes this really easy

        ax.scatter(dirs, counts, color="gray", alpha=0.7)
        ax.errorbar(
            sorted_directions,
            X.mean(axis=0),
            X.std(axis=0),
            marker="o",
            capsize=5,
            linestyle="None",
            color="black",
        )
        angles = np.arange(0, 360, 1)
        ax.plot(angles, vonMises(angles, *p), label=str(p), color="red")
        ax.set_xlabel("Direction (°)")
        ax.set_ylabel("Spike count")
        ax.set_xticks(sorted_directions)
        ax.grid(axis="x")
        plt.show()

        return
    else:
        return p
```

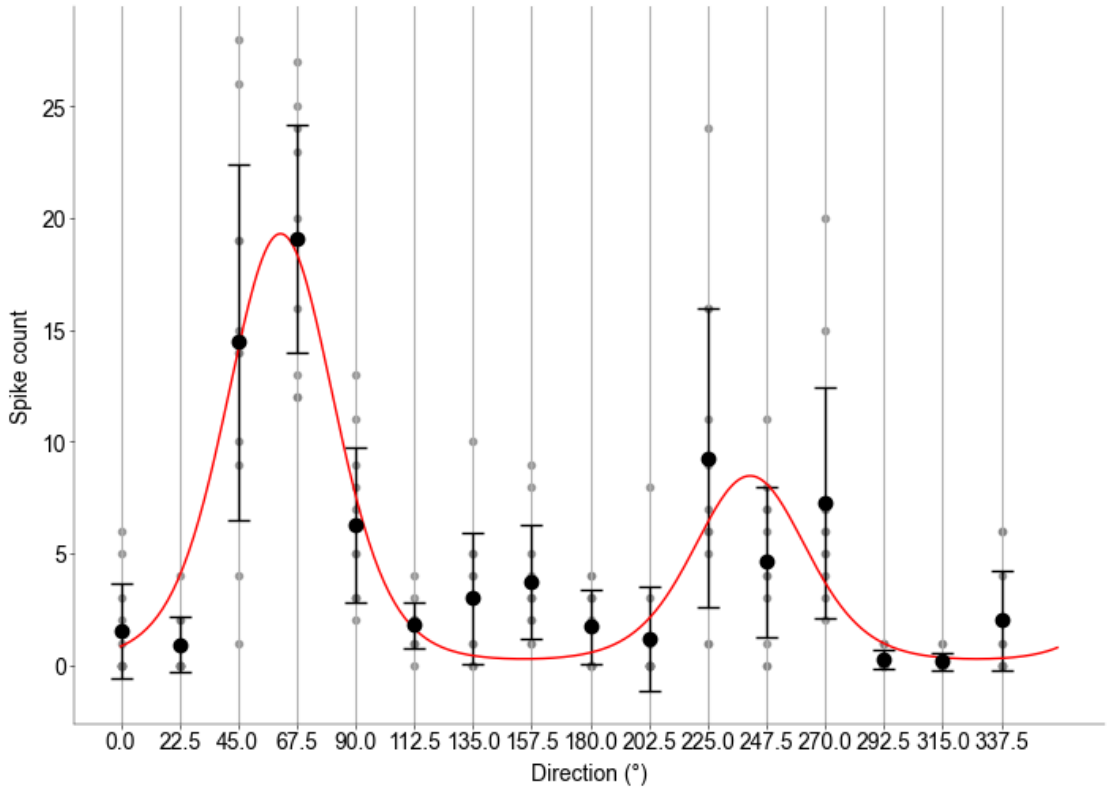
Plot tuning curve and fit for different neurons. Good candidates to check are 28, 29 or 37.

[ ]:
```
def get_data(spikes, neuron):
    spk_by_dir = (
        spikes[spikes["Neuron"] == neuron]
        .groupby(["Dir", "Trial"])["stimPeriod"]
        .sum()
        .astype(int)
        .reset_index()
    )

    dirs = spk_by_dir["Dir"].values
    counts = spk_by_dir["stimPeriod"].values

    # because we count spikes only when they are present, some zero entries in␣
 ↪the count vector are missing
    for i, Dir in enumerate(np.unique(spikes["Dir"])):
        m = nTrials - np.sum(dirs == Dir)
        if m > 0:
            dirs = np.concatenate((dirs, np.ones(m) * Dir))
            counts = np.concatenate((counts, np.zeros(m)))

    idx = np.argsort(dirs)
    dirs_sorted = dirs[idx]  # sorted dirs
    counts_sorted = counts[idx]

    return dirs_sorted, counts_sorted
```

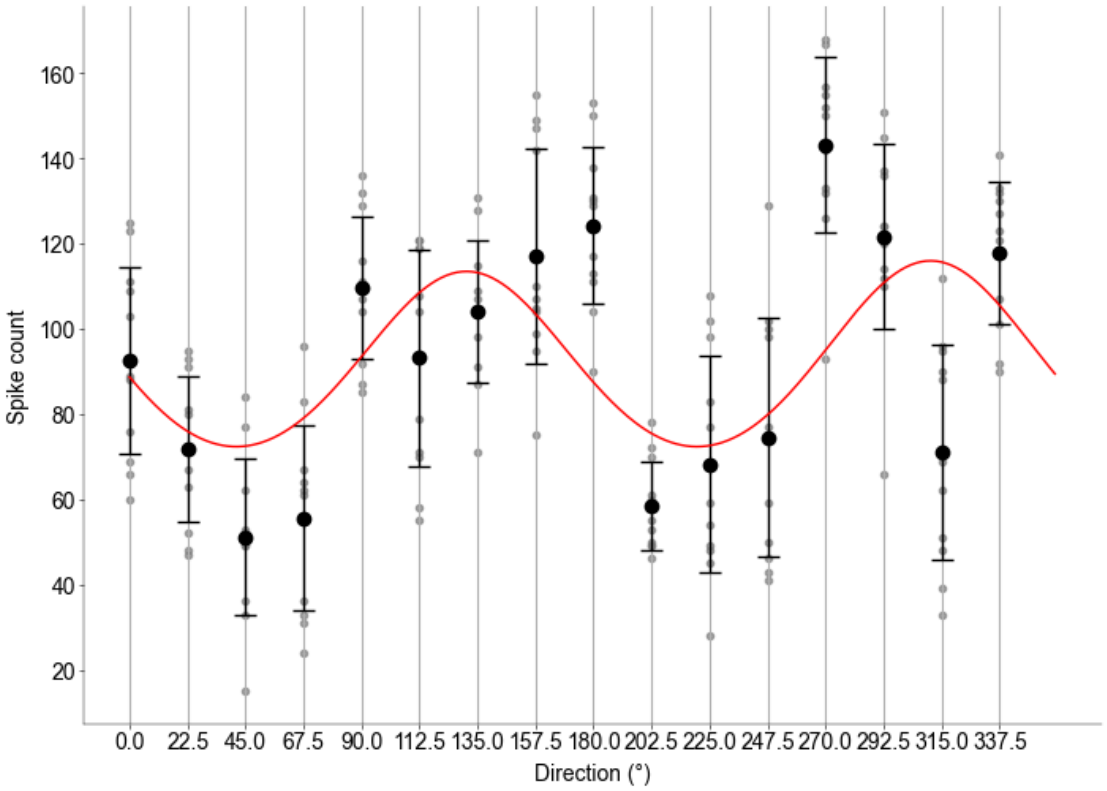
[ ]:
```
# ------------------------------------------------------------
# plot tuning curve and fit for different neurons (0.5 pts)
# ------------------------------------------------------------

dirs, counts = get_data(spikes, 28)
# add plot
tuningCurve(counts, dirs, show=True)
```



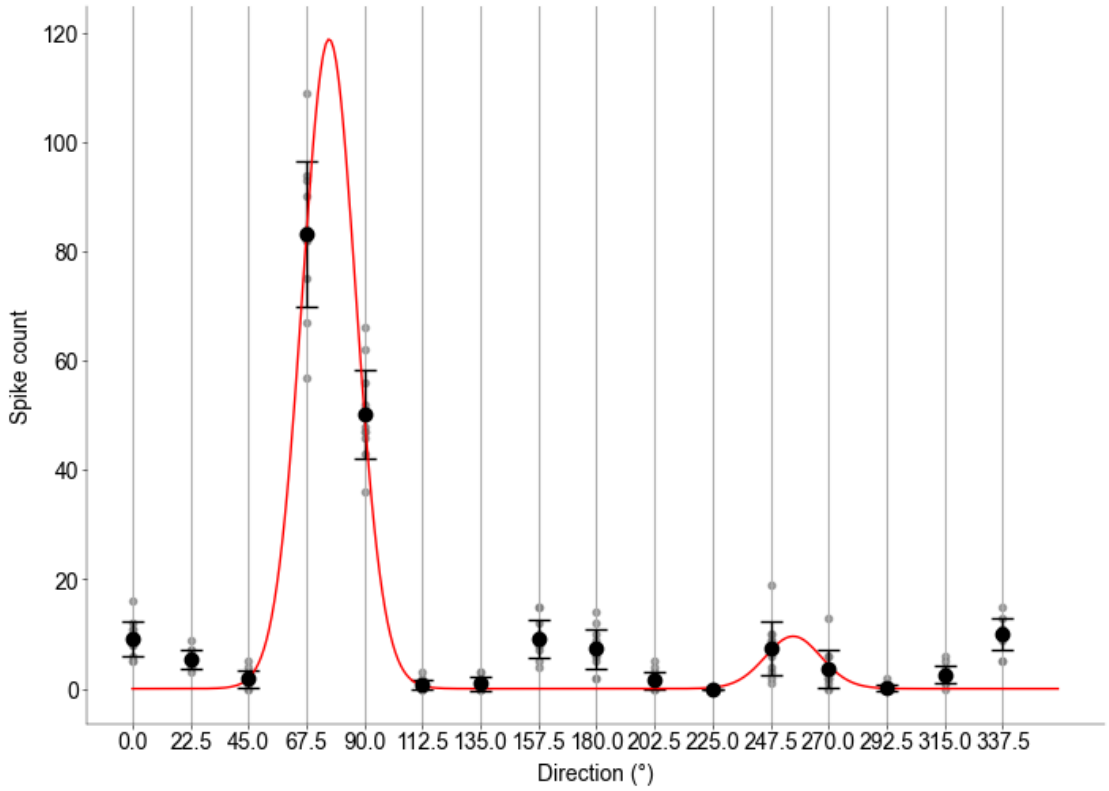
[ ]:
```
dirs, counts = get_data(spikes, 29)
# add plot
tuningCurve(counts, dirs, show=True)
```



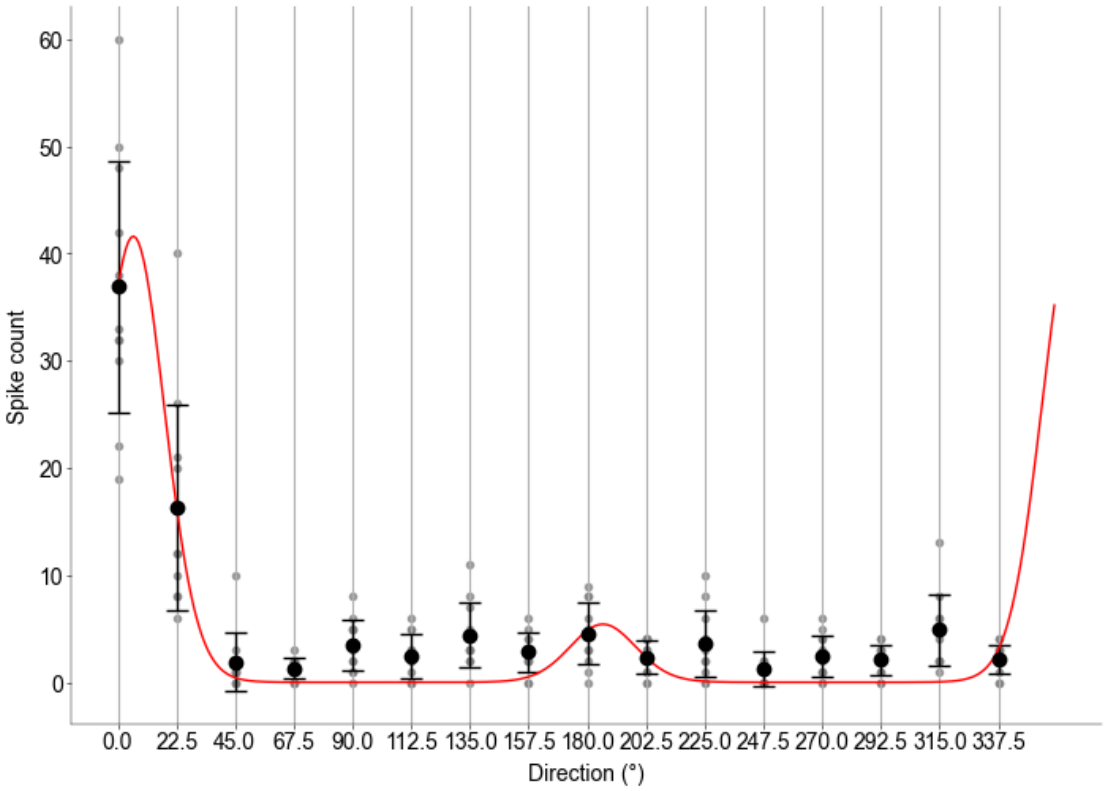
[ ]:
```
dirs, counts = get_data(spikes, 36)
# add plot
tuningCurve(counts, dirs, show=True)
```



[ ]:
```
dirs, counts = get_data(spikes, 37)
# add plot
tuningCurve(counts, dirs, show=True)
```



[ ]:
```
dirs, counts = get_data(spikes, 32)
# add plot
tuningCurve(counts, dirs, show=True)
```



1.5 Task 4: Permutation test for direction tuning

Implement a permutation test to quantitatively assess whether a neuron is direction/orientation selective. To do so, project the vector of average spike counts, $m_k = \frac{1}{N}\sum_j x_{jk}$ on a complex exponential with two cycles, $v_k = \exp(\psi i \theta_k)$, where θ_k is the k-th direction of motion in radians and $\psi \in 1, 2$ is the fourier component to test (1: direction, 2: orientation). Denote the projection by $q = m^T v$. The magnitude $|q|$ tells you how much power there is in the ψ-th fourier component.

Estimate the distribution of |q| under the null hypothesis that the neuron fires randomly across directions by running 1000 iterations where you repeat the same calculation as above but on a random permutation of the trials (that is, randomly shuffle the entries in the spike count matrix x). The fraction of iterations for which you obtain a value more extreme than what you observed in the data is your p-value. Implement this procedure in the function `testTuning()`.

Illustrate the test procedure for one of the cells from above. Plot the sampling distribution of |q| and indicate the value observed in the real data in your plot.

How many cells are tuned at $p < 0.01$?

Grading: 3 pts

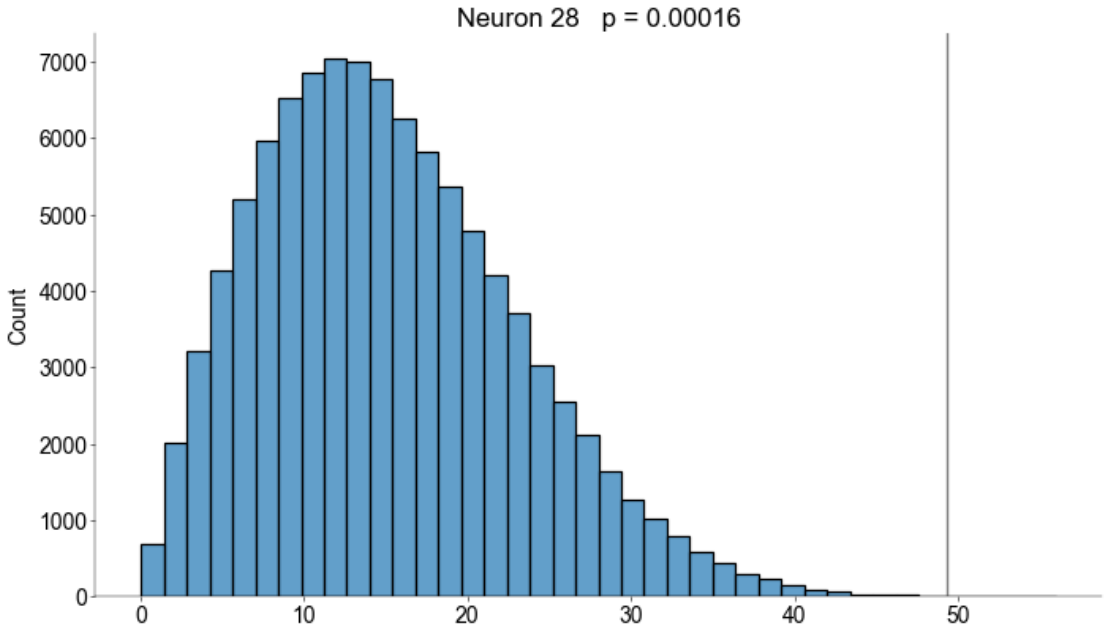
[ ]:
```
def testTuning(counts, dirs, psi=1, niters=1000, show=False, neuron=0):
    """Plot the data if show is True, otherwise just return the fit.
```

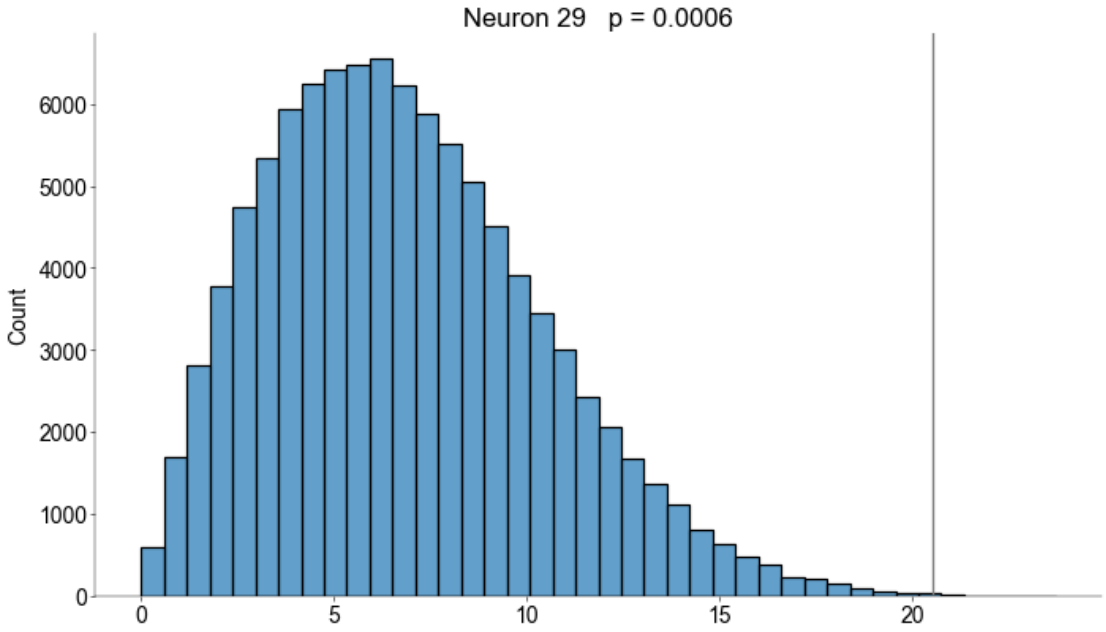
```
    Parameters
    ----------

    counts: np.array, shape=(total_n_trials, )
        the spike count during the stimulation period

    dirs: np.array, shape=(total_n_trials, )
        the stimulus direction in degrees

    psi: int
        fourier component to test (1 = direction, 2 = orientation)

    niters: int
        Number of iterations / permutation

    show: bool
        Plot or not.

    neuron: int
        number of neuron for title

    Returns
    -------
    p: float
        p-value
    q: float
        magnitude of second Fourier component

    qdistr: np.array
        sampling distribution of |q| under the null hypothesis

    """

    # insert your code here

    # --------------------------------
    # calculate m, nu and q (0.5 pts)
    # --------------------------------
    sorted_directions = np.sort(np.unique(dirs))
    X = np.array([counts[dirs == dir] for dir in sorted_directions]).T
    assert X.shape == (dirs.shape[0] // len(sorted_directions),␣
↪len(sorted_directions))

    m = X.mean(axis=0)
    theta = np.deg2rad(sorted_directions)
      = np.exp(psi * 1j * theta)
    q = np.abs(m.T @  )
```

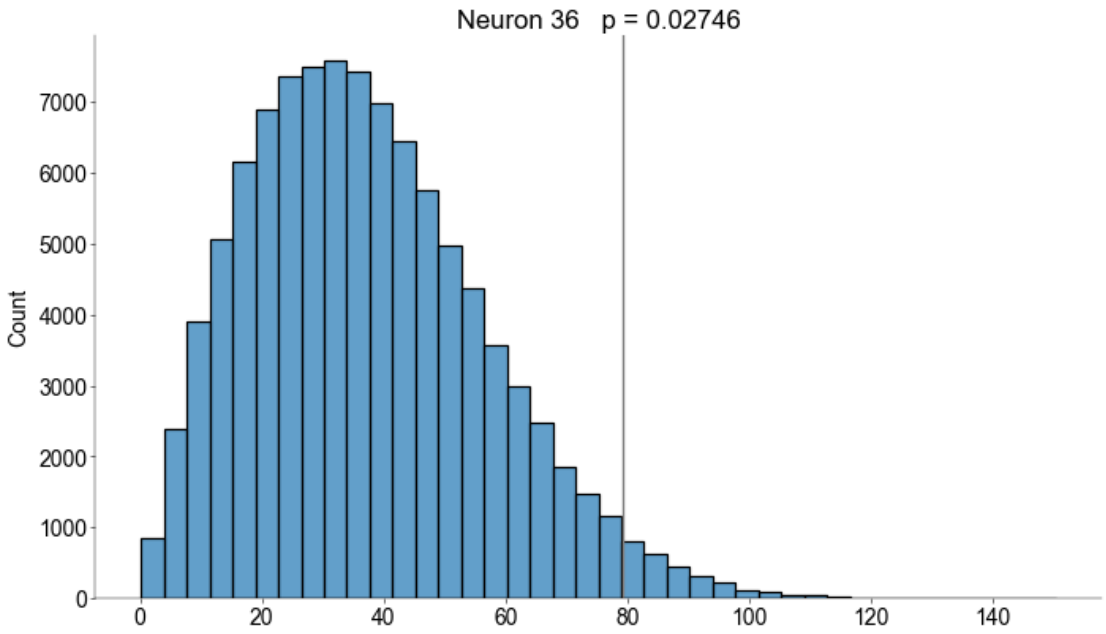
```
    # ------------------------------------------------------------------------
    # Estimate the distribution of q under the H0 and obtain the p value (1 pt)
    # ------------------------------------------------------------------------
    X_sample = X.copy()
    qdistr = np.zeros(niters)
    for i in range(niters):
        X_values = X_sample.flatten()
        np.random.shuffle(X_values)
        X_sample = X_values.reshape(X.shape)

        m_sample = X_sample.mean(axis=0)
        qdistr[i] = np.abs(m_sample.T @  )

    p = (qdistr > q).mean()

    if show == True:
        # -------------------------------
        # plot the test results (0.5 pts)
        # -------------------------------
        fig, ax = plt.subplots(figsize=(7, 4))
        # you can use sns.histplot for the histogram
        sns.histplot(qdistr, ax=ax, bins=40, alpha=0.7)
        ax.axvline(q, color="gray", label="q")
        ax.set_title(f"Neuron {neuron}   p = {p}")
    else:
        return p, q, qdistr
```

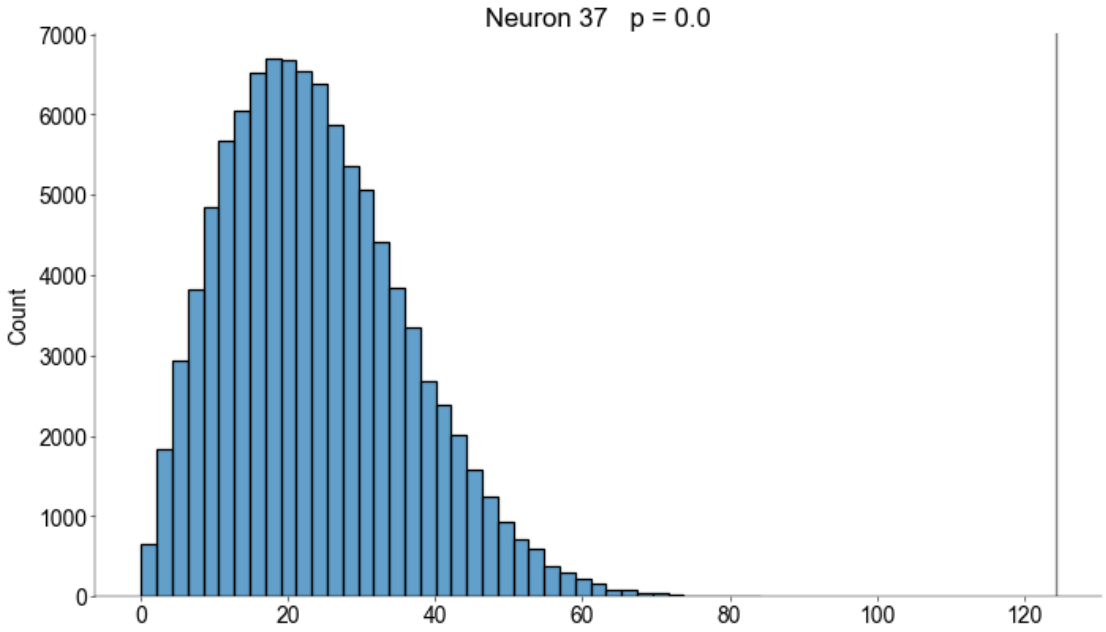
Show null distribution for the example cell:

[ ]:
```
# ------------------------------------------------------------
# Plot null distributions for example cells 28 & 29. (0.5 pts)
# ------------------------------------------------------------
neurons = [28, 29, 36, 37]
for neuron in neurons:
    dirs, counts = get_data(spikes, neuron)
    testTuning(counts, dirs, show=True, niters=100000, neuron=neuron)
```


Neuron 28 p = 0.00016
Count
7000
6000
5000
4000
3000
2000
1000
0
0
10
20
30
40
50


Neuron 29 p = 0.0006
Count
6000
5000
4000
3000
2000
1000
0
0
5
10
15
20





Test all cells for orientation and direction tuning

[ ]:
```
# ----------------------------------------------------
# Test cells for orientation / direction tuning (0.5 pts)
# ----------------------------------------------------
```

```
neurons = np.sort(spikes["Neuron"].unique())
p_values = np.zeros((len(neurons), 2))
for i, neuron in enumerate(neurons):
    dirs, counts = get_data(spikes, neuron)
    p_values[i, 0], _, _ = testTuning(counts, dirs, psi=1)
    p_values[i, 1], _, _ = testTuning(counts, dirs, psi=2)

# collect p values for orientation / direction selectivity
p_values.shape
```

[ ]: (41, 2)

Number of direction tuned neurons:

[ ]:
```
# count cells with p > 0.01 (which ones are they?)
print(
    f"Number of neurons with p > 0.01: {np.sum(p_values[:, 0] > 0.01)} of␣
 ↪{len(neurons)}"
)
print(f"The neurons are: {neurons[p_values[:, 0] > 0.01]}")
print(
    "These are the cells where we can NOT reject H0, i.e. they are not␣
 ↪neccessarily significantly tuned for direction."
)
```

```
Number of neurons with p > 0.01: 29 of 41
The neurons are: [ 1  2  3  4  5  6  7  8  9 10 11 12 14 15 16 17 18 19 21 22 23
26 30 33
 34 35 36 39 41]
These are the cells where we can NOT reject H0, i.e. they are not neccessarily
significantly tuned for direction.
```

[ ]:
```
# count cells with p < 0.01 (which ones are they?)
print(
    f"Number of neurons with p < 0.01: {np.sum(p_values[:, 0] < 0.01)} of␣
 ↪{len(neurons)}"
)
print(f"The neurons are: {neurons[p_values[:, 0] < 0.01]}")
print(
    "These are the cells where we can reject H0, i.e. they are significantly␣
 ↪tuned for direction."
)
```

```
Number of neurons with p < 0.01: 12 of 41
The neurons are: [13 20 24 25 27 28 29 31 32 37 38 40]
These are the cells where we can reject H0, i.e. they are significantly tuned
for direction.
```

Number of orientation tuned neurons:

[ ]:
```
# count cells with p > 0.01 (which ones are they?)
print(
    f"Number of neurons with p > 0.01: {np.sum(p_values[:, 1] > 0.01)} of␣
 ↪{len(neurons)}"
)
print(f"The neurons are: {neurons[p_values[:,1] > 0.01]}")
print(
    "These are the cells where we can NOT reject H0, i.e. they are not␣
 ↪neccessarily significantly tuned for orientation."
)
```

```
Number of neurons with p > 0.01: 7 of 41
The neurons are: [ 1  4  5  9 11 19 35]
These are the cells where we can NOT reject H0, i.e. they are not neccessarily
significantly tuned for orientation.
```

[ ]:
```
# count cells with p < 0.01 (which ones are they?)
print(
    f"Number of neurons with p < 0.01: {np.sum(p_values[:, 1] < 0.01)} of␣
 ↪{len(neurons)}"
)
print(f"The neurons are: {neurons[p_values[:,1] < 0.01]}")
print(
    "These are the cells where we can NOT reject H0, i.e. they are not␣
 ↪neccessarily significantly tuned for orientation."
)
```

```
Number of neurons with p < 0.01: 34 of 41
The neurons are: [ 2  3  6  7  8 10 12 13 14 15 16 17 18 20 21 22 23 24 25 26 27
28 29 30
 31 32 33 34 36 37 38 39 40 41]
These are the cells where we can NOT reject H0, i.e. they are not neccessarily
significantly tuned for orientation.
```