Table of Contents

Directions for homework submission

Submit each of your homework to canvas as the pdf output of the jupyter notebook, and a zip of the jupyter notebook (.ipynb), and all other necessary attachments (e.g. images). Name your files starting in the format of "Last_name_First_name_File_name" separated by underscores.

For example, Jieyu submits two files for her homework this week:

1. Zheng_Jieyu_HW1.pdf (the pdf output of the jupyter notebook)
2. Zheng_Jieyu_HW1.zip, which contains the following files:
   - Zheng_Jieyu_HW1.ipynb
   - Zheng_Jieyu_HW1_q31_schematic1.png (one schematic embedded in the notebook)
   - Zheng_Jieyu_HW1_q31_schematic2.png (another schematic embedded in the notebook)
   - ...and other files.

Please make sure your notebook can be run without errors within the cns187 virtual environment. Any file that fails to be executed on TA's end will be considered as late submissions.

Caltech Honor code: Searching for the solutions online is strictly prohibited. You should refer to the textbooks and lecture slides. If you are citing any external sources online, please include a list of references.

Collaboration on homework assignments is encouraged. However, **you cannot show each other the numerical answers or codes**. Please note at the beginning of each answer whom you have discussed the problems with (including TAs).

All the mathematics should be typed in Latex format. You may work on a piece of paper and then type it into the notebook. Here is a useful cheat sheet (http://users.dickinson.edu/~richesod/latex/latexcheatsheet.pdf). Please do not submit pictures of handwritten maths.

For the schematic and drawings to be submitted, please include clear pictures or screenshots under the same directory as your homework and upload them together. Display the images in markdown cells in your homework.

Please make sure that all your plots include a title and axis labels with units. One point will be deducted for each missing element.

In [174]:
```
# Imports
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
import seaborn as sns
import scipy
```

Order of Magnitude Neuroscience: Sodium indicator? (10 pts)

Optical recording of neuronal activity works by measuring the calcium concentration in the cell with a fluorescent indicator. This works when there are voltage-dependent calcium channels. But why not measure directly the ions that produce the action potential, like Na and K? From what you know about neurons and action potentials, give a back-of-the-envelope estimate of how much the Na concentration in the cell changes after a spike.

Hint: You got some numbers of intracellular Na concentrations, lengths of axons, capacitance through previous homework and lectures. Assume your action potential changes the membrane potential by 100 mV, how many sodium ions need to flow in to change the charges?

Simulate Mainen and Sejnowski 1995 (25 pt)

This exercise explores the response of a neuron to a rapidly varying input current. Specifically the goal is to replicate the experimental results of Mainen and Sejnowski (1995) Figure 1 on a simple model neuron.

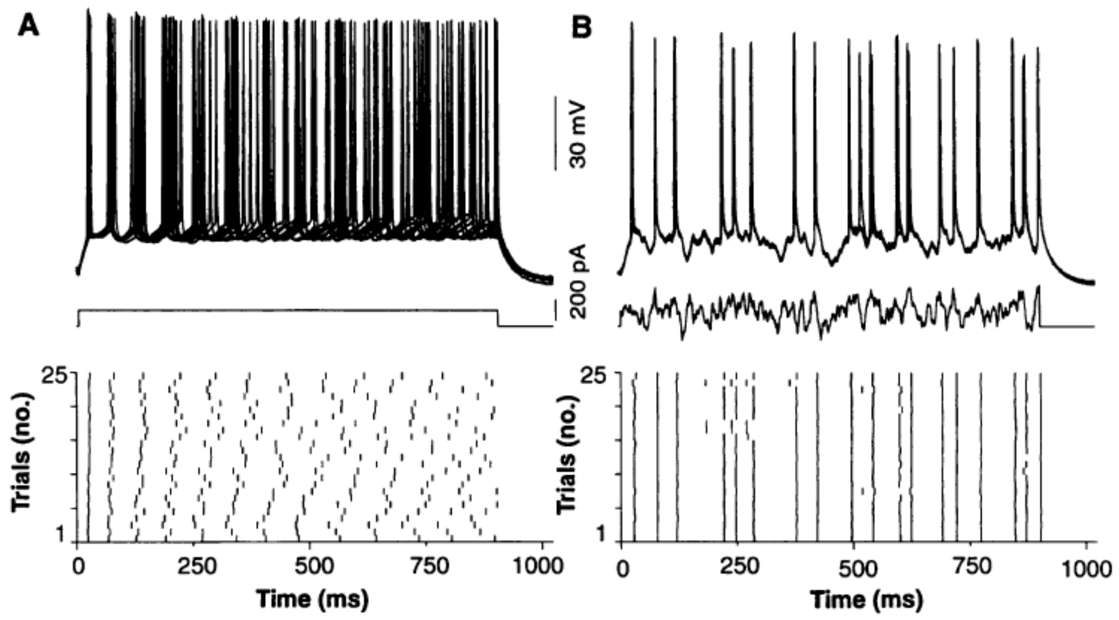


Fig. 1. Reliability of firing patterns of cortical neurons evoked by constant and fluctuating current. (**A**) In this example, a superthreshold dc current pulse (150 pA, 900 ms; middle) evoked trains of action potentials (approximately 14 Hz) in a regular-firing layer-5 neuron. Responses are shown superimposed (first 10 trials, top) and as a raster plot of spike times over spike times (25 consecutive trials, bottom). (**B**) The same cell as in (A) was again stimulated repeatedly, but this time with a fluctuating stimulus [Gaussian white noise, μ_s = 150 pA, σ_s = 100 pA, τ_s = 3 ms; see (*14*)].

The "startling" result of this paper was that injection with a constant curent (A) produces spike trains that are different from trial to trial, whereas injection with a fluctuating current (B) leads to very reproducible spike trains. Your task is to explain this result based on a simple model.

Use the following equation for your simulation:

$$\begin{aligned}
\tau\frac{dV}{dt} &= -V(t) + RI(t) + N(t)\\
I(t) &= \text{ injected current}\\
R &= \text{ membrane resistance}\\
N(t) &= \text{ noise voltage}\\
\text{When } V &= V_{threshold} \text{ emit a spike and reset } V \text{ to } V_{reset}
\end{aligned}$$

Here we use the following parameters: membrane resistance is 200 MΩ, resting potential is 0 (therefore not in the equation), time constant τ is 10 ms, $V_{threshold}$ is 30 mV, V_{reset} is -10 mV, refractory period is 5 ms, amplitude of the noise (standard deviation of the Gaussian noise) is 5 mV.

Provided below is a slightly modified version of the LIF model as you have used from last week. Please add the prompted specs to the model, and use it for simulation of the next two questions.

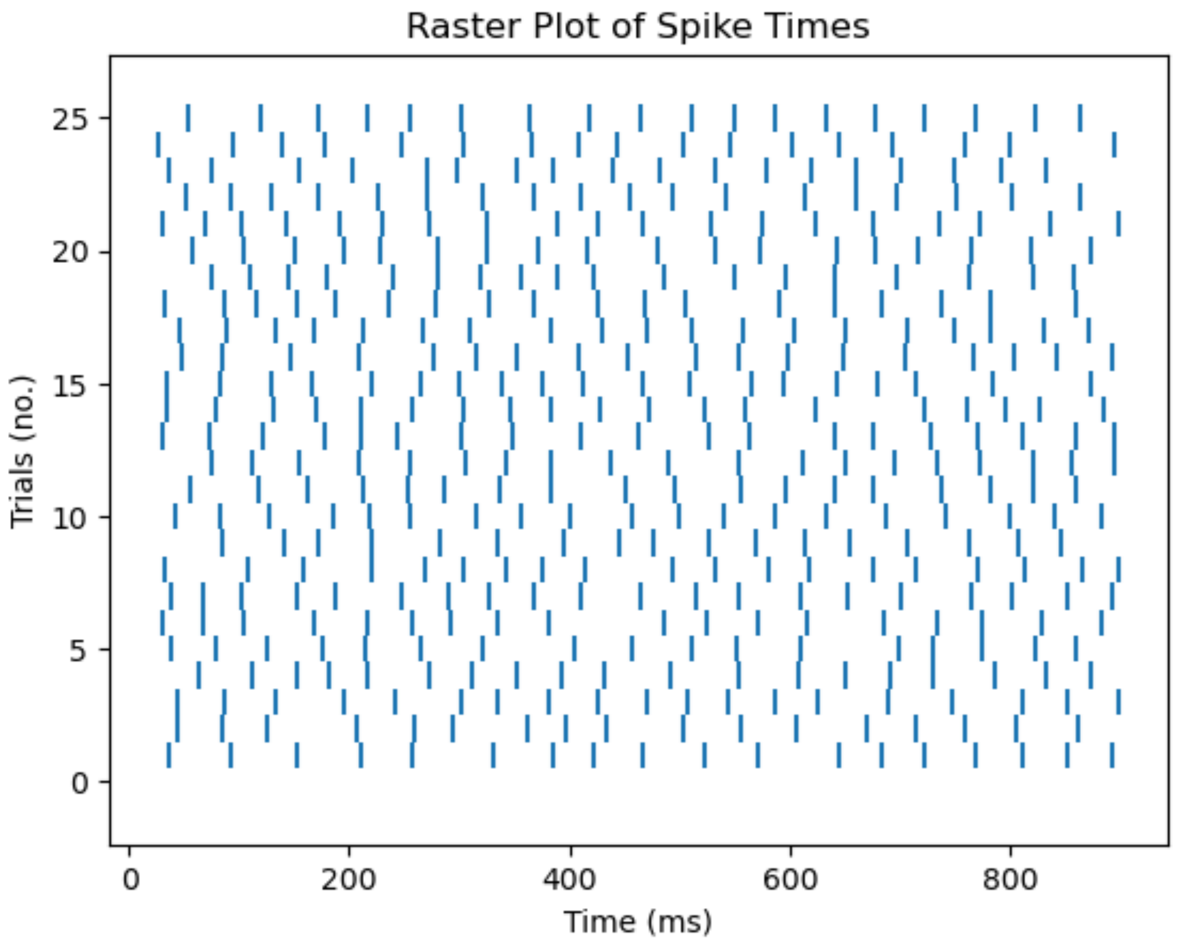
In [65]:
```
def run_LIF(Iinj,Noise):
  """
  Simulate the LIF dynamics with external input current
  With a duration of 1 second.
  Args:
    Iinj       : input current [pA]. The injected current has to be ar
                 of total length 1sec/dt
    Noise      : added noise term [mV]. This also has to be an array o
                 1sec/dt
  To be specified in the section below:
      V_threshold : threshold of action potential for the neuron, in m
      V_reset: post AP reset potential, in mV
      R: membrane resistance, in mega ohm
      E_resting: resting membrane potential, in mV
      tau_rc: membrane RC time constant, in ms
      t_ref: refractory period time length, in ms
  Returns:
    rec_sp     : array of time points at which a spike is fired, in ms
                 firing rate, note that the simulation is set for a to
  """

# Set parameters
  V_threshold = 30
  V_reset = -10
  R = 200e9
  E_resting = 0
  tau_rc =  10
  tref = 5
#some preset values
# timestep size, in msec
  dt = 1
```

```
# range of timesteps covered, in unit of dt, set to total length of 10
  range_t =  np.linspace(0, 1000, int((1000-0)/dt) + 1)

# total number of timesteps
  Lt = range_t.size

#initialize membrane potential at resting potential
  V_init = E_resting

# Initialize voltage
  v = np.zeros(Lt)
  v[0] = V_init


# Loop over time
  rec_spikes = []  # record spike times
  tr = 0.  # the count for refractory duration

  for it in range(Lt - 1):

    if tr > 0:  # check if in refractory period
        v[it] = V_reset  # set voltage to reset
        tr = tr - 1 # reduce running counter of refractory period

    elif v[it] >= V_threshold:  # if voltage over threshold
      rec_spikes.append(it)  # record spike event
      v[it] = V_reset  # reset voltage
      tr = tref / dt  # set refractory time


    # Calculate the increment of the membrane potential
    # Note the additional noise term compared to last week's equation
    dv = (-(v[it] - E_resting) + Iinj[it] *R/1000+Noise[it]) * (dt / t

    # Update the membrane potential
    v[it + 1] = v[it] + dv

   # Get spike times in ms
  rec_spikes = np.array(rec_spikes) * dt

  return rec_spikes
```

Simulate Panel A of Fig 1 (10 pts)

Simulate constant input with Gaussian noise.

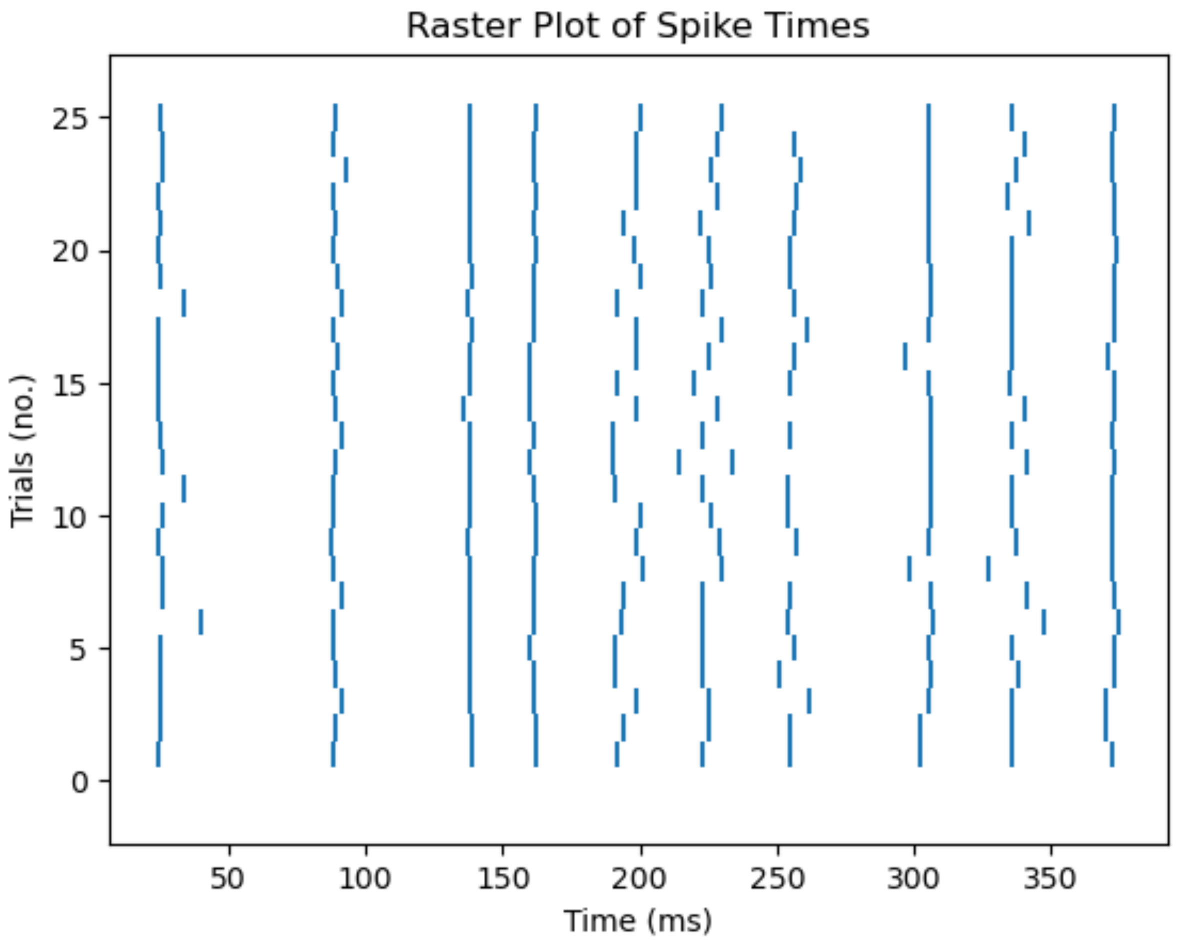
In [66]:
```
Iinj = np.full((1000,), 150e-9)
Iinj[900:] = 0

trials = []
trials.append([])
for trial in range(25):
    Noise = np.random.normal(0, 5, (1000,))
    trials.append(run_LIF(Iinj, Noise))

plt.eventplot(trials)
plt.title("Raster Plot of Spike Times")
plt.xlabel("Time (ms)")
plt.ylabel("Trials (no.)")
plt.show()
```



Simulate Panel B of Fig 1 (10 pts)

The figure caption spells out how the stimuli in A and B are related. $\tau_\mathrm{s} = 3$ ms refers to the correlation time of the fluctuating stimulus. It was obtained by passing Gaussian white noise through a low-pass filter with exponential impulse response $\exp(-t/\tau_\mathrm{s})$ (Hint: use `np.convolve`).

In [161]:
```
trials = []
trials.append([])

Iinj = np.random.normal(150e-9, 100e-9, (1000,))
Iinj[900:] = 0
low_pass_filter = np.exp(-(np.arange(1000) + 1)/3) / np.sum(np.exp(-(r
Iconv = np.convolve(Iinj, low_pass_filter, mode="same")

for trial in range(25):
    Noise = np.random.normal(0, 5, (1000,))
    rec_spikes = run_LIF(Iconv, Noise)
    trials.append(rec_spikes)


plt.eventplot(trials)
plt.title("Raster Plot of Spike Times")
plt.xlabel("Time (ms)")
plt.ylabel("Trials (no.)")
plt.show()
```



Comparison (5 pts)

Explain in a few words why the spike train in B is more predictable.

The spike train in B is more predictable because there is a certain degree of noise in the change of the voltage that occurs even with constant current input. This noise comes simply from the stochasticity that is a natural property of all neurons. However, with the current input being varied by a normal distribution, the average response is consistent due to the randomness on the injected current causing the natural noise from the neuron and system to be considered neglible since it is additive to the varied input with a variability much lower than that of the injected current. Therefore running this same current through various trial, will produce more reliable results since virtually the same noise is introduced in each trial making it more so a property of the experiment that controls consistency by reducing the effects of the neuron's natural noise.

L-N Model Estimation (25 pt)

For this problem, we're going to study the responses of retinal ganglion cells to flickering noise using the Linear-Nonlinear model we learned about in class. `Data_Hwk3.mat` provides you with recordings from two RGC's whose spiking activity was recorded concurrently during stimulus presentation.

Hint: we need to compute the inner product between this strf and the stimulus that was presented over a given N frames (i.e. another 640 x N matrix). To do this, we turn both the strf and the stimulus into vectors, both of length 640N x 1 and compute the standard dot product between them. Be careful when constructing these vectors from matrices! They need to be generated in a consistent way (e.g. stacking all columns on top of each other in order), otherwise the dot products won't make sense anymore.

Preparation: Download the .mat file posted on Canvas. Inside you should find the following variables:

$$\begin{aligned} t &: \text{Nt x 1, vector denoting recording time (sampled at 60 Hz)} \\ N &: \text{number of frames over which we'll be conducting the STA} \\ spikes &: \text{Nt x 3, Matrix of binned spike counts. Each row corresponds to number of sp} \\ r &: \text{Nt x 640, Binary matrix of the full stimulus movie used for the experiment.} \end{aligned}$$

In [73]:
```
mat = scipy.io.loadmat('Data_Hwk3.mat')

#Unpack variables
N = mat['N'][0][0]
r = mat['r']
t = mat['t']
spikes = mat['spikes']
```

Step 1: Computing the linear filter: STA (15 pts)

The "stimulus" used in these experiments was a noisy flickering rectangle on a screen with dimensions 1 x 640 pixels, as a matrix of black and white values (-1,+1) with dimensions 640 x Nt, where Nt is number of time points in the recording. We will compute the STA over a window of time that is N frames long. So, after doing our averaging, we will have computed a "spatio-temporal receptive field" (strf) with dimensions 640 x N.

Fill in the method below to properly compute an STA for the specified inputs.

Hint: We implemented something very similar to this last homework. Two things have changed this time.

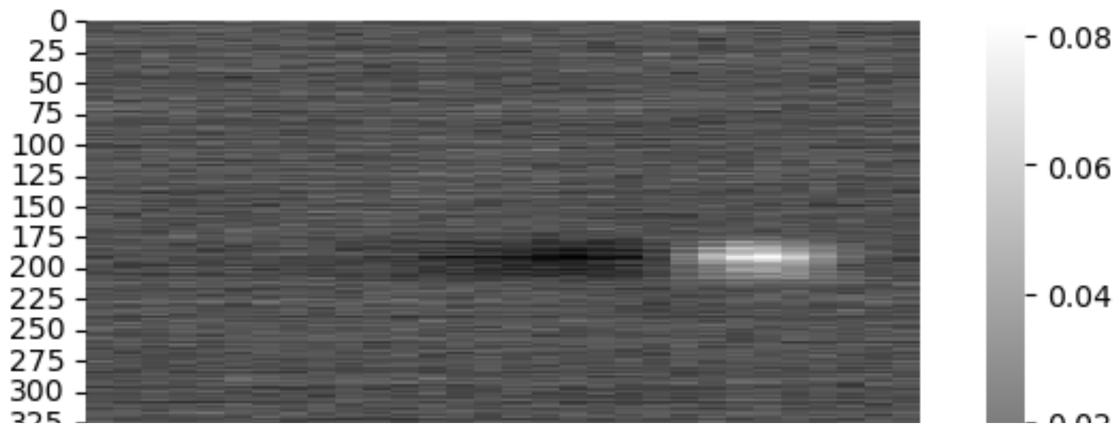
1. Instead of a single point that varies in time as the stimulus, we have a 640 x 1 stimulus that varies with time.
2. There might be multiple spikes within each bin in the spike-count vector. Need to account for this when computing the STA with this data.

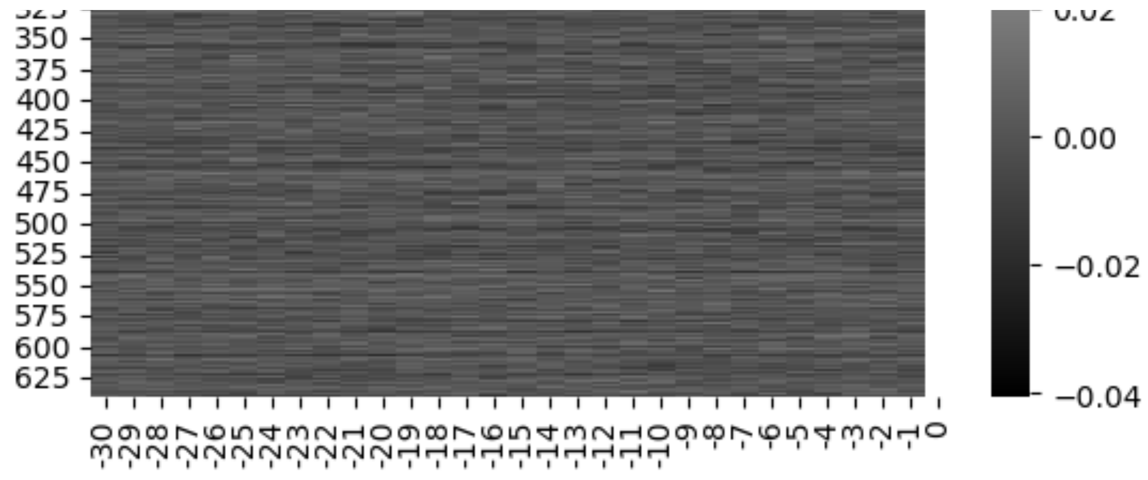
In [97]:
```python
def compute_sta(X, y, k):
    ''' Compute the spike triggered average from a stimulus and a spik

    Args:
    X: matrix of stimulus over time (640D)
    y: spike count vector. Could potentially have more than one spike
    k: time window for STA

    Return:
    sta: vector of length k that is the STA'''

    sta = np.zeros(shape=(X.shape[1], k))

    for i in range(k, len(y)):
        if y[i] != 0:
            sta += np.transpose(X[i-k:i,:])

    sta = sta / np.count_nonzero(y)
    return sta
```

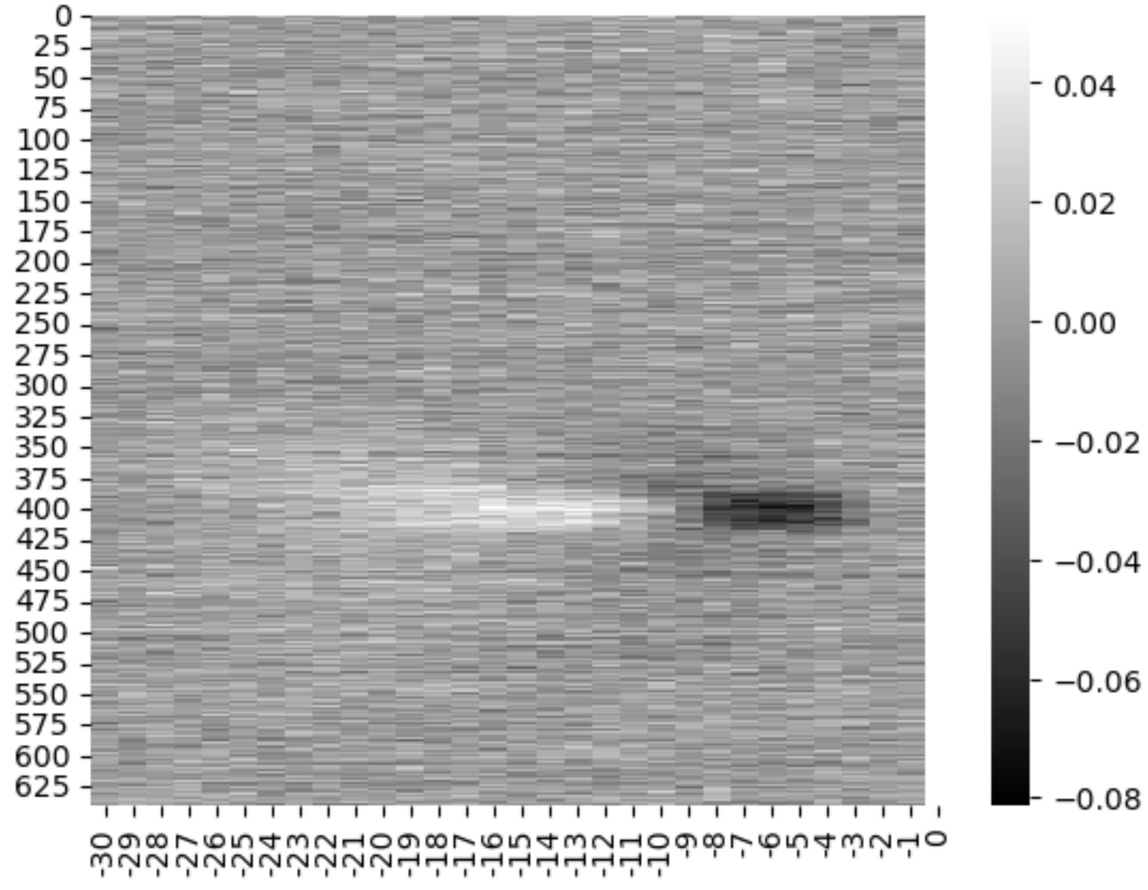
You can then implement the method to compute the linear filter for each of the three neurons we've given you.

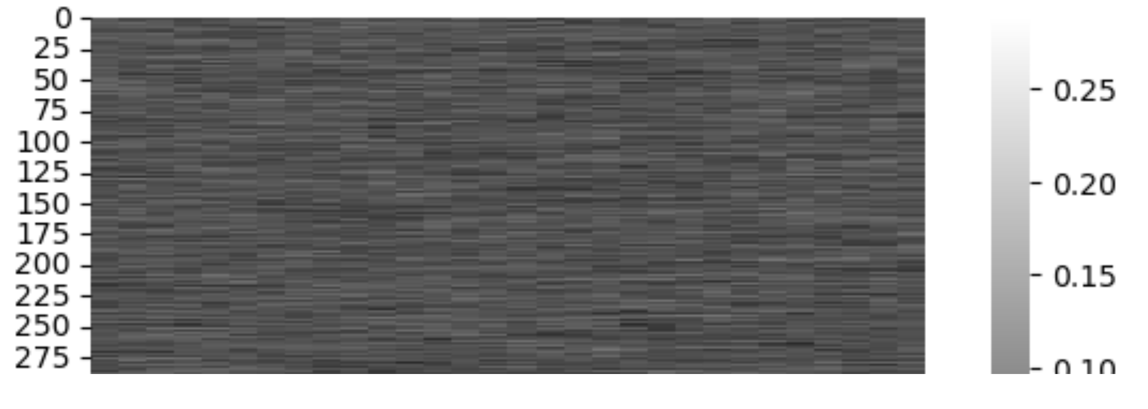
Hint: the data have two dimensions (space and time). To visualize this strf, we need to plot an image. We can do this in grayscale, with everything that appears lighter indicating on-average brighter pixels at that time point preceding a spike, and darker indicating darker pixels at that point preceding a spike.

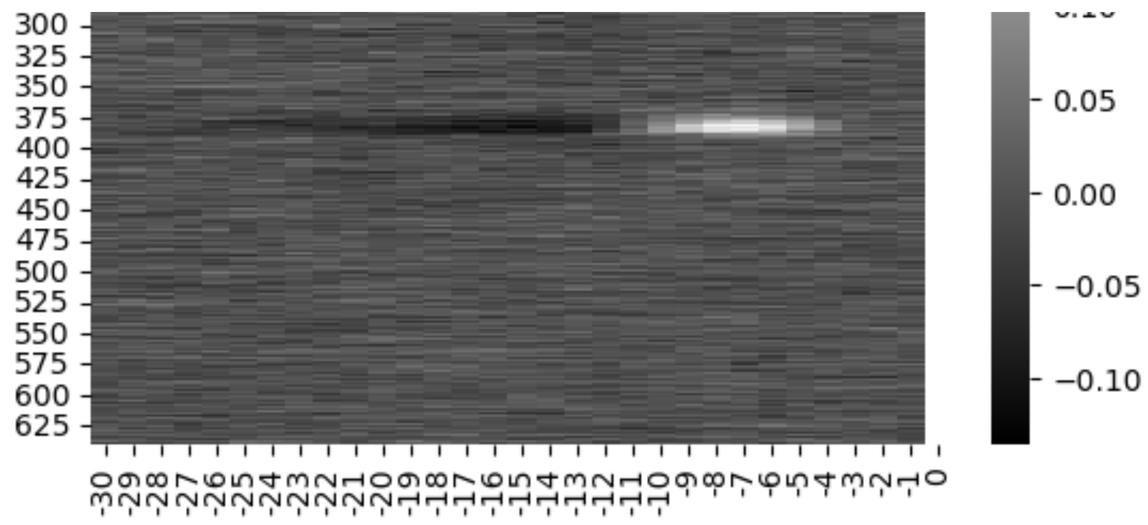
In [115]:
```python
neuron_stas = []
for neuron in range(spikes.shape[1]):
    neuron_stas.append(compute_sta(r, spikes[:,neuron], N))

for sta in neuron_stas:
    sns.heatmap(sta, cmap="gray", xticklabels=np.arange(-30,1))
    plt.show()
```




350
375
400
425
450
475
500
525
550
575
600
625
-30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0
0.00
−0.02
−0.04


0
25
50
75
100
125
150
175
200
225
250
275
300
325
350
375
400
425
450
475
500
525
550
575
600
625
-30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17 -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08


0
25
50
75
100
125
150
175
200
225
250
275
0.25
0.20
0.15



Describe qualitatively what each strf looks like and the kinds of visual features these neurons are responsive to as a result of the shape of their strf. Contrast these STA's to the one you computed for the H1 neuron in last homework. What effect does the difference in shape have on the kinds of features each of these neruons responds to?

Each strf seems like a field representing the brightness/darkness in the visual space of the stimulus preceding a spike. In comparison to the STA from the H1 neuron in the last homework, this seems to represent a spatiotemporal representation of what causes a spike in the 3 neurons, as opposed to simply showing the average time before a spike as computed for the H1 STA. For the 3 neurons, this shape then shows at what position in the stimulus field a certain intensity in the light/darkness triggers a spike on average for a given time before the spike. This can then give information on not only response to intensity but also to responsiveness from light to certain positions in the visual field.

Step 2: Computing the Non-Linearity (10 pts)

We have some linear filter θ that describes the stimulus features that the neuron cares about. In the model, the neuron essentially is taking a dot product between this filter and the incoming stimulus x. The outcome of this dot product, $\theta^T x$, is then fed into the non-linearity f, so that the firing rate is generated at the output as $f(\theta^{\mathrm{T}} x)$. So, we need a function f that is defined over various values of this dot product.

Hint:

a) We will go back and compute the projection of every time window from the initial stimulus onto the strf for each neuron, then do some binning over the values of all of these dot products.

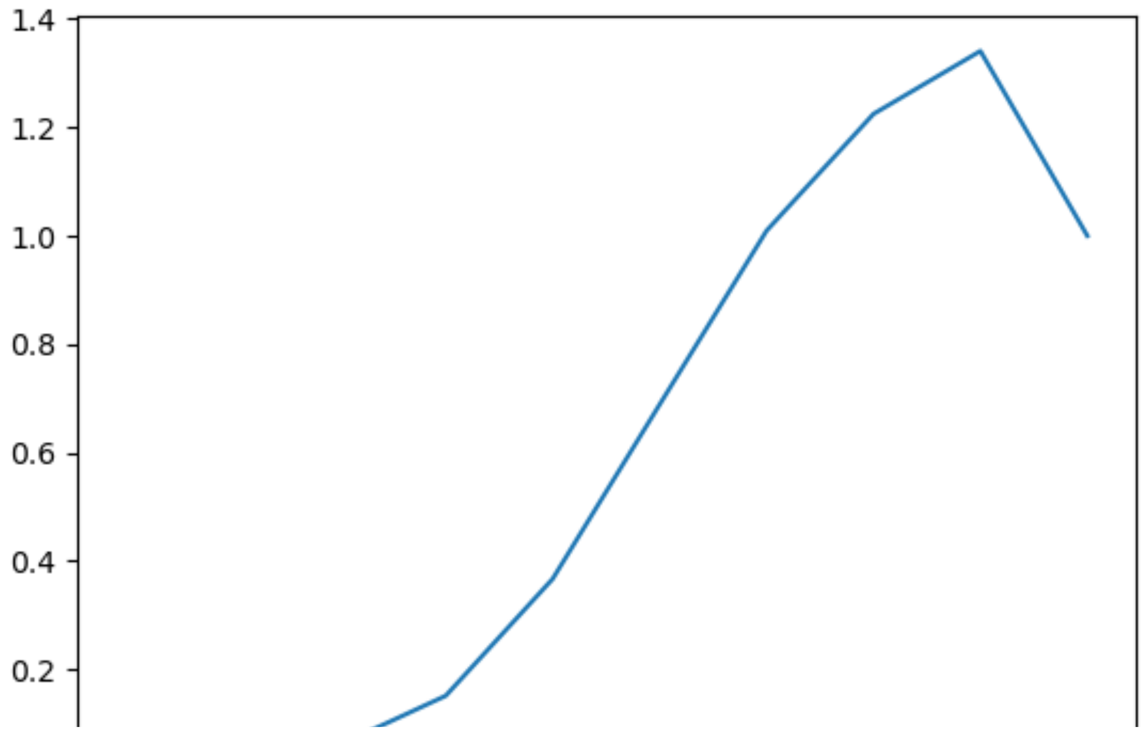
b) We will then compute the average firing rate of each neuron for each of these bins. This way, we get an estimate for the average number of spikes a neuron discharges when the stimulus - STA dot product is between a certain range of values. This is an estimate for the non-linearity.

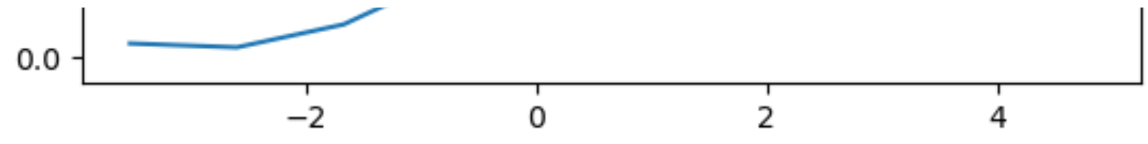
Follow the skeleton code for methods, but feel free to wipe them out and do it your own way.

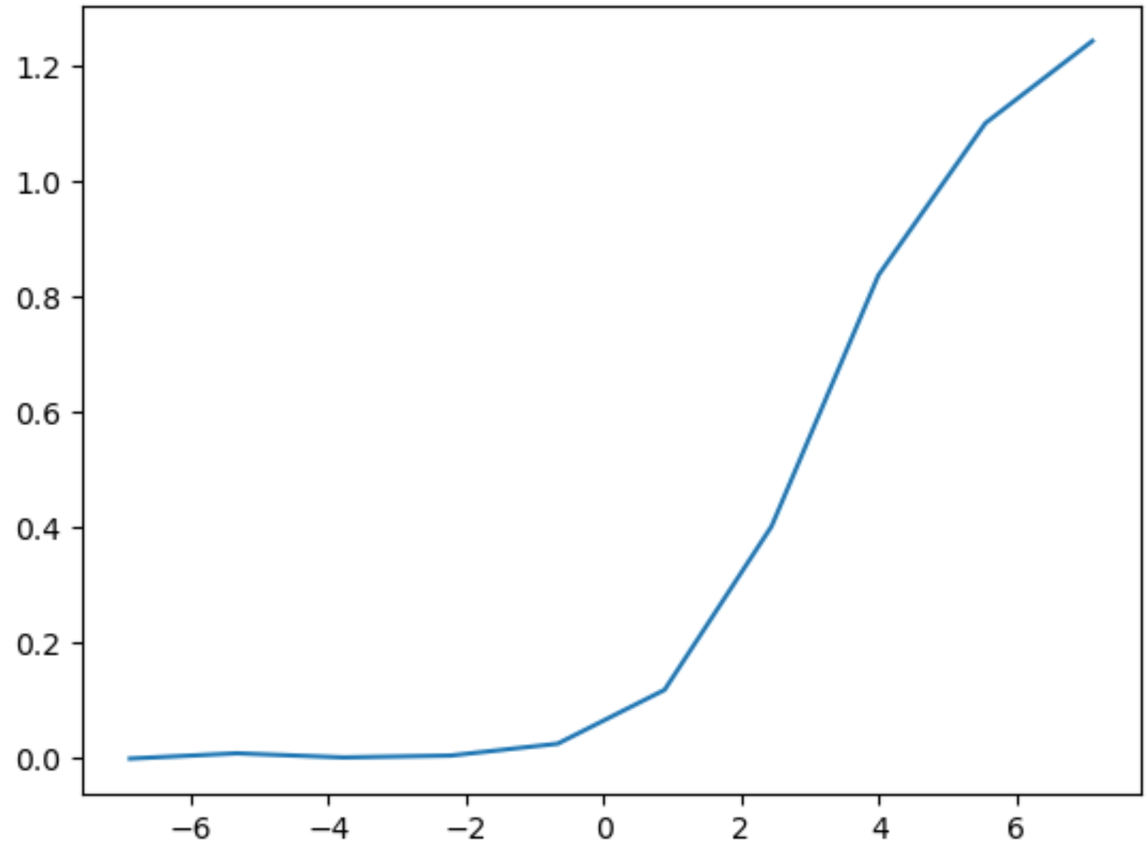
In [148]:
```
def compute_dot_products(u,X):
    ''' Dot product between filter u and all time windows in stimulus

    Args:
      X: matrix of stimulus over time (640D)
      u: linear filter.

    Return:
      all_dp: vector of inner products for all valid time windows'''

    all_dp = [0 for _ in range(u.shape[1])]
    window_size = u.shape[1]
    for i in range(window_size, len(X)):
        window = np.transpose(X[i-window_size:i,:])
        all_dp.append(np.dot(window.flatten(), u.flatten().T))
    return all_dp
```

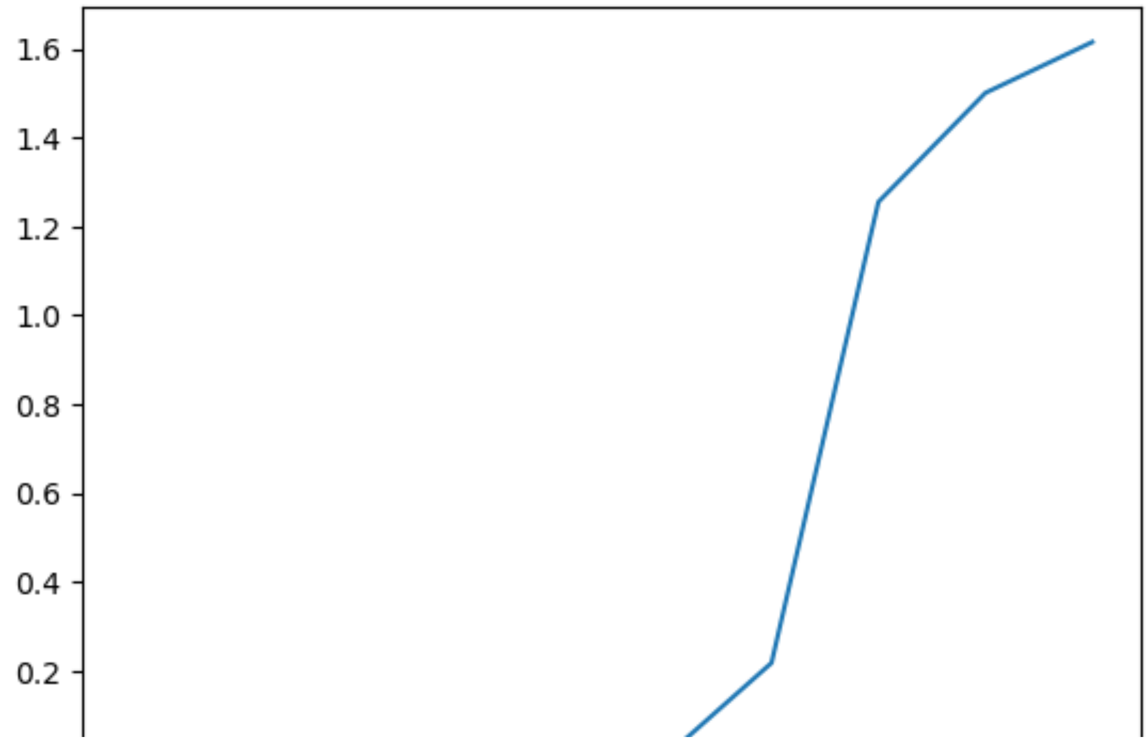
In [149]:
```
def compute_sta_bin_means(all_dp,n_bins,spikes):
    ''' Bin dot products of stimuli with STA. Compute mean fr within e

    Args:
      all_dp: vector of stimulus-filter dot products at every time poi
      n_bins: number of bins for dot product binning
      spikes: vector of spike count at every time point

    Return:
      m_sta: vector of mean firing rate along sta
      x_sta: values of sta bin centers, for plotting against m_sta'''

    bin_edges = np.histogram_bin_edges(all_dp, bins=n_bins)
    digitized_dps = np.digitize(all_dp, bins=bin_edges)
    binned_spikes = [spikes[digitized_dps == i] for i in range(1, len(
    m_sta = np.array([np.mean(spike_list) for spike_list in binned_spi
    x_sta = np.array([(bin_edges[i] + bin_edges[i + 1]) / 2 for i in r

    return m_sta, x_sta
```

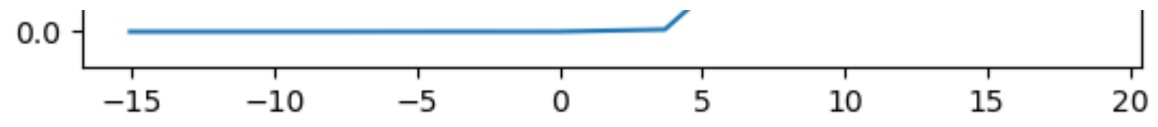
In [155]:
```
#Bring it all together to compute the non-linearity for each neuron
n_bins_sta = 10 #Just use 10 bins for the non-linearity

for i,sta in enumerate(neuron_stas):
    all_dp = compute_dot_products(sta, r)
    m_sta, x_sta = compute_sta_bin_means(all_dp, n_bins_sta, spikes[:,
    plt.plot(x_sta, m_sta)
    plt.show()
```




0.0
−2
0
2
4


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2



Bonus (20 pt)

What do you think the average response along axes in stimulus space that are orthogonal to the STA should be? Make the argument assuming the LN model is correct for how a neuron processes stimuli. Compute the average response ramps along axes that are orthogonal to the STA in stimulus space for each of the three neurons above. Do your findings match your predictions? Discuss.

Type *Markdown* and LaTeX: α^2

Linearity as a default model (40 pts)

One of the central questions in brain science is "How do neurons encode events in the real world?" And one of the most celebrated results in this area is a series of papers in the 1980s by A. P. Georgopoulos and colleagues. Here we will take a closer look at one of these and conclude that it is simply a rediscovery of the Taylor series for function approximation.

Read this article: Georgopoulos A, Schwartz A, Kettner R. Neuronal Population Coding of Movement Direction. Science. 1986;233: 1416–1419. doi:10.1126/science.3749885

The authors measure the firing rate d of a neuron in the monkey brain when the monkey moves its hand in the direction $\mathbf{M} = (m_x, m_y, m_z)$ from the starting point. They discover that for many neurons the firing can be approximated as

$$d(m_x, m_y, m_z) \approx b + b_x m_x + b_y m_y + b_z m_z \tag{1}$$

The equation above can also be written as

$$d(m_x, m_y, m_z) \approx b + k\,\mathbf{C} \cdot \mathbf{M} \tag{2}$$

where $\mathbf{M} = (m_x, m_y, m_z)$ is the motion direction, and $\mathbf{C} = (c_x, c_y, c_z)$ is a unit vector called the cell's "preferred direction".

In the second part the authors measure the activities $d^{(j)}$ of N different neurons, each with its own preferred direction vector $\mathbf{C}^{(j)}$. Then they take a weighted average, multiplying each neuron's preferred direction $\mathbf{C}^{(j)}$ by its response $(d^{(j)} - b^{(j)})$:

$$\mathbf{P}(\mathbf{M}) = \sum_{j}^{N} (d^{(j)}(\mathbf{M}) - b^{(j)})\mathbf{C}^{(j)} \tag{3}$$

They call this the "population vector". They discover that this vector points in the same direction as the reach vector $\mathbf{M}$, and thus encodes the motion direction.

First look (15 pts)

Not knowing anything about monkey brains, show that this result is equivalent to the first-order Taylor series approximation of $d(m_x, m_y, m_z)$, and that the linear relationship is guaranteed to be true for some range of motion excursions of the arm. (10 pts)

What other experiments do you think the author should have done? (5 pts)

To show that equation (1) is equivalent to a first-order Taylor series approximation of $d(m_x, m_y, m_z)$, we can perform a Taylor expansion on the equation about some arbitrary reference point, denoted (x, y, z), using the Taylor series:

$$d(m_x, m_y, m_z) = d(x, y, z) + \frac{\partial d}{\partial m_x} m_x + \frac{\partial d}{\partial m_y} m_y + \frac{\partial d}{\partial m_z} m_z + O(|\mathbf{M}|^2)$$

Let $b := d(x, y, z)$, then

$$d(m_x, m_y, m_z) \approx b + b_x m_x + b_y m_y + b_z m_z$$

where b_x , b_y , and b_z are the partial derivatives of d respectively with respect to m_x , m_y, and m_z evaluated at the point (x, y, z). Therefore, equation (1) is a first-order Taylor series approximation of $d(m_x, m_y, m_z)$.

The linear relationship is guaranteed to be true for some range of motion excursions of the arm because the monkey brain must encode motion direction using a finite number of neurons. Suppose the motion is too large or too complex, then the linearity in the relationship may break down to account for complexity. However, for small, simple motions, the relationship remains linear.

To further support the result, the authors could have performed the following experiments:

The authors could have repeated the experiments for different starting points, as opposed to having the monkey hold at the center in every experiment before introducing the random directions, checking if the population vector remains pointed in the same direction as the reach vector. This would show that the population vector is dependent on a particular start point. Additionally different types of arm movements could have been tested, such as grasping objects in different position, and checked if the linear relationship holds, showing the population vector represents the direction as opposed to representing the type of motion required to press the targets.

"Population vector" (10 pts)

Show that the collinearity of (3) and $\mathbf{M}$ follows directly from (2) as long as the preferred directions $\mathbf{C}^{(j)}$ are distributed evenly in space. The article won't tell you whether that condition is met, but you can find it in a later paper (*J Neurosci*. 8:2913 (1988)).

Bonus point (5 pt): Prove this using a symmetry argument requiring no equations.

Suppose that the N preferred direction vectors $\mathbf{C}^{(j)}$ are evenly distributed in space. The population vector $\mathbf{P}(\mathbf{M})$ is represented as the weighted sum of the $\mathbf{C}^{(j)}$, where the weight is represented by the change in cell activity from a baseline firing level as caused by the movement. The components of the preferred direction vector $\mathbf{C}^{(j)}$ are directly proportional to the corresponding component in the baseline vector. This means that the baseline firing level as represented by the vectors are evenly distributed in space. Thus the deviations from the baseline as a result of the rach vector are all evenly distributed. This means the transformation that occurs weighing each respective $\mathbf{C}^{(j)}$ by the deviation from the baseline represented by $d^{(j)}(\mathbf{M}) - b^{(j)}$ has the effect of simply changing the direction of the evenly distributed $\mathbf{C}^{(j)}$ to each be equally shifted by the vector $\mathbf{M}$ and scaled accordingly. Since this transformation shifts all the evenly distributed $\mathbf{C}^{(j)}$, then the property of being evenly distributed in space is maintained and the only change is the direction of the whole system to have a weighted sum facing the direction of $\mathbf{M}$. This means we have a symmetry for the system, and since the weighted sum corresponds with $\mathbf{P}(\mathbf{M})$, then the resultant vector from the transformation is $\mathbf{P}(\mathbf{M})$, showing that the scaled and shifted vector is collinear to $\mathbf{M}$.

One lesson from this exercise is that it helps to have some baseline expectation when researching a new system, so you can recognize something as unexpected. Approximately linear behavior is the baseline expectation for this system and in many other applications. (Although a pessimist might say that if the authors had recognized their result as being the default expectation for any system, they would have missed an opportunity to publish it in Science.) Indeed the vast majority of mathematical methods are based on some form of linear processing.

Finally, if you want to read something remarkable and unexpected in this series of articles try: Georgopoulos A, Lutiro J, Petrides M, Schwartz A, Massey J. Mental Rotation of the Neuronal Population Vector. Science. 1989;243: 234–236. doi:10.1126/science.2911737.

Biased estimator (15 pts)

Now assume the null hypothesis: The neural activity the authors recorded has nothing to do with the monkey's arm reaches. Each of the neurons produces a response to each of the 8 arm reaches that is a random number.

Show that the authors would still obtain their major result (3), namely that the population vector points in the direction of the movement vector. Explain how this arises. (10 pts)

If the neural activity of the recorded neurons is random and not related to the arm reaches, the weights in Equation (3) will be random and uncorrelated with the movement direction. In this case, the expected value of the population vector $\mathbf{P}(\mathbf{M})$ would be zero in all directions, since the random contributions to $\mathbf{P}(\mathbf{M})$ from different neurons will cancel out.

However, the authors measured the response of multiple neurons with different preferred directions, and the random fluctuations in each neuron's response will be uncorrelated with each other. Therefore, the direction in which the population vector points will be random, but it will not be zero in all directions. Due to the central limit theorem, the random fluctuations in each neuron's response will average out to zero, but the direction in which the population vector points will be determined by the neuron with the largest response. This neuron will dominate the sum in Equation (3), and the population vector will point in the direction of its preferred direction.

Therefore, even if the neural activity is random and uncorrelated with the movement direction, the authors would still observe that the population vector points in the direction of the movement vector, but the precision of the encoding would be much lower than if the neural activity was correlated with the movement direction.

This shows that the population vector is a biased estimator, and it is biased exactly towards the claim of the article. Suggest what the authors might have done to guard against this bias. (5 pts)

To guard against the bias towards the claim that the population vector points in the same direction as the reach vector, the authors could have performed a control experiment in which the monkey did not perform any arm reaches, but the same neurons were recorded while the monkey remained stationary. If the population vector points in a random direction in the stationary condition, then the observed bias towards the reach direction in the main experiment is likely due to the neural activity being correlated with the movement direction.

```
In [ ]:
```