bioqtm385-inclass-04-bayesian-inference-2024

October 28, 2024

##BIO/QTM 385: In class exercise for Wednesday, October 2nd

(answers will be the part of Assignment #3, Due 10/28)

Carol Zhou Collaborated with Aanya Vusirikala and Sweta Balaji. Generative AI is used for question 3, 7-10. Question 3 AI prompt: I copied and pasted the question into AI. Question 7-10: AI was used to make the graphs.

Aspects of this notebook have been adapted from the Neuromatch course materials here.

[2]:
```
#import useful libraries
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as random
import math
```

[3]:
```
#@title Figure Settings and helper functions
import ipywidgets as widgets
plt.style.use("https://raw.githubusercontent.com/NeuromatchAcademy/
 ↪course-content/NMA2020/nma.mplstyle")
%matplotlib inline
%config InlineBackend.figure_format = 'retina'


def my_plot_single(x, px):
    """
    Plots normalized Gaussian distribution

    Args:
        x (numpy array of floats):     points at which the likelihood has been␣
 ↪evaluated
        px (numpy array of floats):    normalized probabilities for prior␣
 ↪evaluated at each `x`

    Returns:
        Nothing.
    """
    if px is None:
        px = np.zeros_like(x)
    fig, ax = plt.subplots()
    ax.plot(x, px, '-', color='C2', LineWidth=2, label='Prior')
    ax.legend()
    ax.set_ylabel('Probability')
    ax.set_xlabel('Orientation (Degrees)')


def posterior_plot(x, likelihood=None, prior=None, posterior_pointwise=None,␣
 ↪ax=None):
    """
    Plots normalized Gaussian distributions and posterior

    Args:
        x (numpy array of floats):         points at which the likelihood has␣
 ↪been evaluated
        auditory (numpy array of floats):  normalized probabilities for␣
 ↪auditory likelihood evaluated at each `x`
        visual (numpy array of floats):    normalized probabilities for visual␣
 ↪likelihood evaluated at each `x`
        posterior (numpy array of floats): normalized probabilities for the␣
 ↪posterior evaluated at each `x`
        ax: Axis in which to plot. If None, create new axis.

    Returns:
        Nothing.
    """
    if likelihood is None:
        likelihood = np.zeros_like(x)

    if prior is None:
        prior = np.zeros_like(x)

    if posterior_pointwise is None:
        posterior_pointwise = np.zeros_like(x)

    if ax is None:
      fig, ax = plt.subplots()

    ax.plot(x, likelihood, '-C1', linewidth=2, label='Likeihood')
    ax.plot(x, prior, '-C0', linewidth=2, label='Prior')
    ax.plot(x, posterior_pointwise, '--C2', linewidth=2, label='Posterior')
    ax.legend()
    ax.set_ylabel('Probability')
    ax.set_xlabel('X')

    return ax
```

```
def plot_visual(mu_visuals, mu_posteriors, max_posteriors):
    """
    Plots the comparison of computing the mean of the posterior analytically and
    the max of the posterior empirically via multiplication.

    Args:
        mu_visuals (numpy array of floats): means of the visual likelihood
        mu_posteriors (numpy array of floats):  means of the posterior,␣
  ↪calculated analytically
        max_posteriors (numpy array of floats): max of the posteriors,␣
  ↪calculated via maxing the max_posteriors.
        posterior (numpy array of floats): normalized probabilities for the␣
  ↪posterior evaluated at each `x`

    Returns:
        Nothing.
    """
    fig_w, fig_h = plt.rcParams.get('figure.figsize')
    fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(fig_w, 2 * fig_h))

    ax[0].plot(mu_visuals, max_posteriors, '-C2', label='mean')
    ax[0].set_xlabel('Visual stimulus position')
    ax[0].set_ylabel('Multiplied posterior mean')
    ax[0].set_title('Sample output')

    ax[1].plot(mu_visuals, mu_posteriors, '--', color='xkcd:gray',␣
  ↪label='argmax')
    ax[1].set_xlabel('Visual stimulus position')
    ax[1].set_ylabel('Analytical posterior mean')
    fig.tight_layout()
    ax[1].set_title('Hurray for math!')


def multimodal_plot(x, example_prior, example_likelihood,
                    mu_visuals, posterior_modes):
  """Helper function for plotting Section 4 results"""

  fig_w, fig_h = plt.rcParams.get('figure.figsize')
  fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(fig_w, 2*fig_h),␣
  ↪sharex=True)

  # Plot the last instance that we tried.
  posterior_plot(x,
    example_prior,
    example_likelihood,
    compute_posterior_pointwise(example_prior, example_likelihood),
    ax=ax[0]
    )
  ax[0].set_title('Example combination')

  ax[1].plot(mu_visuals, posterior_modes, '-C2', label='argmax')
  ax[1].set_xlabel('Visual stimulus position\n(Mean of blue dist. above)')
  ax[1].set_ylabel('Posterior mode\n(Peak of green dist. above)')
  fig.tight_layout()
```

##The Gaussian Distribution

Bayesian analysis operates within the world of probability distributions. Although these distributions can take many forms, the Gaussian distribution is a very common choice. Because of the central limit theorem, many quantities are Gaussian-distributed. Gaussians also have some mathematical properties that permit simple closed-form solutions to several important problems.

The equation for a Gaussian is:

$$N(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)$$

Question #1: Write a function `y = my_gaussian(x,mu,sigma)`, where `x` is a 1-d array of values, `mu` is the mean of the distribution, `sigma` is the standard deviation, and `y` is the value of the prescribed Gaussian distribution at each value of `x`.

[4]:
```
def my_gaussian(x, mu, sigma):
  return 1/np.sqrt(2*np.pi*sigma**2)*np.exp(-(x-mu)**2/(2*sigma**2))

print(my_gaussian(np.array([1,2,3]),1,2))
```

```
[0.19947114 0.17603266 0.12098536]
```

##Bayes' Theorem and the Gaussian Distribution

Bayes' rule tells us how to combine two sources of information: the prior (e.g., a noisy representation of our expectations about where the stimulus might come from) and the likelihood (e.g., a noisy representation of the stimulus position on a given trial), to obtain a posterior distribution taking into account both pieces of information. Bayes' rule states:

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Normalization constant}} \tag{1}$$

or, more explicitly:

$$p(\text{Model}|\text{Data}) = \frac{p(\text{Data}|\text{Model}) \times p(\text{Model})}{p(\text{Data})} \tag{2}$$

If N(μ, σ^2) denotes a Gaussian distribution with parameters μ and σ^2, then if both the prior and likelihood are Gaussians, this translates into the following form:

$$\begin{aligned} \text{Likelihood} &= N(\mu_{likelihood}, \sigma^2_{likelihood}) \\ \text{Prior} &= N(\mu_{prior}, \sigma^2_{prior}) \\ \text{Posterior} &= \frac{N(\mu_{likelihood}, \sigma^2_{likelihood}) \times N(\mu_{prior}, \sigma^2_{prior})}{p(\text{Data})} \end{aligned}$$

It turns out that for multiplying two Gaussians, this can be re-written as:

$$= N\left(\frac{\sigma^2_{likelihood}\mu_{prior} + \sigma^2_{prior}\mu_{likelihood}}{\sigma^2_{likelihood} + \sigma^2_{prior}}, \frac{\sigma^2_{likelihood}\sigma^2_{prior}}{\sigma^2_{likelihood} + \sigma^2_{prior}}\right) .$$

Question #2: Write a function, `posterior = calculateGaussianPosterior(x,mu_likelihood,sigma_likeliho` that computes the posterior distribution for the data points `x`.

```
[5]: def␣
      ↪calculateGaussianPosterior(x,mu_likelihood,sigma_likelihood,mu_prior,sigma_prior):
      ↪
       return my_gaussian(x,␣
      ↪((sigma_likelihood**2)*mu_prior+(sigma_prior**2)*mu_likelihood)/
      ↪(sigma_likelihood**2+sigma_prior**2), (sigma_likelihood**2*sigma_prior**2)/
      ↪(sigma_likelihood**2+sigma_prior**2))

     print(calculateGaussianPosterior(np.array([1,2,3]),1,2,3,4))
```

```
[0.12369928 0.12249716 0.11002041]
```

Question #3: Write code to check that the distribution you generate from the function in Question #2 is normalized (i.e., $\int_{-\infty}^{\infty} \text{Posterior}(x) = 1$).

```
[6]: from scipy import integrate

     print(integrate.quad(calculateGaussianPosterior, -np.inf, np.inf,␣
      ↪args=(1,2,3,4)))
```

```
(1.0, 1.5662233501278158e-09)
```

0.1 Interactive Demo: What affects the posterior?

Now that we can compute the posterior of two Gaussians with *Bayes rule*, let's vary the parameters of those Gaussians to see how changing the prior and likelihood affect the posterior.

Hit the Play button or Ctrl+Enter in the cell below and play with the sliders to get an intuition for how the means and standard deviations of prior and likelihood influence the posterior.

```
[7]: #@title
     #@markdown Make sure you execute this cell to enable the widget!

     x = np.arange(-10, 11, 0.1)
```

```
import ipywidgets as widgets

def refresh(mu_auditory=3, sigma_auditory=1.5, mu_visual=-1, sigma_visual=1.5):
    auditory = my_gaussian(x, mu_auditory, sigma_auditory)
    visual = my_gaussian(x, mu_visual, sigma_visual)
    posterior_pointwise = visual * auditory
    posterior_pointwise /= (posterior_pointwise.sum()*(x[1]-x[0]))

    w_auditory = (sigma_visual** 2) / (sigma_auditory**2 + sigma_visual**2)
    theoretical_prediction = mu_auditory * w_auditory + mu_visual * (1 -␣
 ↪w_auditory)

    ax = posterior_plot(x, auditory, visual, posterior_pointwise)
    ax.plot([theoretical_prediction, theoretical_prediction],
        [0, posterior_pointwise.max() * 1.2], '-.', color='xkcd:medium gray',␣
 ↪linewidth=2)
    ax.set_title(f"Gray line shows analytical mean of posterior:␣
 ↪{theoretical_prediction:0.2f}")
    plt.show()

style = {'description_width': 'initial'}

_ = widgets.interact(refresh,
    mu_auditory=widgets.FloatSlider(value=2, min=-10, max=10, step=0.5,␣
 ↪description="mu_likelihood:", style=style),
    sigma_auditory=widgets.FloatSlider(value=1, min=0.5, max=10, step=0.5,␣
 ↪description="sigma_likelihood:", style=style),
    mu_visual=widgets.FloatSlider(value=-2, min=-10, max=10, step=0.5,␣
 ↪description="mu_prior:", style=style),
    sigma_visual=widgets.FloatSlider(value=1, min=0.5, max=10, step=0.5,␣
 ↪description="sigma_prior:", style=style)
)
```

```
interactive(children=(FloatSlider(value=2.0, description='mu_likelihood:',␣
 ↪max=10.0, min=-10.0, step=0.5, styl…
```

Question #4: Starting from $\sigma_{\text{likelihood}} = \sigma_{\text{prior}} = 1$, $\mu_{\text{likelihood}} = 2$, and $\mu_{\text{prior}} = -2$, what happens to the posterior distribution as σ_{prior} increases? Explain why you think that your observed effect is occuring.

As sigma_prior (variance in prior) increases, the prior distribution becomes wider, meaning there is more uncertainty in your prior. Because of this uncertainty, the posterior now depends more on the likelihood. Thus, as sigma_prior increases, the posterior also becomes more similar to the likelihood distribution and deviates from the true x (becomes more biased).

Question #5: Find a combination of parameters where the posterior distribution is visually indistinguishable from the prior distribution. Explain why this combination of parameters results in little or no information from the likelihood being incorporated into the posterior.

When mu_likelihood = 2, sigma_likelihood = 10, mu_prior = -2, and sigma_prior = 1, the posterior distribution is visually indistinguishable form the prior distribution. Since there is a big variance/uncertainty in the likelihood, indicating the new data is less realiable, bayesian inference will base the posterior more on the prior, which has less variance. This why the posterior distribution becomes more similar to the prior distrbution and that little or no information from the likelihood is being incorporated into the posterior.

##Example: Neuronal tuning curves

As described in lecture last week, many neurons can be modeled as having *receptive fields* - meaning that they have different firing properties in response to different stimuli. There are many examples of this, from vision to hearing to direction to location, and their discovery has been feted with two different Nobel Prizes for Physiology and Medicine (1981 and 2014). Thinking about (simplified) versions of these cells are a great example of Bayesian inference, as an animal needs to incorporate the new information provided about its surrounding environment that these cells provide (the likelihood) into its existing world view (the prior).

For today, we will focus on head direction cells. These are neurons that fire when an animal is facing in a particular direction (e.g., north or east northeast). Specifically, we will model these cells having a peaked *firing rate* curve around an angle, θ_k. That is, when the animal is pointed in direction θ_k, the neuron will be most likely to fire, and this rate will decrease as one turns away from this direction. We will call this curve $f_k(\theta)$.

For simplicity, we would typically want to model $f_k(\theta)$ as a 1-dimensional Gaussian, but the fact that our data is on a circle means that we need $f_k(\theta) = f_k(\theta + 2\pi)$, which is not true for a Gaussian. Thus, we will use the "Gaussian on a circle" - more officially known as the von Mises distribtuion:

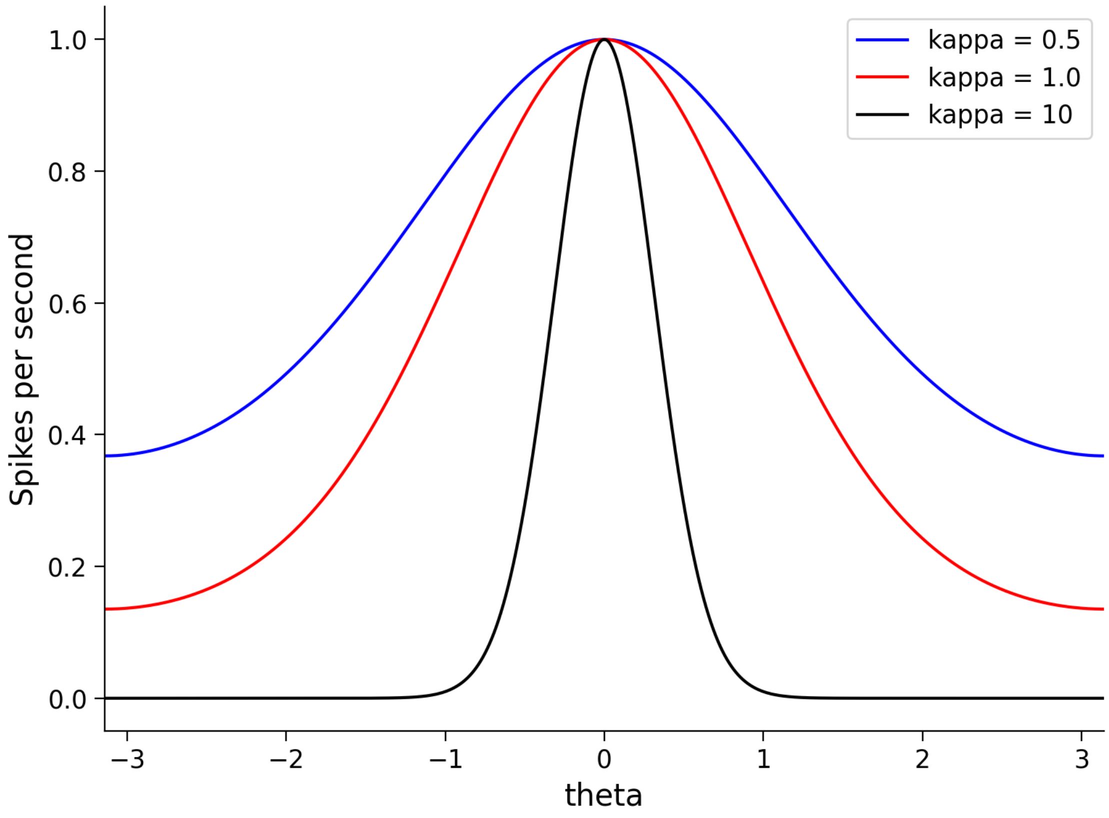
$$f_k(\theta) = r_{max} e^{\kappa \cos(\theta - \theta_k) - \kappa}, \tag{3}$$

where κ is a measure of how peaked the distribution is (think: something akin to the inverse of σ for a Gaussian), and r_{max} is the maximum firing rate.

The following code defines `tuningCruve(theta,thetak,kappa,rmax)` (parameter names are in argreement with the equation above) that computes the von Mises tuning curve above for an array of values (`theta`). Note plotted curves widen as κ decreases.

[8]:
```python
def tuningCurve(theta,thetak=0,kappa=1,rmax=10):
    return rmax*np.exp(kappa*np.cos(theta-thetak)-kappa)

xx = np.arange(-np.pi,np.pi,.001)

fig, ax = plt.subplots()
thetak = 0
rmax = 1
kappas = [.5, 1, 10]
plt.plot(xx,tuningCurve(xx,thetak,kappas[0],rmax),'b-')
plt.plot(xx,tuningCurve(xx,thetak,kappas[1],rmax),'r-')
plt.plot(xx,tuningCurve(xx,thetak,kappas[2],rmax),'k-')

ax.legend(['kappa = 0.5','kappa = 1.0','kappa = 10'])
```

```
plt.xlabel('theta')
plt.ylabel('Spikes per second')
plt.xlim([-np.pi,np.pi])

plt.show()
```



###Poisson Processes

While these curves represent the average number of spike/second a single neuron would provide in a time window, in any finite measuring window, we are likely to measure a slightly different rate due to random noise. For today, we will model the arrival of spikes as being given by a *Poisson process*. In this process, we assume that for any time interval, only the mean firing rate ($f_k(\theta)$) and the interval length (Δt) are needed to compute the expected number of spikes. This implies that there are no weird complications like refractory periods or other types of time-dependency.

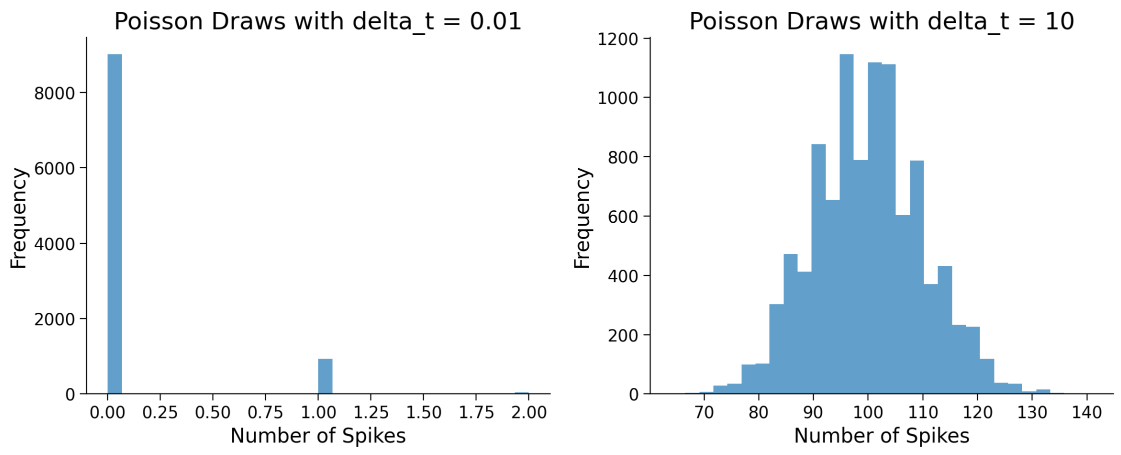
Given that there is a Poisson process, the probability of observing n spikes in time interval Δt from neuron k, when the animal is facing in direction θ is:

$$p(\text{number of spikes} = n|\theta) = \frac{(f_k(\theta)\Delta t)^{n\Delta t}}{(n\Delta t)!}e^{-f_k(\theta)\Delta t}. \tag{4}$$

Question #6: Write a function (`returnPoissonDraws(deltaT)`) that returns 10,000 random draws from the above Poisson distribution at $\theta = \theta_k$ as a function of Δt (you can assume $\kappa = r_{max} = 10$).

Histogram these draws for $\Delta t = .01$ and $\Delta t = 10$. Describe and analyze the differences between the found distributions. (Note: `random.poisson()` will be helpful)

```
[9]: def returnPoissonDraws(deltaT):
       firing_rate = tuningCurve(0, thetak=0, kappa=10, rmax=10)
       return random.poisson(firing_rate*deltaT, 10000)

     fig, axs = plt.subplots(1, 2, figsize=(12, 5))

     axs[0].hist(returnPoissonDraws(0.01), bins=30, alpha=0.7)
     axs[0].set_title('Poisson Draws with delta_t = 0.01')
     axs[0].set_xlabel('Number of Spikes')
     axs[0].set_ylabel('Frequency')

     axs[1].hist(returnPoissonDraws(10), bins=30, alpha=0.7)
     axs[1].set_title('Poisson Draws with delta_t = 10')
     axs[1].set_xlabel('Number of Spikes')
     axs[1].set_ylabel('Frequency')

     plt.tight_layout()
     plt.show()
```



Describe and analyze the differences between the found distributions. When delta_t = 0.01, there is a peaked distribution around number of spikes = 0.1 because we multiply the firing rate by 0.01 (10x0.01=0.1), leading to relatively low observed number of spikes. There is also a small peak at number of spikes = 1 because there is still some probability of seeing one spike according to the poisson distribution. When delta_t = 10, the distribution appears more normal-like with the expected number of spikes around 100 (10x10=100). This is because the probability of observing other number of spikes

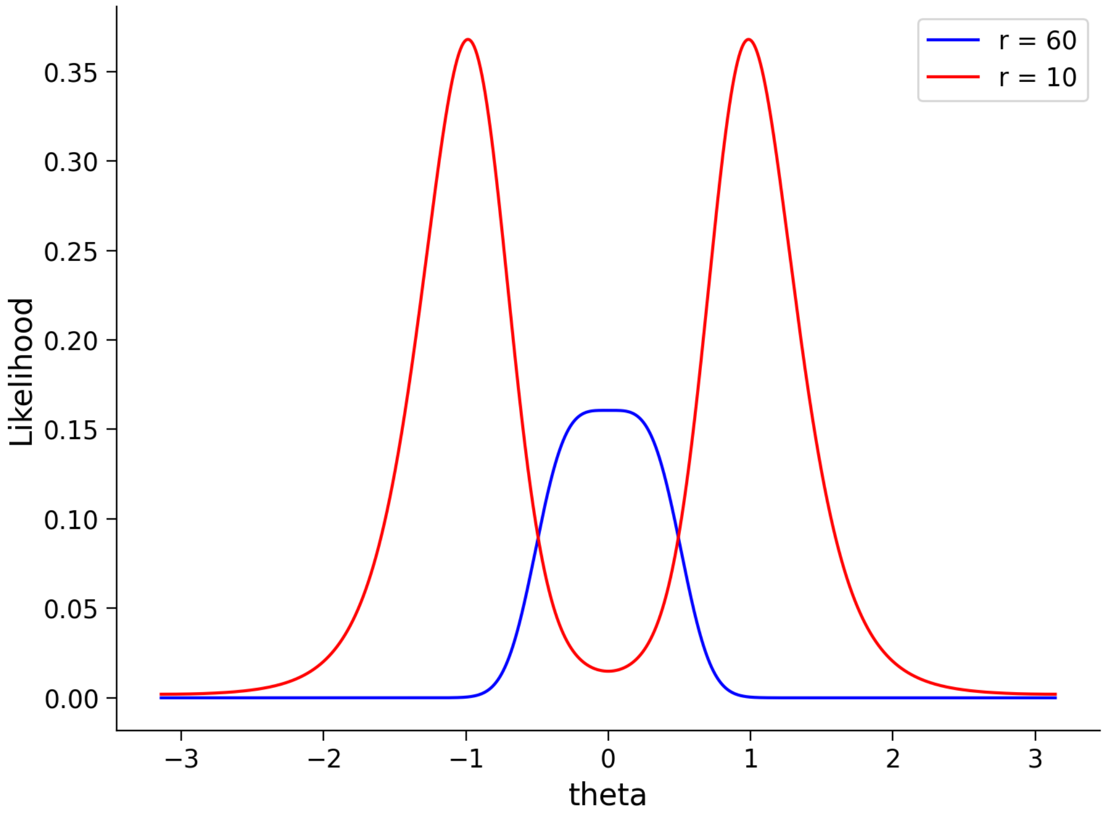
##Calculating the Likelihood

Thus, given an equation for $p(\text{number of spikes} = n|\theta)$ in Δt, we can now write $p(\text{rate} = r|\theta) =$

$\frac{(f_k(\theta)\Delta t)^{r\Delta t}}{(r\Delta t)!}e^{-f_k(\theta)\Delta t}$, where r is the number of spikes observed in Δt, divided by Δt.

Question #7: Write a function to calculate the likelihood of observing firing rate r across θ from $-\pi$ to π (i.e., $p(r|\theta)$). Assume that $\Delta t = 0.1$, $\theta_k = 0$, $r_{max} = 60$, and $\kappa = 4$. Note: you can implement $x!$ in python as `math.gamma(x+1)`. Plot curves for $r = 60$ and $r = 10$ (using `thetas = np.arange(-np.pi,np.pi,.001)`). Describe why you are getting a much different looking result for the two curves.

[10]:
```python
def calculateLikelihood(theta,rate,theta_k,kappa,deltaT,rmax):
    return (tuningCurve(theta,theta_k,kappa,rmax)*deltaT)**(rate*deltaT)/math.
 ↪gamma(rate*deltaT+1)*np.exp(-tuningCurve(theta,theta_k,kappa,rmax)*deltaT)

thetas = np.arange(-np.pi, np.pi, .001)
fig, ax = plt.subplots()
plt.plot(thetas, calculateLikelihood(thetas, 60, 0, 4, 0.1, 60), 'b-')
plt.plot(thetas, calculateLikelihood(thetas, 10, 0, 4, 0.1, 60), 'r-')
ax.legend(['r = 60', 'r = 10'])
plt.xlabel('theta')
plt.ylabel('Likelihood')
plt.show()
```



Describe why you are getting a much different looking result for the two curves. I get a much different result for the two curves because when r = 10, the variance of how that firing rate can be represented is greater. There are multiple configurations of the stimulus direction (angles) that could lead to the observed rate due to the von Mises distribution. That is why the likelihood is affected and there are two peaks: one at theta = -1 and the other at theta = 1. While when r = 60, the likelihood is more centered around the maximum firing direction (theta = 0). This is because when you observe many spikes (60 in this case) over the same period of time, it is likely that most of them came from the most probable firing direction.

##Calculating the Posterior

Question #8: Given the equation for $p(r|\theta)$, if we assume that the prior distribution, $p(\theta)$, is given by another vonMises distribution,

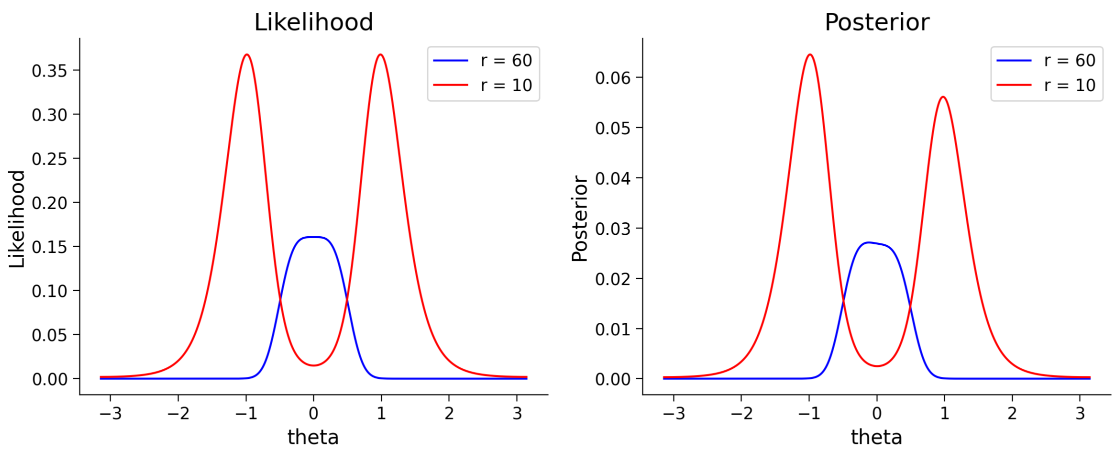
$$p(\theta) = \frac{1}{2\pi I_0(\kappa_p)} e^{\kappa_p \cos(\theta - \theta_p)}, \tag{5}$$

write a function `calculatePosterior(theta,rate,theta_k,kappa,deltaT,rmax,theta_prior,kappa_prior)` that returns the value of the posterior function for an array of `theta` values (don't worry about normalizing - just multiply the likelihood with the prior). Note that $I_0(\kappa_p)$ is the 0th order modified Bessel function - it is just needed for normalization, so don't worry if you haven't seen it before – and it can be implemented in python via `np.i0(kappa_prior)`.

[11]:
```
def␣
↪calculatePosterior(theta,rate,theta_k,kappa,deltaT,rmax,theta_prior,kappa_prior):
↪
    return calculateLikelihood(theta,rate,theta_k,kappa,deltaT,rmax)*np.
↪exp(kappa_prior*np.cos(theta-theta_prior))/(2*np.pi*np.i0(kappa_prior))
```

Question #9: Using the same parameters as in Question #7, plot the posterior for $\kappa_p = 0.1$ and $\theta_p = -1$ for both $r = 60$ and $r = 10$. How do these results differ from your likelihood plots? Why?

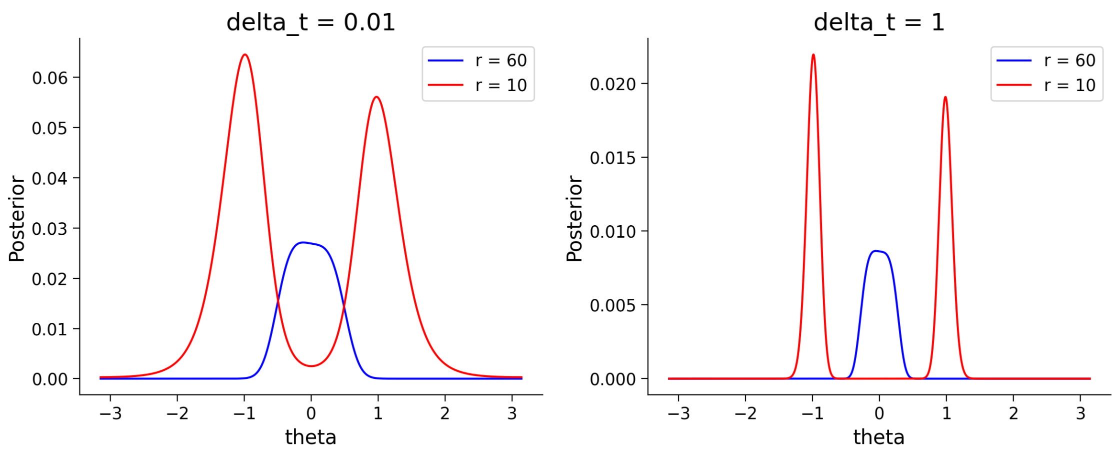
[12]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.plot(thetas, calculateLikelihood(thetas, 60, 0, 4, 0.1, 60), 'b-')
ax1.plot(thetas, calculateLikelihood(thetas, 10, 0, 4, 0.1, 60), 'r-')
ax1.legend(['r = 60', 'r = 10'])
ax1.set_title('Likelihood')
ax1.set_xlabel('theta')
ax1.set_ylabel('Likelihood')

ax2.plot(thetas, calculatePosterior(thetas, 60, 0, 4, 0.1, 60, -1, 0.1), 'b-')
ax2.plot(thetas, calculatePosterior(thetas, 10, 0, 4, 0.1, 60, -1, 0.1), 'r-')
ax2.legend(['r = 60', 'r = 10'])
ax2.set_title('Posterior')
ax2.set_xlabel('theta')
ax2.set_ylabel('Posterior')

plt.tight_layout()
plt.show()
```



How do these results differ from your likelihood plots? Why? The shape of the curve for r = 60 doesn't change much from the likelihood (though the top of the peak is more skewed to -1 probably because theta_prior = -1). This could be because the likelihood is already sharply peaked around theta = 0. Thus, the prior doesn't contribute significantly, and the likelihood dominates. However, for r = 10, the peak at theta = -1 has higher posterior than the peak at theta = 1. This is because the prior plays a more significant role in shaping the posterior (since the likelihood has more uncertainty). Given that our theta_prior = -1, it boosts the posterior likelihood at theta = -1 and decreases the likelihood at theta = 1.

Question #10: What happens to the posterior distributions if you increase Δt to be equal to 1 (keeping all other parameters constant). Describe why this is happening.

[13]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.plot(thetas, calculatePosterior(thetas, 60, 0, 4, 0.1, 60, -1, 0.1), 'b-')
ax1.plot(thetas, calculatePosterior(thetas, 10, 0, 4, 0.1, 60, -1, 0.1), 'r-')
ax1.legend(['r = 60', 'r = 10'])
ax1.set_title('delta_t = 0.01')
ax1.set_xlabel('theta')
ax1.set_ylabel('Posterior')

ax2.plot(thetas, calculatePosterior(thetas, 60, 0, 4, 1, 60, -1, 0.1), 'b-')
ax2.plot(thetas, calculatePosterior(thetas, 10, 0, 4, 1, 60, -1, 0.1), 'r-')
ax2.legend(['r = 60', 'r = 10'])
ax2.set_title('delta_t = 1')
ax2.set_xlabel('theta')
ax2.set_ylabel('Posterior')

plt.tight_layout()
plt.show()
```



Describe why this is happening. The r = 60 curve becomes narrower. This is because with delta_t = 1 and firing rate at 60 spikes per second, the expected number of spikes are 60, causing the likelihood will peak more sharply around r = 60. Thus, the Poisson distribution becomes tighter when the expected number of spikes increases. The posterior is still heavily influenced by the likelihood because there is a significant amount of data (60 spikes) supporting the neuron's peak firing direction. The same thing happens to the two peaks of the r = 10 curve. Both of the peaks become narrower because when delta_t is small (0.01), the influence of the tuning curve is more localized, leading to a broader likelihood distribution. When delta_t increases, the integration over time becomes less sensitive to small variations, leading to a tighter likelihood distribution.

Instructions on how to save PDF files: (repeated from last time)

To make nice looking files to hand-in (which makes grading much easier!), please follow the instructions below.

First, run this code to install texlive and to link your google drive (it will ask your permission to do so)

[1]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
 ↪texlive-plain-generic pandoc

from google.colab import drive
drive.mount('/content/drive')
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
The following additional packages will be installed:
  dvisvgm fonts-droid-fallback fonts-lato fonts-lmodern fonts-noto-mono
  fonts-texgyre fonts-urw-base35 libapache-pom-java
  libcmark-gfm-extensions0.29.0.gfm.3 libcmark-gfm0.29.0.gfm.3
  libcommons-logging-java libcommons-parent-java libfontbox-java libfontenc1
  libgs9 libgs9-common libidn12 libijs-0.35 libjbig2dec0 libkpathsea6
```