In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
np.random.seed(123)
```

Comparing two estimators.

A common method of estimating the size of biological populations is the "capture-mark-recapture" method. One proceeds by repeatedly capturing animals, putting a marker on them, and releasing them. After marking M animals, you then capture a new group of C animals and find that K of them are tagged with your mark. We will treat each of the C 2nd-round captures as independent of the other 2nd-round captures, i.e., as if you caught each animal and then re-released it immediately ("sampling with replacement"). If the full population size is N, then the proportion of the population that is marked is M/N and thus the expected value of the proportion marked in your 2nd sample (which is K/C) should be the same as the population proportion. Thus, an estimate of the population size is $\hat{N} = MC/K$.

In [2]:
```
def estimator(K, C, M):
    return M * C / K
```

---

a)

Check whether $\hat{N}$ is a maximum likelihood estimator. For a few triplets K, C, M, plot the likelihood $L(N) = p(K|C, M, N)$ for a range of values of N (e.g., from $\hat{N} - 5$ to $\hat{N} + 5$. For example, try $K, C, M = 10, 200, 4000$. Note that N has to be an integer, so note whether $\hat{N}$ should be rounded or truncated to the nearest lower integer to maximize likelihood, or whether your results suggest a consistent pattern with regard to the non-integer $\hat{N}$.

The tagged proportion of the population is M/N, which is also the probability that a re-captured animal is marked. Let $\theta = M/N$. Then the probabiliy of K tags among C re-captured animals is given by Binomial distribution describing the probability of K successes given C binary trials with independent probability of success $\theta = M/N$:

$$\text{Binomial}(K|C, \theta) = \binom{C}{K}(\frac{M}{N})^K(1 - \frac{M}{N})^{C-K}.$$

In [3]:
```
from scipy.special import comb

def likelihood(K, C, M, N):
```

```
    theta = M / N
    return comb(C, K) * (theta)**K * (1 - theta)**(C - K)
```

First let K=10, C=200, M=4000.

In [4]:
```
estimator(K=10, C=200, M=4000)
```

Out[4]: 80000.0

We get an estimate N=80000. What if K, C and M don't give a whole number N?

In [5]:
```
estimator(50, 97, 4968)
```

Out[5]: 9637.92

K=50, C=97, M=4968 give N=9637.92. We can compute the likelihood of N=9637 and 9638 to determine which is more likely to explain K.

In [6]:
```
likelihood(50, 97, 4968, 9637) < likelihood(50, 97, 4968, 9638)
```

Out[6]: True

N = 9638 was the better estimate, presumably since it's closer to the true estimate 9637.92.

Now let's try an N with a fractional value less than 0.5 and see if we need to round down. Let K=50, C=103, M=158.

In [7]:
```
estimator(50, 103, 158)
```

Out[7]: 325.48

Now the estimate N=325.48. Which whole number N, 325 or 326, is more likely?

In [8]:
```
likelihood(50, 103, 158, 325) > likelihood(50, 103, 158, 326)
```

Out[8]: True

325 is more likely since 325.48 is closer to 325 than 326.

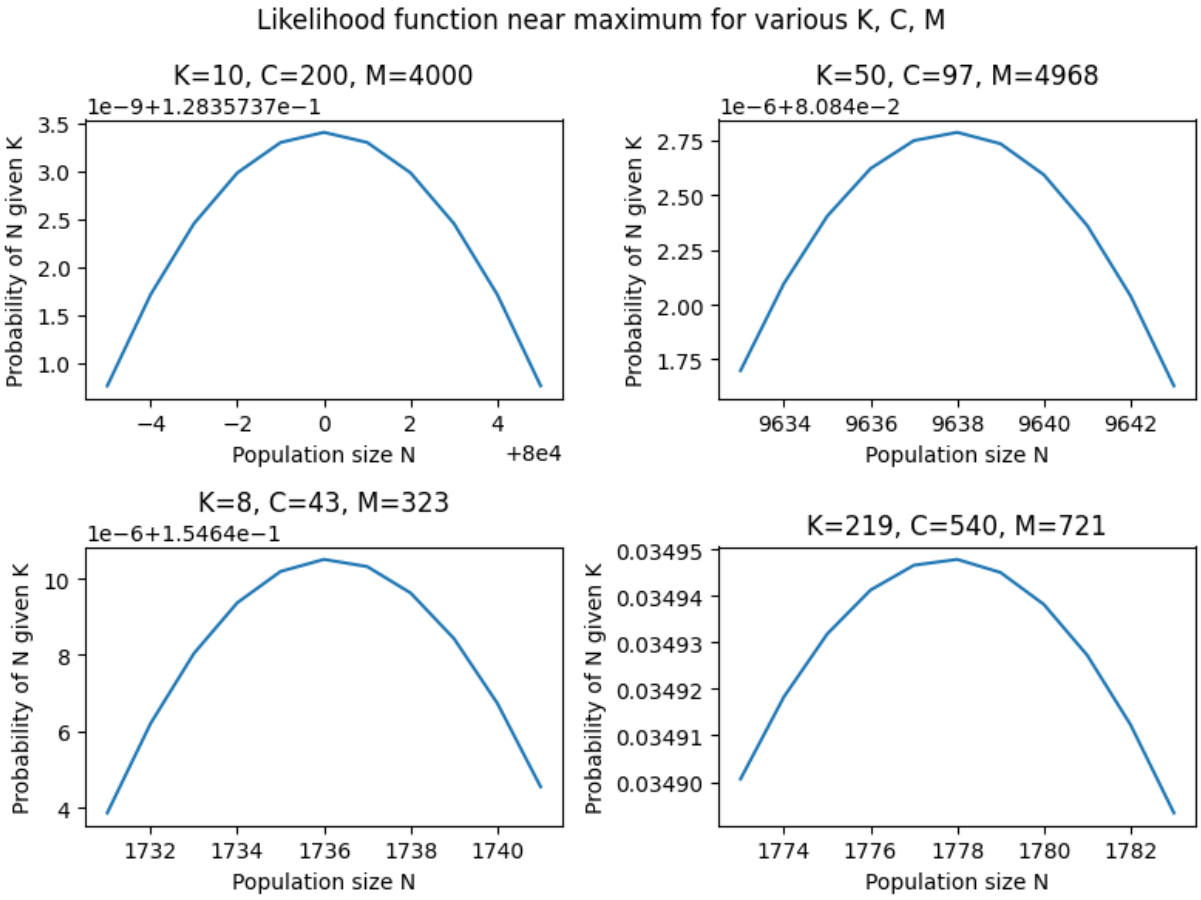
These results suggest a decision rule: when the MLE $\hat{N}$ is a fraction round it up or down to the nearest integer to get the population size estimate.

In [9]:
```
def plot_mle(K, C, M):
    N = np.round(estimator(K=K, C=C, M=M))
    x = np.arange(N - 5, N + 6)
    y = [likelihood(K, C, M, n) for n in x]
    plt.plot(x, y)
    plt.title(f'K={K}, C={C}, M={M}')
```

```
    plt.xlabel('Population size N')
    plt.ylabel('Probability of N given K')
```

Likelihood functions for four different sets of K, C, M are shown below.

In [10]:
```
fig, axs = plt.subplots(2, 2, figsize=(8, 6))
plt.sca(axs.flat[0])
plot_mle(K=10, C=200, M=4000)
plt.sca(axs.flat[1])
plot_mle(K=50, C=97, M=4968)
plt.sca(axs.flat[2])
plot_mle(K=8, C=43, M=323)
plt.sca(axs.flat[3])
plot_mle(K=219, C=540, M=721)
plt.suptitle('Likelihood function near maximum for various K, C, M')
plt.tight_layout()
plt.show()
```



---

b)

Check whether the estimator is unbiased. For a few triplets C, M, N, simulate the 2nd capture value K 1,000 times (i.e., generate appropriately distributed samples of K), and

compute the population estimate from each. Is the mean of the population estimates close to the correct answer?

In [11]:
```
def simulate(C, M, N):
    population = np.zeros(N)
    population[:M] = 1
    caught = np.random.choice(population, C, replace=True)
    K = np.sum(caught)
    return K
```

In [12]:
```
def mean_population_estimate(C, M, N):
    Ks = []
    for i in range(1000):
        K = simulate(C, M, N)
        Ks.append(K)
    Ns = [estimator(K, C, M) for K in Ks]
    return np.mean(Ns)
```

In [13]:
```
mean_population_estimate(C=200, M=4000, N=80000)
```

Out[13]: 90398.72719643617

For C=200, M=4000, N=80000, 1000 simulations usually give a mean population estimate around 90000. This more than 10% off from the true population size of 80000, indicating a bias to overestimate.

In [14]:
```
mean_population_estimate(C=97, M=4968, N=9638)
```

Out[14]: 9798.719854357263

C=97, M=4968, N=9638 usually gives mean population estimates above 9700, exceeding 9638. The estimator is biased to overestimate N.

In [15]:
```
mean_population_estimate(C=43, M=323, N=1736)
```

Out[15]: 1997.4581978132978

For C=43, M=323, N=1736 the estimator overestimates N as well.

---

c)

Write the precise distribution for samples K (when C, M, N are known and fixed), so you can compute the exact probability distribution of estimates $\hat{N}$. Do that for $N = 1000$, $M = 100$, $C = 100$. Plot the distribution and compute its mean and standard deviation. Does this calculation indicate that the estimator is biased? (Note: do this for the non-integer value of $\hat{N}$).

K is a random variable drawn from the distribution $\text{Binomial}(K|C, \theta)$ where $\theta = M/N$:

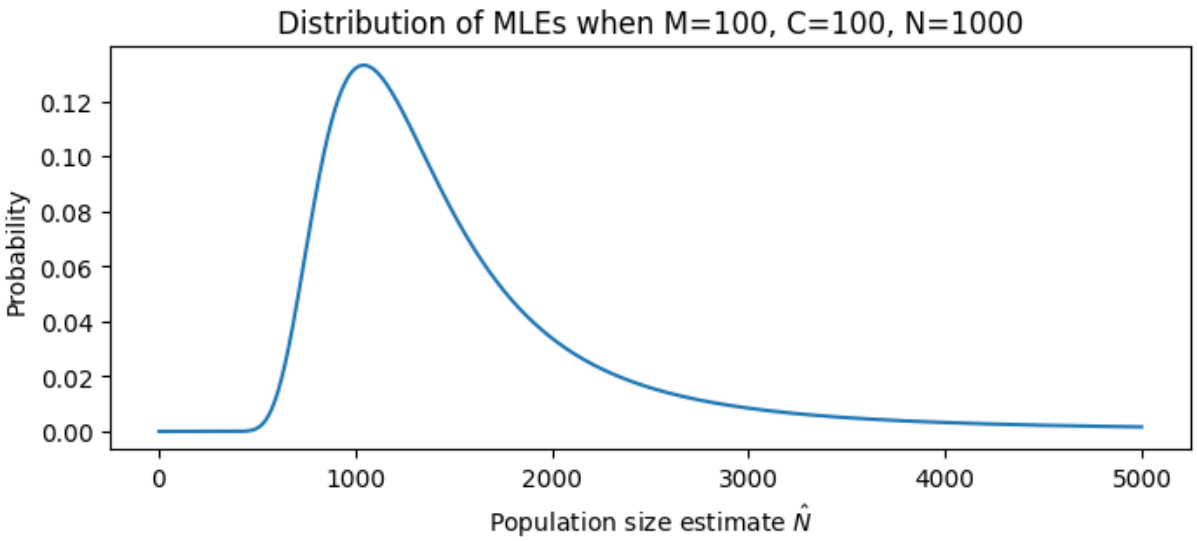
$P(K = k|C, M, N) = \binom{C}{k}(\frac{M}{N})^k(1 - \frac{M}{N})^{C-k}$.

Now, random variable $\hat{N}$ is defined as $\hat{N} = MC/K$. To find the probability distribution of $\hat{N}$, we need to find $P(\hat{N} = n)$, which is the probability that $\frac{MC}{k} = n$. This occurs when $k = \frac{MC}{n}$.

So $P(\hat{N} = n) = P(K = \frac{MC}{n}) = \binom{C}{\frac{MC}{n}}(\frac{M}{N})^{\frac{MC}{n}}(1 - \frac{M}{N})^{C-\frac{MC}{n}}$.

In [16]:
```
def mle_probability(n, N, M, C):
    theta = M / N
    k = M * C / n
    return comb(C, k) * (theta)**k * (1 - theta)**(C - k)
```

In [17]:
```
x = np.arange(2, 5000)
y = [mle_probability(n=n, C=100, M=100, N=1000) for n in x]
plt.subplots(figsize=(8, 3))
plt.plot(x, y)
plt.title('Distribution of MLEs when M=100, C=100, N=1000')
plt.xlabel('Population size estimate $\hat{N}$')
plt.ylabel('Probability')
plt.show()
```



In [18]:
```
x = np.arange(2, 10000)
p_n = [mle_probability(n=n, C=100, M=100, N=1000) for n in x]
p_n /= np.sum(p_n)
mean_N = np.sum(p_n * x)
std_N = np.sum(p_n * np.abs(x - mean_N))
mean_N, std_N
```

Out[18]: (1633.4328341324479, 683.1636297688008)

This calculation shows that the estimator is biased: though the true population is 1000, it estimates 1633 on average. The estimator also has large variance: its estimates have a standard deviation of 683. The variance of this estimator is about the same as its squared bias.

---

d)

Some authors have suggested an alternative estimator:
$\hat{N}' = \lfloor (M+1)(C+1)/(K+1) \rfloor$, where $\lfloor \cdot \rfloor$ indicates truncation to the next lower integer. Repeat part (c) for this estimator and compare the bias and variance of this estimator to the original one.

We can rewrite $n = \lfloor (M+1)(C+1)/(k+1) \rfloor$ as the inequality

$$n \leq \frac{(M+1)(C+1)}{k+1} < n+1.$$

Multiplying both sides by $k+1$ gives

$$(k+1)n \leq (M+1)(C+1) < (k+1)(n+1).$$

We can rewrite this as two inequalities

$$k+1 \leq \frac{(M+1)(C+1)}{n} \text{ and } \frac{(M+1)(C+1)}{n+1} < k+1,$$

and then write the single equality

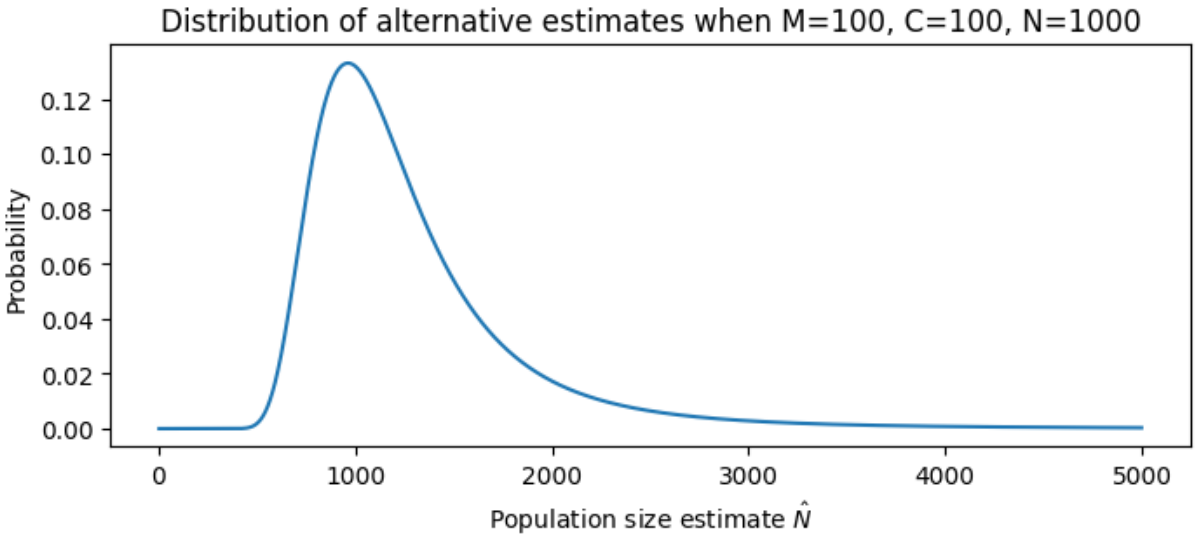
$$\frac{(M+1)(C+1)}{n+1} - 1 < k \leq \frac{(M+1)(C+1)}{n} - 1.$$

Given $n = \lfloor (M+1)(C+1)/(k+1) \rfloor$, we want the k which makes the quantity inside the floor function as low as possible. This happens at the maximum for k in the inequality above:

$$k = \frac{(M+1)(C+1)}{n} - 1.$$

In [19]:
```
def alt_probability(n, N, M, C):
    theta = M / N
    k = (M + 1) * (C + 1) / n - 1
    return comb(C, k) * (theta)**k * (1 - theta)**(C - k)
```

In [20]:
```
x = np.arange(2, 5000)
y = [alt_probability(n=n, C=100, M=100, N=1000) for n in x]
plt.subplots(figsize=(8, 3))
plt.plot(x, y)
plt.title('Distribution of alternative estimates when M=100, C=100, N=1000')
plt.xlabel('Population size estimate $\hat{N}$')
plt.ylabel('Probability')
plt.show()
```



In [21]:
```
x = np.arange(2, 10000)
p_n = [alt_probability(n=n, C=100, M=100, N=1000) for n in x]
p_n /= np.sum(p_n)
mean_N = np.sum(p_n * x)
std_N = np.sum(p_n * np.abs(x - mean_N))
mean_N, std_N
```

Out[21]: (1301.6259717957555, 415.6120984094553)

This estimator has lower bias and variance than the maximum likelihood estimator (bias: 1633, std: 683). It does not overestimate as badly and we can expect it to converge more quickly.

In [ ]:

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

Bayesian estimation.

Xinyuan and Sarah are looking for Ajay in a very large one-dimensional shopping mall. Location is specified by a coordinate X. They know that, all else being equal, Ajay likes to hang out near the center of the shopping mall. Specifically, the probability distribution of his location is Gaussian with mean $X = 75$ and variance 35. The only clue they have is a coffee cup at location $X = 40$, containing the residue of a coffee that only Math Tools TAs drink (a naturally dried, triple-caffeine Ethiopian heirloom varietal, hand-extracted). Given the location of the coffee cup, and the aroma and dryness of the coffee residue, they estimate the likelihood of his position, i.e., the probability of finding a cup at that location in that condition given Ajay's current position, to be a Gaussian with mean $X = 40$ and variance 80.

In [2]:
```
def norm_pdf(x, mu, sigma):
    const = 1 / (np.sqrt(2 * np.pi) * sigma)
    y = const * np.exp(-(x - mu)**2 / (2 * sigma**2))
    return y / np.sum(y)
```

---

a)

Frame this problem as a problem in Bayesian estimation, using appropriate terminology. What is Ajay's posterior distribution? Draw his prior distribution, likelihood function and posterior distribution on a single plot. (Rather than normpdf, compute the probabilities from the formula for the Gaussian distribution.) What are the mean and variance of the posterior?

In [3]:
```
def bayes_rule(likelihood, prior):
    posterior = likelihood * prior
    return posterior / np.sum(posterior)
```

Ajay is a system whose hidden state we infer from observations and a prior expectation over possible states. The state variable X is Ajay's position within the mall. The observation Y is an old coffee cup at a particular location. There exists a likelihood function $P(Y|X)$ which, given a possible location for Ajay X, tells us the probability of observing the coffee cup we saw. We also have a prior distribution $P(X)$ which gives the probability of finding Ajay in some location in the absence of any observations.

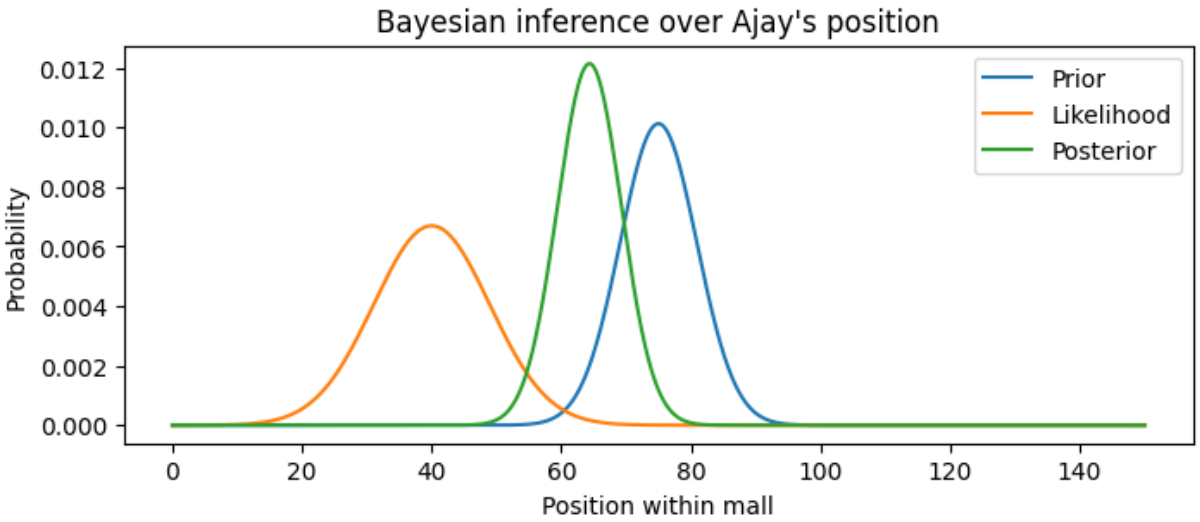
The prior distribution $P(X)$ encodes our beliefs about Ajay's location in the absence of evidence. Bayes' rule gives an equation for optimally updating our belief in light of incoming evidence. It defines a new distribution $P(X|Y)$, called the posterior, in terms of the likelihood and prior:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}.$$

The posterior $P(X|Y)$ encodes our beliefs about Ajay's location after the observation. The prior, likelihood and posterior are plotted below.

In [4]:
```
x = np.linspace(0, 150, 1000)
prior = norm_pdf(x, mu=75, sigma=np.sqrt(35))
likelihood = norm_pdf(x, mu=40, sigma=np.sqrt(80))
posterior = bayes_rule(likelihood, prior)

plt.subplots(figsize=(8, 3))
plt.title("Bayesian inference over Ajay's position")
plt.xlabel('Position within mall')
plt.ylabel('Probability')
plt.plot(x, prior, label='Prior')
plt.plot(x, likelihood, label='Likelihood')
plt.plot(x, posterior, label='Posterior')
plt.legend()
plt.show()
```



In [5]:
```
mean = np.sum(posterior * x)
var = np.sum(posterior * (x - mean)**2)
mean, var
```

Out[5]: (64.34782608695652, 24.34782608695652)

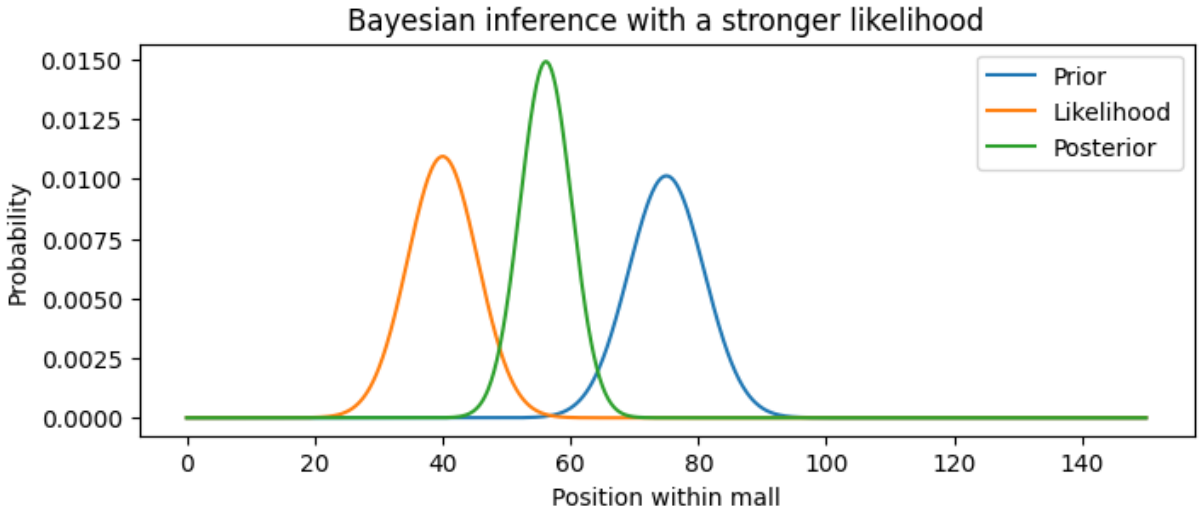
---

b)

Xinyuan and Sarah realize they over-estimated the sitting time of the coffee residue, and decide that Ajay's likelihood function has mean $X = 40$ but with a much smaller variance of 30. Redo part (a), and describe what happened to the posterior distribution, in terms of mean and variance. Does the change make sense?

In [6]:
```
x = np.linspace(0, 150, 1000)
prior = norm_pdf(x, mu=75, sigma=np.sqrt(35))
likelihood = norm_pdf(x, mu=40, sigma=np.sqrt(30))
posterior = bayes_rule(likelihood, prior)

plt.subplots(figsize=(8, 3))
plt.title("Bayesian inference with a stronger likelihood")
plt.xlabel('Position within mall')
plt.ylabel('Probability')
plt.plot(x, prior, label='Prior')
plt.plot(x, likelihood, label='Likelihood')
plt.plot(x, posterior, label='Posterior')
plt.legend()
plt.show()
```



In [7]:
```
mean = np.sum(posterior * x)
var = np.sum(posterior * (x - mean)**2)
mean, var
```

Out[7]: (56.15384615384614, 16.153846153846153)

If they over-estimated how long the coffee residue was sitting there, they over-estimated the age of the coffee cup, which means they may have over-estimated how far Ajay has traveled from the cup since leaving it there. Then the likelihood should become more concentrated, reflecting greater certainty that Ajay is still in the area.

The greater certainty that Ajay is near the coffee cup moves their estimate of Ajay's location further from the prior than in part (a), and toward the peak of the likelihood. Additionally, reducing the variance of the likelihood reduces the variance of the posterior

compared to part (a). Geometrically, decreasing the width of the likelihood results in less overlap between the likelihood and the prior, reducing the width of the posterior.

---

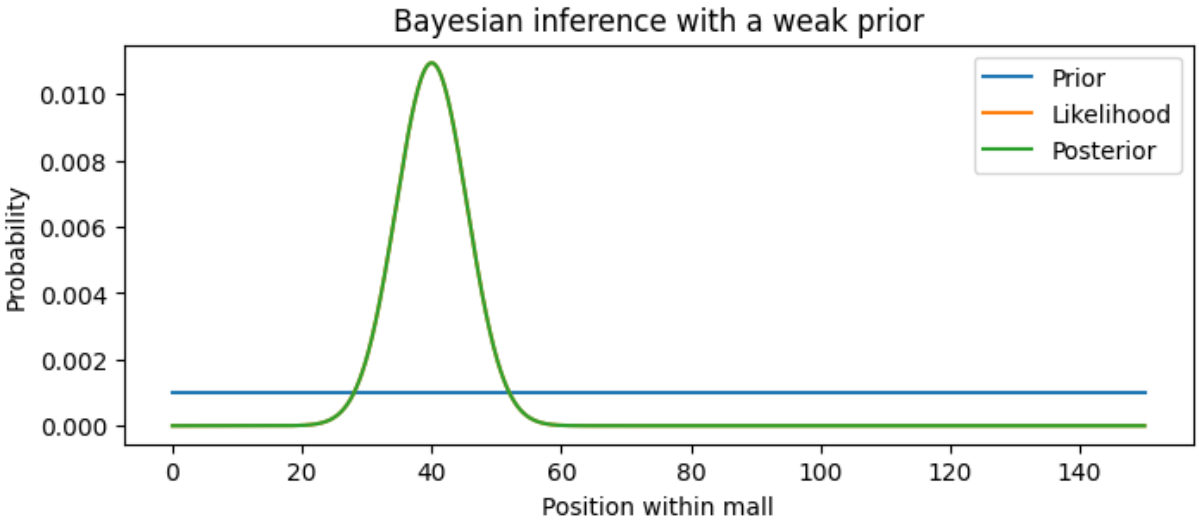
c)

What would the posterior distribution in (a) be if the prior was nearly flat (e.g., variance 10^6). Compare this variance to that of the posterior in (a). How does the inclusion of prior information affect the variance?

As the prior becomes flat, i.e. approaches a uniform distribution, the posterior becomes equal to the likelihood. This is because the posterior is just the likelihood re-weighted by the prior (to a proportional constant). If the prior is a uniform distribution this re-weighting has no effect on the likelihood at all.

In [11]:
```
x = np.linspace(0, 150, 1000)
prior = norm_pdf(x, mu=75, sigma=np.sqrt(10e6))
likelihood = norm_pdf(x, mu=40, sigma=np.sqrt(30))
posterior = bayes_rule(likelihood, prior)

plt.subplots(figsize=(8, 3))
plt.title("Bayesian inference with a weak prior")
plt.xlabel('Position within mall')
plt.ylabel('Probability')
plt.plot(x, prior, label='Prior')
plt.plot(x, likelihood, label='Likelihood')
plt.plot(x, posterior, label='Posterior')
plt.legend()
plt.show()
```



In [12]:
```
mean = np.sum(posterior * x)
var = np.sum(posterior * (x - mean)**2)
mean, var
```

Out[12]: (40.00010499969018, 29.99991000006257)

This variance is large compared to the posterior variance in part (a). Thus, including prior information decreases the variance of the posterior. This makes sense as more certainty at the onset results in greater certainty following some set of observations, all else being held equal.

In [ ]:

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

Bayesian inference of binomial proportions.

Poldrack (2006) published an influential attack on the practice of "reverse inference" in fMRI studies, i.e., inferring that a cognitive process was engaged on the basis of activation in some area. For instance, if Broca's area was found to be activated using standard fMRI statistical-contrast techniques, researchers might infer that the subjects were using language. In a search of the literature, Poldrack found that Broca's area was reported activated in 90 out of 840 fMRI contrasts involving engagement of language, but this area was also active in 215 out of 2754 contrasts not involving language.

---

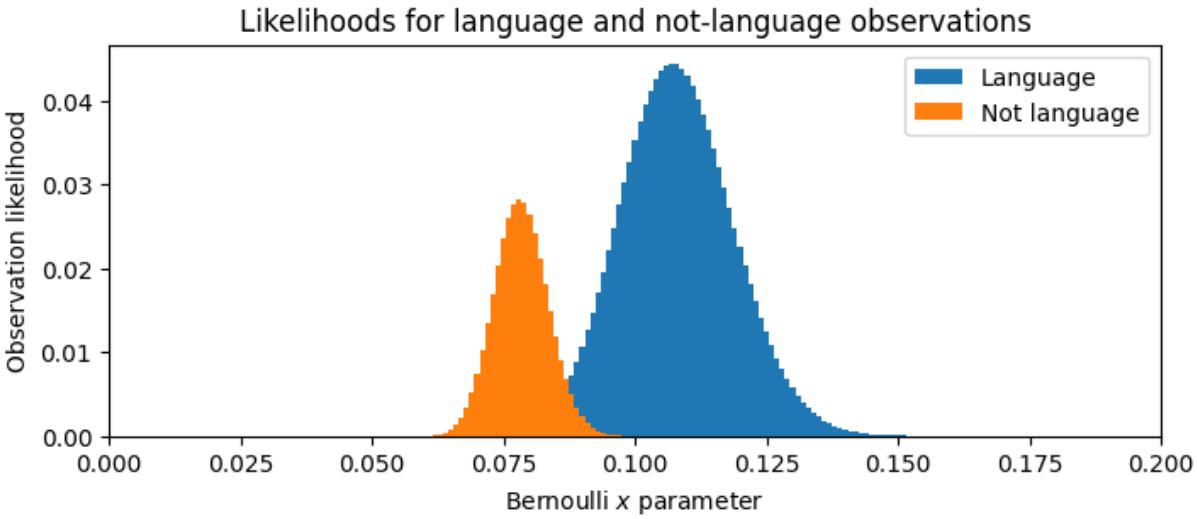
a)

Assume that the conditional probability of activation given language, as well as that of activation given no language, each follow a Bernoulli distribution (i.e., like coin-flipping), with parameters x_l and x_{nl}. Compute the likelihoods of these parameters, given Poldrack's observed frequencies of activation. Compute these functions at the values $x = [0 : .001 : 1]$ and plot them as a bar chart.

In [2]:
```
from scipy.stats import binom
```

In [3]:
```
x = np.linspace(0, 1, 1001)
likelihood_l = binom.pmf(k=90, n=840, p=x)
likelihood_nl = binom.pmf(k=215, n=2754, p=x)
```

In [4]:
```
plt.subplots(figsize=(8, 3))
plt.title('Likelihoods for language and not-language observations')
plt.xlabel('Bernoulli $x$ parameter')
plt.ylabel('Observation likelihood')
plt.xlim([0, 0.2])
plt.bar(x, likelihood_l, 0.001, label='Language')
plt.bar(x, likelihood_nl, 0.001, label='Not language')
plt.legend()
plt.show()
```



---

b)

Find the value of x that maximizes each discretized likelihood function. Compare these to the exact maximum likelihood estimates given by the formula for the ML estimator of a Bernoulli probability.

In [5]:
```
x_l = x[np.argmax(likelihood_l)]
x_l, 90 / 840
```

Out[5]:
```
(0.107, 0.10714285714285714)
```

In [6]:
```
x_nl = x[np.argmax(likelihood_nl)]
x_nl, 215 / 2754
```

Out[6]:
```
(0.078, 0.07806826434277414)
```

The discretized values that maximize the likelihood function are as close as possible to the true values given their discretization in increments of 0.001.

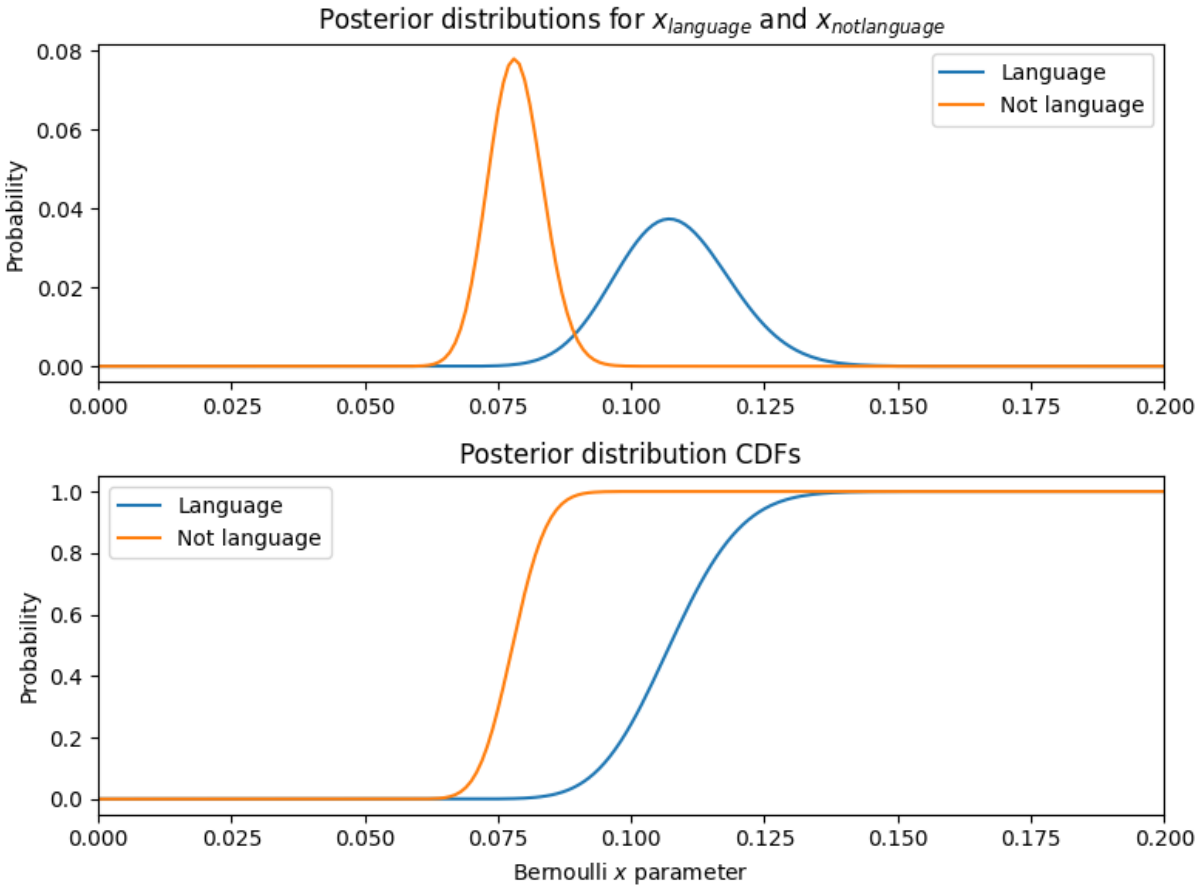
---

c)

Using the likelihood functions computed for discrete x, compute and plot the discrete posterior distributions $P(x|data)$ and the associated cumulative distributions $P(X \leq x|data)$ for both processes. For this, assume a uniform prior $P(x) \propto 1$ and note that it will be necessary to compute (rather than ignore) the normalizing constant for Bayes' rule. Use the cumulative distributions to compute (discrete approximations to) upper and lower 95% confidence bounds on each proportion.

In [7]:
```
posterior_l = likelihood_l / np.sum(likelihood_l)
posterior_nl = likelihood_nl / np.sum(likelihood_nl)
cdf_l = np.cumsum(posterior_l)
cdf_nl = np.cumsum(posterior_nl)
```

In [8]:
```
fig, axs = plt.subplots(2, 1, figsize=(8, 6))
plt.sca(axs[0])
plt.title('Posterior distributions for $x_{language}$ and $x_{not language}$
plt.xlim([0, 0.2])
plt.ylabel('Probability')
plt.plot(x, posterior_l, label='Language')
plt.plot(x, posterior_nl, label='Not language')
plt.legend()
plt.sca(axs[1])
plt.title('Posterior distribution CDFs')
plt.ylabel('Probability')
plt.xlabel('Bernoulli $x$ parameter')
plt.xlim([0, 0.2])
plt.plot(x, cdf_l, label='Language')
plt.plot(x, cdf_nl, label='Not language')
plt.legend()
plt.tight_layout()
plt.show()
```



In [9]:
```
x_l_05 = x[np.where(cdf_l > 0.05)[0]][0]
x_l_95 = x[np.where(cdf_l < 0.95)[0]][-1]
x_nl_05 = x[np.where(cdf_nl > 0.05)[0]][0]
x_nl_95 = x[np.where(cdf_nl < 0.95)[0]][-1]
```

Upper and lower 95% confidence bounds for x_l:

In [10]:
```
x_l_05, x_l_95
```

Out[10]:
```
(0.091, 0.125)
```

For x_{nl}:

In [11]:
```
x_nl_05, x_nl_95
```

Out[11]:
```
(0.07, 0.08600000000000001)
```

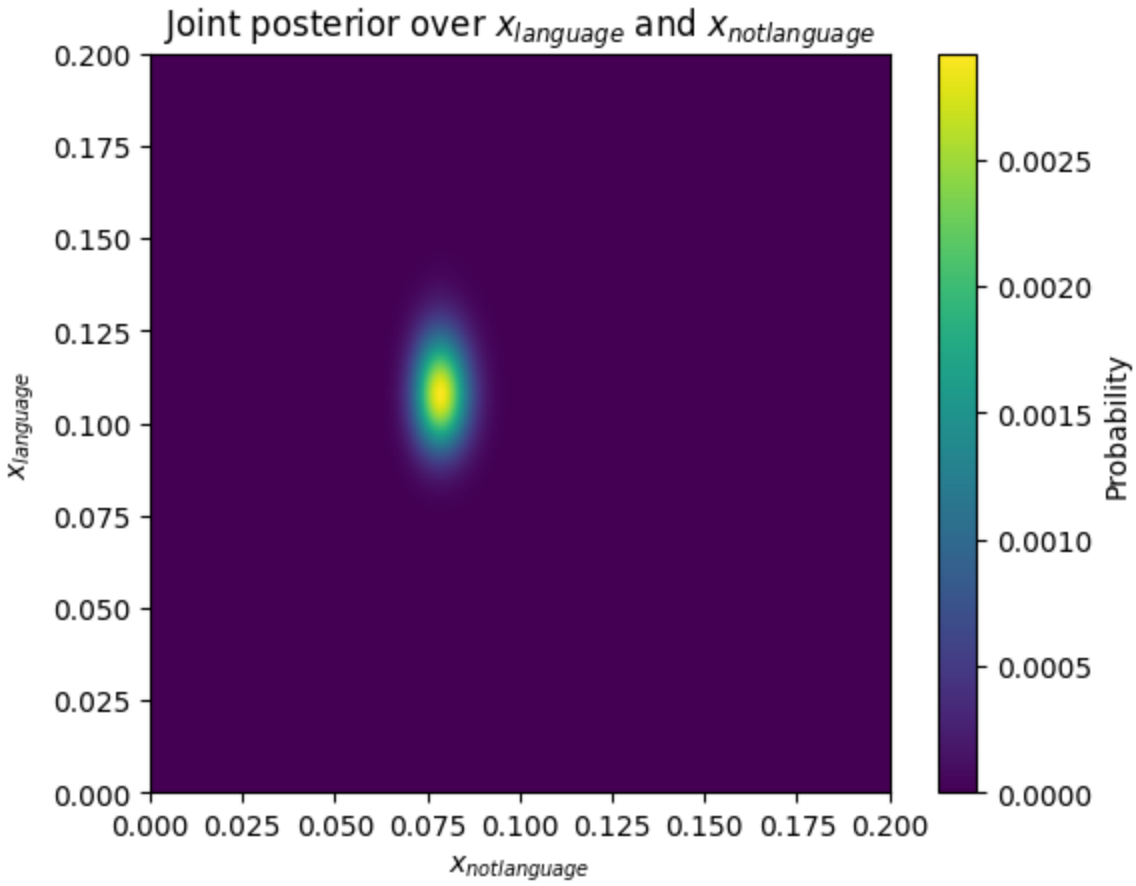
---

d)

Are these frequencies different from one another? Consider the joint posterior distribution over x_l and x_{nl}, the Bernoulli probability parameters for the language and non-language contrasts. Given that these two frequencies are independent, the

(discrete) joint distribution is given by the outer product of the two marginals. Plot it (with imagesc). Compute (by summing the appropriate entries in the joint distribution) the posterior probabilities that $x_l > x_{nl}$ and, conversely, that $x_l \leq x_{nl}$.

In [12]:
```
posterior_joint = posterior_l[:, np.newaxis] @ posterior_nl[np.newaxis, :]
posterior_joint /= np.sum(posterior_joint)
plt.title('Joint posterior over $x_{language}$ and $x_{not language}$')
plt.imshow(posterior_joint, extent=[0, 1, 1, 0])
plt.xlim([0, 0.2])
plt.ylim([0, 0.2])
plt.xlabel('$x_{not language}$')
plt.ylabel('$x_{language}$')
plt.colorbar(label='Probability')
plt.show()
```



In [13]:
```
p_l = 0
p_nl = 0
for row, probs in enumerate(posterior_joint):
    for col, p in enumerate(probs):
        if row > col: # x_l > x_nl
            p_l += p
        else:         # x_nl >= x_l
            p_nl += p
```

Posterior probabilities that $x_l > x_{nl}$ and $x_l \leq x_{nl}$:

In [14]:
```
p_l, p_nl
```

Out[14]:
```
(0.9949447070648347, 0.005055292935156522)
```

Odds:

In [15]:
```
p_l / p_nl
```

Out[15]:
```
196.81247354541864
```

These odds make it overwhelming likely that activation due to language is present.

---

e)

Is this difference sufficient to support reverse inference? Compute the probability $P(language|activation)$. This is the probability that observing activation in Broca's area implies engagement of language processes. To do this use the estimates from part (b) as the relevant conditional probabilities, and assuming the prior that a contrast engages language, $P(language) = 0.5$. Hint: To calculate this probability, you will need to "marginalize", i.e., integrate over the unknown values of x_l and x_{nl}. Poldrack's critique said that we cannot simply conclude that activation in a given area indicates that a cognitive process was engaged without computing the posterior probability. Is this critique correct? To answer this, compute the posterior odds $(\frac{p(language|activation)}{p(notlanguage|activation)})$ using the maximum-likelihood estimates of x_l and x_{nl} from Poldrack's data of activation probabilities and compare the posterior odds to the prior odds before running your experiment $(\frac{p(language)}{p(notlanguage)})$.

In [16]:
```
x_l = 90/840
x_nl = 215/2754
```

In [17]:
```
p_l = 0.5
p_nl = 0.5
```

Denote observing activation by A and language engagement by L. Then

$P(A|L)$ is the probability of observing a contrast given a language task. It is Bernoulli distributed with success probability x_l.

In [18]:
```
p_a_l = x_l
p_na_l = 1 - x_l
```

$P(A|\neg L)$ is the probability of observing a contrast in a non-language task. It is Bernoulli distributed with success probability $x_n l$.

In [19]:
```
p_a_nl = x_nl
p_na_nl = 1 - x_nl
```

To compute $P(A)$, marginalize:

$P(A) = \sum_l P(A|L = l) = P(A|L) + P(A|\neg L)$.

In [20]:
```
p_a = p_a_l + p_a_nl
```

Then Bayes' rule gives $P(language|activation)$ and $P(notlanguage|activation)$ as

$P(L|A) = \frac{P(A|L)P(L)}{P(A)}$

and

$P(\neg L|A) = \frac{P(A|\neg L)P(\neg L)}{P(A)}$.

$P(language|activation)$:

In [21]:
```
p_l = p_a_l * p_l / p_a
p_l
```

Out[21]:
```
0.28924520375297574
```

$P(notlanguage|activation)$:

In [22]:
```
p_nl = p_a_nl * p_nl / p_a
p_nl
```

Out[22]:
```
0.2107547962470242
```

Odds:

In [23]:
```
p_l / p_nl
```

Out[23]:
```
1.3724252491694353
```

The observation moved us from a uniform prior to the odds 1.37:1. With these odds, it is only slightly more likely that the activation was due to language than it was due to something not-language. In the previous case, the odds were 196.8:1, while here they are only 1.37:1. Computing the posterior probability changes the odds that activation due to language is present. Poldrack's critique stands.

In [1]:
```
import numpy as np
from matplotlib import pyplot as plt
```

Signal Detection Theory.

Consider an experiment where a moving-dot visual stimulus is presented to a subject. The difficulty of detecting the motion is varied by changing the coherence of the moving dots, which is the fraction of dots moving to the right (at zero coherence, the dots move randomly, and at 100% coherence, all of the dots move to the right). Suppose we want to decide whether the stimulus is random or is moving to the right, based on the response of a single neuron that fires at a random rate, whose mean is 3 spikes/s in response to a 0% coherence noisy stimulus and 5 spikes/s for 10% coherence. Suppose also that the distribution of firing rates is Gaussian with a standard deviation of 1 spikes/s for both stimuli.

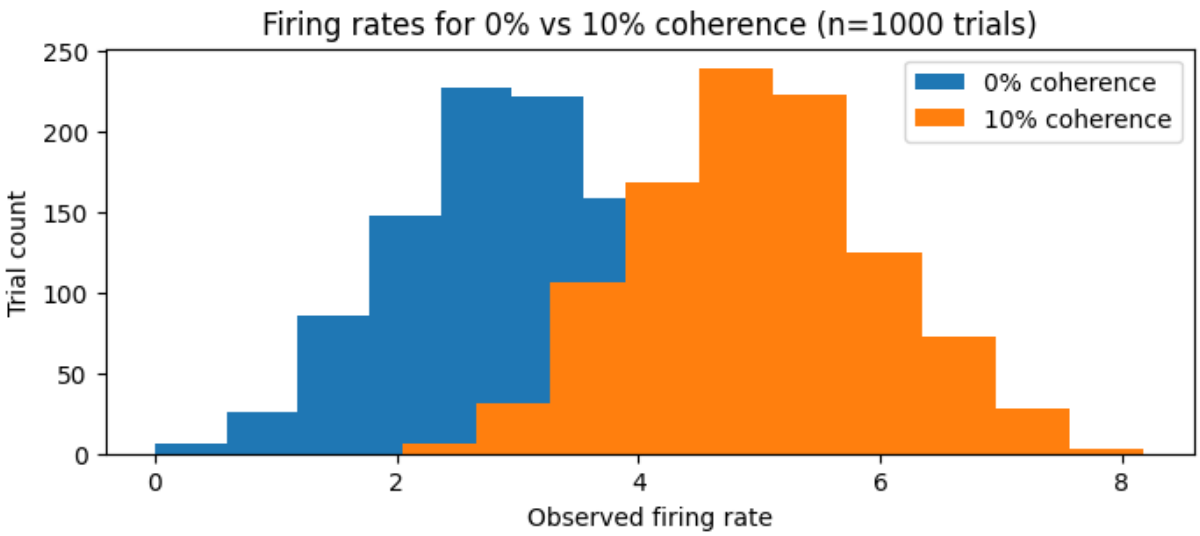
---

a)

For the “no coherence” stimulus, generate 1000 trials of the firing rate of the neuron in response to these stimuli (i.e., draw 1000 random samples from a Gaussian with $\mu = 3$ and $\sigma = 1$). Since we cannot have negative firing rates, set all rates that are below zero to zero. Now do the same thing for the 10% coherence stimulus. On the same figure, plot the histograms of the firing rates for each stimulus type.

In [2]:
```
fr_0 = np.random.randn(1000) + 3
fr_10 = np.random.randn(1000) + 5
fr_0[fr_0 < 0] = 0
fr_10[fr_10 < 0] = 0
```

In [3]:
```
plt.subplots(figsize=(8, 3))
plt.title('Firing rates for 0% vs 10% coherence (n=1000 trials)')
plt.hist(fr_0, label='0% coherence')
plt.hist(fr_10, label='10% coherence')
plt.xlabel('Observed firing rate')
plt.ylabel('Trial count')
plt.legend()
plt.show()
```



b)

The success of the decoder (assuming this model of Gaussian noise) is determined by two things, the separation of the mean firing rates and the standard deviation of the neuron. From class, we know that this is captured in the measure known as d'. Calculate d' for this task and pair of stimuli (ignoring the fact that you are clipping firing rates at zero).

d' is the distance between the two means in units of standard deviations. Stimulus 1 (0% coherence) has a mean of 3 and standard deviation of 1, while stimulus 2 (10% coherence) has a mean of 5 and standard deviation 1. So

$$d' = \frac{5-3}{1} = 2.$$

Measured emperically,

In [4]:
```
avg_std = 1/2 * (np.std(fr_0) + np.std(fr_10))
(np.mean(fr_10) - np.mean(fr_0)) / avg_std
```

Out[4]:
```
1.9995877886983489
```

---

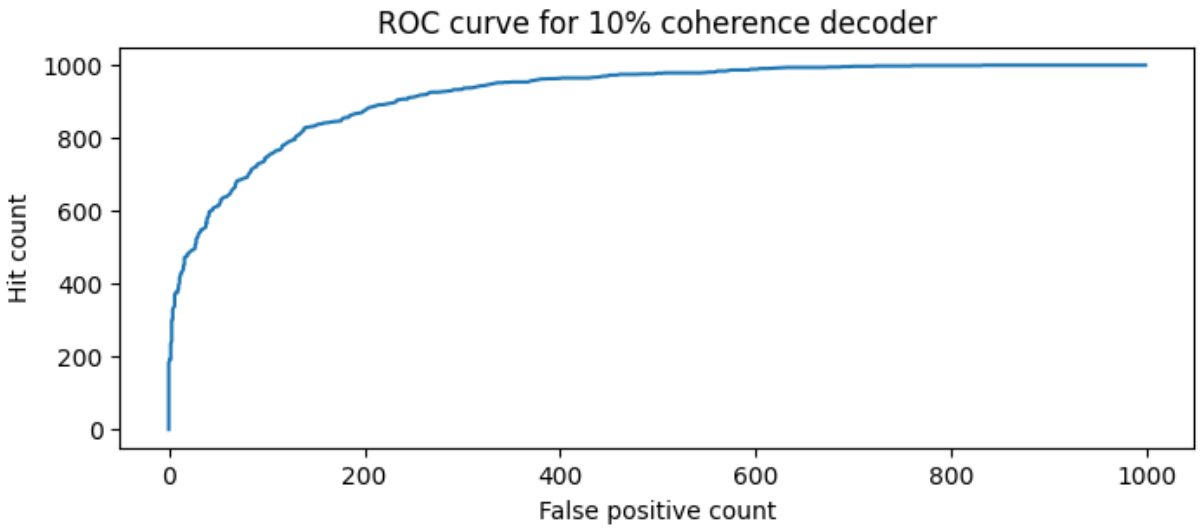
c)

Explain why the maximum likelihood decoder for this problem involves comparing the measurement to a threshold. For various thresholds t, calculate the hit and false-alarm rates using your sample data from (a), and plot these against each other (this is an ROC curve, defined in class).

This decoder tries to guess from which of two Gaussian-distributed random variables a sample was drawn. It does this by comparing the likelihood of drawing that sample from each of the two distributions. For equal-shape, unimodal, symmetric distributions, the ML decision rule can be expressed as a threshold function. This problem is one such problem: the firing rate distributions for 0% and 10% coherence are unimodal and symmetric (since they are Gaussians) and equal shape (since they have the same standard deviation).

In [5]:
```
thresholds = np.linspace(0, 10, 1000)
n_hits = []
n_fps = []
for t in thresholds:
    n_hits.append(np.sum(fr_10 > t))
    n_fps.append(np.sum(fr_0 > t))
```

In [6]:
```
plt.subplots(figsize=(8, 3))
plt.title('ROC curve for 10% coherence decoder')
plt.plot(n_fps, n_hits)
plt.xlabel('False positive count')
plt.ylabel('Hit count')
plt.show()
```



What threshold would you pick based on this curve to maximize the percentage-correct of the decoder, assuming that 0% and 10% coherence stimuli occur equally often. Plot this threshold as a point on the ROC curve and as a vertical line on your histogram from part (a).

Assuming 0% and 10% coherence stimuli occur equally often is a uniform prior. In this case the posterior is directly proportional to the likelihood. Since both likelihoods are equal-shaped, unimodal and symmetric, the optimal threshold lies exactly between them at the firing rate 4 spikes/s.

Numerically, we can find it for our dataset by picking the threshold that maximizes hit rate minus false-positive rate:

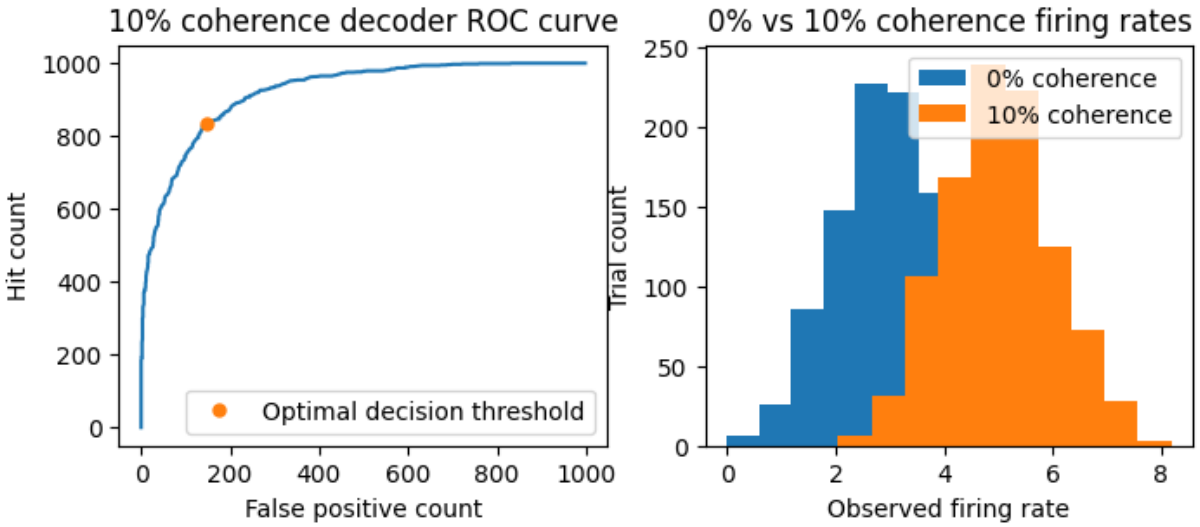
In [7]:
```
thresholds[np.argmax(np.array(n_hits) - np.array(n_fps))]
```

Out[7]:
```
4.044044044044044
```

In [8]:
```
hits = np.sum(fr_10 > 4.024)
fps = np.sum(fr_0 > 4.024)
hits, fps
```

Out[8]:
```
(833, 149)
```

In [9]:
```
fig, axs = plt.subplots(1, 2, figsize=(8, 3))
plt.sca(axs[0])
plt.title('10% coherence decoder ROC curve')
plt.plot(n_fps, n_hits)
plt.plot(fps, hits, '.', markersize=10, label='Optimal decision threshold')
plt.xlabel('False positive count')
plt.ylabel('Hit count')
plt.legend()
plt.sca(axs[1])
plt.title('0% vs 10% coherence firing rates')
plt.hist(fr_0, label='0% coherence')
plt.hist(fr_10, label='10% coherence')
plt.xlabel('Observed firing rate')
plt.ylabel('Trial count')
plt.legend()
plt.show()
```



Next, suppose that 10% coherence stimuli occur 75% of the time. Determine and plot the threshold that maximizes percentage correct for this new prior.

Denote 10% coherence by $S + N$ (signal plus noise), 0% coherence by N (noise), and firing rate by x. Then

$P(S+N) = 0.75$

$P(N) = 0.25$

$P(x|S+N) = \mathcal{N}(\mu = 5, \sigma = 1)$

$P(x|N) = \mathcal{N}(\mu = 3, \sigma = 2).$

Our optimality criterion is to say yes if

$\frac{P(S+N|x)}{P(N|x)} >= 1$

So our decision threshold should lie at the x satisfies $P(S+N|x) = P(N|X)$.

Bayes rule says $P(S+N|x) = \frac{P(x|S+N)P(S+N)}{P(x)}$ and $P(N|x) = \frac{P(x|N)P(N)}{P(x)}$,

so we can write the equation for our decision rule as

$P(x|S+N)P(S+N) = P(x|N)P(N)$. More explicitly,

$0.75 \times e^{\frac{-(x-5)^2}{2}} = 0.25 \times e^{\frac{-(x-3)^2}{2}}$. Solving for x,

$\ln 0.75 - \frac{1}{2}(x-5)^2 = \ln 0.25 - \frac{1}{2}(x-3)^2$

$\ln 3 - \frac{1}{2}(x^2 - 10x + 25) = -\frac{1}{2}(x^2 - 6x + 9)$

$4x = 16 - 2\ln 3$

$x = 4 - \frac{1}{2}\ln 3.$

We can interpret this result as taking the decision threshold $x = 4$ with an equal prior and adjusting it by the term $-\frac{1}{2}\ln 3$, where $\ln 3$ is the log-odds by which we re-weighted the prior $(0.75 : 0.25)$.

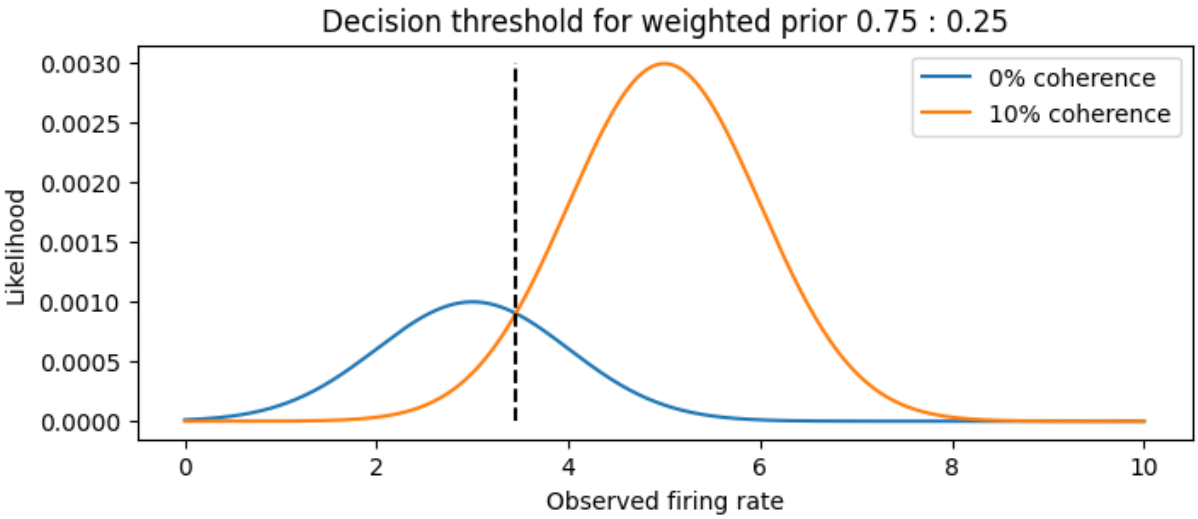
In [10]:
```
t_opt = 4 - (1/2) * np.log(3)
t_opt
```

Out[10]:
```
3.450693855665945
```

In [11]:
```
def norm_pdf(x, mu, sigma):
    const = 1 / (np.sqrt(2 * np.pi) * sigma)
    y = const * np.exp(-(x - mu)**2 / (2 * sigma**2))
    return y / np.sum(y)
```

In [12]:
```
x = np.linspace(0, 10, 1000)
p_n = 0.25 * norm_pdf(x, mu=3, sigma=1)
p_sn = 0.75 * norm_pdf(x, mu=5, sigma=1)
```

In [13]:
```
plt.subplots(figsize=(8, 3))
plt.title('Decision threshold for weighted prior 0.75 : 0.25')
plt.plot(x, p_n, label='0% coherence')
plt.plot(x, p_sn, label='10% coherence')
plt.vlines(t_opt, 0, 0.003, 'k', '--')
plt.xlabel('Observed firing rate')
plt.ylabel('Likelihood')
plt.legend()
plt.show()
```



---

d)

Consider now a neuron with a more "noisy" response so that the mean firing rates are the same but the standard deviation is 2 spikes/s instead of 1 spike/s. What is the new value of d'. Recompute and plot the optimal (maximum accuracy) thresholds for this noisy neuron for both the 50-50 and 75-25 priors. How do they differ from those in the previous part?

In [14]:
```
fr_0 = 2 * np.random.randn(1000) + 3
fr_10 = 2 * np.random.randn(1000) + 5
fr_0[fr_0 < 0] = 0
fr_10[fr_10 < 0] = 0
```

The new value of d' for our model is $\frac{5-3}{2} = 1$. Empirically:

In [15]:
```
avg_std = 1/2 * (np.std(fr_0) + np.std(fr_10))
(np.mean(fr_10) - np.mean(fr_0)) / avg_std
```

Out[15]:
```
1.0698058736842155
```

Compute the optimal threshold for the 50-50 priors:

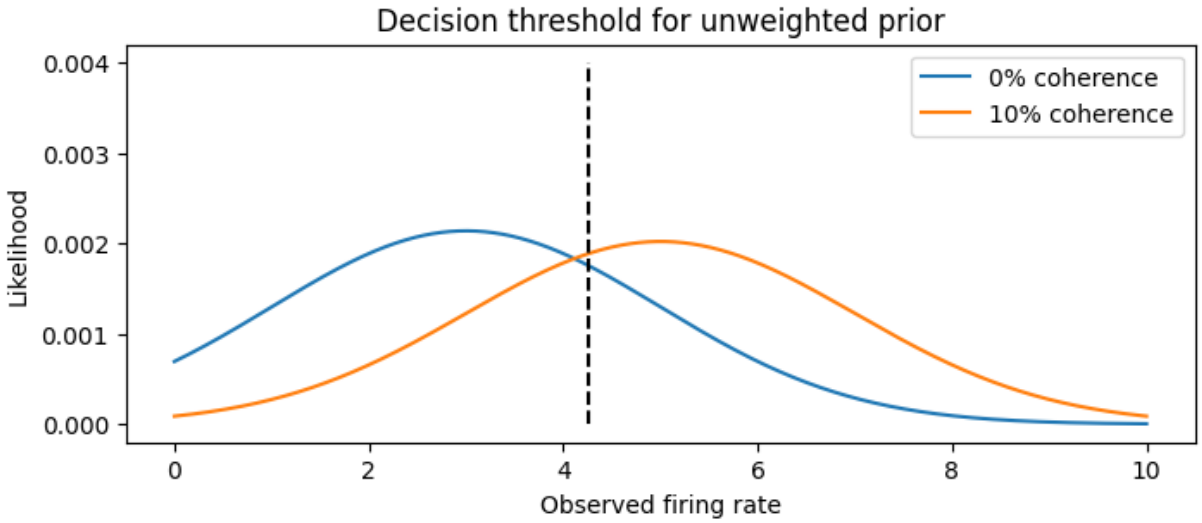
In [16]:
```
n_hits = []
n_fps = []
for t in thresholds:
    n_hits.append(np.sum(fr_10 > t))
    n_fps.append(np.sum(fr_0 > t))

t_opt = thresholds[np.argmax(np.array(n_hits) - np.array(n_fps))]
t_opt
```

Out[16]: 4.2542542542542545

In [17]:
```
x = np.linspace(0, 10, 1000)
p_n = norm_pdf(x, mu=3, sigma=2)
p_sn = norm_pdf(x, mu=5, sigma=2)
```

In [18]:
```
plt.subplots(figsize=(8, 3))
plt.title('Decision threshold for unweighted prior')
plt.plot(x, p_n, label='0% coherence')
plt.plot(x, p_sn, label='10% coherence')
plt.vlines(t_opt, 0, 0.004, 'k', '--')
plt.xlabel('Observed firing rate')
plt.ylabel('Likelihood')
plt.legend()
plt.show()
```



For the 75-25 prior, we follow the same calculation as in (c). The equation for our decision rule is

$P(x|S+N)P(S+N) = P(x|N)P(N)$. Specifically,

$0.75 \times e^{\frac{-(x-5)^2}{8}} = 0.25 \times e^{\frac{-(x-3)^2}{8}}$. Solving for x,

$\ln 0.75 - \frac{1}{8}(x-5)^2 = \ln 0.25 - \frac{1}{8}(x-3)^2$

$\ln 3 - \frac{1}{8}(x^2 - 10x + 25) = -\frac{1}{8}(x^2 - 6x + 9)$

$$4x = 16 - 8\ln 3$$

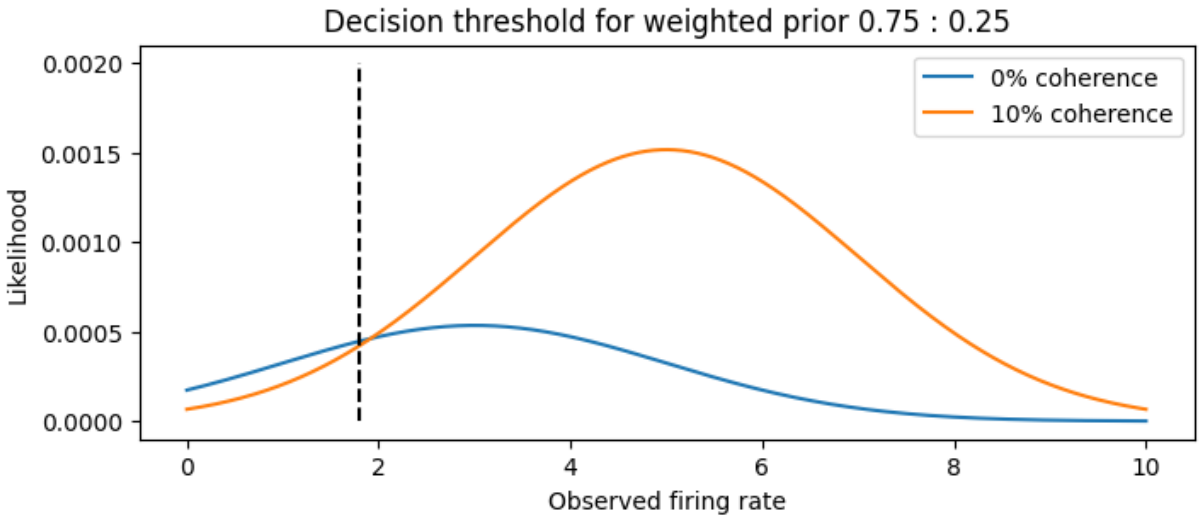
$$x = 4 - 2\ln 3.$$

In [19]:
```
t_opt = 4 - 2 * np.log(3)
t_opt
```

Out[19]:
```
1.8027754226637804
```

In [20]:
```
x = np.linspace(0, 10, 1000)
p_n = 0.25 * norm_pdf(x, mu=3, sigma=2)
p_sn = 0.75 * norm_pdf(x, mu=5, sigma=2)
```

In [21]:
```
plt.subplots(figsize=(8, 3))
plt.title('Decision threshold for weighted prior 0.75 : 0.25')
plt.plot(x, p_n, label='0% coherence')
plt.plot(x, p_sn, label='10% coherence')
plt.vlines(t_opt, 0, 0.002, 'k', '--')
plt.xlabel('Observed firing rate')
plt.ylabel('Likelihood')
plt.legend()
plt.show()
```



In the unweighted case, the optimal decision threshold stays as before. However, since d' is lower than before (due to decreased variance) the decoder accuracy will degrade somewhat compared to before (area under the ROC curve will be lower).

In the weighted case, something interesting happens: the optimal decision threshold actually goes *below* 3, the mean firing rate for noise. Even if we observe a firing rate slightly less than 3 (e.g. 2), the decoder will classify the observation as 10% coherence due to its 75% prior weighting. The firing rate actually needs to be lower than 1.802 for the decoder to classify the observation as noise.

In [ ]:
```
```