ASSIGNMENT 3

LUKAS KOEBE 03723950
KEWIN HAWRYLUK 03692265
THEODOR STOICAN 03725732

Problem 1

Compute the first and second derivative of this likelihood w.r.t. θ . Then compute first and second derivative of the log likelihood

$$log\theta^t(1-0)^h.$$

Given likelihood maximization problem:

$$argmax f(\theta) = \theta^t * (1-\theta)^h$$

First derivative:

$$f'(\theta) = t * \theta^{t-1} * (1-\theta)^h - \theta^t * h(1-\theta)^{h-1}$$

Second derviative:

$$f''(\theta) = t(t-1)\theta^{t-2} * (1-\theta)^h - t\theta^{t-1} * h(1-\theta)^{h-1} - ht\theta^{t-1} * (1-\theta)^{h-1} + \theta^t * (h-1) * (1-\theta)^{h-2}$$

First derivative of log likelyhood $\log\theta^t(1-\theta)^h$:

$$f(\theta) = \log\theta^t(1-\theta)^h$$
$$f'(\theta) = t * \log\theta + h * log(1-\theta)$$
$$f'(\theta) = \frac{t}{\theta} - \frac{h}{1-\theta}$$

Second derivative:

$$f''(\theta) = -t * \theta^{-2} - h * (1-\theta)^{-2}$$
$$f''(\theta) = -\frac{t}{\theta^2} - \frac{h}{(1-\theta)^2}$$

Problem 2

Show that every local maximum of $log f(\theta)$ is also a local maximum of the differentiable,positive function $f(\theta)$. Considering this and the previous exercise, what is your conclusion?

Calculating the first derivative of $g(\theta)$:

$$g'(\theta) = \frac{f'(\theta)}{f(\theta)}$$

Setting the equation to 0:

$$\begin{aligned} g'(\theta) = \frac{f'(\theta)}{f(\theta)} &= 0 \\ 0 &= f'(\theta) \end{aligned}$$

Second derivative of $g(\theta)$:

$$\begin{aligned} g''(\theta) &= f''(\theta) * f(\theta)^{-1} + f'(\theta) * \theta * (-1) * (\theta)^{-2} * f'(\theta) \\ &= \frac{f''(\theta) * f(\theta) - f'(\theta)^2}{f(\theta)^2} \end{aligned}$$

$g''(\theta) < 0$ with $f'(\theta) = g'(\theta) = 0$

$$\begin{aligned} g''(\theta) &< 0 \\ \frac{f''(\theta) * f(\theta) - f'(\theta)^2}{f(\theta)^2} &< 0 \\ \frac{f''(\theta)}{f(\theta)} &< 0 \\ f''(\theta) &< 0 \end{aligned}$$

This shows, that the logarithm of a function preserves the maxima of the funtion.

Problem 3

We know that θ_{MAP} for a prior $Beta(a,b)$ and a likelihood function $Ber(\theta)$ is:

$$\theta_{MAP} = \frac{|T|+a-1}{|H|+|T|+a+b-2} \text{ (proved during lecture)}$$

We know $a = 6$, $b = 4$, and θ_{MAP} so all we need to do is find M, N such that:

$$0.75 = \frac{M+6-1}{M+N+6+4-2}$$

$$3N - M + 4 = 0$$

We notice that for $N = 1$, $M = 7$: $3 \times 1 - 7 + 4 = 0$ and, therefore, such 2 values could lead to a $\theta_{MAP} = 0.75$

Problem 4

The prior:

$$P(\theta) = Beta(a,b) = \frac{\gamma(a+b)}{\gamma(a)\cdot\gamma(b)} \cdot \theta^{a-1} \cdot (1-\theta)^{b-1}$$

The likelihood:

$$P(D|\theta) = \binom{N}{m} \cdot \theta^m \cdot (1-\theta)^{N-m}$$

At the same time, we know that:

$$P(\theta|D) \propto P(D|\theta) \cdot P(\theta) \propto \binom{N}{m}\theta^m \cdot (1-\theta)^{N-m} \cdot \frac{\gamma(a+b)}{\gamma(a)\cdot\gamma(b)} \cdot \theta^{a-1} \cdot (1-\theta)^{b-1}$$

By absorbing the constants:

$$P(\theta|D) \propto \theta^{m+a-1} \cdot (1-\theta)^{N-m+b-1} \text{ - looks similar to the PDF of a Beta distribution}$$

Therefore, the normalizing constant has the following format: $\frac{\gamma(\alpha+\beta)}{\gamma(\alpha)\cdot\gamma(\beta)}$. In our case, by pattern matching, we get:

$$\alpha = m + a$$

$$\beta = N - m + b$$

Therefore, the constant we're looking for is:

$$\frac{\gamma(a+b+N)}{\gamma(m+a)\cdot\gamma(N-m+b)}$$

Hence, our posterior distribution is

$$P(\theta|D) = Beta(m+a, N-m+b)$$

Having the distribution, we can now look up the posterior mean:

$$E[\theta|D] = \frac{m+a}{m+a+N-m+b} = \frac{m+a}{N+a+b}$$

The prior mean (since this is also a Beta distribution) is:

$$\mu_{prior} = \frac{a}{a+b}$$

All there is left to compute is θ_{MLE}:

$$\theta_{MLE} = argmax\binom{N}{m}\theta^m(1-\theta)^{N-m} \text{ which is equivalent with:}$$

$$\theta_{MLE} = argmax \log \binom{N}{m}\theta^m(1-\theta)^{N-m}$$

$$\theta_{MLE} = argmax(\log \binom{N}{m} + \log \theta^m + \log(1-\theta)^{N-m})$$

We denote:

$$f(\theta) = \log \binom{N}{m} + \log \theta^m + \log(1-\theta)^{N-m}$$

For getting θ_{MLE} we have to compute the derivative:

$f'(\theta) = (\log \theta^m)' + (\log(1-\theta)^{N-m})'$ which is the same expression as the one we solved for Problem 1.

$=0$

Therefore:

$$\theta_{MLE} = \frac{m}{m+N-m} = \frac{m}{N}$$

Now, we can show that:

$$E[\theta|D] = \lambda \cdot \mu_{prior} + (1-\lambda) \cdot \theta_{MLE}$$

$$\frac{m+a}{N+a+b} = \lambda \cdot \frac{a}{a+b} + (1-\lambda) \cdot \frac{m}{N}$$

$$\frac{a}{N+a+b} + \frac{m}{N+a+b} = \lambda \cdot \frac{a}{a+b} + (1-\lambda) \cdot \frac{m}{N}$$

We try to match each of the 2 terms from one side to the other:

$$N + a + b = \frac{1}{\lambda}(a+b)$$

$$N + a + b = \frac{N}{1-\lambda}$$

By solving both equations for λ, we get:

$$\lambda = \frac{a+b}{N+a+b}$$

Hence, we have found a λ for which the equality holds and, therefore, the posterior mean lies between the prior mean and θ_{MLE}.

exercise_03_notebook

November 3, 2019

1 Programming assignment 3: Probabilistic Inference

[41]:
```
import numpy as np
import matplotlib.pyplot as plt

from scipy.special import loggamma
%matplotlib inline
```

1.1 Your task

This notebook contains code implementing the methods discussed in `Lecture 3: Probabilistic Inference`. Some functions in this notebook are incomplete. Your task is to fill in the missing code and run the entire notebook.

In the beginning of every function there is docstring, which specifies the format of input and output. Write your code in a way that adheres to it. You may only use plain python and `numpy` functions (i.e. no scikit-learn classifiers).

1.2 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using `nbconvert` **Version 5.5 or later** by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 Simulating data

The following function simulates flipping a biased coin.

[42]:
```python
# This function is given, nothing to do here.
def simulate_data(num_samples, tails_proba):
    """Simulate a sequence of i.i.d. coin flips.

    Tails are denoted as 1 and heads are denoted as 0.

    Parameters
    ----------
    num_samples : int
        Number of samples to generate.
    tails_proba : float in range (0, 1)
        Probability of observing tails.

    Returns
    -------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    """
    return np.random.choice([0, 1], size=(num_samples), p=[1 - tails_proba,␣
↪tails_proba])
```

[43]:
```python
np.random.seed(123)  # for reproducibility
num_samples = 20
tails_proba = 0.7
samples = simulate_data(num_samples, tails_proba)
print(samples)
```

```
[1 0 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1]
```

2 Important: Numerical stability

When dealing with probabilities, we often encounter extremely small numbers. Because of limited floating point precision, directly manipulating such small numbers can lead to serious numerical issues, such as overflows and underflows. Therefore, we usually work in the **log-space**.

For example, if we want to multiply two tiny numbers a and b, we should compute $\exp(\log(a) + \log(b))$ instead of naively multiplying $a \cdot b$.

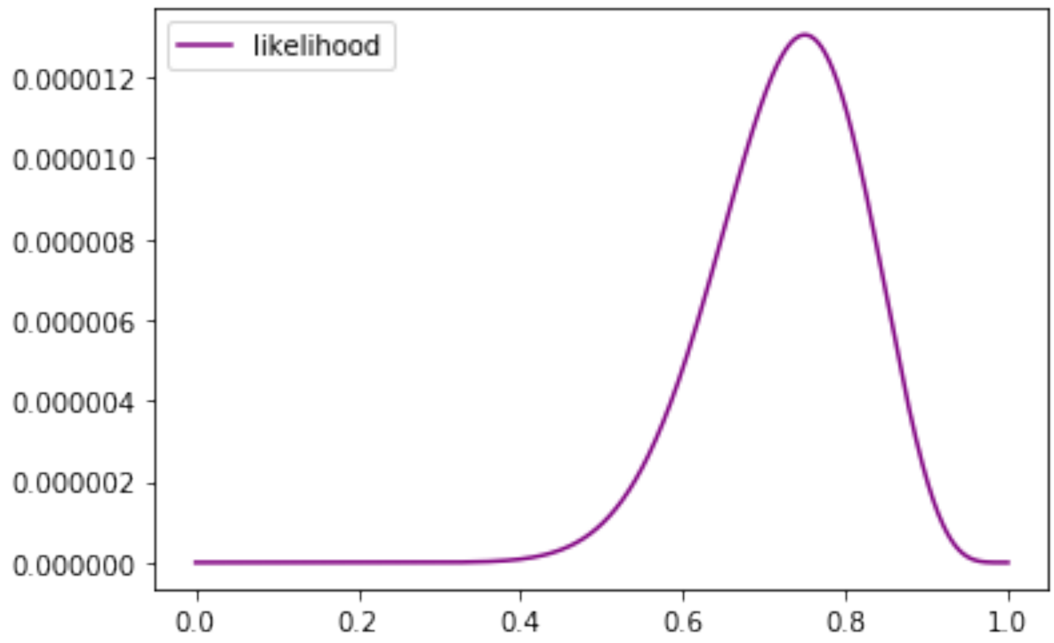
For this reason, we usually compute **log-probabilities** instead of **probabilities**. Virtually all machine learning libraries are dealing with log-probabilities instead of probabilities (e.g. Tensorflow-probability or Pyro).

2.1 Task 1: Compute $\log p(\mathcal{D} \mid \theta)$ for different values of θ

[44]:
```
def compute_log_likelihood(theta, samples):
    """Compute log p(D | theta) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-likelihood.
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.

    Returns
    -------
    log_likelihood : array, shape (num_points)
        Values of log-likelihood for each value in theta.
    """
    log_prob = np.log(np.outer(theta, samples) + np.outer(1 - theta, 1 -␣
↪samples))

    return np.sum(log_prob, axis=1)
```

[46]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)
log_likelihood = compute_log_likelihood(x, samples)
likelihood = np.exp(log_likelihood)
plt.plot(x, likelihood, label='likelihood', c='purple')
plt.legend()
```

[46]: <matplotlib.legend.Legend at 0x7f94b0ff9c10>



Note that the likelihood function doesn't define a probability distribution over θ — the integral $\int_0^1 p(\mathcal{D} \mid \theta) d\theta$ is not equal to one.

To show this, we approximate $\int_0^1 p(\mathcal{D} \mid \theta) d\theta$ numerically using the rectangle rule.

[47]:

```
# 1.0 is the length of the interval over which we are integrating p(D | theta)
int_likelihood = 1.0 * np.mean(likelihood)
print(f'Integral = {int_likelihood:.4}')
```

```
Integral = 3.068e-06
```

2.2 Task 2: Compute $\log p(\theta \mid a, b)$ for different values of θ

The function `loggamma` from the `scipy.special` package might be useful here. (It's already imported - see the first cell)

[48]:

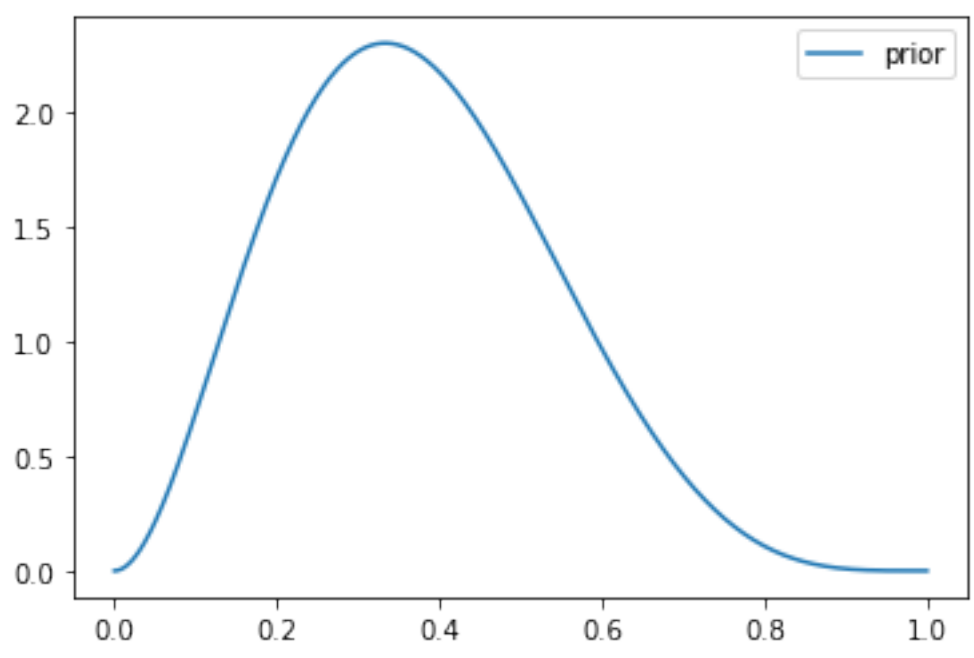
```
def compute_log_prior(theta, a, b):
    """Compute log p(theta | a, b) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-prior.
    a, b: float
        Parameters of the prior Beta distribution.
```

```
    Returns
    -------
    log_prior : array, shape (num_points)
        Values of log-prior for each value in theta.

    """
    ### YOUR CODE HERE ###
    B_ab = loggamma(a) + loggamma(b) - loggamma(a + b)

    return - B_ab + np.log(theta) * (a-1) + np.log(1 - theta) * (b-1)
```

Unlike the likelihood, the prior defines a probability distribution over θ and integrates to 1.

[49]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)
a, b = 3, 5

# Plot the prior distribution
log_prior = compute_log_prior(x, a, b)
prior = np.exp(log_prior)
plt.plot(x, prior, label='prior')
plt.legend()
```

[49]: `<matplotlib.legend.Legend at 0x7f94b0fb9390>`



[50]:
```
int_prior = 1.0 * np.mean(prior)
print(f'Integral = {int_prior:.4}')
```

```
Integral = 0.999
```

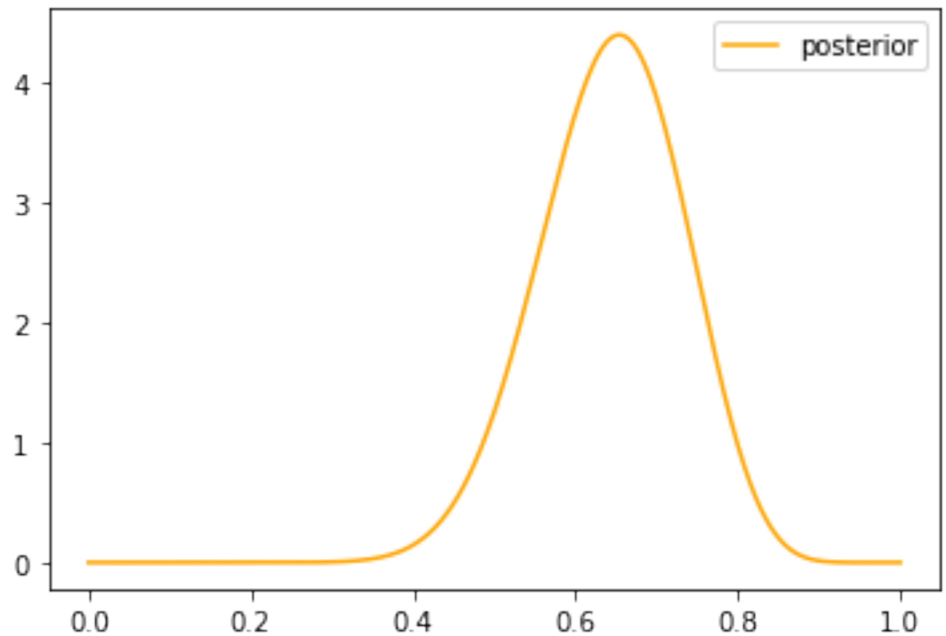
2.3 Task 3: Compute $\log p(\theta \mid \mathcal{D}, a, b)$ for different values of θ

The function `loggamma` from the `scipy.special` package might be useful here.

[51]:
```
def compute_log_posterior(theta, samples, a, b):
    """Compute log p(theta | D, a, b) for the given values of theta.

    Parameters
    ----------
    theta : array, shape (num_points)
        Values of theta for which it's necessary to evaluate the log-prior.
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    a, b: float
        Parameters of the prior Beta distribution.

    Returns
    -------
    log_posterior : array, shape (num_points)
        Values of log-posterior for each value in theta.
    """
    a = a + np.sum(samples)
    b = b + np.sum((1 - samples))

    return compute_log_prior(theta, a, b)
```

[52]:
```
x = np.linspace(1e-5, 1-1e-5, 1000)

log_posterior = compute_log_posterior(x, samples, a, b)
posterior = np.exp(log_posterior)
plt.plot(x, posterior, label='posterior', c='orange')
plt.legend()
```

[52]:
```
<matplotlib.legend.Legend at 0x7f94b12ac750>
```



Like the prior, the posterior defines a probability distribution over θ and integrates to 1.

[53]:
```
int_posterior = 1.0 * np.mean(posterior)
print(f'Integral = {int_posterior:.4}')
```

```
Integral = 0.999
```

2.4 Task 4: Compute θ_{MLE}

[54]:
```
def compute_theta_mle(samples):
    """Compute theta_MLE for the given data.

    Parameters
    ----------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.

    Returns
    -------
    theta_mle : float
        Maximum likelihood estimate of theta.
    """
    ### YOUR CODE HERE ###

    return np.mean(samples)
```

[55]:
```
theta_mle = compute_theta_mle(samples)
print(f'theta_mle = {theta_mle:.3f}')
```

```
theta_mle = 0.750
```

2.5 Task 5: Compute θ_{MAP}

[56]:
```
def compute_theta_map(samples, a, b):
    """Compute theta_MAP for the given data.

    Parameters
    ----------
    samples : array, shape (num_samples)
        Outcomes of simulated coin flips. Tails is 1 and heads is 0.
    a, b: float
        Parameters of the prior Beta distribution.

    Returns
    -------
    theta_map : float
        Maximum a posteriori estimate of theta.
    """
    ### YOUR CODE HERE ###
    return (np.sum(samples) + a - 1) / (samples.size + a + b - 2)
```

[57]:
```
theta_map = compute_theta_map(samples, a, b)
print(f'theta_map = {theta_map:.3f}')
```

```
theta_map = 0.654
```

3 Putting everything together

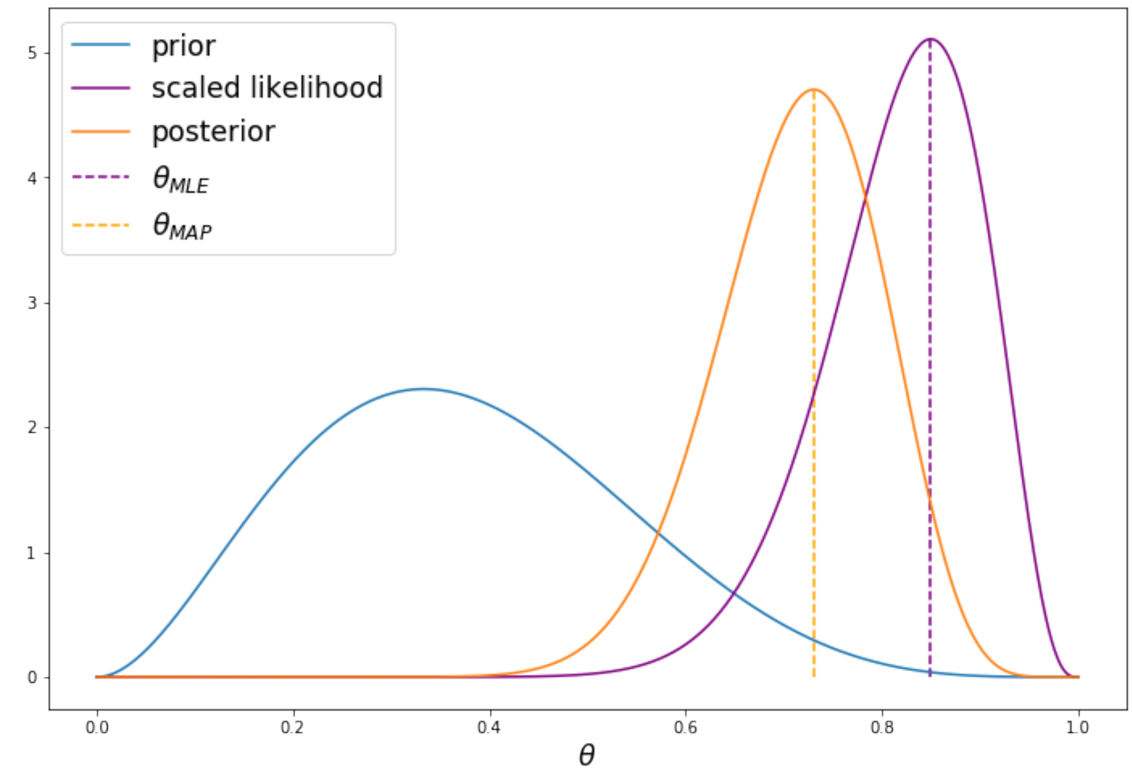
Now you can play around with the values of `a`, `b`, `num_samples` and `tails_proba` to see how the results are changing.

[58]:
```
num_samples = 20
tails_proba = 0.7
samples = simulate_data(num_samples, tails_proba)
a, b = 3, 5
print(samples)
```

```
[1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 1]
```

[59]:
```
plt.figure(figsize=[12, 8])
x = np.linspace(1e-5, 1-1e-5, 1000)
```

```
# Plot the prior distribution
log_prior = compute_log_prior(x, a, b)
prior = np.exp(log_prior)
plt.plot(x, prior, label='prior')

# Plot the likelihood
log_likelihood = compute_log_likelihood(x, samples)
likelihood = np.exp(log_likelihood)
int_likelihood = np.mean(likelihood)
# We rescale the likelihood - otherwise it would be impossible to see in the␣
↪plot
rescaled_likelihood = likelihood / int_likelihood
plt.plot(x, rescaled_likelihood, label='scaled likelihood', color='purple')

# Plot the posterior distribution
log_posterior = compute_log_posterior(x, samples, a, b)
posterior = np.exp(log_posterior)
plt.plot(x, posterior, label='posterior')

# Visualize theta_mle
theta_mle = compute_theta_mle(samples)
ymax = np.exp(compute_log_likelihood(np.array([theta_mle]), samples)) /␣
↪int_likelihood
plt.vlines(x=theta_mle, ymin=0.00, ymax=ymax, linestyle='dashed',␣
↪color='purple', label=r'$\theta_{MLE}$')


# Visualize theta_map
theta_map = compute_theta_map(samples, a, b)
ymax = np.exp(compute_log_posterior(np.array([theta_map]), samples, a, b))
plt.vlines(x=theta_map, ymin=0.00, ymax=ymax, linestyle='dashed',␣
↪color='orange', label=r'$\theta_{MAP}$')

plt.xlabel(r'$\theta$', fontsize='xx-large')
plt.legend(fontsize='xx-large')
plt.show()
```



prior
scaled likelihood
posterior
θ_{MLE}
θ_{MAP}
0
1
2
3
4
5
0.0
0.2
0.4
0.6
0.8
1.0
θ