Probabilistic Machine Learning

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Exercise 4

Summer Term 2023

Full Name	Matriculation No
Batuhan Özcömlekci	6300476
Aakarsh Nair	6546577

Eberhards Karls University of Tübingen, May 22, 2023

Theory:

$$p(r_{oil}, r_{air}) = N\left[\begin{pmatrix} r_{oil} \\ r_{air} \end{pmatrix}; \begin{pmatrix} 0.05 \\ 0.02 \end{pmatrix}, \underbrace{\begin{pmatrix} 0.090 & -0.045 \\ -0.045 & 0.040 \end{pmatrix}}_{\Sigma}\right]$$

$$R = x\, r_{oil} + (B-x)\, r_{air}$$

a) $$R = \underbrace{[x \;\; (B-x)]}_{A} \underbrace{\begin{bmatrix} r_{oil} \\ r_{air} \end{bmatrix}}_{z}$$

$$p(R) = p(Az) = \mathcal{N}(Az; A\mu, A\Sigma A^T)$$

$$= \mathcal{N}\left(R; 0.05x + 0.02(B-x), [x, B-x]\,\Sigma \begin{bmatrix} x \\ B-x \end{bmatrix}\right)$$

$$p(R) = \mathcal{N}\left(R; 0.05x + 0.02(B-x), [x, B-x] \begin{bmatrix} 0.125x - 0.045B \\ -0.085x + 0.040B \end{bmatrix}\right)$$

$$= \mathcal{N}(R; 0.05x + 0.02(B-x), 0.125x^2 - 0.045Bx + 0.085x^2 - 0.085Bx + 0.040B^2 - 0.040Bx)$$

$$= \mathcal{N}(R; \underbrace{0.03x + 0.02B}_{\mu_R}, \underbrace{0.210x^2 - 0.170Bx + 0.040B^2}_{\sigma_R^2})$$

b) $$E(R) = \int_{-\infty}^{\infty} p(R)\, R\, dR$$

$$\stackrel{*}{=} \mu_R = \underbrace{0.03x + 0.02B}_{} \Rightarrow x_R^* = 0.05B$$

* Expectation of a variable with Gaussian pdf is its mean

Maximization of this expression is possible with $x = B$ (i.e. all of the budget is spent on oil futures)

c) $$Var(R) = E(R^2) - E(R)^2 = E((R-\mu_R)^2) \stackrel{**}{=} \sigma_R^2$$

** By definition, the variance of a random variable with Gaussian pdf

$$\frac{\partial \sigma_R^2}{\partial x} = 0.42x - 0.17B = 0$$

$$0.42x = 0.17B$$

$$\boxed{x_0 = \frac{17}{42} B}$$

To minimize variance

d) We would like to maximize $E(R)$ given that

$Var(R) = E(R^2) - E(R)^2 \leq 0.03$

$\sigma_R^2 = 0.210\, x^2 - 0.170\, x + 0.040 \quad \leq 0.03$

$0.21\, x^2 - 0.17x + 0.04 + t^2 = 0.03$ where $t \geq 0$

$E(R) = \mu_R = 0.03x + 0.028 = 0.03x + 0.02$

We can find optimal x given the condition using the method of Lagrange multipliers. Imposing the condition:

$\mathcal{L}(x, \lambda) = 0.03x + 0.02 - \lambda(0.210x^2 - 0.170x + 0.01 + t^2)$

t^2 is for slackness introduced by the inequality

$$\frac{\partial \mathcal{L}}{\partial x} = 0.03 - 0.42\lambda x + 0.17\lambda = 0$$

$$x^* = \frac{17}{42} + \frac{3}{42\lambda}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = -0.21x^2 + 0.17x - 0.01 - t^2 = 0$$

$$0.21x^2 - 0.17x + 0.01 \leq 0$$

$$21x^2 - 17x + 1 \leq 0$$

$$x_{\mp} = \frac{17 \mp \sqrt{205}}{42}$$

Since λ is positive by our initial choice:

$$x^* = \frac{\sqrt{205} + 17}{42} \approx 0.746$$

e) Maximize the probability that the return is positive

$$P(R>0) = 1 - P(R<0)$$

Converting into normal distribution and writing as cumulative

$$P(R>0) = 1 - F\left(-\frac{\mu_R}{\sigma_R}\right)$$

$$\frac{dP}{dx} = -\underbrace{F'\left(-\frac{\mu_R}{\sigma_R}\right)}_{\text{Positive}}\left(\sigma_R \frac{d\mu_R}{dx} - \mu_R \frac{d\sigma_R}{dx}\right) \Big/ \underbrace{\sigma_R^2}_{\text{Positive}} = 0$$

Reduces to $\sigma_R \dfrac{d\mu_R}{dx} = \mu_R \dfrac{d\sigma_R}{dx}$

$$(0.21x^2 - 0.17Bx + 0.01B^2)(0.03) = (0.03x + 0.02B)(0.42x - 0.17B)$$

$$3(21x^2 - 17Bx + B^2) = (3x + 2B)(42x - 17B)$$

$$63x^2 - 51Bx + 12B^2 = 126x^2 - 51Bx + 84Bx - 34B^2$$

e) continued:

$\Rightarrow 63x^2 - 84Bx - 46B^2 = 0$

The equation has solutions $x_1 \approx 0.42B$

$x_2 \approx -1.75B$

since budget cannot be negative $x_1 = x^* = 0.42B$ is the optimal split.

f) $p'(r_{air} \mid \mu'_{air} = 0.03, \sigma'^2_{air} = 0.02) = \mathcal{N}(r_{air} \mid \mu'_{air} = 0.03, \sigma'^2_{air} = 0.04)$

$p(r_{oil} \mid r_{air}) = \dfrac{p(r_{air}, r_{oil})}{p(r_{air})}$ → First find the conditional

$= \mathcal{N}\left(r_{oil} ; \mu_{oil} + \sigma^2_{oa} \frac{1}{\sigma'^2_{air}} (r_{air} - \mu_{air}), \sigma^2_{oil} - \sigma^2_{oa} \frac{1}{\sigma'^2_{air}} \sigma^2_{oa}\right)$

where $\sigma^2_{oa} = \Sigma_{1,2} = \Sigma_{2,1} = -0.045$

$\mu_{oil} = 0.05$, $\mu_{air} = 0.02$

$\sigma^2_{oil} = 0.08$, $\sigma_{air}^2 = 0.04$

See Lecture 6 slides 30-31 for the formula

Inserting the values in their place:

$p(r_{oil} \mid r_{air}) = \mathcal{N}(r_{oil} ; 0.05 + (-0.045) \frac{1}{0.04} (r_{air} - 0.02), \backslash$

$0.08 - (0.045)^2 \frac{1}{0.04})$

$= \mathcal{N}(r_{oil} ; -1.125\, r_{air} + 0.0725, 0.029375)$

We know the conditional and the prior (new data)

Using the info in L8, slide 29:

If $p(r_{air}) = \mathcal{N}(r_{air} ; \mu'_{air}, \Sigma'_{air})$

and $p(r_{oil} \mid r_{air}) = \mathcal{N}(r_{oil} ; A r_{air} + b, \Lambda)$

then $p(r_{oil}) = \mathcal{N}(r_{oil} ; A\mu'_{air} + b, \Lambda + A \Sigma'_{air} A^T)$

$= \mathcal{N}(r_{oil} ; -1.125(0.03) + 0.0725, 0.029375 + (-1.125)^2 (0.04))$

$= \mathcal{N}(r_{oil} ; \underbrace{0.03875}_{\downarrow}, 0.08)$

Mean is the rise in oil futures

Exercise 4.2

May 22, 2023

1 Probabilistic Machine Learning

University of Tübingen, Summer Term 2023 © 2023 P. Hennig

1.1 Exercise Sheet No. 4 — Gaussians

Submission by:

- Batuhan, Oezcoemlekci, Matrikelnummer: 6300476
- Aakarsh, Nair, Matrikelnummer: 6546577

[ ]:
```
import jax.numpy as jnp
import logging

from jax import random
key = random.PRNGKey(0)
#from exponential_families import *
from tueplots import bundles
from matplotlib import pyplot as plt
plt.rcParams.update(bundles.beamer_moml())
plt.rcParams.update({'figure.dpi': 200})

logging.getLogger("matplotlib.font_manager").setLevel(logging.ERROR)
```

```
No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun for
more info.)
```

1.2 Exercise 4.2 (Coding Exercise)

> This week's Exercise is directly taken from Exercise 24.3. of David JC MacKay's *Information Theory, Inference, and Learning Algorithms*. (But don't waste your time trying to find an answer there :)

The terribly important quantity μ has been experimentally measured by seven scientists (A, B, C, D, E, F, G) with wildly differing experimental skills. They have reported the following measurements:

[ ]:
```
# We assume the same latent quantity $\mu$ for all observations.
X = jnp.array([-27.020,3.570,8.191,9.898,9.603,9.945,10.056])
```

[ ]:
```
fig,ax = plt.subplots(1,1)
#ax.plot(X,random.uniform(key, shape=(7,)),'x',ms=3)
ax.plot(X,range(1,8),'x',ms=3)

ax.yaxis.set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['left'].set_visible(False)
ax.set_xlabel('measured value of $x$')
```

[ ]:
```
Text(0.5, 0, 'measured value of $x$')
```

We assume that they have all, independently of each other, made an unbiased Gaussian measurement of μ:

$$p(x \mid \mu, \sigma) = \prod_{i=1}^{7} \mathcal{N}(x_i; \mu, \sigma_i^2).$$

But we have to assume that their measurement errors σ_i vary a lot (some are skilled experimentalists, others are unqualified).

Task A: Implement the likelihood above as a single jax function (this is unfortunately a case where our neat `ExponentialFamily` base class is more awkward than useful). Try using a numerical optimizer to find *maximum likelihood estimators*, i.e. points $(\mu, \vec{\sigma})$ that maximize this function. Alternatively, you can try and identify such points directly by inspecting the likelihood by hand.

You probably agree that, intuitively, it looks pretty certain that A and B are both inept measurers, that D–G are better, and that the true value of μ is somewhere close to 10. Are your findings consistent with this intuition?

[ ]:
```
import jax
import jax.numpy as jnp
import jax.scipy.optimize as optimize
```

```
# Step 1: Define log-likelihood function
def log_likelihood(params, data):
    mean = params[0]
    vars = params[1:]
    log_probs = [0  for _ in range(data.shape[0])]
    for i in range(data.shape[0]):
        log_probs[i] = jax.scipy.stats.norm.logpdf(data[i], loc=mean,␣
 ↪scale=vars[i])
    return -jnp.sum(jnp.array(log_probs))

def log_likelihood_alt(params, data):
    mean = params[0]
    vars = params[1:]
    log_probs = [0  for _ in range(data.shape[0])]
    for i in range(data.shape[0]):
        log_probs[i] = jnp.log(1/(jnp.sqrt(2*jnp.pi*vars[i]))) -␣
 ↪(data[i]-mean)**2/(2*vars[i])
    return -jnp.sum(jnp.array(log_probs))

def log_likelihood_alt2(params, data):
    mean = params[0]
    vars = params[1:]
    log_probs = jnp.log(1/(jnp.sqrt(2*jnp.pi)*vars)) - (data-mean)**2/(2*vars)
    return -jnp.sum(log_probs)

#initial_params = jnp.array([10.0, 10.0-X[0], 10-X[1], 10-X[2], 10-X[3],␣
 ↪10-X[4], 10-X[5], 10-X[6]])  # Initial guess for mean and standard deviation
initial_params = jnp.array([10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0])  #␣
 ↪Initial guess for mean and standard deviation

# Minimize the negative log-likelihood
result = optimize.minimize(log_likelihood, initial_params, args=(X,),␣
 ↪method='BFGS')
estimated_params = result.x

print("Estimated Mean:", estimated_params[0])
print("Estimated Standart Deviations:",jnp.sqrt(estimated_params[1:8]))
```

```
Estimated Mean: 8.281892
Estimated Standart Deviations: [6.670353  1.2636932 0.5205992 1.1591171
1.0567701 1.1756024 1.2146856]
```

[ ]:
```
print('Sample mean:', jnp.mean(X))
```

```
Sample mean: 3.4632857
```

MLE minimizer does not yield intuitive results because of the first two data points A and B

Or alternatively the impact of A and B and their variances to mean can also be realized by:

$$L \equiv \log p\left(x \mid \ ,\ \right) = -\sum_i \log \sigma_i - \sum_i \frac{\left(x_i - \mu\right)^2}{2\sigma_i^2} \frac{\partial L}{\partial \mu} = \sum_i \left(\frac{x_i - \mu}{\sigma_i^2}\right) = \sum_i \frac{x_i}{\sigma_i^2} - \mu \sum_i \frac{1}{\sigma_i^2} = 0 \mu = \frac{\sum_i \frac{x_i}{\sigma_i^2}}{\sum_i \frac{1}{\sigma_i^2}}$$

Task B: We will now instead provide a Bayesian answer. Let the prior on each σ_i^{-2} be a broad Gamma distribution, i.e. the distribution

$$\mathcal{G}(z; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} z^{\alpha-1} e^{-\beta z},$$

and

$$p(\sigma) = \prod_{i=1}^{7} \mathcal{G}(\sigma_i^{-2}; \alpha, \beta),$$

with, say, $\alpha = 1$, $\beta = 0.1$.

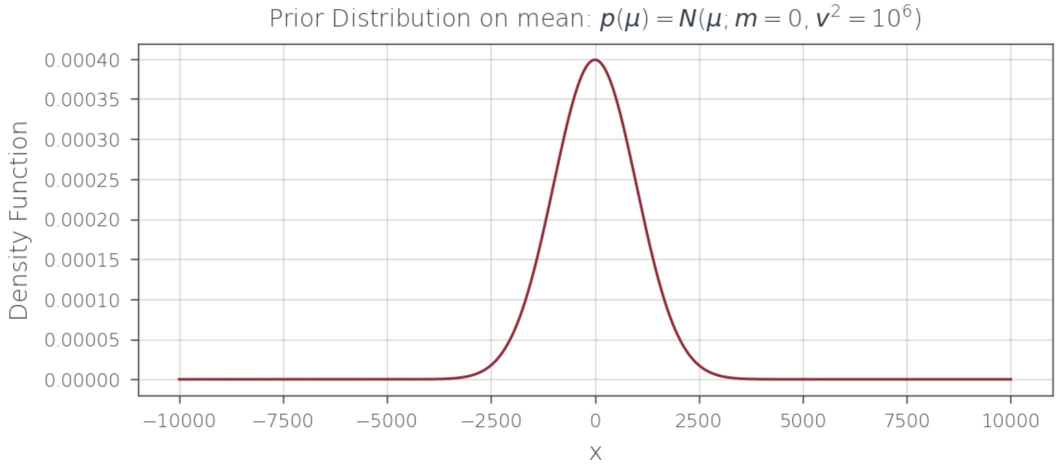
Let the prior for μ be a broad Gaussian $p(\mu) = \mathcal{N}(\mu; m, v^2)$ with mean $m = 0$ and standard deviation $v = 10^3$.

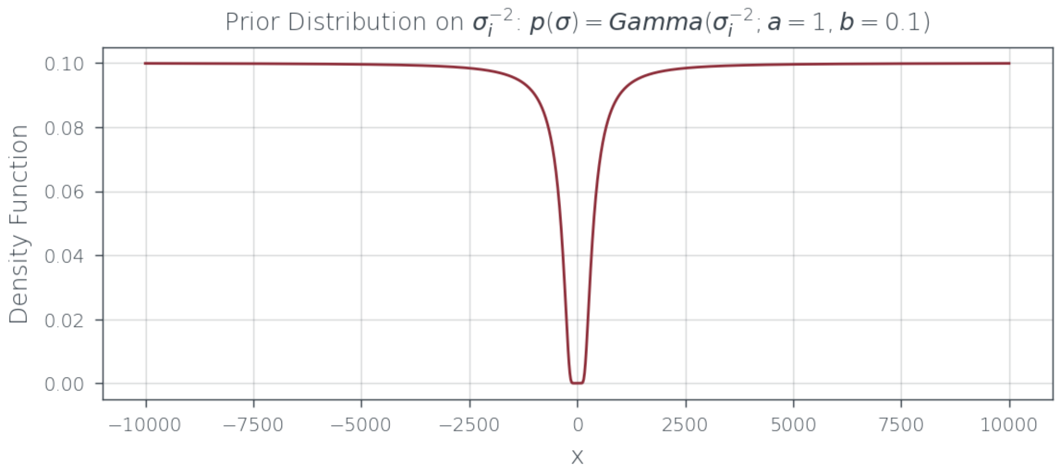
Find the posterior for μ. Plot this posterior for μ, both for the data given above and for $x = \{13.01, 7.39\}$.

Hint: First, remember that the Gamma is the conjugate prior for the Gaussian with fixed mean μ. The marginal $p(x \mid \mu)$ can thus be computed using the `log_marginal_pdf` function you implemented generically for exponential families last week, and which has an analytic form. Then use Bayes' theorem a second time to find $p(\mu \mid x)$ up to normalization, by directly multiplying the prior for μ and the marginal likelihood terms you just found.

[ ]:
```
# Define the parameters of the normal distribution
m = 0.0  # Mean
v = 10**3.0  # Standard deviation

# Define the parameters of the gamma distribution
alpha = 1.0
beta = 0.1

# Data
X_1 = X
X_2 = jnp.array([13.01, 7.39])

def plot(x, pdf, title):
    #plt.figure(figsize=(10, 5))
    plt.plot(x, pdf)
    plt.xlabel('x')
    plt.ylabel('Density Function')
    plt.title(f'{title}')
    plt.grid(True)
    plt.show()
```

```
def Gamma_dist(x, alpha, beta):
    return (beta**alpha)/jax.scipy.special.gamma(alpha)*x**(alpha-1)*jnp.
 ↪exp(-beta*x)

def gaussian_on_mu(mu, m, v):
    # v is std
    return jax.scipy.stats.norm.pdf(mu, loc=m, scale=v)

def gamma_on_sigma(sigma, alpha, beta, use_jax=True):
    if use_jax:
        return jax.scipy.stats.gamma.pdf(1/(sigma**2), a=alpha, scale=1/beta)
    return Gamma_dist(1/(sigma**2), alpha, beta)

# Create a range of x values.
x = jnp.linspace(m-10000, m+10000, 1000)

pdf_prior_mu = gaussian_on_mu(x, m, v)
plot(x, pdf_prior_mu, 'Prior Distribution on mean: $p(\mu)= N(\mu; m=0,␣
 ↪v^2=10^6)$')

sigma = jnp.linspace(-10, 10, 1000)

pdf_prior_sigma = gamma_on_sigma(sigma, alpha, beta)
plot(x, pdf_prior_sigma, 'Prior Distribution on $\sigma_i^{-2}$: $p(\sigma)=␣
 ↪Gamma(\sigma_i^{-2}; a=1, b=0.1)$')
```





Bayesian for the posterior probability of σ:

$$P(\sigma \mid x, \mu) = \frac{P(x \mid \sigma, \mu)\, P(\sigma)}{P(x \mid \mu)}$$

where

$$P(x \mid \sigma, \mu) = \left(\frac{1}{2\pi}\right)^{N/2} \left(\Pi_n \frac{1}{\sigma_n}\right) \exp\left(-\sum_n \frac{(x_n - \mu)^2}{2\sigma_n^2}\right)$$

$$P(\sigma) = \left(\frac{\beta^\alpha}{\Gamma(\alpha)}\right)^N \Pi_n (\sigma_n)^{\alpha-1} \exp\left(-\sum_n \beta\sigma_n\right)$$

where $(\beta, \alpha) = (0.1, 1)$

And the normalizing constant is

$$P(x \mid \mu) = \int_0^\infty P(x \mid \sigma, \mu)\, \Pi_n P(\sigma_n)\, d\sigma$$

The posterior probability of μ:

$$P(\mu \mid x) = \frac{P(x \mid \mu)\, P(\mu)}{P(x)}$$

and the prior is determined by $P(\mu) = \mathcal{N}(\mu; m, v^2)$ and the normalizing constant is

$$P(x) = \int_{-\infty}^{\infty} d\mu P(x \mid \mu)\, P(\mu)$$

Note that, during coding, integrals will be converted into summations for the sake of discreteness.

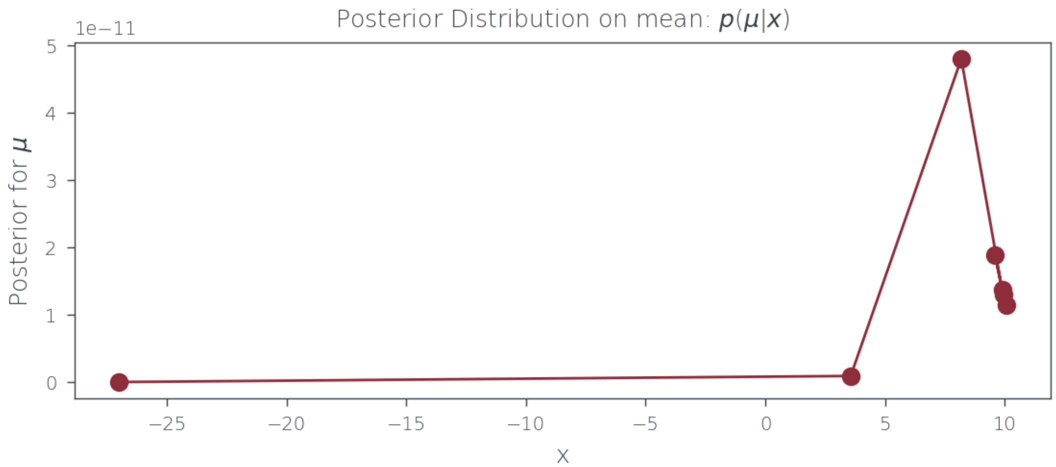
1.2.1 Strategy

- First using the first Bayesian equation, obtain $P(x|\mu)$ which is the normalization term.
- Then using the second Bayesian, obtain $P(\mu|x)$

```
[ ]: X = jnp.array(X_1)

result = optimize.minimize(log_likelihood, initial_params, args=(X,),␣
 ↪method='BFGS')
estimated_params = result.x

mean = estimated_params[0]
std_deviations = jnp.sqrt(estimated_params[1:X.shape[0]+1])

#print('Mean: ', mean)
#print('Stds: ', std_deviations)

def p_x_given_mu_sigma_single(x, mean, std_deviation):
    return jax.scipy.stats.norm.pdf(x, loc=mean, scale=std_deviation)

def p_x_given_mu_sigma(x, mean, std_deviations):
    #use likelihood estimates
    log_product = 0
    for i in range(x.shape[0]):
        log_product += jnp.log(p_x_given_mu_sigma_single(x[i], mean,␣
 ↪std_deviations[i]))
    return jnp.exp(log_product)


def p_sigma(std_deviations, alpha, beta):
    prob = 0
    for i in range(std_deviations.shape[0]):
        prob += jnp.log(gamma_on_sigma(std_deviations[i], alpha, beta))
    return jnp.exp(prob)

#def p_sigma_given_x_mu(std_deviations, alpha, beta, x, mean):
#    log_product = 0
#    for i in range(std_deviations.shape[0]):
#        log_product += jnp.log(p_x_given_mu_sigma(x[i], mean,␣
 ↪std_deviations[i])) + jnp.log(gamma_on_sigma(std_deviations[i], alpha,␣
 ↪beta))


def p_x_given_mu(x, mean, std_deviations, alpha=1, beta=0.1):
    normalization = jnp.zeros(x.shape[0])
    prior_sigma = p_sigma(std_deviations, alpha, beta)
    for i in range(x.shape[0]):
        normalization += p_x_given_mu_sigma_single(x, mean, std_deviations[i])␣
 ↪* prior_sigma
    return normalization

#def p_x(x):
```

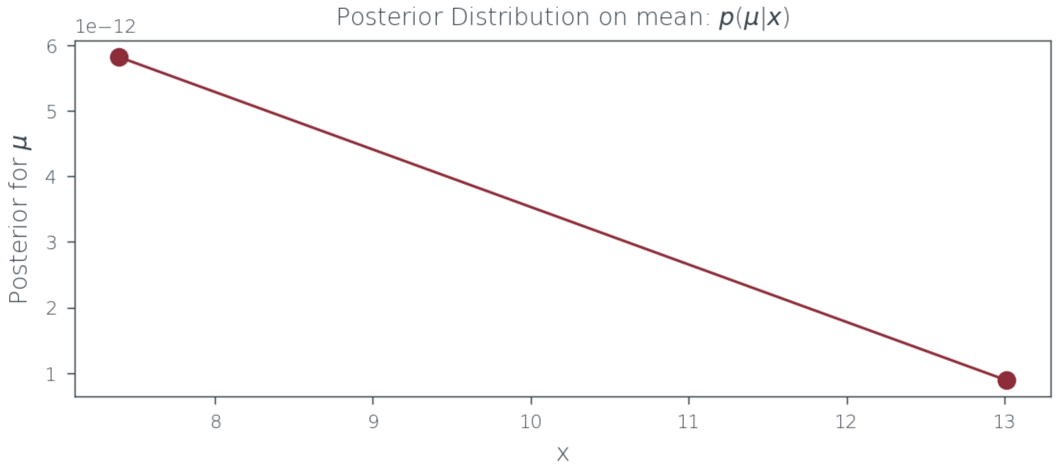
```
#    return jnp.ones(x.shape[0]) / x.shape[0]

def p_mu_given_x(x, mean, std_deviations, m=0, v=10**3):
    dist = p_x_given_mu(x, mean, std_deviations) * gaussian_on_mu(mean, m, v)
    return dist

posterior = p_mu_given_x(X, mean, std_deviations)
#print('Posterior means given x:', posterior)
# Plot posterior
plt.plot(X, posterior, marker="o")
plt.xlabel('x')
plt.ylabel('Posterior for $\mu$')
plt.title('Posterior Distribution on mean: $p(\mu|x)$')
```

[ ]: Text(0.5, 1.0, 'Posterior Distribution on mean: $p(\\mu|x)$')



[ ]:
```
X = jnp.array(X_2)
result = optimize.minimize(log_likelihood, initial_params, args=(X,),␣
 ↪method='BFGS')
estimated_params = result.x

posterior = p_mu_given_x(X, mean, std_deviations)
#print('Posterior means given x:', posterior)
# Plot posterior
plt.plot(X, posterior, marker="o")
plt.xlabel('x')
plt.ylabel('Posterior for $\mu$')
plt.title('Posterior Distribution on mean: $p(\mu|x)$')
```

[ ]: Text(0.5, 1.0, 'Posterior Distribution on mean: $p(\\mu|x)$')



Posterior Distribution on mean: $p(\mu|x)$
1e−12
6
5
4
3
2
1
Posterior for μ
8
9
10
11
12
13
X