# CS-E5710 Bayesian Data Analysis
# Assignment 4

October 6, 2019

## 1 Bioassay model and importance sampling

**a**) Report the mean and covariance of the bivariate normal distribution

```
mu_a = 0
sigma_a = 2
mu_b = 10
sigma_b = 10
corr = 0.5

mean = np.array([mu_a, mu_b])
print('mean:',mean)

cov = np.array([[sigma_a**2, corr * sigma_a * sigma_b],
         [corr * sigma_a * sigma_b, sigma_b**2]])
print('covariance:',cov)
```

```
mean: [ 0 10]
covariance: [[  4.   10.]
          [ 10. 100.]]
```

**b**) Implement a function for computing the logarithm of the density of the prior distribution

```
def p_log_prior(alpha, beta):

    prior= stats.multivariate_normal(mean, cov)
    pos = np.dstack((alpha, beta))
    log_prior = prior.logpdf(pos)

    return log_prior

print('test prior:',p_log_prior(3,9))
```

```
test prior: -6.296434970404113
```

**c)** Implement a function for computing the logarithm of the density of the posterior

$$log(posterior) = log(prior \times likelihood) = log(prior) + log(likelihood) \quad (1)$$

where $log(likelihood)$ is given in part c), $log(likelihood)$ is given in the BDA repository on GitHub. According to the Chapter3.7 Example: analysis of a bioassay experiment, we could implement $p\_log\_posterior$ as following:

```python
dose = np.array([-0.86, -0.30, -0.05, 0.72])
deaths = np.array([0, 1, 3, 5])
animals = np.array([5, 5, 5, 5])

def p_log_posterior(alpha, beta, x, y, n):
#logarithm of the density of the prior distribution
    prior= stats.multivariate_normal(mean, cov)
    pos = np.dstack((alpha, beta))
    log_prior = prior.logpdf(pos)

#logarithm of the density of the likelihood distribution
    alpha = np.expand_dims(alpha, axis=-1)
    beta = np.expand_dims(beta, axis=-1)
    t = alpha + beta*x
    et = np.exp(t)
    z = et/(1.+et)
    log_likelihood = np.sum(y*np.log(z)+ (n-y)*np.log(1.0-z), axis=-1)
##logarithm of the density of the posterior distribution
    log_posterior = log_prior + log_likelihood
    return log_posterior

print('test posterior',p_log_posterior(3, 9, dose, deaths, animals))
```

```
test posterior: -15.788012556775058
```

**d)** Plot the posterior density in a grid of points $\alpha \in [-4, 4],\ \beta \in [-10, 30]$

```python
alpha, beta = np.meshgrid(np.linspace(-4,4,100),np.linspace(-10,30,100))
posterior = np.exp(p_log_posterior(alpha, beta, dose, deaths, animals))

plt.contourf(alpha, beta,posterior, cmap=plt.cm.Greys)
plt.xlabel('alpha')
plt.ylabel('beta')
plt.title('Posterior Distribution')
plt.grid(linewidth=0.8, alpha=0.2)
plt.colorbar(plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys))
plt.savefig('./log_posterior.png')
plt.show()
```

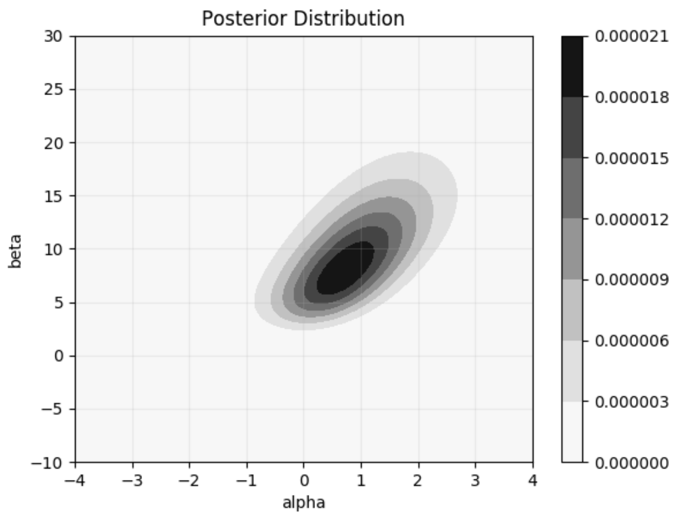

Figure 1: Heatmap of the posterior density

**e)** Sample draws of alpha and beta from the prior distribution

```
prior = stats.multivariate_normal(mean, cov)
samples = prior.rvs(10000)
```

Compute the importance ratios (importance weights) for each draw when the target distribution is the posterior distribution.

$$\begin{aligned} w &= \theta_i^{y_i}(1-\theta_i)^{n_i-y_i} \\ \theta_i &= \frac{1}{1+exp(-\alpha+\beta x_i)} \end{aligned} \tag{2}$$

Normalize the weights so that they sum to 1.

$$\tilde{\mathbf{w}}(\theta^s) = \frac{\mathbf{w}(\theta^s)}{\sum_{s'=1}^{S}\mathbf{w}(\theta^{s'})} \tag{3}$$

```
theta =1/(1+np.exp(-(samples[:,0,None] + samples[:,1,None]*dose)))
weights = np.prod(
    theta**deaths * (1 - theta)**(animals - deaths), axis=1)

#Normalize the weights
weights_norm = (weights) / np.sum(weights)
```

**f)** Compute the posterior mean using importance sampling and draws from e)

```python
posterior_mean = sum(weights[ : , None] * samples) / sum(weights)
print('posterior mean of alpha : ', posterior_mean[0])
print('posterior mean of beta  : ', posterior_mean[1])
```

```
posterior mean of alpha: 0.96466253
posterior mean of beta: 10.48528471
```

**g)** Compute the effective sample size

$$
\begin{aligned}
S_{eff} &= \frac{1}{\sum_{s=1}^{S}(\tilde{\mathbf{w}}(\theta^s))^2} \\
\tilde{\mathbf{w}}(\theta^s) &= \frac{\mathbf{w}(\theta^s)}{\sum_{s'=1}^{S}\mathbf{w}(\theta^{s'})}
\end{aligned} \tag{4}
$$

```python
s_eff = 1 / np.sum(weights_norm**2)
print('effective sample size: ', s_eff)
```

```
effective sample size: 2750.282727444427
```

**h)** Use importance resampling without replacement to obtain a posterior sample of size 1000 of alpha and beta

```python
scode = np.random.choice(a=10000,size=1000,replace=False,p=weights_norm)
resamples = samples[scode]
print('mean of resampled alpha: ', np.mean(resamples[:, 0]))
print('mean of resampled beta: ', np.mean(resamples[:, 1]))
```

```
mean of resampled alpha: 0.9247278915535617
mean of resampled beta: 10.650671142369971
```

Plot a scatterplot of the obtained posterior sample

```python
plt.xlim([-4, 4])
plt.ylim([-10, 30])
plt.xlabel('alpha')
plt.ylabel('beta')
plt.grid(linewidth=0.8, alpha=0.2)
plt.scatter(resamples[:, 0], resamples[:, 1],8,color='grey')
plt.title('Posterior Samples')
plt.savefig('./posterior_samples.png')
plt.show()

plt.xlim([-4, 4])
plt.ylim([-10, 30])
plt.xlabel('alpha')
plt.ylabel('beta')
plt.grid(linewidth=0.8, alpha=0.2)
plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys)
plt.colorbar(plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys))
plt.scatter(resamples[:, 0], resamples[:, 1], 8, alpha=.15, color='grey')
plt.title('Contourf & Ccatter Comparision')
plt.savefig('./contourf_scatter.png')
plt.show()
```

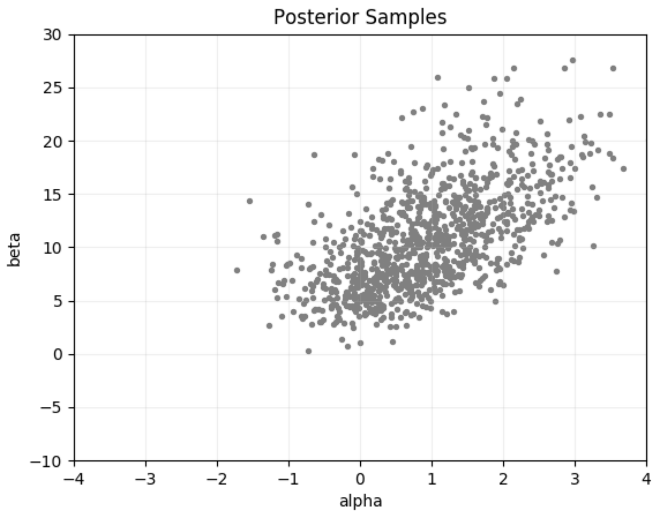

Figure 2: Scatterplot of the obtained posterior sample

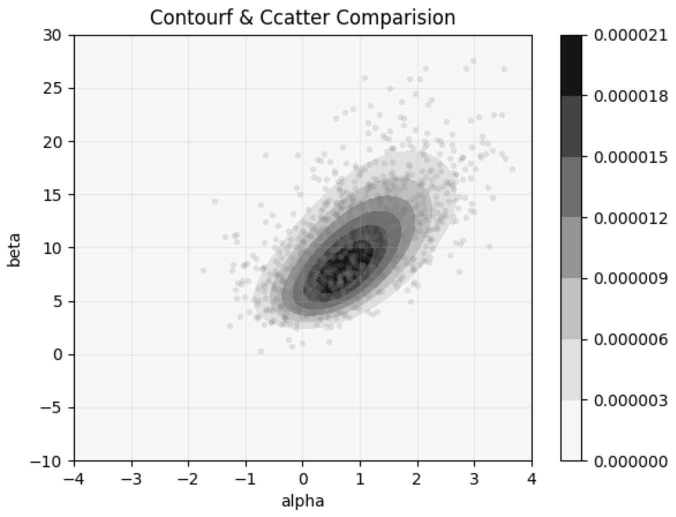

Figure 3: Comparison of Scatterplot and Heatmap

**i)** Using the posterior sample obtained via importance resampling, report an estimate for $p(\beta > 0|x, n, y)$, i.e., the probability that the drug is harmful.

```python
beta_resample = resamples[:, 1]
alpha_resample = resamples[:, 0]
pos = beta_resample > 0
p_harmful = (beta_resample[pos].size/(beta_resample.size + 1))
print('Probability that the drug is harmful:', p_harmful)
```

```
Probability that the drug is harmful: 0.999000999000999
```

**j)** Using the posterior sample obtained via importance resampling, draw a histogram of the draws from the posterior distribution of the LD50 conditional on $\beta > 0$

LD50 is $x_i = -\frac{\alpha}{\beta}$

```python
ld50 = - alpha_resample[pos]/beta_resample[pos]
y = np.arange(-0.4, 0.4, 0.01)
plt.hist(ld50, y, ec='white', color='grey')

plt.xlabel('LD50')
plt.title('Histogram')
plt.savefig('./histogram.png')
plt.show()
```

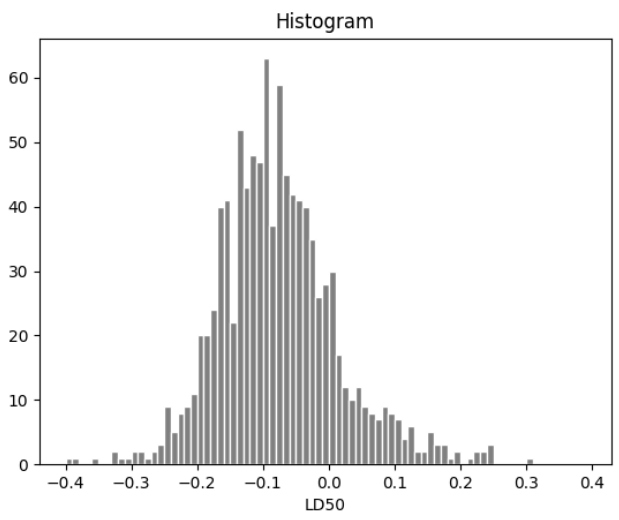

Figure 4: Histogram

## A Code

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy import stats
import numpy as np

def bioassaylp(a, b, x, y, n):
    # last axis for the data points
    a = np.expand_dims(a, axis=-1)
    b = np.expand_dims(b, axis=-1)
    # these help using chain rule in derivation
    t = a + b*x
    et = np.exp(t)
    z = et/(1.+et)
    # negative log posterior (error function to be minimized)
    lp = np.sum(y*np.log(z)+ (n-y)*np.log(1.0-z), axis=-1)
    return lp
'''
a)
'''
mu_a = 0
sigma_a = 2
mu_b = 10
sigma_b = 10
corr = 0.5

mean = np.array([mu_a, mu_b])
print('mean:',mean)

cov = np.array([[sigma_a**2, corr * sigma_a * sigma_b],
[corr * sigma_a * sigma_b, sigma_b**2]])
print('covariance:',cov)

alpha,beta = np.meshgrid(np.linspace(-4,4,100), np.linspace(-10,30,100))

'''
b)
'''

def p_log_prior(alpha, beta):

    prior= stats.multivariate_normal(mean, cov)
    pos = np.dstack((alpha, beta))
    log_prior = prior.logpdf(pos)

    return log_prior
#print('test',p_log_prior(3,9))

dose = np.array([-0.86, -0.3, -0.05, 0.72])
deaths = np.array([0, 1, 3, 5])
animals = np.array([5, 5, 5, 5])

'''
c)
'''
```

```

def p_log_posterior(alpha, beta, x, y, n):
    prior= stats.multivariate_normal(mean, cov)
    pos = np.dstack((alpha, beta))
    log_prior = prior.logpdf(pos)

    alpha = np.expand_dims(alpha, axis=-1)
    beta = np.expand_dims(beta, axis=-1)
    t = alpha + beta*x
    et = np.exp(t)
    z = et/(1.+et)
    log_likelihood = np.sum(y*np.log(z)+ (n-y)*np.log(1.0-z), axis=-1)

    log_posterior = log_prior + log_likelihood

    return log_posterior
#print('testposterior',p_log_posterior(3, 9, dose, deaths, animals))

'''
d)
'''
alpha,beta = np.meshgrid(np.linspace(-4,4,100), np.linspace(-10,30,100))
posterior = np.exp(p_log_posterior(alpha, beta, dose, deaths, animals))
plt.contourf(alpha, beta,posterior, cmap=plt.cm.Greys)
plt.xlabel('alpha')
plt.ylabel('beta')
plt.title('Posterior Distribution')
plt.grid(linewidth=0.8, alpha=0.2)
plt.colorbar(plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys))
plt.savefig('./log_posterior.png')
plt.show()

'''
e) 2. Sample draws of alpha and beta from the prior distribution.
'''
prior = stats.multivariate_normal(mean, cov)
samples = prior.rvs(10000)
#print('Shape of the samples from prior: ', samples.shape)

theta = 1 / (1+np.exp(-(samples[:,0,None] + samples[:,1,None] * dose)))
weights = np.prod(
    theta**deaths * (1 - theta)**(animals - deaths), axis=1)

weights_norm = (weights) / np.sum(weights)
#print('Shape of the weights of the likelihood: ', weights.shape)

'''
f)
'''
posterior_mean = sum(weights[ : , None] * samples) / sum(weights)
print('posterior mean of alpha : ', posterior_mean[0])
print('posterior mean of beta  : ', posterior_mean[1])

'''
g)
'''
s_eff = 1 / np.sum(weights_norm**2)
```

```python
print('effective sample size: ', s_eff)

'''
h)
'''
scode = np.random.choice(a=10000,size=1000,replace=False,p=weights_norm)
resamples = samples[scode]
print('mean of resampled alpha: ', np.mean(resamples[:, 0]))
print('mean of resampled beta: ', np.mean(resamples[:, 1]))

plt.xlim([-4, 4])
plt.ylim([-10, 30])
plt.xlabel('alpha')
plt.ylabel('beta')
plt.grid(linewidth=0.8, alpha=0.2)
plt.scatter(resamples[:, 0], resamples[:, 1],8,color='grey')
plt.title('Posterior Samples')
plt.savefig('./posterior_samples.png')
plt.show()

plt.xlim([-4, 4])
plt.ylim([-10, 30])
plt.xlabel('alpha')
plt.ylabel('beta')
plt.grid(linewidth=0.8, alpha=0.2)
plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys)
plt.colorbar(plt.contourf(alpha, beta, posterior, cmap=plt.cm.Greys))
plt.scatter(resamples[:, 0], resamples[:, 1], 8, alpha=.15, color='grey')
plt.title('Contourf & Ccatter Comparision')
plt.savefig('./contourf_scatter.png')
plt.show()

'''
i)
'''
beta_resample = resamples[:, 1]
alpha_resample = resamples[:, 0]
pos = beta_resample > 0
p_harmful = (beta_resample[pos].size/(beta_resample.size + 1))
print('Probability that the drug is harmful:', p_harmful)

'''
j)
'''
ld50 = - alpha_resample[pos]/beta_resample[pos]
y = np.arange(-0.4, 0.4, 0.01)
plt.hist(ld50, y, ec='white', color='grey')
plt.xlabel('LD50')
plt.title('Histogram')
plt.savefig('./histogram.png')
plt.show()
```