Statistical Machine Learning
Assignment 2

Inez Wijnands (s4149696) & Guido Zuidhof (s4160703)
Radboud University Nijmegen
04/11/2015

The entire code listing is in a separate file. The listings shown here are merely code snippets.

1 Sequential learning

1.1 Obtaining the prior

1.

$$\tilde{\boldsymbol{\Lambda}}_{a,b} = \tilde{\boldsymbol{\Sigma}}_{a,b}^{-1} \tag{1.1}$$

$$= \left(\begin{array}{cc|cc} 60 & 50 & -48 & 38 \\ 50 & 50 & -50 & 40 \\ \hline -48 & -50 & 52.4 & -41.4 \\ 38 & 40 & -41.4 & 33.4 \end{array}\right) \tag{1.2}$$

Using the precision matrix $\tilde{\boldsymbol{\Lambda}}$ we can use equations 2.69, 2.73 and 2.75 from Bishop to obtain the mean and covariance of the conditional distribution $p([x_1, x_2]^T | x_3 = x_4 = 0)$.

$$\boldsymbol{\Sigma}_p = \boldsymbol{\Lambda}_{aa}^{-1} \tag{Bishop 2.73}$$

$$\boldsymbol{\Lambda}_{aa} = \begin{pmatrix} 60 & 50 \\ 50 & 50 \end{pmatrix} \tag{1.3}$$

$$\boldsymbol{\Sigma}_p = \begin{pmatrix} 0.1 & -0.1 \\ -0.1 & 0.12 \end{pmatrix} \tag{1.4}$$

$$\boldsymbol{\mu}_p = \boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b) \tag{Bishop 2.75}$$

We can fill in this equation, since $\tilde{\boldsymbol{\mu}}$ and $\boldsymbol{x}_b$ (the second partition of $\boldsymbol{x}$) are known.

$$\boldsymbol{\mu}_p = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 60 & 50 \\ 50 & 50 \end{pmatrix}^{-1} \begin{pmatrix} -48 & 38 \\ -50 & 40 \end{pmatrix} (\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 2 \end{pmatrix}) \tag{1.5}$$

$$= \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.1 & -0.1 \\ -0.1 & 0.12 \end{pmatrix} \begin{pmatrix} -48 & 38 \\ -50 & 40 \end{pmatrix} (\begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 2 \end{pmatrix}) \tag{1.6}$$

$$= \begin{pmatrix} 0.8 \\ 0.8 \end{pmatrix} \tag{1.7}$$

2. Using the prior $\boldsymbol{\mu}_p$ and $\boldsymbol{\Sigma}_p$, we used the numpy-equivalent in Python for the MATLAB-function `mvnrnd` to obtain the $\boldsymbol{\mu}_t$ we used for the remainder of this assignment:

```
np.random.multivariate_normal(mu_p, sigma_p, 1)
```

This resulted in:

$$\boldsymbol{\mu}_t = \begin{pmatrix} 0.28584241 \\ 1.42626702 \end{pmatrix} \tag{1.8}$$

3. The probability density is highest at the mean (as illustrated in Figure 1.1). The density decreases quickly as both x and y change, but less so when x XOR y change. In $\boldsymbol{\Sigma}_p$, the values for the x XOR y are lower, so this is consistent with our density plot.

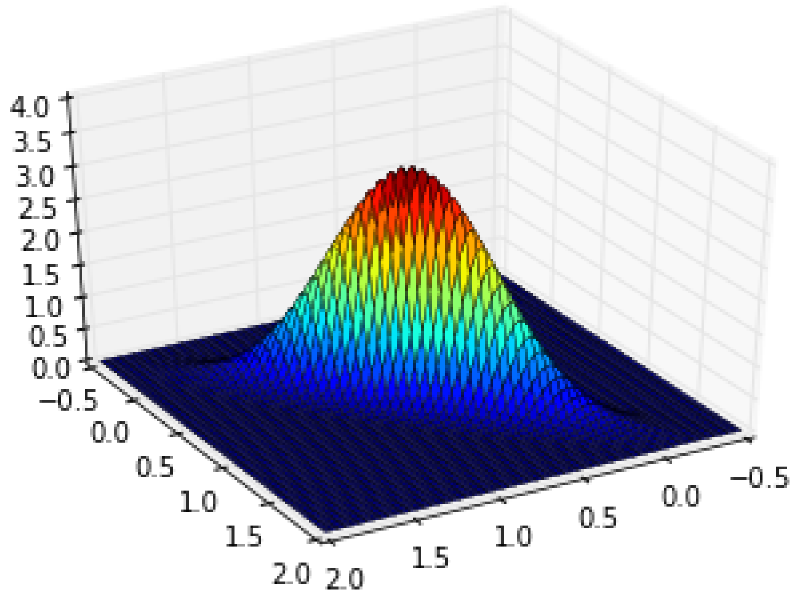

Figure 1.1: The probability density of the distribution.

1.2 Generating the data

1. We used the following function to generate our data:

```
np.random.multivariate_normal(mu_t, sigma_t, 1000)
```

2.

$$\boldsymbol{\mu}_{ML} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \tag{Bishop 2.121}$$

$$= [\frac{1}{1000}\sum_{n=1}^{1000} \boldsymbol{x}_n, \frac{1}{1000}\sum_{n=1}^{1000} \boldsymbol{y}_n] \tag{1.9}$$

$$= \begin{pmatrix} 0.25383138 \\ 1.38260838 \end{pmatrix} \tag{1.10}$$

$$\boldsymbol{\Sigma}_{ML} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})^T \tag{Bishop 1.22}$$

$$= \begin{pmatrix} \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})^T & \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})^T \\ \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})^T & \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})^T \end{pmatrix} \tag{1.11}$$

$$= \begin{pmatrix} 1.90513804 & 0.72479489 \\ 0.72479489 & 3.81690496 \end{pmatrix} \tag{1.12}$$

This is calculated using the code in Listing 1:

Listing 1: Python code to calculate $\boldsymbol{\mu}_{ML}$ and $\boldsymbol{\Sigma}_{ML}$.

```
mu_ml = sum(data)/len(data)

sse = [0,0]
for point in data:
            point = np.matrix(point)
            sse += (point-mu_ml).T*(point-mu_ml)
sigma_ml =  sse/len(data)
```

The differences with the 'true' values are:

$$\boldsymbol{\mu}_t - \boldsymbol{\mu}_{ML} = \begin{pmatrix} 0.28584241 \\ 1.42626702 \end{pmatrix} - \begin{pmatrix} 0.25383138 \\ 1.38260838 \end{pmatrix} = \begin{pmatrix} 0.03201103 \\ 0.04365864 \end{pmatrix} \tag{1.13}$$

$$\boldsymbol{\Sigma}_t - \boldsymbol{\Sigma}_{ML} = \begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 4.0 \end{pmatrix} - \begin{pmatrix} 1.90513804 & 0.72479489 \\ 0.72479489 & 3.81690496 \end{pmatrix} = \begin{pmatrix} 0.09486196 & 0.07520511 \\ 0.07520511 & 0.18309504 \end{pmatrix} \tag{1.14}$$

For the unbiased covariance:

$$\tilde{\boldsymbol{\Sigma}} = \frac{1}{N-1} \sum_{n=1}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})^T \tag{Bishop 2.125}$$

We added this line of code to the function of Listing 1:

```
sigma_ml_unbiased = sse * (1/(len(data)-1))
```

The difference between the unbiased covariance and the 'true' covariance is:

$$\boldsymbol{\Sigma}_t - \tilde{\boldsymbol{\Sigma}} = \begin{pmatrix} 2.0 & 0.8 \\ 0.8 & 4.0 \end{pmatrix} - \begin{pmatrix} 1.90704509 & 0.72552041 \\ 0.72552041 & 3.82072569 \end{pmatrix} = \begin{pmatrix} 0.09295491 & 0.07447959 \\ 0.07447959 & 0.17927431 \end{pmatrix} \tag{1.15}$$

The difference is slightly smaller than the difference between $\boldsymbol{\Sigma}_t$ and $\boldsymbol{\Sigma}_{ML}$.

1.3 Sequential learning algorithms

1. See Listing 2 for our procedure to process all data points one-by-one to calculate an estimate of $\boldsymbol{\mu}_{ML}$.

Listing 2: Python code for function *sequential_learning_ml(data)*.

```
def sequential_learning_ml(data):
    N = 0
    mu_ml = 0
    mus = []

    for point in data:
        N += 1
        mu_ml = mu_ml + (1/N)*(point-mu_ml)
        mus.append(mu_ml)

    print "Sequential mu_ml:", mu_ml
    return mus
```

This resulted in:

$$\boldsymbol{\mu}_{ML} = \begin{pmatrix} 0.25383138 \\ 1.38260838 \end{pmatrix} \tag{1.16}$$

2.

$$p(\boldsymbol{x}|D_{n-1}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \tag{Bishop 2.113}$$

where: $\boldsymbol{x} = \boldsymbol{\mu}, \boldsymbol{\mu} = \boldsymbol{\mu}_{(n-1)}, \boldsymbol{\Lambda}^{-1} = \boldsymbol{\Sigma}_{(n-1)}$

$$p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{L}^{-1}) \tag{Bishop 2.114}$$

where $\boldsymbol{y} = \boldsymbol{x}_n, \boldsymbol{A} = \boldsymbol{I}, \boldsymbol{x} = \boldsymbol{\mu}, \boldsymbol{b} = 0, \boldsymbol{L}^{-1} = \boldsymbol{\Sigma}^t$

$$p(\boldsymbol{x}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\Sigma}\{\boldsymbol{A}^T\boldsymbol{L}(\boldsymbol{y} - \boldsymbol{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\}, \boldsymbol{\Sigma}) \tag{Bishop 2.116}$$

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \boldsymbol{A}^T\boldsymbol{L}\boldsymbol{A})^{-1} \tag{Bishop 2.117}$$

Matching the variables we get the following equations:

$$\begin{aligned}
p(\boldsymbol{\mu}|\boldsymbol{x}_n) &= \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}(\boldsymbol{x}_n - 0) + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) && (1.17)\\
&= \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) && (1.18)\\
&= \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) && (1.19)\\
\boldsymbol{S} &= (\boldsymbol{\Sigma}_{(n-1)}^{-1} + \boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}\boldsymbol{I})^{-1} && (1.20)\\
&= (\boldsymbol{\Sigma}_{(n-1)}^{-1} + \boldsymbol{\Sigma}_t^{-1})^{-1} && (1.21)
\end{aligned}$$

$\boldsymbol{\mu}_n$ is the mean of the distribution $p(\boldsymbol{\mu}|\boldsymbol{x}_n)$, so the functions we use for our sequential learning algorithm are:

$$\begin{aligned}
\boldsymbol{\Sigma}_n &= \boldsymbol{S} && (1.22)\\
\boldsymbol{\mu}_n &= \boldsymbol{\Sigma}_n\{\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{n-1}^{-1}\}, \boldsymbol{\Sigma}_n) && (1.23)
\end{aligned}$$

3. See Listing 3 for our procedure to make a MAP estimation of $\boldsymbol{\mu}$ by processing the data points one-by-one.

Listing 3: Python code for function *sequential_learning_map(data, mu_p, sigma_p, sigma_t)*.

```
def sequential_learning_map(data, mu_p, sigma_p, sigma_t):
    sigma = sigma_p
    mu = mu_p
    mus = []

    for point in data:
        point = np.matrix(point).T
        S = np.linalg.inv( np.linalg.inv(sigma) + np.linalg.inv(sigma_t))
        mu = np.dot(S, np.dot( np.linalg.inv(sigma_t), point) + np.dot( np.linalg.inv(sigma), mu))
        sigma = S
        mus.append(np.array(mu))

    print "Sequential mu_map:", mu
    return mus
```

This resulted in:

$$\boldsymbol{\mu}_{MAP} = \begin{pmatrix} 0.25941079 \\ 1.37796331 \end{pmatrix} \tag{1.24}$$

4. See Figure 1.2 for our results: We observed that the MAP estimate performs better for less data points. This seems logical enough, since the MAP estimate is a regularized version of the ML estimate. When more data points are observed, the difference is almost non-existent.

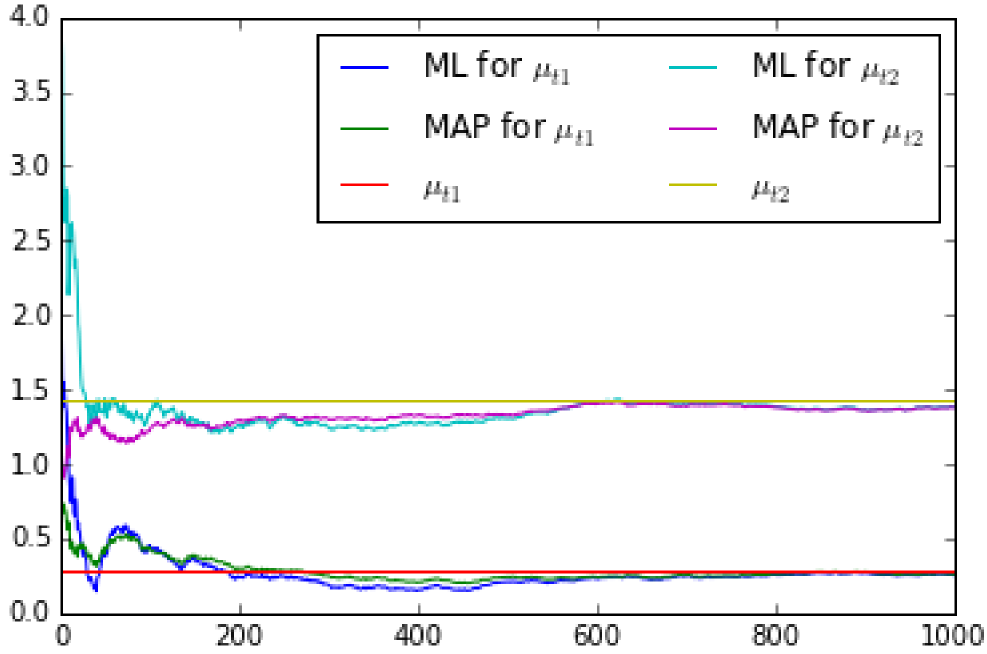

Figure 1.2: The ML and MAP estimates for $\boldsymbol{\mu}$ are given, plotted against the amount of data points observed. The 'true' value $\boldsymbol{\mu}_t$ is also indicated. This is done for both $\boldsymbol{\mu}_{t1}$ and $\boldsymbol{\mu}_{t2}$.

2 The faulty lighthouse

2.1 Constructing the model

1.

2. First, using the given $\beta \tan(\theta_k) = x_k - \alpha$, we will calculate the derivation of θ.

$$\beta \tan(\theta_k) = x_k - \alpha \qquad \text{(Assignment eq. 7)}$$

$$\tan(\theta_k) = \frac{x_k - \alpha}{\beta} \qquad (2.1)$$

$$\theta_k = \tan^{-1} \frac{x_k - \alpha}{\beta} \qquad (2.2)$$

$$\left|\frac{d\theta}{dx}\right| = \frac{1}{1 + (\frac{x_k-\alpha}{\beta})^2} \cdot \left|\frac{d\frac{x_k-\alpha}{\beta}}{dx}\right| \qquad (2.3)$$

$$= \frac{1}{1 + \frac{x_k-\alpha}{\beta}^2} \cdot \frac{\beta}{\beta^2} \qquad (2.4)$$

$$= \frac{\beta}{\beta^2 + \beta^2(\frac{x_k-\alpha}{\beta})^2} \qquad (2.5)$$

$$= \frac{\beta}{\beta^2 + \beta^2\frac{(x_k-\alpha)^2}{\beta^2}} \qquad (2.6)$$

$$= \frac{\beta}{\beta^2 + (x_k - \alpha)^2} \qquad (2.7)$$

Since the following equation holds:

$$p_x(x) = p_\theta(\theta_k)\left|\frac{d\theta}{dx}\right| \qquad \text{(Bishop 1.27)}$$

We need to multiply the derivation of θ with $p(\theta_k|\alpha,\beta)$ (Assignment eq. 6):

$$p(x_k|\alpha,\beta) = \frac{\beta}{\beta^2 + (x_k-\alpha)^2} \cdot \frac{1}{\pi} \qquad (2.8)$$

$$= \frac{\beta}{\pi[\beta^2 + (x_k-\alpha)^2]} \qquad \text{(Assignment eq. 8)}$$

We have plotted the distribution for $\beta = 1$ and for α we chose the value 0.5, as illustrated in Figure 2.1:

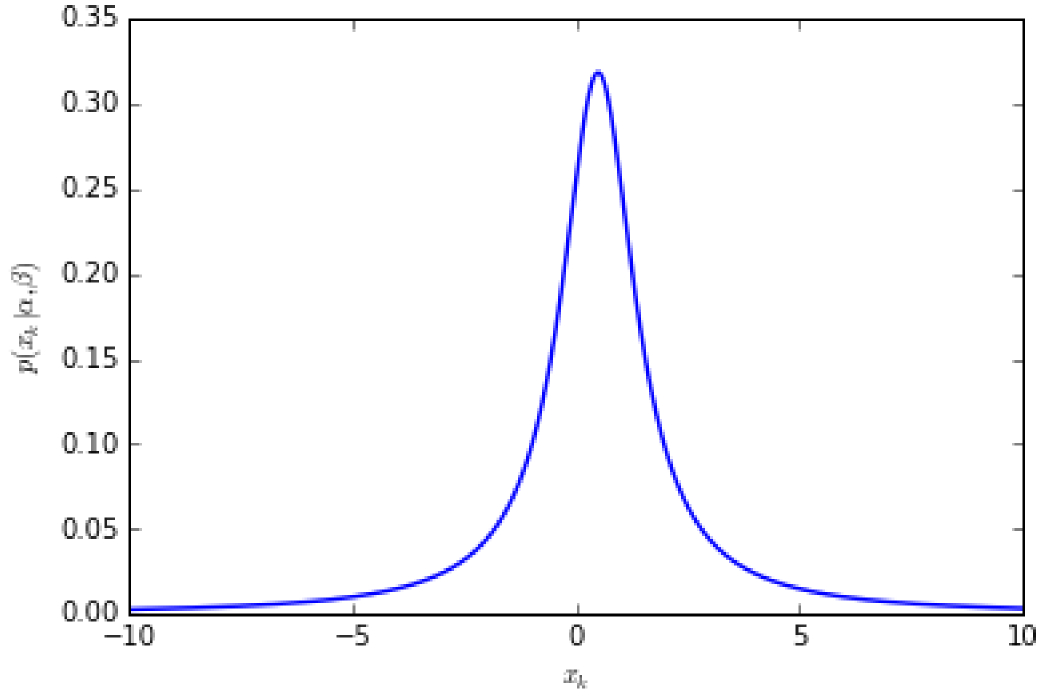

Figure 2.1: The probability distribution $p(x_k|\alpha,\beta)$ plotted against x_k, with $\alpha = 0.5$ and $\beta = 1$.

3.

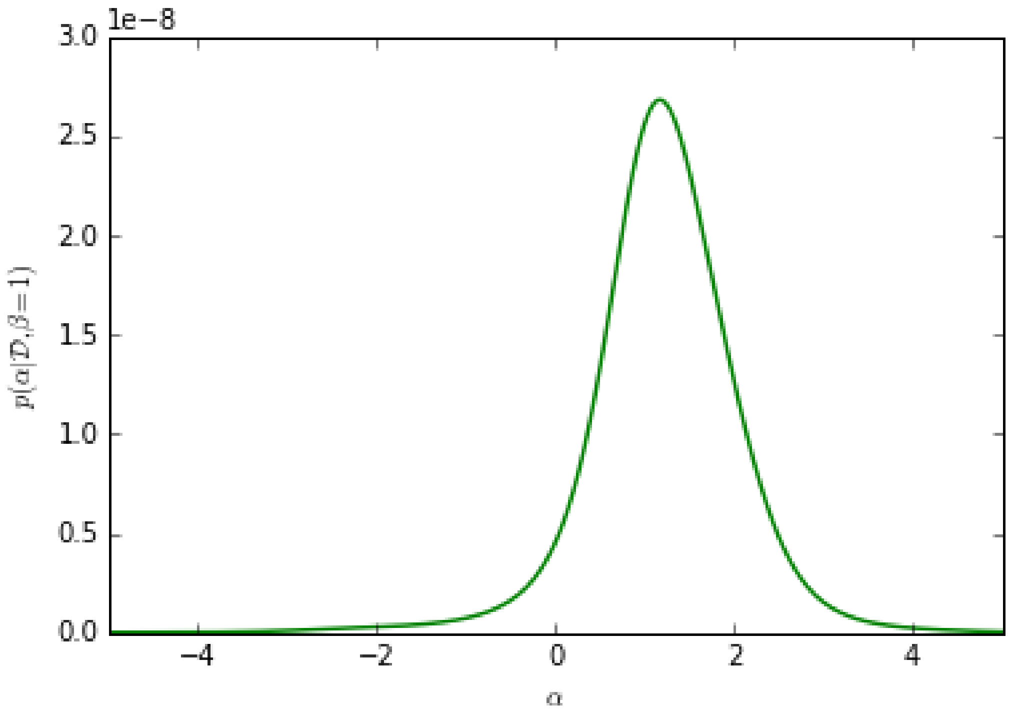

Figure 2.2: The probability density $p(\alpha|\mathscr{D},\beta = 1)$ plotted against α.

4. The most likely estimate for $\hat{\alpha}$ = 1.17136. The mean of α = -0.18333. The difference is probably caused by the amount of data points we have, these are very limited. Outliers in the data will have a great effect on the mean.

2.2 Generate the lighthouse data

1.

2.

2.3 Find the lighthouse

1.

2.

3.