Statistical Machine Learning
Assignment 2

Inez Wijnands (s4149696) & Guido Zuidhof (s4160703)

Radboud University Nijmegen

04/11/2015

The entire code listing is included in the zip-file. The listings shown here are merely code snippets.

1 Sequential learning

1.1 Obtaining the prior

1.

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

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

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

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

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

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

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

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 $\mathbf{\Sigma}_p$, 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.

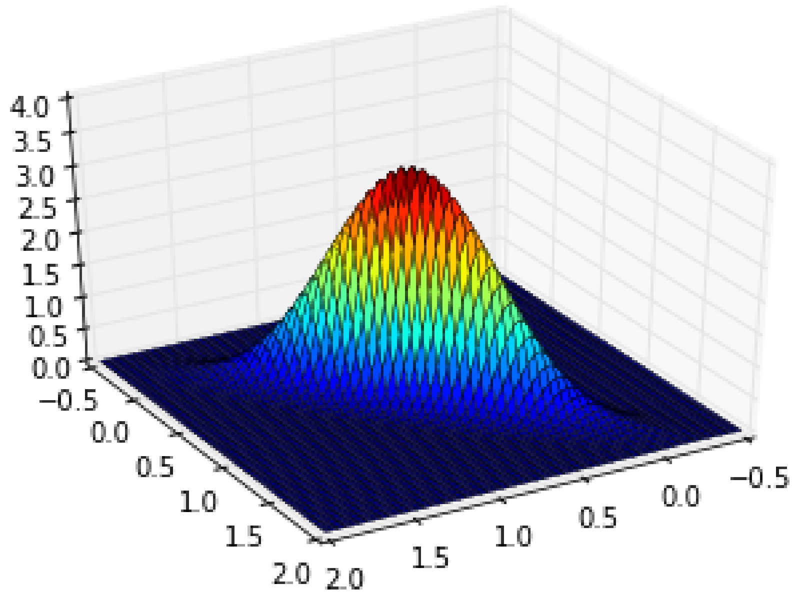


Figure 1.1: The probability density of the distribution.

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.

$$\begin{aligned}
\boldsymbol{\mu}_{ML} &= \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n && \text{(Bishop 2.121)}\\
&= [\frac{1}{1000}\sum_{n=1}^{1000} \boldsymbol{x}_n, \frac{1}{1000}\sum_{n=1}^{1000} \boldsymbol{y}_n] && (1.9)\\
&= \begin{pmatrix} 0.25383138 \\ 1.38260838 \end{pmatrix} && (1.10)
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{\Sigma}_{ML} &= \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{ML})^T && \text{(Bishop 1.22)}\\
&= \begin{pmatrix} \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})^T & \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})^T \\ \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})(\boldsymbol{x}_n - \boldsymbol{\mu}_{(1)ML})^T & \frac{1}{1000}\sum_{n=1}^{1000}(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})(\boldsymbol{y}_n - \boldsymbol{\mu}_{(2)ML})^T \end{pmatrix} && (1.11)\\
&= \begin{pmatrix} 1.90513804 & 0.72479489 \\ 0.72479489 & 3.81690496 \end{pmatrix} && (1.12)
\end{aligned}$$

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:

$$p(\boldsymbol{\mu}|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}(\boldsymbol{x}_n - 0) + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) \tag{1.17}$$

$$= \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) \tag{1.18}$$

$$= \mathcal{N}(\boldsymbol{\mu}|\boldsymbol{S}\{\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{(n-1)}^{-1}\}, \boldsymbol{S}) \tag{1.19}$$

$$\boldsymbol{S} = (\boldsymbol{\Sigma}_{(n-1)}^{-1} + \boldsymbol{I}^T\boldsymbol{\Sigma}_t^{-1}\boldsymbol{I})^{-1} \tag{1.20}$$

$$= (\boldsymbol{\Sigma}_{(n-1)}^{-1} + \boldsymbol{\Sigma}_t^{-1})^{-1} \tag{1.21}$$

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

$$\boldsymbol{\Sigma}_n = \boldsymbol{S} \tag{1.22}$$

$$\boldsymbol{\mu}_n = \boldsymbol{\Sigma}_n\{\boldsymbol{\Sigma}_t^{-1}\boldsymbol{x}_n + \boldsymbol{\Sigma}_{n-1}^{-1}\}, \boldsymbol{\Sigma}_n) \tag{1.23}$$

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:

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

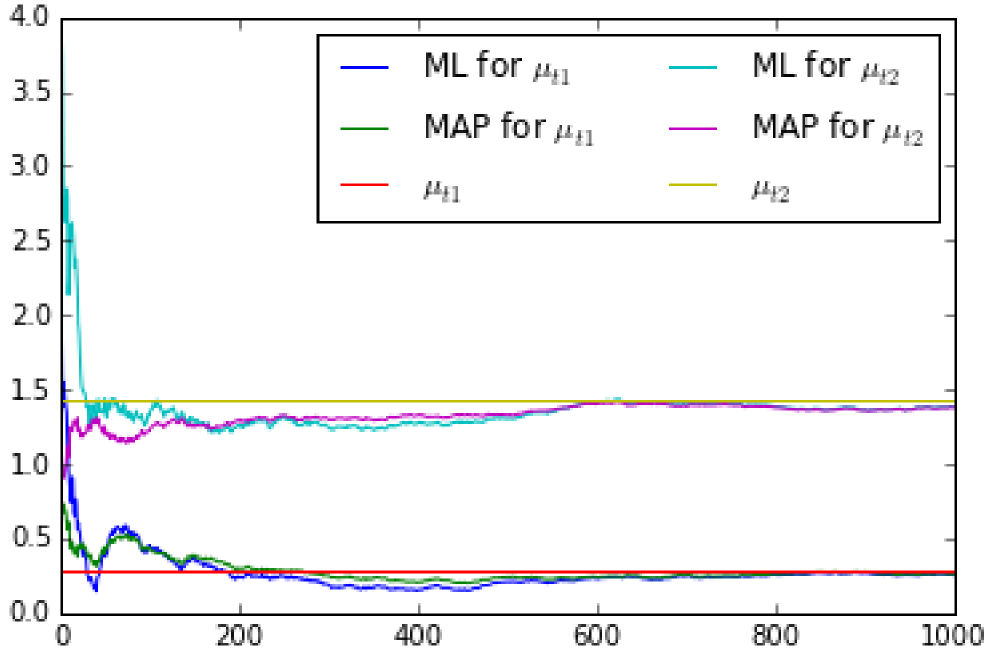


Figure 1.2: The ML and MAP estimates for $\boldsymbol{\mu}$ are given, plotted against the amount of data points observed. The 'true' value $\boldsymbol{\mu}_t$ is also indicated. This is done for both $\boldsymbol{\mu}_{t1}$ and $\boldsymbol{\mu}_{t2}$.

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.

2 The faulty lighthouse

2.1 Constructing the model

1. A full circle is 360 degrees and corresponds to 2π rad. Since the light house can only be observed from the coast, which is a straight line, the light can only reach half a circle. This means that we need to see if the distribution for the values $-\frac{1}{2}\pi$ rad to $\frac{1}{2}\pi$ rad adds up to one. If it is a reasonable distribution, the following should hold:

$$\int_{-\frac{1}{2}\pi}^{\frac{1}{2}\pi} \frac{1}{\pi} dx = 1 \tag{2.1}$$

$$= \frac{x}{\pi} + c\Big|_{-\frac{1}{2}\pi}^{\frac{1}{2}\pi} \tag{2.2}$$

$$= \frac{\frac{1}{2}\pi}{\pi} - \frac{-\frac{1}{2}\pi}{\pi} = 1 \tag{2.3}$$

   We can conclude that since the light of the lighthouse can reach half a circle, $\frac{1}{\pi}$ is a fine distribution (as it is a uniform distribution over the possible angles).

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

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

$$\tan(\theta_k) = \frac{x_k - \alpha}{\beta} \tag{2.4}$$

$$\theta_k = \tan^{-1} \frac{x_k - \alpha}{\beta} \tag{2.5}$$

$$\left|\frac{d\theta}{dx}\right| = \frac{1}{1 + (\frac{x_k-\alpha}{\beta})^2} \cdot \left|\frac{d\frac{x_k-\alpha}{\beta}}{dx}\right| \tag{2.6}$$

$$= \frac{1}{1 + \frac{x_k-\alpha}{\beta}^2} \cdot \frac{\beta}{\beta^2} \tag{2.7}$$

$$= \frac{\beta}{\beta^2 + \beta^2(\frac{x_k-\alpha}{\beta})^2} \tag{2.8}$$

$$= \frac{\beta}{\beta^2 + \beta^2 \frac{(x_k-\alpha)^2}{\beta^2}} \tag{2.9}$$

$$= \frac{\beta}{\beta^2 + (x_k - \alpha)^2} \tag{2.10}$$

   Since the following equation holds:

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

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

$$p(x_k|\alpha,\beta) = \frac{\beta}{\beta^2 + (x_k - \alpha)^2} \cdot \frac{1}{\pi} \tag{2.11}$$

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

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

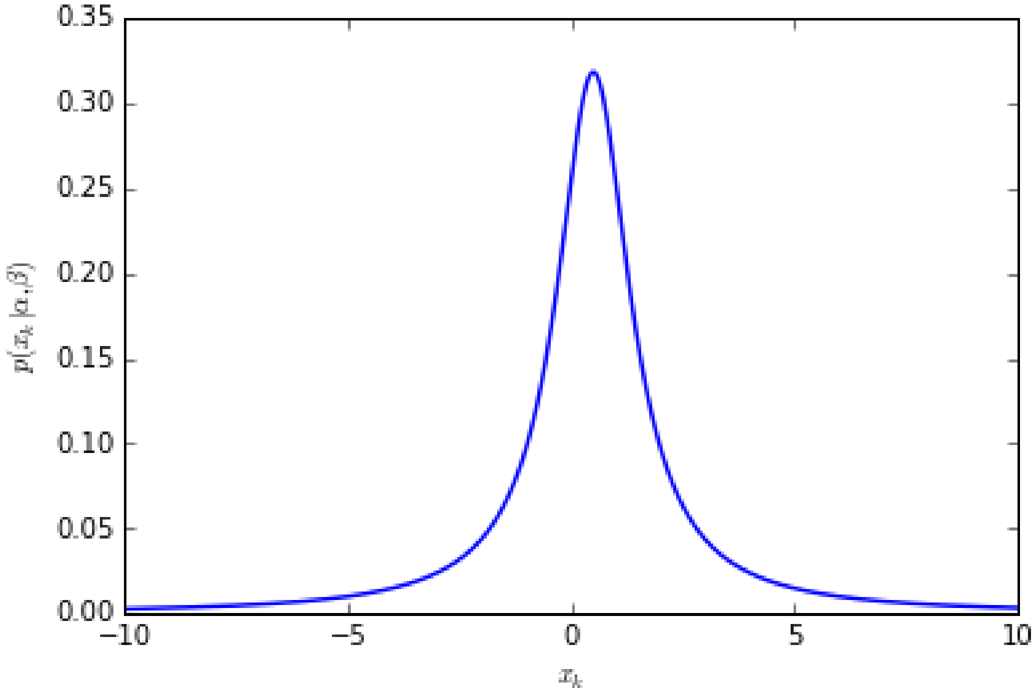


Figure 2.1: The probability distribution $p(x_k|\alpha,\beta)$ plotted against x_k, with $\alpha = 0.5$ and $\beta = 1$.

3.

$$p(\alpha|\mathscr{D},\beta) = p(\mathscr{D}|\alpha,\beta)p(\alpha|\beta) \tag{2.12}$$

$$p(x_k|\alpha,\beta) = \frac{\beta}{\pi[\beta^2 + (x_k-\alpha)^2]} \tag{Assignment eq. 8}$$

$$= \frac{\beta}{\pi} \cdot \frac{1}{\beta^2 + (x_k-\alpha)^2} \tag{2.13}$$

$$\ln(p(x_k|\alpha,\beta)) = \ln(\frac{\beta}{\pi} \cdot \frac{1}{\beta^2 + (x_k-\alpha)^2}) \tag{2.14}$$

$$= \ln\frac{\beta}{\pi} + \ln(\frac{1}{\beta^2 + (x_k-\alpha)^2}) \tag{2.15}$$

$$= \ln\frac{\beta}{\pi} - \ln[\beta^2 + (x_k-\alpha)^2] \tag{2.16}$$

$$p(\mathscr{D}|\alpha,\beta) = \prod_{x_k\in\mathscr{D}} p(x_k|\alpha,\beta) \tag{2.17}$$

$$\ln(p(\mathscr{D}|\alpha,\beta)) = \left|\mathscr{D}\right| \cdot \ln(\frac{\beta}{\pi}) - \sum_{x_k\in\mathscr{D}} \ln([x_k-\alpha)^2 + \beta^2]) \tag{2.18}$$

Since $\left|\mathscr{D}\right| \cdot \ln(\frac{\beta}{\pi})$ is a constant, the log of the posterior density can be written like this:

$$\ln(p(\alpha|\mathscr{D},\beta)) = \left|\mathscr{D}\right| \cdot \ln(\frac{\beta}{\pi}) - \sum_{x_k\in\mathscr{D}} \ln([x_k-\alpha)^2 + \beta^2]) \tag{Assignment eq. 9}$$

Maximizing the posterior density gives the following expression:

$$\hat{\alpha} = \arg\max_{\alpha}[p(\mathscr{D}|\alpha,\beta)] \tag{2.19}$$

$$= \arg\max_{\alpha}[\prod_{x_k\in\mathscr{D}} p(x_k|\alpha,\beta)] \tag{2.20}$$

$$= \arg\max_{\alpha}[\prod_{x_k\in\mathscr{D}} \frac{\beta}{\pi[\beta^2 + (x_k-\alpha)^2]}] \tag{2.21}$$

4. The most likely estimate for $\hat{\alpha}$ = 1.17136. The mean of α = -0.18333. The difference is probably caused by

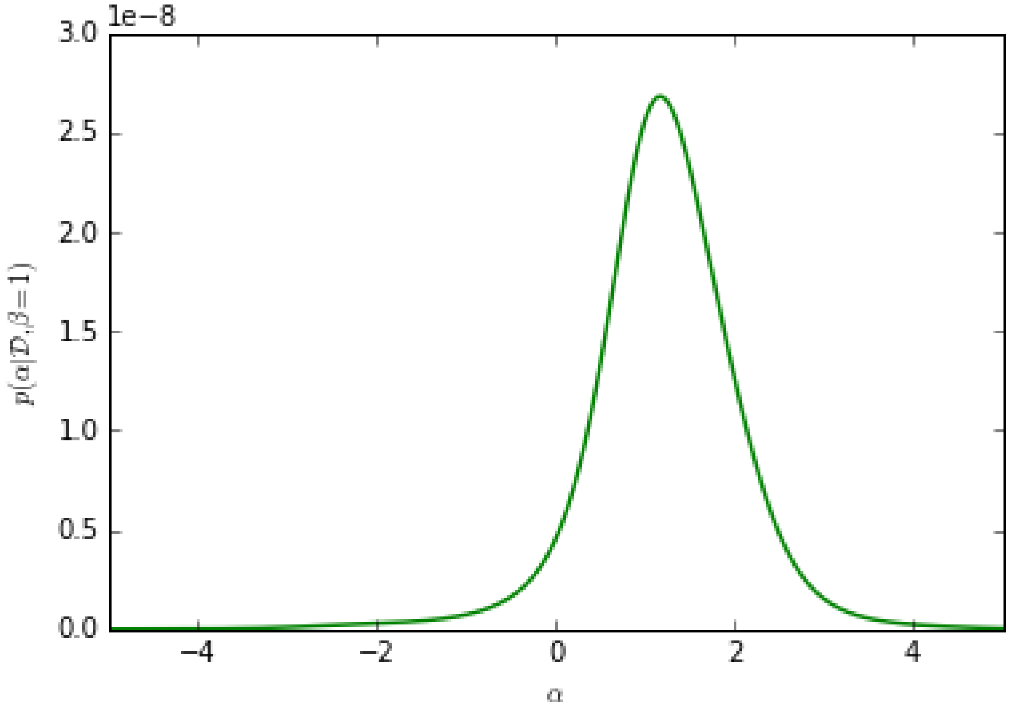


Figure 2.2: The probability density $p(\alpha|\mathscr{D},\beta=1)$ plotted against α.

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. The code for sampling a position for our light house is listed in Listing 4:

Listing 4: Python code for function *generate_random_position()*.

```
def generate_random_position():
    a = np.random.uniform(0.00,10.0)
    b = np.random.uniform(1.0,2.0)
    return a,b
```

This resulted in the following coordinates:

$$\alpha_t = 3.74540 \tag{2.22}$$

$$\beta_t = 1.95071 \tag{2.23}$$

2. We generate the data set $\mathscr{D}$ with the code from Listing 5:

Listing 5: Python code for function *generate_data(a,b,n)*.

```
def generate_data(a,b,n):
    data = []
    for _ in xrange(n):
        angle = np.random.uniform(0.5*-math.pi, 0.5*math.pi)
        value = b*math.tan(angle)+a
        data.append(value)

    return data
```

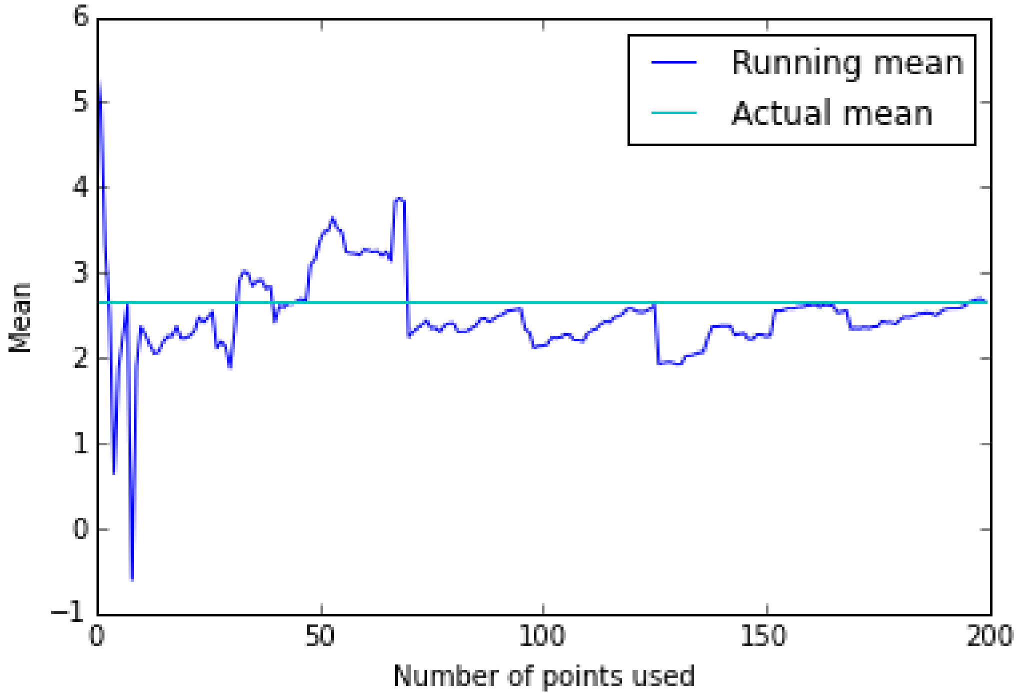


Figure 2.3: The mean of the data set as a function of the number of data points added.

3. Observing Figure 2.3, we think we need between 75-100 data points to get a reasonable estimate of the mean, but this is with our data. If a dataset contains more outliers, it may take more data points, and if it contains fewer outliers, it may take less points.

2.3 Find the lighthouse

1. See section 2.1.3 for how we obtained the log likelihood function:

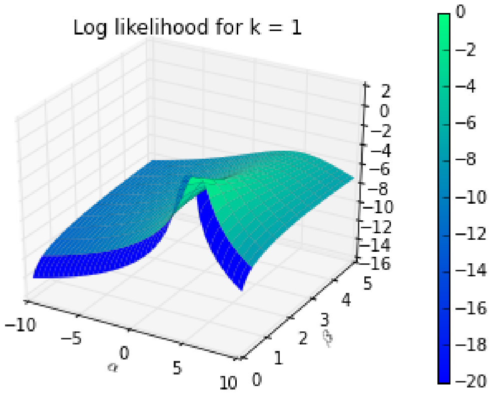
$$\ln(p(\mathscr{D}|\alpha,\beta)) = \left|\mathscr{D}\right| \cdot \ln(\frac{\beta}{\pi}) - \sum_{x_k \in \mathscr{D}} \ln([x_k - \alpha)^2 + \beta^2]) \tag{2.18}$$

2. See Figure 2.4 for the plots of the log likelihood calculations with different amounts of data (k) used for the calculation. For few data points (smaller k), β is always around 0. However, the value for α is estimated near the true value already with limited data points. This might be because α and β contribute differently to the log likelihood, and thus observations would change the likelihood in a different manner as well.
We use the log likelihood instead of the likelihood, because the square term in the likelihood function decreases the likelihood very fast. The log likelihood is a regularized version of the likelihood and thus the value would change in a less extreme way, making smaller changes better visible because not all values are scaled to one extreme peak.

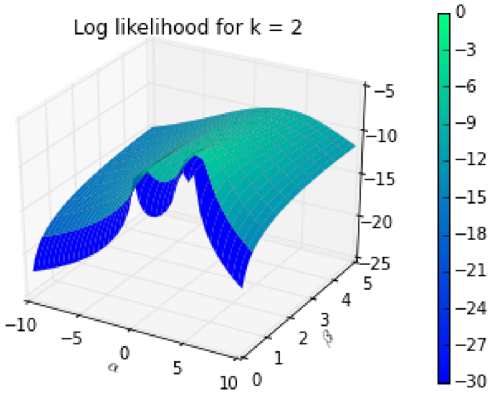
3. We used the scipy-equivalent `scipy.optimize.fmin(function,*args)` in Python of the MATLAB-function `fminsearch`). See Figure 2.5 for our results: When we compare the found values for α and β with the 'true' values as in section 2.2.1, we get the following differences:

$$\alpha_t - \alpha_{fmin} = 3.74540 - 3.56695 = 0.17845 \tag{2.24}$$

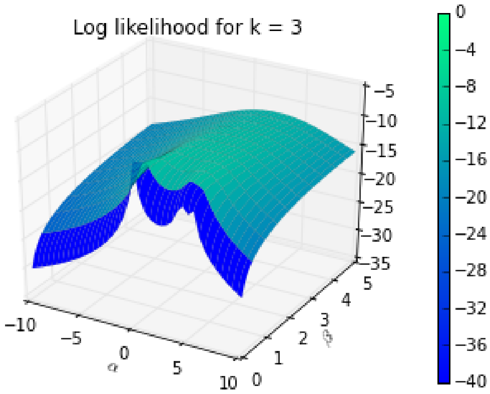
$$\beta_t - \beta_{fmin} = 1.95071 - 2.13500 = -0.18429 \tag{2.25}$$



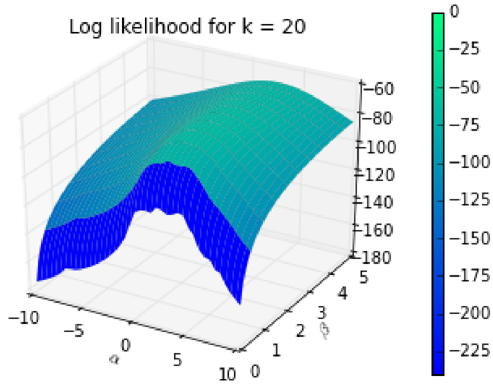
(a) k = 1.



(b) k = 2.



(c) k = 3.



(d) k = 20.

Figure 2.4: Log likelihood for k.

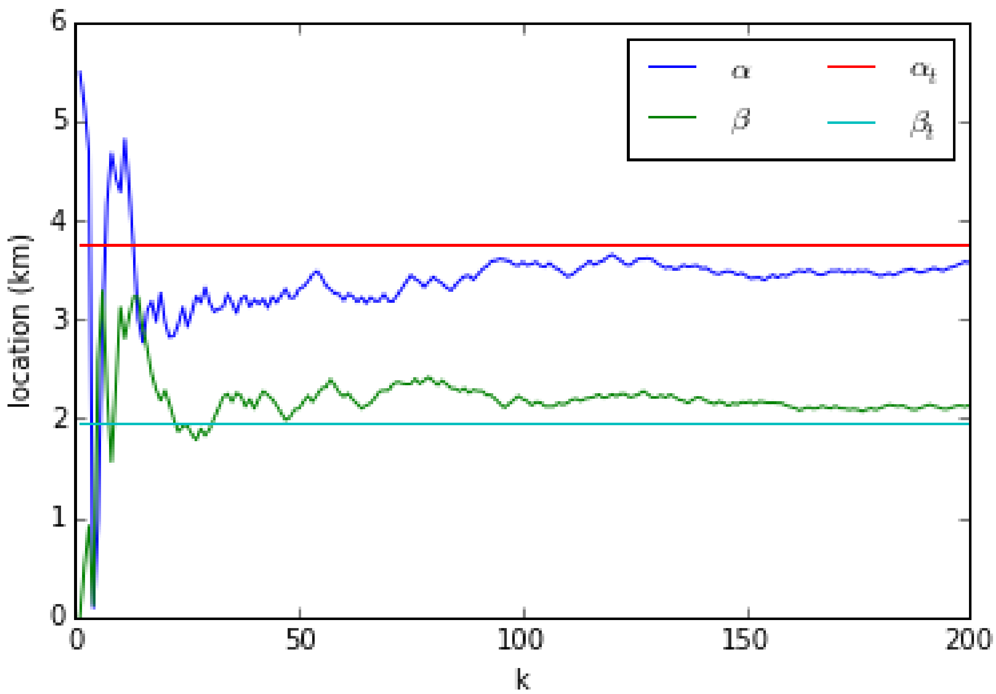


Figure 2.5: Minimum α and β that are found by using the `fmin` function to minimize the log likelihood function.