Statistical Machine Learning 2018
Assignment 3
Deadline: 25th of November 2018

Christoph Schmidl
s4226887
c.schmidl@student.ru.nl

Mark Beijer
s4354834
mbeijer@science.ru.nl

November 25, 2018

Exercise 1 - The faulty lighthouse (weight 5)

A lighthouse is somewhere off a piece of straight coastline at a position α along the shore and a distance β out to sea. Due to a technical fault, as it rotates the light source only occasionally and briefly flickers on and off. As a result it emits short, highly focused beams of light at random intervals. These pulses are intercepted on the coast by photo-detectors that record only the fact that a flash has occurred, but not the angle from which it came. So far, N flashes have been recorded at positions $\mathcal{D} = x1, ..., xN$. Where is the lighthouse?

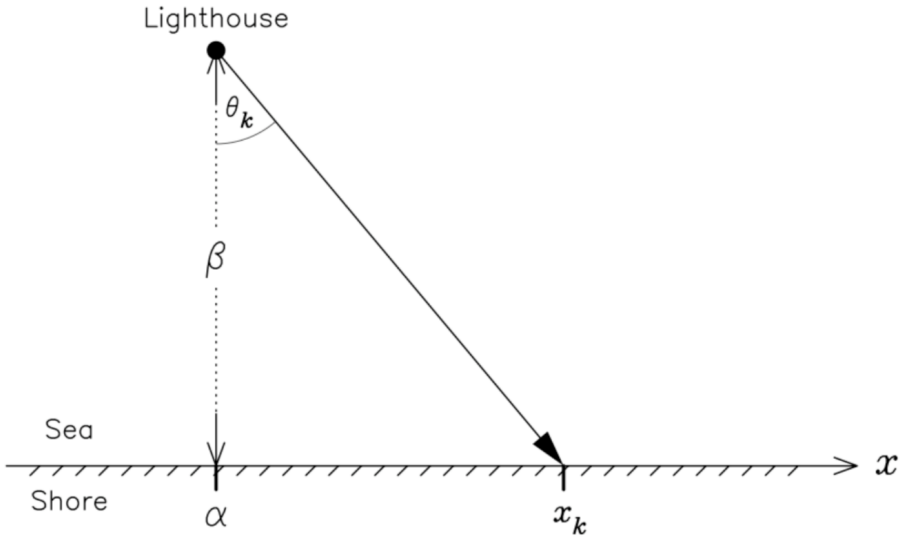

Figure 1: Geometry of the lighthouse problem.

Part 1 - Constructing the model

1.1.1

Let θ_k be the (unknown) angle for the k-th recorded flash, see fig.1. Argue why

$$p(\theta_k|\alpha, \beta) = \frac{1}{\pi} \tag{1}$$

would be a reasonable distribution over θ_k between $\pm\frac{\pi}{2}$ (zero otherwise).

Answer:

The description states that photo-detectors are only located on the coast side but not on the shore side. Normally a lighthouse emits light in a rotating fashion of 360° but based on the before mentioned fact we can assume that photo-detectors can only detect light beams in an angle of 180° around the light house on the coast side, whereas the remaining 180° on the shore side are not detected by any photo-detectors. If we transform the degrees into radians then we have $360° = 2\pi$ radians and $180° = \pi$ radians. Based on the fact that we are only interested in the specific angle θ_k and not the whole region of 180° or π radians, $\frac{1}{\pi}$ seems like a reasonable distribution in this case.

We can also show that $p(\theta_k|\alpha, \beta) = \frac{1}{\pi}$ is a reasonable distribution over θ_k between $\pm\frac{\pi}{2}$ by demsonstrating that it integrates to 1:

$$\begin{aligned}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{1}{\pi}\, dx &= \Big[\frac{x}{\pi}\Big]_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \\ &= \big[\frac{\frac{\pi}{2}}{\pi}\big] - \big[-\frac{\frac{\pi}{2}}{\pi}\big] \\ &= \frac{\frac{\pi}{2} + \frac{\pi}{2}}{\pi} \\ &= \frac{\pi}{\pi} \\ &= 1\end{aligned}$$

1.1.2

We only have the position x_k of the detector that recorded flash k, but we can relate this to the unknown θ_k via elementary geometry as

$$\beta \tan(\theta_k) = x_k - \alpha \tag{2}$$

Show that the expected distribution over x given α and β can be written as

$$p(x_k|\alpha, \beta) = \frac{\beta}{\pi[\beta^2 + (x_k - \alpha)^2]} \tag{3}$$

by using (2) to substitute variable x_k for θ_k in the distribution (1). Plot the distribution for $\beta = 1$ and a particular value of α.
Hint: use the Jacobian $|\frac{d\theta}{dx}|$ (Bishop, p.18) and the fact that $(tan^{-1}x)' = \frac{1}{1+x^2}$.

Answer:

Isolating variable θ_k using arctan:

$$\begin{aligned}\beta \tan(\theta_k) &= x_k - \alpha \\ \tan(\theta_k) &= \frac{x_k - \alpha}{\beta} \\ \theta_k &= \tan^{-1}(\frac{x_k - \alpha}{\beta})\end{aligned}$$

Computing the jacobian and using the little hint that $(tan^{-1}x)' = \frac{1}{1+x^2}$:

$$\begin{aligned}|\frac{d\theta}{dx}| &= \frac{1}{1 + (\frac{x_k - \alpha}{\beta})^2} \times \frac{\beta}{\beta^2} \\ &= \frac{\beta}{\beta^2 + \beta^2(\frac{x_k - \alpha}{\beta})^2} \\ &= \frac{\beta}{\beta^2 + (x_k - \alpha)^2}\end{aligned}$$

Showing that the expected distribution over x given α and β can be written as

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

by:

$$\begin{aligned} p(x_k|\alpha,\beta) &= p(\theta_k) \times \left|\frac{d\theta}{dx}\right| \\ &= \frac{1}{\pi} \times \frac{\beta}{\beta^2 + (x_k - \alpha)^2} \\ &= \frac{\beta}{\pi[\beta^2 + (x_k - \alpha)^2]} \end{aligned}$$

Plotting the distribution as shown in figure 2 using the following code:

```
# Exercise 1.1.2

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.optimize import fmin
%matplotlib inline

def get_probability_distribution(x, alpha, beta):
    return beta/(np.pi*(beta**2+(x-alpha)**2))

x = np.linspace(-10, 10, num=1000)
probs = get_probability_distribution(x, 3, 1)
plt.xlabel(r'$x_k$')
plt.ylabel(r'$p(x_k|\alpha, \beta)$')
plt.plot(x, probs)
plt.savefig('1_1_2.png')
plt.show()
```

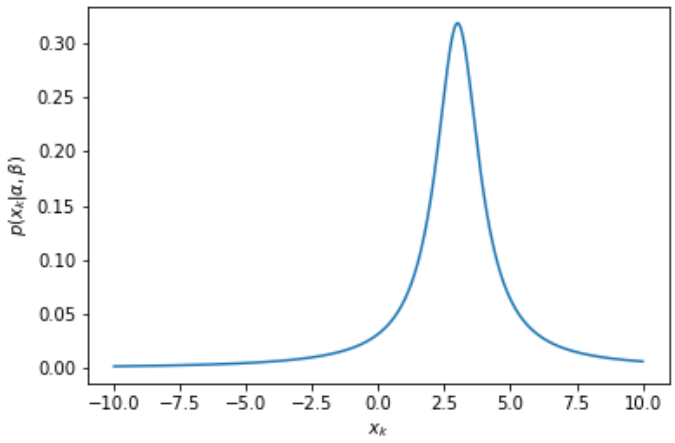

Figure 2: Plotting $p(x_k|\alpha,\beta)$ with $\alpha = 3$ and $\beta = 1$

1.1.3

Inferring the position of the lighthouse corresponds to estimating α and β from the data $\mathcal{D}$. This is still quite difficult, but if we assume that β is known, then from Bayes' theorem we know that $p(\alpha|\mathcal{D},\beta) \propto p(\mathcal{D}|\alpha,\beta)p(\alpha|\beta)$. We have no a priori knowledge about the position α along the coast other than that it should not depend on the distance out at sea.

Show that with these assumptions the log of the posterior density can be written as

$$L = \ln(p(\alpha|\mathcal{D},\beta)) = \texttt{constant} - \sum_{k=1}^{N} \ln[\beta^2 + (x_k - \alpha)^2] \tag{4}$$

and give an expression for the value $\hat{\alpha}$ that maximizes this posterior density.

Answer:

$$
\begin{aligned}
L &= \ln(p(\alpha|\mathcal{D}, \beta)) \\
&= \ln\left(\prod_{k=1}^{N} \frac{\beta}{\pi[\beta^2 + (x_k - \alpha)^2]}\right) \\
&= \sum_{k=1}^{N} \ln\left(\frac{\beta}{\pi} \times \frac{1}{\beta^2 + (x_k - \alpha)^2}\right) \\
&= N \ln \frac{\beta}{\pi} - \sum_{k=1}^{N} \ln[\beta^2 + (x_k - \alpha)^2] \\
&= constant - \sum_{k=1}^{N} \ln[\beta^2 + (x_k - \alpha)^2]
\end{aligned}
$$

Expression for the value $\hat{\alpha}$ that maximizes this posterior density:

$$
\hat{\alpha} = \arg\max_{\alpha} \sum_{k=1}^{N} \ln[\beta^2 + (x_k - \alpha)^2]
$$

1.1.4

Suppose we have a data set (in km) of $\mathcal{D} = \{3.6, 7.7, -2.6, 4.9, -2.3, 0.2, -7.3, 4.4, 7.3, -5.7\}$. We also assume that the distance β from the shore is known to be 2 km. As it is difficult to find a simple expression for the value of $\hat{\alpha}$ that maximizes (4), we try an alternative approach instead.

Plot $p(\alpha|\mathcal{D}, \beta = 2)$ as a function of α over the interval $[-10, 10]$. What is your most likely estimate for $\hat{\alpha}$ based on this graph? Compare with the mean estimate of the dataset. Can you explain the difference?

Answer:

The mean of the data is ≈ 1.02 but the value which maximizes the likelihood is $\hat{\alpha} \approx 2.57$. We assume that 10 data points are not sufficiently large to compute a reliable result.

```python
# 10 points in km
data = np.array([3.6, 7.7, -2.6, 4.9, -2.3, 0.2, -7.3, 4.4, 7.3, -5.7])
# distance beta from the shore is known to be 2 km
beta = 2
alpha_list = np.linspace(-10, 10, num=1000)

def get_probability_1_1_4(x, alpha, beta):
    return np.product(beta/(np.pi*beta**2+(x-alpha)**2))


likelihoods = [get_probability_1_1_4(data, alpha, beta) for alpha in alpha_list]
plt.xlabel(r'$\alpha$')
plt.ylabel(r'$p(\alpha|D, \beta = 2)$')
plt.plot(alpha_list, likelihoods)
plt.savefig('1_1_4.png')
plt.show()

print("Mean of the data: {}".format(np.mean(data)))

max_likelihood_index = np.argmax(likelihoods)
max_likelihood_value = np.max(likelihoods)

print("Max likelihood value {} at position {}.".format(max_likelihood_value,
    max_likelihood_index))
print("Alpha value which maximizes likelihood: {}".format(alpha_list[max_likelihood_index]))
```

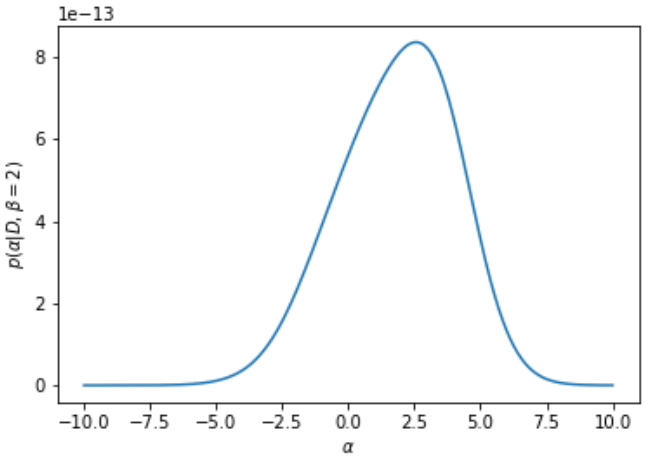

Figure 3: Plotting probability density over α

Part 2 - Generate the lighthouse data

We will try to solve the original problem by letting matlab/Python find the lighthouse for us. For that we first need a data set.

1.2.1

Sample a random position (α_t, β_t) from a uniform distribution over an interval of 10 km along the coast and between 2 and 4 km out to sea.

Answer:

```
# Exercise 1.2.1

alpha_t = np.random.uniform(0, 10)
beta_t = np.random.uniform(2, 4)

print(alpha_t)
print(beta_t)
```

We will work with:

- $\alpha_t = 5.022$
- $\beta_t = 2.436$

1.2.2

From this position generate a data set $\mathcal{D} = \{x_1, ..., x_N\}$ of 500 flashes in random directions that have been registered by a detector at point x_i along the coast. Assume that the flashes are i.i.d. according to (1).

Answer:

We already know from exercise 1.1.2 that we can calculate the isolated angle θ_k by restructuring the following equation:

$$\beta \tan(\theta_k) = x_k - \alpha$$

Getting the isolated position x_k is even easier to do:

$$\begin{aligned} \beta \tan(\theta_k) &= x_k - \alpha \\ x_k &= \beta \tan(\theta_k) + \alpha \end{aligned}$$

```python
# Exercise 1.2.2

def position(angle, alpha, beta):
    return beta * np.tan(angle) + alpha

N_points = 500
angles = np.random.uniform(-np.pi/2, np.pi/2, N_points)
positions = [position(angle, alpha_t, beta_t) for angle in angles]

print(positions)
```

1.2.3

Make a plot of the mean of the data set as a function of the number of points. Compare with the true position of the lighthouse α_t. How many points do you expect to need to obtain a reasonable estimate of α_t from the mean? Explain.

Answer:

The true mean of the data points is ≈ 5.579 which is close to the original position of the lighthouse $\alpha_t = 5.022$. I assume that we probably need between 250 and 500 more data points because we can already see from the plot that the graph converges around the point of 400 data points.

```python
mus = [np.mean(positions[:i + 1]) for i in range(N_points)]
mean = [np.mean(positions)] * (N_points)
X = np.arange(1, N_points + 1)
plt.plot(X, mus, label='Mean over time')
plt.plot(X, mean, label='True mean')
plt.xlabel('data points')
plt.ylabel(r'$\alpha$')
plt.legend()
plt.savefig('1_2_3.png', bbox_inches='tight', dpi=300)
plt.show()

print("Mean of the data points: {}".format(np.mean(positions)))
```

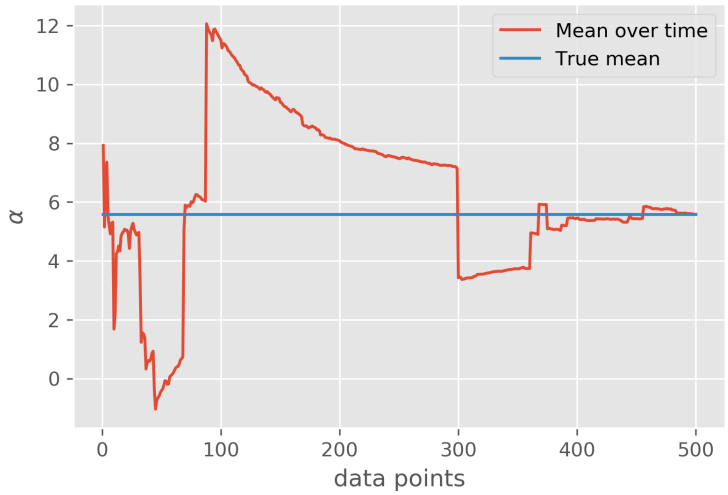

Figure 4: Plotting probability density over α

Part 3 - Find the lighthouse

From the analysis in the first part we know that trying to find a maximum likelihood estimate in the usual way is possible (compute gradient, set equal to zero and solve), but that this does not result in a 'nice' closed-form expression for the solution, even when one of the parameters is assumed to be known. As we want to find estimates of both α and β from the data, we will try a different approach instead.

1.3.1

Use (3) to get an expression for the loglikelihood of the data $\mathcal{D}$ as a function of α and β.

Answer:

$$
\begin{aligned}
p(\mathcal{D}|\alpha,\beta) &= \prod_{k=1} N p(x_k|\alpha,\beta) \\
\ln p(\mathcal{D}|\alpha,\beta) &= \ln \prod_{k=1}^{N} p(x_k|\alpha,\beta) \\
&= \sum_{k=1}^{N} \ln p(x_k|\alpha,\beta) \\
&= \sum_{k=1}^{N} \ln \frac{\beta}{\pi[\beta^2 + (x_k-\alpha)^2]} \\
&= N \ln \frac{\beta}{\pi} - \sum_{k=1}^{N} \ln[\beta^2 + (x_k-\alpha)^2]
\end{aligned}
$$

1.3.2

We can see how this likelihood (as a function of α and β) changes, as data points come in.

Process your data set $\mathcal{D}$ one by one and make a plot of the (log)likelihood after one, two, three, and 20 points have arrived, respectively. Explain what happens.

Hint: Create a function that calculates the (log)likelihood at a specific point (α, β) after the first k data points $\{x_1, ..., x_k\}$ have come in. Use this with the matlab `meshgrid` and `surf` functions to make plots over the interval $[-10 \leq \alpha \leq +10] \times [0 \leq \beta \leq 5]$. Decide if/when it makes more sense to use the likelihood directly or the log of the likelihood.

Answer:

The following code produced figure 5.

```
arrived_datapoints = [1, 2, 3, 20]
alpha_list, beta_list = np.meshgrid(np.linspace(-10, 10, num=500), np.linspace(0, 5, num
    =250))
plt.style.use('classic')

for k in arrived_datapoints:
    x = positions[:k]
    likelihood = k * np.log(beta_list/np.pi)
    for loc in x:
        likelihood -= np.log(beta_list**2 + (loc - alpha_list)**2)

    fig = plt.figure()
    ax = fig.gca(projection='3d')
    ax.plot_surface(alpha_list, beta_list, likelihood, cmap=plt.cm.viridis, vmin=-200, vmax=
    likelihood.max())
    plt.xlabel(r'$\alpha$')
    plt.ylabel(r'$\beta$')
    ax.set_zlabel('$\ln p(D | \alpha, \beta)$')
    plt.title('Log likelihood for $k = {}$'.format(k))
    plt.savefig('1_3_2_{}.png'.format(k), bbox_inches='tight', dpi=300)
    plt.show()
```

1.3.3

We can make a reasonable (visual) estimate of the most probable position of the lighthouse from the graph, after a few data points have been observed. However, as we are working with a computer, we will let matlab do the dirty work for us.

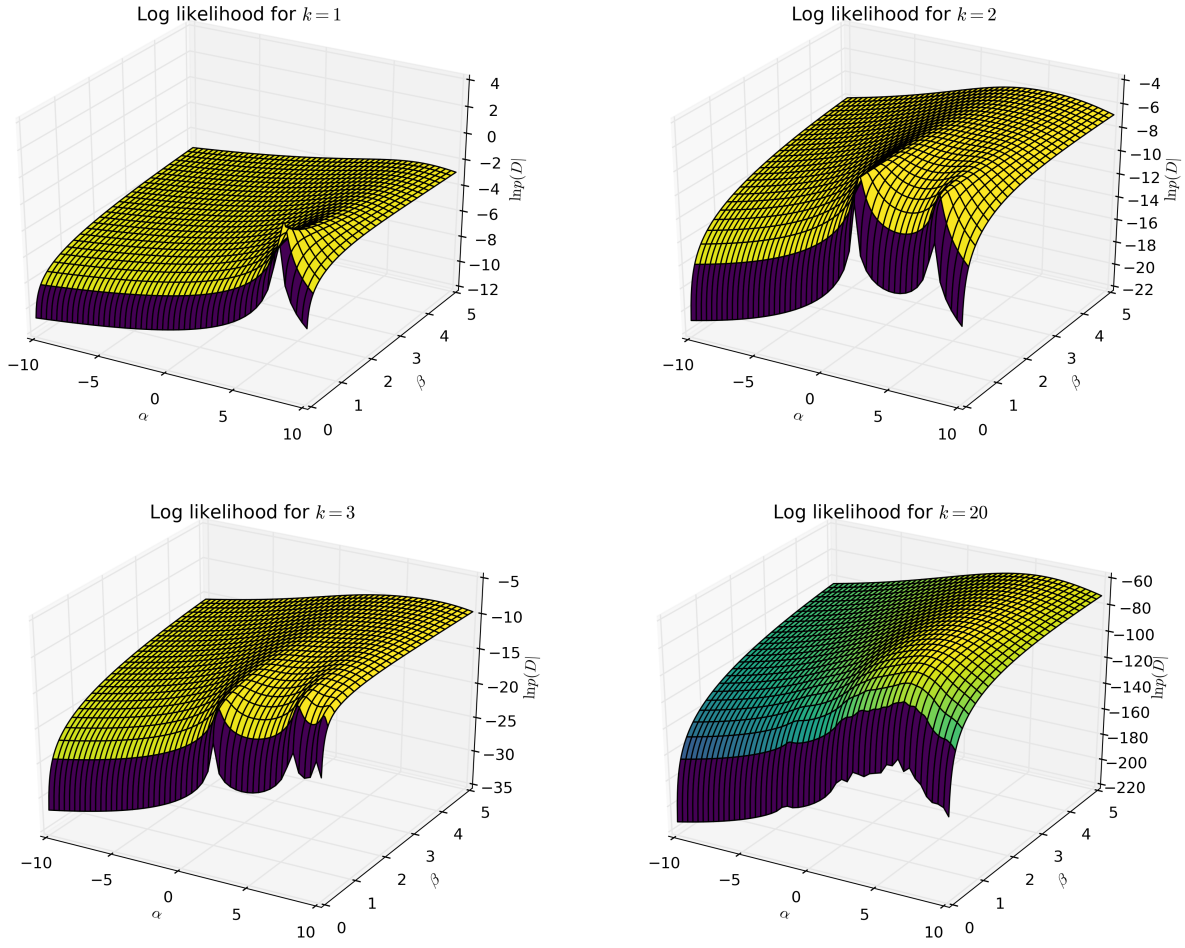

Figure 5: Log likelihood at point (α, β) after k points have arrived

Create a function that uses matlab function `fminsearch` to compute the values of α and β that maximize the likelihood for a data set of k points, and plot these as a function of the number of points. Use $[0, 1]$ as the initial starting value for `fminsearch` (see examples in matlab-help). Compare your final estimate with the true values (α_t, β_t).

Answer:

The estimated values are $\alpha_t = 4.969409930825474$ and $\beta_t = 2.1621554383456547$, which is pretty close to the real values of $\alpha = 5.022$ and $\beta = 2.436$.

```python
# Exercise 1.3.3

def log_likelihood(params, positions):
    alpha, beta = params
    likelihood = len(positions) * np.log(beta/np.pi)
    for loc in positions:
        likelihood -= np.log(beta**2 + (loc - alpha)**2)
    return -likelihood

def plot_maximize_log_likelihood(data, alpha_t, beta_t):
    alpha_list, beta_list = [], []
    x = np.arange(len(data))
    for k in x:
        [alpha, beta] = fmin(log_likelihood, (0, 1), args=(data[:k],))
        alpha_list.append(alpha)
        beta_list.append(beta)

    plt.style.use('ggplot')
    plt.plot(x,alpha_list,label=r'$\alpha$')
    plt.plot(x,beta_list,label=r'$\beta$')
    plt.plot(x,[alpha_t]*len(data),label=r'$\alpha_t$')
    plt.plot(x,[beta_t]*len(data),label=r'$\beta_t$')
    plt.xlabel('$k$')
    plt.ylabel('positions')
    plt.legend()
    plt.savefig('1_3_3.png',bbox_inches='tight',dpi=300)
    plt.show()

    print(alpha_list[-1], beta_list[-1])

plot_maximize_log_likelihood(positions, alpha_t, beta_t)
```

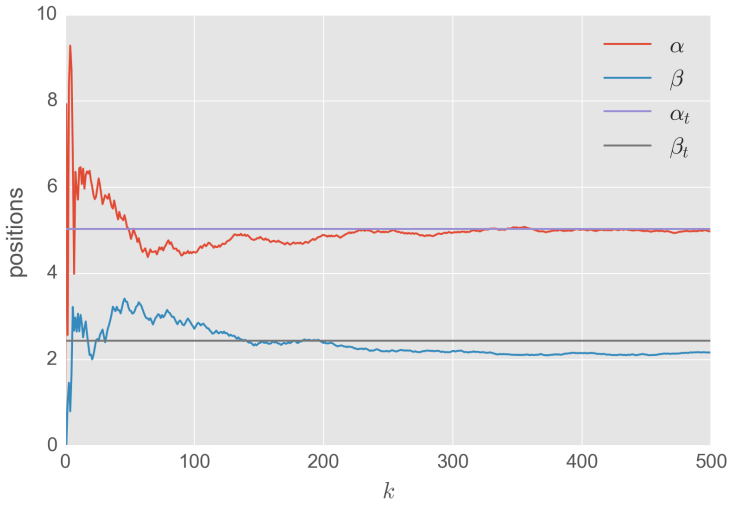

Figure 6: Maximum likelihood estimates for α and β

Exercise 2 - Bayesian linear regression (weight 2)

This exercise builds on exercise 2, week 7, "Fitting a straight line to data". For a detailed de- scription (and explanation) see Exercises and Answers, Week 7 in Brightspace. The final part of that exercise computed the

predictive distribution after a single data point was observed. Here we consider a new data set, consisting of no less than two points: $x_1, t_1 = (0.4, 0.1)$ and $x_2, t_2 = (0.6, -0.4)$.

2.1

Assume $\alpha = 1$ and $\beta = 15$. Compute the predictive distribution $p(t|x, \mathbf{t}, \mathbf{x}, \alpha, \beta)$ after these two points are observed.

Answer:

2.2

Plot the mean of the predictive Gaussian distribution and one standard deviation on both sides as a function of x over the interval $[0, 1]$. Plot the data in the same figure. See `a009plotideas.m` in Brightspace for some plotting hints. Compare your plot with Figure 3.8b (Bishop, p.157) and explain the difference.

Answer:

2.3

Sample five functions $y(x, \mathbf{w})$ from the posterior distribution over $\mathbf{w}$ for this data set and plot them in the same graph (i.e. with the predictive distribution). You may use the Matlab function `mvnrnd`. See again `a009plotideas.m` for some plotting hints.

Answer:

Exercise 3 - Gradient descent revisited (weight 3)

In this exercise, we will have a closer look at the gradient descent algorithm for function minimization. When the function to be minimized is $E(x)$, the gradient descent iteration is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \eta \nabla E(\mathbf{x}_n) \tag{5}$$

where $\eta > 0$ is the so-called learning-rate.

3.1

Consider the function $f(x) = \frac{\lambda}{2}(x - a)^2$ with parameters $\lambda > 0$, and a arbitrary.

3.1a

Write down the gradient descent iteration rule. Verify that the minimum of f is a fixed point[1] of the gradient descent iteration rule.

Answer:

The gradient of f is:

$$\nabla f(x) = \lambda(x - a) \tag{6}$$

And therefore the gradient descent iteration rule is:

$$x_{n+1} = x_n - \eta\lambda(x_n - a) \tag{7}$$

The minimum of f can be found by setting it's gradient to 0:

[1] A fixed point x^* of an iteration $x_{n+1} = F(x_n)$ satisfies $x^* = F(x^*)$.

$$\lambda(x-a) = 0 \tag{8}$$
$$x-a = 0 \tag{9}$$
$$x = a \tag{10}$$

To show it's a minimum we can take the second derivitive at point a:

$$f''(x) = \lambda \rightarrow f''(a) = \lambda \tag{11}$$

Since λ is strictly greater then 0, the second derivative at a is strictly greater then 0 and therefor a is the minimum.
a is a fixed point since:

$$x_{n+1} = a - \eta \nabla f(a) \tag{12}$$
$$= a - \eta\lambda(a-a) = a \tag{13}$$

3.1b

Find η for which convergence is the fastest (actually in one step).

Answer:

Since it is given that the fastest is in one step, we can write :

$$a = x - \eta\lambda f(x) \tag{14}$$
$$= x - \eta\lambda(x-a) \tag{15}$$
$$\eta\lambda(x-a) = x-a \tag{16}$$
$$\eta\lambda = 1 \tag{17}$$
$$\eta = \frac{1}{\lambda} \tag{18}$$

You can also see this by looking at the iteration rule. Since when $\eta = \lambda^{-1}$, in the next iteration x_n will cancel and you're left with (-(-a))=a, thus arriving at the minimum in one step, independent of x.

3.1c

We will investigate the convergence properties for different values of η. For this we look at the ratio of the distance to the fixed point after the step and the distance before the step:

$$r_n = \frac{|x_{n+1} - x^*|}{|x_n - x^*|} \tag{19}$$

3.1c.i

What does it mean if all $r_n < c < 1$ (i.e. all ratio's are smaller than a certain number below 1). Give for this case the (optimal) upper bound of the distance $|x_n - x^*|$ in terms of $|x_0 - x^*|$, c and n.

Answer:

If for all $r_n < c < 1$ then it means that always, independent of the iteration step, beginning starting point x_0, the ratios of the next iteration and the current are always lower then some number strictly smaller then 1. This means that the distance r_n to the fixed point always decreases with a rate bounded by c. It also means that every new step brings you closer to a, it's not like there is some other fixed point you might end up at.

At every step the distance is multiplied by at least c, since the difference between the distance of the current and next step is c or lower. So the upper bound on the distance would be:

$$|x_n - x^*| = c^n |x_0 - x^*| \tag{20}$$

Note that $c^n < 1$ since $0 < c < 1$.

3.1c.ii

What is the consequence if all $r_> c > 1$? Give for this case the optimal lower bound of the distance $|x_n - x^*|$ in terms of $|x_0 - x^*|$, c and n.

Answer:

Then with every step you get further and further away from the optimal point x^*. Therefore you'll diverge from the point x^*. The constant c is larger than 1, so it is bounded by 1. While c can't be 1, it is the lower bound. Taking this we get that the best case scenario the distance won't change, so we get:

$$|x_n - x^*| = |x_0 - x^*| + \epsilon \tag{21}$$

With $0 < \epsilon$.

3.1d

Show that in our case, $r_n = |1 - \eta\lambda| \equiv r$, independent of n (we refer to r as the convergence rate). For which η is the algorithm convergent?

Answer:

$$\begin{aligned} r_n &= \frac{|x_{n+1} - x^*|}{|x_n - x^*|} && (22)\\ &= \frac{|x_n - \eta\lambda(x_n - a) - a|}{|x_n - a|} && (23)\\ &= \frac{|(1 - \eta\lambda)(x_n - a)|}{|x_n - a|} && (24)\\ &= |1 - \eta\lambda| && (25) \end{aligned}$$

This converges for when r_n is strictly smaller then 1, so when $0 < \eta\lambda < 2$. So:

$$0 < \eta < \frac{2}{\lambda} \tag{26}$$

3.2

Consider the function $g(x, y) = \frac{\lambda_1}{2}(x - a_1)^2 + \frac{\lambda_2}{2}(y - a_2)^2$ with parameters $0 < \lambda_1 \leq \lambda_2$, and a_i arbitrary.

3.2a

Write down the gradient descent iteration rule. Verify that the minimum of f is a fixed point.

Answer:

The gradient of the function g is:

$$\frac{\partial g}{\partial x} = \lambda_1(x - a_1) \tag{27}$$
$$\frac{\partial g}{\partial y} = \lambda_2(y - a_2) \tag{28}$$
$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} \lambda_1(x - a_1) \\ \lambda_2(y - a_2) \end{bmatrix} \tag{29}$$
$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \eta \nabla g(x_n) \tag{30}$$
$$= \begin{bmatrix} x_n \\ y_n \end{bmatrix} - \eta \begin{bmatrix} \lambda_1(x_n - a_1) \\ \lambda_2(y_n - a_2) \end{bmatrix} \tag{31}$$

The minimum of g is: (I think the f was a typing error)

$$\nabla g = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{32}$$
$$= \begin{bmatrix} \lambda_1(x - a_1) \\ \lambda_2(y - a_2) \end{bmatrix} \tag{33}$$
$$\lambda_1(x - a_1) = 0 \rightarrow x = a_1 \tag{34}$$
$$\lambda_2(y - a_2) = 0 \rightarrow y = a_2 \tag{35}$$
$$\tag{36}$$

Now to show it is a fixed point:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} - \eta \begin{bmatrix} \lambda_1(a_1 - a_1) \\ \lambda_2(a_2 - a_2) \end{bmatrix} \tag{37}$$
$$= \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{38}$$

3.2b

Show that $\eta = \frac{2}{\lambda_2 + \lambda_1}$ minimizes the larger ratio between $r_{n,x} = \frac{|x_{n+1} - x^*|}{|x_n - x^*|}$ and $r_{n,y} = \frac{|y_{n+1} - y^*|}{|y_n - y^*|}$, i.e., $\arg\min_\eta \{max\{r_{n,x}, r_{n,y}\}\} = \frac{2}{\lambda_2 + \lambda_1}$. What happens if η is smaller than this optimal value? What happens if it is larger?

Answer:

For the first ratio:

$$r_{n,x} = \frac{|x_{n+1} - x^*|}{|x_n - x^*|} \tag{39}$$
$$= \frac{|x_n - \eta\lambda_1(x_n - a_1) - a_1|}{|x_n - a_1|} \tag{40}$$
$$= \frac{|(1 - \eta\lambda_1)(x_n - a_1)|}{|x_n - a_1|} \tag{41}$$
$$= |1 - \eta\lambda_1| \tag{42}$$

For the second ratio:

$$r_{n,y} = \frac{|y_{n+1} - y^*|}{|y_n - y^*|} \tag{43}$$
$$= \frac{|y_n - \eta\lambda_2(y_n - a_2) - a_2|}{|y_n - a_2|} \tag{44}$$
$$= \frac{|(1 - \eta\lambda_2)(y_n - a_2)|}{|y_n - a_2|} \tag{45}$$
$$= |1 - \eta\lambda_2| \tag{46}$$

When they are at the minimum, the values are equal. For if one is not equal but higher you can always go in one direction to lower the first until it lowers so much it equals the second. This is easier to understand by looking at an example plot, here $\lambda_1 = 1$ and $\lambda_2 = 2$.

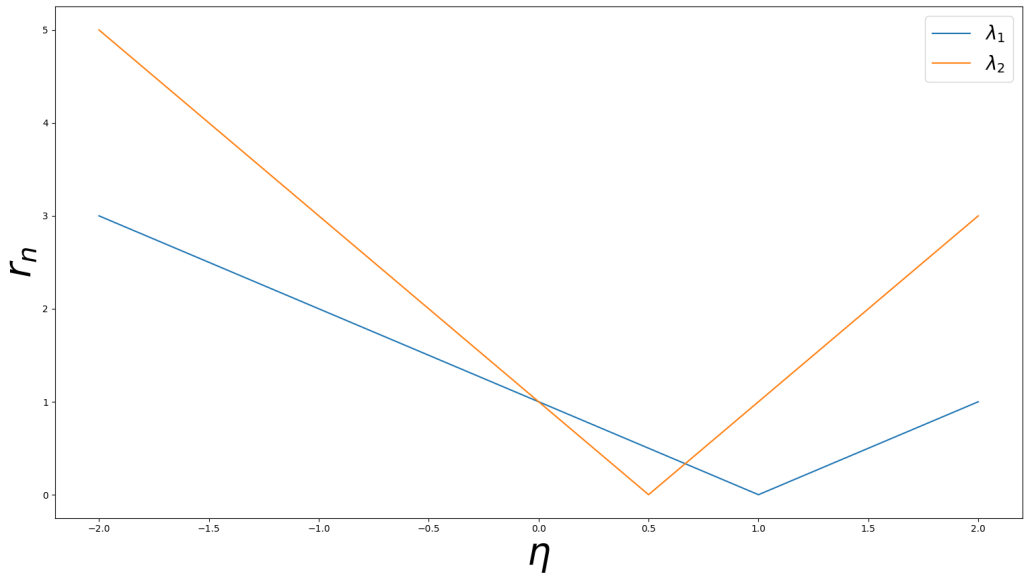

Figure 7: Two ratios for $\lambda_1 = 1$ and $\lambda_2 = 2$. Notice that the optimal value is found at the second intersection, where $\eta \neq 0$, for the highest value of both of the functions is the lowest, moving left or right of that point either one is higher then at the intersection.

$$
\begin{aligned}
|1 - \eta\lambda_1| &= |1 - \eta\lambda_2| && (47)\\
(1 - \eta\lambda_1)^2 &= (1 - \eta\lambda_2)^2 && (48)\\
1 - \eta\lambda_1 &= \pm(1 - \eta\lambda_2) && (49)\\
1 - \eta\lambda_1 &= 1 - \eta\lambda_2 && (50)\\
\eta\lambda_2 - \eta\lambda_1 &= 0 && (51)\\
\eta &= 0 \qquad \text{Not the lowest solution, let's try the -} && (52)\\
1 - \eta\lambda_1 &= \eta\lambda_2 - 1 && (53)\\
2 &= \eta(\lambda_1 + \lambda_2) && (54)\\
\eta &= \frac{2}{\lambda_1 + \lambda_2} && (55)
\end{aligned}
$$

We can say without loss of generality that $\lambda_1 < \lambda_2$. If that's not the case we can either swap lables. If they are the same then the ideal rate is optimal in both dimensions, and using an η different then the optimal will decrease the converging rate for both directions x and y.
As we can see in figure 7, a value of η which is lower then the optimal will make y converge faster, though if it's negative it would make x diverge not as fast. If η is greater then the optimal value, x will converge faster and after some point diverge slower then y.

3.2c

What is the convergence rate for this η? What does this say about the applicability of gradient descent to functions with steep hills and flat valleys (i.e., if $\lambda_2 \gg \lambda_1$)?

Answer:

The convergence rate for x and y will be:

$$r_{n,x} = \left|1 - \frac{2\lambda_1}{\lambda_1 + \lambda_2}\right| \tag{56}$$

$$r_{n,y} = \left|1 - \frac{2\lambda_2}{\lambda_1 + \lambda_2}\right| \tag{57}$$

In the case that $\lambda_2 \gg \lambda_1$, we can say that $\lambda_1 = \epsilon\lambda_2$. Then we can take the limit for $\epsilon \to 0$.

$$\lim_{\epsilon\to 0}(1+\epsilon) = 1 \tag{58}$$

$$r_{n,x} = \left|1 - \frac{2\epsilon\lambda_2}{(1+\epsilon)\lambda_2}\right| \tag{59}$$

$$\lim_{\epsilon\to 0} r_{n,x} = \lim_{\epsilon\to 0}\left|1 - \frac{2\epsilon\lambda_2}{(1+\epsilon)\lambda_2}\right| \tag{60}$$

$$= \lim_{\epsilon\to 0}\left|1 - \frac{2\epsilon\lambda_2}{\lambda_2}\right| \tag{61}$$

$$= \lim_{\epsilon\to 0}|1 - 2\epsilon| \tag{62}$$

$$= 1 \tag{63}$$

Now for y:

$$r_{n,y} = \left|1 - \frac{2\lambda_2}{(1+\epsilon)\lambda_2}\right| \tag{64}$$

$$\lim_{\epsilon\to 0} r_{n,y} = \lim_{\epsilon\to 0}\left|1 - \frac{2\lambda_2}{(1+\epsilon)\lambda_2}\right| \tag{65}$$

$$= \lim_{\epsilon\to 0}\left|1 - \frac{2\lambda_2}{\lambda_2}\right| \tag{66}$$

$$= |1 - 2| = 1 \tag{67}$$

So it won't converge it either direction.

3.2d

Implement the gradient descent algorithm for g in matlab.

Answer:

3.2e

Make some plots of the trajectories $\{(x_n, y_n)\}_{n=0}^{N}$ for different values of λ_i and η (η optimal, larger than optimal, and smaller than optimal) to illustrate what is going on. Plot these trajectories on top of a contour plot of g. Monitor the convergence rates. Explain what happens.

Answer:

Let's first take an example from what we saw in 3.2c. We've chosen $\lambda_1 = 10^7$ and $\lambda_2 = 1$. We've taken η to be the optimal value ($\frac{2}{\lambda_1+\lambda_2}$). $x_0 = 3$ and $y_0 = 4$ for all the situations.

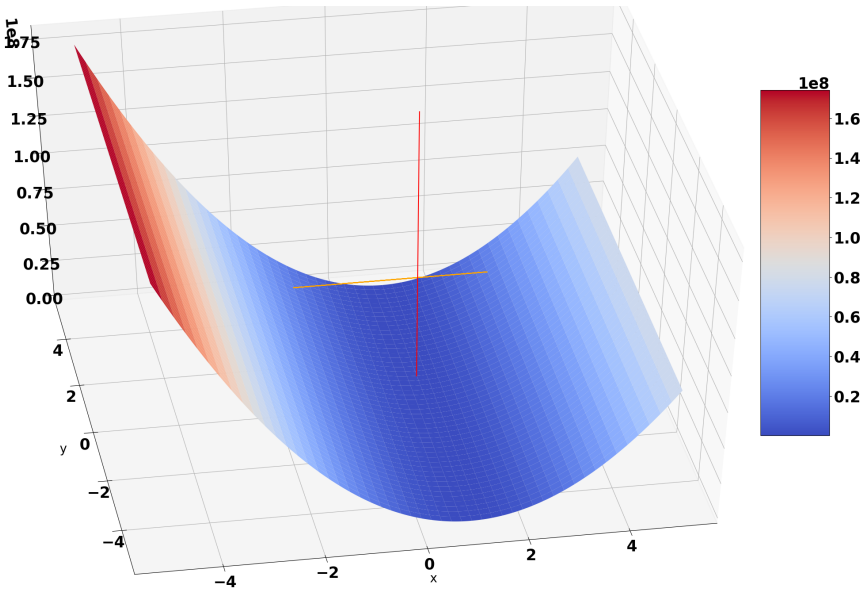

Figure 8: Gradient descent for high λ_1. Notice how high the hill is (10^8). In the x direction it hops between ≈ 3 and $\approx$ -1. y stays at ≈ 4.

What happens at 8 is that in x, $\eta\lambda$ equals 2. So it would need to go to x=1, but it overshoots by going exactly twice the required amount. So it will land on $3 - (2 \cdot -2) = -1$. From there it will overshoot going back and it will land again on $x \approx 2$. For y $\eta\lambda$ almost equals 0, so it doesn't go anywhere and stays at y=4. I use approximate because it doesn't go exactly to -1, it first for examples goes to -0.9999996000000402. This is because it will have this behaviour in the event of $\lambda_1 \to \infty$ and we're not quite there yet for $\lambda_1 = 10^7$, but we can get arbitrary close to it if we make λ_1 larger. (This all assumes λ_2 stays the same)

Another way to make it stuck forever (this time truly forever) is by chosing η in such a way that it will always overshoot. If we take $\lambda_1 = \lambda_2 = \lambda$, then $\eta = \frac{2}{\lambda}$ will give the desired result, since then $r_n = |1 - \eta\lambda| = |1 - 2| = 1$. An example is that for $\lambda = 1$ and $\eta = 2$. This you can see in figure 9.

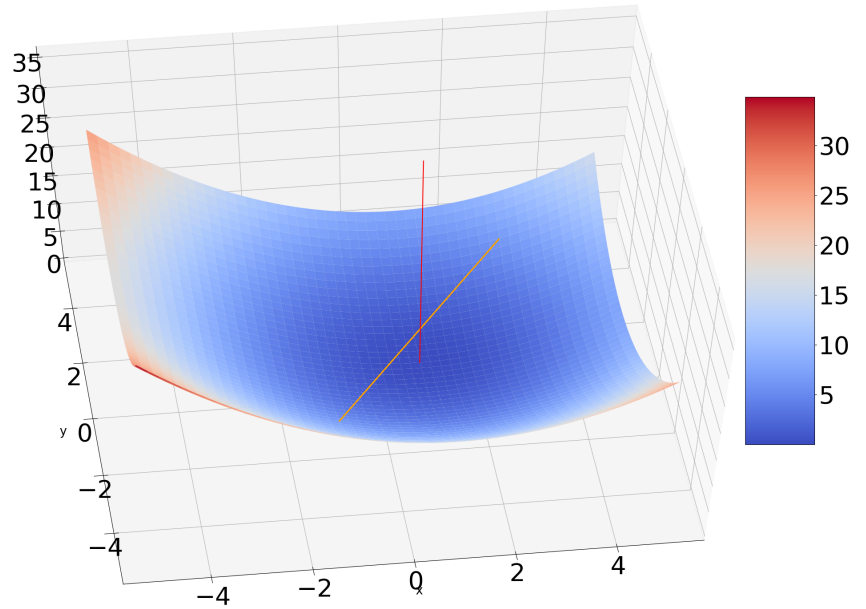

Figure 9: Gradient descent for $\lambda_1 = \lambda_2 = 1$ and $\eta = 2$. Now the distance to the optimal point won't increase or decrease, and it jumps to and from the two only reachable points.

Now let's look at two examples of converging paths. If we set η at the optimal value it can converge in one step if the $\lambda_1 = \lambda_2$. Then for both directions the ideal η is the same. We took $\lambda_1 = \lambda_2 = 1$ and the ideal number for η. Then we get the following graph as seen in figure 10.

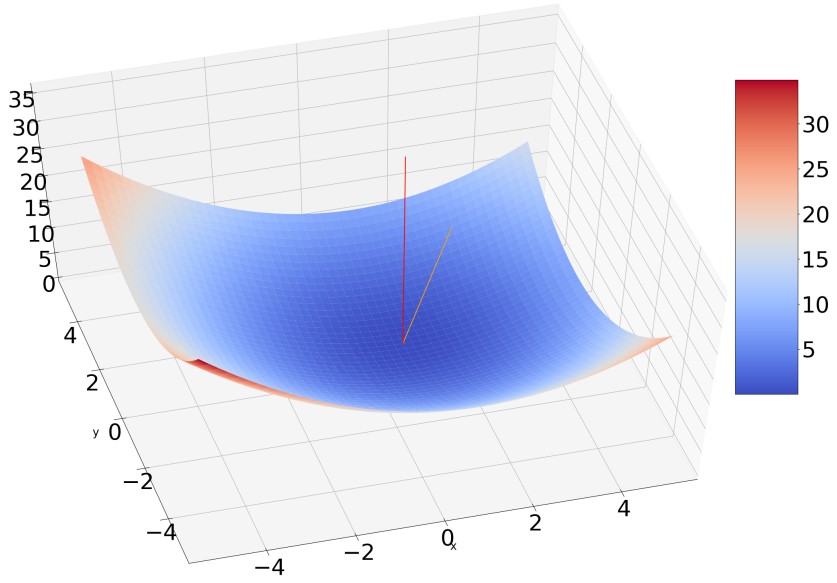

Figure 10: Gradient descent for $\lambda_1 = \lambda_2 = 1$ and η the optimal value of 1. It converges in one step.

Now if the $\lambda_1 \neq \lambda_2$ we won't have a convergence. For both $r_{n,x}$ and $r_{n,y}$ will have a fixed value between 0 and 1, but not quite 0. So even after 10^5 iterations we won't get the value to be exactly at the minimum, but ended up at x = 1.0000000000000002 and y = 1.0000000000000004 . In this case you could however say you're satisfied if you're within a certain range of the minimum. This we tested with $\lambda_1 = 10$, $\lambda_2 = 1$ and η optimal.

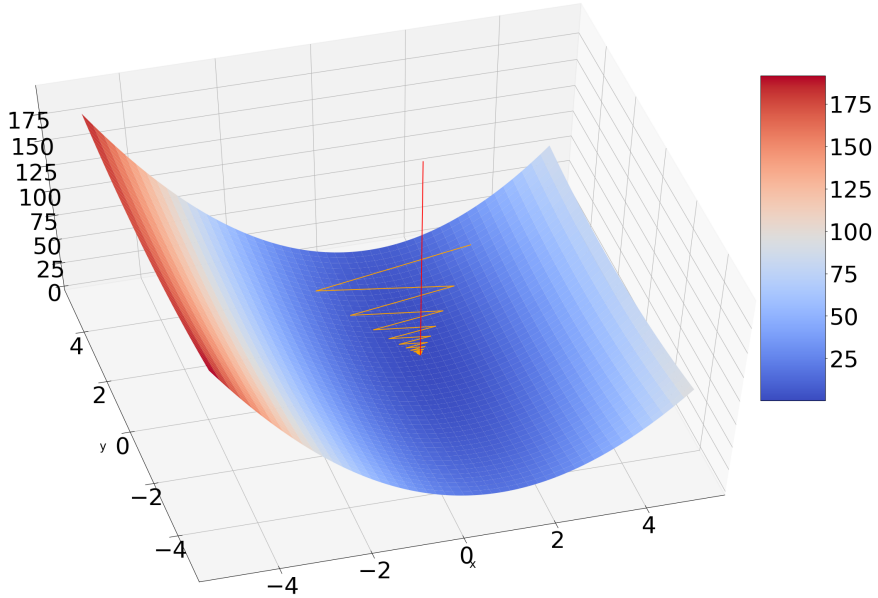

Figure 11: Gradient descent for $\lambda_1 = 10$ and $\lambda_2 = 1$, with an optimal η. Notice how it overshoots for x, since $1 - \eta\lambda_1 < 0$. Vice versa it slowly converges in the y direction. This is because the if the subtraction $x_{n+1} - x^*$ flips sign, it will be on the other side of x^*. Therefore it will overshoot. And vice versa.

We can also have the same situation as before, where it converges, but $\eta = \frac{\eta_{optimal}}{10} = \frac{1}{10}\frac{2}{\lambda_1+\lambda_2} = \frac{2}{110} = \frac{1}{55}$. Then we will have a slower converge. For example, in 100 steps it will still be at a distance of $r = |x_n - x^*| + |y_n - y^*| \approx 0.48$ while at the optimal rate it would be at distance $r \approx 9.6 \cdot 10^{-9}$. This you can see at figure 12:

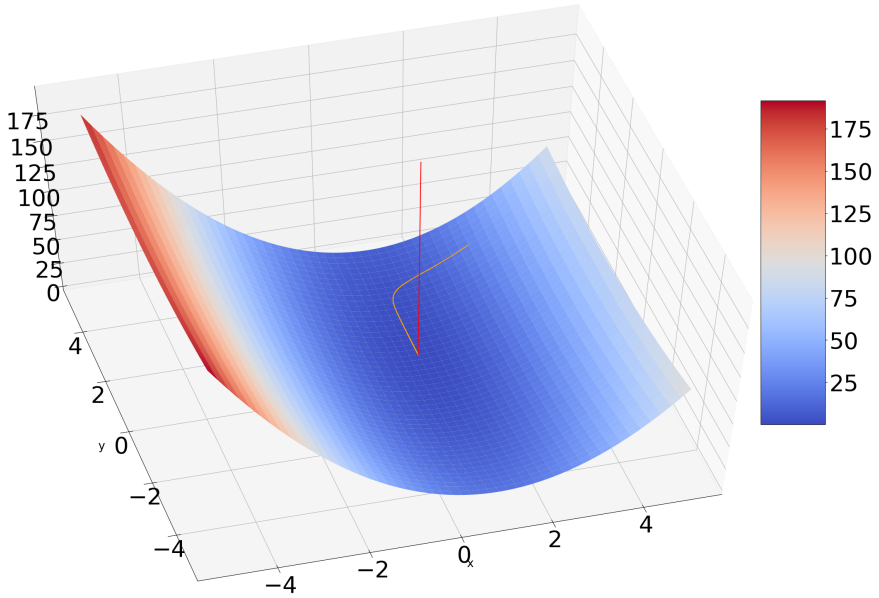

Figure 12: Gradient descent which converges for the same parameters as at 10, but with $\eta = \frac{\eta_{optimal}}{10}$. It looks smoother but it doesn't converge as fast.

What can also happen is that it converges in one dimension but diverges in another. This happens if $r_n = |1 - \eta\lambda|$ is smaller then 1 for one direction and greater then 1 for the other. For example, this happens for $\eta = 3$, $\lambda_1 = 1$ and $\lambda_2 = 0.5$. Then we will have $r_{n,x} = |1 - 3| = 2$ and $r_{n,y} = |1 - 1.5| = 0.5$. And since the sum in de absolute argument is negative for both, they will both jump over the minimum, but this is always the case for diverging paths since η and λ are both positive. This you can see in figure 13, where I have chosen the values for the $\lambda's$ I just mentioned, and the starting points are $x_0 = 3$ and $x_y = 90$, so it is more evident it converges in y and diverges in x. Also, in this case η should have been lower, it should have been $\eta_{optimal} = \frac{2}{\lambda_1+\lambda+2} = \frac{2}{1.5} < 2 < 3$.

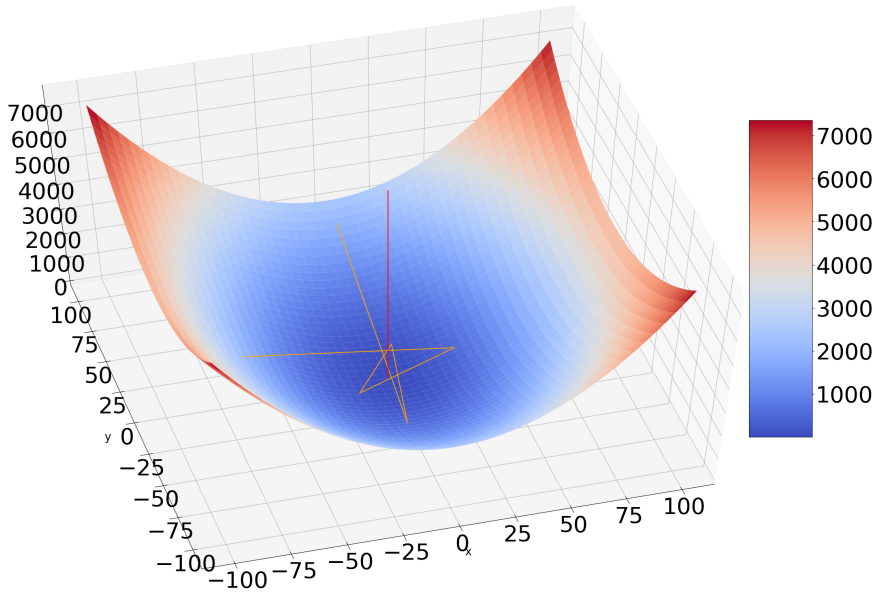

Figure 13: Gradient descent where it diverges in x and converges in y.

Now the final situation is when it diverges in both directions. If I fix $\eta = 1$ then it would happen for $\lambda_1 = 3$, $\lambda_2 = 2.5$. This you can see in figure 14.

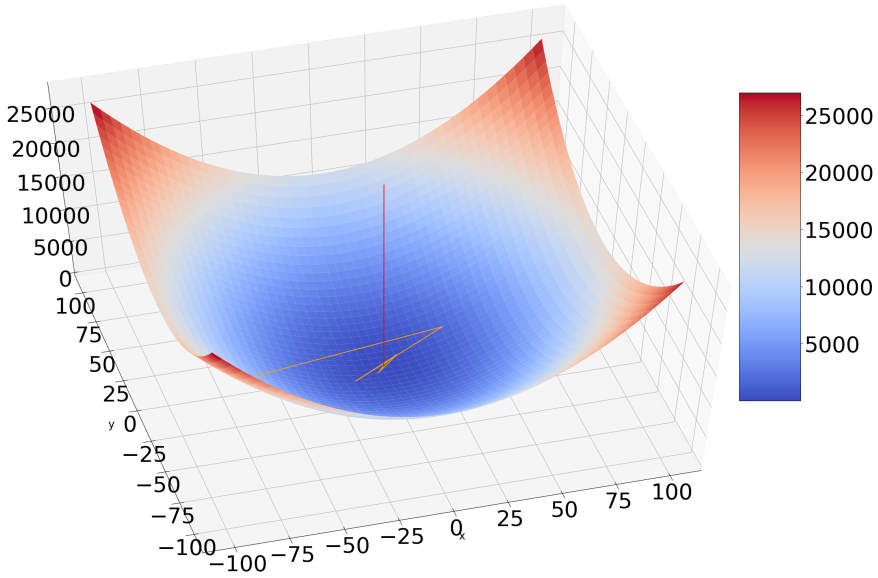

Figure 14: Gradient descent where it diverges in both x and y since $\lambda_1 = 3$, $\lambda_2 = 2.5$ and $\eta = 1$.