AB-EM

November 15, 2017

1 Expectation Maximization

2 Aufgabe 1

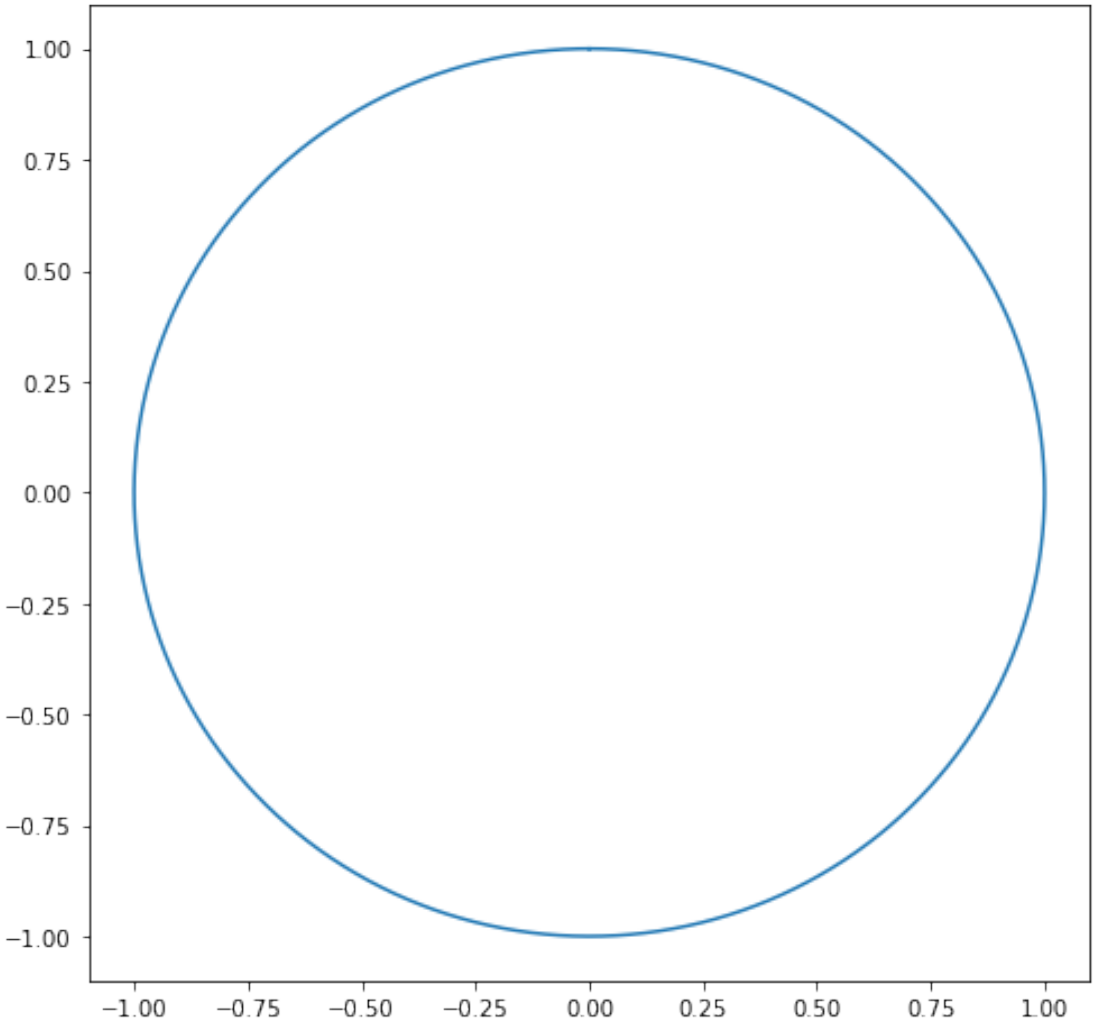
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
from numpy import pi, sin, cos
```

In [3]:
```
num_points = 1000
radius = 1

arcs = np.linspace(0, 2 * pi, num_points)
x = radius * sin(arcs)
y = radius * cos(arcs)
```

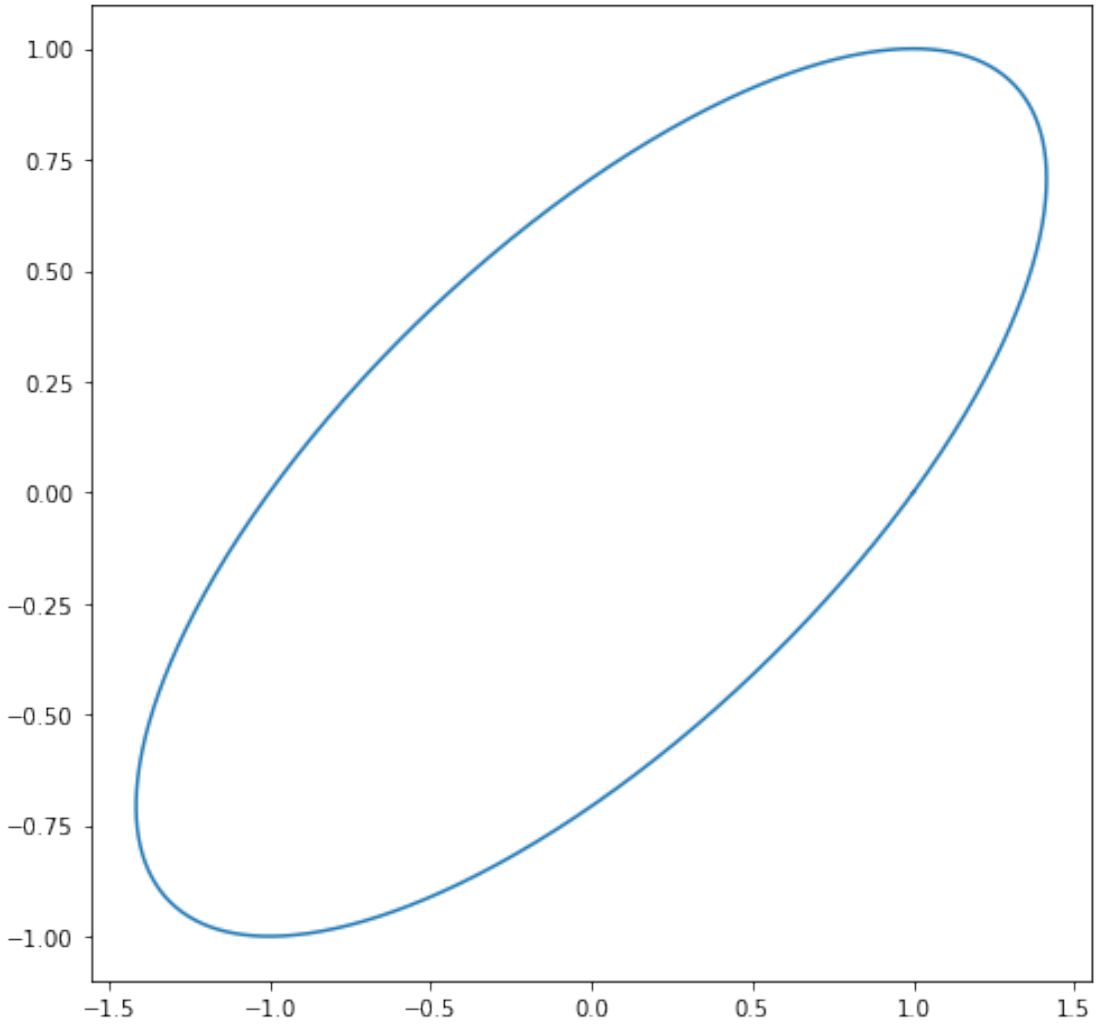
In [4]:
```
plt.figure(figsize=(8, 8))
plt.plot(x, y)
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x11193d750>]
```

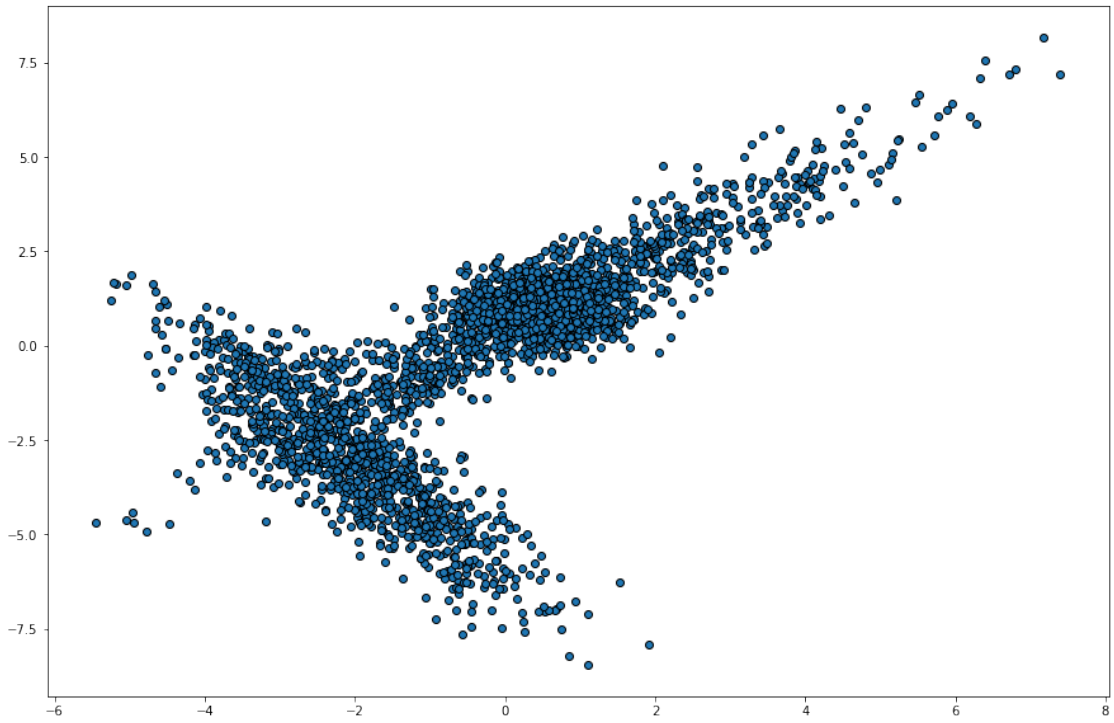


Um die Kovarianzmatrix als Ellipse darzustellen, kann man die einzelnen Punkte mit der Kovarianzmatrix transformieren. Dafür speichern wir die Bildpunkte in eine Matrix xy.

```
In [5]: cov = np.array([
            [1, 1],
            [1, 0]
        ])

In [6]: xy = np.array(zip(x, y))
        x, y = zip(*xy.dot(cov))

In [7]: plt.figure(figsize=(8, 8))
        plt.plot(x, y)

Out[7]: [<matplotlib.lines.Line2D at 0x111bbd590>]
```




```
In [8]: X = pd.read_csv("data/2d/2d-em.csv", header=None).as_matrix()

In [9]: x, y = zip(*X)
        plt.figure(figsize=(15, 10))
        plt.scatter(x, y, edgecolors="black")

Out[9]: <matplotlib.collections.PathCollection at 0x111d82ad0>
```



2.1 Aufgabe 2

Um die Pixel eines Bildes zu importieren, kann man in Python beispielsweise `mpimg` von `matplotlib` benutzen. Das Ergebnis ist eine Matrix mit den einzelnen Bildpunkten.

```
In [10]: import matplotlib.image as mpimg

In [11]: path = '/Users/florian/Desktop/photo_smaller.jpg'

In [12]: img = mpimg.imread(path)
         img[:5]

Out[12]: array([[[ 17,  34,  62],
                 [ 19,  36,  64],
                 [ 21,  38,  66],
                 ...,
                 [ 12,  56, 153],
                 [ 10,  54, 151],
                 [  9,  53, 150]],

                [[ 20,  37,  63],
                 [ 21,  38,  64],
                 [ 22,  39,  67],
                 ...,
```

```
        [ 14,  58, 155],
        [ 12,  56, 153],
        [ 11,  55, 152]],

       [[ 24,  41,  67],
        [ 24,  41,  67],
        [ 23,  40,  66],
        ...,
        [ 15,  59, 156],
        [ 14,  58, 155],
        [ 14,  58, 155]],

       [[ 28,  45,  71],
        [ 26,  43,  69],
        [ 25,  42,  68],
        ...,
        [ 17,  61, 158],
        [ 16,  60, 157],
        [ 15,  59, 156]],

       [[ 30,  48,  72],
        [ 28,  46,  70],
        [ 27,  45,  69],
        ...,
        [ 16,  62, 160],
        [ 16,  62, 160],
        [ 15,  61, 157]]], dtype=uint8)
```

AnschlieSSend kann das Bild wie folgt angezeigt werden:

```
In [13]: def show_img(img):
             plt.figure(figsize=(8, 8))
             plt.imshow(img)
             plt.show()

In [14]: show_img(img)
```

Zum Clustering ist es hilfreich diese Matrix noch in einen passenden Shape umzuwandeln:

```
In [15]: rows = img.shape[0]
         cols = img.shape[1]

         img = img.reshape(img.shape[0] * img.shape[1], 3)

In [16]: img[:10]

Out[16]: array([[17, 34, 62],
                [19, 36, 64],
                [21, 38, 66],
                [20, 37, 65],
                [16, 33, 61],
                [14, 31, 59],
                [11, 31, 58],
                [11, 31, 58],
                [11, 30, 60],
                [12, 31, 61]], dtype=uint8)

In [17]: img.shape

Out[17]: (43776, 3)
```

AnschlieSSend kann der Datensatz direkt zum Clustering benutzt werden. Zum Visualisieren muss der ursprüngliche Shape verwendet werden.