Algorithmen in der Bioinformatik

Übung 03

by Alen Kocaj (S1510458019)

7 December 2017

2. K-Means Clustering

Implementieren Sie in einer Programmiersprache Ihrer Wahl den k-means Clustering Algorithmus. Zeigen Sie anhand Ihrer Implementierung, wie die Cluster-Zentren iterativ modifiziert werden. Implementieren Sie weiters die Ausgabe wichtiger Information bzw. Kennzahlen (mqe, Davies-Bouldin Index, ...), die Sie aus der Vorlesung kennen. Verwenden Sie sinnvoll erstellte Test-Daten, um die Funktionsweise Ihrer Implementierung zu demonstrieren.

Diese Aufgabe wurde in Python umgesetzt, da die mathematische Library Numpy sehr viele Vorteile bietet und leichter zu bedienen ist, als alles selbst zu schreiben.

Die Struktur der Aufgabe besteht aus den folgenden Files:

- main.py
- optimize.py
- lib/__init__.py
- lib/cluster.py
- lib/distance.py
- lib/kmeans.py

Kurze Erklärung der einzelnen Files:

- main.py: präsentiert das Hauptprogramm, welches Daten generiert, Kommandozeilenparameter einliest, den Kmeans Algorithmus startet und anschließend die Qualitätsmaße ausgibt.
- optimize.py: wird verwendet, um die optimate Anzahl an Cluster K zu schätzen. Für diese Aufgabe nicht von belang.
- lib/__init__.py: indiziert, dass dieses Verzeichnis ein Python Package ist
- lib/cluster.py: implementiert eine Clusterstruktur, so wie sie im Kmeans verwendet wird
- lib/distance.py: definiert Hilfsfunktionen für Distanzmaße, z.b.: Euclidian.
- lib/kmeans.py: beinhaltet die eigentliche Kmeansimplementierung

Das Programm arbeitet hauptsächlich mit Kommandozeilenargumenten, welche aber auch über einen Defaultvalue verfügen.

Ein Beispielaufruf, wie in main.py dargestellt: "$ python main.py -plot -N 100 5". Wird dieser Befehl ausgeführt, erzeugt das Programm zunächst Inputdaten und wendet anschließend Kmeans für ein bestimmtes K darauf an.

Das Ergebnis auf der Konsole schaut primär so aus:

```
============================================================
[i] Starting k-means algorithm
# Number of samples from normal distribution: 100
# Min size: 0.0
# Max size: 10000.0
# Dimensionality of data: 2
# Number of clusters K: 5
============================================================
[i] Seeding centers
[[ -4.47017309   6.19981885]
 [  5.89956309  12.57676366]
 [  3.10092465   7.93631477]
 [ 14.82723076  11.0310276 ]
 [ -0.59909093  11.38339237]]
============================================================
[i] Save plots at 'iterations'
============================================================
# 0 Iteration
[[ -5.18512695  -1.96832257]
 [  6.59732214  14.10606062]
 [  5.62167811   1.14092711]
 [ 13.91765934   6.28053497]
 [ -1.79180703  11.50475841]]
> Average distance (MQE) 6.210325400694229
# 1 Iteration
[[ -4.78504985  -5.18525279]
 [  7.1264866   14.12833863]
 [  6.01321581  -0.94535902]
```

```
 [ 12.27443048   6.22798863]
 [ -2.10921602   9.65083827]]
> Average distance (MQE) 5.400715762195027
# 2 Iteration
[[ -4.49357233  -6.13571644]
 [  7.1264866   14.12833863]
 [  6.75827449  -3.07401605]
 [ 10.69853515   6.66144401]
 [ -1.88012624   8.62948526]]
> Average distance (MQE) 5.164838002825965
# 3 Iteration
[[ -4.88095562  -5.96060556]
 [  7.1264866   14.12833863]
 [  6.9780882   -5.48533247]
 [  9.79919541   5.98540907]
 [ -1.94131994   8.24933882]]
> Average distance (MQE) 5.021514713945871
# 4 Iteration
[[ -4.88095562  -5.96060556]
 [  6.81292565  13.90887061]
 [  7.81312009  -7.37849811]
 [  9.2950209    5.58415053]
 [ -1.80449321   7.76864682]]
> Average distance (MQE) 4.911748142704003
# 5 Iteration
[[ -4.88095562  -5.96060556]
 [  6.81292565  13.90887061]
 [  7.92996955  -7.83503914]
 [  8.86727629   5.18509012]
 [ -1.96861906   7.9302899 ]]
> Average distance (MQE) 4.90202942033714
```

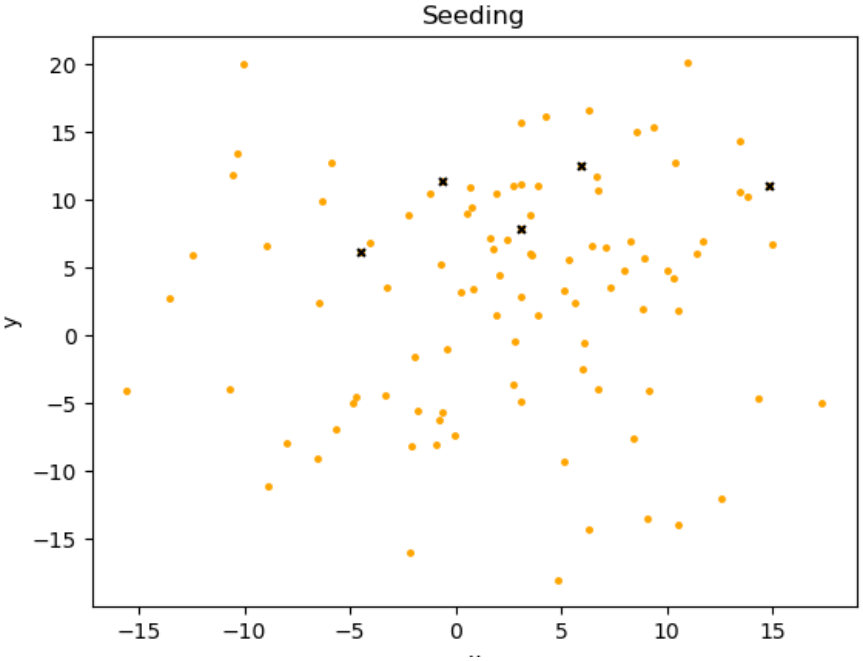
```
# 6 Iteration
[[ -4.88095562  -5.96060556]
 [  6.81292565  13.90887061]
 [  8.29769019  -8.3622851 ]
 [  8.19806572   5.02320146]
 [ -2.36243771   8.0695374 ]]
> Average distance (MQE) 4.884736798963603
# 7 Iteration
[[ -4.88095562  -5.96060556]
 [  7.72906473  13.5221184 ]
 [  8.29769019  -8.3622851 ]
 [  7.44105109   4.39311037]
 [ -2.52095984   8.3134344 ]]
> Average distance (MQE) 4.857480959186244
# 8 Iteration
[[ -4.40017352  -6.44766856]
 [  8.0858793   13.33308797]
 [  8.29769019  -8.3622851 ]
 [  6.67454393   4.47915751]
 [ -3.6139511    8.23748839]]
> Average distance (MQE) 4.834353518047147
# 9 Iteration
[[ -4.40017352  -6.44766856]
 [  8.0858793   13.33308797]
 [  8.29769019  -8.3622851 ]
 [  6.15807528   4.59876576]
 [ -4.33612811   8.59232263]]
> Average distance (MQE) 4.817630170153262
# 10 Iteration
[[ -4.40017352  -6.44766856]
 [  8.0858793   13.33308797]
```

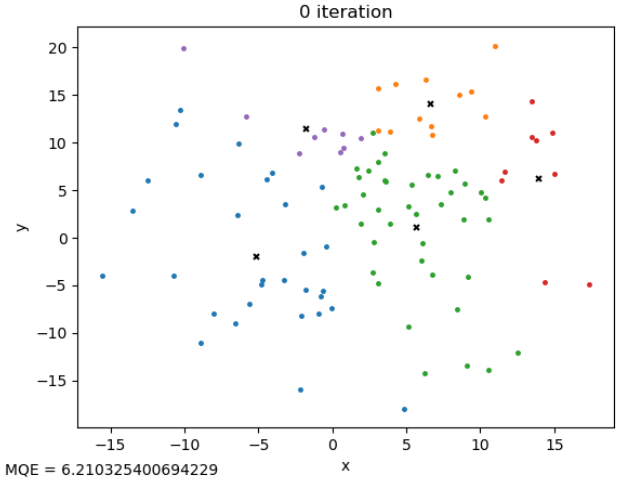
```
 [  8.29769019  -8.3622851 ]
 [  5.82208098   4.63985878]
 [ -4.86475729   8.92798415]]
> Average distance (MQE) 4.8026343233432955
# 11 Iteration
[[ -4.40017352  -6.44766856]
 [  7.74289264  13.17505928]
 [  8.29769019  -8.3622851 ]
 [  5.82208098   4.63985878]
 [ -5.221435     8.84584798]]
> Average distance (MQE) 4.799655010116306
# 12 Iteration
[[ -4.40017352  -6.44766856]
 [  7.74289264  13.17505928]
 [  8.29769019  -8.3622851 ]
 [  5.82208098   4.63985878]
 [ -5.221435     8.84584798]]
> Average distance (MQE) 4.799655010116306
============================================================
[i] Quality measures for k = 5
# Number of iterations: 13
# Empty clusters: 0
# [MQE] Mean Quantisation Error: 4.799655010116306
# [DBI] Davies Bouldin Index: 0.8853884675523028
# [SSE] Sum of Squared Errors: 2833.9829304773293
# [AIC] Akaike Information Criterion: 768.9627459488399
# [BIC] Baysian Information Criterion: 822.3687347615958
```

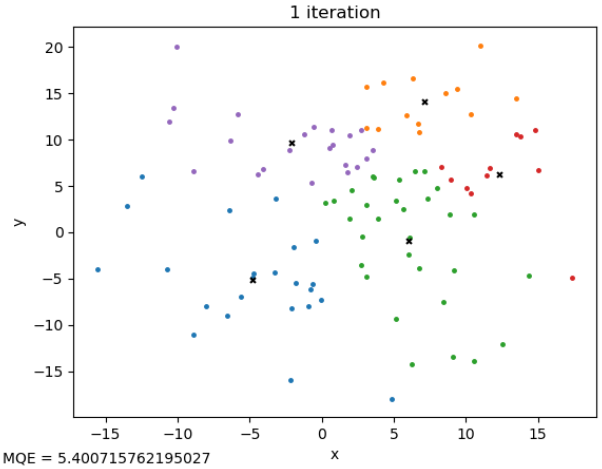
Man erkennt, wie Informationen über die Parameter gezeigt werden, bevor das eigentliche Clustering beginnt. Für jede Iteration wird der Mean Quantisation Error angezeigt und solange fortgeführt, bis sich MQE nicht mehr verbessert.

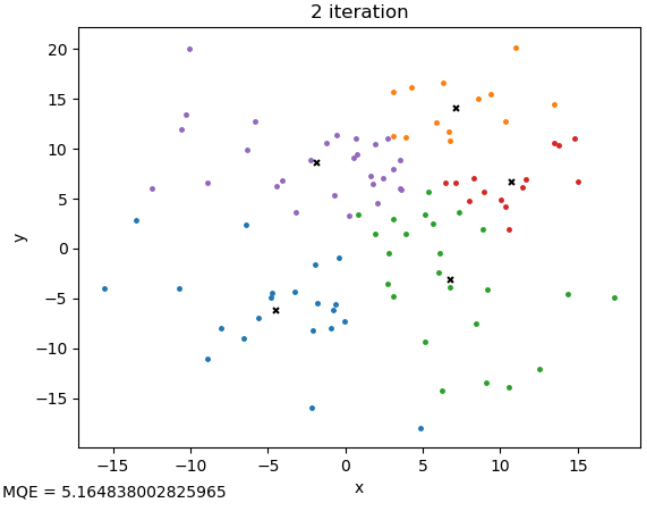
Bricht das Clustering ab, so gibt es noch eine Qualitätsmaße aus; manche mehr aussagekräftiger (MQE; DBI; SSE; AIC; BIC), manche weniger.

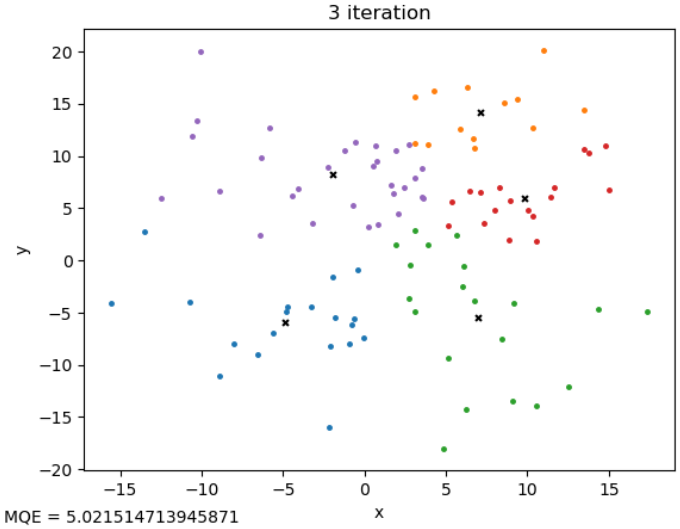
Nachdem man an das Progamm auch den "-plot" Parameter übergeben hat, wurde für jede Iteration die Daten mitsamt dem jeweiligen Clustering + Centroid als Bild abgespeichert.

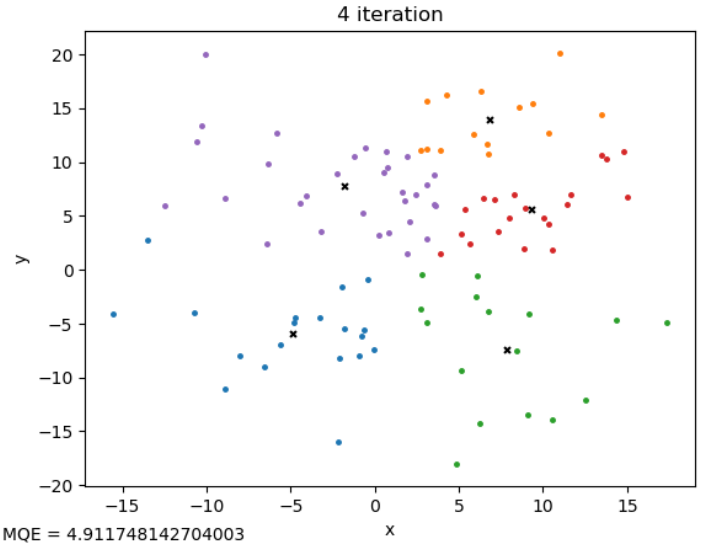




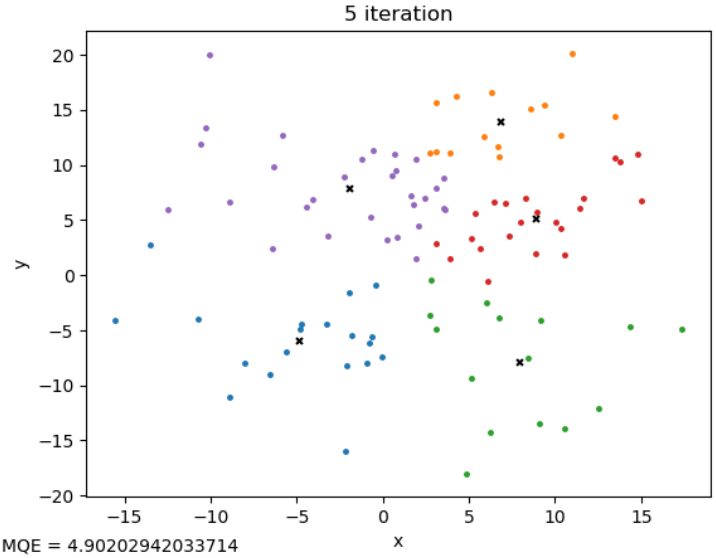




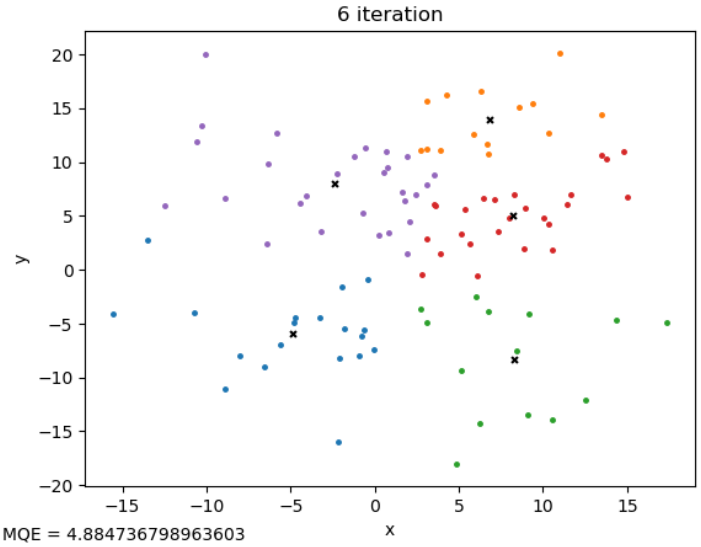




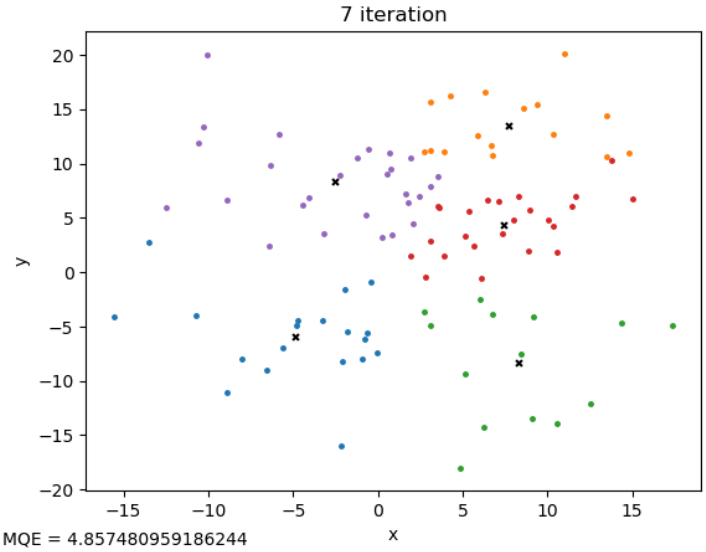
MQE = 4.911748142704003



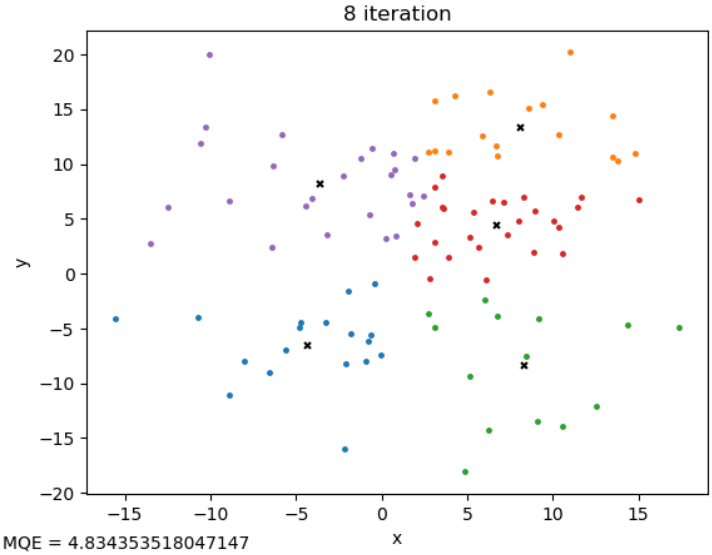
MQE = 4.90202942033714



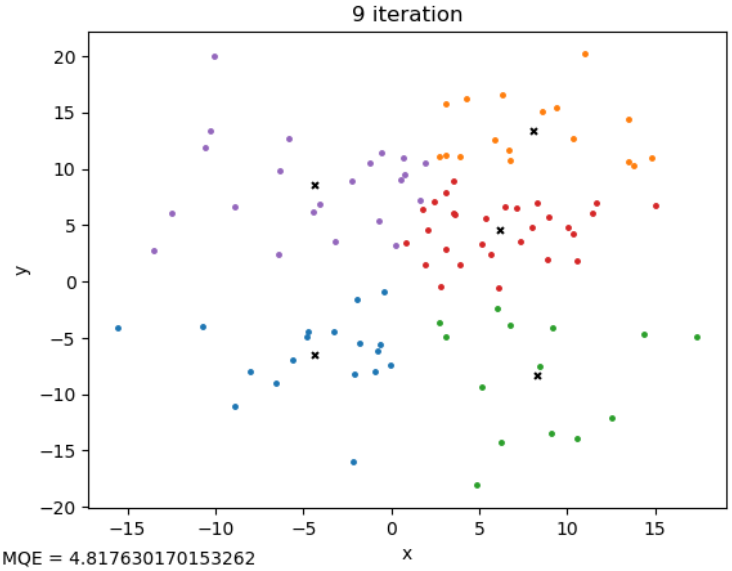
MQE = 4.884736798963603



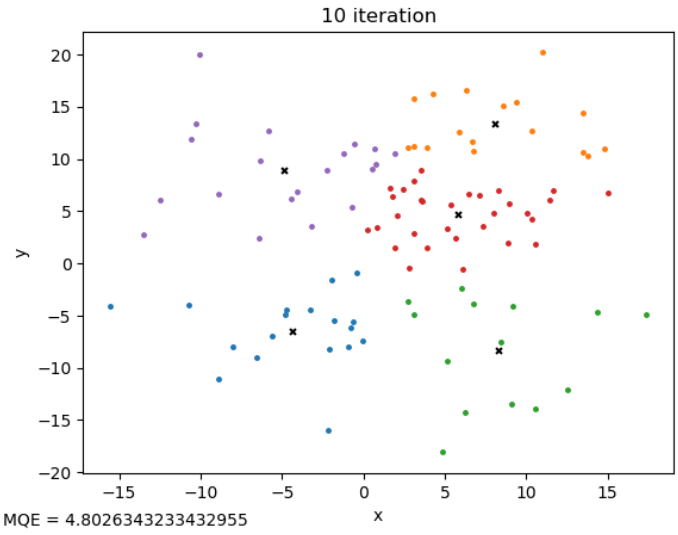
MQE = 4.857480959186244

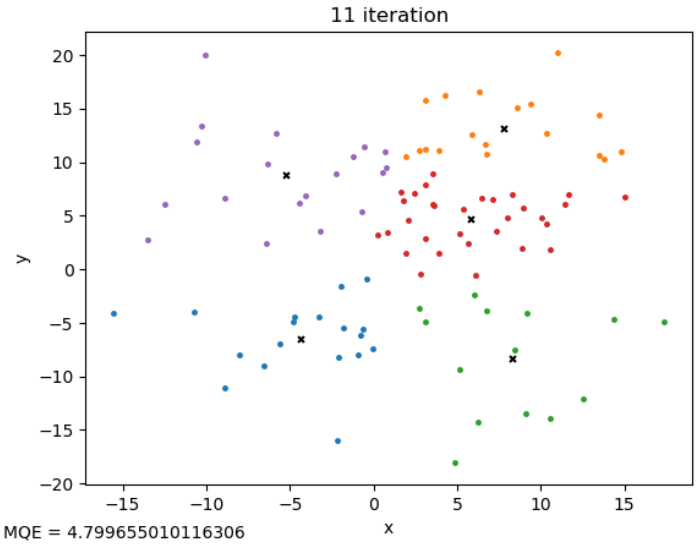


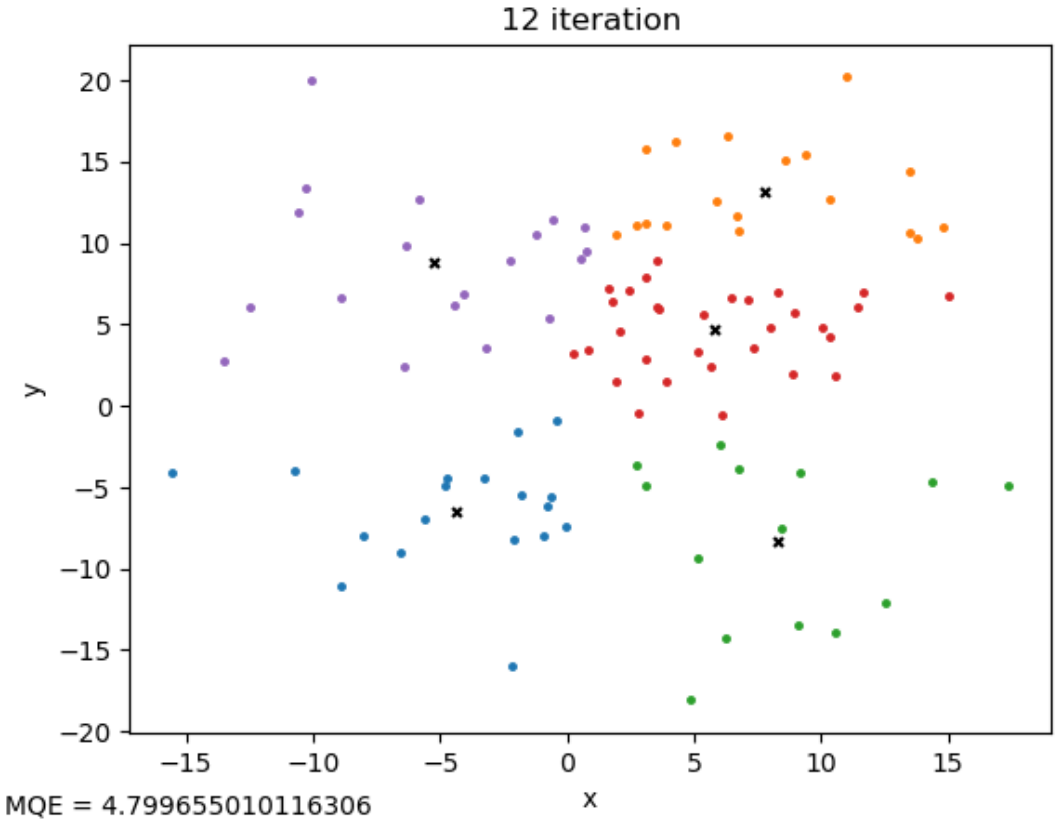
MQE = 4.834353518047147



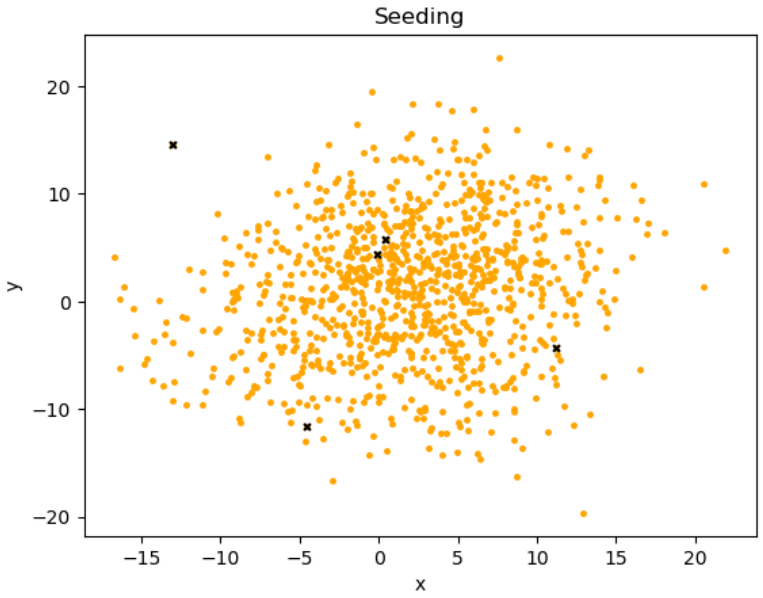
MQE = 4.817630170153262

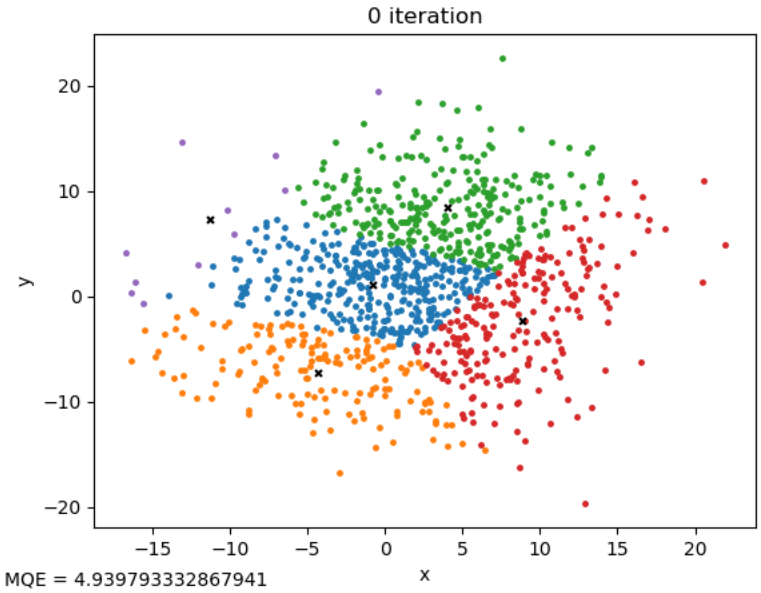

10 iteration
y
x
MQE = 4.8026343233432955

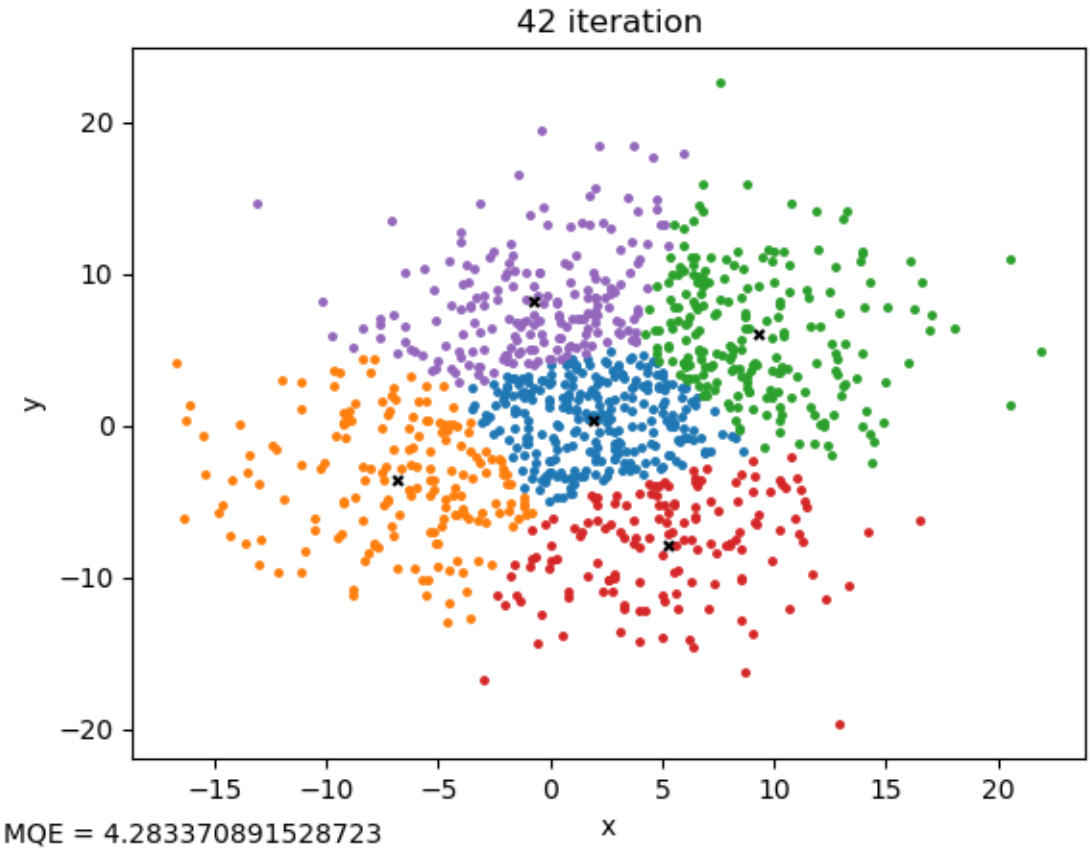

11 iteration
y
x
MQE = 4.799655010116306


12 iteration
y
x
MQE = 4.799655010116306

Ein weiteres Beispiel mit mehr Daten und der ersten / letzen Iteration.







3. Optimierung

Verbessern Sie Ihre k-means Implementierungen um eine semi-automatische (d.h. auch auf User-Input reagierende) Optimierung der Anzahl der Cluster. Verwenden Sie sinnvoll erstellte Test-Daten, um zu zeigen, wie eine optimale Anzahl der Cluster auf diese Weise gefunden werden kann.

Im 2.ten Abschnitt wurde die "optimize.py" bereits vorgestellt. Diese soll verwendet werden, um für beliebige Daten, um die optimale Anzahl der Cluser herauszufinden.

Diese Programm verwendet wie "main.py" Kommandozeilenparameter, um den Input und vor allem die Range - also von welchem K zu welchem K - auszulesen.

Innerhalb dieser Range, für das Optimierungstool den Kmeans für das jeweilge K aus und speichert sich bestimmte Qualitätsmaße. Diese Qualitätsmaße werden dann über die betrachteten K's geplottet.

Es werden folgende Qualitätsmaße verwendet:

- MQE (Mean Quantisation Error): Den durchschnittlichen Fehler, den man macht, wenn man die Daten durch ein bestimmtes Zetrum approximiert
- DBI (Davies-Bouldin Index): Ein Maß, um die geschätze Anzahl der Cluster zu der echten Anzahl an Cluster in den Daten abzuschätzen
- SSE (Sum of Squared Error): Den Abweichung der Daten zu unserem Modell, unserem Clustering
- AIC (Akaike Information Criterion): Ein Qualitätsmaß für die Model Selection, welche gute goodness-of-fit fördert und overfitting oder zu starke Komplexität bestraft. Je höher der Criterion Value, desto besser.
- BIC (Bayesian Information Criterion): Ähnlich zum Akaike, aber betraft komplexe Modelle anhand der freien Parameter und der Anzahl der Daten stärker. Je höher der Criterion Value, desto besser.
- Anzahl der leeren Cluster
- Anzahl der Iterationen

Ein Beispielaufruf wäre u.a:

$ python optimize.py -plot -N 1000 -K 2-20

Das Resultat sieht dann aus wie:

```
============================================================
[i] Optimizing k for k-means algorithm
# Number of samples from uniform distribution: 1000
# Min size: 0.0
# Max size: 10000.0
# Dimensionality of data: 2
# Clustering range: 2-20
============================================================
## K = 2
> Mean Quantisation Error (MQE): 7.144529751038992
> Davies-Bouldin Index (DBI): 1.1262494489512138
> Sum of Squared Error (SSE): 64356.64157949701
> Akaike Information Criteria (AIC): 7200.6661751348975
> Baysian Information Criteria (BIC): 7212.935563332353
> Number of Empty Clusters: 0
> Number of Iterations: 11
============================================================
## K = 3
> Mean Quantisation Error (MQE): 5.969312879882169
> Davies-Bouldin Index (DBI): 0.9410600158731137
> Sum of Squared Error (SSE): 45079.59111581351
> Akaike Information Criteria (AIC): 7254.301522602077
> Baysian Information Criteria (BIC): 7286.201931915461
> Number of Empty Clusters: 0
> Number of Iterations: 9
============================================================
## K = 4
> Mean Quantisation Error (MQE): 5.173463115649389
> Davies-Bouldin Index (DBI): 0.912437477960334
```

> Sum of Squared Error (SSE): 33681.48845233888

> Akaike Information Criteria (AIC): 7260.024233587071

> Baysian Information Criteria (BIC): 7321.371174574348

> Number of Empty Clusters: 0

> Number of Iterations: 14

==========

K = 5

> Mean Quantisation Error (MQE): 4.729867414480784

> Davies-Bouldin Index (DBI): 0.9107067610905825

> Sum of Squared Error (SSE): 28465.669085606416

> Akaike Information Criteria (AIC): 7321.602394052425

> Baysian Information Criteria (BIC): 7422.211377271558

> Number of Empty Clusters: 0

> Number of Iterations: 10

==========

K = 6

> Mean Quantisation Error (MQE): 4.350752361270133

> Davies-Bouldin Index (DBI): 0.8527472192290024

> Sum of Squared Error (SSE): 23565.448670146638

> Akaike Information Criteria (AIC): 7344.260317978129

> Baysian Information Criteria (BIC): 7493.946853987084

> Number of Empty Clusters: 0

> Number of Iterations: 15

==========

K = 7

> Mean Quantisation Error (MQE): 4.024251628291675

> Davies-Bouldin Index (DBI): 0.8425077723708884

> Sum of Squared Error (SSE): 20459.20773460612

> Akaike Information Criteria (AIC): 7373.412405914616

> Baysian Information Criteria (BIC): 7581.992005271357

> Number of Empty Clusters: 0

> Number of Iterations: 14

==========

K = 8

> Mean Quantisation Error (MQE): 3.8231297053494235

> Davies-Bouldin Index (DBI): 0.8751713164828259

> Sum of Squared Error (SSE): 18360.795956461054

> Akaike Information Criteria (AIC): 7413.542752676678

> Baysian Information Criteria (BIC): 7690.830925939169

> Number of Empty Clusters: 0

> Number of Iterations: 11

==========

K = 9

> Mean Quantisation Error (MQE): 3.5980655792403233

> Davies-Bouldin Index (DBI): 0.8406154731199219

> Sum of Squared Error (SSE): 16156.61090945364

> Akaike Information Criteria (AIC): 7446.983288607988

> Baysian Information Criteria (BIC): 7802.795546334192

> Number of Empty Clusters: 0

> Number of Iterations: 16

==========

K = 10

> Mean Quantisation Error (MQE): 3.5008275740587034

> Davies-Bouldin Index (DBI): 0.9058734601914781

> Sum of Squared Error (SSE): 15332.923163871917

> Akaike Information Criteria (AIC): 7526.349689765837

> Baysian Information Criteria (BIC): 7970.501542513721

> Number of Empty Clusters: 0

> Number of Iterations: 17

==========

K = 11

> Mean Quantisation Error (MQE): 3.4013789587210006

> Davies-Bouldin Index (DBI): 0.8904886751145227

> Sum of Squared Error (SSE): 14326.921947422312

> Akaike Information Criteria (AIC): 7560.378430775955

> Baysian Information Criteria (BIC): 8102.685389103482

> Number of Empty Clusters: 0

> Number of Iterations: 19

==========

K = 12

> Mean Quantisation Error (MQE): 3.2703717497113804

> Davies-Bouldin Index (DBI): 0.8905663282180888

> Sum of Squared Error (SSE): 13666.705581338269

> Akaike Information Criteria (AIC): 7654.201681189385

> Baysian Information Criteria (BIC): 8304.479255654518

> Number of Empty Clusters: 0

> Number of Iterations: 11

==========

K = 13

> Mean Quantisation Error (MQE): 3.1080047172735843

> Davies-Bouldin Index (DBI): 0.8441045443162138

> Sum of Squared Error (SSE): 11940.59076212564

> Akaike Information Criteria (AIC): 7643.176034514549

> Baysian Information Criteria (BIC): 8411.239735675254

> Number of Empty Clusters: 0

> Number of Iterations: 18

==========

K = 14

> Mean Quantisation Error (MQE): 2.9694363303649016

> Davies-Bouldin Index (DBI): 0.8736809773517756

> Sum of Squared Error (SSE): 11181.109003546413

> Akaike Information Criteria (AIC): 7724.190149952946

> Baysian Information Criteria (BIC): 8619.855488367186

> Number of Empty Clusters: 0

> Number of Iterations: 15

==========

K = 15

> Mean Quantisation Error (MQE): 2.8513342133754076

> Davies-Bouldin Index (DBI): 0.8440748287927557

> Sum of Squared Error (SSE): 10260.4739936369

> Akaike Information Criteria (AIC): 7756.972184522882

> Baysian Information Criteria (BIC): 8790.054670748623

> Number of Empty Clusters: 0

> Number of Iterations: 14

==========

K = 16

> Mean Quantisation Error (MQE): 2.7662519268186405

> Davies-Bouldin Index (DBI): 0.8483007955072985

> Sum of Squared Error (SSE): 9730.12144601235

> Akaike Information Criteria (AIC): 7813.876430424539

> Baysian Information Criteria (BIC): 8994.191575019742

> Number of Empty Clusters: 0

> Number of Iterations: 13

==========

K = 17

> Mean Quantisation Error (MQE): 2.7706567886418516

> Davies-Bouldin Index (DBI): 0.9160620289647771

> Sum of Squared Error (SSE): 9810.408805153756

> Akaike Information Criteria (AIC): 7908.50442286526

> Baysian Information Criteria (BIC): 9245.867736387892

> Number of Empty Clusters: 0

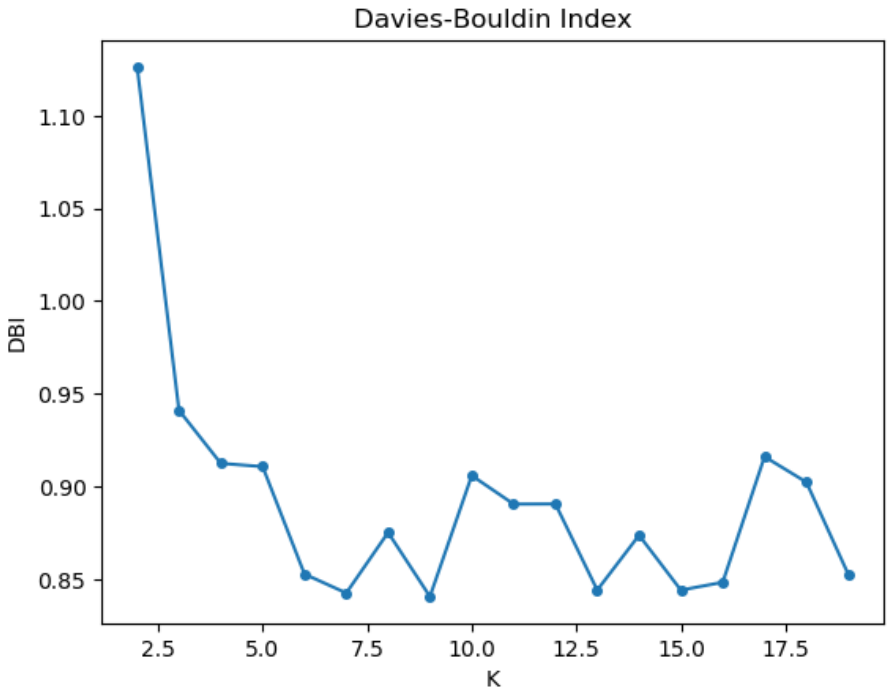
> Number of Iterations: 18

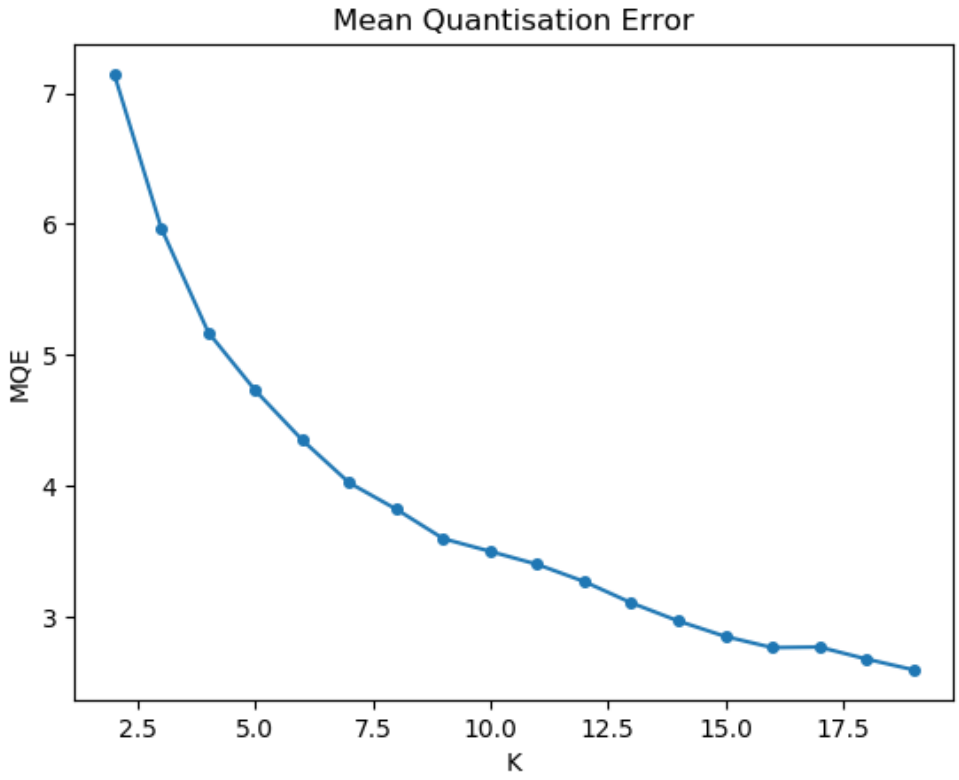
==========

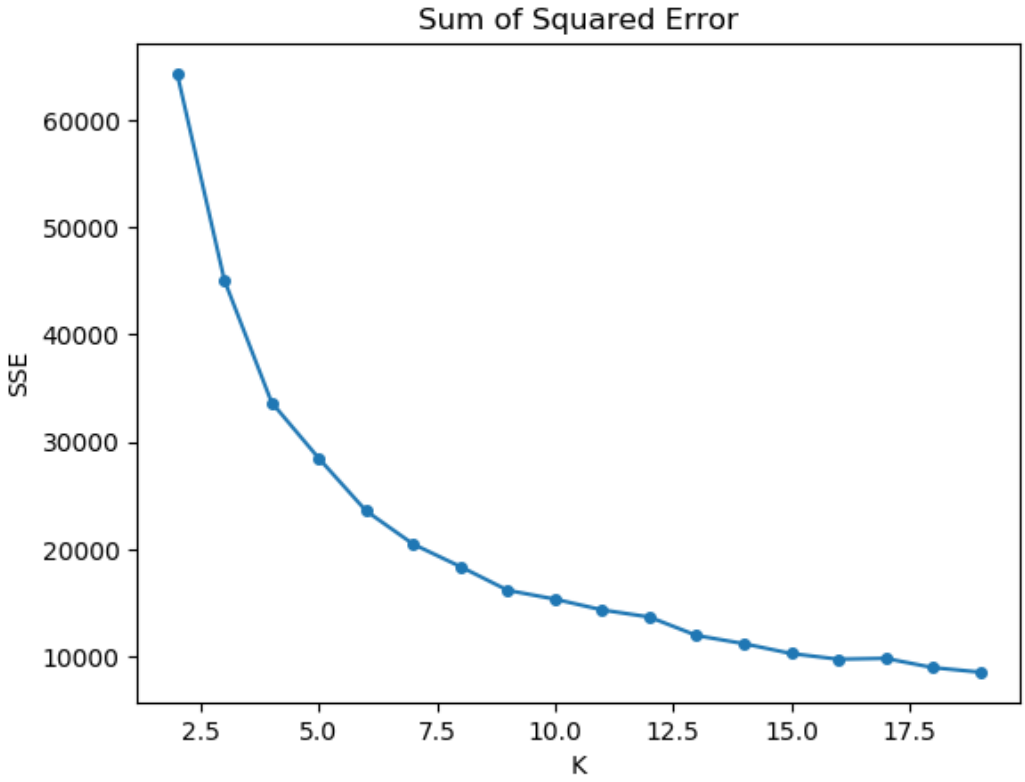
K = 18

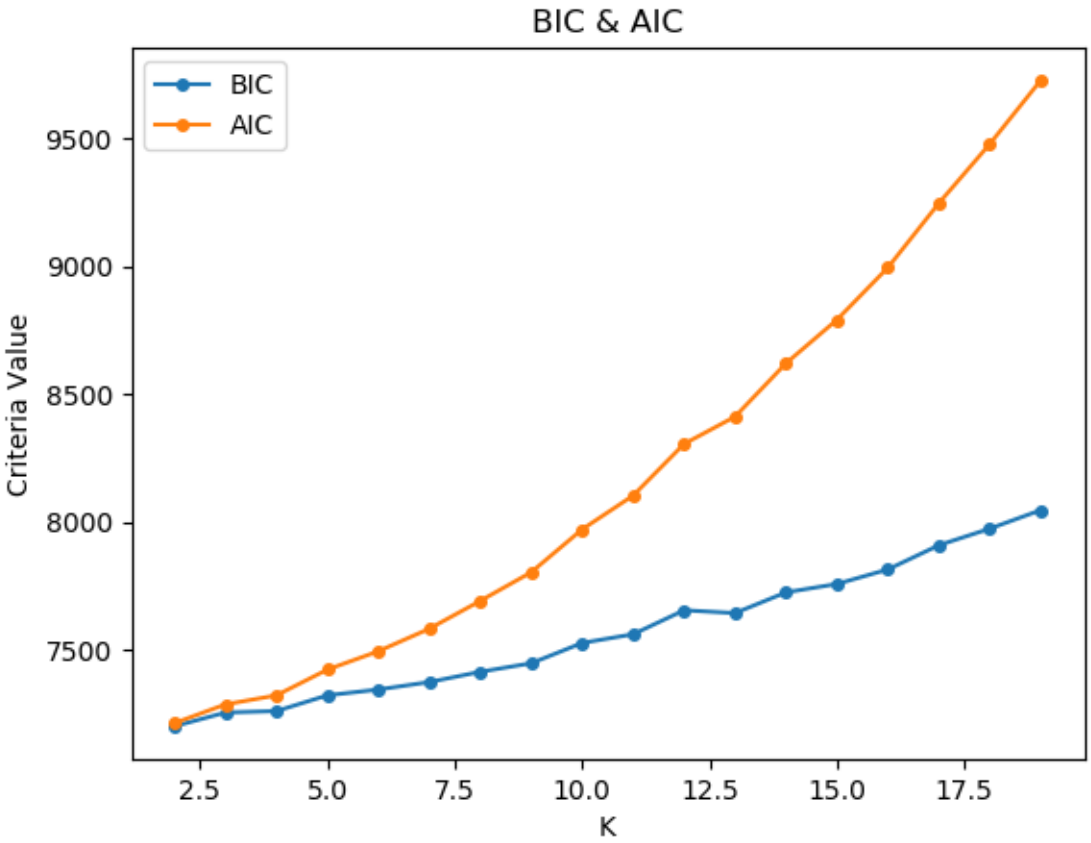
```
> Mean Quantisation Error (MQE): 2.6792390137141813
> Davies-Bouldin Index (DBI): 0.9023193040486779
> Sum of Squared Error (SSE): 8951.117640446231
> Akaike Information Criteria (AIC): 7973.378114544631
> Baysian Information Criteria (BIC): 9477.605107552656
> Number of Empty Clusters: 0
> Number of Iterations: 21
============================================================
## K = 19
> Mean Quantisation Error (MQE): 2.596879342229343
> Davies-Bouldin Index (DBI): 0.8527230488016612
> Sum of Squared Error (SSE): 8522.89983645021
> Akaike Information Criteria (AIC): 8045.726277886022
> Baysian Information Criteria (BIC): 9726.632460937404
> Number of Empty Clusters: 0
> Number of Iterations: 16
============================================================
```

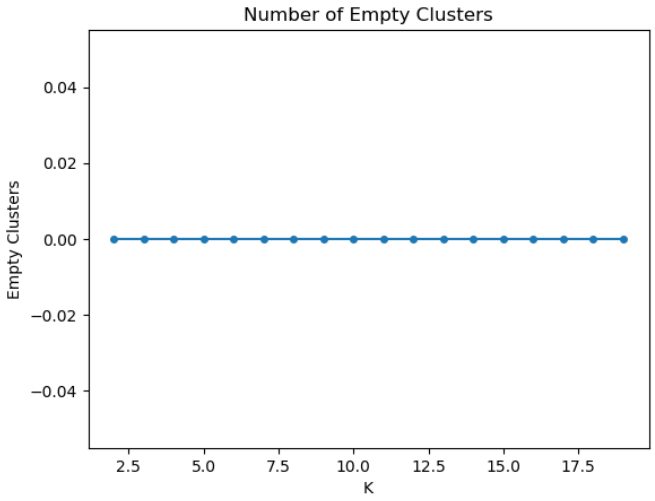
Basierend auf diesen Werten kann man anschließend das K wählen, dessen e.g SSE oder DBI am Niedrigsten ist, aber laut AIC und BIC noch keine zu starke Strafen aufgrund der Gefahr zum Overfitting erhält. Geplotted sehen diese Werte foldendermaßen aus.

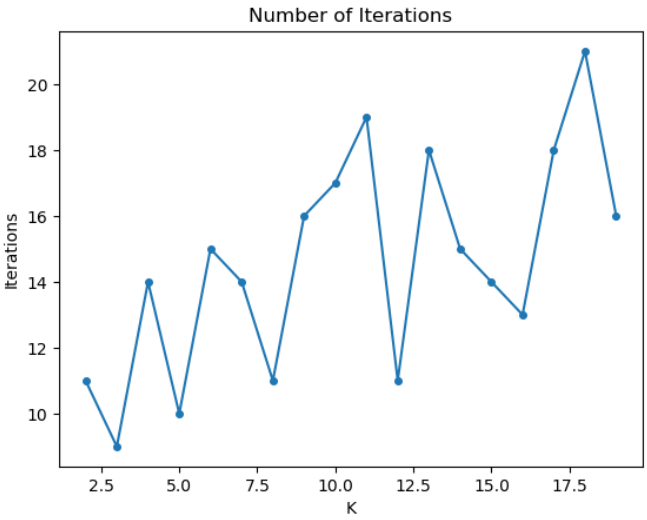



Mean Quantisation Error
MQE
K
7
6
5
4
3
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Sum of Squared Error
SSE
K
60000
50000
40000
30000
20000
10000
2.5
5.0
7.5
10.0
12.5
15.0
17.5







Um nun zumindest optisch einen Threshold zu wählen, ab wann ein K zu erwägen ist das "optimale" K zu sein, nutzt man die u.a die "Elbow Method". D.h man sucht in allen Qualitätsmaßen nach ellbogenartigen Knicks. Erfolg so ein Knick, kann man annehmen die darauffolgend auftretenden Werte werden schlecter abschneiden nach Qualität.

In unserem Beispiel wäre K = 7 laut DBI ein guter geeigneter Wert (laut DBI). Im DBI Verlauf sieht man auch einen auftauchenden Knick; in den anderen Qualitätsmaßen eher nicht.

4. Gaussian Mixture Models und Evolutionsstrategien

Implementieren Sie einen Algorithmus, der Gaussian Mixture Models für eine gegebene Menge von Datenpunkten optimiert.
Implementieren Sie dafür auch eine semi-automatische (d.h. auch auf User-Input reagierende) Optimierung der Anzahl der Cluster.
Verwenden Sie sinnvoll erstellte Test-Daten, um zu zeigen, wie eine optimale Anzahl der Cluster auf diese Weise gefunden werden kann.

Diese Aufgabe wurde in Python umgesetzt, da die mathematische Library Numpy sehr viele Vorteile bietet und leichter zu bedienen ist, als alles selbst zu schreiben.

Die Struktur der Aufgabe besteht aus den folgenden Files:

- main.py
- lib/__init__.py
- lib/es.py
- lib/gmm.py
- lib/utils.py

Kurze Erklärung der einzelnen Files:

- main.py: präsentiert das Hauptprogramm, welches normalverteilte Gaussche Mixture Models nach Input generiert, die Evolutionsstrategie parametrisiert und initiiert, diese startet und das Ergebnis auf der Konsole anzeigt oder plottet.
- lib/__init__.py: indiziert, dass dieses Verzeichnis ein Python Package ist
- lib/es.py: implementiert die eigentliche Evolutionsstrategie und enthält wichtige Klassendeklarationen
- lib/gmm.py: implementiert Gaussche Komponentenparameterlisten, welche als Lösungskandidaten in der ES gehandhabt werden.
- lib/utils.py: definiert Hilfsfunktionen (u.a die Softmax Funktion)

Das Programm lässt sich einfach über die Kommandozeile starten; sämtliche Parameterdefintion passiert hardkodiert in main.py. Dort finden einige Parameterdefinitionen statt:

- Datenparameter: wie die initialen Gausschen Mixture Models generiert werden sollen

- ES Parameter: die eigentlichen Parameter für die Evolutionsstrategie
- K Range: für welche K's die Optimierung mittels ES vorgenommen werden sollen.

Innerhalb der ES gibt es zwei Arten der Darstellung, welche beide für die Konsole und das resultierende Plotten gelten:

- Für eine K Range, die Darstellung der jeweils besten Lösungskandidaten in der Konsole und geplottet über die ausprobierten K's
- Für ein fixes K, die Darstellung der jeweils besten Lösungskandidaten jeder erstellten Generation, damit man den Konvergenzverlauf sehen kann. Dieser wird ebenso geplottet.

Der Output auf der Konsole für eine K Range sieht prinzipiell so aus. Alle Parameter für die Daten und die ES, sowie der K Range lassen sich aus dem Output auslesen:

```
============================================================
# Parameter Optimization of GMMs with Evolution Strategies
============================================================
[i] Testing Data
# Generated by multiple isophonic Gaussian Distributions
# > N (number of data points) = 500
# > D (dimensionality of data) = 2
# > C (number of components) = 6

# The centoids and corvariances of the components are drawn
# normally distributed
============================================================
[i] Evolution Strategy
EvolutionStrategy (
      μ = 1
      ρ = 1
      λ = 5
      s = 1/5
```

```
      type = '+'
      κ = inf
      σ = 0.05
      τ = 0.5
      max_iter = -1
      min_sigma = 0.01
)
============================================================
[i] Begin optimization
# K (number of components) in [2, 10]
============================================================
## K = 2
# Best solution candidate after 32 generations
Solution (
      σ = 0.00625
      κ = inf
      fitness = -4785.882156318536
      ParameterVector (
            GMM Component (
               ω: 0.6211157825953791
               μ: [ 1.34558626 -4.7075853 ]
               Σ: [[-1.4290385056190007, 0.42688555533760936],
                   [0.42688555533760936, 9.406173173850526]]
            )
            GMM Component (
               ω: 0.3788842174046208
               μ: [ 7.54095883  3.35365428]
               Σ: [[-0.47708806465057485, -10.071336334390205],
                   [-10.071336334390205, 8.084178752559268]]
            )
      )
)
```

```
============================================================
## K = 3
# Best solution candidate after 49 generations
Solution (
      σ = 0.00625
      κ = inf
      fitness = -3813.824726400538
      ParameterVector (
            GMM Component (
               ω: 0.3312480108435758
               μ: [ 5.33632423  4.30916261]
               Σ: [[14.892375612814234, -1.5215312913270798],
                   [-1.5215312913270798, 1.2114871936007212]]
            )
            GMM Component (
               ω: 0.32510082866986545
               μ: [ 2.35520378  2.23162468]
               Σ: [[7.715470100019112, -8.001058612018682],
                   [-8.001058612018682, 8.170032396096598]]
            )
            GMM Component (
               ω: 0.34365116048655875
               μ: [ 6.81690826 -8.84090711]
               Σ: [[5.570487293899351, -3.49024769570907],
                   [-3.49024769570907, 1.2969512390933382]]
            )
      )
)
============================================================
## K = 4
# Best solution candidate after 43 generations
```

```
Solution (
      σ = 0.00625
      κ = inf
      fitness = -2945.029258785456
      ParameterVector (
            GMM Component (
               ω: 0.24494193720372467
               μ: [-1.15739642 -0.8230085 ]
               Σ: [[1.0415814726088362, -3.134341638791424],
                   [-3.134341638791424, 2.778038225066258]]
            )
            GMM Component (
               ω: 0.2488852383944959
               μ: [  2.93444573 -11.6594681 ]
               Σ: [[5.031278621644076, -5.304683503147135],
                   [-5.304683503147135, 5.467679198960748]]
            )
            GMM Component (
               ω: 0.25356305283835345
               μ: [-1.66961401 -5.64933357]
               Σ: [[-0.30552502002311516, -0.5027982439530152],
                   [-0.5027982439530152, -0.6469495721024348]]
            )
            GMM Component (
               ω: 0.25260977156342607
               μ: [-2.06582561  5.44715669]
               Σ: [[3.927854304063108, -3.6711933232670497],
                   [-3.6711933232670497, 1.7734154747802053]]
            )
      )
)
```

```
============================================================
## K = 9
# Best solution candidate after 18 generations
Solution (
      σ = 0.00625
      κ = inf
      fitness = -2763.703132953732
      ParameterVector (
            GMM Component (
               ω: 0.11577138703320726
               μ: [-3.19001171  9.0593636 ]
               Σ: [[1.9218251636259158, -1.4483377820985448],
                   [-1.4483377820985448, 3.9561037959106584]]
            )
            GMM Component (
               ω: 0.11620442596037964
               μ: [-3.25499735 -3.77218468]
               Σ: [[2.2360598585187637, 1.1356892276048753],
                   [1.1356892276048753, 1.7346229714478871]]
            )
            GMM Component (
               ω: 0.09848627858959891
               μ: [-3.41102136  7.09579168]
               Σ: [[1.5229811530688906, -1.4320687281989002],
                   [-1.4320687281989002, 0.297711074252564]]
            )
            GMM Component (
               ω: 0.0962616383204768
               μ: [-3.05692635  5.48059974]
               Σ: [[5.5126909573045815, -0.4654895973456509],
                   [-0.4654895973456509, 3.343228346596009]]
```

```
)
GMM Component (
   ω: 0.10843977383388824
   μ: [-1.14992049 -6.5567907 ]
   Σ: [[3.5961488240574124, -2.095930163603188],
       [-2.095930163603188, 4.689089519949547]]
)
GMM Component (
   ω: 0.10518290662448487
   μ: [ 3.6814864  -1.99578534]
   Σ: [[2.9826671773602027, -2.4233838686161446],
       [-2.4233838686161446, 0.3471422202521679]]
)
GMM Component (
   ω: 0.11033190330681265
   μ: [-1.2728287   9.17047337]
   Σ: [[3.239019556340364, 2.1086854297462696],
       [2.1086854297462696, 2.0061087848999684]]
)
GMM Component (
   ω: 0.11063838790128122
   μ: [ 5.97078043  2.56438988]
   Σ: [[2.332296271568849, -1.8078615313081028],
       [-1.8078615313081028, 1.042979707009295]]
)
GMM Component (
   ω: 0.13868329842987034
   μ: [-2.37800906 -5.05350071]
   Σ: [[1.7722807598305608, -1.397813715872916],
       [-1.397813715872916, 0.6574240296664443]]
)
```

```
      )
)
============================================================
## K = 10
# Best solution candidate after 19 generations
Solution (
      σ = 0.00625
      κ = inf
      fitness = -2366.506285937375
      ParameterVector (
            GMM Component (
               ω: 0.10004348917182902
               μ: [-0.6747472  -2.61651177]
               Σ: [[1.6143907641921307, 0.6541370544511428],
                   [0.6541370544511428, 2.782626740670601]]
            )
            GMM Component (
               ω: 0.1006993511039656
               μ: [ 3.06197833  2.47048523]
               Σ: [[2.2821341191764435, -1.0249824492469788],
                   [-1.0249824492469788, 2.5766335053843266]]
            )
            GMM Component (
               ω: 0.0999092770536828
               μ: [-4.32034774  7.38734027]
               Σ: [[3.2635269876891937, -1.7539720774489445],
                   [-1.7539720774489445, 3.7693344410988496]]
            )
            GMM Component (
               ω: 0.09980042131504761
               μ: [ 4.53387285  2.45466851]
```

```
   Σ: [[1.6073367342759295, -0.597962213742815],
       [-0.597962213742815, 1.6501174875095488]]
)
GMM Component (
   ω: 0.09904682203063521
   μ: [-2.1416498  -4.47675557]
   Σ: [[1.6245937761351454, 0.012528282348920712],
       [0.012528282348920712, 4.506001301825079]]
)
GMM Component (
   ω: 0.1002186373401978
   μ: [ 4.12206364 -6.78262571]
   Σ: [[2.7960599850574654, -1.5692187038666237],
       [-1.5692187038666237, 1.226310703287934]]
)
GMM Component (
   ω: 0.09702772856827674
   μ: [-3.35220179  8.04515651]
   Σ: [[1.230171919523904, 2.2407754994332207],
       [2.2407754994332207, 3.8680724328127702]]
)
GMM Component (
   ω: 0.10191822748486813
   μ: [ 4.30672127 -5.18045859]
   Σ: [[2.9203856137757422, -0.29880307412404533],
       [-0.29880307412404533, 2.430983350715165]]
)
GMM Component (
   ω: 0.10100659981282535
   μ: [ 0.18578442 -7.50661353]
   Σ: [[2.0188358117034606, -0.2895699753432725],
       [-0.2895699753432725, -0.1172013514511352]]
```

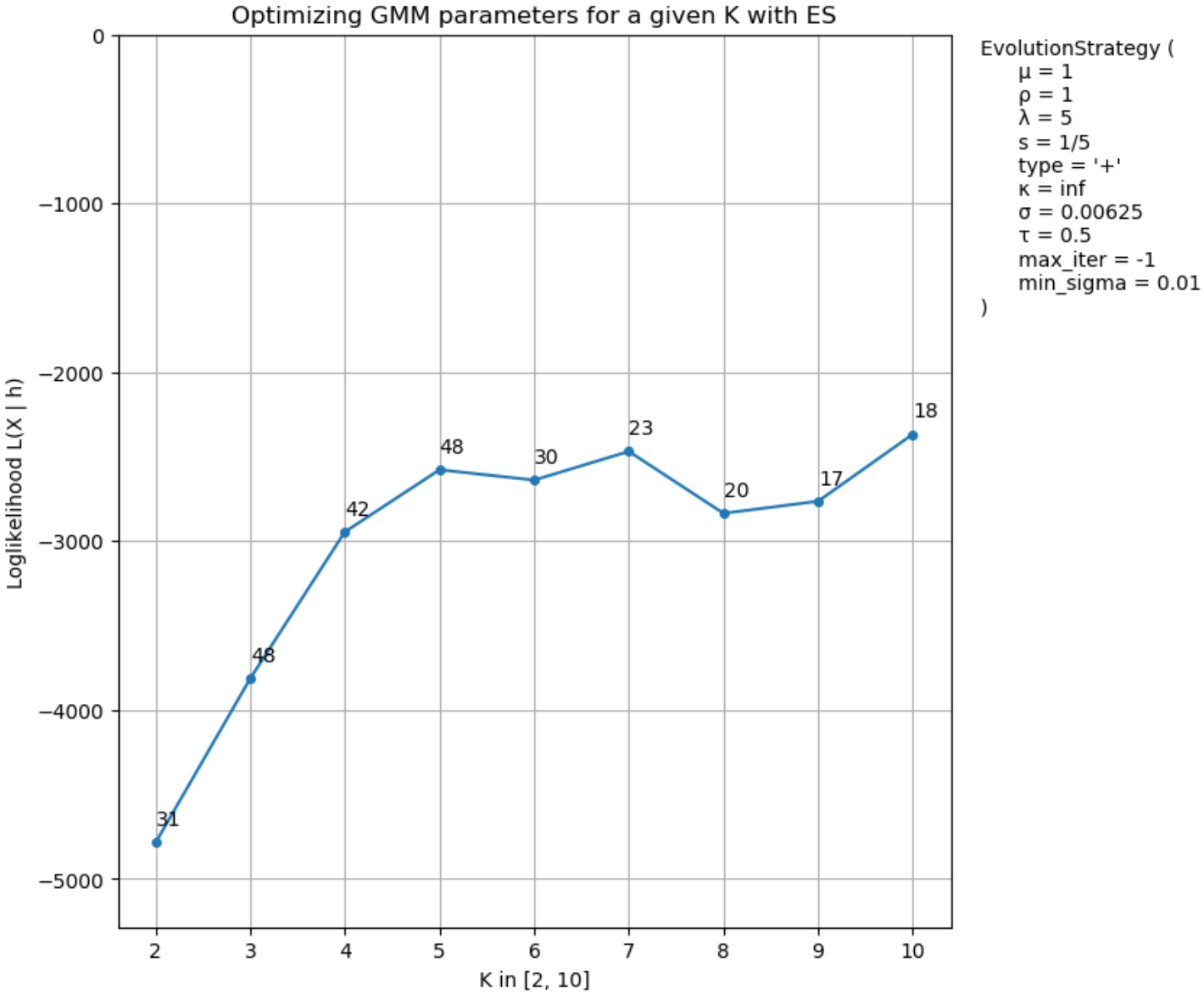
```
            )
            GMM Component (
               ω: 0.10032944611867159
               μ: [-1.16376133 -9.21376085]
               Σ: [[2.3704476339937854, 0.2997935377378405],
                   [0.2997935377378405, 1.8228949160165673]]
            )
      )
)
============================================================
[i] saved to 'optimization.png'
[i] thank you and goodnight
```

Aus Platzgrünen, wurden einige K's entfernt.

Man sieht sehr schön die jeweilige Optimierung für jedes einzelne K. Zudem wird für jedes K auch K GMMComponenten gewählt, welche jeweils die Parameter für ein Mixture Model enthalten. Wichtig zu beachten ist, dass die Gewichtungen in jedem ParameterVector über alle GMMComponenten summiert 1 ergibt und auf die Gleichhaltung der Kovarianzen geachtet wurde.

Auch ganz schön zu sehen ist die Parametrisierung der ES mitsamt all ihren Parametern. Alle Parameter und ihre Funktionsweise ist im Code ausreichend beschrieben.

Der Verlauf der besten Lösungskandidaten für jedes K wird auch geplottet.



Deutlich zu erkennen ist ein "Reverse Elbow", welche zwischen K4 und K deutliche sichtbar wird und anschließend wieder abschwächt.

Die Annotation zu jedem Punkt markieren die Anzahl der Generationen, die benötigt wurden um den besten Lösungskandidaten (aus mu Lösungskandidaten) zu erzeugen, bevor eines der drei Abbruchkriterien eingetreten ist.

Der Sigmawert - die Mutationschrittweite - unterscheidet sich hier im Vergleich zum Konsolenoutput. Das liegt daran, dass das Sigma nach jeder Generation laut der 1/5 Erfolgregel durch Tau verändert werden würde.

Die andere Darstellungsart und Weise nimmt ein fixes K und zeigt innerhalb der ES für dieses K, alle stattgefundenen Generationen und zeigt den Verlauf, bist die ES konvergiert.

Mit K = 4:

```
============================================================
# Parameter Optimization of GMMs with Evolution Strategies
============================================================
[i] Testing Data
# Generated by multiple isophonic Gaussian Distributions
# > N (number of data points) = 500
# > D (dimensionality of data) = 2
# > C (number of components) = 6

# The centoids and corvariances of the components are drawn
# normally distributed
============================================================
[i] Evolution Strategy
EvolutionStrategy (
      μ = 1
      ρ = 1
      λ = 5
      s = 1/5
      type = '+'
      κ = inf
      σ = 0.05
      τ = 0.5
      max_iter = -1
      min_sigma = 0.01
)
============================================================
[i] Begin optimization
# K (number of components) in [4, 4]
```

```
============================================================
## K = 4
~~ 1th Generation
# Best solution candidate
Solution (
      σ = 0.05
      κ = inf
      fitness = -5986.155451962684
      ParameterVector (
            GMM Component (
               ω: 0.25
               μ: [-1.16282408  7.9587817 ]
               Σ: [[6.259074115234618, 0.0], [0.0, 6.259074115234618]]
            )
            GMM Component (
               ω: 0.25
               μ: [ 1.24582694  5.44080964]
               Σ: [[6.259074115234618, 0.0], [0.0, 6.259074115234618]]
            )
            GMM Component (
               ω: 0.25
               μ: [-2.38744348 -7.87553122]
               Σ: [[6.259074115234618, 0.0], [0.0, 6.259074115234618]]
            )
            GMM Component (
               ω: 0.25
               μ: [-0.88169065 -8.54631196]
               Σ: [[6.259074115234618, 0.0], [0.0, 6.259074115234618]]
            )
      )
)
```

```
~~ 2th Generation
# Best solution candidate
Solution (
      σ = 0.1
      κ = inf
      fitness = -5974.403293801048
      ParameterVector (
            GMM Component (
               ω: 0.244217041909548
               μ: [-1.20206019  7.9706605 ]
               Σ: [[6.237761604182323, 0.09712243014927777],
                   [0.09712243014927777, 6.2457081175269655]]
            )
            GMM Component (
               ω: 0.26444300316542935
               μ: [ 1.27395902  5.44315088]
               Σ: [[6.152770345940712, 0.005544336247785261],
                   [0.005544336247785261, 6.330580414262184]]
            )
            GMM Component (
               ω: 0.2406278829511556
               μ: [-2.4128514  -7.90669752]
               Σ: [[6.145124319654897, -0.03764280199352146],
                   [-0.03764280199352146, 6.3958114412707285]]
            )
            GMM Component (
               ω: 0.250712071973867
               μ: [-0.85334887 -8.52509539]
               Σ: [[6.1646453908436225, -0.10625592159411552],
                   [-0.10625592159411552, 6.345953457942902]]
            )
      )
```

```
)
~~ 3th Generation
# Best solution candidate
Solution (
      σ = 0.1
      κ = inf
      fitness = -5973.993748557839
      ParameterVector (
            GMM Component (
               ω: 0.24764526775600187
               μ: [-1.17841831  7.93822717]
               Σ: [[6.251475846557746, 0.22030666459987477],
                   [0.22030666459987477, 6.234283092014077]]
            )
            GMM Component (
               ω: 0.2694186071719512
               μ: [ 1.25228385  5.50320861]
               Σ: [[6.215057540562849, 0.22861872900083977],
                   [0.22861872900083977, 6.270656859348257]]
            )
            GMM Component (
               ω: 0.24150995023078484
               μ: [-2.65659314 -7.68799049]
               Σ: [[6.145179016169171, -0.0653237724764592],
                   [-0.0653237724764592, 6.336172258597147]]
            )
            GMM Component (
               ω: 0.24142617484126208
               μ: [-0.74512562 -8.54806769]
               Σ: [[6.124065362068676, -0.24497821894784305],
                   [-0.24497821894784305, 6.434449914202156]]
```

```
            )
      )
)
~~ 4th Generation
# Best solution candidate
Solution (
      σ = 0.1
      κ = inf
      fitness = -5970.348312620551
      ParameterVector (
            GMM Component (
               ω: 0.24971772776872858
               μ: [-1.17091358  7.88436065]
               Σ: [[6.250505576274411, 0.24070076424024317],
                   [0.24070076424024317, 6.152650844727919]]
            )
            GMM Component (
               ω: 0.25789394448471303
               μ: [ 1.3462511   5.57872481]
               Σ: [[6.230280847688226, 0.32676796538742076],
                   [0.32676796538742076, 6.255534692700113]]
            )
            GMM Component (
               ω: 0.21746911458340515
               μ: [-2.74125198 -7.72880874]
               Σ: [[6.079498522286886, 0.07566274432435897],
                   [0.07566274432435897, 6.405280500392675]]
            )
            GMM Component (
               ω: 0.2749192131631532
               μ: [-0.7061199  -8.45771888]
```

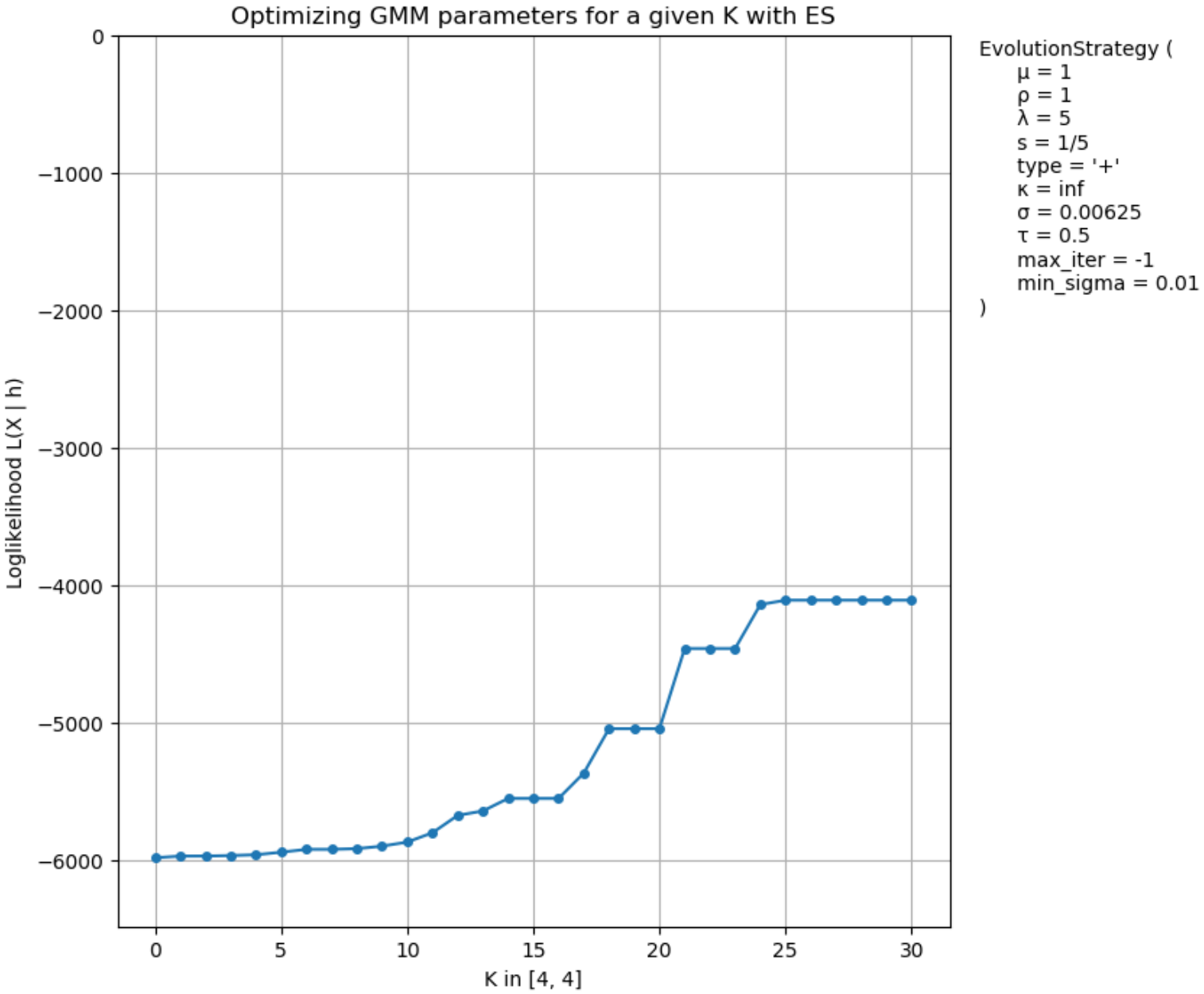
```
            Σ: [[6.23900797087438, -0.15394753893145305],
                [-0.15394753893145305, 6.497773934385862]]
         )
      )
)
~~ 30th Generation
# Best solution candidate
Solution (
      σ = 0.025
      κ = inf
      fitness = -4112.0642849435935
      ParameterVector (
            GMM Component (
               ω: 0.11904934469374502
               μ: [-7.17220649  7.49729301]
               Σ: [[8.471762797974836, 0.3999781490259392],
                   [0.3999781490259392, 6.930620852749877]]
            )
            GMM Component (
               ω: 0.3941138202056439
               μ: [ 2.33099571  6.68421375]
               Σ: [[0.3276605249222853, 0.7146396017734399],
                   [0.7146396017734399, 0.6573880971871477]]
            )
            GMM Component (
               ω: 0.16656058949404437
               μ: [ -7.49903377 -11.525351  ]
               Σ: [[5.440096418755191, 2.2131767460408303],
                   [2.2131767460408303, 6.720564816369033]]
            )
            GMM Component (
```

```
            ω: 0.32027624560656665
            μ: [-1.75908842 -8.77538443]
            Σ: [[6.232442353979597, 1.1057546654350152],
                [1.1057546654350152, 11.990925570363984]]
         )
   )
)
~~ 31th Generation
# Best solution candidate
Solution (
   σ = 0.0125
   κ = inf
   fitness = -4112.0642849435935
   ParameterVector (
      GMM Component (
         ω: 0.11904934469374502
         μ: [-7.17220649  7.49729301]
         Σ: [[8.471762797974836, 0.3999781490259392],
             [0.3999781490259392, 6.930620852749877]]
      )
      GMM Component (
         ω: 0.3941138202056439
         μ: [ 2.33099571  6.68421375]
         Σ: [[0.3276605249222853, 0.7146396017734399],
             [0.7146396017734399, 0.6573880971871477]]
      )
      GMM Component (
         ω: 0.16656058949404437
         μ: [ -7.49903377 -11.525351  ]
         Σ: [[5.440096418755191, 2.2131767460408303],
             [2.2131767460408303, 6.720564816369033]]
```

```
            )
            GMM Component (
               ω: 0.32027624560656665
               μ: [-1.75908842 -8.77538443]
               Σ: [[6.232442353979597, 1.1057546654350152],
                  [1.1057546654350152, 11.990925570363984]]
            )
      )
)
```

Aus Platzgründen, wurden nicht alle Generationen angezeigt.

31 Generationen ist bei weitem keine leichte Angelegenheit. Man kann sich die Fitnesswerte - die Loglikelihoods geplotted darstellen lassen.



Die X - Achse beschreibt hier nicht die Range [4, 4], sondern die in der ES erstellten Generationen

Man sieht hier auch einen fastschon ellbogenförmigen Verlauf, bis diser ab 25 zu konvergieren beginnt und tatsächlich bei 31 ein lokales Optimum erreicht hat.

Zum Abschluss noch ein Screenshot der Daten, wie aus den Gausschen Mixture Models generiert werden.

Mit C (Componenten) = 6.