

目次

第 1 章	序論	1
1	背景	1
2	目的	2
3	構成	2
第 2 章	提案手法	3
1	はじめに	3
2	定義	3
3	sFCMA	4
4	eFCMA	4
5	qFCMA	5
6	おわりに	5
第 3 章	人工データによる実験	6
1	はじめに	6
2	人工データについて	6
3	アルゴリズム	6
4	分類関数による特性比較	6
5	おわりに	7
第 4 章	実データによる実験	10
1	はじめに	10
2	実データについて	10
3	アルゴリズム	10
4	ARI による精度比較	10
5	おわりに	11

第 5 章 結論	12
参考文献	13
付録 A プログラムソース	14
vector.h	14
vector.cxx	15
matrix.h	21
matrix.cxx	22
randGaussianMain.cxx	30
hcm.cxx	31
hcm.h	36
sfcma.h	37
sfcma.cxx	38
efcm.h	39
efcm.cxx	40
qfcm.h	41
qfcm.cxx	42
sfcma.h	43
sfcma.cxx	43
efcma.h	44
efcma.cxx	45
qfcma.h	46
qfcma.cxx	46
sfcma_main_2d-Gaussian-2clusters.cxx	48
efcma_main_2d-Gaussian-2clusters.cxx	52
qfcma_main_2d-Gaussian-2clusters.cxx	57
sfcma_main_2d-Gaussian-2clusters.cxx	62
efcma_main_2d-Gaussian-2clusters.cxx	66
qfcma_main_2d-Gaussian-2clusters.cxx	71
sfcma_main_user_knowledge.cxx	76
efcma_main_user_knowledge.cxx	76
qfcma_main_user_knowledge.cxx	76
sfcma_main_user_knowledge.cxx	76

efcma_main_user_knowledge.cxx	79
qfcma_main_user_knowledge.cxx	81

目次

1.1	クラスタリングについて	2
3.1	人工データ	8
3.2	sFCMA の人工データの実験結果	8
3.3	eFCMA の人工データの実験結果	9
3.4	qFCMA の人工データの実験結果	9
4.1	sFCMA の実データの実験結果	11
4.2	eFCMA の実データの実験結果	11
4.3	qFCMA の実データの実験結果	11

表目次

2.1	ファジイクラスタリングの最適化問題における定義	3
4.1	各手法の ARI の最高値とパラメータ	11

第 1 章

序論

第 1 節 背景

近年，情報通信社会の発展に伴いデータ量が増大し，日々多様なデータがコンピュータに蓄積されている．検索エンジンなどのインターネット上のサービスでは，蓄積されたビッグデータの解析や分類を行うことで，利用者に適切な情報を素早く送ることを可能にしている．ビッグデータを人の手によって分類することには困難が伴うため，計算機を用いて自動的にデータの分類を行うための技術であるクラスタリングが必要となる．クラスタリングとは，与えられたデータの個体間に存在する類似性に基づいて，個体をいくつかのクラスタと呼ばれるグループに分割を行う教師なし機械学習の手法である．図 1.1 に示すように，ある一定のルールに従って存在するデータがあったときに，クラスタリングを行うことでそれぞれのデータがどのクラスタに属するかということを示すことができる．

データをクラスタに分類した際に，それぞれのデータが各クラスタに属す度合いを表した値を帰属度と呼ぶ．帰属度が 0 と 1 のみで表され，それぞれのデータが各クラスタに明確に分類されるクラスタリングをハードクラスタリングと呼び，一方で帰属度が 0 と 1 の間の値で表され，データが属するクラスタを柔軟に表すことができるクラスタリングをファジィクラスタリングと呼ぶ．現実存在しているデータには，明確に分類できるものだけでなく本質的に分類できない複雑なものも存在し，そういったデータの分類にはファジィクラスタリングが有効である．

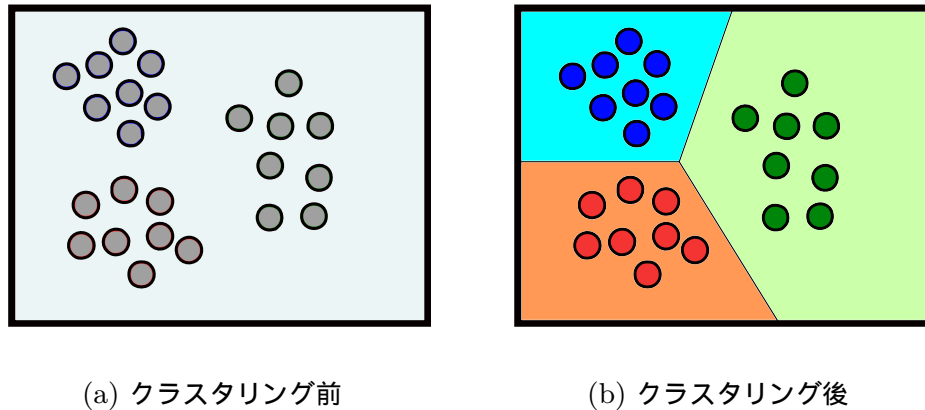


図 1.1: クラスタリングについて

第 2 節 目的

既存の手法における課題として、各クラスタのサイズに差がある場合、クラスタリングから有意な結果が得られないというものがある。ここで、クラスタのサイズとは、クラスタに属するデータの数と、そのクラスタに属するデータ間の類似度に基づくものであり、データの数が多し、または類似度が小さいクラスタをサイズが大きいクラスタとし、データの数が少ない、または類似度が大きいクラスタをサイズが大きいクラスタとする。現在、各クラスタのサイズを考慮してクラスタリングを行う手法が複数提案されており、本研究は其中でも、Standard Fuzzy c -Means with vAriable controlling cluster size (sFCMA) [1], Entropy-regularized Fuzzy c -Means vAriable controlling clusters size (eFCMA) [2], q -divergence based Fuzzy c -Means with vAriable controlling cluster size (qFCMA) [3] の 3 手法について、各手法の特性を把握するとともに、最も有効な手法を発見することを目的とする。

第 3 節 構成

本文書の構成を次に示す。第 2 章では、提案手法について説明する。第 3 章では、人工データ実験による各手法の特性比較を行う。第 4 章では、実データ実験による各手法の精度比較を行う。最後に第 5 章では、本文書の結論を述べる。また、付録では、プログラムソースを掲載している。

第 2 章 提案手法

第 1 節 はじめに

本章では，本研究で提案するファジィクラスタリング手法について説明する．まず第 2 節で定義を示し，次に第 4 節から第 5 節で各手法の最適化問題と，各変数の更新式について述べる．

第 2 節 定義

次節で述べるファジィクラスタリングの最適化問題における各変数の定義について，表 2.1 に示す．

表 2.1: ファジィクラスタリングの最適化問題における定義

N	データ数	x_k	データ
C	クラスタ数	v_i	クラスタ中心
λ, m	ファジィ化パラメータ	$u_{i,k}$	帰属度
α_i	クラスタサイズ調整変数		

第 3 節 sFCMA

Standard Fuzzy c -Means with vAriable controlling cluster size (sFCMA) [1] の最適化問題を以下に示す .

$$\underset{u,v,\alpha}{\text{minimize}} \sum_{i=1}^C \sum_{k=1}^N (\alpha_i)^{1-m} (u_{i,k})^m \|x_k - v_i\|_2^2 \quad (2.1)$$

$$\text{subject to } \sum_{i=1}^C u_{i,k} = 1, \sum_{i=1}^C \alpha_i = 1 \text{ and } m > 1, \alpha_i > 0 \quad (2.2)$$

次に , クラスタ中心 v_i の更新式を以下に示す .

$$v_i = \frac{\sum_{k=1}^N (u_{i,k})^m x_k}{\sum_{k=1}^N (u_{i,k})^m} \quad (2.3)$$

次に , 帰属度 $u_{i,k}$ の更新式を以下に示す .

$$u_{i,k} = \frac{1}{\sum_{j=1}^C \frac{\alpha_j}{\alpha_i} \left(\frac{d_{j,k}}{d_{i,k}} \right)^{\frac{1}{1-m}}} \quad (2.4)$$

次に , クラスタサイズ調整変数 α_i の更新式を以下に示す .

$$\alpha_i = \frac{1}{\sum_{j=1}^C \left(\sum_{k=1}^N \frac{(u_{j,k})^m d_{j,k}}{(u_{i,k})^m d_{i,k}} \right)^{\frac{1}{m}}} \quad (2.5)$$

第 4 節 eFCMA

Entropy-regularized Fuzzy c -Means vAriable controlling clusters size (eFCMA) [2] の最適化問題を以下に示す .

$$\underset{u,v,\alpha}{\text{minimize}} \sum_{i=1}^C \sum_{k=1}^N u_{i,k} \|x_k - v_i\|_2^2 + \lambda^{-1} \sum_{i=1}^C \sum_{k=1}^N u_{i,k} \log \left(\frac{u_{i,k}}{\alpha_i} \right) \quad (2.6)$$

$$\text{subject to } \sum_{i=1}^C u_{i,k} = 1, \sum_{i=1}^C \alpha_i = 1 \text{ and } \lambda > 0, \alpha_i > 0 \quad (2.7)$$

次に , クラスタ中心 v_i の更新式を以下に示す .

$$v_i = \frac{\sum_{k=1}^N u_{i,k} x_k}{\sum_{k=1}^N u_{i,k}} \quad (2.8)$$

次に，帰属度 $u_{i,k}$ の更新式を以下に示す．

$$u_{i,k} = \frac{\pi_i \exp(-\lambda \|x_k - v_i\|_2^2)}{\sum_{j=1}^C \alpha_j \exp(-\lambda \|x_k - v_j\|_2^2)} \quad (2.9)$$

次に，クラスサイズ調整変数 α_i の更新式を以下に示す．

$$\alpha_i = \frac{\sum_{k=1}^N u_{i,k}}{N} \quad (2.10)$$

第 5 節 qFCMA

q -divergence based Fuzzy c -Means with vAriable controlling cluster size (qFCMA) [3] の最適化問題を以下に示す．

$$\underset{u,v,\alpha}{\text{minimize}} \sum_{i=1}^C \sum_{k=1}^N (\alpha_i)^{1-m} (u_{i,k})^m \|x_k - v_i\|_2^2 + \frac{\lambda^{-1}}{m-1} \sum_{i=1}^C \sum_{k=1}^N (\alpha_i)^{1-m} (u_{i,k})^m \quad (2.11)$$

$$\text{subject to } \sum_{i=1}^C u_{i,k} = 1, \sum_{i=1}^C \alpha_i = 1 \text{ and } \lambda > 0, m > 1, \alpha_i > 0 \quad (2.12)$$

次に，クラスタ中心 v_i の更新式を以下に示す．

$$v_i = \frac{\sum_{k=1}^N (u_{i,k})^m x_k}{\sum_{k=1}^N (u_{i,k})^m} \quad (2.13)$$

次に，帰属度 $u_{i,k}$ の更新式を以下に示す．

$$u_{i,k} = \frac{\alpha_i (1 + \lambda(1-m) \|x_i - v_k\|_2^2)^{\frac{1}{1-m}}}{\sum_{j=1}^C \alpha_j (1 + \lambda(1-m) \|x_j - v_k\|_2^2)^{\frac{1}{1-m}}} \quad (2.14)$$

次に，クラスサイズ調整変数 α_i の更新式を以下に示す．

$$\alpha_i = \frac{1}{\sum_{j=1}^C \left(\sum_{k=1}^N \frac{(u_{j,k})^m (1 - \lambda(1-m) d_{j,k})}{(u_{i,k})^m (1 - \lambda(1-m) d_{i,k})} \right)^{\frac{1}{m}}} \quad (2.15)$$

第 6 節 おわりに

本章では，本研究で提案するファジィクラスタリング手法について説明した．まず第 2 節で定義を示し，次に第 4 節から 5 節で各手法の最適化問題と，各変数の更新式について述べた．

第 3 章

人工データによる実験

第 1 節 はじめに

本章では，人工データを用いた実験について述べる．まず第 2 節で本実験で用いる人工データについて説明する．次に第 3 節でアルゴリズムについて述べる．最後に第 4 節で実験により得られた分類関数を用いて各手法の特性比較を行う．

第 2 節 人工データについて

人工データとして，クラス数 2，各クラスのデータ数 50，合計データ数 100 のデータを平均値 $(-1, -1)$ ，標準偏差 $(0.5, 0.5)$ 及び平均値 $(1, 1)$ ，標準偏差 $(0.5, 0.5)$ のガウスサンプリングで生成したデータを用いた (図 3.1)．

第 3 節 アルゴリズム

1. クラスタ中心 v をランダムに与え，クラスタサイズ調整変数 α をクラスタ数の逆数で初期化する．
2. クラスタ中心 v を用いて帰属度 u を更新する．
3. 帰属度 u を用いてクラスタ中心 v 及びクラスタサイズ調整変数 α を更新する．
4. u, v, α の変化の合計が 10^{-10} 未満に収束すれば終了し，そうでない場合は 2 に戻る．

第 4 節 分類関数による特性比較

分類関数は，各クラスタに対する帰属度を座標空間上に可視化したもので，分類関数を見ることにより，データがどのクラスタに属するかということが調べることができるとともに，各手法がファジィであるかクリスプであるかを判別することができる．

sFCMA の実験結果を図 3.2a, 3.2b に示す．パラメータ m を 2.00 から 1.01 に変化させた

ところ，分類関数は m の値が大きいほどファジィになり，小さいほどクリスプになることが分かった．

次に，eFCMA の実験結果を図 3.3a, 3.3b に示す．垂直軸は分類関数値を，底面はデータ空間を表す．網掛けで示されるのが分類関数であり，各点がデータを表している．パラメータ λ を 1 から 10 に変化させたところ，分類関数は λ の値が小さいほどファジィになり，大きいほどクリスプになることが分かった．

qFCMA の実験結果を図 3.4a, 3.4b, 3.4c に示す．こちらは，パラメータ (m, λ) の組み合わせとして， $(2.00, 1)$, $(1.01, 1)$, $(1.01, 10)$ の 3 通りでクラスタリングを行った．図 3.4a 及び図 3.4b の分類関数より， m の値が大きいほどファジィになり，小さいほどクリスプになることが分かった．また，図 3.4b 及び図 3.4c の分類関数より， λ の値が小さいほどファジィになり，大きいほどクリスプになることが分かった．そして，図 3.2 及び図 3.4a, 3.4b の分類関数より qFCMA において $m - 1 \rightarrow +0$ とすると sFCMA と同じ特性が得られ，図 3.3 及び図 3.4b, 3.4c より， $\lambda \rightarrow \infty$ とすると eFCMA と同様の特性を示すことがわかった．これらの実験結果より qFCMA は sFCMA と eFCMA の特性を併せ持つと言える．

第 5 節 おわりに

本章では，人工データを用いた実験について述べた．まず第 2 節で本実験で用いる人工データについて説明した．次に第 3 節でアルゴリズムについて述べた．最後に第 4 節で実験により得られた分類関数を用いて各手法の特性比較を行った．

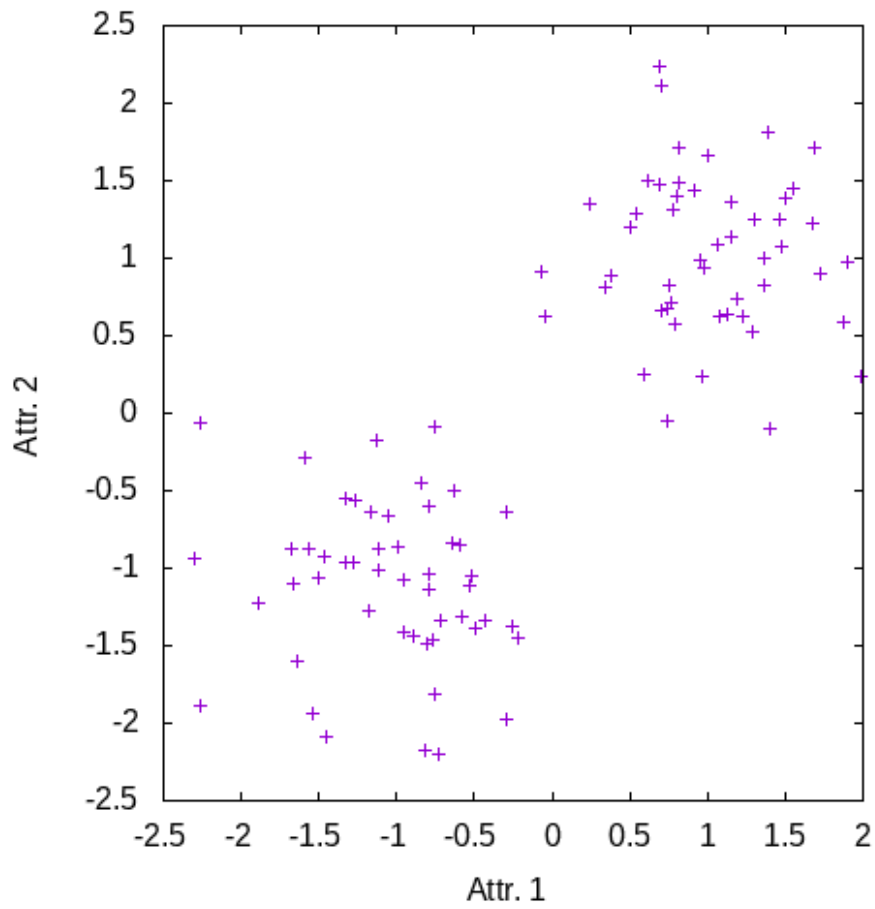


図 3.1: 人工データ

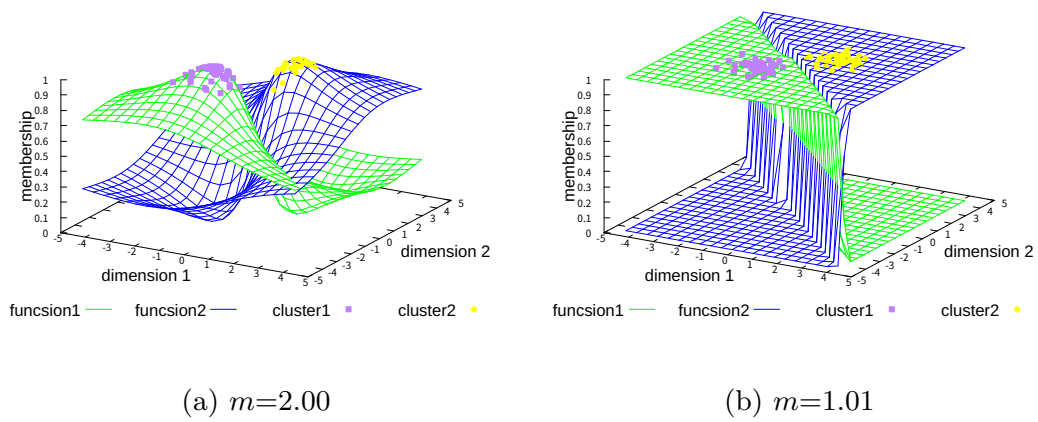


図 3.2: sFCMA の人工データの実験結果

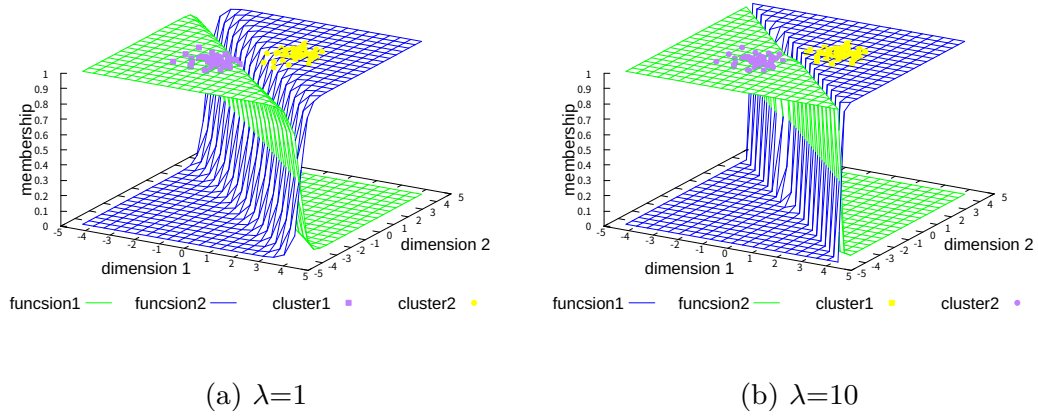


図 3.3: eFCMA の人工データの実験結果

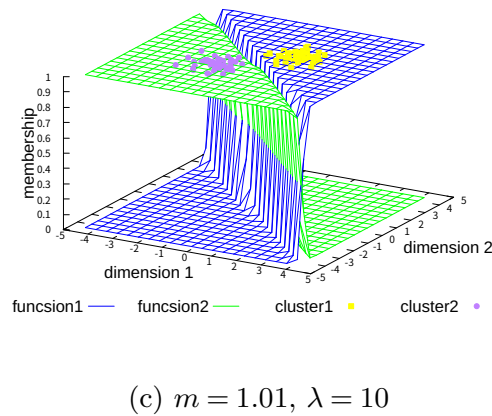
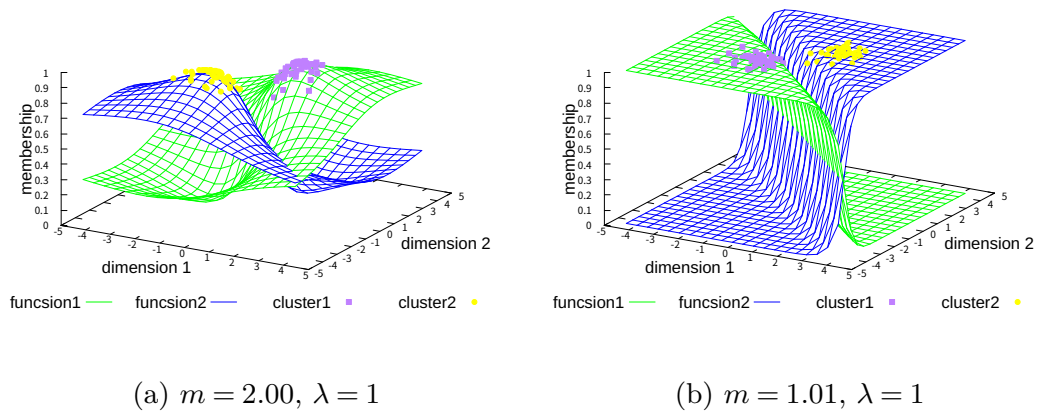


図 3.4: qFCMA の人工データの実験結果

第 4 章

実データによる実験

第 1 節 はじめに

本章では、実データを用いた実験について述べる。まず第 2 節で本実験で用いる実データについて説明する。次に第 3 節でアルゴリズムについて述べる。最後に第 4 節で実験により得られた評価指標を用いて各手法の精度比較を行う。

第 2 節 実データについて

実データとしては、個体数 403、クラス数 4 の、被験者の勉強時間や試験結果などの 5 属性を収録した “User Knowledge Modeling Dasta Set” を用いた。

第 3 節 アルゴリズム

1. 帰属度 u を正解帰属度で初期化し、クラスタサイズ調整変数 α をクラスタ数の逆数で初期化する。
2. 帰属度 u を用いてクラスタ中心 v 及びクラスタサイズ調整変数 α を更新する。
3. u, v, α の変化の合計が 10^{-10} 未満に収束すれば終了し、そうでない場合は 2 に戻る。

第 4 節 ARI による精度比較

sFCMA, eFCMA, qFCMA の実データ実験の結果について、それぞれ図 4.1, 4.2, 4.3 に示す。sFCMA では m の値を 1.1 から 3.0 まで 0.1 刻み、eFCMA では λ の値を 1 から 100 まで 1 刻み、qFCMA では m の値を 1.1 から 3.0 まで 0.1 刻み、 λ の値を 1 から 100 まで 1 刻みで変化させた。

それぞれの手法の最高 ARI を表 4.1 に示す。最も高い ARI を示した手法は sFCMA であり、他の 2 手法と比較して ARI に 0.4 以上の差が見られた。

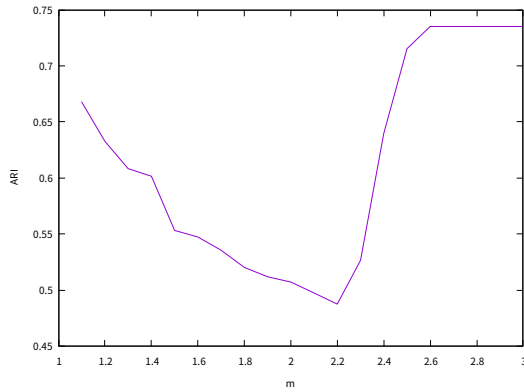


図 4.1: sFCMA の実データの実験結果

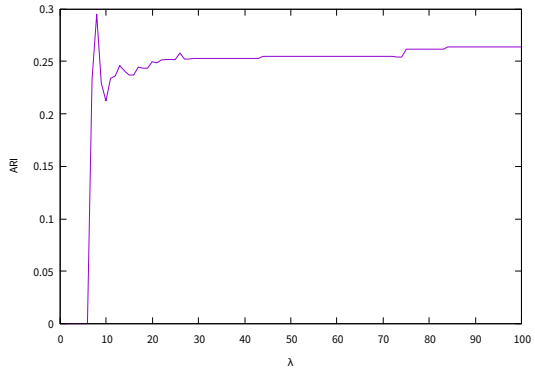


図 4.2: eFCMA の実データの実験結果

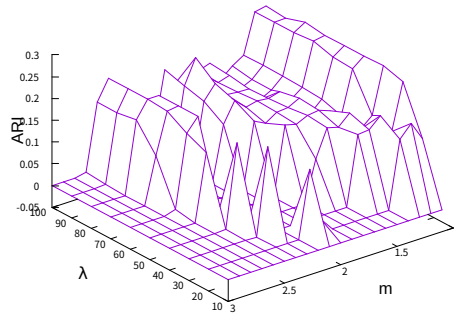


図 4.3: qFCMA の実データの実験結果

表 4.1: 各手法の ARI の最高値とパラメータ

手法名	ARI の最高値	パラメータ値
sFCMA	0.73515	$m = 3$
eFCMA	0.29500	$\lambda = 8$
qFCMA	0.26286	$\lambda = 80, m = 1.1$

第 5 節 おわりに

本章では，実データを用いた実験について述べた．まず第 2 節で本実験で用いる人工データについて説明した．次に第 3 節でアルゴリズムについて述べた．最後に第 4 節で実験により得られた評価指標を用いて各手法の精度比較を行った．

第 5 章

結論

本文書では、第 2 章では、提案手法について説明した。第 3 章では、人工データ実験により各手法の特性比較を行った。第 4 章では、実データ実験により各手法の精度比較を行った。最後に第 5 章では、本文書の結論を述べた。また、付録では、プログラムソースを掲載した。

本研究では、既に提案されていた 3 種のクラスタリング手法の特性と精度について現在に至るまで明らかになっていなかったため、人工データを用いた特性比較及び実データを用いた精度比較を行った。その結果として、sFCMA は m が大きくなるとファジィになり、eFCMA は λ が大きくなるほどクリスプになることが分かった。また、qFCMA は sFCMA と eFCMA の両方の特性を併せ持つということが分かった。精度は sFCMA が最も高評価となった。要因として、この手法の最適化問題にエントロピー項が含まれないということが考えられる。sFCMA の精度には、エントロピー項が含まれる eFCMA, qFCMA の 2 手法と比較して大きな差が見られた。今後の課題は、今回用いなかった他の実データで 3 手法の比較を行い、精度についての裏付けを行うことである。

参考文献

- [1] Miyamoto, S., Kurosawa, N.: “Controlling Cluster Volume Sizes in Fuzzy c -means Clustering”, Proc. SCIS&ISIS2004, pp. 1–4, (2004).
- [2] Ichihashi, H., Honda, K., Tani, N.: “Gaussian Mixture PDF Approximation and Fuzzy c -means Clustering with Entropy Regularization”, Proc. 4th Asian Fuzzy System Symposium, pp. 217–221, (2000).
- [3] Miyamoto, S., Ichihashi, H., and Honda, K.: Algorithms for Fuzzy Clustering, Springer (2008).
- [4] 宮本 定明, 馬屋原 一孝, 向殿 政男: “ファジィ c -平均法とエントロピー正則化法におけるファジィ分類関数”, 日本ファジィ学会誌 Vol. 10, No. 3 pp. 548–557, (1998).
- [5] Hubert, L., and Arabie, P.: “Comparing Partitions”, Journal of Classification, Vol. 2, No. 1, pp. 193–218, (1985).

付録 A

プログラムソース

vector.h

```
1  #include<iostream>
2  #include<cstring>
3
4  #ifndef __VECTOR__
5  #define __VECTOR__
6
7  class Matrix;
8
9
10 class Vector{
11     private:
12         int Size;
13         double *Element;
14     public:
15         Vector(int size=0);
16         ~Vector(void);
17         explicit Vector(int dim, double arg, const char *s);
18         Vector(const Vector &arg);
19         Vector &operator=(const Vector &arg);
20         Vector(Vector &&arg);
21         Vector &operator=(Vector &&arg);
22         int size(void) const;
23         double operator[](int index) const;
24         double &operator[](int index);
25         Vector operator+(void) const;
26         Vector operator-(void) const;
27         Vector &operator+=(const Vector &rhs);
28         Vector &operator-=(const Vector &rhs);
29         Vector &operator*=(double rhs);
30         Vector &operator/=(double rhs);
31         Vector operator+(const Vector &rhs) const;
32         Vector operator-(const Vector &rhs) const;
```

```
33     double operator*(const Vector &rhs) const;
34     bool operator==(const Vector &rhs) const;
35     bool operator!=(const Vector &rhs) const;
36     Vector sub(int begin, int end) const;
37     void set_sub(int begin, int end, const Vector &arg);
38 };
39
40 Vector operator*(double lhs, const Vector &rhs);
41 Vector operator/(const Vector &lhs, double rhs);
42 std::ostream &operator<<(std::ostream &os, const Vector &rhs);
43 double max_norm(const Vector &arg);
44 double squared_norm(const Vector &arg);
45 double norm_square(const Vector &arg);
46 double L1norm_square(const Vector &arg);
47 Vector fraction(const Vector &arg);
48
49 #endif
```

vector.cxx

```
1  #include<iostream>
2  #include<cstdlib>
3  #include<cmath>
4  #include"vector.h"
5
6
7  Vector::Vector(int size) try :
8      Size(size), Element(new double[Size]){
9  }
10 catch(std::bad_alloc){
11     std::cerr << "Vector::Vector(int size): Out of Memory!" << std::endl;
12     throw;
13 }
14
15 Vector::~Vector(void){
16     delete []Element;
17 }
18
19 Vector::Vector(const Vector &arg) try :
20     Size(arg.Size), Element(new double[Size]){
21     for(int i=0;i<Size;i++){
22         Element[i]=arg.Element[i];
23     }
24 }
25 catch(std::bad_alloc){
26     std::cerr << "Vector::Vector(int size): Out of Memory!" << std::endl;
27     throw;
28 }
```

```
29
30 Vector::Vector(Vector &&arg)
31     : Size(arg.Size), Element(arg.Element){
32     arg.Size=0;
33     arg.Element=nullptr;
34 }
35
36 Vector::Vector(int dim, double arg, const char *s) try :
37     Size(dim), Element(new double[Size]){
38     if(strcmp(s, "all")!=0){
39         std::cerr << "Invalid string parameter" << std::endl;
40         exit(1);
41     }
42     for(int i=0;i<Size;i++){
43         Element[i]=arg;
44     }
45 }
46 catch(std::bad_alloc){
47     std::cerr << "Vector::Vector(int size): Out of Memory!" << std::endl;
48     throw;
49 }
50
51 Vector &Vector::operator=(const Vector &arg){
52     if(this==&arg) return *this;
53     if(this->Size != arg.Size){
54         Size=arg.Size;
55         delete []Element;
56         try{
57             Element=new double[Size];
58         }
59         catch(std::bad_alloc){
60             std::cerr << "Out of Memory" << std::endl;
61             throw;
62         }
63     }
64     for(int i=0;i<Size;i++){
65         Element[i]=arg.Element[i];
66     }
67     return *this;
68 }
69
70 Vector &Vector::operator=(Vector &&arg){
71     if(this!=&arg){
72         Size=arg.Size;
73         Element=arg.Element;
74         arg.Size=0;
75         arg.Element=nullptr;
76     }
77     return *this;
78 }
```

```
79
80 int Vector::size(void) const{
81     return Size;
82 }
83
84 double Vector::operator[](int index) const{
85     return Element[index];
86 }
87
88 double &Vector::operator[](int index){
89     return Element[index];
90 }
91
92 Vector Vector::operator+(void) const{
93     return *this;
94 }
95
96 Vector Vector::operator-(void) const{
97     Vector result=*this;
98     for(int i=0;i<result.Size;i++){
99         result[i]*=-1.0;
100     }
101     return result;
102 }
103
104 Vector &Vector::operator+=(const Vector &rhs){
105     if(rhs.Size==0){
106         std::cout << "Vector::operator+=:Size 0" << std::endl;
107         exit(1);
108     }
109     else if(Size!=rhs.Size){
110         std::cout << "Vector::operator+=:Size Unmatched" << std::endl;
111         exit(1);
112     }
113     else{
114         for(int i=0;i<Size;i++){
115             Element[i]+=rhs[i];
116         }
117     }
118     return *this;
119 }
120
121 Vector &Vector::operator*=(double rhs){
122     for(int i=0;i<Size;i++){
123         Element[i]*=rhs;
124     }
125     return *this;
126 }
127
128 Vector &Vector::operator/=(double rhs){
```

```
129     for(int i=0;i<Size;i++){
130         Element[i]/=rhs;
131     }
132     return *this;
133 }
134
135
136 Vector &Vector::operator-=(const Vector &rhs){
137     if(rhs.Size==0){
138         std::cout << "Vector::operator-=:Size 0" << std::endl;
139         exit(1);
140     }
141     else if(Size!=rhs.Size){
142         std::cout << "Vector::operator-=:Size Unmatched" << std::endl;
143         exit(1);
144     }
145     else{
146         for(int i=0;i<Size;i++){
147             Element[i]-=rhs[i];
148         }
149     }
150     return *this;
151 }
152
153 Vector Vector::operator+(const Vector &rhs) const{
154     Vector result=*this;
155     return result+=rhs;
156 }
157
158 Vector Vector::operator-(const Vector &rhs) const{
159     Vector result=*this;
160     return result-=rhs;
161 }
162
163 Vector operator*(double lhs, const Vector &rhs){
164     if(rhs.size()==0){
165         std::cout << "Vector operator*:Size 0" << std::endl;
166         exit(1);
167     }
168     Vector result=rhs;
169     for(int i=0;i<result.size();i++){
170         result[i]*=lhs;
171     }
172     return result;
173 }
174
175 Vector operator/(const Vector &lhs, double rhs){
176     if(lhs.size()==0){
177         std::cout << "Vector operator/:Size 0" << std::endl;
178         exit(1);
```

```
179     }
180     Vector result=lhs;
181     return (result/=rhs);
182 }
183
184
185 std::ostream &operator<<(std::ostream &os, const Vector &rhs){
186     os << "(";
187     if(rhs.size()>0){
188         for(int i=0;;i++){
189             os << rhs[i];
190             if(i>=rhs.size()-1) break;
191             os << ", ";
192         }
193     }
194     os << ')';
195     return os;
196 }
197
198 bool Vector::operator==(const Vector &rhs) const{
199     if(Size!=rhs.size()) return false;
200     for(int i=0;i<Size;i++){
201         if(Element[i]!=rhs[i]) return false;
202     }
203     return true;
204 }
205
206 double max_norm(const Vector &arg){
207     if(arg.size()<1){
208         std::cout << "Can't calculate norm for 0-sized vector" << std::endl;
209         exit(1);
210     }
211     double result=fabs(arg[0]);
212     for(int i=1;i<arg.size();i++){
213         double tmp=fabs(arg[i]);
214         if(result<tmp) result=tmp;
215     }
216     return result;
217 }
218
219 double squared_norm(const Vector &arg){
220     return sqrt(norm_square(arg));
221 }
222
223 double norm_square(const Vector &arg){
224     double result=0.0;
225     for(int i=0;i<arg.size();i++){
226         result+=arg[i]*arg[i];
227     }
228     return result;
```



```
229 }
230
231 double Llnorm_square(const Vector &arg){
232     double result=0.0;
233     for(int i=0;i<arg.size();i++){
234         result+=fabs(arg[i]);
235     }
236     return result;
237 }
238
239 double Vector::operator*(const Vector &rhs) const{
240     if(Size<1 || rhs.size()<1 || Size!=rhs.size()){
241         std::cout << "Can't calculate innerproduct";
242         std::cout << "for 0-sized vector";
243         std::cout << "or for different sized vector";
244         std::cout << std::endl;
245         exit(1);
246     }
247     double result=Element[0]*rhs[0];
248     for(int i=1;i<Size;i++){
249         result+=Element[i]*rhs[i];
250     }
251     return result;
252 }
253
254 bool Vector::operator!=(const Vector &rhs) const{
255     if(Size!=rhs.size()) return true;
256     for(int i=0;i<Size;i++){
257         if(Element[i]!=rhs[i]) return true;
258     }
259     return false;
260 }
261
262 Vector Vector::sub(int begin, int end) const{
263     if(end<begin){
264         std::cerr << "Vector::sub:invalid parameter" << std::endl;
265         exit(1);
266     }
267     Vector result(end-begin+1);
268     for(int i=0;i<result.size();i++){
269         result[i]=Element[begin+i];
270     }
271     return result;
272 }
273
274 void Vector::set_sub(int begin, int end, const Vector &arg){
275     if(end<begin){
276         std::cerr << "Vector::sub:invalid parameter" << std::endl;
277         exit(1);
278     }
```

```
279     if(end-begin+1!=arg.size()){
280         std::cerr << "Vector::sub:invalid parameter" << std::endl;
281         exit(1);
282     }
283     for(int i=0;i<arg.size();i++){
284         Element[begin+i]=arg[i];
285     }
286     return;
287 }
288
289 Vector fraction(const Vector &arg){
290     Vector result(arg.size());
291     for(int i=0;i<result.size();i++){
292         result[i]=1.0/arg[i];
293     }
294     return result;
295 }
```

matrix.h

```
1  #include<iostream>
2  #include<cstring>
3  #include"vector.h"
4
5  #ifndef __MATRIX__
6  #define __MATRIX__
7
8  class Matrix{
9  private:
10     int Rows;
11     Vector *Element;
12 public:
13     //Matrix(int rows=0);
14     Matrix(int rows=0, int cols=0);
15     explicit Matrix(int dim, const char *s);
16     explicit Matrix(const Vector &arg, const char *s);
17     ~Matrix(void);
18     Matrix(const Matrix &arg);
19     Matrix &operator=(const Matrix &arg);
20     Matrix(Matrix &&arg);
21     Matrix &operator=(Matrix &&arg);
22     int rows(void) const;
23     int cols(void) const;
24     Vector operator[](int index) const;
25     Vector &operator[](int index);
26     Matrix operator+(void) const;
27     Matrix operator-(void) const;
28     Matrix &operator+=(const Matrix &rhs);
```

```
29     Matrix &operator--(const Matrix &rhs);
30     Matrix &operator*=(double rhs);
31     Matrix &operator/=(double rhs);
32     Matrix sub(int row_begin,
33               int row_end,
34               int col_begin,
35               int col_end) const;
36     void set_sub(int row_begin,
37                 int row_end,
38                 int col_begin,
39                 int col_end,
40                 const Matrix &arg);
41 };
42
43 Matrix operator+(const Matrix &lhs, const Matrix &rhs);
44 Matrix operator-(const Matrix &lhs, const Matrix &rhs);
45 Matrix operator*(double lhs, const Matrix &rhs);
46 Vector operator*(const Matrix &lhs, const Vector &rhs);
47 Matrix operator*(const Matrix &lhs, const Matrix &rhs);
48 Matrix operator*(const Matrix &lhs, double rhs);
49 Matrix operator/(const Matrix &lhs, double rhs);
50 std::ostream &operator<<(std::ostream &os, const Matrix &rhs);
51 bool operator==(const Matrix &lhs, const Matrix &rhs);
52 double max_norm(const Matrix &arg);
53 double frobenius_norm(const Matrix &arg);
54 Matrix transpose(const Matrix &arg);
55 Vector diag(const Matrix &arg);
56 Matrix pow(const Matrix &arg, double power);
57 Matrix transpose(const Vector &arg);
58 Matrix operator*(const Vector &lhs, const Matrix &rhs);
59
60 #endif
```

matrix.cxx

```
1  #include<iostream>
2  #include<cstdlib>
3  #include<cmath>
4  #include"vector.h"
5  #include"matrix.h"
6
7  Matrix::Matrix(int rows, int cols) try :
8      Rows(rows), Element(new Vector[Rows]){
9      for(int i=0;i<Rows;i++){
10         Element[i]=Vector(cols);
11     }
12 }
13 catch(std::bad_alloc){
```

```
14         std::cerr << "Out of Memory" << std::endl;
15         throw;
16     }
17
18     Matrix::Matrix(int dim, const char *s) try :
19         Rows(dim), Element(new Vector[Rows]){
20         if(strcmp(s, "I")!=0){
21             std::cerr << "Invalid string parameter" << std::endl;
22             exit(1);
23         }
24     }
25     for(int i=0;i<Rows;i++){
26         Element[i]=Vector(dim);
27     }
28     for(int i=0;i<Rows;i++){
29         for(int j=0;j<Rows;j++){
30             Element[i][j]=0.0;
31         }
32         Element[i][i]=1.0;
33     }
34 }
35 catch(std::bad_alloc){
36     std::cerr << "Out of Memory" << std::endl;
37     throw;
38 }
39
40 Matrix::Matrix(const Vector &arg, const char *s) try :
41     Rows(arg.size()), Element(new Vector[Rows]){
42     if(strcmp(s, "diag")!=0){
43         std::cerr << "Invalid string parameter" << std::endl;
44         exit(1);
45     }
46 }
47 for(int i=0;i<Rows;i++){
48     Element[i]=Vector(Rows);
49 }
50 for(int i=0;i<Rows;i++){
51     for(int j=0;j<Rows;j++){
52         Element[i][j]=0.0;
53     }
54     Element[i][i]=arg[i];
55 }
56 }
57 catch(std::bad_alloc){
58     std::cerr << "Out of Memory" << std::endl;
59     throw;
60 }
61
62 Matrix::~Matrix(void){
63     delete []Element;
```

```
64  }
65
66  Matrix::Matrix(const Matrix &arg) try :
67      Rows(arg.Rows), Element(new Vector[Rows]){
68      for(int i=0;i<Rows;i++){
69          Element[i]=arg.Element[i];
70      }
71  }
72  catch(std::bad_alloc){
73      std::cerr << "Out of Memory" << std::endl;
74      throw;
75  }
76
77  Matrix::Matrix(Matrix &&arg)
78      : Rows(arg.Rows), Element(arg.Element){
79      arg.Rows=0;
80      arg.Element=nullptr;
81  }
82
83  Matrix &Matrix::operator=(Matrix &&arg){
84      if(this==&arg){
85          return *this;
86      }
87      else{
88          Rows=arg.Rows;
89          Element=arg.Element;
90          arg.Rows=0;
91          arg.Element=nullptr;
92          return *this;
93      }
94  }
95
96  Matrix &Matrix::operator=(const Matrix &arg){
97      if(this==&arg)          return *this;
98      //Rows=arg.Rows; ここでは Rows を更新してはいけない
99      if(this->Rows != arg.Rows || this->cols() != arg.cols()){
100          Rows=arg.Rows;
101          delete []Element;
102          try{
103              Element=new Vector[Rows];
104          }
105          catch(std::bad_alloc){
106              std::cerr << "Out of Memory" << std::endl;
107              throw;
108          }
109      }
110      for(int i=0;i<Rows;i++){
111          Element[i]=arg.Element[i];
112      }
113      return *this;
```

```
114 }
115
116 int Matrix::rows(void) const{
117     return Rows;
118 }
119
120 int Matrix::cols(void) const{
121     return Element[0].size();
122 }
123
124 Vector Matrix::operator[](int index) const{
125     return Element[index];
126 }
127
128 Vector &Matrix::operator[](int index){
129     return Element[index];
130 }
131
132 Matrix Matrix::operator+(void) const{
133     return *this;
134 }
135
136 Matrix Matrix::operator-(void) const{
137     Matrix result=*this;
138     for(int i=0;i<result.Rows;i++){
139         result[i]=-1.0*result[i];
140     }
141     return result;
142 }
143
144 Matrix &Matrix::operator+=(const Matrix &rhs){
145     if(rhs.Rows==0){
146         std::cout << "Rows 0" << std::endl;
147         exit(1);
148     }
149     else if(Rows!=rhs.Rows){
150         std::cout << "Rows Unmatched" << std::endl;
151         exit(1);
152     }
153     else{
154         for(int i=0;i<Rows;i++){
155             Element[i]+=rhs[i];
156         }
157     }
158     return *this;
159 }
160
161 Matrix &Matrix::operator-=(const Matrix &rhs){
162     if(rhs.Rows==0){
163         std::cout << "Rows 0" << std::endl;
```

```
164     exit(1);
165 }
166 else if(Rows!=rhs.Rows){
167     std::cout << "Rows Unmatched" << std::endl;
168     exit(1);
169 }
170 else{
171     for(int i=0;i<Rows;i++){
172         Element[i]-=rhs[i];
173     }
174 }
175 return *this;
176 }
177
178 Matrix operator+(const Matrix &lhs, const Matrix &rhs){
179     Matrix result=lhs;
180     return result+=rhs;
181 }
182
183 Matrix operator-(const Matrix &lhs, const Matrix &rhs){
184     Matrix result=lhs;
185     return result-=rhs;
186 }
187
188 Matrix operator*(double lhs, const Matrix &rhs){
189     if(rhs.rows()==0){
190         std::cout << "Rows 0" << std::endl;
191         exit(1);
192     }
193     Matrix result=rhs;
194     for(int i=0;i<result.rows();i++){
195         result[i]=lhs*result[i];
196     }
197     return result;
198 }
199
200 Matrix &Matrix::operator/=(double rhs){
201     for(int i=0;i<Rows;i++){
202         Element[i]/=rhs;
203     }
204     return *this;
205 }
206
207 Matrix operator/(const Matrix &lhs, double rhs){
208     Matrix result(lhs);
209     return result/=rhs;
210 }
211
212 std::ostream &operator<<(std::ostream &os, const Matrix &rhs){
213     os << "(";
```

```
214     if(rhs.rows()>0){
215         for(int i=0;;i++){
216             os << rhs[i];
217             if(i>=rhs.rows()-1) break;
218             os << "\n";
219         }
220     }
221     os << '))';
222     return os;
223 }
224
225 bool operator==(const Matrix &lhs, const Matrix &rhs){
226     if(lhs.rows()!=rhs.rows()) return false;
227     for(int i=0;i<lhs.rows();i++){
228         if(lhs[i]!=rhs[i]) return false;
229     }
230     return true;
231 }
232
233 double abssum(const Vector &arg){
234     double result=fabs(arg[0]);
235     for(int i=1;i<arg.size();i++){
236         result+=fabs(arg[i]);
237     }
238     return result;
239 }
240
241 double max_norm(const Matrix &arg){
242     if(arg.rows()<1){
243         std::cout << "Can't calculate norm for 0-sized vector" << std::endl;
244         exit(1);
245     }
246     double result=abssum(arg[0]);
247     for(int i=1;i<arg.rows();i++){
248         double tmp=abssum(arg[i]);
249         if(result<tmp) result=tmp;
250     }
251     return result;
252 }
253
254 double frobenius_norm(const Matrix &arg){
255     double result=0.0;
256     for(int i=0;i<arg.rows();i++){
257         for(int j=0;j<arg.cols();j++){
258             result+=arg[i][j]*arg[i][j];
259         }
260     }
261     return sqrt(result);
262 }
263
264 Vector operator*(const Matrix &lhs, const Vector &rhs){
```



```
264     if(lhs.rows()<1 || lhs.cols()<1 || rhs.size()<1 || lhs.cols()!=rhs.size()){
265         std::cout << "operator*(const Matrix &, const Vector &):";
266         std::cout << "Can't calculate innerproduct ";
267         std::cout << "for 0-sized vector ";
268         std::cout << "or for different sized vector:";
269         std::cout << "lhs.Cols=" << lhs.cols() << ", ";
270         std::cout << "lhs.Rows=" << lhs.rows() << ", ";
271         std::cout << "rhs.Size=" << rhs.size();
272         std::cout << std::endl;
273         exit(1);
274     }
275     Vector result(lhs.rows());
276     for(int i=0;i<lhs.rows();i++){
277         result[i]=lhs[i]*rhs;
278     }
279     return result;
280 }
281
282 Matrix operator*(const Matrix &lhs, const Matrix &rhs){
283     if(lhs.rows()<1 || rhs.cols()<1 || lhs.cols()!=rhs.rows()){
284         std::cout << "Can't calculate innerproduct";
285         std::cout << "for 0-sized vector";
286         std::cout << "or for different sized vector";
287         std::cout << std::endl;
288         exit(1);
289     }
290     Matrix result(lhs.rows(), rhs.cols());
291     for(int i=0;i<result.rows();i++){
292         for(int j=0;j<result.cols();j++){
293             result[i][j]=0.0;
294             for(int k=0;k<lhs.cols();k++){
295                 result[i][j]+=lhs[i][k]*rhs[k][j];
296             }
297         }
298     }
299     return result;
300 }
301
302 Matrix Matrix::sub(int row_begin, int row_end,
303                    int col_begin, int col_end) const{
304     if(row_end<row_begin || col_end<col_begin){
305         std::cerr << "Matrix::sub:invalid parameter" << std::endl;
306         std::cerr << "row_begin:" << row_begin << std::endl;
307         std::cerr << "row_end:" << row_end << std::endl;
308         std::cerr << "col_begin:" << col_begin << std::endl;
309         std::cerr << "col_end:" << col_end << std::endl;
310         exit(1);
311     }
312     if(row_end>=this->rows() || col_end>=this->cols()){
313         std::cerr << "Matrix::sub:invalid parameter" << std::endl;
```

```
314     std::cerr << "row_end:" << row_end << std::endl;
315     std::cerr << "Rows:" << this->rows() << std::endl;
316     std::cerr << "col_end:" << col_end << std::endl;
317     std::cerr << "Cols:" << this->cols() << std::endl;
318     exit(1);
319 }
320 if(row_begin<0 || col_begin<0){
321     std::cerr << "Matrix::sub:invalid parameter" << std::endl;
322     std::cerr << "row_begin:" << row_begin << std::endl;
323     std::cerr << "col_begin:" << col_begin << std::endl;
324     exit(1);
325 }
326 Matrix result(row_end-row_begin+1, col_end-col_begin+1);
327 for(int i=0;i<result.rows();i++){
328     for(int j=0;j<result.cols();j++){
329         result[i][j]=Element[i+row_begin][j+col_begin];
330     }
331 }
332 return result;
333 }
334 void Matrix::set_sub(int row_begin, int row_end,
335                     int col_begin, int col_end,
336                     const Matrix &arg){
337
338     if(row_end<row_begin || col_end<col_begin){
339         std::cerr << "Matrix::sub:invalid parameter" << std::endl;
340         exit(1);
341     }
342     for(int i=row_begin;i<=row_end;i++){
343         for(int j=col_begin;j<=col_end;j++){
344             Element[i][j]=arg[i-row_begin][j-col_begin];
345         }
346     }
347     return;
348 }
349 Matrix transpose(const Matrix &arg){
350     Matrix result(arg.cols(), arg.rows());
351     for(int i=0;i<result.rows();i++){
352         for(int j=0;j<result.cols();j++){
353             result[i][j]=arg[j][i];
354         }
355     }
356     return result;
357 }
358 Vector diag(const Matrix &arg){
359     if(arg.rows()!=arg.cols()){
360         std::cerr << "No Diag" << std::endl;
361         exit(1);
362     }
363     Vector result(arg.rows());
```

```
364     for(int i=0;i<result.size();i++){
365         result[i]=arg[i][i];
366     }
367     return result;
368 }
369
370 Matrix pow(const Matrix &arg, double power){
371     Matrix result(arg);
372     for(int i=0;i<result.rows();i++){
373         for(int j=0;j<result.cols();j++){
374             result[i][j]=pow(result[i][j],power);
375         }
376     }
377     return result;
378 }
379
380 Matrix transpose(const Vector &arg){
381     Matrix result(1, arg.size());
382     for(int j=0;j<result.cols();j++){
383         result[0][j]=arg[j];
384     }
385     return result;
386 }
387
388 Matrix operator*(const Vector &lhs, const Matrix &rhs){
389     if(rhs.rows()!=1){
390         std::cerr << "Size unmatched for Vector*Matrix:" << rhs.rows() << ":" << rhs.cols() << endl;
391         exit(1);
392     }
393     Matrix result(lhs.size(), rhs.cols());
394     for(int i=0;i<result.rows();i++){
395         for(int j=0;j<result.cols();j++){
396             result[i][j]=lhs[i]*rhs[0][j];
397         }
398     }
399     return result;
400 }
```

randGaussianMain.cxx

```
1  #include<random>
2  #include<iostream>
3
4  int main(void){
5      std::random_device rnd;
6      std::mt19937 mt(rnd());
7      std::normal_distribution<> normDist(0.0, 1.0);
8
9      const int dimension=2, eachDataNum=50, clusterNum=2;
10     const double means[clusterNum][dimension]={
```

```

11     {-1,-1},{1,1}};
12     const double stddevs[clusterNum][dimension]={
13         {0.5,0.5},{0.5,0.5}};
14
15     std::cout << eachDataNum*clusterNum << "\t" << dimension << std::endl;
16
17     for(int i=0;i<clusterNum;i++){
18         for(int k=0;k<eachDataNum;k++){
19             for(int ell=0;ell<dimension;ell++){
20                 std::cout << normDist(mt)*stddevs[i][ell]+means[i][ell] << "\t";
21             }
22             std::cout << std::endl;
23         }
24     }
25     return 0;
26 }

```

hcm.cxx

```

1  #include"hcm.h"
2  #include <boost/math/special_functions/binomial.hpp>
3
4  Hcm::Hcm(int dimension,
5           int data_number,
6           int centers_number):
7      Data(data_number, dimension),
8      Centers(centers_number, dimension),
9      Tmp_Centers(centers_number, dimension),
10     Membership(centers_number, data_number),
11     Tmp_Membership(centers_number, data_number),
12     Alpha(centers_number),
13     Tmp_Alpha(centers_number),
14     Dissimilarities(centers_number, data_number),
15     CrispMembership(centers_number, data_number),
16     CorrectCrispMembership(centers_number, data_number),
17     ContingencyTable(centers_number+1, centers_number+1),
18     Iterates(0){
19     /** 収束判定のために DBL_MAX に設定***/
20     for(int i=0;i<centers_number;i++){
21         Centers[i]=Vector(dimension);
22         for(int ell=0;ell<dimension;ell++){
23             Centers[i][ell]=DBL_MAX;
24         }
25     }
26     /** 収束判定のために DBL_MAX に設定***/
27     for(int i=0;i<centers_number;i++){
28         for(int k=0;k<data_number;k++){
29             Membership[i][k]=DBL_MAX;

```

```
30     }
31 }
32 }
33
34 void Hcm::revise_dissimilarities(void){
35     for(int i=0;i<centers_number();i++){
36         for(int k=0;k<data_number();k++){
37             Dissimilarities[i][k]=norm_square(Data[k]-Centers[i]);
38         }
39     }
40     return;
41 }
42
43 void Hcm::revise_membership(void){
44     Tmp_Membership=Membership;
45     for(int k=0;k<data_number();k++){
46         int min_index=0; double min_dissimilarity=Dissimilarities[0][k];
47         for(int i=1;i<centers_number();i++){
48             if(min_dissimilarity>Dissimilarities[i][k]){
49                 min_index=i;
50                 min_dissimilarity=Dissimilarities[i][k];
51             }
52         }
53         for(int i=0;i<centers_number();i++){
54             Membership[i][k]=0.0;
55             Membership[min_index][k]=1.0;
56         }
57     }
58     return;
59 }
60
61 void Hcm::revise_centers(void){
62     Tmp_Centers=Centers;
63     for(int i=0;i<centers_number();i++){
64         double denominator=0.0;
65         Vector numerator(Centers[i].size());
66         for(int ell=0;ell<numerator.size();ell++){
67             numerator[ell]=0.0;
68         }
69         for(int k=0;k<data_number();k++){
70             denominator+=Membership[i][k];
71             numerator+=Membership[i][k]*Data[k];
72         }
73         Centers[i]=numerator/denominator;
74     }
75     return;
76 }
77
78 int Hcm::dimension(void) const{
79     return Data[0].size();
80 }
```

```
80
81  int Hcm::data_number(void) const{
82      return Data.rows();
83  }
84
85  int Hcm::centers_number(void) const{
86      return Centers.rows();
87  }
88
89  Matrix Hcm::centers(void) const{
90      return Centers;
91  }
92
93  Matrix Hcm::tmp_centers(void) const{
94      return Tmp_Centers;
95  }
96
97  Matrix Hcm::data(void) const{
98      return Data;
99  }
100
101  Matrix Hcm::membership(void) const{
102      return Membership;
103  }
104
105  Matrix Hcm::tmp_membership(void) const{
106      return Tmp_Membership;
107  }
108
109  int &Hcm::iterates(void){
110      return Iterates;
111  }
112
113  Matrix Hcm::dissimilarities(void) const{
114      return Dissimilarities;
115  }
116
117  double &Hcm::data(int index1, int index2){
118      return Data[index1][index2];
119  }
120
121  double &Hcm::centers(int index1, int index2){
122      return Centers[index1][index2];
123  }
124
125  double &Hcm::membership(int row, int col){
126      return Membership[row][col];
127  }
128
129  double Hcm::objective(void) const{
```

```
130     return Objective;
131 }
132
133 void Hcm::set_objective(void){
134     Objective=0.0;
135     for(int i=0;i<centers_number();i++){
136         for(int k=0;k<data_number();k++){
137             Objective+=Membership[i][k]*Dissimilarities[i][k];
138         }
139     }
140     return;
141 }
142
143 double &Hcm::dissimilarities(int index1, int index2){
144     return Dissimilarities[index1][index2];
145 }
146
147 void Hcm::set_crispMembership(void){
148     for(int k=0;k<data_number();k++){
149         for(int i=0;i<centers_number();i++){
150             CrispMembership[i][k]=0.0;
151         }
152         double max=-DBL_MAX;
153         int max_index=-1;
154         for(int i=0;i<centers_number();i++){
155             if(Membership[i][k]>max){
156                 max=Membership[i][k];
157                 max_index=i;
158             }
159         }
160         CrispMembership[max_index][k]=1.0;
161     }
162     return;
163 }
164
165 Matrix Hcm::crispMembership(void) const{
166     return CrispMembership;
167 }
168
169 double &Hcm::crispMembership(int index1, int index2){
170     return CrispMembership[index1][index2];
171 }
172
173 Matrix Hcm::correctCrispMembership(void) const{
174     return CorrectCrispMembership;
175 }
176
177 double &Hcm::correctCrispMembership(int index1, int index2){
178     return CorrectCrispMembership[index1][index2];
179 }
```

```

180 void Hcm::set_contingencyTable(void){
181     ContingencyTable.set_sub(0,centers_number()-1, 0, centers_number()-1,CrispMembershi
182
183     for(int i=0;i<ContingencyTable.rows()-1;i++){
184         ContingencyTable[i][ContingencyTable.cols()-1]=0.0;
185         for(int j=0;j<ContingencyTable.cols()-1;j++){
186             ContingencyTable[i][ContingencyTable.cols()-1]+=ContingencyTable[i][j];
187         }
188     }
189     for(int j=0;j<ContingencyTable.cols()-1;j++){
190         ContingencyTable[ContingencyTable.rows()-1][j]=0.0;
191         for(int i=0;i<ContingencyTable.rows()-1;i++){
192             ContingencyTable[ContingencyTable.rows()-1][j]+=ContingencyTable[i][j];
193         }
194     }
195     ContingencyTable[ContingencyTable.rows()-1][ContingencyTable.cols()-1]=data_number(
196     return;
197 }
198
199 Matrix Hcm::contingencyTable(void) const{
200     return ContingencyTable;
201 }
202
203 double combination(int n, int k){
204     if(n<k) return 0.0;
205     return boost::math::binomial_coefficient<double>(n, k);
206 }
207
208 double Hcm::ARI(void) const{
209     double Index=0.0;
210     for(int i=0;i<ContingencyTable.rows()-1;i++){
211         for(int j=0;j<ContingencyTable.cols()-1;j++){
212             Index+=ContingencyTable[i][j]*ContingencyTable[i][j];
213         }
214     }
215     Index=0.5*(Index-ContingencyTable[ContingencyTable.rows()-1][ContingencyTable.cols(
216     // std::cout << "Index:" << Index << std::endl;
217     double ExpectedIndexI=0.0;
218     for(int i=0;i<ContingencyTable.rows()-1;i++){
219         ExpectedIndexI+=combination(ContingencyTable[i][ContingencyTable.cols()-1], 2);
220     }
221     // std::cout << "ExpectedIndexI:" << ExpectedIndexI << std::endl;
222     double ExpectedIndexJ=0.0;
223     for(int j=0;j<ContingencyTable.cols()-1;j++){
224         ExpectedIndexJ+=combination(ContingencyTable[ContingencyTable.rows()-1][j], 2);
225     }
226     // std::cout << "ExpectedIndexJ:" << ExpectedIndexJ << std::endl;
227     double ExpectedIndex=ExpectedIndexI*ExpectedIndexJ/comboination(ContingencyTable[Con
228     // std::cout << "Denom:" << comboination(ContingencyTable[ContingencyTable.rows()-1
229     double MaxIndex=0.5*(ExpectedIndexI+ExpectedIndexJ);

```



```
230
231     return (Index-ExpectedIndex)/(MaxIndex-ExpectedIndex);
232 }
233
234 Vector &Hcm::data(int index1){
235     return Data[index1];
236 }
237
238 Vector &Hcm::centers(int index1){
239     return Centers[index1];
240 }
241
242 Vector Hcm::alpha(void) const{
243     return Alpha;
244 }
245
246 double &Hcm::alpha(int index1){
247     return Alpha[index1];
248 }
249
250 Vector Hcm::tmp_alpha(void) const{
251     return Tmp_Alpha;
252 }
```

hcm.h

```
1  #include<cmath>
2  #include<cstdio>
3  #include"matrix.h"
4
5  #ifndef __HCM__
6  #define __HCM__
7
8  class Hcm{
9  protected:
10     Matrix Data, Centers, Tmp_Centers;
11     Matrix Membership, Tmp_Membership, Dissimilarities;
12     Matrix CrispMembership, CorrectCrispMembership, ContingencyTable;
13     Vector Alpha, Tmp_Alpha;
14     int Iterates;
15     double Objective;
16 public:
17     Hcm(int dimension,
18         int data_number,
19         int centers_number);
20     virtual void revise_membership(void);
21     virtual void revise_dissimilarities(void);
22     virtual void revise_centers(void);
```

```
23     int dimension(void) const;
24     int data_number(void) const;
25     int centers_number(void) const;
26     Matrix centers(void) const;
27     Matrix tmp_centers(void) const;
28     Matrix data(void) const;
29     Matrix membership(void) const;
30     Matrix tmp_membership(void) const;
31     Vector alpha(void) const;
32     double &alpha(int index);
33     Vector tmp_alpha(void) const;
34     int &iterates(void);
35     Matrix dissimilarities(void) const;
36     double &data(int index1, int index2);
37     Vector &data(int index1);
38     double &centers(int index1, int index2);
39     Vector &centers(int index1);
40     double &membership(int index1, int index2);
41     double &dissimilarities(int index1, int index2);
42     void set_objective(void);
43     double objective(void) const;
44     void set_crispMembership(void);
45     Matrix crispMembership(void) const;
46     double &crispMembership(int index1, int index2);
47     Matrix correctCrispMembership(void) const;
48     double &correctCrispMembership(int index1, int index2);
49     void set_contingencyTable(void);
50     Matrix contingencyTable(void) const;
51     double ARI(void) const;
52 };
53
54 #endif
```

sfcmm.h

```
1  #include<cmath>
2  #include<cstdio>
3  #include"matrix.h"
4  #include"hcm.h"
5
6  #ifndef __SFCM__
7  #define __SFCM__
8
9  class Sfcmm: virtual public Hcm{
10 protected:
11     double FuzzifierEm;
12 public:
13     Sfcmm(int dimension,
```

```
14         int data_number,
15         int centers_number,
16         double fuzzifierEm);
17     double fuzzifierEm(void) const;
18     double &fuzzifierEm(void);
19     virtual void revise_membership(void);
20     virtual void revise_centers(void);
21 };
22
23 #endif
24
```

sfcm.cxx

```
1  #include "sfcm.h"
2
3  Sfcm::Sfcm(int dimension,
4             int data_number,
5             int centers_number,
6             double fuzzifierEm)
7      : Hcm(dimension, data_number, centers_number), FuzzifierEm(fuzzifierEm){
8  }
9
10 double Sfcm::fuzzifierEm(void) const{
11     return FuzzifierEm;
12 }
13
14 double &Sfcm::fuzzifierEm(void){
15     return FuzzifierEm;
16 }
17
18 void Sfcm::revise_membership(void){
19     Tmp_Membership=Membership;
20     for(int k=0;k<data_number();k++){
21         int numZeroDissimilarities=0;
22         Vector indexZeroDissimilarities(centers_number(), 0.0, "all");
23         for(int i=0;i<centers_number();i++){
24             if(Dissimilarities[i][k]==0.0){
25                 numZeroDissimilarities++;
26                 indexZeroDissimilarities[i]=1.0;
27             }
28         }
29         if(numZeroDissimilarities!=0){
30             for(int i=0;i<centers_number();i++){
31                 Membership[i][k]=indexZeroDissimilarities[i]/numZeroDissimilarities;
32             }
33         }
34     }
35 }
```

```

35         for(int i=0;i<centers_number();i++){
36             double denominator=0.0;
37             for(int j=0;j<centers_number();j++){
38                 denominator+=pow(Dissimilarities[i][k]/Dissimilarities[j][k], 1.0/(Fuzzifie
39             })
40             Membership[i][k]=1.0/denominator;
41         }
42     }//else
43 }//k
44     return;
45 }
46
47 void Sfcm::revise_centers(void){
48     Tmp_Centers=Centers;
49     for(int i=0;i<centers_number();i++){
50         double denominator=0.0;
51         Vector numerator(dimension(), 0.0, "all");
52         for(int k=0;k<data_number();k++){
53             denominator+=pow(Membership[i][k], FuzzifierEm);
54             numerator+=pow(Membership[i][k], FuzzifierEm)*Data[k];
55         }
56         Centers[i]=numerator/denominator;
57     }
58     return;
59 }

```

efcm.h

```

1  #include<cmath>
2  #include<cfloat>
3  #include"matrix.h"
4  #include"hcm.h"
5
6  #ifndef __EFCM__
7  #define __EFCM__
8
9  class Efcm: virtual public Hcm{
10 protected:
11     double FuzzifierLambda;
12 public:
13     Efcm(int dimension,
14         int data_number,
15         int centers_number,
16         double fuzzifierLambda);
17     double fuzzifierLambda(void)const;
18     double &fuzzifierLambda(void);
19     virtual void revise_membership(void);
20     virtual void revise_centers(void);

```

```
21  };
22
23  #endif
24
```

efcm.cxx

```
1  #include"efcm.h"
2
3  EfcM::EfcM(int dimension,
4             int data_number,
5             int centers_number,
6             double fuzzifierLambda)
7      : Hcm(dimension, data_number, centers_number), FuzzifierLambda(fuzzifierLambda){
8  }
9
10
11 double EfcM::fuzzifierLambda(void)const{
12     return FuzzifierLambda;
13 }
14
15 double &EfcM::fuzzifierLambda(void){
16     return FuzzifierLambda;
17 }
18
19 void EfcM::revise_membership(void){
20     Tmp_Membership=Membership;
21     for(int k=0;k<data_number();k++){
22         int numZeroDissimilarities=0;
23         Vector indexZeroDissimilarities(centers_number(), 0.0, "all");
24         for(int i=0;i<centers_number();i++){
25             if(Dissimilarities[i][k]==0.0){
26                 numZeroDissimilarities++;
27                 indexZeroDissimilarities[i]=1.0;
28             }
29         }
30         if(numZeroDissimilarities!=0){
31             for(int i=0;i<centers_number();i++){
32                 Membership[i][k]=indexZeroDissimilarities[i]/numZeroDissimilarities;
33             }
34         }
35         else{
36             for(int i=0;i<centers_number();i++){
37                 double denominator=0.0;
38                 for(int j=0;j<centers_number();j++){
39                     denominator+=exp(-1.0*FuzzifierLambda*Dissimilarities[j][k])/exp(-1.0*Fuzzi
40                 }
41                 Membership[i][k]=1.0/denominator;

```

```
42     }
43     }//else
44 }//k
45     return;
46 }
47
48 void Efc::revise_centers(void){
49     Tmp_Centers=Centers;
50     for(int i=0;i<centers_number();i++){
51         double denominator=0.0;
52         Vector numerator(dimension(), 0.0, "all");
53         for(int k=0;k<data_number();k++){
54             denominator+=Membership[i][k];
55             numerator+=Membership[i][k]*Data[k];
56         }
57         Centers[i]=numerator/denominator;
58     }
59     return;
60 }
```

qfcm.h

```
1  #include<cmath>
2  #include<cstdio>
3  #include"matrix.h"
4  #include"hcm.h"
5  #include"sfc.h"
6  #include"efcm.h"
7
8  #ifndef __QFCM__
9  #define __QFCM__
10
11  class Qfcm: public Sfc, public Efc{
12  public:
13      Qfcm(int dimension,
14          int data_number,
15          int centers_number,
16          double fuzzifierEm,
17          double fuzzifierLambda);
18      void revise_membership(void);
19      void revise_centers(void);
20 };
21
22 #endif
23
```

qfcm.cxx

```
1  #include"qfcm.h"
2
3  Qfcm::Qfcm(int dimension,
4             int data_number,
5             int centers_number,
6             double fuzzifierEm,
7             double fuzzifierLambda):
8      Hcm(dimension, data_number, centers_number),
9      Sfcm(dimension, data_number, centers_number,fuzzifierEm),
10     Efcm(dimension, data_number, centers_number,fuzzifierLambda){
11 }
12
13 void Qfcm::revise_membership(void){
14     Tmp_Membership=Membership;
15     for(int k=0;k<data_number();k++){
16         int numZeroDissimilarities=0;
17         Vector indexZeroDissimilarities(centers_number(), 0.0, "all");
18         for(int i=0;i<centers_number();i++){
19             if(Dissimilarities[i][k]==0.0){
20                 numZeroDissimilarities++;
21                 indexZeroDissimilarities[i]=1.0;
22             }
23         }
24         if(numZeroDissimilarities!=0){
25             for(int i=0;i<centers_number();i++){
26                 Membership[i][k]=indexZeroDissimilarities[i]/numZeroDissimilarities;
27             }
28         }
29         else{
30             for(int i=0;i<centers_number();i++){
31                 double denominator=0.0;
32                 for(int j=0;j<centers_number();j++){
33                     //帰属度
34                     denominator+=(FuzzifierLambda*(FuzzifierEm-1.0)*Dissimilarities[i][j]+1)/(F
35                 }
36                 Membership[i][k]=1.0/pow(denominator,1/(1.0-FuzzifierEm));
37             }
38         }//else
39     }//k
40     return;
41 }
42
43 void Qfcm::revise_centers(void){
44     Sfcm::revise_centers();
45     return;
46 }
```

sfcma.h

```
1  #include"hcm.h"
2  #include"sfcma.h"
3
4  #ifndef __SFCMA__
5  #define __SFCMA__
6
7  class Sfcma: virtual public Hcm, public Sfcma{
8  public:
9      Sfcma(const int &dimension,
10             const int &data_number,
11             const int &centers_number,
12             const double &fuzzifierEm);
13     virtual void revise_membership(void);
14     virtual void revise_centers(void);
15     virtual void revise_alpha(void);
16 };
17 #endif
```

sfcma.cxx

```
1  #include"sfcma.h"
2
3  Sfcma::Sfcma(const int &dimension,
4               const int &data_number,
5               const int &centers_number,
6               const double &fuzzifierEm) :
7      Hcm(dimension, data_number, centers_number),
8      Sfcma(dimension, data_number, centers_number,fuzzifierEm){
9  }
10
11 void Sfcma::revise_membership(void){
12     Tmp_Membership=Membership;
13     for(int k=0;k<data_number();k++){
14         int numZeroDissimilarities=0;
15         Vector indexZeroDissimilarities(centers_number(), 0.0, "all");
16         for(int i=0;i<centers_number();i++){
17             if(Dissimilarities[i][k]==0.0){
18                 numZeroDissimilarities++;
19                 indexZeroDissimilarities[i]=1.0;
20             }
21         }
22         if(numZeroDissimilarities!=0){
23             for(int i=0;i<centers_number();i++){
24                 Membership[i][k]=indexZeroDissimilarities[i]/numZeroDissimilarities;
```



```
25     }
26 }
27 else{
28     for(int i=0;i<centers_number();i++){
29         double denominator=0.0;
30         for(int j=0;j<centers_number();j++){
31             denominator+=Alpha[j]/Alpha[i]
32             *pow(Dissimilarities[i][k]/Dissimilarities[j][k],
33                 1.0/(FuzzifierEm-1.0));
34         }
35         Membership[i][k]=1.0/denominator;
36     }
37 }
38 }
39 return;
40 }
41
42 void Sfcma::revise_centers(void){
43     Sfcma::revise_centers();
44     return;
45 }
46
47 void Sfcma::revise_alpha(void){
48     Tmp_Alpha=Alpha;
49     double denominator=0.0;
50     for(int j=0;j<centers_number();j++){
51         double tmp1=0.0;
52         for(int k=0;k<data_number();k++){
53             tmp1+=pow(Membership[j][k],FuzzifierEm)*Dissimilarities[j][k];
54         }
55         denominator+=pow(tmp1,1.0/FuzzifierEm);
56     }
57     for(int i=0;i<centers_number();i++){
58         double tmp2=0.0;
59         for(int k=0;k<data_number();k++){
60             tmp2+=pow(Membership[i][k],FuzzifierEm)*Dissimilarities[i][k];
61         }
62         Alpha[i]=pow(tmp2,1.0/FuzzifierEm)/denominator;
63     }
64     return;
65 }
```

efcma.h

```
1
2 #include"efcm.h"
3
4 #ifndef __EFCMA__
```

```
5  #define __EFCMA__
6
7  class Efcma: public Efcma {
8  public:
9      Efcma(
10         int dimension,
11         int data_number,
12         int centers_number,
13         double fuzzifierLambda);
14      void revise_membership(void);
15      void revise_alpha(void);
16  };
17
18 #endif
19
```

efcma.cxx

```
1
2  #include"efcma.h"
3
4  Efcma::Efcma(int dimension,
5              int data_number,
6              int centers_number,
7              double fuzzifierLambda)
8      : Hcm(dimension, data_number, centers_number),
9        Efcma(dimension, data_number, centers_number, fuzzifierLambda){
10 }
11
12 void Efcma::revise_membership(void){
13     Tmp_Membership=Membership;
14     for(int k=0;k<data_number();k++){
15         for(int i=0;i<centers_number();i++){
16             double denominator=0.0;
17             for(int j=0;j<centers_number();j++){
18                 denominator+=(Alpha[j]/Alpha[i])*exp(FuzzifierLambda*(Dissimilarities[i][k]-D
19             })
20             Membership[i][k]=1.0/denominator;
21         }
22     }
23     return;
24 }
25
26 void Efcma::revise_alpha(void){
27     Tmp_Alpha=Alpha;
28     for(int i=0;i<centers_number();i++){
29         double numerator=0;
30         for(int k=0;k<data_number();k++){
```

```
31     numerator+=Membership[i][k];
32     }
33     Alpha[i]=numerator/data_number();
34     }
35     return;
36     }
```

qfcma.h

```
1  #include"efcma.h"
2  #include"sfcma.h"
3
4  #ifndef __QFCMA__
5  #define __QFCMA__
6
7  class Qfcma: public Efcma, public Sfcma{
8
9  public:
10     Qfcma(int dimension,
11           int data_number,
12           int centers_number,
13           double fuzzifierEm,
14           double fuzzifierLambda);
15     virtual void revise_membership(void);
16     virtual void revise_centers(void);
17     virtual void revise_alpha(void);
18 };
19
20 #endif
21
```

qfcma.cxx

```
1  #include"qfcma.h"
2
3  Qfcma::Qfcma(int dimension,
4               int data_number,
5               int centers_number,
6               double fuzzifierEm,
7               double fuzzifierLambda) :
8     Hcm(dimension, data_number, centers_number),
9     Sfcma(dimension, data_number, centers_number,fuzzifierEm),
10     Efcma(dimension, data_number, centers_number,fuzzifierLambda){
11 }
12
```

```
13 void Qfcma::revise_membership(void){
14     Tmp_Membership=Membership;
15     for(int k=0;k<data_number();k++){
16         int numZeroDissimilarities=0;
17         Vector indexZeroDissimilarities(centers_number(), 0.0, "all");
18         for(int i=0;i<centers_number();i++){
19             if(Dissimilarities[i][k]==0.0){
20                 numZeroDissimilarities++;
21                 indexZeroDissimilarities[i]=1.0;
22             }
23         }
24         if(numZeroDissimilarities!=0){
25             for(int i=0;i<centers_number();i++){
26                 Membership[i][k]=indexZeroDissimilarities[i]/numZeroDissimilarities;
27             }
28         }
29         else{
30             for(int i=0;i<centers_number();i++){
31                 double denominator=0.0;
32                 for(int j=0;j<centers_number();j++){
33                     denominator+=Alpha[j]/Alpha[i]
34                     *pow((1.0-FuzzifierLambda*(1.0-FuzzifierEm)
35                     *Dissimilarities[j][k])
36                     /(1.0-FuzzifierLambda*(1.0-FuzzifierEm)
37                     *Dissimilarities[i][k])
38                     ,1.0/(1.0-FuzzifierEm));
39                 }
40                 Membership[i][k]=1.0/denominator;
41             }
42         }//else
43     }//k
44     return;
45 }
46
47 void Qfcma::revise_centers(void){
48     Sfcma::revise_centers();
49     return;
50 }
51
52 void Qfcma::revise_alpha(void){
53     Tmp_Alpha=Alpha;
54     double denominator=0.0;
55     for(int j=0;j<centers_number();j++){
56         double tmp1=0.0;
57         for(int k=0;k<data_number();k++){
58             tmp1+=pow(Membership[j][k],FuzzifierEm)
59             *(1.0-FuzzifierLambda*(1.0-FuzzifierEm)*Dissimilarities[j][k]);
60         }
61         denominator+=pow(tmp1,1.0/FuzzifierEm);
62     }
```

```
63     for(int i=0;i<centers_number();i++){
64         double tmp2=0.0;
65         for(int k=0;k<data_number();k++){
66             tmp2+=pow(Membership[i][k],FuzzifierEm)
67                 *(1.0-FuzzifierLambda*(1.0-FuzzifierEm))*Dissimilarities[i][k];
68         }
69         Alpha[i]=pow(tmp2,1/FuzzifierEm)/denominator;
70     }
71     return;
72 }
```

sfcm_main_2d-Gaussian-2clusters.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"sfcm.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9
10 const int centers_number=2;
11
12 int main(void){
13     std::string filenameData("2d-Gaussian-2clusters.dat");
14     #ifdef CHECK_ANSWER
15         std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
16     #endif
17
18     std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
19     if(filenameDataDotPosition==std::string::npos){
20         std::cerr << "File:" << filenameData
21             << " needs \"\\.\" and filename-extension." << std::endl;
22         exit(1);
23     }
24
25     std::ifstream ifs(filenameData);
26     if(!ifs){
27         std::cerr << "File:" << filenameData
28             << " could not open." << std::endl;
29         exit(1);
30     }
31     int data_number, data_dimension;
32     ifs >> data_number;
33     ifs >> data_dimension;
34
35     Sfcm test(data_dimension, data_number, centers_number, 2.0);
```

```
36
37     for(int cnt=0;cnt<data_number;cnt++){
38         for(int ell=0;ell<data_dimension;ell++){
39             ifs >> test.data(cnt, ell);
40         }
41     }
42
43     /**Initial Centers Setting***/
44     std::random_device rnd;
45     std::mt19937 mt(rnd());
46     std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
47     for(int i=0;i<test.centers_number();i++){
48         test.centers(i)=test.data()[randDataNumber(mt)];
49     }
50 #ifdef VERBOSE
51     std::cout << "v:\n" << test.centers() << std::endl;
52 #endif
53
54     test.iterates()=0;
55     while(1){
56         test.revise_dissimilarities();
57 #ifdef VERBOSE
58         std::cout << "d:\n" << test.dissimilarities() << std::endl;
59 #endif
60         test.revise_membership();
61 #ifdef VERBOSE
62         std::cout << "u:\n" << test.membership() << std::endl;
63 #endif
64         test.revise_centers();
65 #ifdef VERBOSE
66         std::cout << "v:\n" << test.centers() << std::endl;
67 #endif
68
69         double diff_u=max_norm(test.tmp_membership()-test.membership());
70         double diff_v=max_norm(test.tmp_centers()-test.centers());
71         double diff=diff_u+diff_v;
72 #ifdef DIFF
73         std::cout << "#diff:" << diff << "\t";
74         std::cout << "#diff_u:" << diff_u << "\t";
75         std::cout << "#diff_v:" << diff_v << "\n";
76 #endif
77         if(diff<DIFF_FOR_STOP)break;
78         if(test.iterates()>=MAX_ITERATES)break;
79         test.iterates()++;
80     }
81 #ifdef VERBOSE
82     std::cout << "v:\n" << test.centers() << std::endl;
83 #endif
84
85 #ifdef CHECK_ANSWER
```

```
86     test.set_crispMembership();
87
88     std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
89     if(!ifs_correctCrispMembership){
90         std::cerr << "File:" << filenameCorrectCrispMembership
91             << " could not open." << std::endl;
92         exit(1);
93     }
94     for(int i=0;i<test.centers_number();i++){
95         for(int k=0;k<test.data_number();k++){
96             ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
97         }
98     }
99     test.set_contingencyTable();
100    std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
101    std::cout << "ARI:" << test.ARI() << std::endl;
102 #endif
103
104    std::string filenameResultMembership
105        =std::string("sFCM-Em")+std::to_string(test.fuzzifierEm())+std::string("-")
106        +filenameData.substr(0, filenameDataDotPosition)
107        +std::string(".result_membership");
108    std::ofstream ofs_membership(filenameResultMembership);
109    if(!ofs_membership){
110        std::cerr << "File:" << filenameResultMembership
111            << "could not open." << std::endl;
112        exit(1);
113    }
114
115    for(int k=0;k<test.data_number();k++){
116        for(int ell=0;ell<test.dimension();ell++){
117            ofs_membership << test.data()[k][ell] << "\t";
118        }
119        for(int i=0;i<test.centers_number();i++){
120            ofs_membership << test.membership()[i][k] << "\t";
121        }
122        ofs_membership << std::endl;
123    }
124    ofs_membership.close();
125
126    std::string filenameResultCenters
127        =std::string("sFCM-Em")+std::to_string(test.fuzzifierEm())+std::string("-")
128        +filenameData.substr(0, filenameDataDotPosition)
129        +std::string(".result_centers");
130    std::ofstream ofs_centers(filenameResultCenters);
131    if(!ofs_centers){
132        std::cerr << "File:" << filenameResultCenters
133            << "could not open." << std::endl;
134        exit(1);
135    }
```

```

136     for(int i=0;i<test.centers_number();i++){
137         for(int ell=0;ell<test.dimension();ell++){
138             ofs_centers << test.centers()[i][ell] << "\t";
139         }
140         ofs_centers << std::endl;
141     }
142     ofs_centers.close();
143
144 #ifdef CLASSIFICATION_FUNCTION
145     //Classification Function
146     if(test.dimension()>2){
147         std::cerr << "Dimension:" << test.dimension()
148             << "is too high for classification function visualization."
149             << std::endl;
150         exit(1);
151     }
152     Sfcm ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierEm());
153     std::string filenameClassificationFunction
154         =std::string("sFCM-Em")+std::to_string(test.fuzzifierEm())+std::string("-")
155         +filenameData.substr(0, filenameDataDotPosition)
156         +std::string(".result_classificationFunction");
157     std::ofstream ofs_classificationFunction(filenameClassificationFunction);
158     if(!ofs_classificationFunction){
159         std::cerr << "File:" << filenameClassificationFunction
160             << "could not open." << std::endl;
161         exit(1);
162     }
163     for(int i=0;i<test.centers_number();i++){
164         ClassFunction.centers(i)=test.centers(i);
165     }
166     Vector Min(test.dimension(), DBL_MAX, "all");
167     Vector Max(test.dimension(), -DBL_MAX, "all");
168     for(int k=0;k<test.data_number();k++){
169         for(int ell=0;ell<test.dimension();ell++){
170             if(Min[ell]>test.data(k, ell)){
171                 Min[ell]=test.data(k, ell);
172             }
173             if(Max[ell]<test.data(k, ell)){
174                 Max[ell]=test.data(k, ell);
175             }
176         }
177     }
178     Vector Mid=0.5*(Max+Min);
179     Vector Width=Max-Min;
180     Min=Mid-Width;
181     Max=Mid+Width;
182
183     for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
184         for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
185 #ifdef VERBOSE

```



```

186         std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
187     #endif
188         ClassFunction.data(0,0)=x0;
189         ClassFunction.data(0,1)=x1;
190         while(1){
191             ClassFunction.revise_dissimilarities();
192             ClassFunction.revise_membership();
193             double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
194 #ifdef DIFF
195             std::cout << "diff_u:" << diff_u << std::endl;
196 #endif
197             if(diff_u<DIFF_FOR_STOP)break;
198         }
199         for(int ell=0;ell<ClassFunction.dimension();ell++){
200             ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
201         }
202         for(int i=0;i<ClassFunction.centers_number();i++){
203             ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
204         }
205         double max=0.0;
206         for(int i=0;i<ClassFunction.centers_number();i++){
207             if(max<ClassFunction.membership()[i][0]){
208                 max=ClassFunction.membership()[i][0];
209             }
210         }
211         ofs_classificationFunction << max << "\t";
212         ofs_classificationFunction << std::endl;
213     }
214     ofs_classificationFunction << std::endl;
215 }
216
217 #endif
218
219     return 0;
220 }

```

efcm_main_2d-Gaussian-2clusters.cxx

```

1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"efcm.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9
10  const int centers_number=2;

```

```
11
12 int main(void){
13     std::string filenameData("2d-Gaussian-2clusters.dat");
14 #ifdef CHECK_ANSWER
15     std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
16 #endif
17
18     std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
19     if(filenameDataDotPosition==std::string::npos){
20         std::cerr << "File:" << filenameData
21             << " needs \".\" and filename-extension." << std::endl;
22         exit(1);
23     }
24
25     std::ifstream ifs(filenameData);
26     if(!ifs){
27         std::cerr << "File:" << filenameData
28             << " could not open." << std::endl;
29         exit(1);
30     }
31     int data_number, data_dimension;
32     ifs >> data_number;
33     ifs >> data_dimension;
34
35     Efcmm test(data_dimension, data_number, centers_number, 10.0);
36
37     for(int cnt=0;cnt<data_number;cnt++){
38         for(int ell=0;ell<data_dimension;ell++){
39             ifs >> test.data(cnt, ell);
40         }
41     }
42
43     /**Initial Centers Setting**/
44     std::random_device rnd;
45     std::mt19937 mt(rnd());
46     std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
47     for(int i=0;i<test.centers_number();i++){
48         test.centers(i)=test.data()[randDataNumber(mt)];
49     }
50 #ifdef VERBOSE
51     std::cout << "v:\n" << test.centers() << std::endl;
52 #endif
53
54     test.iterates()=0;
55     while(1){
56         test.revise_dissimilarities();
57 #ifdef VERBOSE
58         std::cout << "d:\n" << test.dissimilarities() << std::endl;
59 #endif
60         test.revise_membership();
```

```

61  #ifdef VERBOSE
62      std::cout << "u:\n" << test.membership() << std::endl;
63  #endif
64      test.revise_centers();
65  #ifdef VERBOSE
66      std::cout << "v:\n" << test.centers() << std::endl;
67  #endif
68
69      double diff_u=max_norm(test.tmp_membership()-test.membership());
70      double diff_v=max_norm(test.tmp_centers()-test.centers());
71      double diff=diff_u+diff_v;
72  #ifdef DIFF
73      std::cout << "#diff:" << diff << "\t";
74      std::cout << "#diff_u:" << diff_u << "\t";
75      std::cout << "#diff_v:" << diff_v << "\n";
76  #endif
77      if(diff<DIFF_FOR_STOP)break;
78      if(test.iterates()>=MAX_ITERATES)break;
79      test.iterates()++;
80  }
81  #ifdef VERBOSE
82      std::cout << "v:\n" << test.centers() << std::endl;
83  #endif
84
85  #ifdef CHECK_ANSWER
86      test.set_crispMembership();
87
88      std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
89      if(!ifs_correctCrispMembership){
90          std::cerr << "File:" << filenameCorrectCrispMembership
91                  << " could not open." << std::endl;
92          exit(1);
93      }
94      for(int i=0;i<test.centers_number();i++){
95          for(int k=0;k<test.data_number();k++){
96              ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
97          }
98      }
99      test.set_contingencyTable();
100     std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
101     std::cout << "ARI:" << test.ARI() << std::endl;
102 #endif
103
104     std::string filenameResultMembership
105         =std::string("eFCM-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("-"
106         +filenameData.substr(0, filenameDataDotPosition)
107         +std::string(".result_membership");
108     std::ofstream ofs_membership(filenameResultMembership);
109     if(!ofs_membership){
110         std::cerr << "File:" << filenameResultMembership

```

```

111             << "could not open." << std::endl;
112     exit(1);
113 }
114
115     for(int k=0;k<test.data_number();k++){
116         for(int ell=0;ell<test.dimension();ell++){
117             ofs_membership << test.data()[k][ell] << "\t";
118         }
119         for(int i=0;i<test.centers_number();i++){
120             ofs_membership << test.membership()[i][k] << "\t";
121         }
122         ofs_membership << std::endl;
123     }
124     ofs_membership.close();
125
126     std::string filenameResultCenters
127         =std::string("eFCM-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("-"
128         +filenameData.substr(0, filenameDataDotPosition)
129         +std::string(".result_centers");
130     std::ofstream ofs_centers(filenameResultCenters);
131     if(!ofs_centers){
132         std::cerr << "File:" << filenameResultCenters
133             << "could not open." << std::endl;
134         exit(1);
135     }
136     for(int i=0;i<test.centers_number();i++){
137         for(int ell=0;ell<test.dimension();ell++){
138             ofs_centers << test.centers()[i][ell] << "\t";
139         }
140         ofs_centers << std::endl;
141     }
142     ofs_centers.close();
143
144     #ifdef CLASSIFICATION_FUNCTION
145         //Classification Function
146         if(test.dimension()>2){
147             std::cerr << "Dimension:" << test.dimension()
148                 << "is too high for classification function visualization."
149                 << std::endl;
150             exit(1);
151         }
152         Efcm ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierEm());
153         std::string filenameClassificationFunction
154             =std::string("eFCM-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("-"
155             +filenameData.substr(0, filenameDataDotPosition)
156             +std::string(".result_classificationFunction");
157         std::ofstream ofs_classificationFunction(filenameClassificationFunction);
158         if(!ofs_classificationFunction){
159             std::cerr << "File:" << filenameClassificationFunction
160                 << "could not open." << std::endl;

```

```

161     exit(1);
162 }
163 for(int i=0;i<test.centers_number();i++){
164     ClassFunction.centers(i)=test.centers(i);
165 }
166 Vector Min(test.dimension(), DBL_MAX, "all");
167 Vector Max(test.dimension(), -DBL_MAX, "all");
168 for(int k=0;k<test.data_number();k++){
169     for(int ell=0;ell<test.dimension();ell++){
170         if(Min[ell]>test.data(k, ell)){
171             Min[ell]=test.data(k, ell);
172         }
173         if(Max[ell]<test.data(k, ell)){
174             Max[ell]=test.data(k, ell);
175         }
176     }
177 }
178 Vector Mid=0.5*(Max+Min);
179 Vector Width=Max-Min;
180 Min=Mid-Width;
181 Max=Mid+Width;
182
183 for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
184     for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
185 #ifdef VERBOSE
186         std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
187 #endif
188         ClassFunction.data(0,0)=x0;
189         ClassFunction.data(0,1)=x1;
190         while(1){
191             ClassFunction.revise_dissimilarities();
192             ClassFunction.revise_membership();
193             double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
194 #ifdef DIFF
195             std::cout << "diff_u:" << diff_u << std::endl;
196 #endif
197             if(diff_u<DIFF_FOR_STOP)break;
198         }
199         for(int ell=0;ell<ClassFunction.dimension();ell++){
200             ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
201         }
202         for(int i=0;i<ClassFunction.centers_number();i++){
203             ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
204         }
205         double max=0.0;
206         for(int i=0;i<ClassFunction.centers_number();i++){
207             if(max<ClassFunction.membership()[i][0]){
208                 max=ClassFunction.membership()[i][0];
209             }
210         }

```

```
211         ofs_classificationFunction << max << "\t";
212         ofs_classificationFunction << std::endl;
213     }
214     ofs_classificationFunction << std::endl;
215 }
216
217 #endif
218
219     return 0;
220 }
```

qfcm_main_2d-Gaussian-2clusters.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"qfcm.h"
6
7
8  #define MAX_ITERATES 100000
9  #define DIFF_FOR_STOP 1.0E-10
10
11  const int centers_number=2;
12
13  int main(void){
14
15      double fuzzifierEm = 2.0;
16      double fuzzifierLambda = 0.2;
17
18      std::string filenameData("2d-Gaussian-2clusters.dat");
19  #ifdef CHECK_ANSWER
20      std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
21  #endif
22
23      std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
24      if(filenameDataDotPosition==std::string::npos){
25          std::cerr << "File:" << filenameData
26                  << " needs \".\" and filename-extension." << std::endl;
27          exit(1);
28      }
29
30      std::ifstream ifs(filenameData);
31      if(!ifs){
32          std::cerr << "File:" << filenameData
33                  << " could not open." << std::endl;
34          exit(1);
35      }
```

```
36     int data_number, data_dimension;
37     ifs >> data_number;
38     ifs >> data_dimension;
39
40     Qfcm test(data_dimension, data_number, centers_number, fuzzifierEm, fuzzifierLambda);
41
42     for(int cnt=0;cnt<data_number;cnt++){
43         for(int ell=0;ell<data_dimension;ell++){
44             ifs >> test.data(cnt, ell);
45         }
46     }
47
48     /**Initial Centers Setting***/
49     std::random_device rnd;
50     std::mt19937 mt(rnd());
51     std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
52     for(int i=0;i<test.centers_number();i++){
53         test.centers(i)=test.data()[randDataNumber(mt)];
54     }
55     #ifdef VERBOSE
56         std::cout << "v:\n" << test.centers() << std::endl;
57     #endif
58
59     test.iterates()=0;
60     while(1){
61         test.revise_dissimilarities();
62     #ifdef VERBOSE
63         std::cout << "d:\n" << test.dissimilarities() << std::endl;
64     #endif
65         test.revise_membership();
66     #ifdef VERBOSE
67         std::cout << "u:\n" << test.membership() << std::endl;
68     #endif
69         test.revise_centers();
70     #ifdef VERBOSE
71         std::cout << "v:\n" << test.centers() << std::endl;
72     #endif
73
74         double diff_u=max_norm(test.tmp_membership()-test.membership());
75         double diff_v=max_norm(test.tmp_centers()-test.centers());
76         double diff=diff_u+diff_v;
77     #ifdef DIFF
78         std::cout << "#diff:" << diff << "\t";
79         std::cout << "#diff_u:" << diff_u << "\t";
80         std::cout << "#diff_v:" << diff_v << "\n";
81     #endif
82         if(diff<DIFF_FOR_STOP)break;
83         if(test.iterates()>=MAX_ITERATES)break;
84         test.iterates()++;
85     }
```

```
86  #ifdef VERBOSE
87      std::cout << "v:\n" << test.centers() << std::endl;
88  #endif
89
90  #ifdef CHECK_ANSWER
91      test.set_crispMembership();
92
93      std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
94      if(!ifs_correctCrispMembership){
95          std::cerr << "File:" << filenameCorrectCrispMembership
96                  << " could not open." << std::endl;
97          exit(1);
98      }
99      for(int i=0;i<test.centers_number();i++){
100          for(int k=0;k<test.data_number();k++){
101              ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
102          }
103      }
104      test.set_contingencyTable();
105      std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
106      std::cout << "ARI:" << test.ARI() << std::endl;
107  #endif
108
109      std::string filenameResultMembership
110          =std::string("qFCM-Em")+std::to_string(test.fuzzifierEm())
111          +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
112          +std::string("-")
113          +filenameData.substr(0, filenameDataDotPosition)
114          +std::string(".result_membership");
115      std::ofstream ofs_membership(filenameResultMembership);
116      if(!ofs_membership){
117          std::cerr << "File:" << filenameResultMembership
118                  << " could not open." << std::endl;
119          exit(1);
120      }
121
122      for(int k=0;k<test.data_number();k++){
123          for(int ell=0;ell<test.dimension();ell++){
124              ofs_membership << test.data()[k][ell] << "\t";
125          }
126          for(int i=0;i<test.centers_number();i++){
127              ofs_membership << test.membership()[i][k] << "\t";
128          }
129          ofs_membership << std::endl;
130      }
131      ofs_membership.close();
132
133      std::string filenameResultCenters
134          =std::string("qFCM-Em")+std::to_string(test.fuzzifierEm())
135          +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
```



```

136     +std::string("-")
137     +filenameData.substr(0, filenameDataDotPosition)
138     +std::string(".result_centers");
139     std::ofstream ofs_centers(filenameResultCenters);
140     if(!ofs_centers){
141         std::cerr << "File:" << filenameResultCenters
142             << "could not open." << std::endl;
143         exit(1);
144     }
145     for(int i=0;i<test.centers_number();i++){
146         for(int ell=0;ell<test.dimension();ell++){
147             ofs_centers << test.centers()[i][ell] << "\t";
148         }
149         ofs_centers << std::endl;
150     }
151     ofs_centers.close();
152
153 #ifdef CLASSIFICATION_FUNCTION
154     //Classification Function
155     if(test.dimension()>2){
156         std::cerr << "Dimension:" << test.dimension()
157             << "is too high for classification function visualization."
158             << std::endl;
159         exit(1);
160     }
161     Qfcm ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierEm(), t
162     std::string filenameClassificationFunction
163         =std::string("qFCM-Em")+std::to_string(test.fuzzifierEm())
164         +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
165         +std::string("-")
166         +filenameData.substr(0, filenameDataDotPosition)
167         +std::string(".result_classificationFunction");
168     std::ofstream ofs_classificationFunction(filenameClassificationFunction);
169     if(!ofs_classificationFunction){
170         std::cerr << "File:" << filenameClassificationFunction
171             << "could not open." << std::endl;
172         exit(1);
173     }
174     for(int i=0;i<test.centers_number();i++){
175         ClassFunction.centers(i)=test.centers(i);
176     }
177     Vector Min(test.dimension(), DBL_MAX, "all");
178     Vector Max(test.dimension(), -DBL_MAX, "all");
179     for(int k=0;k<test.data_number();k++){
180         for(int ell=0;ell<test.dimension();ell++){
181             if(Min[ell]>test.data(k, ell)){
182                 Min[ell]=test.data(k, ell);
183             }
184             if(Max[ell]<test.data(k, ell)){
185                 Max[ell]=test.data(k, ell);

```

```
186     }
187     }
188     }
189     Vector Mid=0.5*(Max+Min);
190     Vector Width=Max-Min;
191     Min=Mid-Width;
192     Max=Mid+Width;
193
194     for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
195         for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
196 #ifdef VERBOSE
197         std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
198 #endif
199         ClassFunction.data(0,0)=x0;
200         ClassFunction.data(0,1)=x1;
201         while(1){
202             ClassFunction.revise_dissimilarities();
203             ClassFunction.revise_membership();
204             double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
205 #ifdef DIFF
206             std::cout << "diff_u:" << diff_u << std::endl;
207 #endif
208             if(diff_u<DIFF_FOR_STOP)break;
209         }
210         for(int ell=0;ell<ClassFunction.dimension();ell++){
211             ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
212         }
213         for(int i=0;i<ClassFunction.centers_number();i++){
214             ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
215         }
216         double max=0.0;
217         for(int i=0;i<ClassFunction.centers_number();i++){
218             if(max<ClassFunction.membership()[i][0]){
219                 max=ClassFunction.membership()[i][0];
220             }
221         }
222         ofs_classificationFunction << max << "\t";
223         ofs_classificationFunction << std::endl;
224     }
225     ofs_classificationFunction << std::endl;
226 }
227
228 #endif
229
230     return 0;
231 }
```

sfcma_main_2d-Gaussian-2clusters.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"sfcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9  #define EM 1.01
10
11  const int centers_number=2;
12
13  int main(void){
14      double Em=EM;
15
16      std::string filenameData("2d-Gaussian-2clusters.dat");
17  #ifdef CHECK_ANSWER
18      std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
19  #endif
20
21      std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
22      if(filenameDataDotPosition==std::string::npos){
23          std::cerr << "File:" << filenameData
24                  << " needs \".\" and filename-extension." << std::endl;
25          exit(1);
26      }
27
28      std::ifstream ifs(filenameData);
29      if(!ifs){
30          std::cerr << "File:" << filenameData
31                  << " could not open." << std::endl;
32          exit(1);
33      }
34      int data_number, data_dimension;
35      ifs >> data_number;
36      ifs >> data_dimension;
37
38      Sfcma test(data_dimension, data_number, centers_number, Em);
39
40      for(int cnt=0;cnt<data_number;cnt++){
41          for(int ell=0;ell<data_dimension;ell++){
42              ifs >> test.data(cnt, ell);
43          }
44      }
45
46      /**Initial Centers Setting**/
```

```
47     std::random_device rnd;
48     std::mt19937 mt(rnd());
49     std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
50     for(int i=0;i<test.centers_number();i++){
51         test.centers(i)=test.data()[randDataNumber(mt)];
52         test.alpha(i)=1.0/centers_number;
53     }
54     test.iterates()=0;
55     while(1){
56         test.revise_dissimilarities();
57 #ifdef VERBOSE
58         std::cout << "d:\n" << test.dissimilarities() << std::endl;
59 #endif
60         test.revise_membership();
61 #ifdef VERBOSE
62         std::cout << "u:\n" << test.membership() << std::endl;
63 #endif
64         test.revise_centers();
65 #ifdef VERBOSE
66         std::cout << "v:\n" << test.centers() << std::endl;
67 #endif
68         test.revise_alpha();
69 #ifdef VERBOSE
70         std::cout << "a:\n" << test.alpha() << std::endl;
71 #endif
72
73         double diff_u=max_norm(test.tmp_membership()-test.membership());
74         double diff_v=max_norm(test.tmp_centers()-test.centers());
75         double diff_a=max_norm(test.tmp_alpha()-test.alpha());
76         double diff=diff_u+diff_v+diff_a;
77 #ifdef DIFF
78         std::cout << "#diff:" << diff << "\t";
79         std::cout << "#diff_u:" << diff_u << "\t";
80         std::cout << "#diff_v:" << diff_v << "\t";
81         std::cout << "#diff_a:" << diff_a << "\n";
82 #endif
83         if(diff<DIFF_FOR_STOP)break;
84         if(test.iterates()>=MAX_ITERATES)break;
85         test.iterates()++;
86     }
87 #ifdef VERBOSE
88     std::cout << "v:\n" << test.centers() << std::endl;
89 #endif
90
91 #ifdef CHECK_ANSWER
92     test.set_crispMembership();
93
94     std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
95     if(!ifs_correctCrispMembership){
96         std::cerr << "File:" << filenameCorrectCrispMembership
```

```
97         << " could not open." << std::endl;
98     exit(1);
99 }
100 for(int i=0;i<test.centers_number();i++){
101     for(int k=0;k<test.data_number();k++){
102         ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
103     }
104 }
105 test.set_contingencyTable();
106 std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
107 std::cout << "ARI:" << test.ARI() << std::endl;
108 #endif
109
110 std::string filenameResultMembership
111     =std::string("sFCMA-Em")
112     +std::to_string(test.fuzzifierEm())+std::string("-")
113     +filenameData.substr(0, filenameDataDotPosition)
114     +std::string(".result_membership");
115 std::ofstream ofs_membership(filenameResultMembership);
116 if(!ofs_membership){
117     std::cerr << "File:" << filenameResultMembership
118         << "could not open." << std::endl;
119     exit(1);
120 }
121
122 for(int k=0;k<test.data_number();k++){
123     for(int ell=0;ell<test.dimension();ell++){
124         ofs_membership << test.data()[k][ell] << "\t";
125     }
126     for(int i=0;i<test.centers_number();i++){
127         ofs_membership << test.membership()[i][k] << "\t";
128     }
129     ofs_membership << std::endl;
130 }
131 ofs_membership.close();
132
133 std::string filenameResultCenters
134     =std::string("sFCMA-Em")
135     +std::to_string(test.fuzzifierEm())+std::string("-")
136     +filenameData.substr(0, filenameDataDotPosition)
137     +std::string(".result_centers");
138 std::ofstream ofs_centers(filenameResultCenters);
139 if(!ofs_centers){
140     std::cerr << "File:" << filenameResultCenters
141         << "could not open." << std::endl;
142     exit(1);
143 }
144 for(int i=0;i<test.centers_number();i++){
145     for(int ell=0;ell<test.dimension();ell++){
146         ofs_centers << test.centers()[i][ell] << "\t";
```

```
147     }
148     ofs_centers << std::endl;
149 }
150 ofs_centers.close();
151
152 #ifdef CLASSIFICATION_FUNCTION
153 //Classification Function
154 if(test.dimension()>2){
155     std::cerr << "Dimension:" << test.dimension()
156               << "is too high for classification function visualization."
157               << std::endl;
158     exit(1);
159 }
160 Sfcma ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierEm())
161 std::string filenameClassificationFunction
162     =std::string("sFCMA-Em")
163     +std::to_string(test.fuzzifierEm())+std::string("-")
164     +filenameData.substr(0, filenameDataDotPosition)
165     +std::string(".result_classificationFunction");
166 std::ofstream ofs_classificationFunction(filenameClassificationFunction);
167 if(!ofs_classificationFunction){
168     std::cerr << "File:" << filenameClassificationFunction
169               << "could not open." << std::endl;
170     exit(1);
171 }
172 for(int i=0;i<test.centers_number();i++){
173     ClassFunction.centers(i)=test.centers(i);
174 }
175 Vector Min(test.dimension(), DBL_MAX, "all");
176 Vector Max(test.dimension(), -DBL_MAX, "all");
177 for(int k=0;k<test.data_number();k++){
178     for(int ell=0;ell<test.dimension();ell++){
179         if(Min[ell]>test.data(k, ell)){
180             Min[ell]=test.data(k, ell);
181         }
182         if(Max[ell]<test.data(k, ell)){
183             Max[ell]=test.data(k, ell);
184         }
185     }
186 }
187 Vector Mid=0.5*(Max+Min);
188 Vector Width=Max-Min;
189 Min=Mid-Width;
190 Max=Mid+Width;
191
192 for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
193     for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
194 #ifdef VERBOSE
195         std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
196 #endif
```

```

197     ClassFunction.data(0,0)=x0;
198     ClassFunction.data(0,1)=x1;
199     while(1){
200         ClassFunction.revise_dissimilarities();
201         ClassFunction.revise_membership();
202         double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
203 #ifdef DIFF
204         std::cout << "diff_u:" << diff_u << std::endl;
205 #endif
206         if(diff_u<DIFF_FOR_STOP)break;
207     }
208     for(int ell=0;ell<ClassFunction.dimension();ell++){
209         ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
210     }
211     for(int i=0;i<ClassFunction.centers_number();i++){
212         ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
213     }
214     double max=0.0;
215     for(int i=0;i<ClassFunction.centers_number();i++){
216         if(max<ClassFunction.membership()[i][0]){
217             max=ClassFunction.membership()[i][0];
218         }
219     }
220     ofs_classificationFunction << max << "\t";
221     ofs_classificationFunction << std::endl;
222 }
223 ofs_classificationFunction << std::endl;
224 }
225
226 #endif
227
228     return 0;
229 }

```

efcma_main_2d-Gaussian-2clusters.cxx

```

1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"efcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9  #define LAMBDA 1
10
11  const int centers_number=2;
12

```

```
13  int main(void){
14      double Lambda = LAMBDA;
15
16      std::string filenameData("2d-Gaussian-2clusters.dat");
17  #ifdef CHECK_ANSWER
18      std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
19  #endif
20
21      std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
22      if(filenameDataDotPosition==std::string::npos){
23          std::cerr << "File:" << filenameData
24                  << " needs \".\" and filename-extension." << std::endl;
25          exit(1);
26      }
27
28      std::ifstream ifs(filenameData);
29      if(!ifs){
30          std::cerr << "File:" << filenameData
31                  << " could not open." << std::endl;
32          exit(1);
33      }
34      int data_number, data_dimension;
35      ifs >> data_number;
36      ifs >> data_dimension;
37
38      Efcma test(data_dimension, data_number, centers_number, Lambda);
39
40      for(int cnt=0;cnt<data_number;cnt++){
41          for(int ell=0;ell<data_dimension;ell++){
42              ifs >> test.data(cnt, ell);
43          }
44      }
45
46      /**Initial Centers Setting**/
47      std::random_device rnd;
48      std::mt19937 mt(rnd());
49      std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
50      for(int i=0;i<test.centers_number();i++){
51          test.centers(i)=test.data()[randDataNumber(mt)];
52          test.alpha(i)=1.0/centers_number;
53      }
54
55      test.iterates()=0;
56      while(1){
57          test.revise_dissimilarities();
58  #ifdef VERBOSE
59          std::cout << "d:\n" << test.dissimilarities() << std::endl;
60  #endif
61          test.revise_membership();
62  #ifdef VERBOSE
```



```

63         std::cout << "u:\n" << test.membership() << std::endl;
64     #endif
65     test.revise_centers();
66     #ifdef VERBOSE
67         std::cout << "v:\n" << test.centers() << std::endl;
68     #endif
69     test.revise_alpha();
70     #ifdef VERBOSE
71         std::cout << "a:\n" << test.alpha() << std::endl;
72     #endif
73     double diff_u=max_norm(test.tmp_membership()-test.membership());
74     double diff_v=max_norm(test.tmp_centers()-test.centers());
75     double diff_a=max_norm(test.tmp_alpha()-test.alpha());
76     double diff=diff_u+diff_v+diff_a;
77     #ifdef DIFF
78         std::cout << "#diff:" << diff << "\t";
79         std::cout << "#diff_u:" << diff_u << "\t";
80         std::cout << "#diff_v:" << diff_v << "\n";
81     #endif
82     if(diff<DIFF_FOR_STOP)break;
83     if(test.iterates()>=MAX_ITERATES)break;
84     test.iterates()++;
85 }
86 #ifdef VERBOSE
87     std::cout << "v:\n" << test.centers() << std::endl;
88 #endif
89
90 #ifdef CHECK_ANSWER
91     test.set_crispMembership();
92     std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
93     if(!ifs_correctCrispMembership){
94         std::cerr << "File:" << filenameCorrectCrispMembership
95             << " could not open." << std::endl;
96         exit(1);
97     }
98     for(int i=0;i<test.centers_number();i++){
99         for(int k=0;k<test.data_number();k++){
100             ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
101         }
102     }
103     test.set_contingencyTable();
104     std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
105     std::cout << "ARI:" << test.ARI() << std::endl;
106 #endif
107
108     std::string filenameResultMembership
109         =std::string("eFCMA-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("
110         +filenameData.substr(0, filenameDataDotPosition)
111         +std::string(".result_membership");
112     std::ofstream ofs_membership(filenameResultMembership);

```

```

113     if(!ofs_membership){
114         std::cerr << "File:" << filenameResultMembership
115             << "could not open." << std::endl;
116         exit(1);
117     }
118
119     for(int k=0;k<test.data_number();k++){
120         for(int ell=0;ell<test.dimension();ell++){
121             ofs_membership << test.data()[k][ell] << "\t";
122         }
123         for(int i=0;i<test.centers_number();i++){
124             ofs_membership << test.membership()[i][k] << "\t";
125         }
126         ofs_membership << std::endl;
127     }
128     ofs_membership.close();
129
130     std::string filenameResultCenters
131         =std::string("eFCMA-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("
132         +filenameData.substr(0, filenameDataDotPosition)
133         +std::string(".result_centers");
134     std::ofstream ofs_centers(filenameResultCenters);
135     if(!ofs_centers){
136         std::cerr << "File:" << filenameResultCenters
137             << "could not open." << std::endl;
138         exit(1);
139     }
140     for(int i=0;i<test.centers_number();i++){
141         for(int ell=0;ell<test.dimension();ell++){
142             ofs_centers << test.centers()[i][ell] << "\t";
143         }
144         ofs_centers << std::endl;
145     }
146     ofs_centers.close();
147
148 #ifdef CLASSIFICATION_FUNCTION
149     //Classification Function
150     if(test.dimension()>2){
151         std::cerr << "Dimension:" << test.dimension()
152             << "is too high for classification function visualization."
153             << std::endl;
154         exit(1);
155     }
156     Efcma ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierLambda
157     std::string filenameClassificationFunction
158         =std::string("eFCMA-Lambda")+std::to_string(test.fuzzifierLambda())+std::string("
159         +filenameData.substr(0, filenameDataDotPosition)
160         +std::string(".result_classificationFunction");
161     std::ofstream ofs_classificationFunction(filenameClassificationFunction);
162     if(!ofs_classificationFunction){

```

```

163         std::cerr << "File:" << filenameClassificationFunction
164                 << "could not open." << std::endl;
165         exit(1);
166     }
167     for(int i=0;i<test.centers_number();i++){
168         ClassFunction.centers(i)=test.centers(i);
169     }
170     Vector Min(test.dimension(), DBL_MAX, "all");
171     Vector Max(test.dimension(), -DBL_MAX, "all");
172     for(int k=0;k<test.data_number();k++){
173         for(int ell=0;ell<test.dimension();ell++){
174             if(Min[ell]>test.data(k, ell)){
175                 Min[ell]=test.data(k, ell);
176             }
177             if(Max[ell]<test.data(k, ell)){
178                 Max[ell]=test.data(k, ell);
179             }
180         }
181     }
182     Vector Mid=0.5*(Max+Min);
183     Vector Width=Max-Min;
184     Min=Mid-Width;
185     Max=Mid+Width;
186
187     for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
188         for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
189 #ifdef VERBOSE
190             std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
191 #endif
192             ClassFunction.data(0,0)=x0;
193             ClassFunction.data(0,1)=x1;
194             while(1){
195                 ClassFunction.revise_dissimilarities();
196                 ClassFunction.revise_membership();
197                 double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
198 #ifdef DIFF
199                 std::cout << "diff_u:" << diff_u << std::endl;
200 #endif
201                 if(diff_u<DIFF_FOR_STOP)break;
202             }
203             for(int ell=0;ell<ClassFunction.dimension();ell++){
204                 ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
205             }
206             for(int i=0;i<ClassFunction.centers_number();i++){
207                 ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
208             }
209             double max=0.0;
210             for(int i=0;i<ClassFunction.centers_number();i++){
211                 if(max<ClassFunction.membership()[i][0]){
212                     max=ClassFunction.membership()[i][0];

```

```
213     }
214     }
215     ofs_classificationFunction << max << "\t";
216     ofs_classificationFunction << std::endl;
217     }
218     ofs_classificationFunction << std::endl;
219     }
220
221 #endif
222
223     return 0;
224 }
```

qfcma_main_2d-Gaussian-2clusters.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"qfcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9  #define EM 2.5
10 #define LAMBDA 1
11
12 const int centers_number=2;
13
14 int main(void){
15     double Em=EM;
16     double Lambda=LAMBDA;
17
18     std::string filenameData("2d-Gaussian-2clusters.dat");
19 #ifdef CHECK_ANSWER
20     std::string filenameCorrectCrispMembership("2d-Gaussian-2clusters.correctCrispMembe
21 #endif
22
23     std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
24     if(filenameDataDotPosition==std::string::npos){
25         std::cerr << "File:" << filenameData
26                 << " needs \".\" and filename-extention." << std::endl;
27         exit(1);
28     }
29
30     std::ifstream ifs(filenameData);
31     if(!ifs){
32         std::cerr << "File:" << filenameData
33                 << " could not open." << std::endl;
```

```
34     exit(1);
35 }
36 int data_number, data_dimension;
37 ifs >> data_number;
38 ifs >> data_dimension;
39
40 Qfcma test(data_dimension, data_number, centers_number, Em, Lambda);
41
42 for(int cnt=0;cnt<data_number;cnt++){
43     for(int ell=0;ell<data_dimension;ell++){
44         ifs >> test.data(cnt, ell);
45     }
46 }
47
48 /**Initial Centers Setting**/
49 std::random_device rnd;
50 std::mt19937 mt(rnd());
51 std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
52 for(int i=0;i<test.centers_number();i++){
53     test.centers(i)=test.data()[randDataNumber(mt)];
54     test.alpha(i)=1.0/centers_number;
55 }
56 test.iterates()=0;
57 while(1){
58     test.revise_dissimilarities();
59 #ifdef VERBOSE
60     std::cout << "d:\n" << test.dissimilarities() << std::endl;
61 #endif
62     test.revise_membership();
63 #ifdef VERBOSE
64     std::cout << "u:\n" << test.membership() << std::endl;
65 #endif
66     test.revise_centers();
67 #ifdef VERBOSE
68     std::cout << "v:\n" << test.centers() << std::endl;
69 #endif
70     test.revise_alpha();
71 #ifdef VERBOSE
72     std::cout << "a:\n" << test.alpha() << std::endl;
73 #endif
74
75     double diff_u=max_norm(test.tmp_membership()-test.membership());
76     double diff_v=max_norm(test.tmp_centers()-test.centers());
77     double diff_a=max_norm(test.tmp_alpha()-test.alpha());
78     double diff=diff_u+diff_v+diff_a;
79 #ifdef DIFF
80     std::cout << "#diff:" << diff << "\t";
81     std::cout << "#diff_u:" << diff_u << "\t";
82     std::cout << "#diff_v:" << diff_v << "\t";
83     std::cout << "#diff_a:" << diff_a << "\n";
```

```

84  #endif
85
86      if(diff<DIFF_FOR_STOP)break;
87      if(test.iterates()>=MAX_ITERATES)break;
88      test.iterates()++;
89  }
90  #ifdef VERBOSE
91      std::cout << "v:\n" << test.centers() << std::endl;
92  #endif
93
94  #ifdef CHECK_ANSWER
95      test.set_crispMembership();
96      std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
97      if(!ifs_correctCrispMembership){
98          std::cerr << "File:" << filenameCorrectCrispMembership
99                  << " could not open." << std::endl;
100         exit(1);
101     }
102     for(int i=0;i<test.centers_number();i++){
103         for(int k=0;k<test.data_number();k++){
104             ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
105         }
106     }
107     test.set_contingencyTable();
108     std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
109     std::cout << "ARI:" << test.ARI() << std::endl;
110 #endif
111
112     std::string filenameResultMembership
113         =std::string("qFCMA-Em")+std::to_string(test.fuzzifierEm())
114         +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
115         +std::string("-")
116         +filenameData.substr(0, filenameDataDotPosition)
117         +std::string(".result_membership");
118     std::ofstream ofs_membership(filenameResultMembership);
119     if(!ofs_membership){
120         std::cerr << "File:" << filenameResultMembership
121                 << "could not open." << std::endl;
122         exit(1);
123     }
124
125     for(int k=0;k<test.data_number();k++){
126         for(int ell=0;ell<test.dimension();ell++){
127             ofs_membership << test.data()[k][ell] << "\t";
128         }
129         for(int i=0;i<test.centers_number();i++){
130             ofs_membership << test.membership()[i][k] << "\t";
131         }
132         ofs_membership << std::endl;
133     }

```

```
134     ofs_membership.close();
135
136     std::string filenameResultCenters
137         =std::string("qFCMA-Em")+std::to_string(test.fuzzifierEm())
138         +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
139         +std::string("-")
140         +filenameData.substr(0, filenameDataDotPosition)
141         +std::string(".result_centers");
142     std::ofstream ofs_centers(filenameResultCenters);
143     if(!ofs_centers){
144         std::cerr << "File:" << filenameResultCenters
145             << "could not open." << std::endl;
146         exit(1);
147     }
148     for(int i=0;i<test.centers_number();i++){
149         for(int ell=0;ell<test.dimension();ell++){
150             ofs_centers << test.centers()[i][ell] << "\t";
151         }
152         ofs_centers << std::endl;
153     }
154     ofs_centers.close();
155
156     #ifdef CLASSIFICATION_FUNCTION
157         //Classification Function
158         if(test.dimension()>2){
159             std::cerr << "Dimension:" << test.dimension()
160                 << "is too high for classification function visualization."
161                 << std::endl;
162             exit(1);
163         }
164         Qfcma ClassFunction(test.dimension(), 1, test.centers_number(), test.fuzzifierEm(),
165             std::string filenameClassificationFunction
166                 =std::string("qFCMA-Em")+std::to_string(test.fuzzifierEm())
167                 +std::string("-Lambda")+std::to_string(test.fuzzifierLambda())
168                 +std::string("-")
169                 +filenameData.substr(0, filenameDataDotPosition)
170                 +std::string(".result_classificationFunction");
171             std::ofstream ofs_classificationFunction(filenameClassificationFunction);
172             if(!ofs_classificationFunction){
173                 std::cerr << "File:" << filenameClassificationFunction
174                     << "could not open." << std::endl;
175                 exit(1);
176             }
177             for(int i=0;i<test.centers_number();i++){
178                 ClassFunction.centers(i)=test.centers(i);
179             }
180             Vector Min(test.dimension(), DBL_MAX, "all");
181             Vector Max(test.dimension(), -DBL_MAX, "all");
182             for(int k=0;k<test.data_number();k++){
183                 for(int ell=0;ell<test.dimension();ell++){
```

```
184         if(Min[ell]>test.data(k, ell)){
185             Min[ell]=test.data(k, ell);
186         }
187         if(Max[ell]<test.data(k, ell)){
188             Max[ell]=test.data(k, ell);
189         }
190     }
191 }
192 Vector Mid=0.5*(Max+Min);
193 Vector Width=Max-Min;
194 Min=Mid-Width;
195 Max=Mid+Width;
196
197 for(double x0=Min[0];x0<=Max[0];x0+=Width[0]/10.0){
198     for(double x1=Min[1];x1<=Max[1];x1+=Width[1]/10.0){
199 #ifdef VERBOSE
200     std::cout << "x0:" << x0 << "\t" << "x1:" << x1 << std::endl;
201 #endif
202     ClassFunction.data(0,0)=x0;
203     ClassFunction.data(0,1)=x1;
204     while(1){
205         ClassFunction.revise_dissimilarities();
206         ClassFunction.revise_membership();
207         double diff_u=frobenius_norm(ClassFunction.tmp_membership()-ClassFunction.mem
208 #ifdef DIFF
209         std::cout << "diff_u:" << diff_u << std::endl;
210 #endif
211         if(diff_u<DIFF_FOR_STOP)break;
212     }
213     for(int ell=0;ell<ClassFunction.dimension();ell++){
214         ofs_classificationFunction << ClassFunction.data()[0][ell] << "\t";
215     }
216     for(int i=0;i<ClassFunction.centers_number();i++){
217         ofs_classificationFunction << ClassFunction.membership()[i][0] << "\t";
218     }
219     double max=0.0;
220     for(int i=0;i<ClassFunction.centers_number();i++){
221         if(max<ClassFunction.membership()[i][0]){
222             max=ClassFunction.membership()[i][0];
223         }
224     }
225     ofs_classificationFunction << max << "\t";
226     ofs_classificationFunction << std::endl;
227 }
228 ofs_classificationFunction << std::endl;
229 }
230
231 #endif
232
233     return 0;
```


234 }

sfcmain_user_knowledge.cxx

efcmain_user_knowledge.cxx

qfcmmain_user_knowledge.cxx

sfcma_main_user_knowledge.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"sfcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9
10 const int centers_number=2;
11
12 int main(void){
13     double max_ARI_Em, max_ARI;
14
15     std::ofstream outputfile("sfcma_user_knowledge_ARI.txt");
16
17     std::string filenameData("user_knowledge.dat");
18     std::string filenameCorrectCrispMembership("user_knowledge.correctCrispMembership");
19
20     std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
21     if(filenameDataDotPosition==std::string::npos){
22         std::cerr << "File:" << filenameData
23             << " needs \"\\.\" and filename-extention." << std::endl;
24         exit(1);
25     }
```

```
26     for(double Em=3.0;Em>1.0;Em-=0.1){
27         std::ifstream ifs(filenameData);
28         if(!ifs){
29             std::cerr << "File:" << filenameData
30                 << " could not open." << std::endl;
31             exit(1);
32         }
33         int data_number, data_dimension;
34         ifs >> data_number;
35         ifs >> data_dimension;
36
37         Sfcma test(data_dimension, data_number, centers_number, Em);
38
39         for(int cnt=0;cnt<data_number;cnt++){
40             for(int ell=0;ell<data_dimension;ell++){
41                 ifs >> test.data(cnt, ell);
42             }
43         }
44
45         /**Initial Centers Setting**/
46         std::random_device rnd;
47         std::mt19937 mt(rnd());
48         std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
49         std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
50         if(!ifs_correctCrispMembership){
51             std::cerr << "File:" << filenameCorrectCrispMembership
52                 << " could not open." << std::endl;
53             exit(1);
54         }
55         for(int i=0;i<test.centers_number();i++){
56             for(int k=0;k<test.data_number();k++){
57                 ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
58             }
59         }
60         for(int i=0;i<test.centers_number();i++){
61             test.centers(i)=test.data()[randDataNumber(mt)];
62             test.alpha(i)=1.0/centers_number;
63             for(int k=0;k<test.data_number();k++){
64                 test.membership(i,k)=test.correctCrispMembership(i, k);
65             }
66         }
67
68         test.iterates()=0;
69         while(1){
70             test.revise_centers();
71 #ifdef VERBOSE
72             std::cout << "v:\n" << test.centers() << std::endl;
73 #endif
74             test.revise_dissimilarities();
75 #ifdef VERBOSE
```

```

76         std::cout << "d:\n" << test.dissimilarities() << std::endl;
77     #endif
78         test.revise_membership();
79     #ifdef VERBOSE
80         std::cout << "u:\n" << test.membership() << std::endl;
81     #endif
82         test.revise_alpha();
83     #ifdef VERBOSE
84         std::cout << "a:\n" << test.alpha() << std::endl;
85     #endif
86
87         double diff_u=max_norm(test.tmp_membership()-test.membership());
88         double diff_v=max_norm(test.tmp_centers()-test.centers());
89         double diff_a=max_norm(test.tmp_alpha()-test.alpha());
90         double diff=diff_u+diff_v+diff_a;
91     #ifdef DIFF
92         std::cout << "#diff:" << diff << "\t";
93         std::cout << "#diff_u:" << diff_u << "\t";
94         std::cout << "#diff_v:" << diff_v << "\t";
95         std::cout << "#diff_a:" << diff_a << "\n";
96     #endif
97         if(diff<DIFF_FOR_STOP)break;
98         if(test.iterates())>=MAX_ITERATES)break;
99         test.iterates()++;
100     }
101     #ifdef VERBOSE
102         std::cout << "v:\n" << test.centers() << std::endl;
103     #endif
104
105     #ifdef CHECK_ANSWER
106         test.set_crispMembership();
107         test.set_contingencyTable();
108         //std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
109         std::cout << "Em:" << Em << "\tARI:" << test.ARI() << std::endl;
110         outputfile<<Em<<"\t";
111         outputfile<<test.ARI()<<"\t";
112         outputfile<<"\n";
113     #endif
114
115         if(max_ARI<test.ARI()){
116             max_ARI_Em=Em;
117             max_ARI=test.ARI();
118         }
119     }
120
121     outputfile.close();
122
123     std::cout << "max_ARI_Em:" << max_ARI_Em << std::endl;
124     std::cout << "max_ARI:" << max_ARI << std::endl;
125

```

```
126     return 0;
127 }
```

efcma_main_user_knowledge.cxx

```
1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"efcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-10
9
10 const int centers_number=4;
11
12
13 int main(void){
14
15     double max_ARI_Lambda, max_ARI;
16
17     std::ofstream outputfile("efcma_user_knowledge_ARI.txt");
18
19     std::string filenameData("user_knowledge.dat");
20     std::string filenameCorrectCrispMembership("user_knowledge.correctCrispMembership");
21
22     std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
23     if(filenameDataDotPosition==std::string::npos){
24         std::cerr << "File:" << filenameData
25                 << " needs \"\\.\" and filename-extention." << std::endl;
26         exit(1);
27     }
28
29     for(double Lambda=1;Lambda<=100;Lambda+=1){
30         std::ifstream ifs(filenameData);
31         if(!ifs){
32             std::cerr << "File:" << filenameData
33                     << " could not open." << std::endl;
34             exit(1);
35         }
36         int data_number, data_dimension;
37         ifs >> data_number;
38         ifs >> data_dimension;
39
40         Efcma test(data_dimension, data_number, centers_number, Lambda);
41
42         for(int cnt=0;cnt<data_number;cnt++){
43             for(int ell=0;ell<data_dimension;ell++){
```

```
44         ifs >> test.data(cnt, ell);
45     }
46 }
47
48 /**Initial Centers Setting**/
49 std::random_device rnd;
50 std::mt19937 mt(rnd());
51 std::uniform_int_distribution<> randDataNumber(0, test.data_number()-1);
52 std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
53 if(!ifs_correctCrispMembership){
54     std::cerr << "File:" << filenameCorrectCrispMembership
55               << " could not open." << std::endl;
56     exit(1);
57 }
58
59 for(int i=0; i<test.centers_number(); i++){
60     for(int k=0; k<test.data_number(); k++){
61         ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
62     }
63 }
64 for(int i=0; i<test.centers_number(); i++){
65     test.centers(i)=test.data()[randDataNumber(mt)];
66     test.alpha(i)=1.0/centers_number;
67     for(int k=0; k<test.data_number(); k++){
68         test.membership(i,k) = test.correctCrispMembership(i,k);
69     }
70 }
71
72 test.iterates()=0;
73 while(1){
74     test.revise_centers();
75 #ifdef VERBOSE
76     std::cout << "v:\n" << test.centers() << std::endl;
77 #endif
78     test.revise_dissimilarities();
79 #ifdef VERBOSE
80     std::cout << "d:\n" << test.dissimilarities() << std::endl;
81 #endif
82     test.revise_membership();
83 #ifdef VERBOSE
84     std::cout << "u:\n" << test.membership() << std::endl;
85 #endif
86     test.revise_alpha();
87 #ifdef VERBOSE
88     std::cout << "a:\n" << test.alpha() << std::endl;
89 #endif
90     double diff_u=max_norm(test.tmp_membership()-test.membership());
91     double diff_v=max_norm(test.tmp_centers()-test.centers());
92     double diff_a=max_norm(test.tmp_alpha()-test.alpha());
93     double diff=diff_u+diff_v+diff_a;
```

```

94  #ifdef DIFF
95      std::cout << "#diff:" << diff << "\t";
96      std::cout << "#diff_u:" << diff_u << "\t";
97      std::cout << "#diff_v:" << diff_v << "\n";
98  #endif
99      if(diff<DIFF_FOR_STOP)break;
100      if(test.iterates()>=MAX_ITERATES)break;
101      test.iterates()++;
102  }
103  #ifdef VERBOSE
104      std::cout << "v:\n" << test.centers() << std::endl;
105  #endif
106
107  #ifdef CHECK_ANSWER
108      test.set_crispMembership();
109      test.set_contingencyTable();
110      //std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
111      std::cout << "Lambda:" << Lambda << "\tARI:" << test.ARI() << std::endl;
112      outputfile<<Lambda<<"\t";
113      outputfile<<test.ARI()<<"\t";
114      outputfile<<"\n";
115  #endif
116
117      if(max_ARI<test.ARI()){
118          max_ARI_Lambda=Lambda;
119          max_ARI=test.ARI();
120      }
121  }
122
123      outputfile.close();
124
125      std::cout << "max_ARI_Lambda:" << max_ARI_Lambda << std::endl;
126      std::cout << "max_ARI:" << max_ARI << std::endl;
127
128      return 0;
129  }

```

qfcma_main_user_knowledge.cxx

```

1  #include<iostream>
2  #include<fstream>
3  #include<cstdlib>
4  #include<random>
5  #include"qfcma.h"
6
7  #define MAX_ITERATES 100000
8  #define DIFF_FOR_STOP 1.0E-2
9

```

```
10  const int centers_number=4;
11
12  int main(void){
13
14      double max_ARI_Em, max_ARI_Lambda, max_ARI;
15
16      std::ofstream outputfile("qfcma_user_knowledge_ARI.txt");
17
18      std::string filenameData("user_knowledge.dat");
19      std::string filenameCorrectCrispMembership("user_knowledge.correctCrispMembership");
20
21      std::string::size_type filenameDataDotPosition=filenameData.find_last_of(".");
22      if(filenameDataDotPosition==std::string::npos){
23          std::cerr << "File:" << filenameData
24                  << " needs \".\" and filename-extention." << std::endl;
25          exit(1);
26      }
27
28      for(double Lambda=100;Lambda>0;Lambda-=10){
29          for(double Em=3.0;Em>1.0;Em-=0.1){
30
31              std::ifstream ifs(filenameData);
32              if(!ifs){
33                  std::cerr << "File:" << filenameData
34                          << " could not open." << std::endl;
35                  exit(1);
36              }
37              int data_number, data_dimension;
38              ifs >> data_number;
39              ifs >> data_dimension;
40
41              Qfcma test(data_dimension, data_number, centers_number, Em, Lambda);
42
43              for(int cnt=0;cnt<data_number;cnt++){
44                  for(int ell=0;ell<data_dimension;ell++){
45                      ifs >> test.data(cnt, ell);
46                  }
47              }
48
49              /**Initial Centers Setting***/
50              std::random_device rnd;
51              std::mt19937 mt(rnd());
52              std::uniform_int_distribution<> randDataNumber(0,test.data_number()-1);
53              std::ifstream ifs_correctCrispMembership(filenameCorrectCrispMembership);
54              if(!ifs_correctCrispMembership){
55                  std::cerr << "File:" << filenameCorrectCrispMembership
56                          << " could not open." << std::endl;
57                  exit(1);
58              }
59              for(int i=0;i<test.centers_number();i++){
```

```
60         for(int k=0;k<test.data_number();k++){
61             ifs_correctCrispMembership >> test.correctCrispMembership(i, k);
62         }
63     }
64     for(int i=0;i<test.centers_number();i++){
65         //test.centers(i)=test.data()[randDataNumber(mt)];
66         test.alpha(i)=1.0/centers_number;
67         for(int k=0;k<test.data_number();k++){
68             test.membership(i,k)=test.correctCrispMembership(i, k);
69         }
70     }
71
72     test.iterates()=0;
73     while(1){
74         test.revise_centers();
75 #ifdef VERBOSE
76         std::cout << "v:\n" << test.centers() << std::endl;
77 #endif
78         test.revise_dissimilarities();
79 #ifdef VERBOSE
80         std::cout << "d:\n" << test.dissimilarities() << std::endl;
81 #endif
82         test.revise_membership();
83 #ifdef VERBOSE
84         std::cout << "u:\n" << test.membership() << std::endl;
85 #endif
86         test.revise_alpha();
87 #ifdef VERBOSE
88         std::cout << "a:\n" << test.alpha() << std::endl;
89 #endif
90
91         double diff_u=max_norm(test.tmp_membership()-test.membership());
92         double diff_v=max_norm(test.tmp_centers()-test.centers());
93         double diff_a=max_norm(test.tmp_alpha()-test.alpha());
94         double diff=diff_u+diff_v+diff_a;
95 #ifdef DIFF
96         std::cout << "#diff:" << diff << "\t";
97         std::cout << "#diff_u:" << diff_u << "\t";
98         std::cout << "#diff_v:" << diff_v << "\t";
99         std::cout << "#diff_a:" << diff_a << "\n";
100 #endif
101
102         if(diff<DIFF_FOR_STOP)break;
103         if(test.iterates()>=MAX_ITERATES)break;
104         test.iterates()++;
105     }
106 #ifdef VERBOSE
107     std::cout << "v:\n" << test.centers() << std::endl;
108 #endif
109
```



```
110  #ifdef CHECK_ANSWER
111      test.set_crispMembership();
112      test.set_contingencyTable();
113      //std::cout << "Contingency Table:\n" << test.contingencyTable() << std::endl;
114      std::cout << "Lambda:" << Lambda << "\tEm:" << Em << "\tARI:" << test.ARI() <<
115      outputfile<<Lambda<<"\t";
116      outputfile<<Em<<"\t";
117      outputfile<<test.ARI()<<"\t";
118      outputfile<<"\n";
119  #endif
120
121      if(max_ARI<test.ARI()){
122          max_ARI_Em=Em;
123          max_ARI_Lambda=Lambda;
124          max_ARI=test.ARI();
125      }
126      outputfile<<"\n";
127  }
128  }
129
130  outputfile.close();
131
132  std::cout << "max_ARI_Em:" << max_ARI_Em << std::endl;
133  std::cout << "max_ARI_Lambda:" << max_ARI_Lambda << std::endl;
134  std::cout << "max_ARI:" << max_ARI << std::endl;
135
136  return 0;
137  }
```