生物情報科学演習 課題 2 クラスタリング

生物情報科学科 05-145508 谷川洋介 (yk.tanigawa@gmail.com)

2014 年 12 月 25 日

1 課題 1 : 古典的な Lloyd の方法の実装

1.1 データ生成

データの生成のために，次のようなプログラムを書いた。このプログラムは n 個の点 $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \quad (\forall i. \boldsymbol{x}_i \in [0,1]^d)$ を C++11 の乱数ライブラリのメルセヌスツイスタを用いて生成する。生成されたデータは，ファイルに書きだされる。

データの個数 n，次元の大きさ d は引数として与える。また，reproducible research の観点から重要である，乱数のシードも引数として与えることができる。プログラムがデータの生成の際に使用したシードの値は標準出力に表示される。

ソースコード 1　rand_data_generator.cpp (データ生成のためのプログラム)

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <random>

using namespace std;

/* define constant values */
static const double DATA_RANGE_MIN = 0.0;
static const double DATA_RANGE_MAX = 1.0;

#define DEBUG 0

int main(int argc, char *argv[]){
  if(argc < 4){
    cerr << "usage: " << argv[0]
   << " <n: data num> <d: dimension> <file name> [seed]" << endl;
    exit(1);
  }else{
    int n = atoi(argv[1]), d = atoi(argv[2]), seed;
    char *file = argv[3];
    {
      std::random_device rd;
      if(argc < 5){
#if DEBUG
  seed = 0;
#else
  seed = rd();
#endif
      }else{
  seed = atoi(argv[4]);
      }
    }
    cout << "seed = " << seed << endl;
    {
```

```
      ofstream fs(file);
      if(fs.fail()){ exit(1); }
      {
  std::mt19937 mt(seed);
  std::uniform_real_distribution<double> rand(DATA_RANGE_MIN, DATA_RANGE_MAX);

  for(int i = 0; i < n; ++i){
    for(int k = 0; k < d - 1; ++k){
      fs << rand(mt) << " ";
    }
    fs << rand(mt) << endl;
  }
      }
      fs.close();
    }
    return 0;
  }
}
```

1.2 プログラムの実装

ベクトルの和などの基本的な演算は，ヘッダーファイル my_vector.hpp 内に実装した。

ソースコード 2　my_vector.hpp (ベクトルの基本演算)

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>

using namespace std;

/* ベクトルに対する加算・減算・定数倍などを定義 */
template <class T>
std::vector<T>& operator+=(std::vector<T> &self,
          const std::vector<T> &other){
  for(int i = 0; i < (int)self.size(); i++)
    self[i] += other[i];
  return self;
}

template <class T>
std::vector<T> operator+(const std::vector<T> &self,
                         const std::vector<T> &other){
  std::vector<T> result = self;
  result += other;
  return result;
}

template <class T>
std::vector<T>& operator-=(std::vector<T> &self,
          const std::vector<T> &other){
  for(int i = 0; i < (int)self.size(); i++)
    self[i] -= other[i];
  return self;
}

template <class T>
```

```
34 std::vector<T> operator-(const std::vector<T> &self,
35                          const std::vector<T> &other){
36   std::vector<T> result = self;
37   result -= other;
38   return result;
39 }
40
41 template <class T>
42 std::vector<T>& operator*=(std::vector<T> &self, const T &mul){
43   for(int i = 0; i < (int)self.size(); i++)
44     self[i] *= mul;
45   return self;
46 }
47
48 template <class T>
49 std::vector<T> operator*(const std::vector<T> &self,
50                          const T &mul){
51   std::vector<T> result = self;
52   result *= mul;
53   return result;
54 }
55
56 template <class T>
57 std::vector<T> operator*(const T &mul, const std::vector<T> &self){
58   std::vector<T> result = self;
59   result *= mul;
60   return result;
61 }
62
63 template <class T>
64 ostream &operator<<(ostream &stream, vector<T> vector){
65   for(int i = 0; i < (int)vector.size() - 1; i++){
66     stream << vector.at(i) << ", ";
67   }
68   stream << vector.back() << endl;
69   return stream;
70 }
71
72 template <class T>
73 ostream &operator<<(ostream &stream, vector< vector<T> > matrix){
74   for(int i = 0; i < (int)matrix.size(); i++){ stream << matrix.at(i); }
75   return stream;
76 }
77
78 template <class T>
79 inline double Euclid_norm(vector<T> vec){
80   /* compute Euclid norm of a vector 'vec' */
81   double results = 0;
82   for(int i = 0; i < (int)vec.size(); ++i){
83     results += vec.at(i) * vec.at(i);
84   }
85   return sqrt(results);
86 }
87
88 template <class T>
89 inline double sum(vector<T> vec){
90   double results = 0;
91   for(int i = 0; i < (int)vec.size(); ++i){
```

```
92      results += vec.at(i);
93    }
94    return results;
95  }
96
97  template <class T>
98  double norm(vector<T> vec, int p = 2){
99    /* compute norm of a vector 'vec' */
100   if(p == 2){
101     return Euclid_norm(vec);
102   }else if(p == 1){
103     return sum(vec);
104   }else{
105     double results = 0;
106     for(int i = 0; i < (int)vec.size(); ++i){
107       results += pow(vec.at(i), p);
108     }
109     return pow(results, 1.0 / p);
110   }
111 }
112
113 template <class T>
114 double inner_product(vector<T> v1, vector<T> v2){
115   /* compute inner product of two vectors v1, v2 */
116   double results = 0;
117   int minsize = (((int)v1.size() < (int)v2.size())
118      ? (int)v1.size() : (int)v2.size());
119   for(int i = 0; i < minsize; ++i){
120     results += v1.at(i) * v2.at(i);
121   }
122   return results;
123 }
```

プログラムの実行時間を計測するためのサブルーチンは，ヘッダーファイル time_bench.hpp 内に実装した。

ソースコード 3　time_bench.hpp (実行時間計測のためのサブルーチン)

```
1   #include <time.h>
2   #include <sys/time.h>
3   #include <sys/types.h>
4   #include <unistd.h>
5
6   long unsigned int diff_timeval(struct timeval start, struct timeval end){
7     /* struct timeval の差を micro秒の単位で返す */
8     time_t diff_sec = end.tv_sec - start.tv_sec;
9     suseconds_t diff_usec = end.tv_usec - start.tv_usec;
10    return ((long unsigned int)diff_sec*1000000 + diff_usec);
11  }
12
13  #if 0 /* usage */
14  {
15    struct timeval t_start, t_end;
16    gettimeofday(&t_start, NULL);
17      {
18        /* 計測したい処理 */
19      }
20    gettimeofday(&t_end, NULL);
21    diff_timeval(t_start, t_end);
22  }
```

```
#endif
```

プログラムの本体は，Lloyd.cpp 内に実装した。

ソースコード 4　Lloyd.cpp (Lloyd の K-means アルゴリズム)

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cstdlib>
#include <time.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#include "my_vector.hpp"
#include "time_bench.hpp"

using namespace std;

void re_clustering(const int n, const int d, const int k,
       vector<int> &cluster,
       const vector<vector <double> > representative,
       const vector<vector <double> > data){
  for(int i = 0; i < n; ++i){
    int closest = 0;
    double dist = norm(data.at(i) - representative.at(closest));
    double min_dist = dist;
    for(int z = 1; z < k; ++z){
      dist = norm(data.at(i) - representative.at(z));
      if(dist < min_dist){
  min_dist = dist;
  closest = z;
      }
    }
    cluster.at(i) = closest;
  }
  return;
}

void calc_representative(const int n, const int d, const int k,
       const vector<int> cluster,
       vector<vector <double> > &representative,
       const vector<vector <double> > data){
  vector<int> count(k, 0); /* 各クラスタに何点あるか数える */
  vector<vector <double> > sum(k); /* 各クラスタの座標の和 */
  {
    vector<double> zero(d, 0);
    for(int z = 0; z < k; ++z){
      sum.at(z) = zero;
    }
  }

  /* それぞれのクラスタについて全データ点の座標の和を計算 */
  for(int i = 0; i < n; ++i){
    count.at(cluster.at(i))++;
    sum.at(cluster.at(i)) += data.at(i);
  }
```

```
54
55    /* 代表点を更新する */
56    for(int z = 0; z < k; ++z){
57      representative.at(z) = (1.0 / count.at(z)) * sum.at(z);
58    }
59
60    return;
61  }
62
63  double average_square_err(const int n, const int d, const int k,
64          const vector<int> cluster,
65          const vector<vector<double> > representative,
66          const vector<vector<double> > data){
67    double sum = 0;
68    for(int i = 0; i < n; ++i){
69      sum += norm(data.at(i) - representative.at(cluster.at(i))) * norm(data.at(i) -
          representative.at(cluster.at(i)));
70    }
71    return (sum / n);
72  }
73
74  double Lloyd_repeat(const int n, const int d, const int k,
75          vector<int> &cluster,
76          vector<vector <double> > &representative,
77          const vector<vector <double> > data){
78    re_clustering(n, d, k, cluster, representative, data);
79    calc_representative(n, d, k, cluster, representative, data);
80
81    return average_square_err(n, d, k, cluster, representative, data);
82  }
83
84  void Lloyd(const int n, const int d, const int k,
85       const vector<vector <double> > data){
86    if(n < k){
87      cerr << "data size n is smaller than cluster size d" << endl;
88      exit(1);
89    }else{
90
91      struct timeval t_start, t_end;
92      gettimeofday(&t_start, NULL); /* 時間計測開始 */
93
94      vector<int> cluster(n);
95      /* 各点x_iがどのクラスタに属しているか */
96      vector<vector <double> > representative(k);
97      /* 各クラスタの代表点 */
98      for(int z = 0; z < k; ++z){ /* ランダムに初期化する */
99        representative.at(z) = data.at(z);
100     }
101
102     double sq_err, sq_err_before = Lloyd_repeat(n, d, k, cluster, representative, data);
103     int t = 1; /* 繰り返しを何ステップ行ったか */
104     while(1){
105       sq_err = Lloyd_repeat(n, d, k, cluster, representative, data);
106       if(sq_err == sq_err_before){ break; }
107       t++;
108       sq_err_before = sq_err;
109     }
110
```

```
    gettimeofday(&t_end, NULL); /* 時間計測終了 */

    cout << n << "\t"                              /* サンプル数 */
   << d << "\t"                                 /* 次元 */
         << k << "\t"                               /* クラスタ数 */
   << diff_timeval(t_start, t_end) << "\t" /* 実行時間(us) */
   << sq_err << "\t"                            /* 平均二乗誤差 */
   << t << endl;                                /* 繰り返し回数*/
    return;
  }
}

int main(int argc, char *argv[]){
  if(argc < 5){
    cerr << "usage: " << argv[0]
   << " <n: data num> <d: dimension> <k: cluster num> <data file>" << endl;
    exit(1);
  }else{
    int n = atoi(argv[1]), d = atoi(argv[2]), k = atoi(argv[3]);
    char *file = argv[4];
    vector<vector <double> > data(n);
    {
      ifstream fs(file);
      if(fs.fail()){ exit(1); }
      for(int i = 0; i < n; ++i){
  data.at(i).resize(d, 0);
  for(int j = 0; j < d; ++j){
    fs >> data.at(i).at(j);
  }
      }
      fs.close();
    }
    Lloyd(n, d, k, data);
    return 0;
  }
}
```

1.3 プログラムの実行

下記のようなシェルスクリプトによりプログラムを実行した。レポートとして提出するファイルの中には，生成したデータファイルは含まれていない。

ソースコード 5　exec.sh (プログラムの実行のためのスクリプト)

```
#!/bin/sh

make

seed=0
out_file="../results.txt"

for n in 1000 10000 100000
do
    for d in 2 10 100
    do
  for k in 10 100 1000
  do
      file="../data/n${n}_d${d}.dat"
```

```
      ./rand_data_generator ${n} ${d} ${file} ${seed}
      ./Lloyd ${n} ${d} ${k} ${file} >> ${out_file}
  done
    done
done

make clean
```

1.4 実行結果

プログラムの実行結果を次に示す。各列は順に，n(データサイズ)，d(次元), k(クラスタ数), プログラム実行時間 (マイクロ秒単位), 平均二乗誤差, 収束に要した繰り返し回数を表している。ただし，$n = 100000, d = 100, k = 1000$ のケースについては，プログラムの実行時間がとても長くなったので途中で実行を中止したため，データがない。

ソースコード 6　results.txt (プログラムの実行結果)

```
1000   2 10   145360   0.0178654 17
1000   2 100 684206   0.0015768 10
1000   2 1000   1150707 0 1
1000   10   10   393966   0.571266   30
1000   10   100 1081357 0.282467   10
1000   10   1000   1910273 0 1
1000   100 10   1471656 7.90357 23
1000   100 100 2963917 6.76475 5
1000   100 1000   9589658 0 1
10000 2 10   8903443 0.016817   108
10000 2 100 54266508   0.00164102   90
10000 2 1000   116081399 0.000137662 19
10000 10   10   19665332   0.595875   152
10000 10   100 76076117   0.342454   76
10000 10   1000   161746013 0.167454   16
10000 100 10   102368756 8.05287 160
10000 100 100 192431935 7.55779 38
10000 100 1000   765194613 6.41137 15
100000   2 10   62957070   0.0170205 75
100000   2 100 1678796753   0.00165097   281
100000   2 1000   4590171912   0.000159512 79
100000   10   10   326341487 0.601624   248
100000   10   100 2629831794   0.354804   266
100000   10   1000   7651529662   0.19913 79
100000   100 10   4464783923   8.09676 673
100000   100 100 12602891707 7.73588 252
```

1.5 データの解析

1.5.1 R によるデータのプロット

先の節で得た結果を解釈するために，R を用いてグラフを作成した。下記のプロットに用いたスクリプトを示す。

ソースコード 7　analysis.R (プロット用のスクリプト)

```
path = "~/moris/2_clustering/ytanigawa/"
fig_output_path = paste(path, "report/", sep="")

setwd(dir = path)
data <- read.table("results.txt")
name <-c("n", "d", "k", "time", "err", "rep")
names(data)=name
```

```
v_range<-rbind(c(1000,10000,100000), # 変数nの値域
               c(2,10,100),          # 変数dの値域
               c(10,100,1000))       # 変数kの値域

for(variable in 1:3){                # グラフの横軸にする変数
  for(observed in 4:6){              # グラフの縦軸にする変数
    v2<-((variable %% 3) + 1)        # グラフの点の形を変化させる変数
    v3<-(((variable + 1) %% 3) + 1)  # グラフの色を変化させる変数
    v2_range<-v_range[v2,]           # 変数の値域をセット
    v3_range<-v_range[v3,]           # 変数の値域をセット
    png(paste(fig_output_path, name[variable],"_",name[observed],".png", sep = ""),
        pointsize = 30, width = 1200, height = 1200)
                                     # 保存するpng画像の設定
    par(new=F)                       # 前の画像に重ね書きしない
    # par(xpd=T)                       # 凡例の枠外への描画を許可
    x_lim<-c(min(data[variable]), max(data[variable]))
    y_lim<-c(min(data[observed]), max(data[observed]))
                                     # グラフの描画範囲の設定
    plot(1,0, type="n", xlim=x_lim, ylim=y_lim, log="x",
         main = paste(name[variable]," vs. ",name[observed], sep = ""),
         xlab = name[variable], ylab = name[observed])
                                     # 軸とタイトルのみからなるグラフを描画
    cols<-rainbow(3)
    for(v2_loop in 1:3){
      for(v3_loop in 1:3){
        par(new=T)
        plot_data<-data[intersect(which(data[v2]==v2_range[v2_loop]),
                                  which(data[v3]==v3_range[v3_loop])),]
                                     # 描画するデータを抽出
        plot(plot_data[,c(variable, observed)],
             xlim=x_lim, ylim=y_lim, ann=F, axes=F, log="x",
             pch=v2_loop, col=cols[v3_loop])
        lines(plot_data[,c(variable, observed)],
              xlim=x_lim, ylim=y_lim, ann=F,
              col=cols[v3_loop])
      }
    }
    legend("topleft",bty = "n",inset = c(0, 0),
           legend = c(paste(name[v2], " = ", v2_range[1],sep=""),
                      paste(name[v2], " = ", v2_range[2],sep=""),
                      paste(name[v2], " = ", v2_range[3],sep="")),
           pch = 1:3)
    legend("topleft",bty = "n",inset = c(0, 0.15),
           legend = c(paste(name[v3], " = ", v3_range[1],sep=""),
                      paste(name[v3], " = ", v3_range[2],sep=""),
                      paste(name[v3], " = ", v3_range[3],sep="")),
           lty = 1, col = cols)
    dev.off()
  }
}
```

1.5.2 プロットした結果

図 1 にプロットした結果を示す。プロットはすべて片対数軸で示した。なお，個々のプロットの画像のファイルは，レポートの tex ファイルと同じディレクトリ内にある。

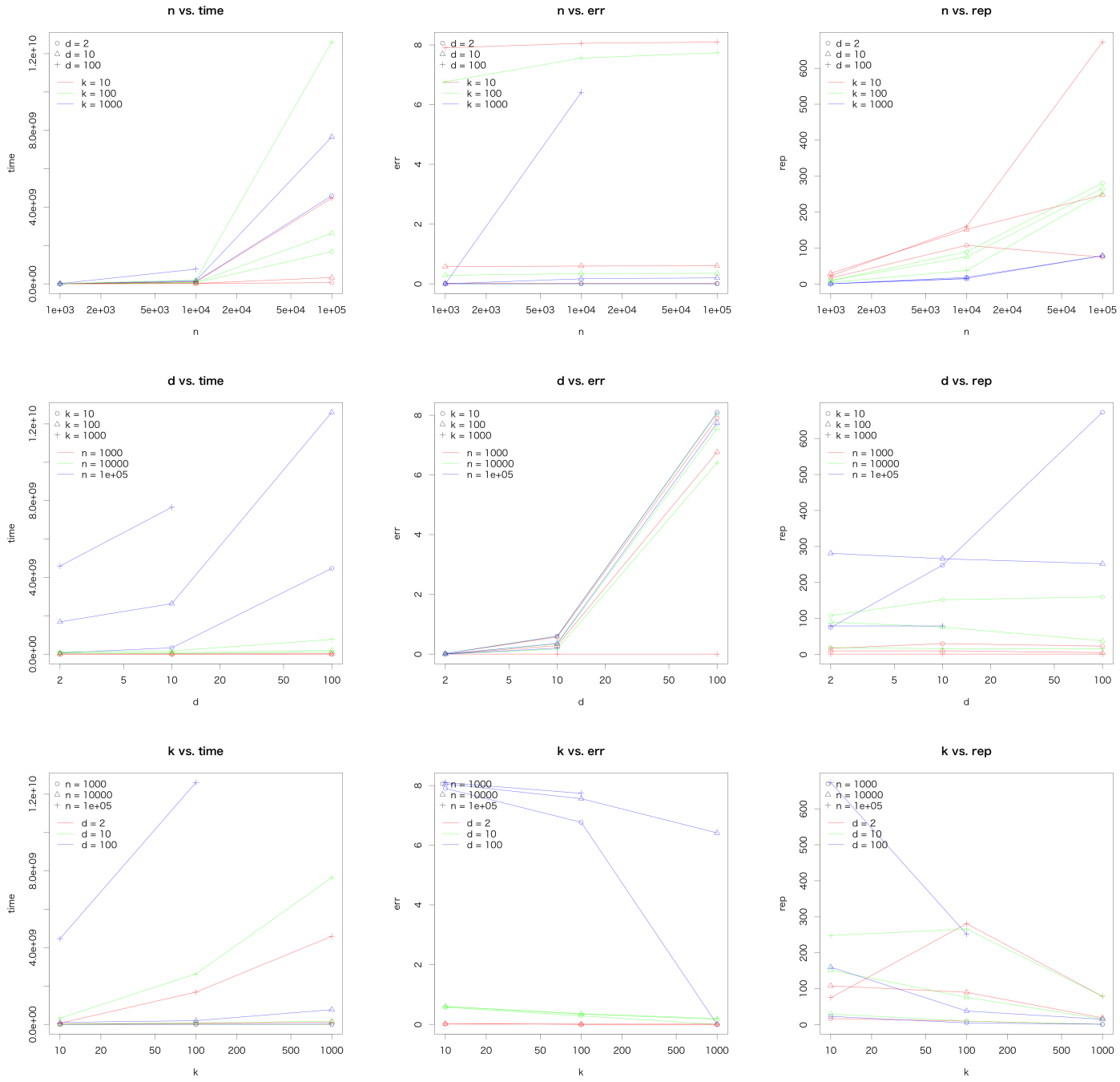

図 1　変数 n, d, k を変化させたときのプログラム実行時間 (time)，平均二乗誤差 (err)，反復回数 (rep) の変化

1.6　考察

1.6.1　Lloyd の k-means 法の計算量

まず，Lloyd の k-means 法のアルゴリズムの繰り返しステップの計算量について考える。

再クラスタリングの手続きにおいては，各データ点について，最近接のクラスタを探すために $O(k)$ の時間がかかり，全データ点について再クラスタリングを行うには $O(nk)$ 時間かかる。

代表点の再計算については，代表点の重心を考えるためには，各クラスタに属する全ての点の座標を見る必要があるため，全体で $O(n+k)$ 時間かかる。

これより，繰り返しステップ全体では，$O(nk+n+k) = O(nk)$ 時間かかると推論できる。

アルゴリズム全体では，この繰り返しステップが何回か繰り返される。何度の繰り返しが必要であるかは，データの性質に依存するため，一般的な解析は難しいが，仮に繰り返し回数を r とおくと，全体で $O(rnk)$ となると推測できる。

1.6.2 データ点の数 n の影響

まず，データ点数 n に対して，平均二乗誤差 (err) はほぼ一定の値をとった。これは，平均をとる操作をしているため，n の大きさに平均二乗誤差は影響を受けないからであると解釈できるだろう。$d = 100, k = 1000$ の振る舞いが一見不思議に見えるかもしれないが，これは $n = 1000$ のとき $k = 1000$ と，データ数と等しい数のクラスタに分類すると，誤差が 0 になるということを示しているだけである。

次に，データ点数 n に対する繰り返し回数 (rep) の変化であるが，データによってばらつきがあるものの，n の増加に対してほぼ指数の増加をしているとみなしてよいのではなかろうか。

データ点の数 n に対してプログラムの実行時間は single exponential より早い速度で増加するようだ。これは，先のアルゴリズムの計算量の考察で $O(rnk)$ と求めたが，r 自体が n の指数で増加するため，$O(rnk)$ 全体では n の指数よりも速い速度で増加すると推論でき，プログラムの実行結果と符合する。

1.6.3 次元 d の影響（「次元の呪い効果の考察」）

まず，次元の呪いの効果について。次元 d と平均二乗誤差 (err) の関係のグラフを見ると，d の増加にしたがって，平均二乗誤差は急速に増加していることがわかる。増加の速度は一重指数よりも速く，$(k = 1000, n = 1000)$ を除く全てのサンプルで同様の結果が得られれている。これは，高次元になると，点同士の間隔が互いに粗になり，クラスタ内での分散が大きくなってしまうというふうに解釈できる。まさに「次元の呪い」効果の一例であろう。$(k = 1000, n = 1000)$ のサンプルにて次元の呪い効果が見られなかった理由は，各点を個々のクラスタとみなした場合に平均二乗誤差が 0 となるからである。

次に，次元の大きさと繰り返し回数 (rep) について。これは，$(k = 10, n = 100000)$ のサンプルを除いてほぼ一定とみてよいだろう。$(k = 10, n = 100000)$ のケースだけ例外的であった理由は，次の通りである。このパラメタの組合せの場合，解としてありえるクラスタリングの数が他のパラメタよりも多くなる。このため，膨大な解空間の中で，現在の解の近傍を渡り歩きながら最適解を探索するために必要な繰り返し回数が多くなったと解釈できるだろう。

最後に次元の大きさとプログラムの実行時間について。次元の大きさによらずほぼ一定の結果と考えることができるが，n が大きい場合には，プログラムの実行時間が長くなるという傾向が見られた。これについては，先に n の効果のところで見たとおりである。

1.6.4 クラスタ数 k の影響

クラスタ数 k を増やすことにより，平均二乗誤差は小さくなる。また，クラスタ数を大きくしたほうが，繰り返し回数が小さくなる傾向にあるのは，解空間が小さくなることによるものであろう。最後に，クラスタ数とプログラムの実行時間であるが，k の増加にたいして指数的に実行時間が増加している。繰り返し回数 r の減少も加味して，プログラム実行時間の k に対する指数増加の説明を考えようとしたが，これについてはあまり良くわからなかった。

2 高速化の実装

3 生物データでの比較

これらについては，まだ試していません。