x-meansクラスタリングプログラム(MATLAB)

x-means法の概要とその実装

一般に多くの点をクラスタリングする手法としてk-means法が知られている．正規分布状に，ある中心点付近に点がまとまって散らばっているクラスターが複数存在するとき，k-means法を用いることである点がどのクラスターに属するかを決定することができる．k-means法はクラスタリング開始時にある仮中心（重心）点を適当に選び，その仮中心点から他の点までの距離を求め，その距離の和が小さくなるよう仮中心点を選びなおし，また距離の総和を求める．仮中心が正規分布の中心にあるとき，中心に密集した多くの点と仮中心の距離が微小になるため，距離の総和が小さくなる．このように仮中心を距離の総和が小さくなるよう繰り返し移動させることであるクラスターの中心に近づける．仮中心がクラスターの真の中心に近づくとほとんど移動することがないので，この仮中心の移動がほぼ収束した点があるクラスターの中心といえる．これをクラスターの個数分の仮中心で同時に行う．仮中心を移動させながら，ある点がクラスタAの仮中心とクラスタBの仮中心のどちらにより近いかを求め，ある点に最も近い仮中心をもつクラスターに点を所属させる．k-means法は繰り返し回数を増やしたり，収束したかを判断する移動の大きさを小さくすることで，数学的に良いクラスタリングが得られる．一方k-means法の問題は，予めプログラム使用者がクラスタの個数を指定する必要があることである．例えば，真のクラスター数が10なのにk-meansでのクラスタ数を5とすると，10個のクラスターを構成しているはずの点群を5個のクラスターにしか分けられない．同様に真のクラスター数が5なのにk-meansでのクラスタ数を100とすると，5個のクラスターを構成しているはずの点群を無理やり100個のクラスターに分けてしまう．従ってk-means法を用いる際は使用者が前もってクラスタ数にあたりをつけておく必要がある．k-means法では必ず「人の手」を加えあらかじめクラスタ数を教えてやる必要があるが，このクラスタ数まで自動で推測し，クラスタリングを行うのがx-means法である．この名前にはkが人が教えてやる明示的な値なのに対し，kの値が分かっていなくても（変数xでも）使用できるという意味が込められている．
x-meansの挙動の概要は以下の通りである．最初にすべての点が1つのクラスタに属する(1-means)として「BIC」という値を求める．BICはベイズ情報量基準と呼ばれ，あるクラスタが正規分布状に広がると仮定したとき，その仮定に対する実際の点の広がりの尤度である．このBICはクラスタが正しく分類できているほど良い値をとる．ここで良い値とはより小さい値かより大きい値のどちらかだが，このどちらであるかはBICの求め方の符号により異なる．続いて2-meansとしたときのBICを求め，1-meansと2-meansのどちらのBICが優れているか，すなわちそのままで良いか，それとも2分割した方がうまく分けられているかを求める．2分割した場合，生じた2つのクラスタのそれぞれもまた再帰的に2分割すべきかBICを用いて判断する．この2分割を繰り返していくとBICの総和が最大になる分割が生じる．この分割方法が最も尤度の高い（最尤推定である）クラスタの分け方，およびクラスタの個数である．
BICの計算は以下の様に行う．ここで，BICはある1つのクラスターに対し「クラスターi自体」と，クラスターiを2分割した「クラスターi_1+クラスターi_2」の2回計算する必要がある．まず，分割前のクラスターi自体のBICは次の様に求める．

$$BIC = -2\log\prod_{j=1}^{R_i}\left(\frac{1}{\sqrt{(2\pi)^M|\boldsymbol{V_i}|}}\exp\left[-\frac{1}{2}(\boldsymbol{x_l}-\boldsymbol{\mu_i})^t{\boldsymbol{V_i}}^{-1}(\boldsymbol{x_l}-\boldsymbol{\mu_i})\right]\right)+\frac{M(M+3)}{2}\log R_i$$

ここで，R_iはクラスターiに属する点の個数である．Mはデータの次元数でx,y座標の2次元データならば$M=2$となる．$\boldsymbol{V_i}$はクラスターiに属するデータの分散・共分散行列である．$\boldsymbol{x_l}$はクラスターiに属するR_i個のデータである．$\boldsymbol{\mu_i}$はクラスターiのデータ$\boldsymbol{x}$の平均である．この式からわかるように，BICはクラスタリング結果の多

変量最尤推定を行う．
一方クラスターiを2分割した際のBICは以下の様に求める．

$$\alpha = \frac{0.5}{K\left(\sqrt{\frac{||\boldsymbol{\mu_1}-\boldsymbol{\mu_2}||^2}{|\boldsymbol{V_1}|+|\boldsymbol{V_2}|}}\right)}$$

$$BIC = -2\log\prod_{j=1}^{R_i}\left(\frac{\alpha}{\sqrt{(2\pi)^M|\boldsymbol{V_{i(l)}}|}}\exp\left[-\frac{1}{2}(\boldsymbol{x_l}-\boldsymbol{\mu_{i(l)}})^t{\boldsymbol{V_{i(l)}}}^{-1}(\boldsymbol{x_l}-\boldsymbol{\mu_{i(l)}})\right]\right)+M(M+3)\log R_i$$

ここで，$K(p)$は標準正規分布においてp以下の値をとる確率（下側確率）である．また$\boldsymbol{\mu_{i(l)}}$，$\boldsymbol{V_{i(l)}}$はクラスターi_1,i_2それぞれの平均，分散・共分散行列である．個々のデータ$\boldsymbol{x_l}$に対し，$\boldsymbol{x_l}$がクラスターi_1に属せばクラスターi_1の$\boldsymbol{\mu_{i(l)}}$，$\boldsymbol{V_{i(l)}}$を使用し，クラスターi_2に属せばクラスターi_2の$\boldsymbol{\mu_{i(l)}}$，$\boldsymbol{V_{i(l)}}$を使用する．
クラスターiの親BICとクラスターiの2分割後の子BICを比較し，親BICの方が大きければ2分割後の方がクラスタリングとしてより優れている，分けた方が自然だと判断し，分割を続行する．クラスターi_1，クラスターi_2に対しそれぞれx-means法を適用し，子BICの値が親BICより大きくなる，すなわちこれ以上分割すると不自然になってしまう時を探し，その直前のクラスターを解とする．

実行結果

図1. 5個のクラスタを乱数により生成

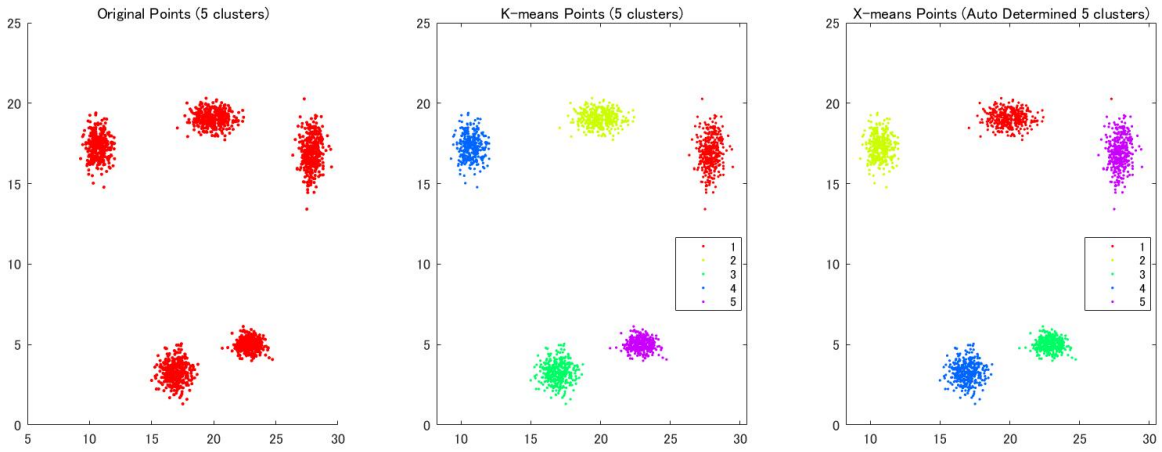


図2. 8個のクラスタを乱数により生成

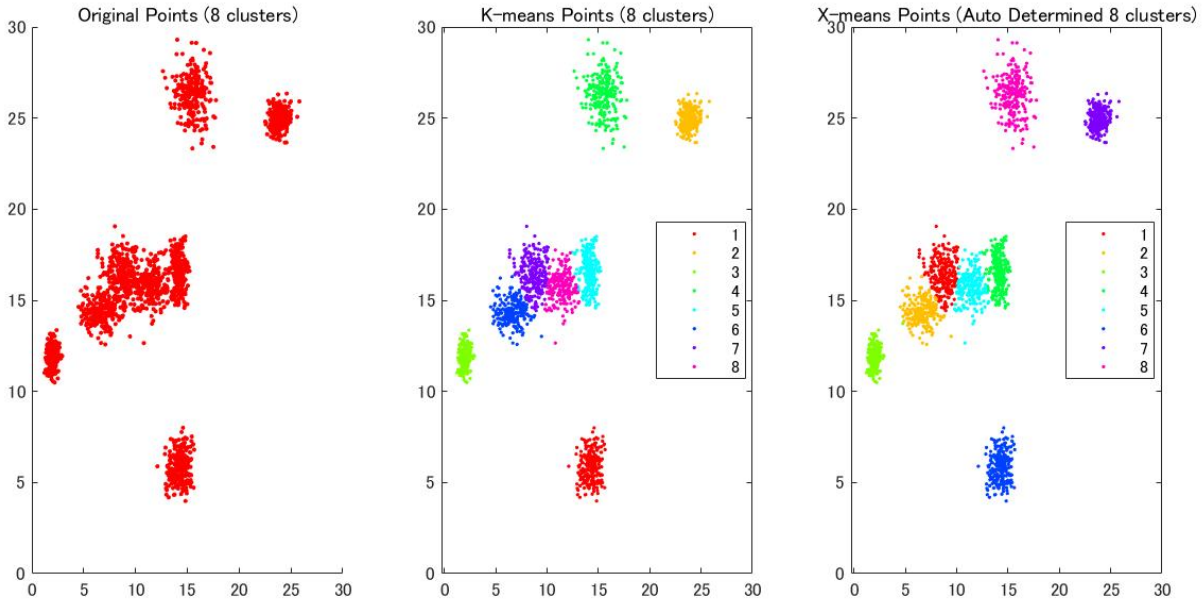
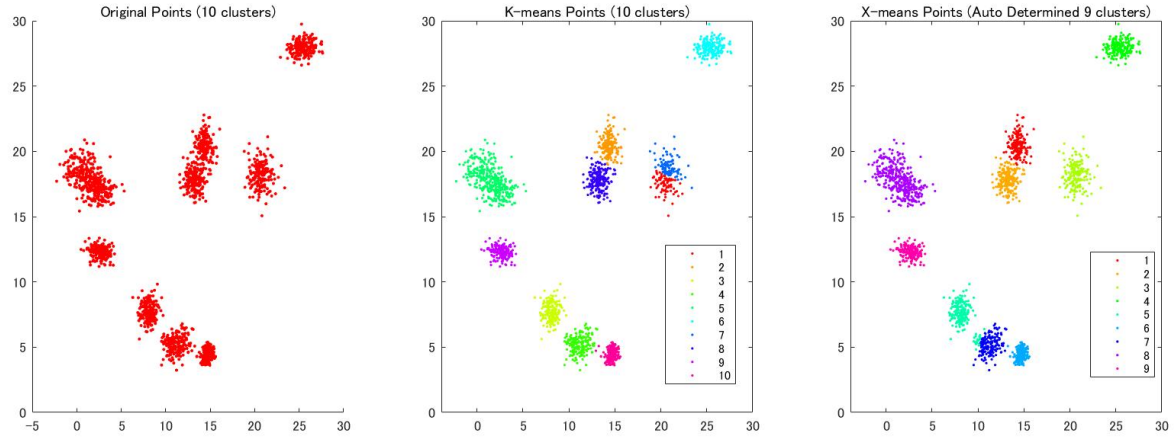


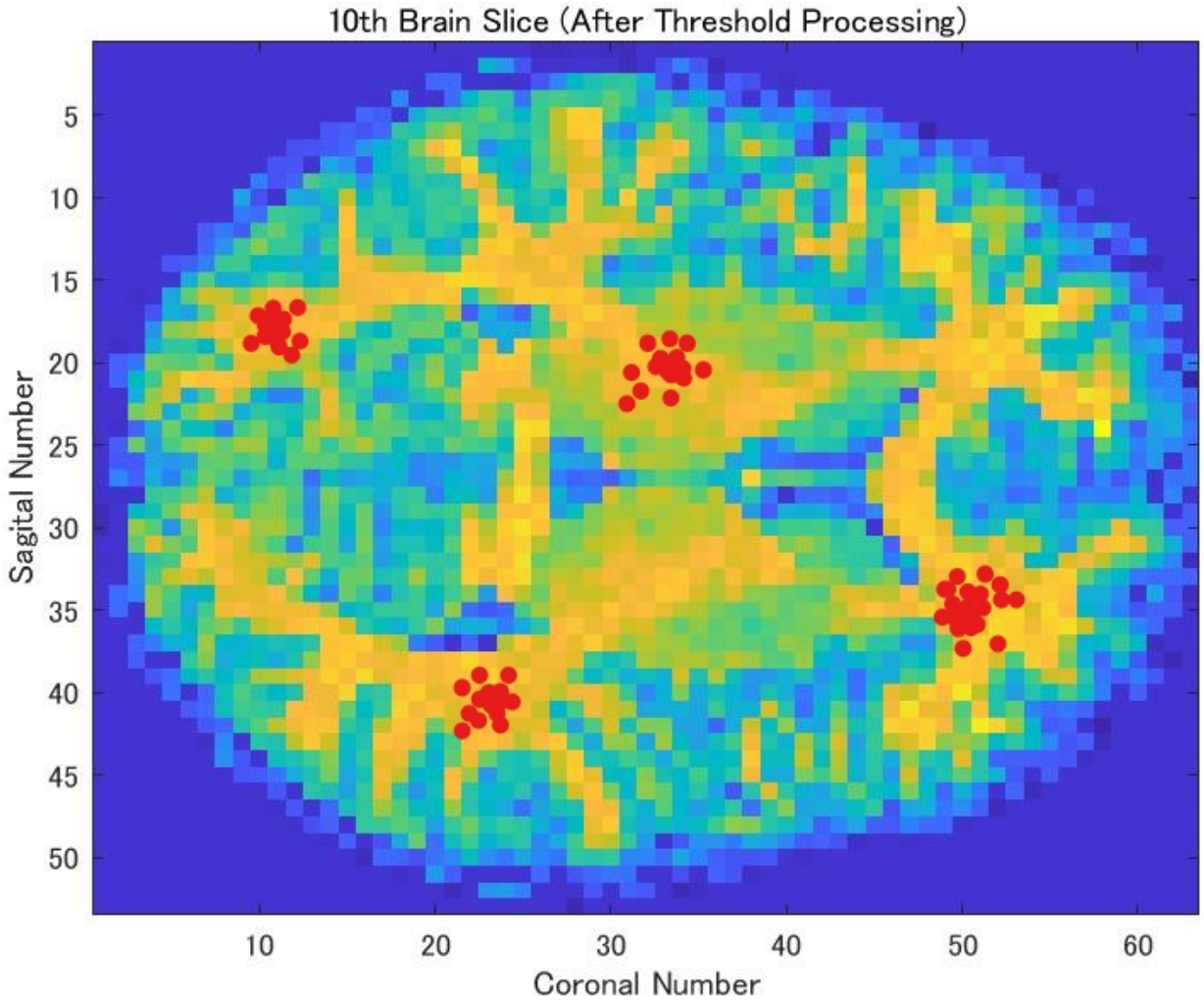
図3. 10個のクラスタを乱数により生成



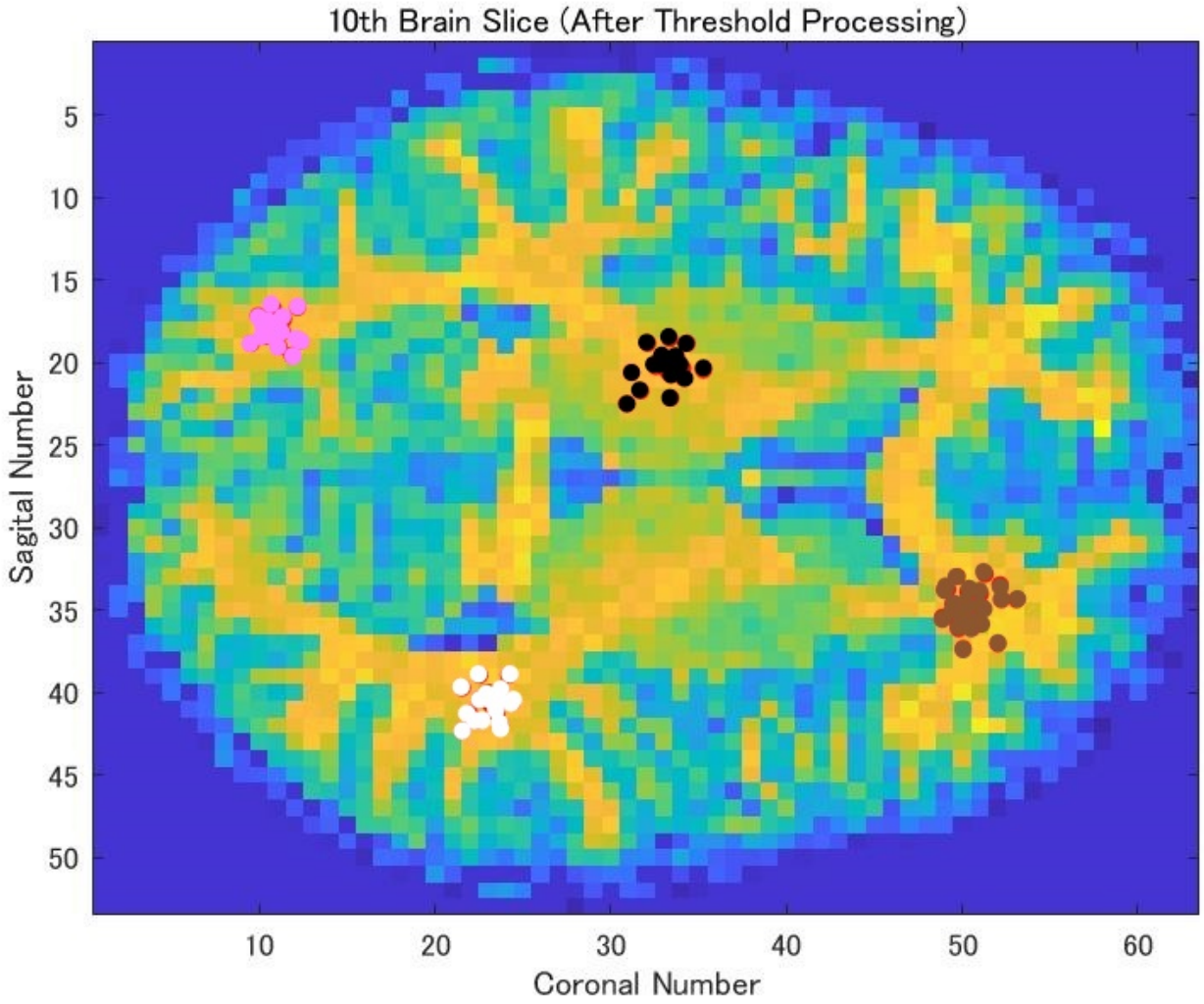
上の図は全て2次元平面上に計1800個の点をある個数のクラスタを構成するように乱数配置したものである．それぞれの行の左がその1800個の2次元平面上での配置である．中央が点生成時に与えたクラスタ数でk-meansクラスタリングした結果である．右がx-meansクラスタリングによりクラスタ数も自動で求めた結果のクラスタリング結果である．x-measn法により，図1,2ではクラスタ数，分割結果ともにk-means法による分割と一致している．図3でx-means法により得られたクラスタ数は9個で，クラスタ生成時に与えたクラスタ数10個に比べ1小さい．しかしこの時クラスタ生成が乱数で行われたため，(x,y)=(0,17)の周辺に2つのクラスタが重なっている．x-means法においてはこの重なりを2つの別のクラスタにするより1つの大きなクラスタとする方が自然である（尤度が高い）と判定したため，より「自然な」分割となるよう9個のクラスタに分割したといえる．

応用

以下の図のような脳の活動マップについて考える．



この活動マップにおいて赤い点が活動状態にある領域やボクセルとする．この場合4つ赤い点が凝集している．
このような点に対してx-means法を実行することで，



上図のようなピンク，黒，白，茶色の点で表されるようなクラスタリングが可能となる．x-means法の最大の利点は人間がクラスタリングごとにクラスタ数を指定する必要がないことで，これを用いることで人の手では対処しきれないほどの大量な脳画像の活動領域を自動でクラスタリングすることが可能である．そのため実験により大量の脳画像が集まった時に有用である．また，大量に保存された脳活動マップを片端から読み取り，活動領域をクラスタリングし，似たような活動領域クラスタリングが出現したデータを抽出する際の前処理として有向だと考えられる．これにより人の手でk-meansクラスタリングする際は手間を惜しんで見向きもしなかったデータでも，とりあえずクラスタリングプログラムにかけることで，思いもよらぬ発見がある可能性がある．

MATLABコード本体

```
clear all;
clc;

pointLength = 1800; % 乱数生成する点の個数
actPlaceM = zeros(pointLength,2);
KTrue = 8; % 正解のクラスタの数

% pointLength個の点をKTrue個のクラスタになるよう乱数で求める
for i = 1:KTrue
    mu = [30*rand 30*rand];
    varRand = -0.05 + rand*0.1;
    rand1 = rand+0.1;
    rand2 = rand+0.1;
    sigma = [rand1 varRand; varRand rand2];
    X = mvnrnd(mu,sigma,pointLength/KTrue);
    actPlaceM((i-1)*pointLength/KTrue+1:i*pointLength/KTrue,:) = X;
end

% 乱数生成してみた点を表示
figure;
subplot(1,3,1);
scatter(actPlaceM(:,1),actPlaceM(:,2), '.','red');
title(['Original Points (',num2str(KTrue),' clusters)']);

% 正しいクラスタ数(KTrue個)でクラスタ数を明示して，従来のk-meansクラスタリングをしてみる
idx = kmeans(actPlaceM, KTrue);
% KTrue個のクラスタに応じた点の分布を図時
%figure;
subplot(1,3,2);
gscatter(actPlaceM(:,1),actPlaceM(:,2),idx);
title(['K-means Points (',num2str(KTrue),' clusters)']);

% 点のxy情報をmatファイルに保存
% save('C:\Users\Dell\WinRobots\develop\hw\biology\言語学C/actPlaceM.mat', 'actPlaceM');

% x-meansにより，クラスタ数を与えず，自動でクラスタ数を判別し，さらに点を分ける
% 第1引数はデータの（データ数,次元）行列
% xmeans()の呼び出し時には第2引数を1で初期化
% xkはクラスタリング結果のクラスタ数，xIndexはデータの属するクラスタ番号
[xK,xIndex] = xmeans(actPlaceM,1);
% 求めたクラスタ数
xK
% 求めたクラスタ数に応じた点の分布を図示
%figure;
subplot(1,3,3);
gscatter(actPlaceM(:,1),actPlaceM(:,2),xIndex);
title(['X-means Points (Auto Determined ',num2str(xK),' clusters)']);

%%

% xは(データ数，次元)のデータ行列例えば(x,y)2次元分布なら(点1のx座標1,点1のy座標1; 点2のx座標1,点2のy座標1; ...;
% prepKは呼び出し時に1として初期化する.
% 返り値 xK: x-meansクラスタリング結果のクラスタ数
% 返り値 xIndex: x-meansクラスタリング結果の各データ点の属するクラスタ番号インデックス
function [xK, xIndex] = xmeans(x,preK)
    R = length(x(:,1)); % クラスタiの点数
```

```
    M = length(x(1,:)); % 点の次元，2次元平面上に点が散らばるならM=2
    index = ones(R,1); % 点がどのクラスタに属するか
    xK = 1;
    xIndex = ones(R,1);
    bicOri = bic(x,1,index);
    idxChild = kmeans(x, 2);

    bicChild = bic(x,2,idxChild);

    % bic(クラスタiそのまま) > bic(クラスタiを2分割した時) なら2分割した方が良い->再帰的に分割を続行
    if(bicOri > bicChild)
        Rchild = zeros(2,1);
        for i = 1:R
            Rchild(idxChild(i)) = Rchild(idxChild(i)) + 1;
        end
        xChild1 = zeros(Rchild(1),M);
        xChild2 = zeros(Rchild(2),M);
        tmp1 = 1;
        tmp2 = 1;
        for i = 1:R
            if(idxChild(i) == 1)
                xChild1(tmp1,:) = x(i,:);
                tmp1 = tmp1 + 1;
            else
                xChild2(tmp2,:) = x(i,:);
                tmp2 = tmp2 + 1;
            end
        end

        % 2分割後のクラスタそれぞれに対し，再帰的にx-meansを行い分割を続ける
        [xKChild1,xIndexChild1] = xmeans(xChild1,preK);
        [xKChild2,xIndexChild2] = xmeans(xChild2,preK + xKChild1);

        % 返り値を整理
        tmp1 = 1;
        tmp2 = 1;
        for i = 1:R
            if(idxChild(i) == 1)
                xIndex(i) = xIndexChild1(tmp1);
                tmp1 = tmp1 + 1;
            else
                xIndex(i) = xKChild1 + xIndexChild2(tmp2);
                tmp2 = tmp2 + 1;
            end
        end

        xK = xKChild1 + xKChild2;
        xIndex = xIndex;
    end
end

%%


function BIC = bic(x,K,index)
    R = length(x(:,1)); % 要素の数:x(要素数,次元)
```

```
M = length(x(1,:)); % 要素の次元
Ri = zeros(K,1);
mui = zeros(K,M); % i番目クラスタの平均座標，i番目のクラスタを2分割したら，(2,M)行列となり，[mu_x1 mu_y1; mu
BIC = 0;

for i = 1:R
    for j = 1:K
        if(index(i) == j)
            Ri(j,1) = Ri(j,1) + 1;
            mui(j,:) = mui(j,:) + x(i,:);
        end
    end
end
for i = 1:K
    mui(i,:) = mui(i,:) / Ri(i,1);
end

% クラスタi全体のBIC
% BIC = -2log[クラスタiの最尤推定量] + M(M+3)log(クラスタiの点の数)/2
if(K == 1)
    x_mu = zeros(R,2);
    % x - mu ベクトルx_mu
    for i = 1:M
        x_mu(:,i) = x(:,i) - mui(1,i);
    end
    Vi = x_mu'*x_mu/R;
    ViInv = inv(Vi);
    l1 = 0;
    % i全体の点のsum((x-mu)^(t)V^(-1)(x-mu))
    for i = 1:R
        l1 = l1 + x_mu(i,:)*ViInv*x_mu(i,:)';
    end
    l1 = -l1/2;
    BIC = -2*(-R*M*log(2*pi)/2 - R*log(det(Vi))/2 + l1) + M*(M+3)*log(R)/2;
end


% クラスタiを2分割した時のクラスタiのBIC
% BIC = -2log[点ごとに2分割後に対応する平均・分散を用いたクラスタiの最尤推定量] + M(M+3)log(クラスタiの点の数
if(K == 2)
    x1_mu = zeros(Ri(1),2);
    x2_mu = zeros(Ri(2),2);
    tmp1 = 1;
    tmp2 = 1;
    % クラスタiの点がi_1,i_2のどちらに属すか判断し，x_mu = x - mu より平均から各点へのベクトルとして格納
    for i = 1:R
        if(index(i) == 1)
            x1_mu(tmp1,1) = x(i,1) - mui(1,1);
            x1_mu(tmp1,2) = x(i,2) - mui(1,2);
            tmp1 = tmp1 + 1;
        else
            x2_mu(tmp2,1) = x(i,1) - mui(2,1);
            x2_mu(tmp2,2) = x(i,2) - mui(2,2);
            tmp2 = tmp2 + 1;
        end
    end
```

```
        % i_1, i_2 それぞれの分散・共分散行列を求める
        Vi1 = x1_mu'*x1_mu/Ri(1);
        Vi1Inv = inv(Vi1);
        Vi2 = x2_mu'*x2_mu/Ri(2);
        Vi2Inv = inv(Vi2);

        normMu1_mu2_2 = power(mui(1,1)-mui(2,1),2) + power(mui(1,2)-mui(2,2),2);
        detVi1 = det(Vi1);
        detVi2 = det(Vi2);
        if detVi1 == 0 && detVi2 == 0 % 0割り対策
            beta = 0;
        else
            beta = power(normMu1_mu2_2/(detVi1+detVi2), 0.5);
        end
        alpha = 0.5/normcdf(beta);

        l1 = 0;
        l2 = 0;
        tmp1 = 1;
        tmp2 = 1;
        % i_1,i_2ごとのsum((x-mu)^(t)V^(-1)(x-mu))
        for i = 1:R
            if(index(i) == 1)
                l1 = l1 + x1_mu(tmp1,:)*Vi1Inv*x1_mu(tmp1,:)';
                tmp1 = tmp1 + 1;
            else
                l2 = l2 + x2_mu(tmp2,:)*Vi2Inv*x2_mu(tmp2,:)';
                tmp2 = tmp2 + 1;
            end
        end
        l1 = -l1/2;
        l2 = -l2/2;
        % log[i_1,i_2ごとの最尤推定量]
        L1 = Ri(1)*log(alpha) - Ri(1)*M*log(2*pi)/2 - Ri(1)*log(det(Vi1))/2 + l1;
        L2 = Ri(2)*log(alpha) - Ri(2)*M*log(2*pi)/2 - Ri(2)*log(det(Vi2))/2 + l2;
        % i_1,i_2を統一
        BIC = -2*(L1+L2) + M*(M+3)*log(R);
    end
end
```