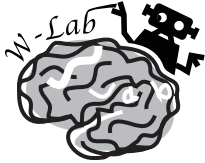
九州工業大学大学院生命体工学科
我妻研究室



幾何学を計算する：ポリゴンを扱ってみよう（2）

講師：我妻 広明 *1,*2,*3

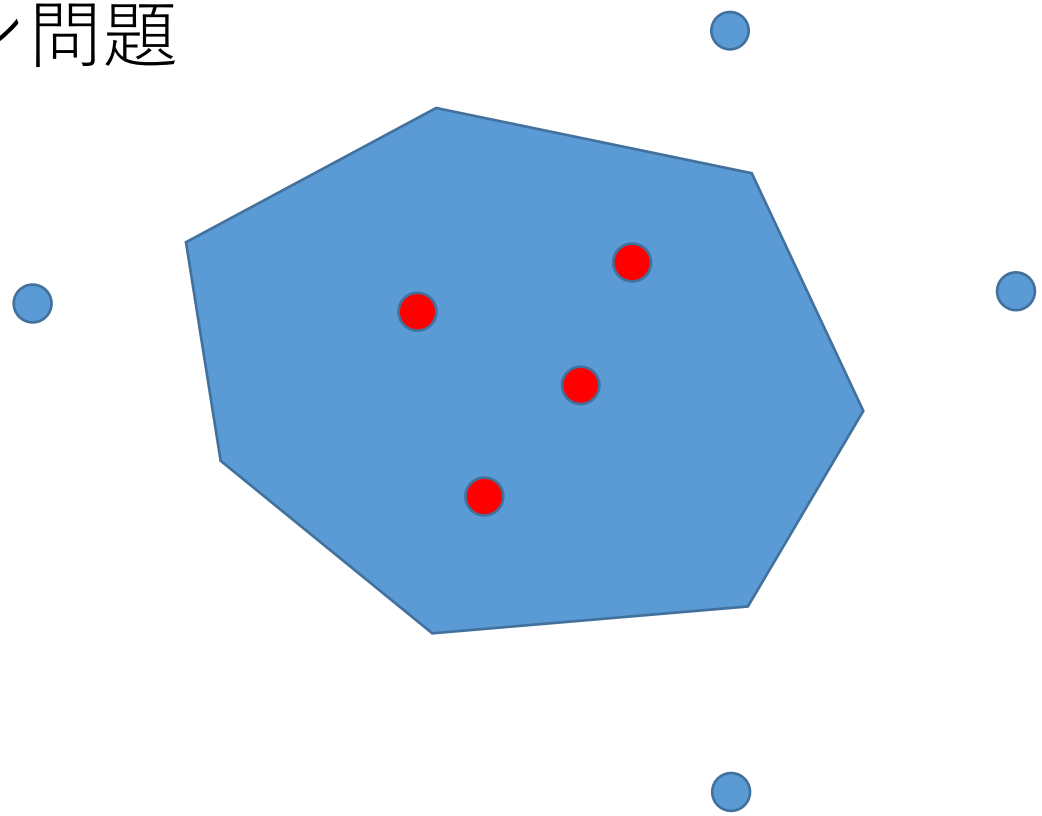
*1 九州工業大学大学院生命体工学科

*2 理化学研究所脳科学総合研究センター

*3 産業技術総合研究所 人工知能研究センター

最初に取り組む課題

- ポリゴン問題



ある点が、多角形（Polygon）に内包されているかどうかを判別する問題

前回のおさらい

1. ポリゴン問題の第一課題である領域内外の点の識別
   ～三角形編
   1.1　問題を解くステップの考察
   1.2　直線と点の関係
   　　～直線の方程式とMATLABプログラミングでの実装

   1.3　不等式の組み合わせ、シフト操作など

   1.4　宿題（次回課題予告）ー意性と最短経路

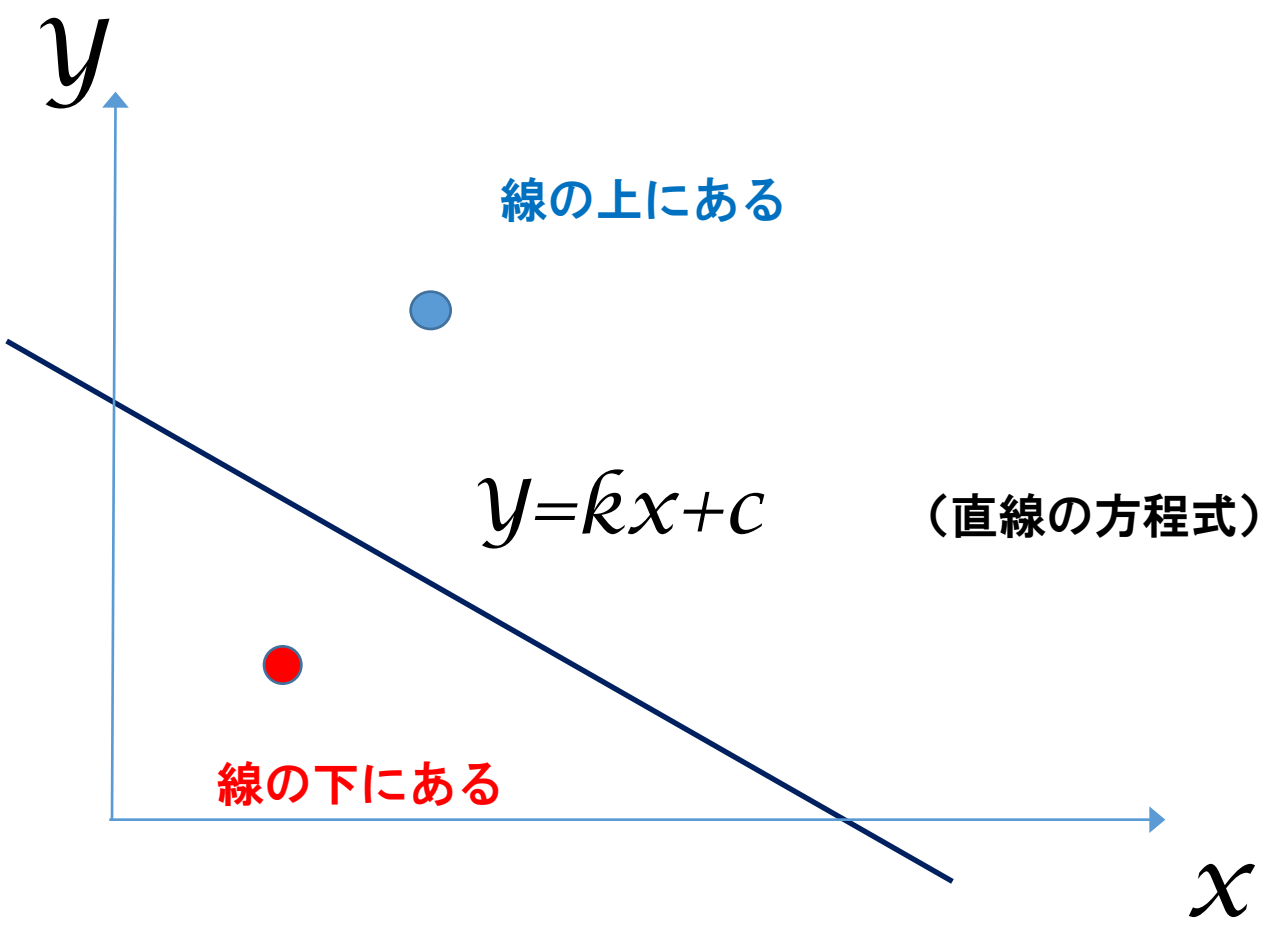
問題を解くステップ

1. 複雑な問題を、解法が得られる要素問題に分解する（problem decomposition）
2. 要素問題の解法は、個別解法でなく、一般化可能な構造化を進める（形式化; structural formulation ）
3. 構造化した解法は、より一般性の高い問題に適用できるように階層化し、再構築する（solution reconstruction）
4. 得られた解法の効果・有効性を、公正に評価する方法を確立する、またはこれまでの方法と比較する（verification/comparison）

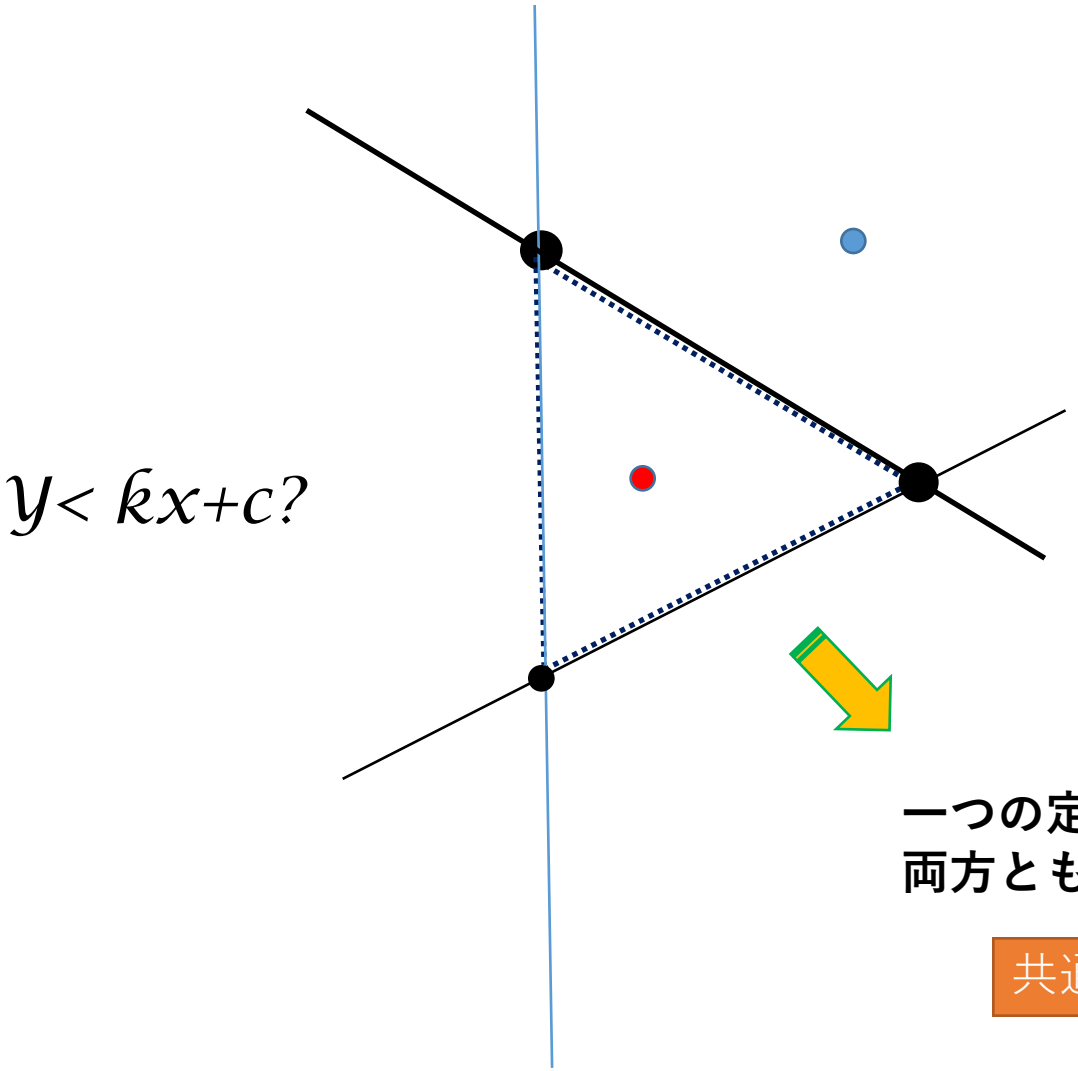
   ⇒ 有効性が他の問題へ適用できるかなど、新規性・独自性検証
5. 得られた解法の特性を分析し、解法の適用範囲の例外となる事例がないかどうか探求する（find an exception）

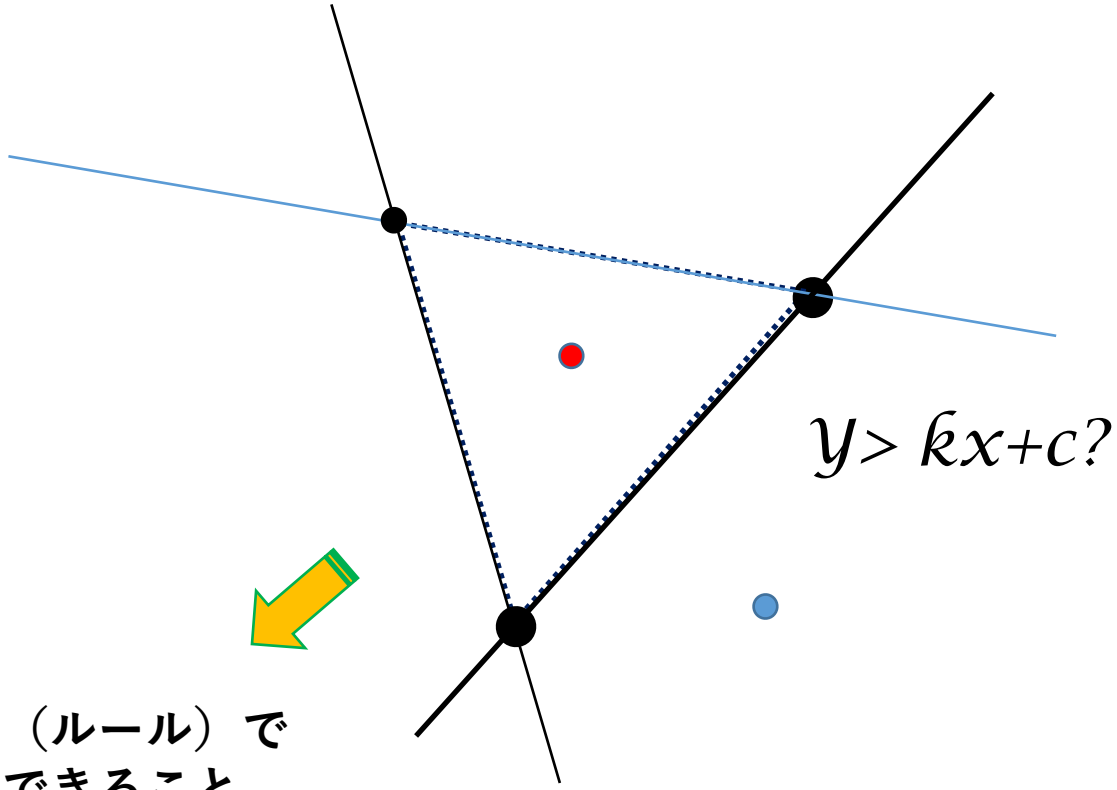
   ⇒ 新たな問題設定への鍵

直線の方程式から復習



一本から　⇒　３本の条件の組み合わせへの拡張



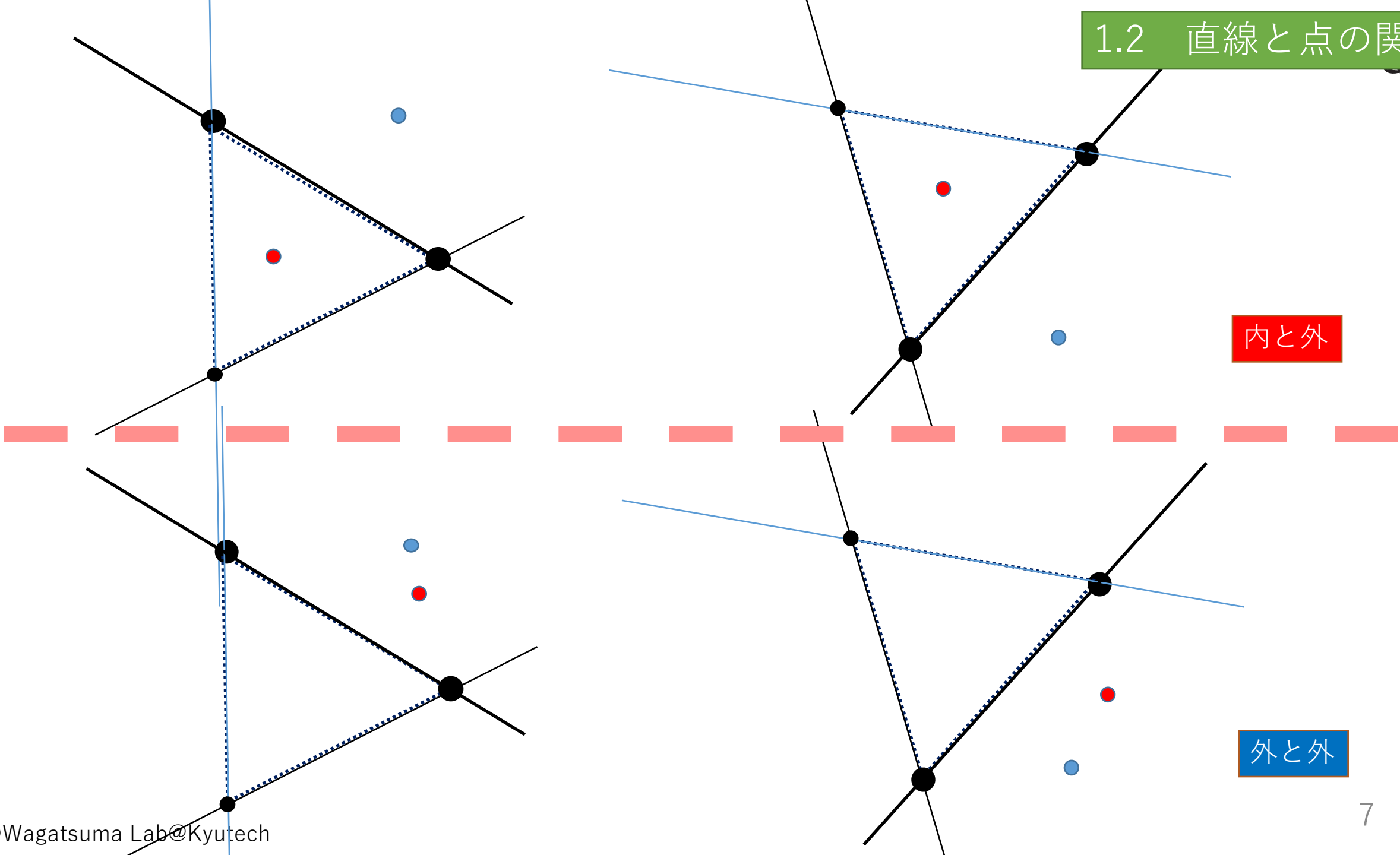


一つの定式化（ルール）で
両方とも識別できること

共通性の抽出

アルゴリズム設計の基本の「き」：常に一般化を考える

1.2　直線と点の関係



提案したアルゴリズムの検証
　⇒ 見通しのよい構造への最適化

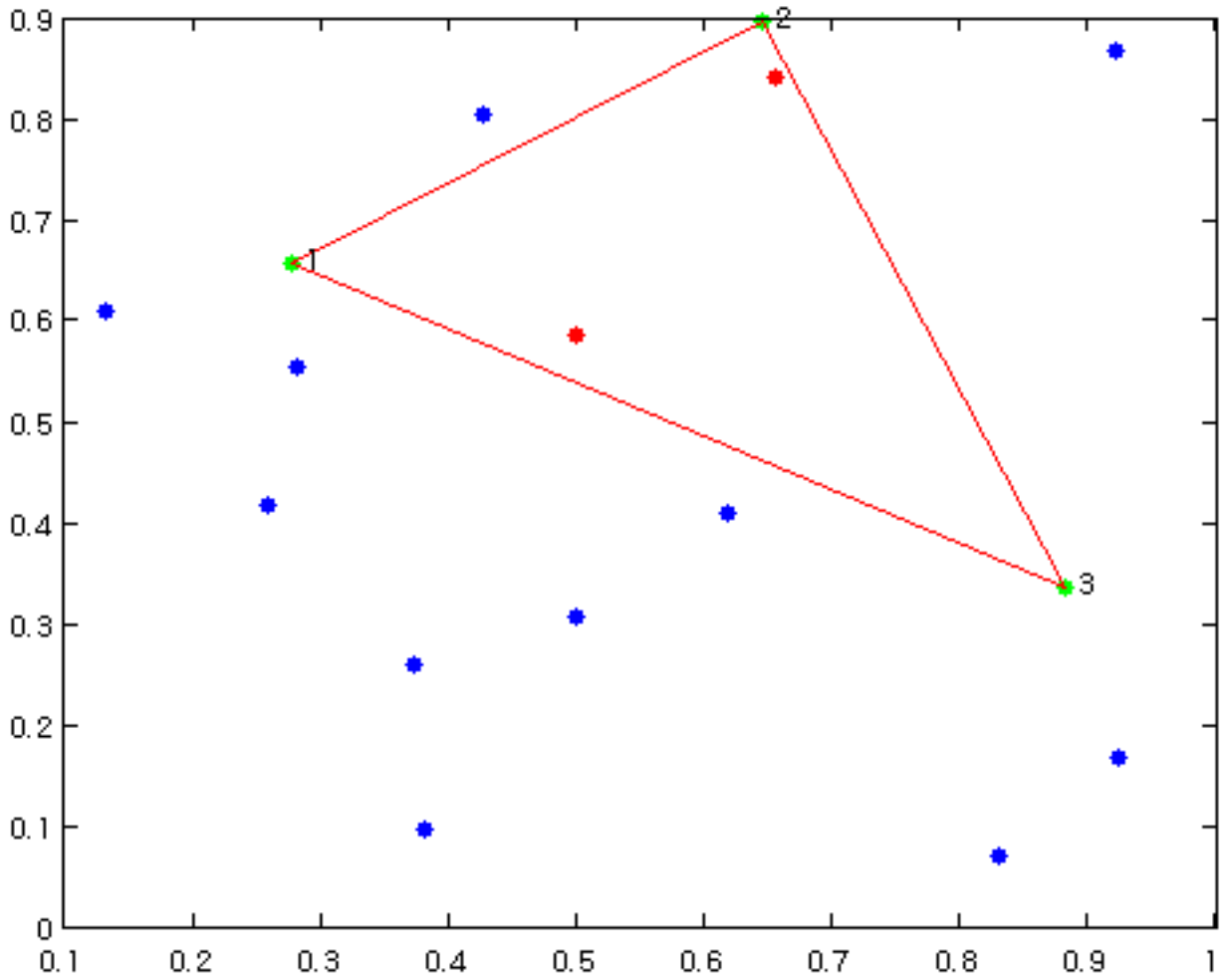
```
% inq=@(x,y,x1,y1,x2,y2) y-y1>(y2-y1)/(x2-x1)*(x-x1);
inq=@(P,P1,P2) P(2)-P1(2)>(P2(2)-P1(2))/(P2(1)-P1(1))*(P(1)-P1(1));
Cinq=@(cPa,cPb,P1,P2) inq(cPa,P1,P2)==inq(cPb,P1,P2);

for j=1:DPnum
if Cinq(DPxy(:,j),FPxy2(:,3),FPxy2(:,1),FPxy2(:,2)) &&
   Cinq(DPxy(:,j),FPxy2(:,1),FPxy2(:,2),FPxy2(:,3)) &&
   Cinq(DPxy(:,j),FPxy2(:,2),FPxy2(:,3),FPxy2(:,4))
      pcol=[1 0 0];
    else
      pcol=[0 0 1];
    end
    plot(DPxy(1,j),DPxy(2,j),'.','MarkerSize',22,'color',pcol), hold on;
end
```

```
if inq(DPxy(:,j),FPxy2(:,1),FPxy2(:,2))==inq(FPxy2(:,3),FPxy2(:,1),FPxy2(:,2)) ...
    && inq(DPxy(:,j),FPxy2(:,2),FPxy2(:,3))==inq(FPxy2(:,1),FPxy2(:,2),FPxy2(:,3))...
    && inq(DPxy(:,j),FPxy2(:,3),FPxy2(:,4))==inq(FPxy2(:,2),FPxy2(:,3),FPxy2(:,1))
```

Polygon_Detect_Plus.m（三角形・内部外部識別・冗長化解消）
https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA006

1.4　宿題（次回課題予告）一意性と最短経路

三角形の場合は各点を結ぶ直線は**一意**（答えが一つしかない）で、**最短**です。



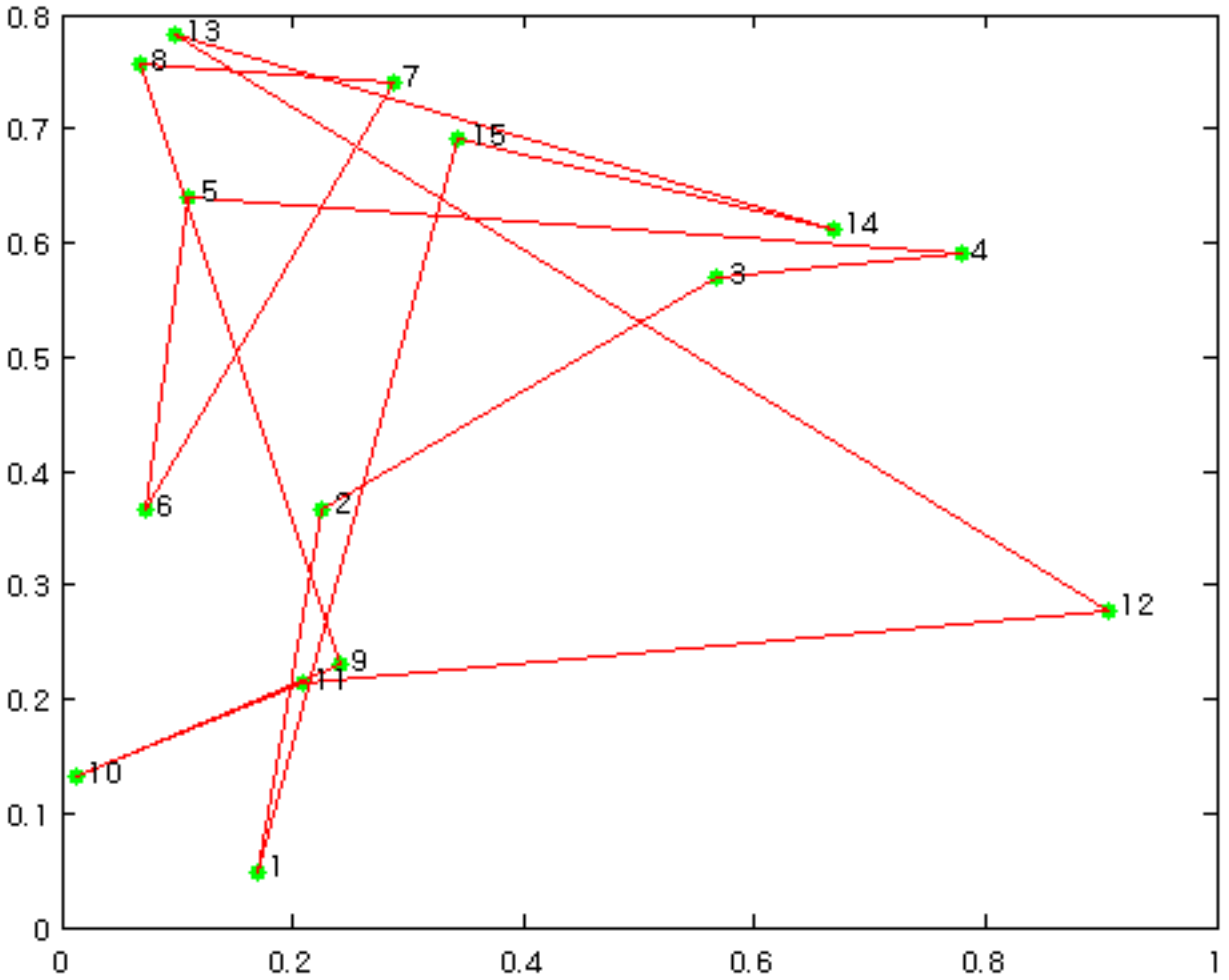
1.4　宿題（次回課題予告）一意性と最短経路

さて、点が３つ以上のときには

i) 一意性

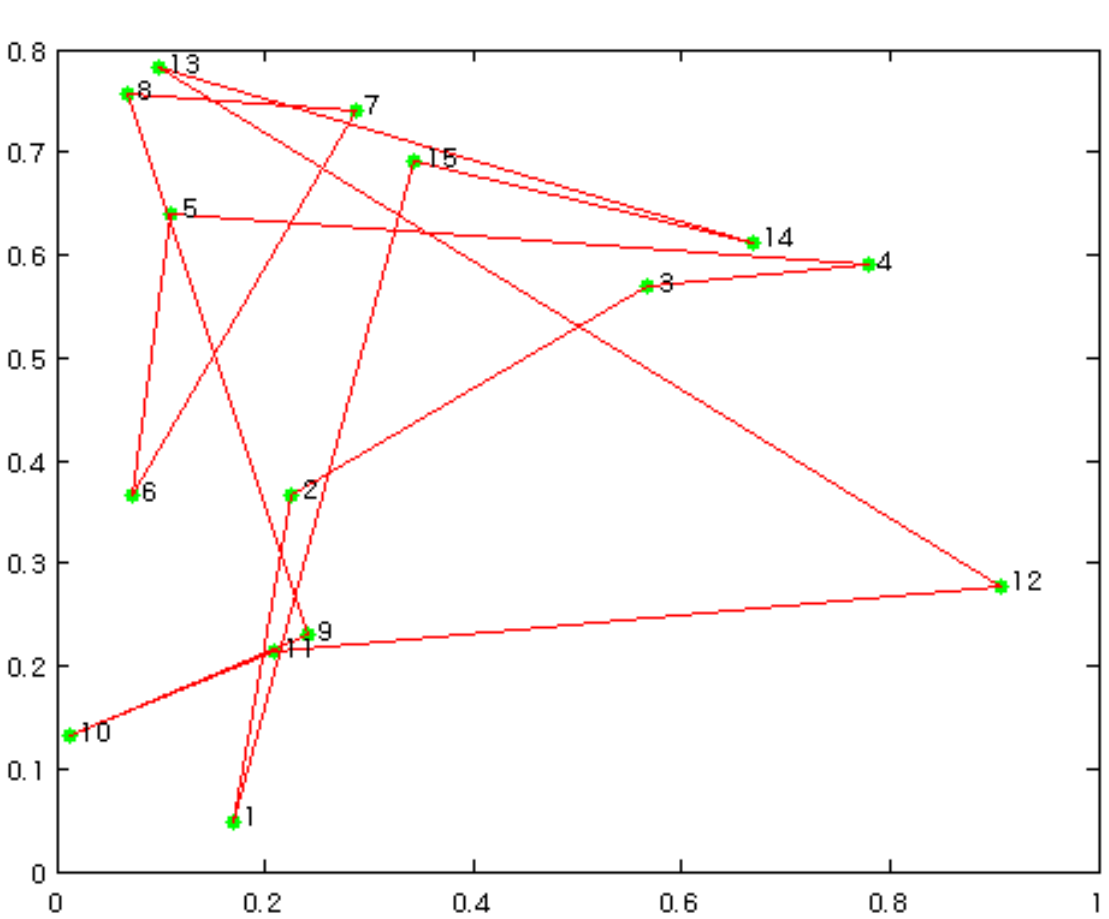
ii) 最短経路

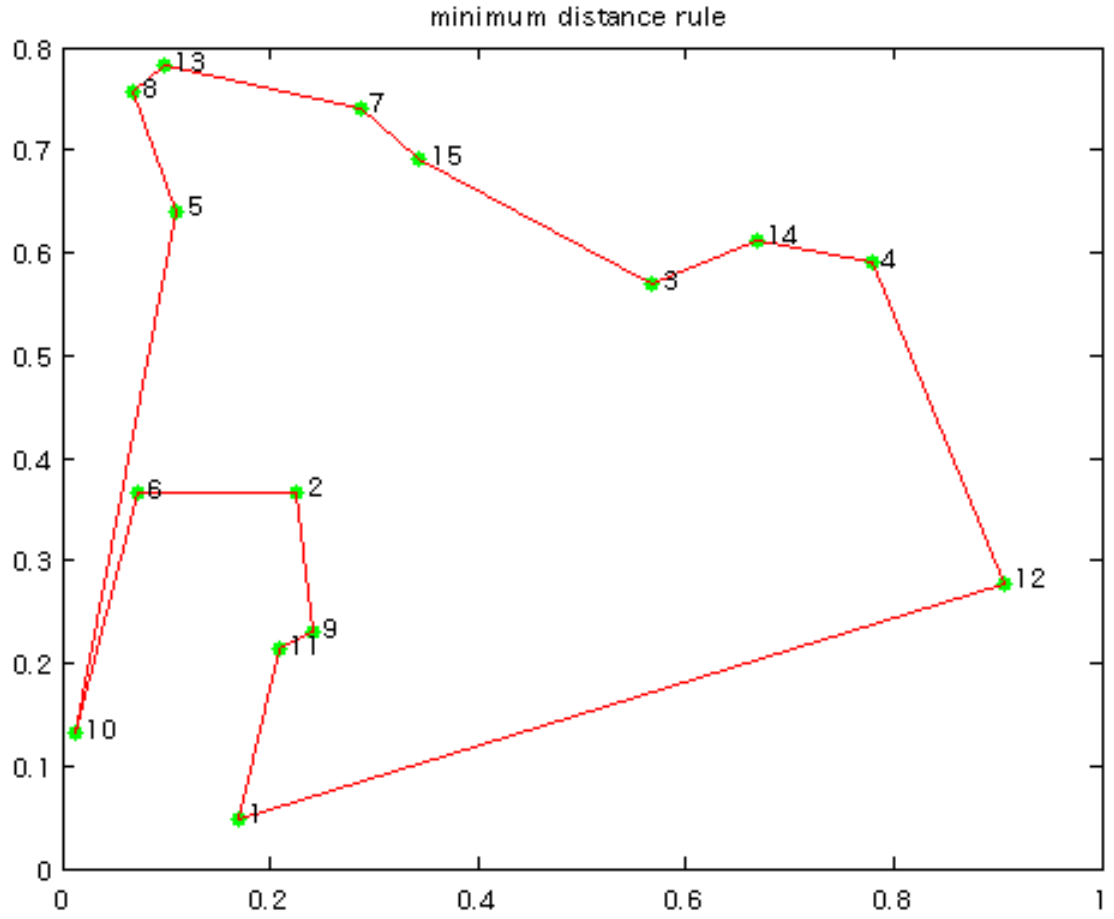
はどうなるでしょう？



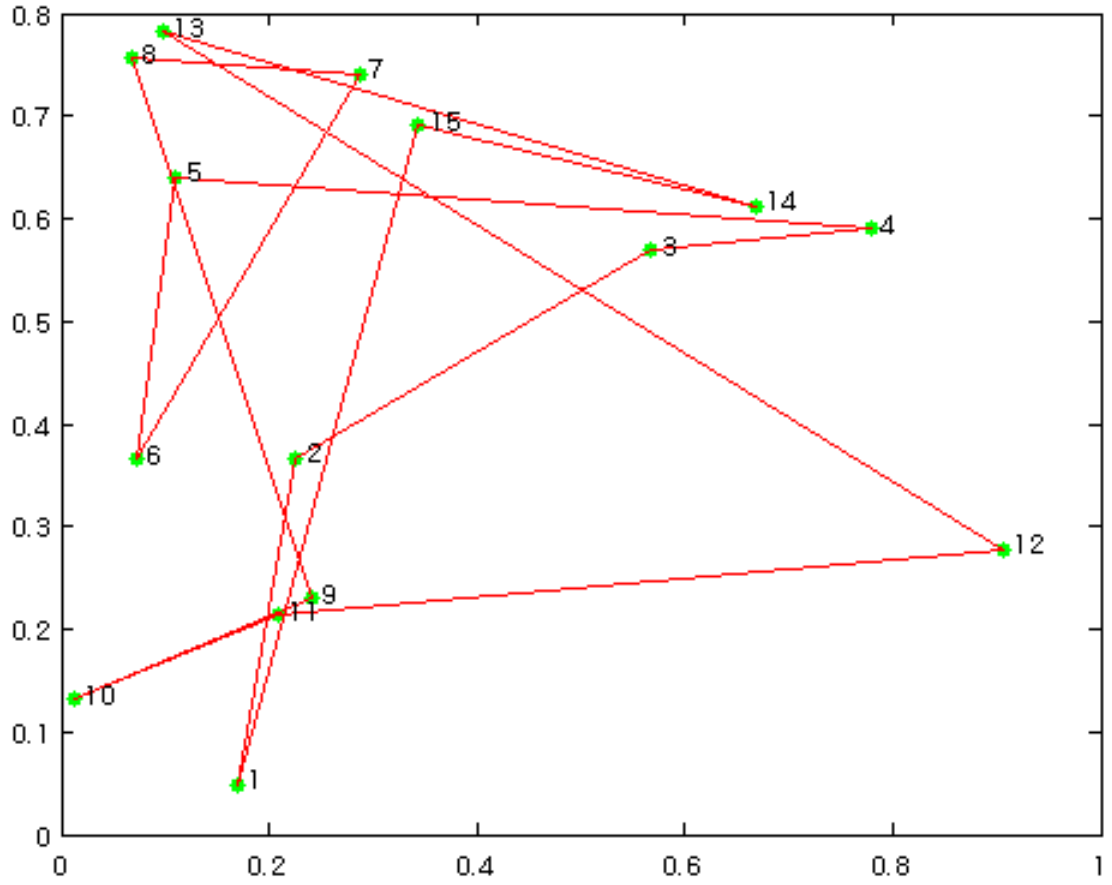
たとえ、点と点をつなぐ直線を１本だけにしても、描く組み合わせは複数あることは明らかですね。では質問を変えて、**最短経路というのは、どうやって与えられるでしょうか？**　それは一意でしょうか？（解が一意に定まらないときには、アルゴリズムによる自動化はできません。）

i) 一意性
ii) 最短経路





課題の一歩：問題の分解



プログラムソースコード
以下からダウンロード可能です。
（但し、ユーザ登録が必要；無料）

Dynamic Brain - INCF Japan Node

https://dynamicbrain.neuroinf.jp/

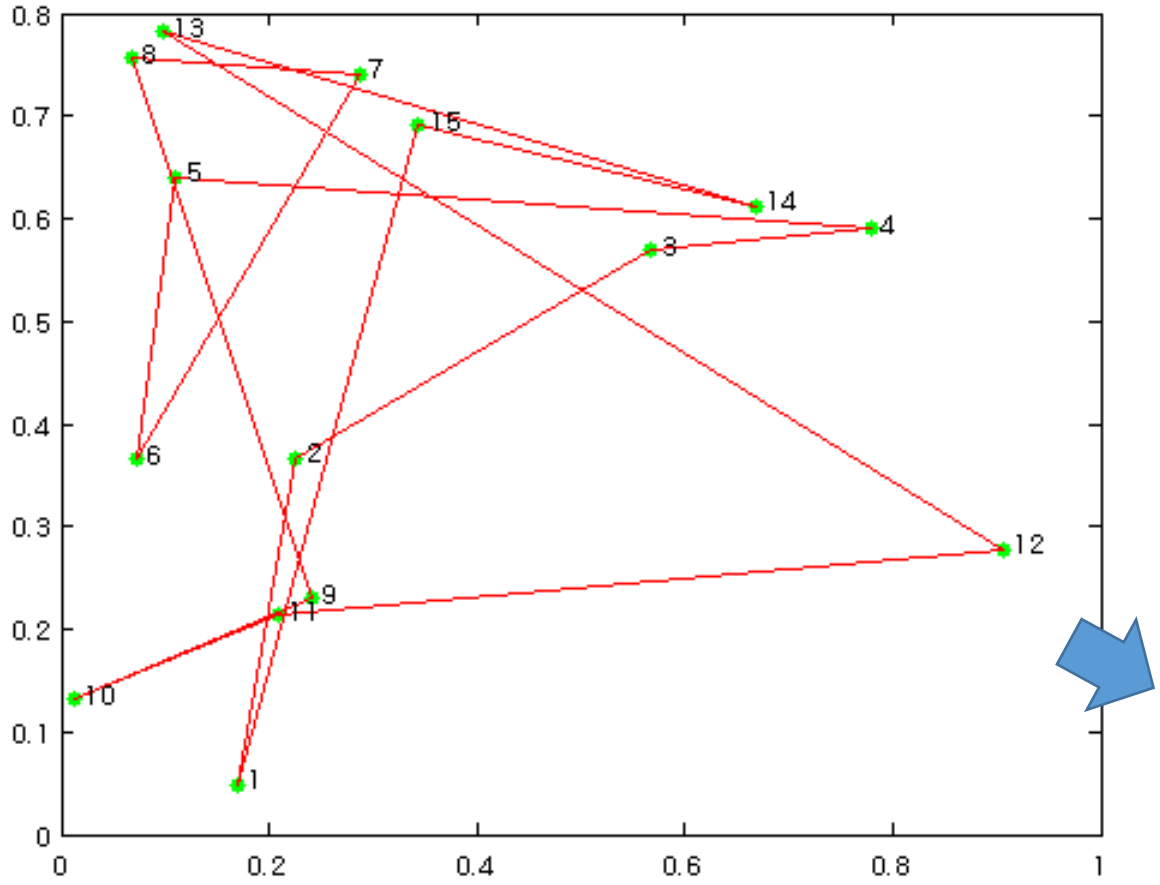
講義タイトル：
「幾何学計算を考える会 2016」

URL：
https://dynamicbrain.neuroinf.jp/modules/mediawiki/index.php?title=幾何学計算を考える_2016&style=m&ml_lang=ja

https://dynamicbrain.neuroinf.jp/modules/mediawiki/index.php?title=幾何学計算を考える_2016&style=m&ml_lang=ja

課題の一歩：問題の分解

1. 線をつなぐ

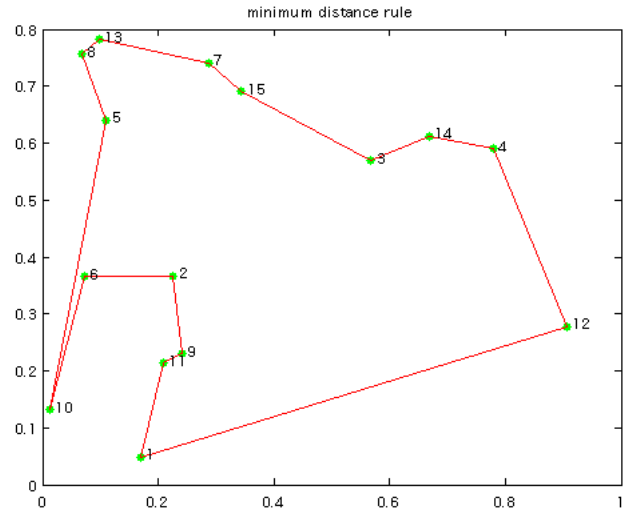


どのように線をつないだら領域を囲むようにポリゴン図形を描けるでしょうか？

手がかり（Key Idea）：

1. 距離（Distance）
2. 角度（Angle）

例：



1.1 とにかく線をつなぐ

```
% frame data generation
FPnum=15;
```

試しに15個データ点列を[x y]' を用意します。

```
disPt = @(P1,P2) sqrt((P1(1)-P2(1)).^2+(P1(2)-P2(2)).^2);
```

準備のために、距離を測る関数を用意します

```
FPxy=rand(2,FPnum);
FPxy2=[FPxy FPxy(:,1)];
```

$$\begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix}$$

原義上、タテベクトルにします

閉じたループを作るために最後列に、最初の点を追加します

```
figure(4); clf
plot(FPxy(1,:),FPxy(2,:),'g.','MarkerSize',22), hold on;
plot(FPxy2(1,:),FPxy2(2,:),'r-');
for j=1:FPnum
    text(FPxy(1,j)+0.01,FPxy(2,j)+0.01,num2str(j));
end
```

緑のマーカー（点）を描画します

赤線で（ランダムに生成された順に）点をつなぎます

マーカーの各点に番号のラベルを追加します

Polygon_Detect_Copper.m（ログインユーザ・ダウンロード）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA008

1.2 点と点の距離を測る

n=Fpnum
（データ数）

```
DupX=repmat(FPxy(1,:),[FPnum 1]);
DupY=repmat(FPxy(2,:),[FPnum 1]);
disXY=sqrt((DupX-DupX').^2+(DupY-DupY').^2);
```

ここでは、今後人工知能技術に展開しやすいように行列演算を基本にしています。

無用な、for 文の二重ループはしないように気をつけましょう。

少なくとも、MATLABで行列演算の代わりにfor文を使うと、計算処理に大幅な計算量が加わって、計算が遅くなります。

$$DupX=\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_n \end{bmatrix}\ n個 \qquad DupY=\begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_1 & y_2 & \cdots & y_n \\ \vdots & \vdots & & \vdots \\ y_1 & y_2 & \cdots & y_n \end{bmatrix}\ n個$$

$$DupX-DupX' = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_1 & x_2 & \cdots & x_n \\ \vdots & \vdots & & \vdots \\ x_1 & x_2 & \cdots & x_n \end{bmatrix} - \begin{bmatrix} x_1 & x_1 & \cdots & x_1 \\ x_2 & x_2 & \cdots & x_2 \\ \vdots & \vdots & & \vdots \\ x_n & x_n & \cdots & x_n \end{bmatrix} = \begin{bmatrix} (x_1-x_1) & (x_2-x_1) & \cdots & (x_n-x_1) \\ (x_1-x_2) & (x_2-x_2) & \cdots & (x_n-x_2) \\ \vdots & \vdots & & \vdots \\ (x_1-x_n) & (x_2-x_n) & \cdots & (x_n-x_n) \end{bmatrix}$$

$$disXY=\sqrt{(DupX-DupX')^2+(DupY-DupY')^2}$$

この行列演算で、
点と点の全ての組み合わせにおける距離が行列（つまり、参照テーブル）として求められる

1.3 アルゴリズムA: 最短経路ルール（minimum distance rule）

仮説（Hypothesis）：

ある点からスタートし、最も距離が近い点を連結するようにつないでいけば、矩形領域が生成できるのではないか？

```
% for j=1:Fpnum は使わない
（ルートは順番通りでないから）

RouteNum=1:FPnum;
RoutePos=zeros(1,FPnum);
NewOrder=[];

j=1; RouteNum(j)=0;
NewOrder=j;
while sum(RouteNum)>0
    key=RouteNum([1:j-1,j+1:end]);
    kData=disXY(j,[1:j-1,j+1:end]);
    [valD, tID]=min(kData(key>0));
    key=key(key>0);
    RoutePos(j)=key(tID);
    RouteNum(key(tID))=0;
    j=key(tID);
    NewOrder=[NewOrder j];
end
RoutePos(j)=1;
```

通った場所にマークするためのflag用

全部0になったら、すべての点を通ったということ

自分自身(j=1)は抜く

key==0は、すでに通過した点なので、それは除外して、最小値を検索(min)する

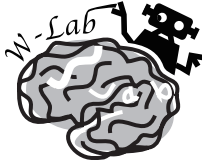
すでに通過した点は外して、IDを振りなおす

自分自身から、最も近い点を次の起点とする

通った点は０にする

経路記録用ベクトル

一番最後は、最初の点に戻す



While ループ 1週目

自分自身(j)を抜いた距離ベクトル

```
kData=disXY(j,[1:j-1,j+1:end]);
```

```
kData =

  Columns 1 through 8

    0.8479    0.6157    0.5470    0.8671    0.3018    0.2467    0.2752    0.2138

  Columns 9 through 14

    0.6940    0.2326    0.3625    0.8974    0.3589    0.2656

valD = 0.2138      tID =8

NewOrder =     1     9
```

8番目

しかし、これはj=1を抜いた番号なので正しくは「9」

j=key(tID)

key()：ID参照テーブル：番号を抜く前のオリジナルのID番号が入っている

2週目

```
kData =

  Columns 1 through 8

    0.2138    0.6425    0.4412    0.5490    0.6536    0.0942    0.1551    0.2830

  Columns 9 through 14

    0.5768    0.2360    0.2031    0.7275    0.1598    0.3129

tID =     5

NewOrder =     1     9     6
```

1番目 2番目 3番目 4番目 5番目

min(kData(key>0))

すでに通過した1,9は除外; key=RouteNum([1:j-1,j+1:end]);

3週目

```
tID =   11

NewOrder =     1     9     6    14
```

RouteNum

通った場所にマークするためのflag用配列

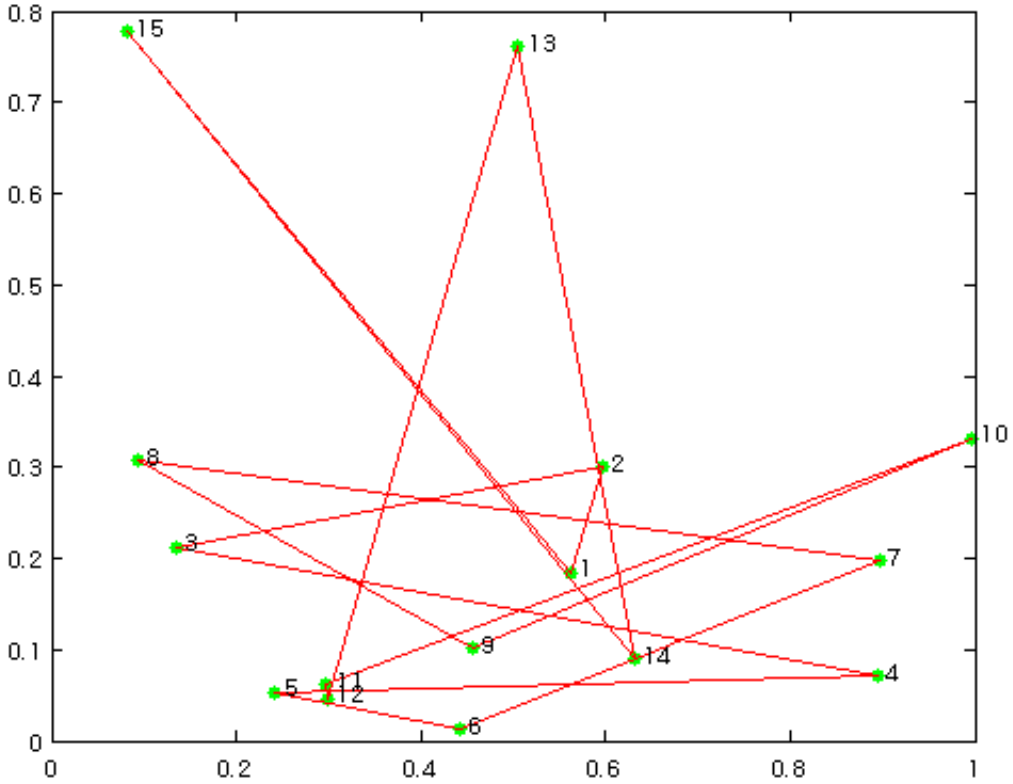
n週目

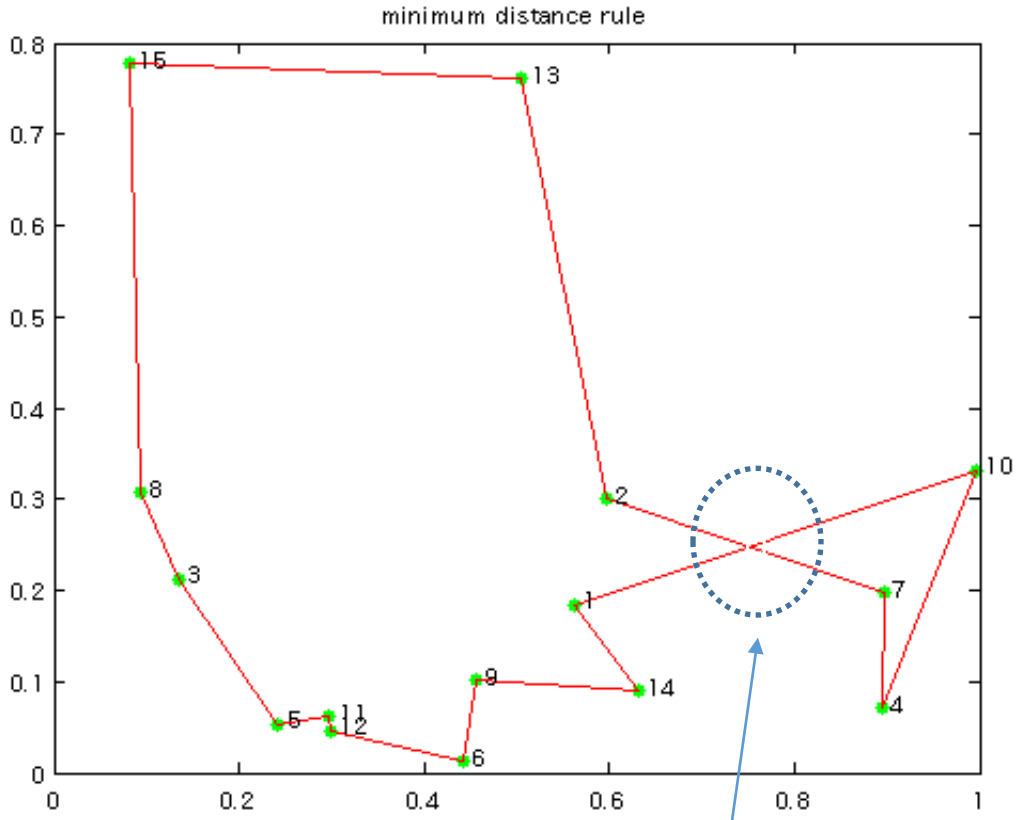
```
tID = 1

NewOrder =  1     9     6    14    12     7    15    11     8     4    10     3    13     5     2
```

次の一歩：問題の解析

Polygon_Detect_Copper.m（ログインユーザ・ダウンロード）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA008



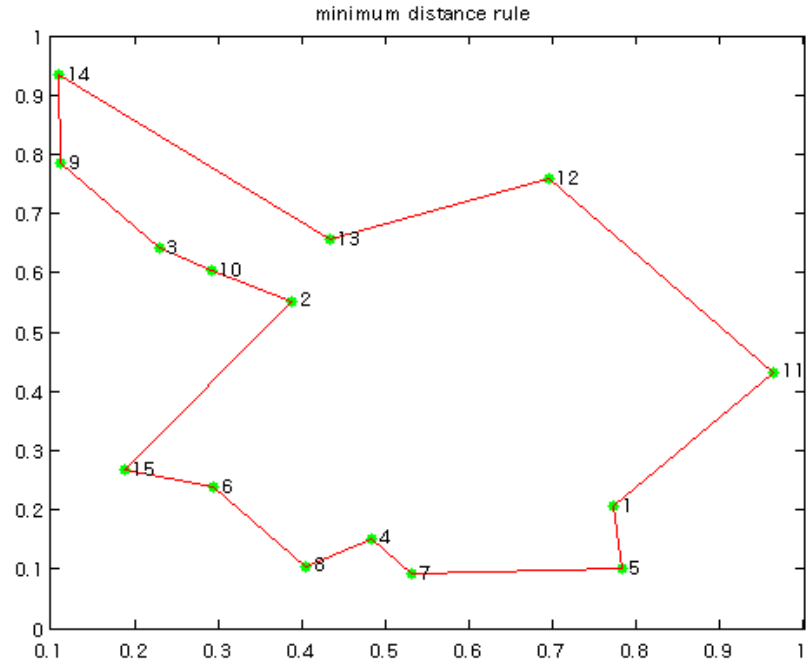


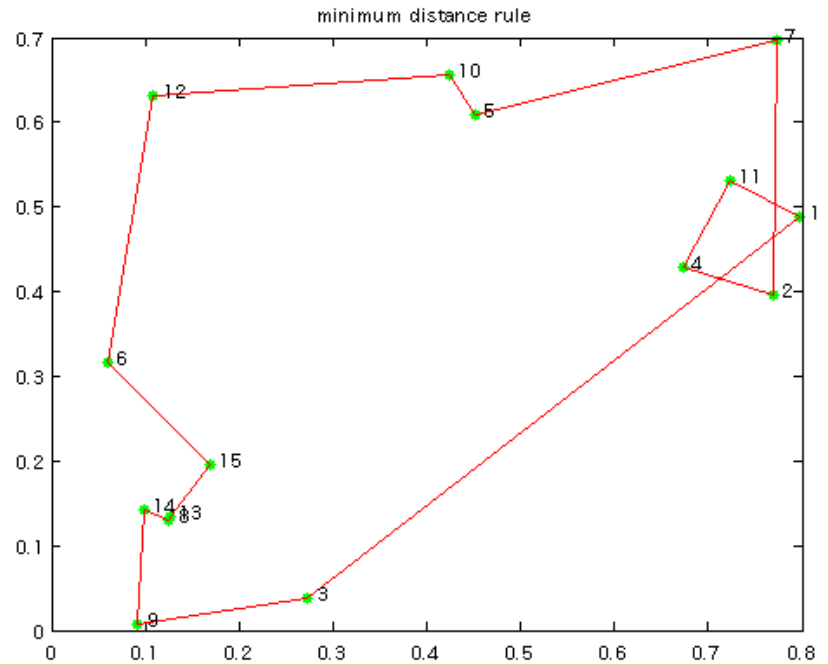
仮説（Hypothesis）：

ある点からスタートし、最も距離が近い点を連結するようにつないでいけば、矩形領域が生成できるのではないか？

仮説は正しかったのか？

ねじれ無し

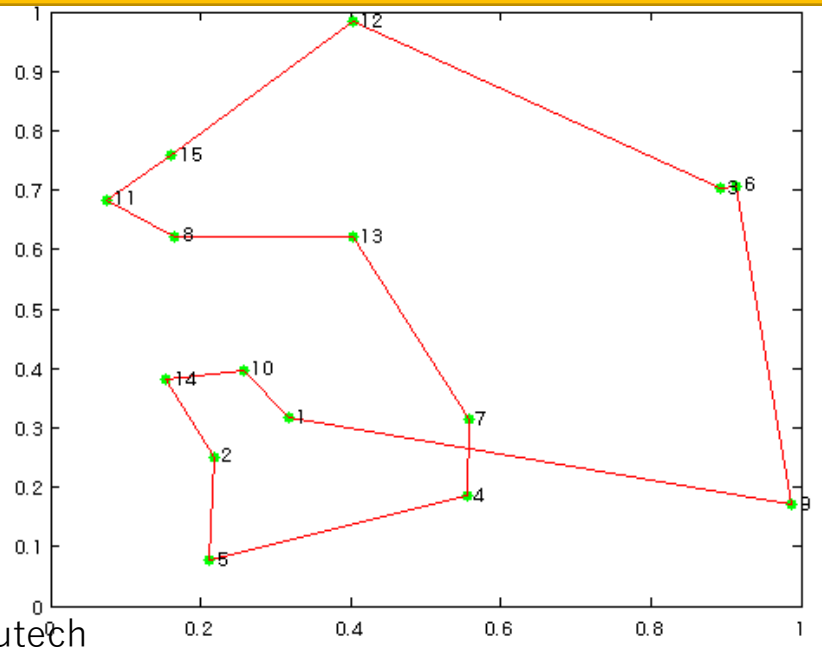


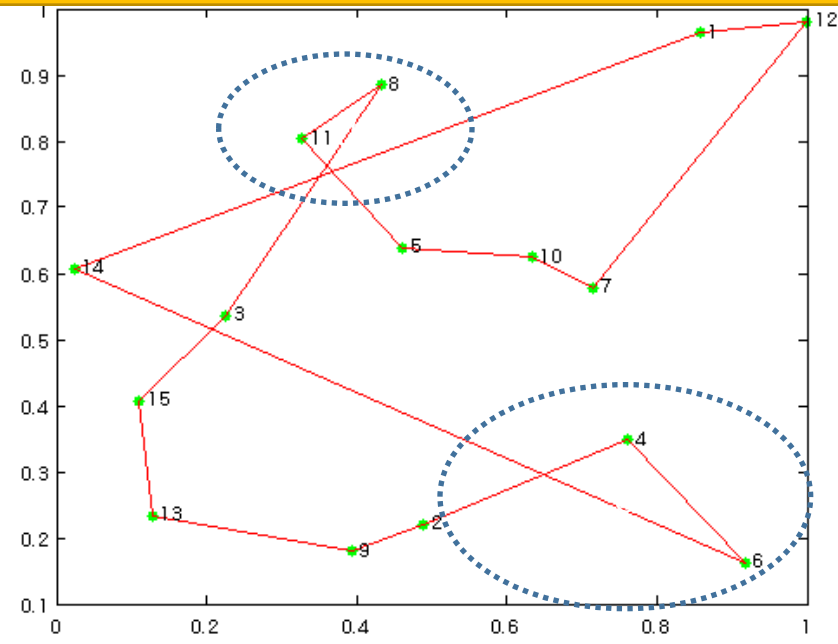


部分ねじれ

本来の目的：矩形（連続）領域の抽出 ⇒ ねじれは好ましくない

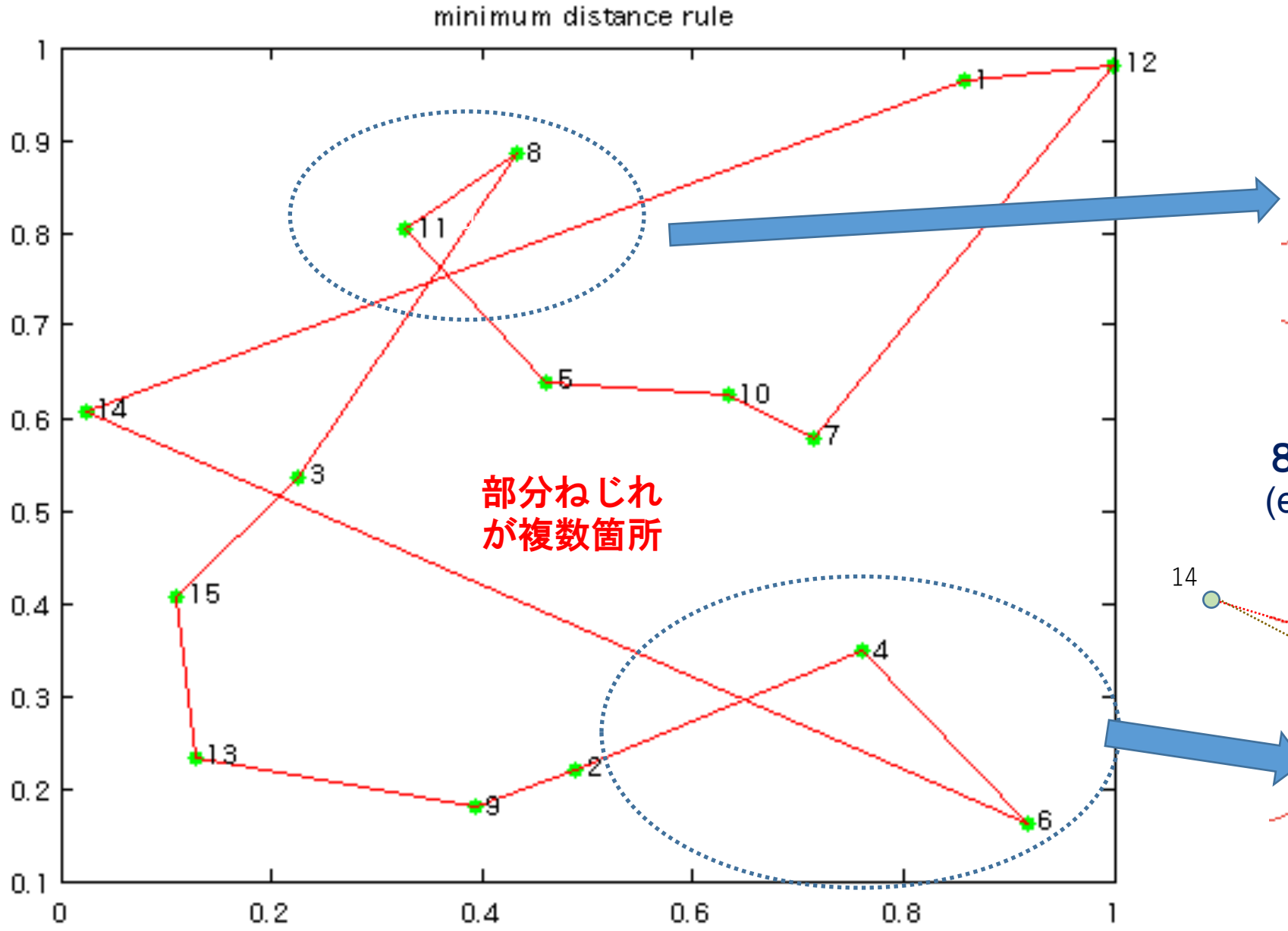
大域ねじれ



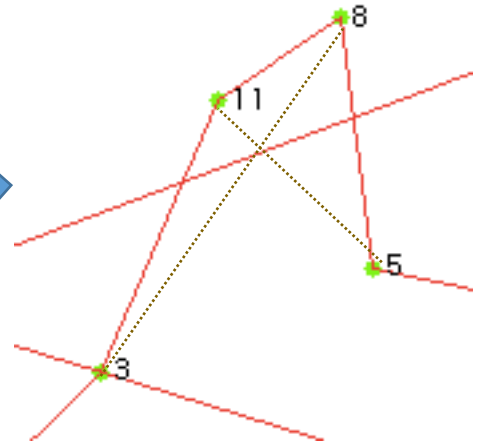


部分ねじれ
が複数箇所

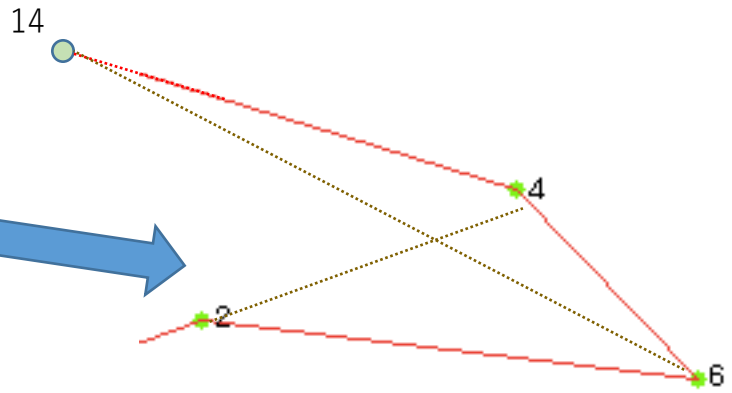
2.1 まずは部分ねじれの解消から、問題解決の糸口を見つけてみよう



部分ねじれ
が複数箇所



８と11を入れ替えればよい
(exchange of 11 with 8)

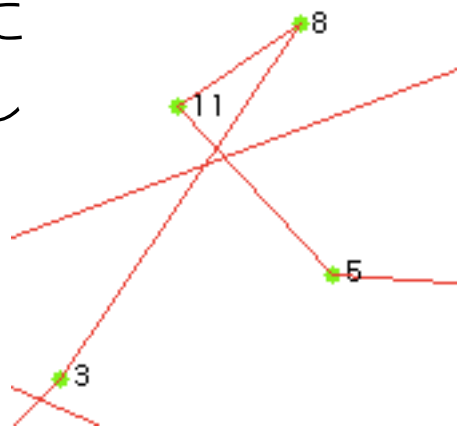


４と６を入れ替えればよい
(exchange of 4with 6)

2.2 アルゴリズムB: 局所ねじれ回転（re-twist at local point）

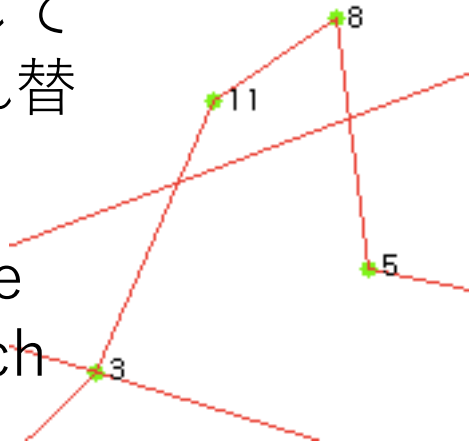
もし、経路中にねじれを発見したら

If there exists a twisting on the route



もし、接続してる番号を入れ替えて、

exchange the numbers each other



ねじれを正す

correst the twisting

Q1: How to find

？

Q2: How to correct

Old [… 5, 8, 11, 3, …]

New [… 5, 11, 8, 3, …]

配列番号を入れ替える

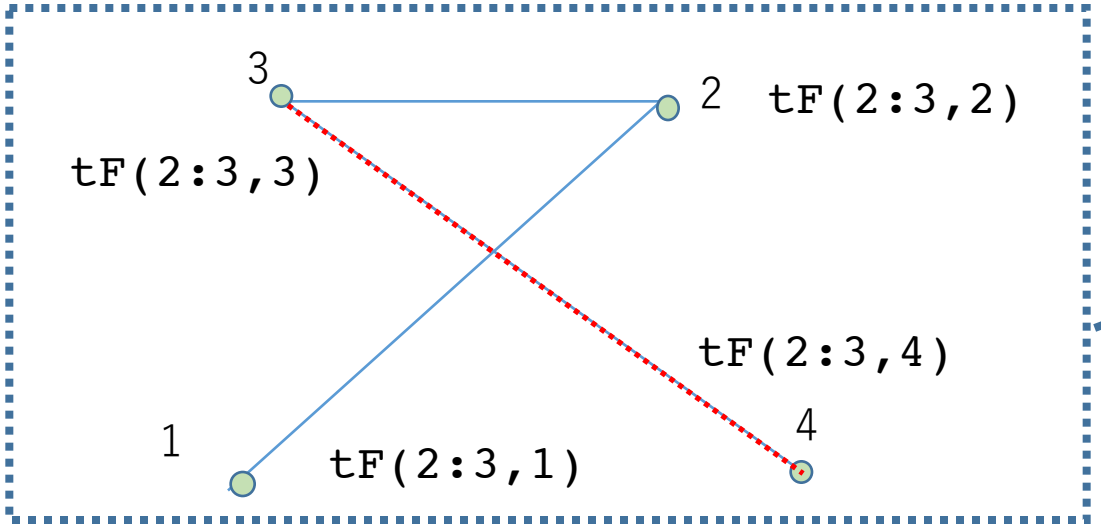
`fliplr()`

これまで定義した関数で処理してみる　（try to use defined functions ）

```
Cinq=@(cPa,cPb,P1,P2) inq(cPa,P1,P2)==inq(cPb,P1,P2);
```

```
tF=[1:FPnum; FPxySort]; tFP=FPxySort; reNewOrder=NewOrder;
for j=1:FPnum

    if ~Cinq(tF(2:3,1),tF(2:3,2),tF(2:3,3),tF(2:3,4))
        display(tF(1,1));

        tFP(:,tF(1,2:3))=fliplr(tFP(:,tF(1,2:3)));
        reNewOrder(tF(1,2:3))=fliplr(reNewOrder(tF(1,2:3)));
    end
    tF=circshift(tF,[0 -1]);
end
```

3-4番でつながれた線を点1,2が両側に分かれて存在する場合は、線が交差していると判断する

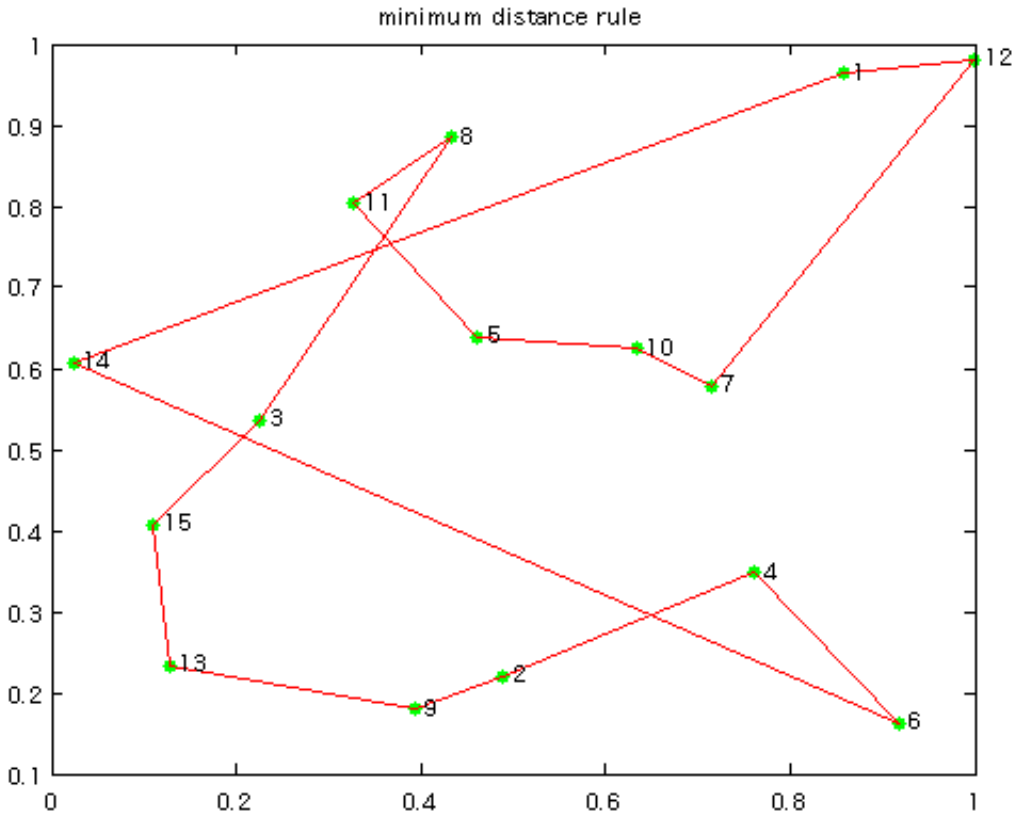


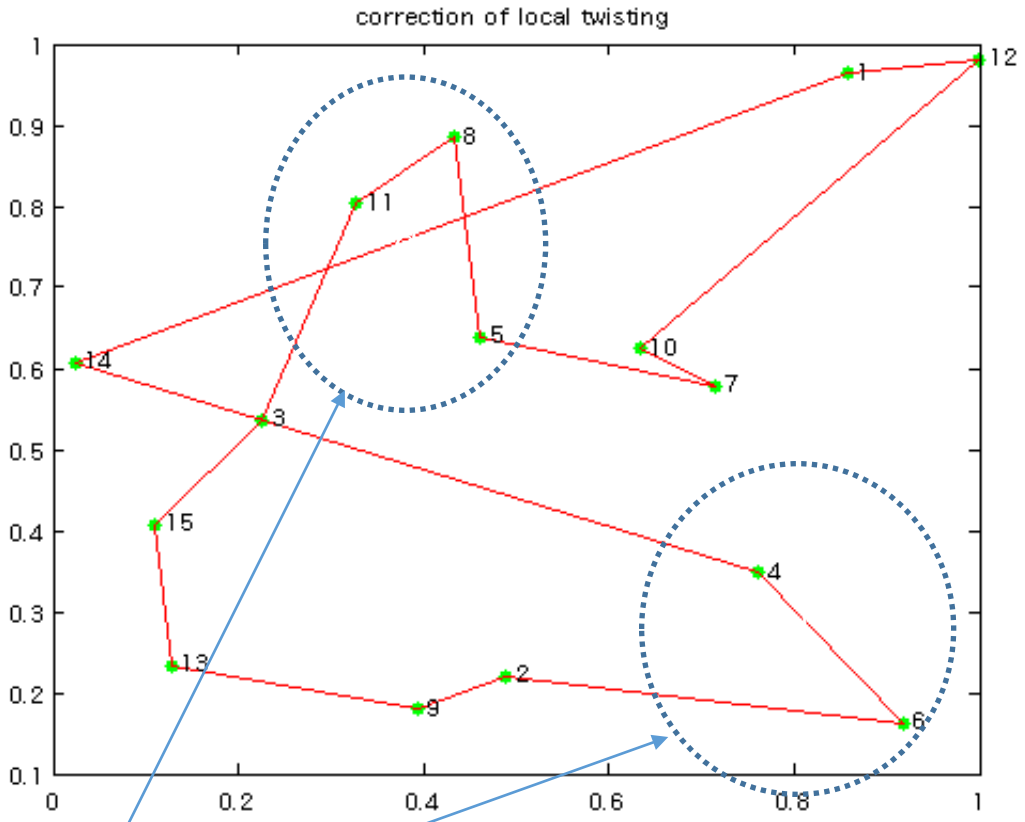
Polygon_Detect_Silver.m
（局所ねじり解消）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA009

その結果…

（results are… ）

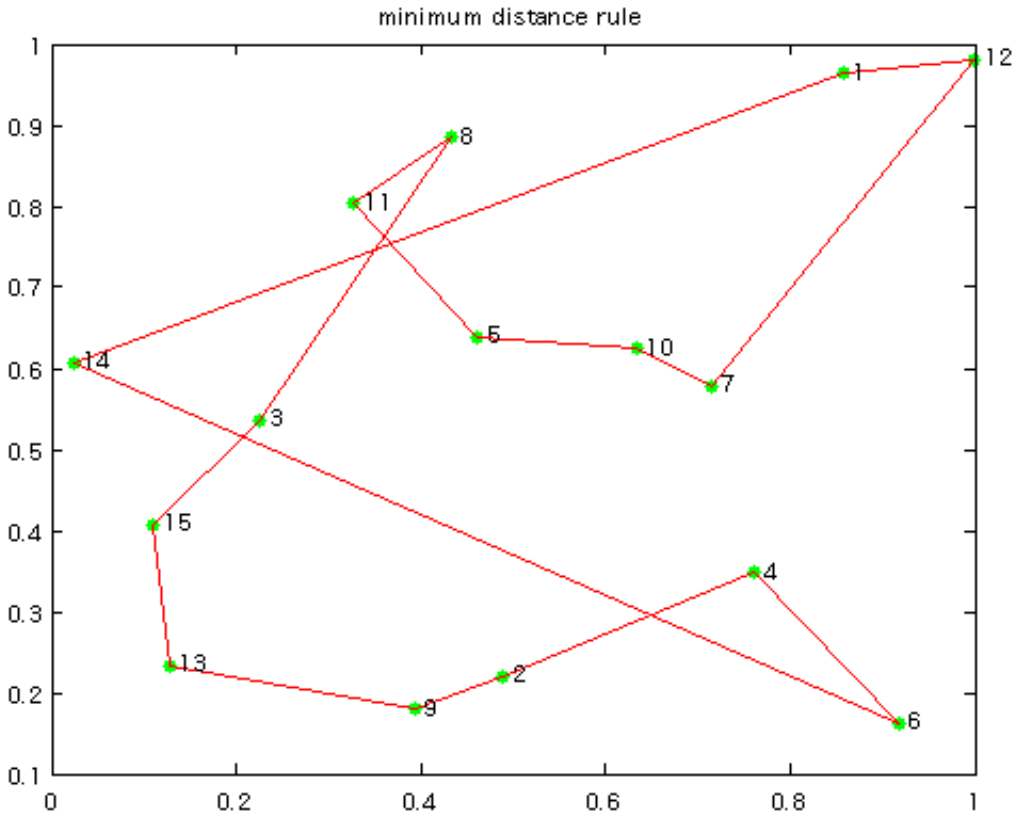


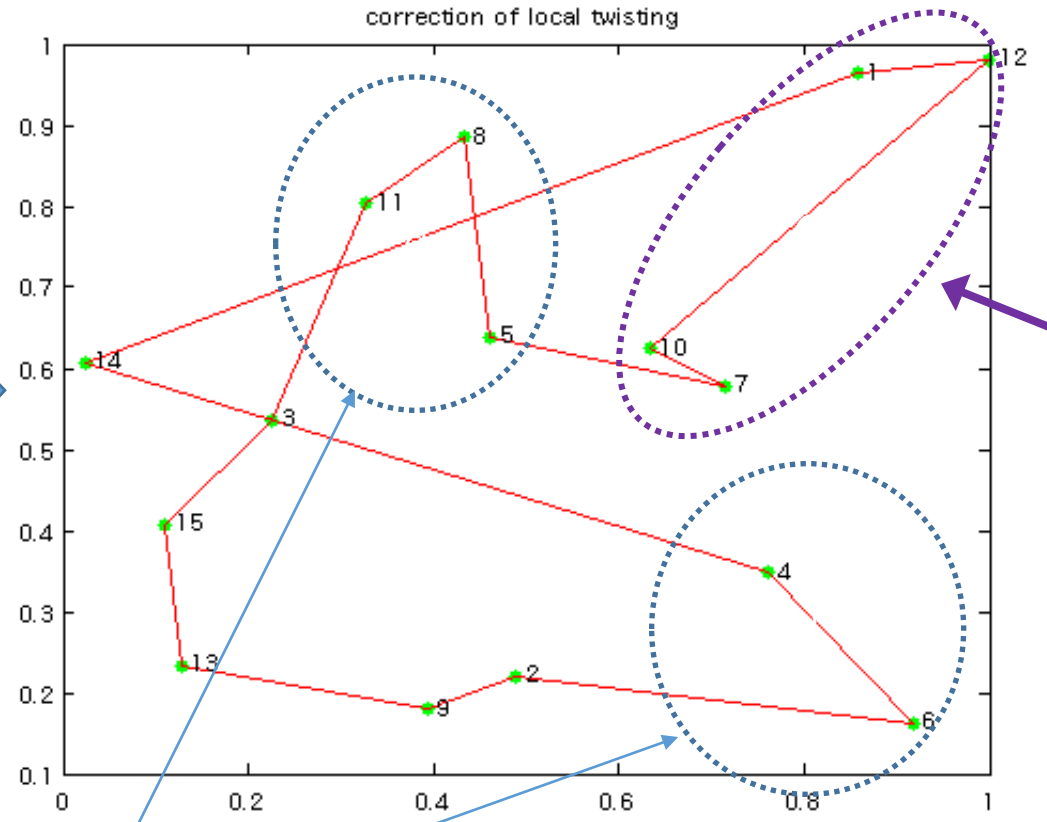


Corrected!!

その結果…

（results are… ）





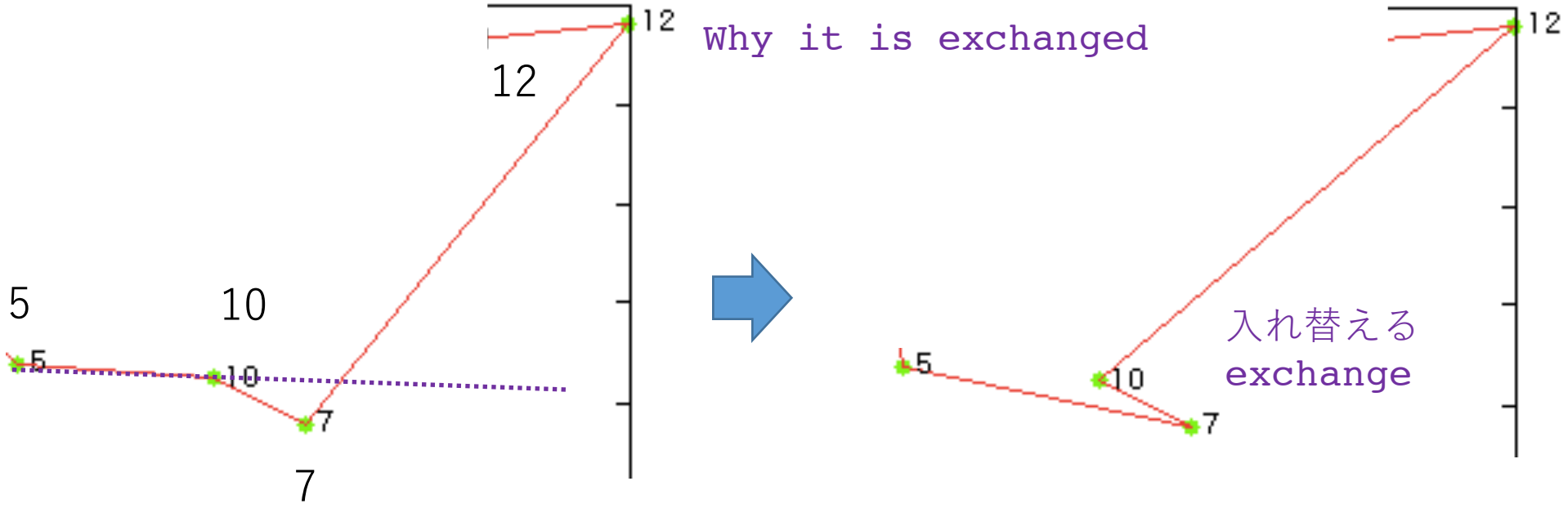
Corrected!!

なぜか、余計なところまで入れ替えている

Find something wrong …

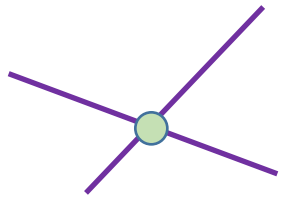
入れ替えてしまう理由は…

Why it is exchanged



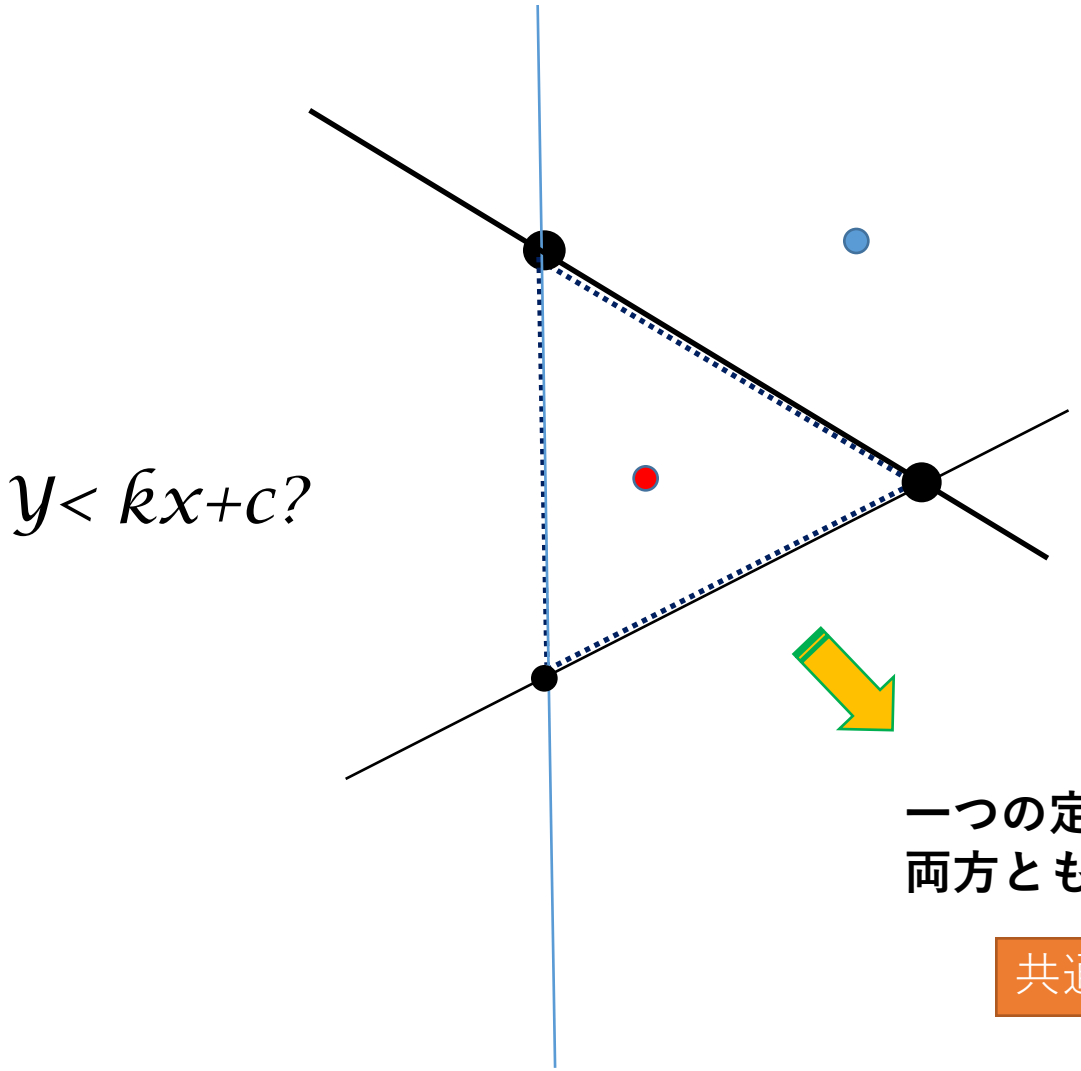
点5,10を「通過する」直線に対し、点7,12が両側に分かれて存在するため、「線が交差している」と判断

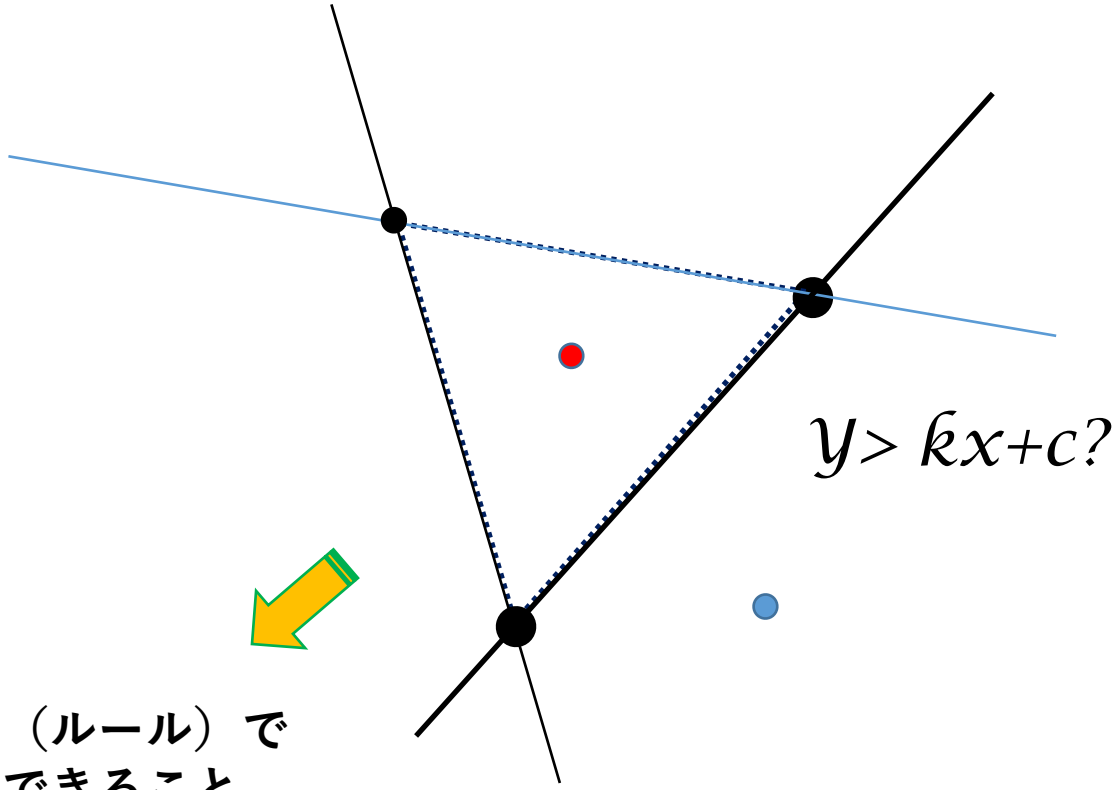
直線の上、下では十分な判定ができない
（点間の長さに限定した交差判定が必要）



（前回のスライド）　不等式の組み合わせ

まず、一本だけ考えてみる
⇒ 後で、３本の条件の「&」を取れば良い





一つの定式化（ルール）で
両方とも識別できること

共通性の抽出

アルゴリズム設計の基本の「き」：常に一般化を考える

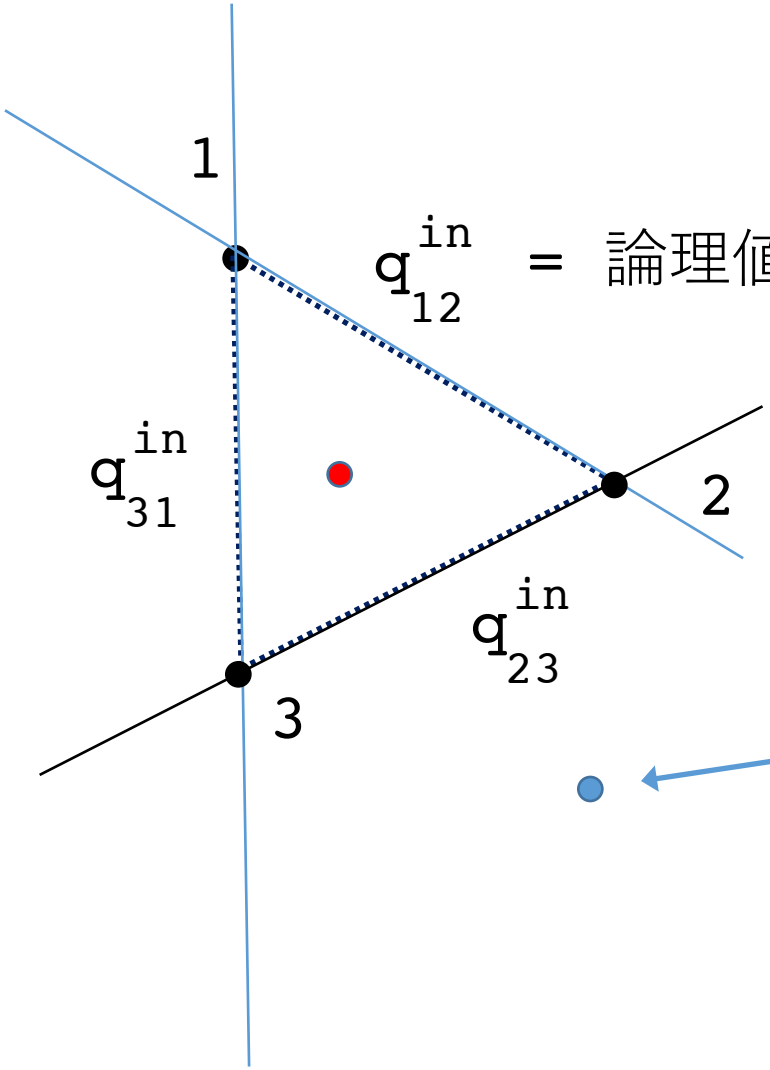
（前回のスライド）　不等式の組み合わせ

$y > kx+c$?

$q_{12}^{in} \wedge q_{23}^{in} \wedge q_{31}^{in}$

$1 \wedge 1 \wedge 1 = 1$

$1 \wedge 0 \wedge 1 = 0$



q_{12}^{in} ＝ 論理値：直線12の内側にあるかどうか

TRUE：内側にある
FALSE：ない（外側）

Q_{23}^{in} =0 (FALSE)

2.3 交点検出を正確におこなう（detect a crossing point）

2 点 (x_1, y_1), (x_2, y_2) を通る直線の方程式は

$$y - y_1 = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) \Rightarrow (y - y_1)(x_2 - x_1) = (y_2 - y_1)(x - x_1)$$

2 点 (x_3, y_3), (x_4, y_4) を通る直線の方程式は

$$y - y_3 = \frac{(y_4 - y_3)}{(x_4 - x_3)}(x - x_3) \Rightarrow (y - y_3)(x_4 - x_3) = (y_4 - y_3)(x - x_3)$$

したがって、交点は

$$\begin{aligned} & (y - y_1)(x_2 - x_1) = (y_2 - y_1)(x - x_1) \\ -) \; & (y - y_3)(x_4 - x_3) = (y_4 - y_3)(x - x_3) \end{aligned}$$

の連立方程式を解けばよい（手で解いてよいのだが…）

2.4 連立方程式の解析解を求める（to get the analytic solution）

MATLAB におけるシンボリック計算

Perform Symbolic Computations - MATLAB

https://www.mathworks.com/help/symbolic/product-description.html?lang=en

```
clear all
clc
N=4;

syms x y
xj=sym('x',[1 N]);
yj=sym('y',[1 N]);
f1= (y-yj(1))*(xj(2)-xj(1)) == (x-xj(1))*(yj(2)-yj(1));
f2= (y-yj(3))*(xj(4)-xj(3)) == (x-xj(3))*(yj(4)-yj(3));

[x,y]=solve(f1,f2,x,y)
```

という方法もある…

シンボリック計算の出力結果 (results of the symbolic calculation)

```
>>
x =(x1*x3*y2 - x2*x3*y1 - x1*x4*y2 + x2*x4*y1 - x1*x3*y4 + x1*x4*y3 + x2*x3*y4
- x2*x4*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2)

y =(x1*y2*y3 - x2*y1*y3 - x1*y2*y4 + x2*y1*y4 - x3*y1*y4 + x4*y1*y3 + x3*y2*y4
- x4*y2*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2)
```

少し整理して計算処理をすると

シンボリック計算の定義式（definition of equations ）

分子

numerator
=(yj(2)-yj(1))*(xj(4)-xj(3))*xj(1)+(yj(4)-yj(3))*(xj(2)-xj(1))*xj(3)+(yj(3)-yj(1))*(xj(2)-xj(1))*(xj(4)-xj(3))

分母

denominator=(yj(2)-yj(1))*(xj(4)-xj(3))-(yj(4)-yj(3))*(xj(2)-xj(1));

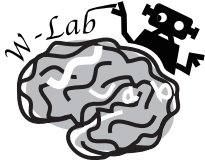
が得られる

関数形式で記述すると

数値計算につかうための関数定義式（but, the definition of function for numerical procedure is in the different form ）

```
nume=@(P1,P2,P3,P4) P1(1)*(P3(1) - P4(1))*(P1(2) - P2(2)) - P3(1)*(P1(1) -
P2(1))*(P3(2) - P4(2)) - (P1(1) - P2(1))*(P3(1) - P4(1))*(P1(2) - P3(2));

denom=@(P1,P2,P3,P4) (P3(1) - P4(1))*(P1(2) - P2(2)) - (P1(1) - P2(1))*(P3(2) - P4(2));
```

Perform Symbolic Computations - MATLAB



alcalcLine20161205.zip（連立方程式の解法＋TeX表現変換器）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA014

```
N=4;

syms x0 y0
xj=sym('x',[1 N]);
yj=sym('y',[1 N]);
vAll=[x0; y0; x(:); y(:); xj(:); yj(:)];
```

xj and yj are assigned as algebraic variables for the following solver

All the variables are collected as the set of variables, vAll

Algebraic Equations and Systems
Solve algebraic equations and systems of such equations analytically

```
f1= (y0-yj(1))*(xj(2)-xj(1)) == (x0-xj(1))*(yj(2)-yj(1));
f2= (y0-yj(3))*(xj(4)-xj(3)) == (x0-xj(3))*(yj(4)-yj(3));

[x0,y0]=solve(f1,f2,x0,y0);

expand(x0);
```

代数方程式 (algebraic equation)

連立方程式f1,f2を変数x0,y0で解く（simultaneous equations (f1,f2) will be solved with respect to variable x0, y0)

expand the solution equation

```
denominator=(yj(2)-yj(1))*(xj(4)-xj(3))-(yj(4)-yj(3))*(xj(2)-xj(1));

convert2TeXF_p(denominator,vAll)
```

a part of the solution equation

will be converted to the TeX description with respect to a set of variables, vAll

シンボリック計算の出力結果 (results of the symbolic calculation)

ans =

(x1*x3*y2)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x2*x3*y1)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x1*x4*y2)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x2*x4*y1)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x1*x3*y4)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x1*x4*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x2*x3*y4)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x2*x4*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2)

ans =

(x1*x3*y2)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x2*x3*y1)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x1*x4*y2)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x2*x4*y1)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x1*x3*y4)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x1*x4*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) + (x2*x3*y4)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2) - (x2*x4*y3)/(x1*y3 - x3*y1 - x1*y4 - x2*y3 + x3*y2 + x4*y1 + x2*y4 - x4*y2)

{¥bm x_b =

TeX形式への変換結果 (converted form for TeX compiler)

ans = x_1¥, ¥left(x_3 - x_4¥right)¥, ¥left(y_1 - y_2¥right) - x_3¥, ¥left(x_1 - x_2¥right)¥, ¥left(y_3 - y_4¥right) - ¥left(x_1 - x_2¥right)¥, ¥left(x_3 - x_4¥right)¥, ¥left(y_1 - y_3¥right)

ans = ¥left(x_3 - x_4¥right)¥, ¥left(y_1 - y_2¥right) - ¥left(x_1 - x_2¥right)¥, ¥left(y_3 - y_4¥right)

2.5 解析解から関数形式への変換
（How does the analytic solution is systematically converted into the normal function form?）

```
numerator   = 'x1*(x3 - x4)*(y1 - y2) - x3*(x1 - x2)*(y3 - y4) - (x1 - x2)*(x3 - x4)*(y1 - y3)';

denominator = '(x3 - x4)*(y1 - y2) - (x1 - x2)*(y3 - y4)';
```

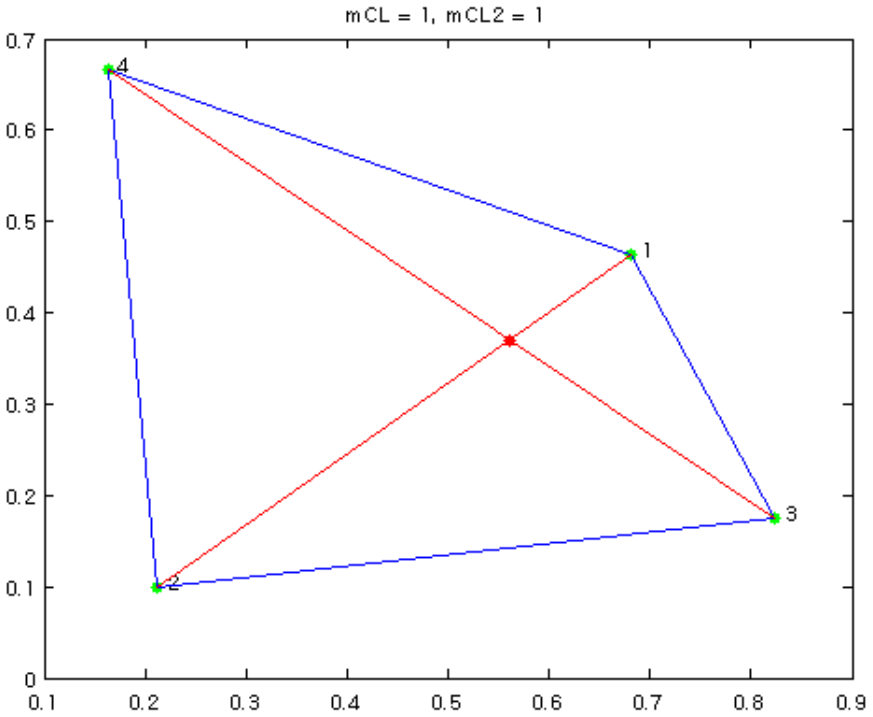
```
% denom=@(P1,P2,P3,P4) (P2(2)-P1(2))*(P4(1)-P3(1))-(P4(2)-P3(2))*(P2(1)-P1(1));

% nume=@(P1,P2,P3,P4) (P2(2)-P1(2))*(P4(1)-P3(1))*P1(1)+(P4(2)-P3(2))*(P2(1)-P1(1))*P3(1)+(P3(2)-P1(2))*(P2(1)-P1(1))*(P4(1)-P3(1));
```

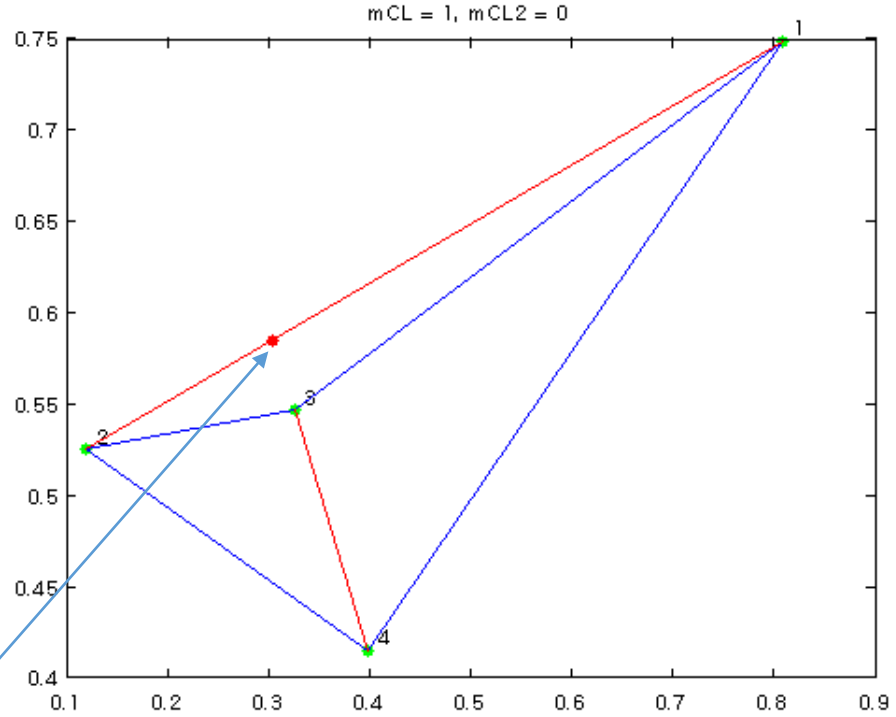
repEqRes_Plus.m（Symbolic Math解析解表現 → 通常の数値解用関数形式）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA015

```
prefix={'nume=@(P1,P2,P3,P4) ','denom=@(P1,P2,P3,P4) '};
stOri={'x1','x2','x3','x4','y1','y2','y3','y4'};
stRep={'P1(1)','P2(1)','P3(1)','P4(1)','P1(2)','P2(2)','P3(2)','P4(2)'};

stTarget={numerator,denominator};
stTarget=cellfun(@(x) [' ',x],stTarget,'UniformOutput', false);
stOri2=cell(size(stTarget));
stRep2=stOri2;

if length(stOri)==length(stRep)
    for k=1:length(stOri)
        stOri2(:)={stOri{k}}; stRep2(:)={stRep{k}};
        stTarget=cellfun(@strrep,stTarget,stOri2,stRep2,'UniformOutput', false);
    end
else
    error('Mismatch of two lengths');
end

stTarget2=cellfun(@strcat,prefix,stTarget,'UniformOutput', false);
cellfun(@display,stTarget2);
```

```
nume=@(P1,P2,P3,P4) P1(1)*(P3(1) - P4(1))*(P1(2) - P2(2)) - P3(1)*(P1(1) - P2(1))*(P3(2) - P4(2)) - (P1(1) -
P2(1))*(P3(1) - P4(1))*(P1(2) - P3(2))

denom=@(P1,P2,P3,P4) (P3(1) - P4(1))*(P1(2) - P2(2)) - (P1(1) - P2(1))*(P3(2) - P4(2))
```





交点は検出できるが、直線の長さを入れた判定は十分でない
（交点検出とは別に判定式が必要）

CrossL_Check_Copper.m（交点検出）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA011

```
Yeq=@(x,P1,P2) (P2(2)-P1(2))/(P2(1)-P1(1))*(x-P1(1))+P1(2);
```

交点検出

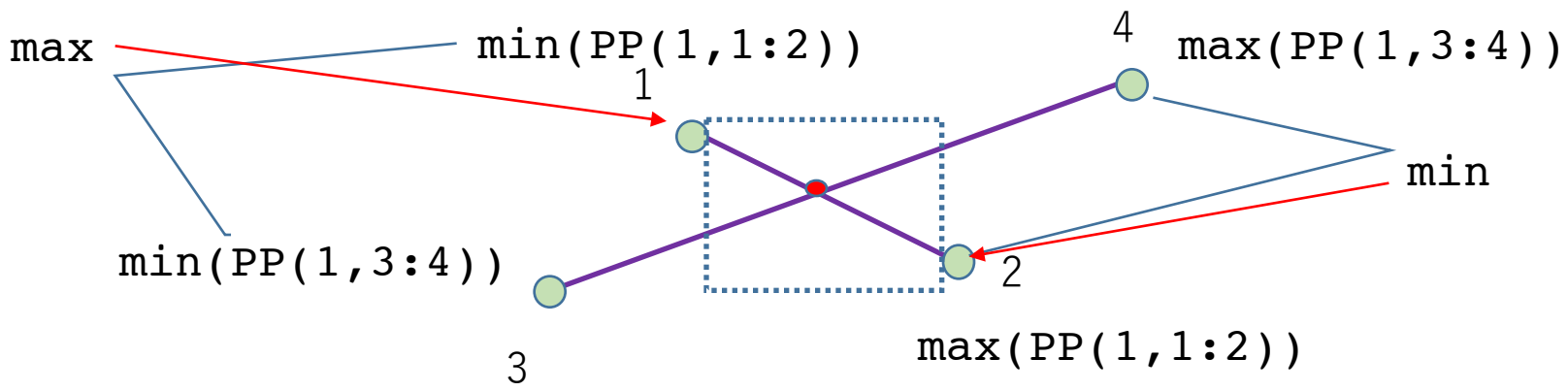
```
CrossP=@(P1,P2,P3,P4) [nume(P1,P2,P3,P4)./denom(P1,P2,P3,P4);
                       Yeq(nume(P1,P2,P3,P4)/denom(P1,P2,P3,P4),P1,P2)];
```

```
CrossL=@(P1,P2,P3,P4) nume(P1,P2,P3,P4)~=0 && (Cinq(CrossP(P1,P2,P3,P4),P4,P1,P2) ||
                                                Cinq(CrossP(P1,P2,P3,P4),P3,P1,P2));
```

二直線が平行（解無し）の判断

```
CrossL2=@(PP,Pc) max(min(PP(1,1:2)),min(PP(1,3:4)))<= Pc(1)
                                  && Pc(1) <= min(max(PP(1,1:2)),max(PP(1,3:4)));
```

交点が直線内部にあるという判断

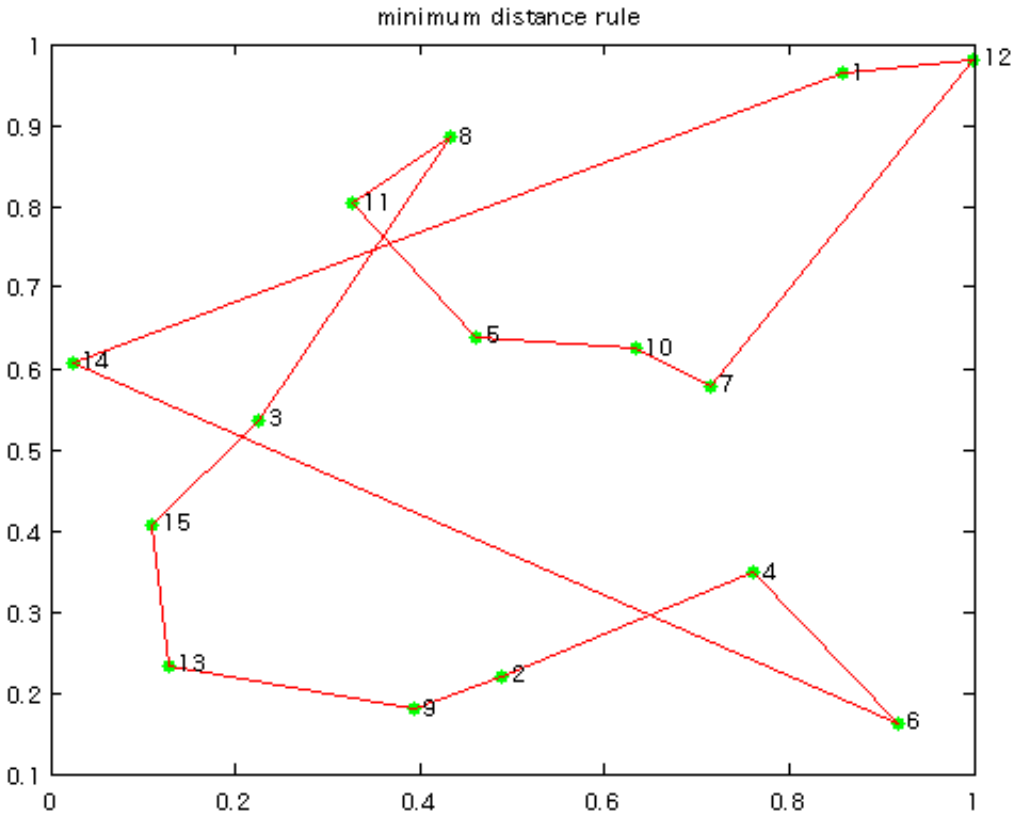


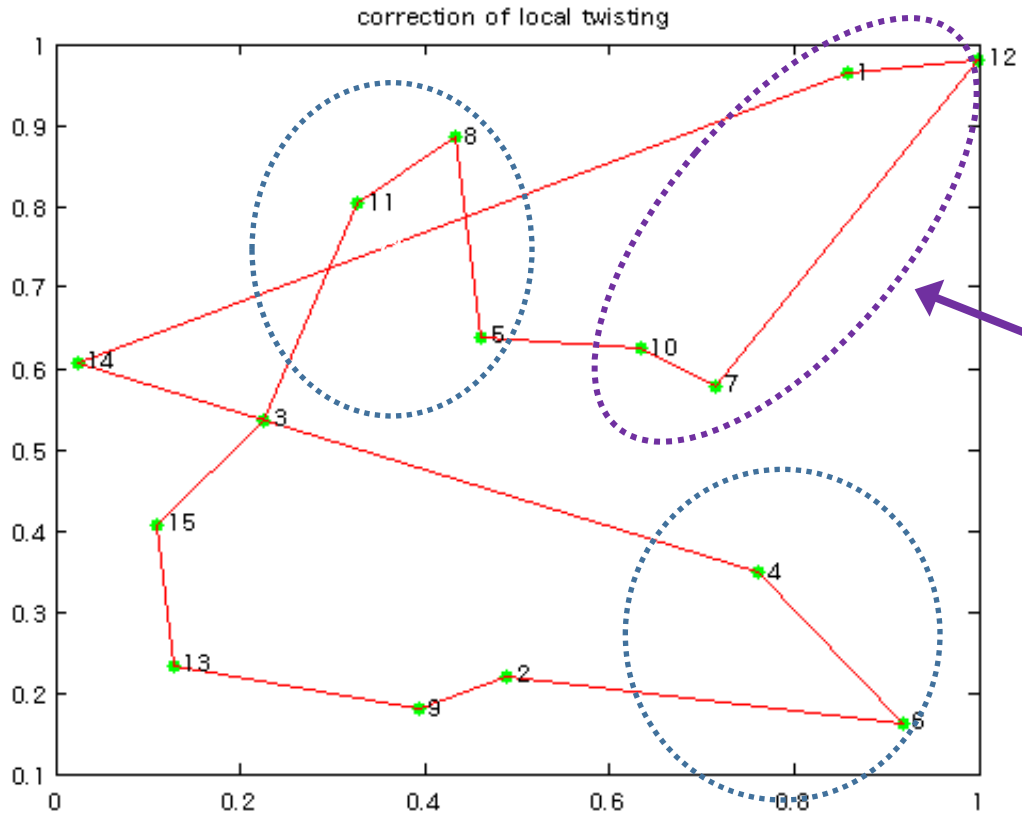
この判定式を導入すると

```
tF=[1:FPnum; FPxySort]; tFP=FPxySort; reNewOrder=NewOrder;
TargetIndex=1:FPnum;
for j=1:FPnum
    TargetWindow=TargetIndex(1:4);
    tF=FPxySort(:,TargetWindow);

    crossPoint=CrossP(tF(:,1),tF(:,2),tF(:,3),tF(:,4));

    if CrossL(tF(:,1),tF(:,2),tF(:,3),tF(:,4)) && CrossL2(tF,crossPoint)
        tFP(:,TargetWindow(2:3))=fliplr(tFP(:,TargetWindow(2:3)));
        reNewOrder(TargetWindow(2:3))=fliplr(reNewOrder(TargetWindow(2:3)));

    end
    TargetIndex=circshift(TargetIndex,[0 -1]);
end
```

Polygon_Detect_Gold.m（局所ねじりの解決2）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA012





今度は、問題が発生しない

Corrected properly …

この判定式をもう少し整理すると

Polygon_Detect_Gold2.m（局所ねじりの解決2-1・visibility up）

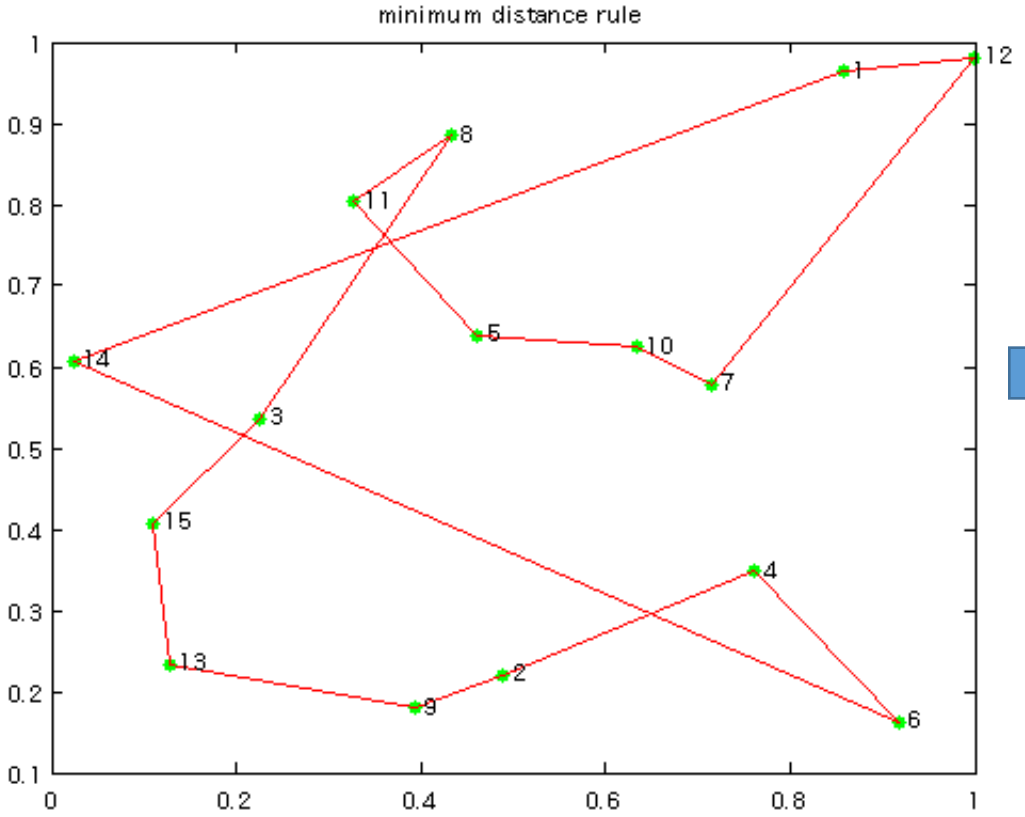
https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA013

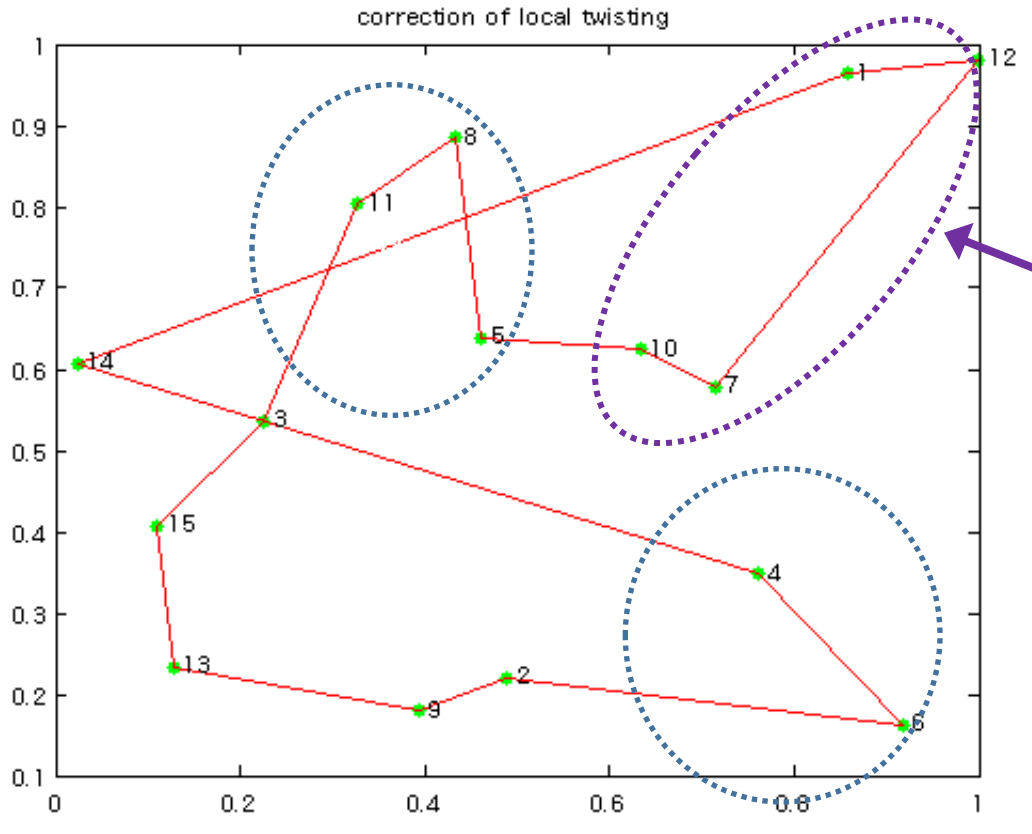
```
inq=@(P,P1,P2) P(2)-P1(2)>(P2(2)-P1(2))/(P2(1)-P1(1))*(P(1)-P1(1));
Cinq=@(cPa,cPb,P1,P2) inq(cPa,P1,P2)==inq(cPb,P1,P2);

Yeq=@(x,P1,P2) (P2(2)-P1(2))/(P2(1)-P1(1))*(x-P1(1))+P1(2);

% ============
nume_x=@(P) P(1,1)*(P(1,3) - P(1,4))*(P(2,1) - P(2,2)) - P(1,3)*(P(1,1) -
P(1,2))*(P(2,3) - P(2,4)) - (P(1,1) - P(1,2))*(P(1,3) - P(1,4))*(P(2,1) - P(2,3));

denom_x=@(P) (P(1,3) - P(1,4))*(P(2,1) - P(2,2)) - (P(1,1) - P(1,2))*(P(2,3) -
P(2,4));

Yeq=@(x,P1,P2) (P2(2)-P1(2))/(P2(1)-P1(1))*(x-P1(1))+P1(2);
CrossP_x=@(P) [nume_x(P)./denom_x(P); Yeq(nume_x(P)/denom_x(P),P(:,1),P(:,2))];

checkParallel=@(P) nume_x(P)~=0 && (Cinq(CrossP_x(P),P(:,4),P(:,1),P(:,2)) ||
                                    Cinq(CrossP_x(P),P(:,3),P(:,1),P(:,2)));
findCross=@(PP,Pc) max(min(PP(1,1:2)),min(PP(1,3:4)))<= Pc(1) && Pc(1) <=
                   min(max(PP(1,1:2)),max(PP(1,3:4))); % for x validation
% ============
```

```
tFP=FPxySort; reNewOrder=NewOrder;
TargetIndex=1:FPnum;
for j=1:FPnum
    TargetWindow=TargetIndex(1:4);
    tF=FPxySort(:,TargetWindow);

    if denom_x(tF)~=0
        crossPoint=CrossP_x(tF);
        if  findCross(tF,crossPoint)

            display(['[',num2str(NewOrder(TargetWindow(2))),...
               '<==>',num2str(NewOrder(TargetWindow(3))),']']);

            tFP(:,TargetWindow(2:3))=fliplr(tFP(:,TargetWindow(2:3)));
            reNewOrder(TargetWindow(2:3))=fliplr(reNewOrder(TargetWindow(2:3)));
        end
    end
    TargetIndex=circshift(TargetIndex,[0 -1]);
end
```

Polygon_Detect_Gold2.m（局所ねじりの解決2-1・visibility up）

https://dynamicbrain.neuroinf.jp/modules/xoonips/detail.php?id=GeoCalcA013





しかし、矩形領域抽出という根本的な問題は解決したのか？

今度は、問題が発生しない

Corrected properly …

今日のまとめ

×いいアイディアが空から降ってくるわけではない

×漠然とした思考では解決策は見えない

×有名な理論が解決してくれるという他力本願もダメ
（自分で分析して、問題の根源をつかまないと本質には届かない）

1. 問題を発見したら、仮説検証型で取り組む（hypothesis driven approach）
2. 過去に成功した方法を試すのは良いが、使えなかったら、執着を捨て、新しい方法を構築すること（do not stick your past idea, and rethink from the beginning ）

×過去の成功に執着しない

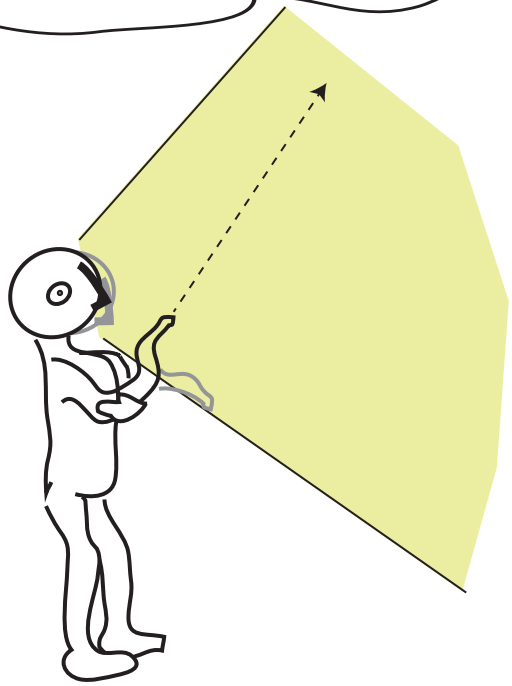
3. 局所的な方法（ボトムアップ）で解決しないときには（するなら、すべての場合で可能という証明が必要）、問題の描像を俯瞰的に見て、根本的な解決策を模索すること

（bottom-up solution do not always solve the problem, and the coupling with the top-down view is necessary ）

◯ いつも全体を見ること

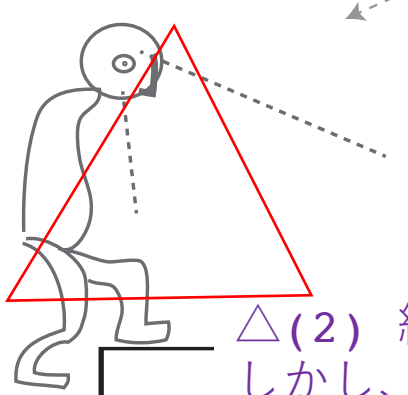
×(1) アイディアは空からは降ってきません。技術のないところに根本解決はないからです。
Don’t believe that a good idea falls down to you from Heaven. No art, no fundamental completion.

？(3) 内容を熟知せずに有名な理論に頼ってはいけません。また、むやみに探しても徒労です。誇大評価と過剰な期待に翻弄されます。欠点のない理論や技術は存在しません。
There is no perfect theory and technique. Don’t rely on those before understanding inside. You will be swayed by excess rumors and expectations.



(o) 上を見ながら、下を見る。多視点を持つことは言うまでもありませんが、目的と手段を整合的に取り扱うかが課題です。手段が目的になっても発展がなくつまらないですし、目的ありきで、手段を無理やり当てはめると歪みを生じます。一番良いのは、過去の道具を徹底して分解し、目的に合わせて再構築するプロセスです。Focus on the integration of the basement (bottom-up thinking) and vision/target/aim of what you really want (top-down thinking).

大切なことは、分解・再構築しやすい理論、技術を創っておくことです。



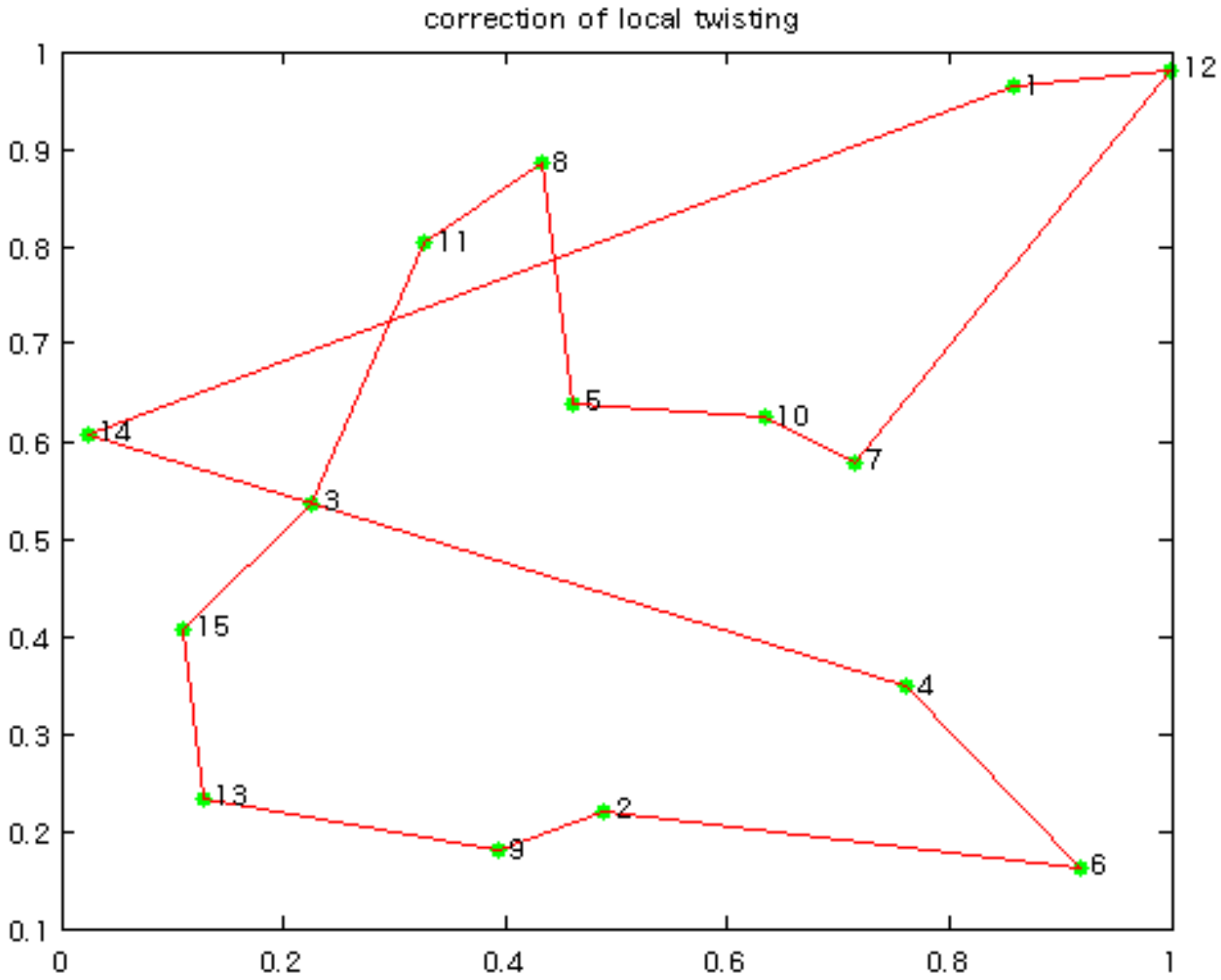
△(2) 経験と知識、技術の積み上げが問題を解決します。しかし、過去の事例に囚われすぎると前に進めません
There is no doubt that past experience and knowledge help you. But don’t stick too much your past successful evidences. It sometime narrows your eyesight.

今日の課題はここまで。

ですが、次回の予告と、

宿題をちょっと…

局所的に点を結ぶと
ねじれが発生する
可能性があります。

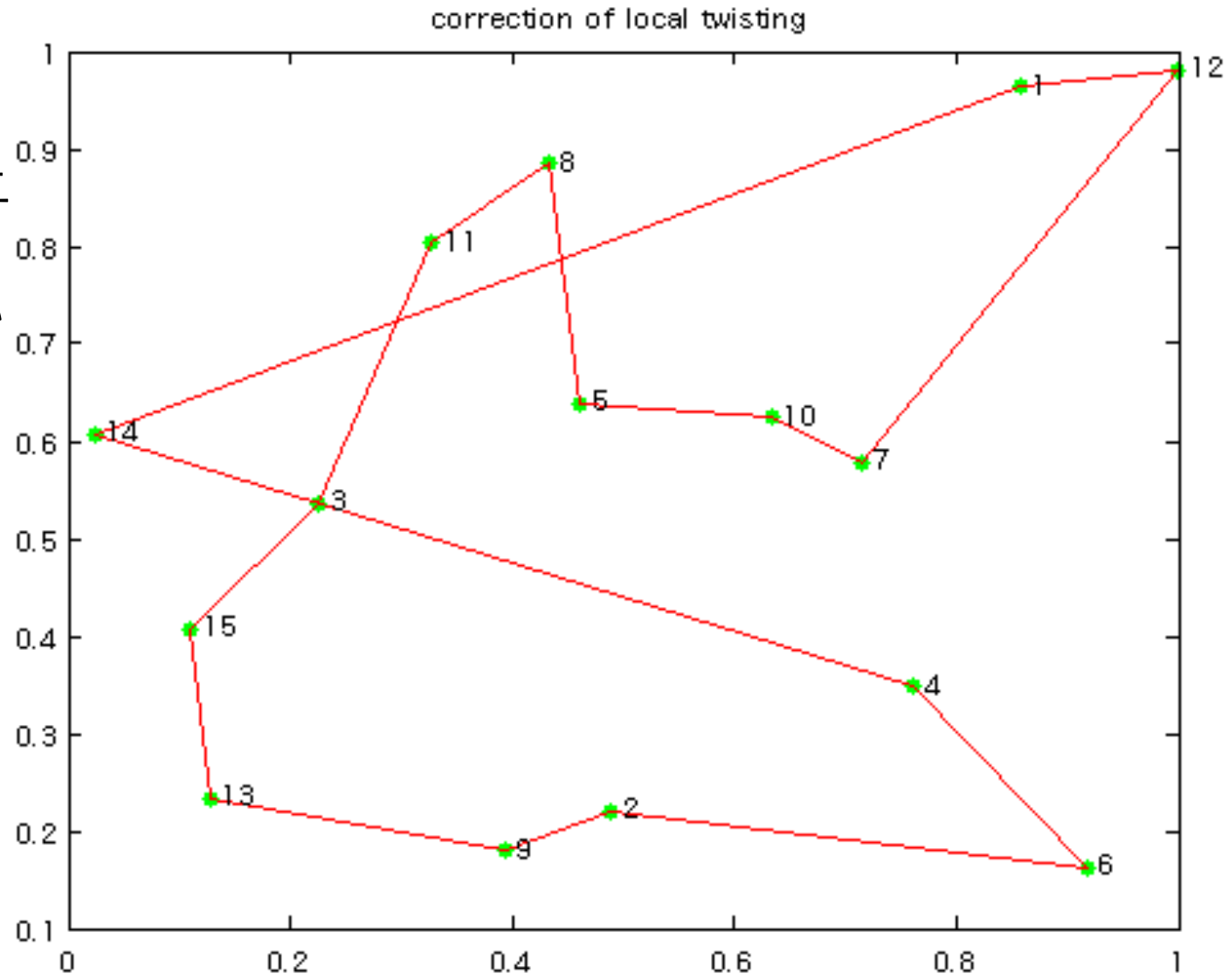


さて、どういう条件のときにねじれないのでしょう。

i) 点の数？

ii) 一筆書き経路

は任意の場合に得られるのでしょうか？



では、いったんすべての点を扱うのはやめて、外皮として点をつなぐ直線を考えてみましょう。このとき**その経路で描いた図形は凸包（凹みがない）になるでしょうか？**　それはどのようなアルゴリズムによって得られるでしょうか？（解が一意に定まらないときには、アルゴリズムによる自動化はできません。）

i) 方法（アルゴリズム）
ii) 一筆書き経路への展開は？