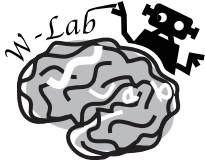
九州工業大学大学院生命体工学科
我妻研究室



幾何学を計算する：ポリゴンを扱ってみよう（1）

講師：我妻 広明 *1,*2,*3

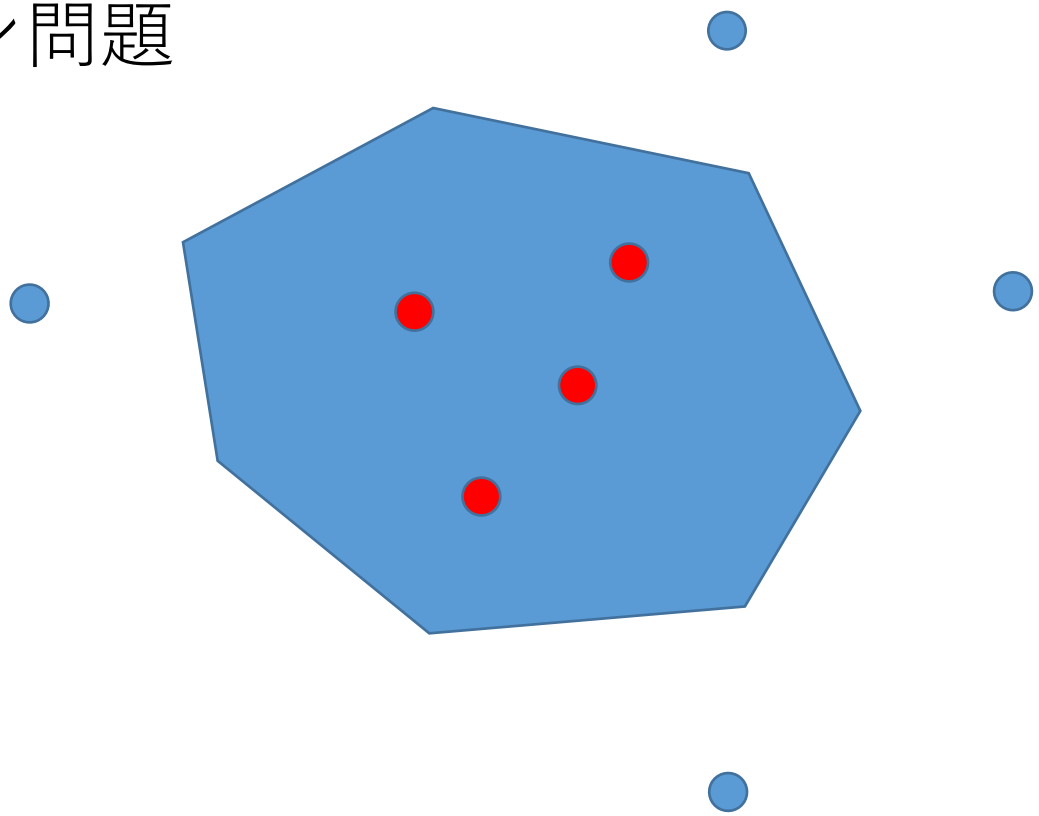
*1 九州工業大学大学院生命体工学科

*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

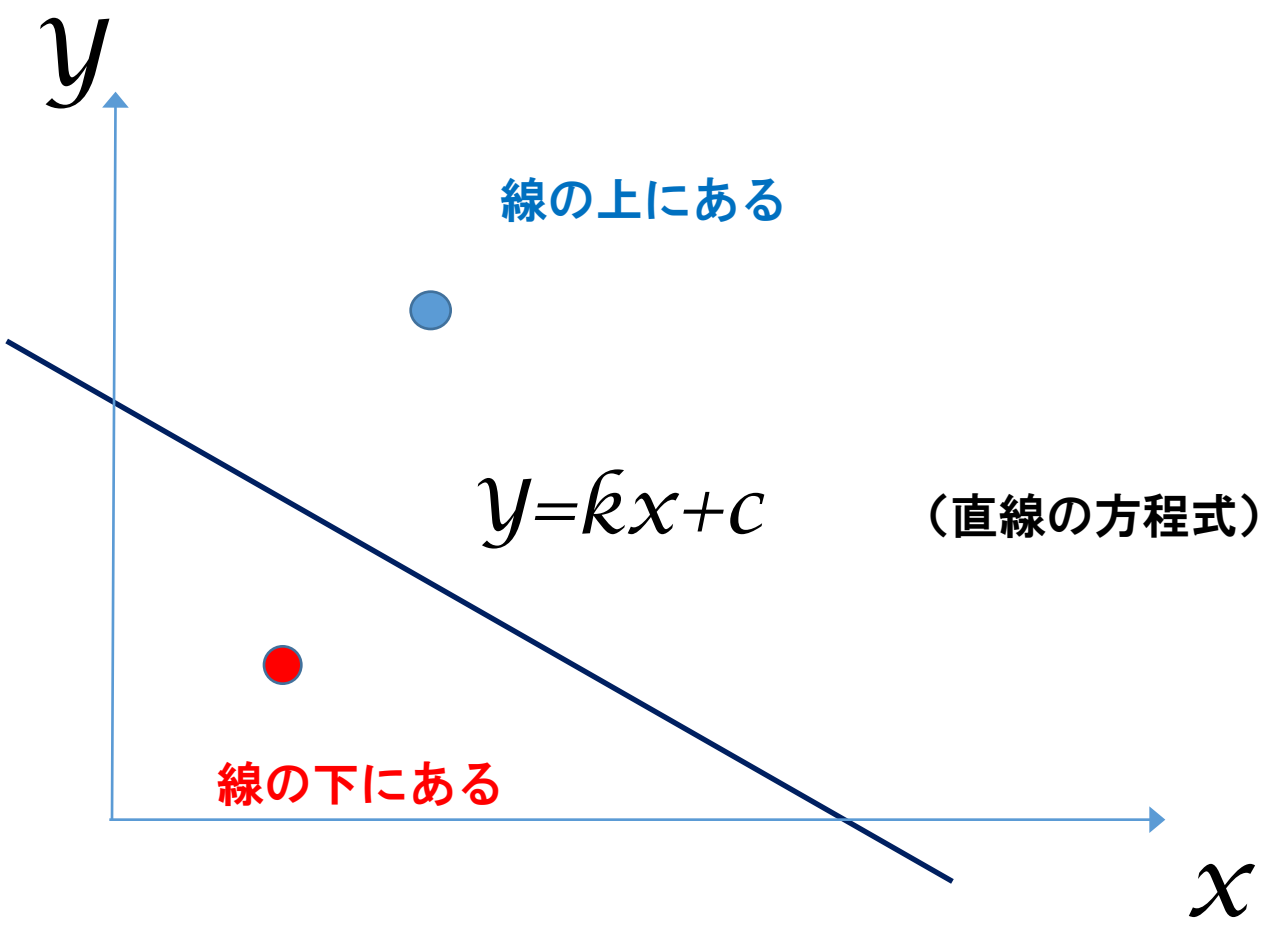
問題を解くステップ

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

まず、直線の方程式から復習



MATLABの行列表現の「きほんのき」

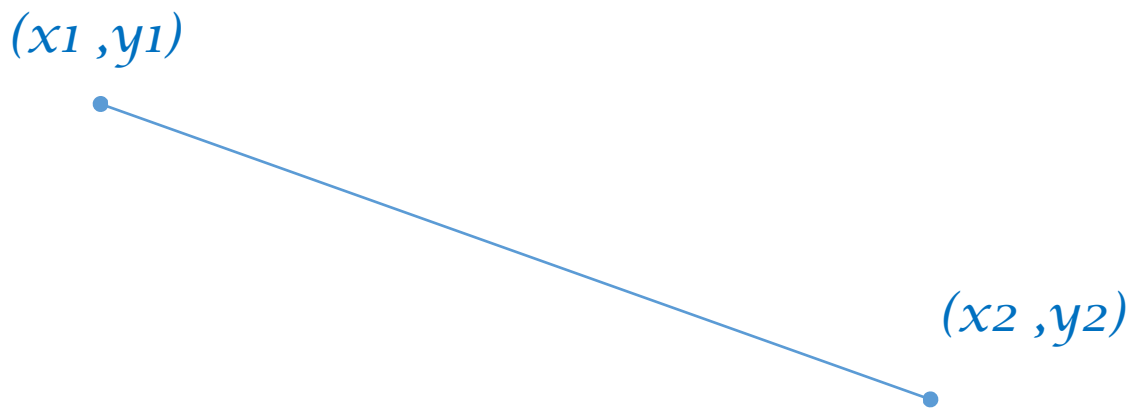
```
clear all;

Points=[0,2; 5,0];
xP=Points(:,1);
yP=Points(:,2);
```

```
>> Points

Points =

     0     2
     5     0
```

```
>> xP=Points(:,1)

xP =

     0     x1
     5     x2
```

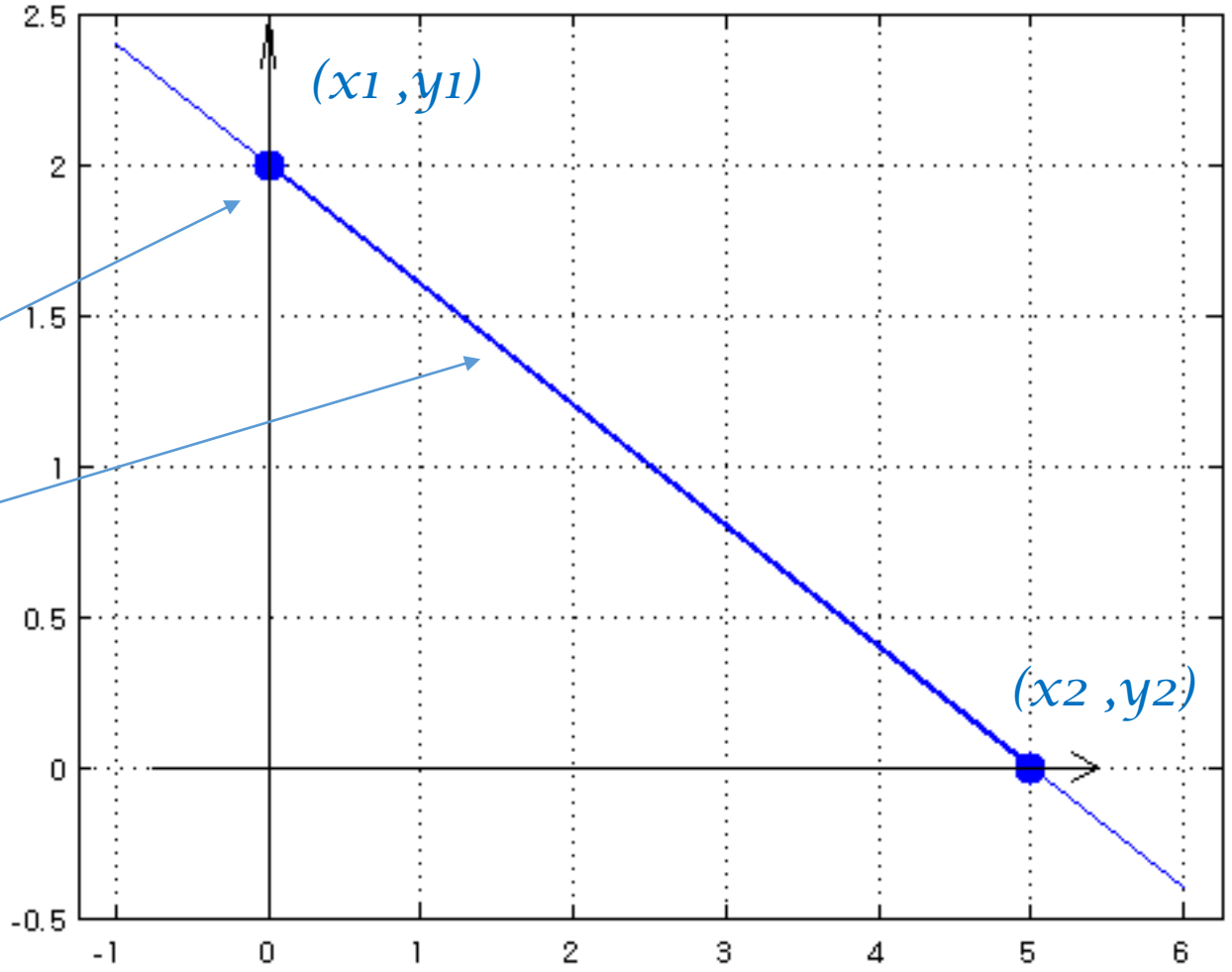
```
>> Points(1,:)

ans =

          x1        y1
     0         2
```

```
figure(10); clf
plot(xP,yP,'.-','MarkerSize',22,'LineWidth',2);
grid on
```



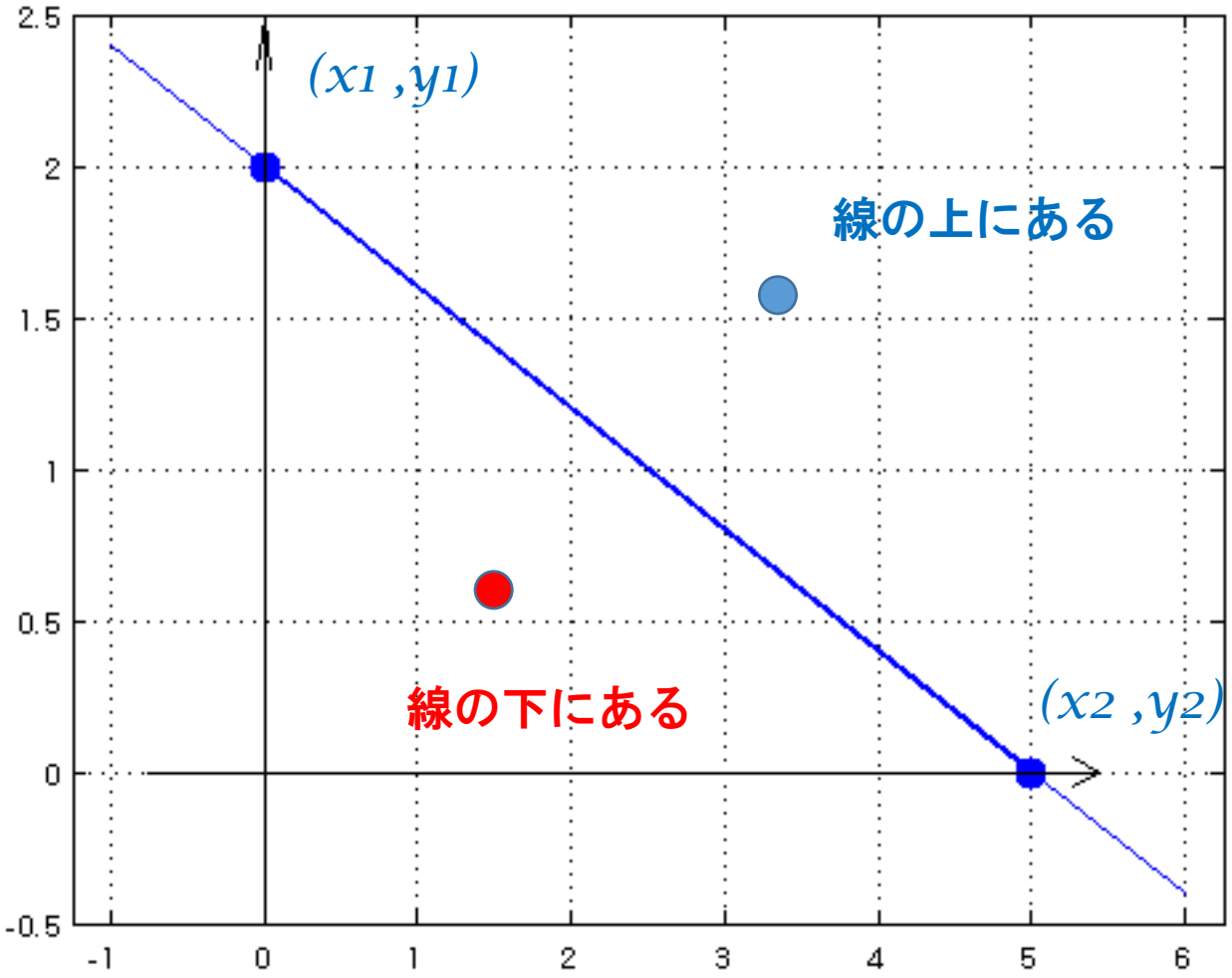
MATLABによる描画

```
clear all;

Points=[0,2; 5,0];
xP=Points(:,1);
yP=Points(:,2);


figure(10); clf
plot(xP,yP,'.-
','MarkerSize',22,'LineWidth'
,2);
grid on
```



LinePlotSample1.m（ログインユーザ・ダウンロード）

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」

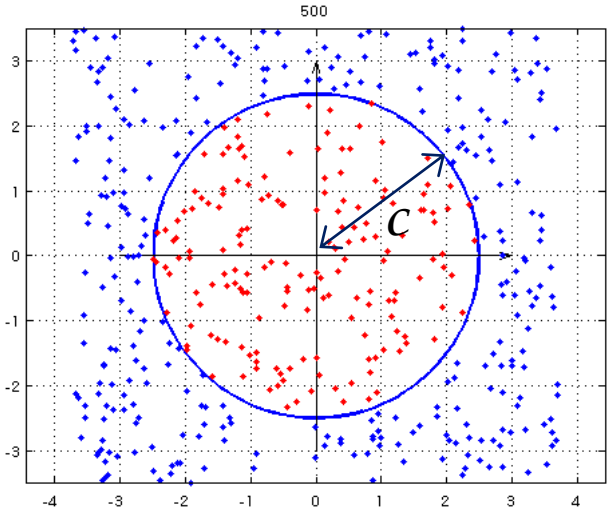
URL：
https://dynamicbrain.neuroinf.jp/modules/mediawiki/index.php?title=幾何学計算を考える_2016&style=m&ml_lang=ja

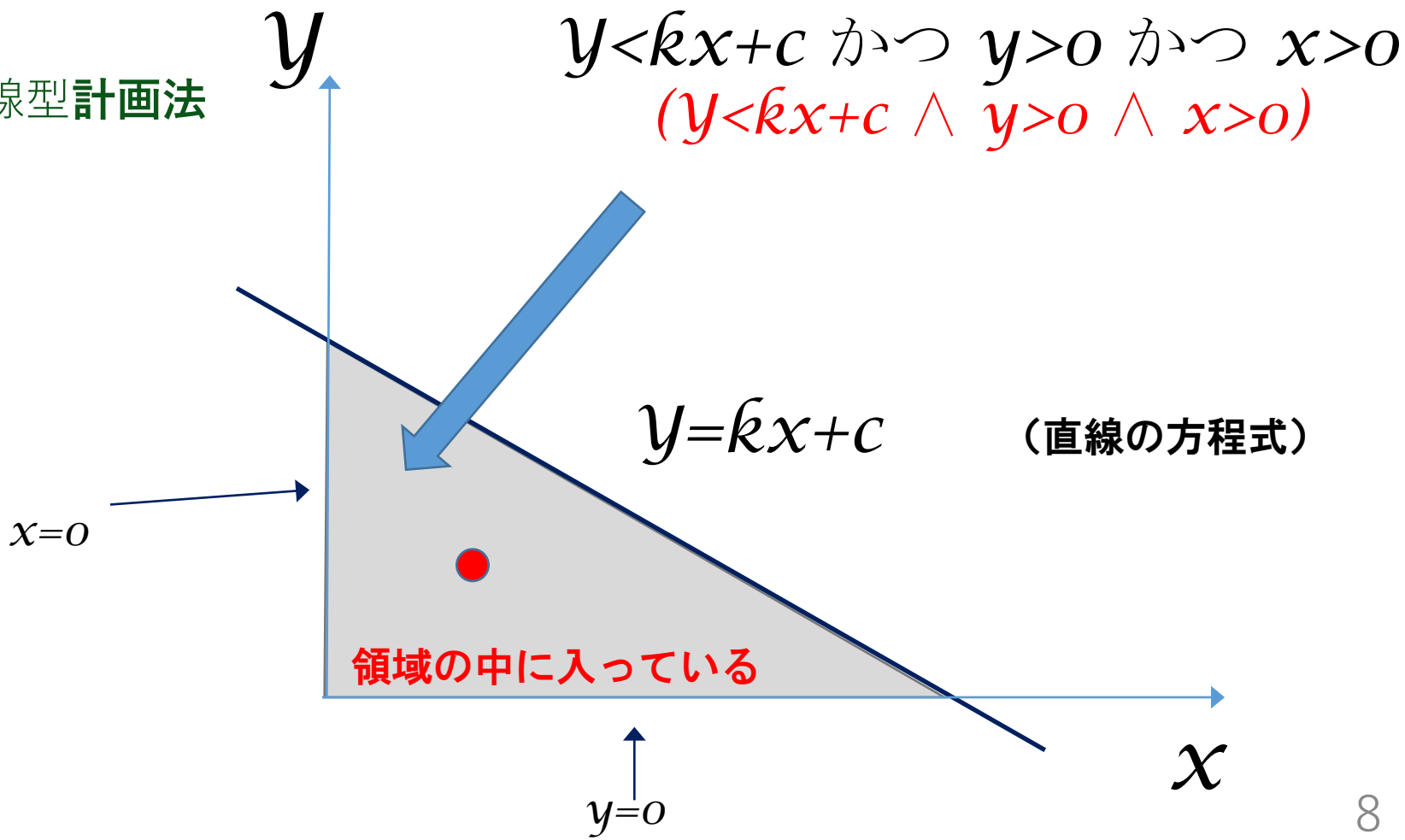
LinePlotSample1.m（ログインユーザ・ダウンロード）

線型計画法（せんけいけいかくほう LP; linear programming ）とは、いくつかの1次不等式および1次等式を満たす変数の値の中で、ある1次式を最大化または最小化する値を求める方法である。 **線型計画問題**を解く手法。

線型計画法 - Wikipedia
https://ja.wikipedia.org/wiki/線型**計画法**

例: $x^2+y^2< c^2$

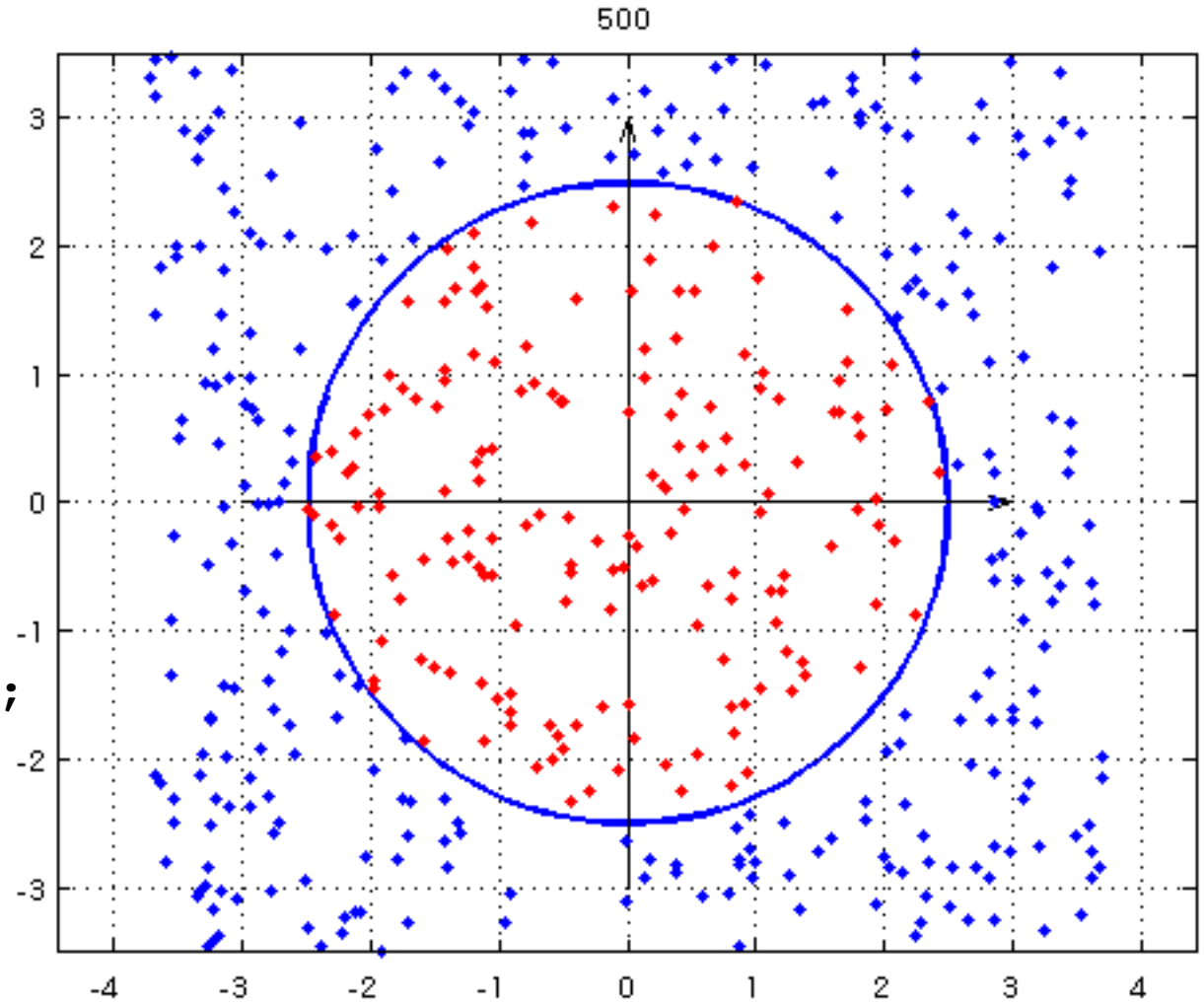




領域区分の単純な例：MATLABプログラミングの事例とともに

1. 枠線を描く

```
step=0.05; cR=2.5;

Points=0:step:pi*2+step;
xP=cR*cos(Points);
yP=cR*sin(Points);

figure(1); clf
plot(xP,yP,'b-','LineWidth',2),hold on;
```



CirclePlotSample3.m（円形領域抽出）

2. 点を区別する

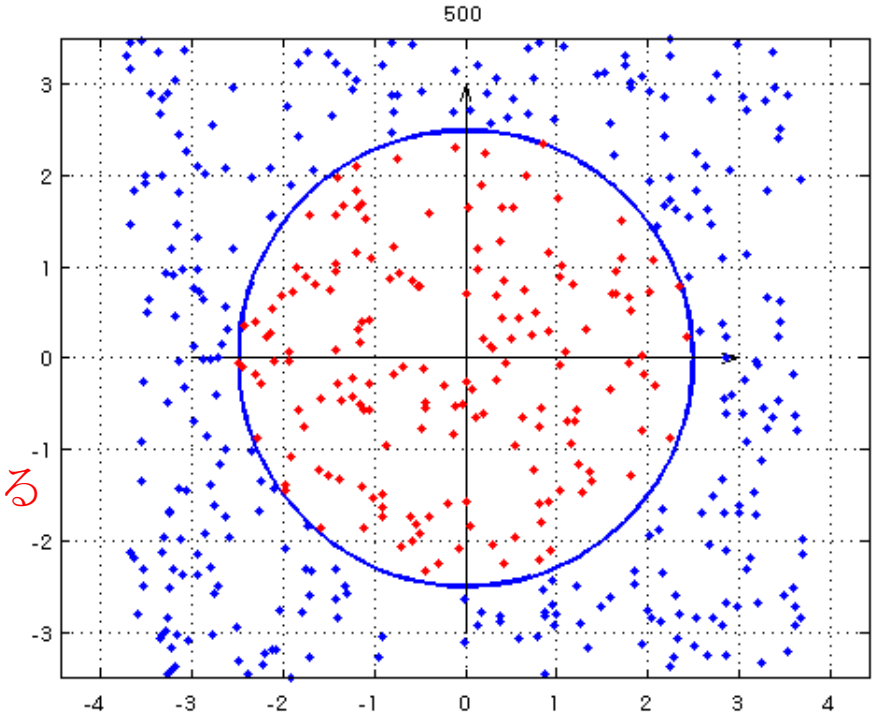
CirclePlotDataFind.m（円形領域抽出のデータ検索部分）

```
N=500;

rPoints=3*cR*(rand(N,2)-0.5);
% 0<rand<1; uniformly random variable

for i=1:N
    xp=rPoints(i,1); yp=rPoints(i,2);
    if (xp.^2+yp.^2)<=cR.^2
        cL=[1 0 0];
    else
        cL=[0 0 1];
    end
    plot(rPoints(i,1),rPoints(i,2),'.','MarkerSize',10,'Color',cL),hold on;
    title(i);
    drawnow;
end
```

点の色を変える

Red: [R:1 G:0 B:0]

Blue:[R:0 G:0 B:1]



描画だけでなく、データの詳細を知りたいときは…

```
key = find((xp.^2+yp.^2)<=cR.^2)
```

など、find関数を使う

データの詳細を取り出す：データのIDや中身（座標など）

```
% You need to run "CirclePlotSample3.m" in the first place

xp=rPoints(:,1); yp=rPoints(:,2);
key=find((xp.^2+yp.^2)<=cR.^2);

figure(1); clf; dSizeOri=length(xp);
plot(1:dSizeOri,xp,1:dSizeOri,yp), hold on;
plot(1:dSizeOri,cR*ones(1,dSizeOri),'r-');
plot(1:dSizeOri,-cR*ones(1,dSizeOri),'r-');

set(gca,'ylim',[min([xp; yp]) max([xp; yp])]);

figure(2); clf; dSizeSel=length(key);
plot(1:dSizeSel,xp(key),1:dSizeSel,yp(key)), hold on;
plot(1:dSizeSel,cR*ones(1,dSizeSel),'r-');
plot(1:dSizeSel,-cR*ones(1,dSizeSel),'r-');

set(gca,'ylim',[min([xp; yp]) max([xp; yp])]);
```

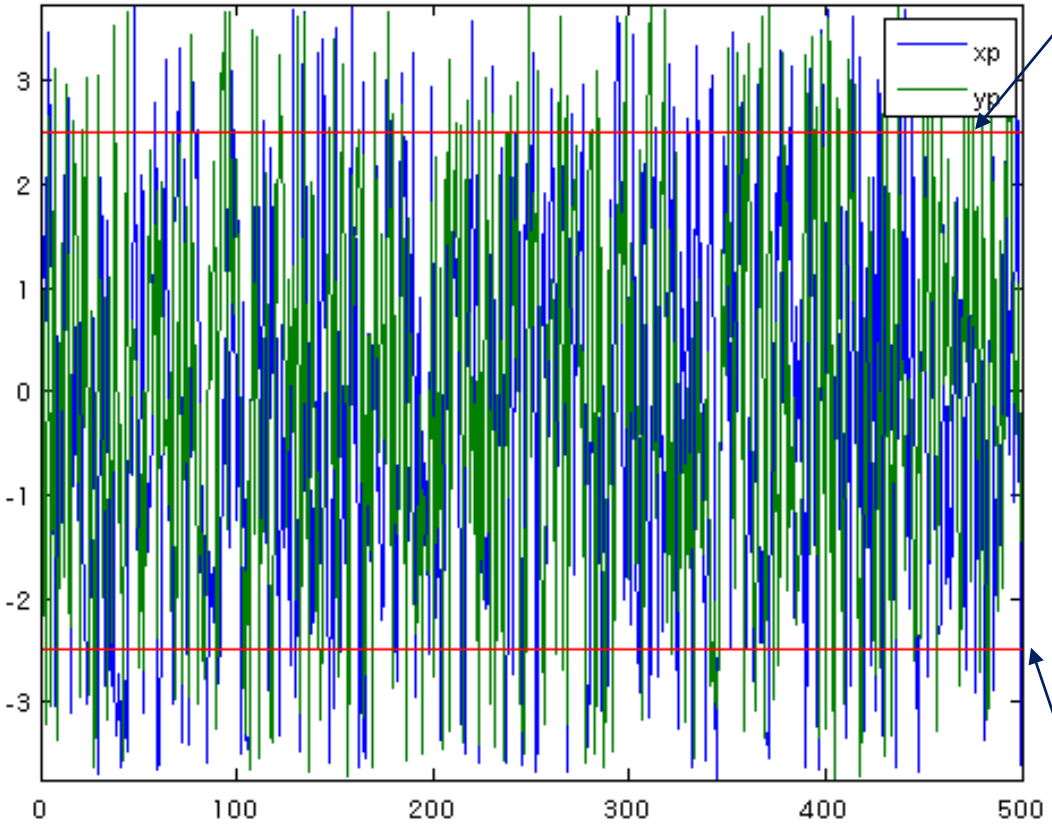
条件に適合するデータの配列番号（つまりID）を取り出せる

IDを入れる変数=find(条件)

条件に適合するIDで呼び出すことでデータの中身も取り出せる

Aselect=A(key)

全てのデータをプロット
（可視化上、線で表現しただけで
つなぐことには意味なし）

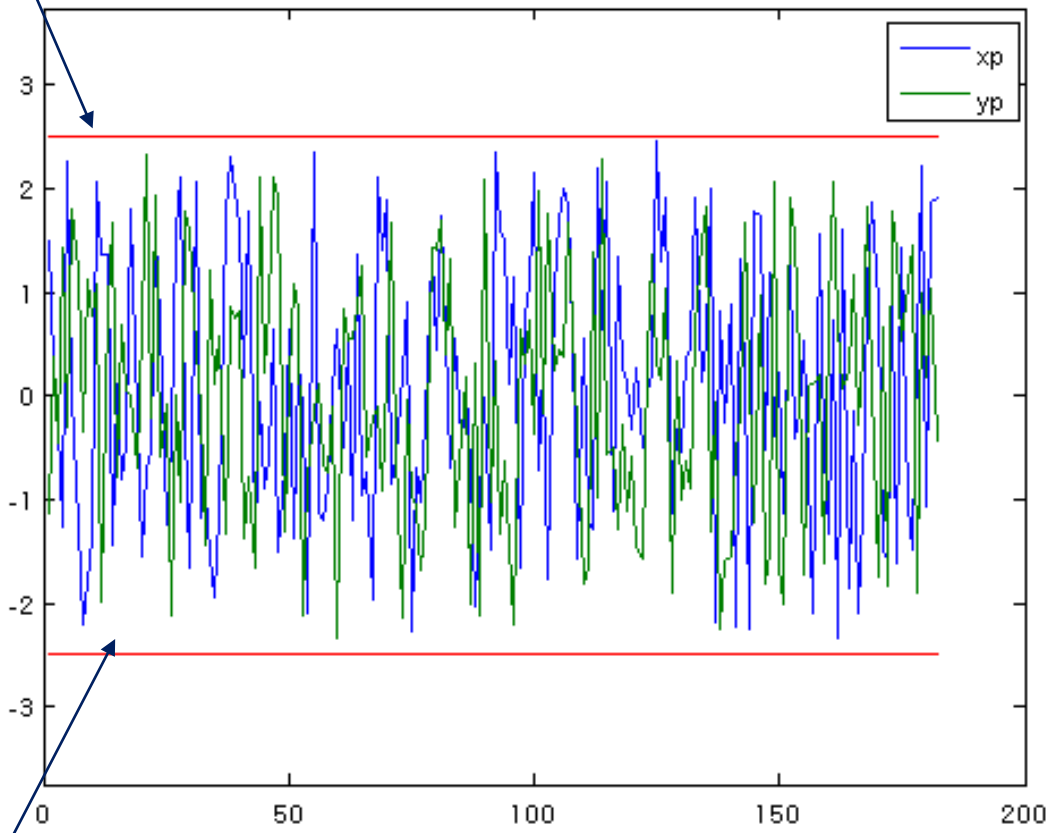


N=500;

cR=2.5

-cR=-2.5

条件に適合するデータのみ
（ 条件(xp.^2+yp.^2)<=cR.^2 ）



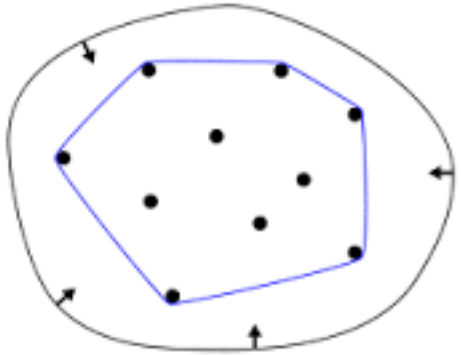
length(key) = 182;

さてもう少し難しい問題に進みます。

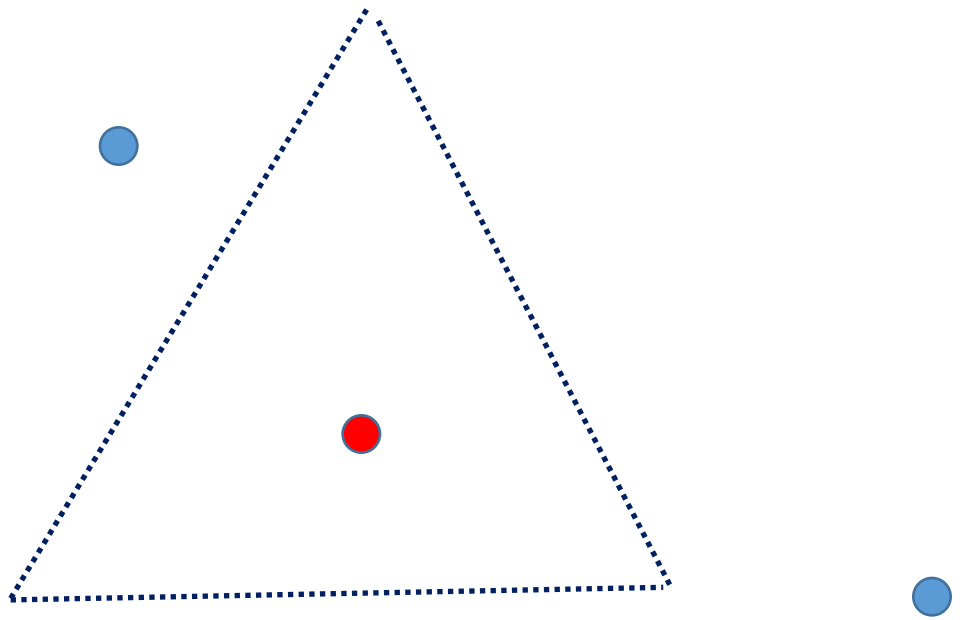
ポリゴン問題

数学における**凸包**（とつほう、英: convex hull）または**凸**包絡（とつほうらく、英: convex envelope）は、与えられた集合を含む最小の**凸**集合である。 例えば X がユークリッド平面内の有界な点集合のとき、その**凸包**は直観的には X をゴム膜で包んだときにゴム膜が作る図形として視認することができる。

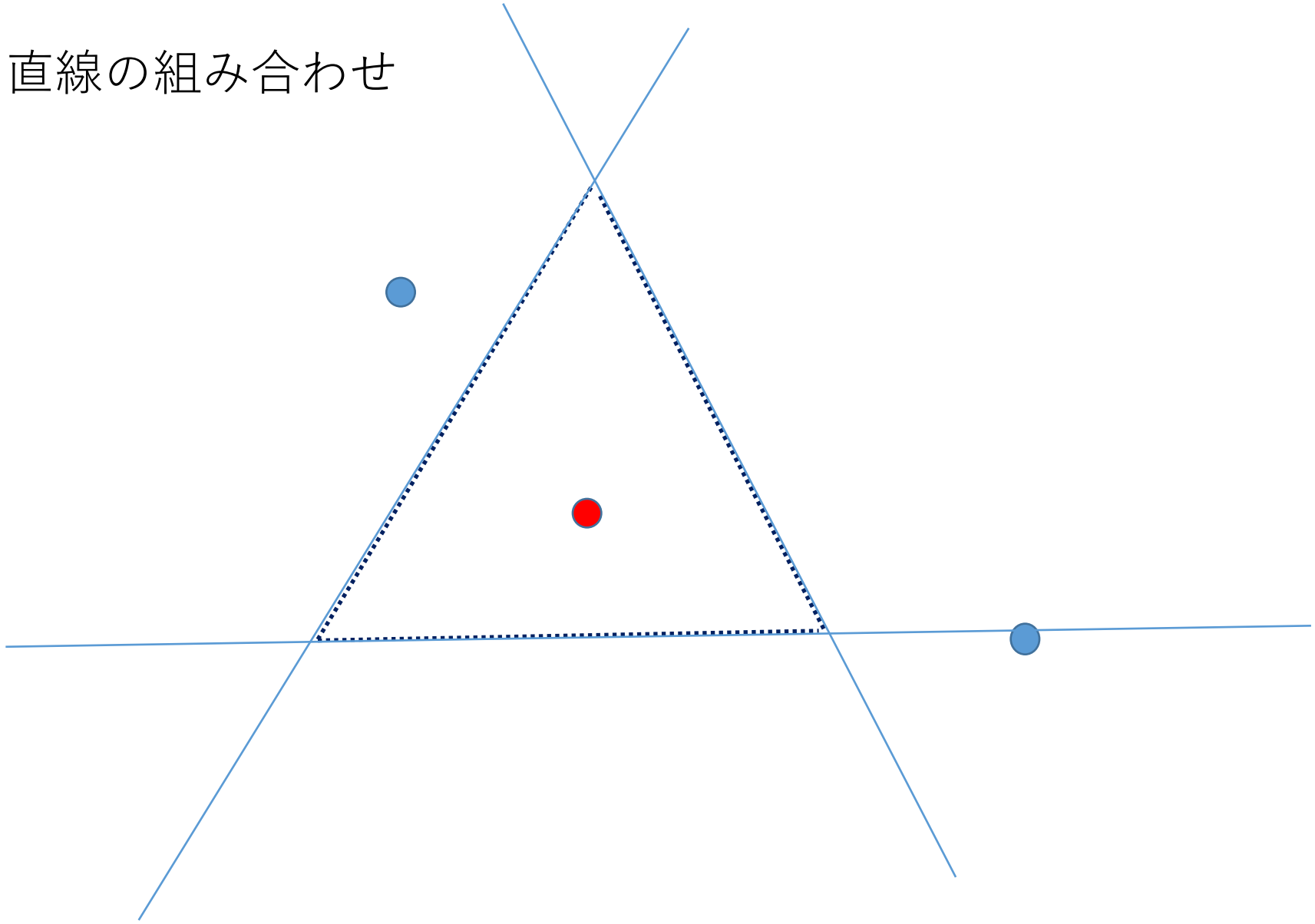
凸包 - Wikipedia
https://ja.wikipedia.org/wiki/凸包



$$\left\{ \sum_{i=1}^{|S|} \alpha_i x_i \middle| (\forall i : \alpha_i \geq 0) \wedge \sum_{i=1}^{|S|} \alpha_i = 1 \right\}$$

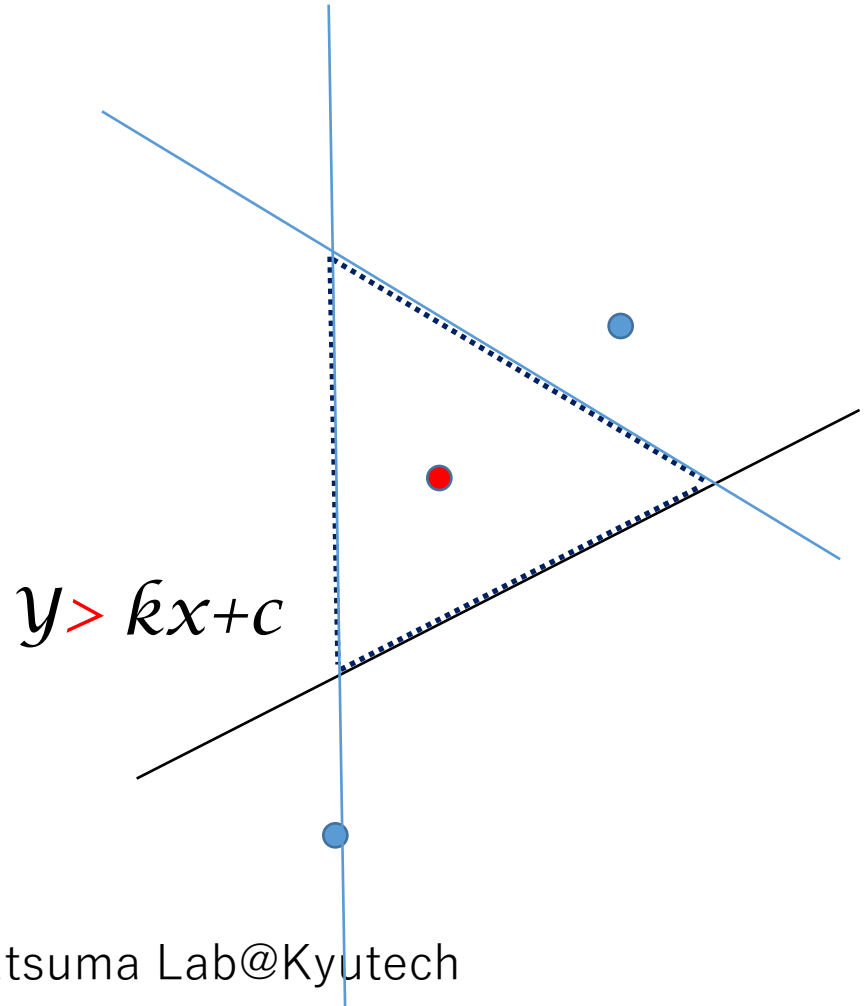


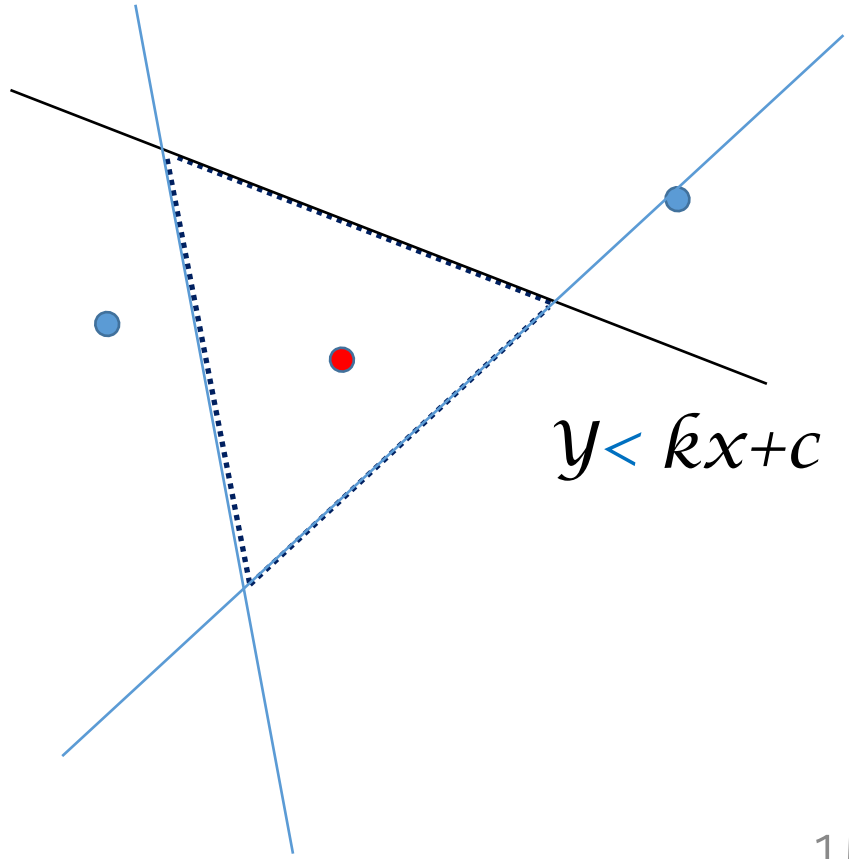
３つの直線の組み合わせ



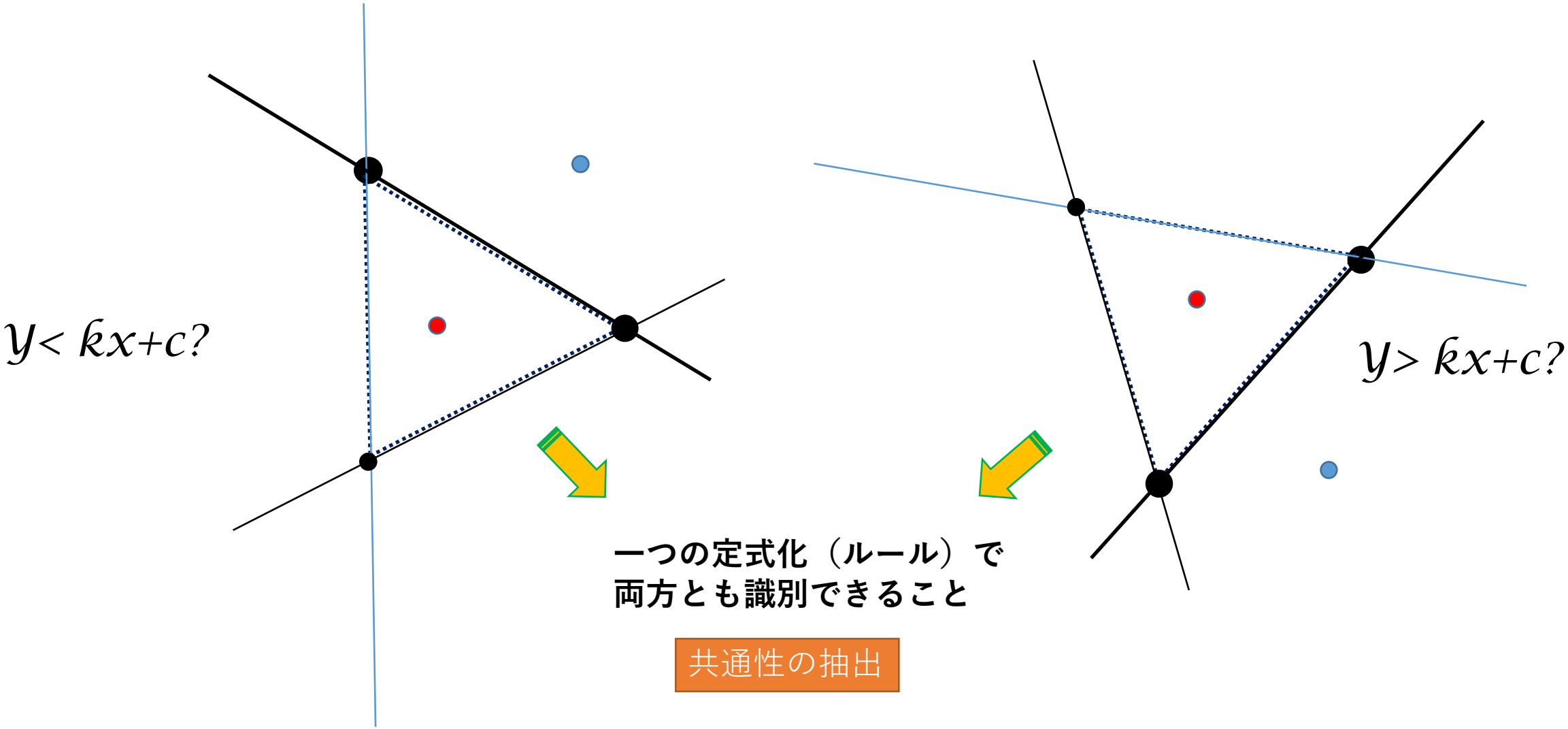
不等式の不等号の向きをどうするか？

⇒ 場合分けをする（しかし、計算が煩雑になる）
⇒ プログラミングを精緻化する必要が有る





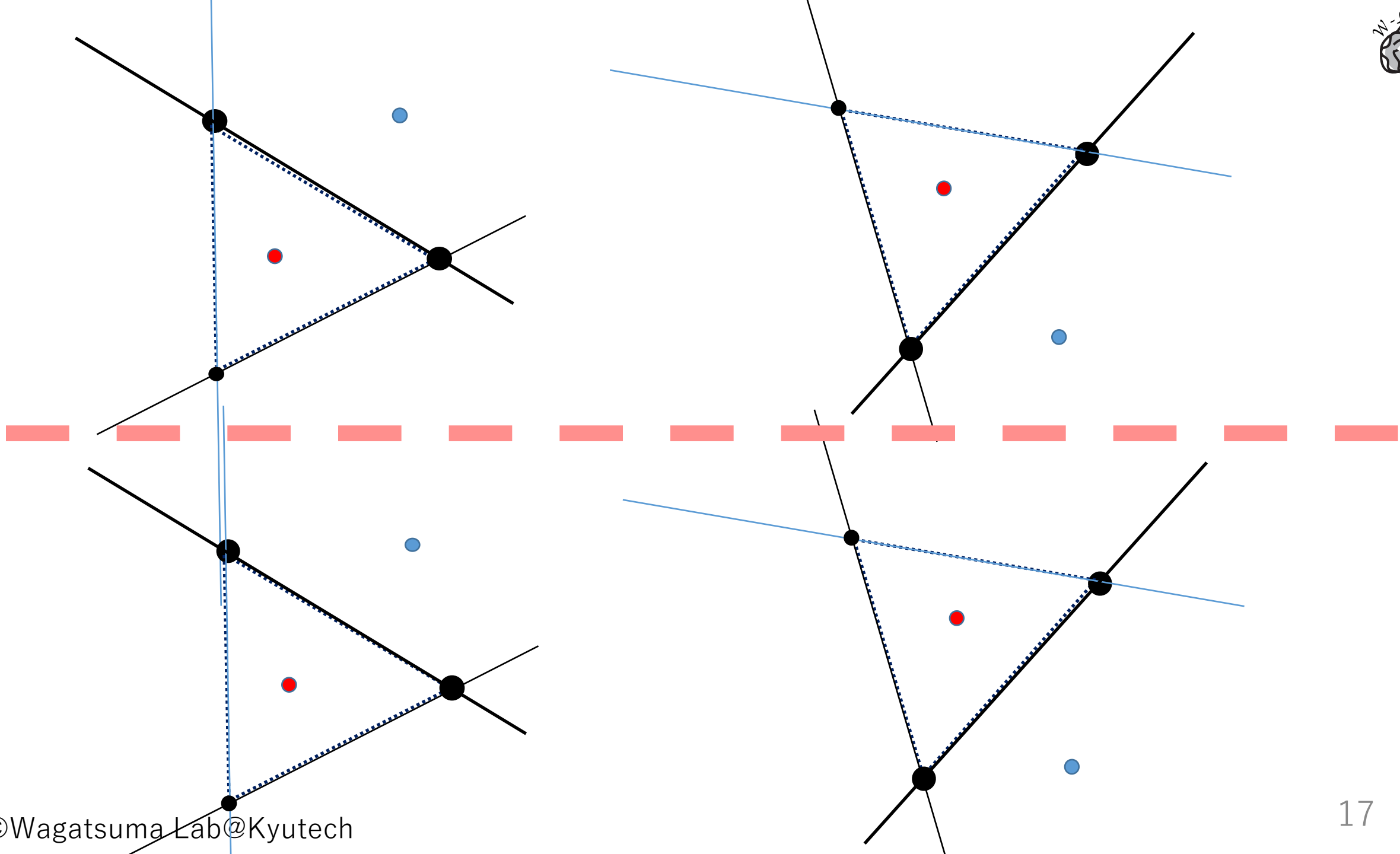
まず、一本だけ考えてみる
⇒ 後で、３本の条件の「&」を取れば良い



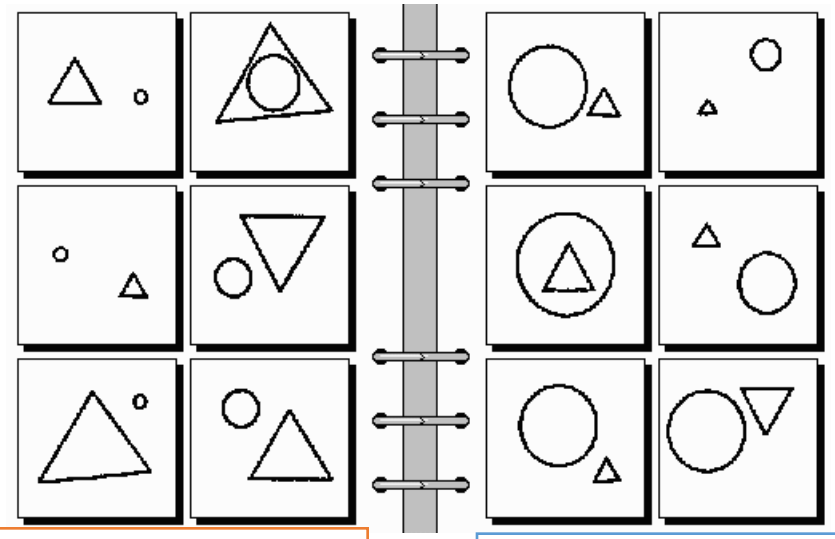
一つの定式化（ルール）で
両方とも識別できること

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

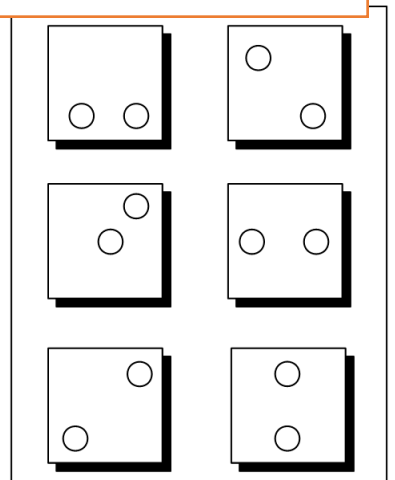


ボンガード（Bongard Problem）知能パズルの一種で、人間でも解くのが難しいといわれている

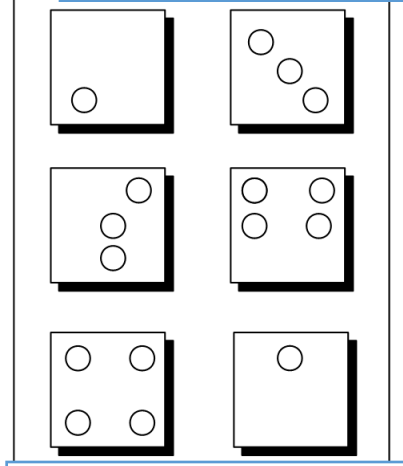


Circle **is smaller than** triangle

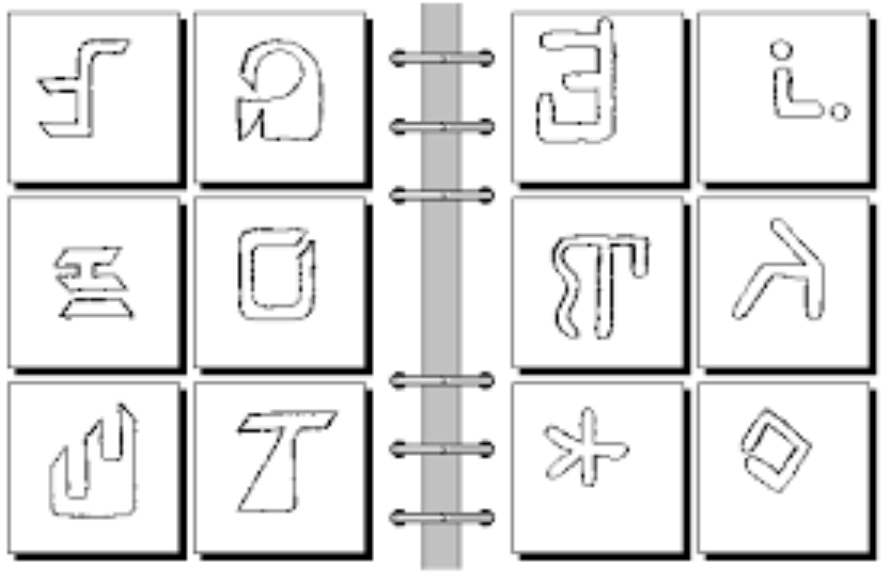
Triangle **is smaller than** circle

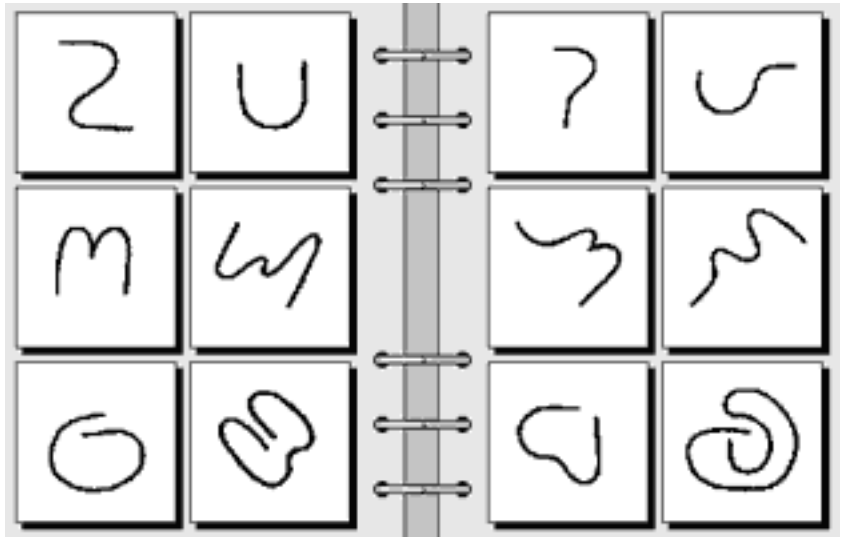


 # items is two



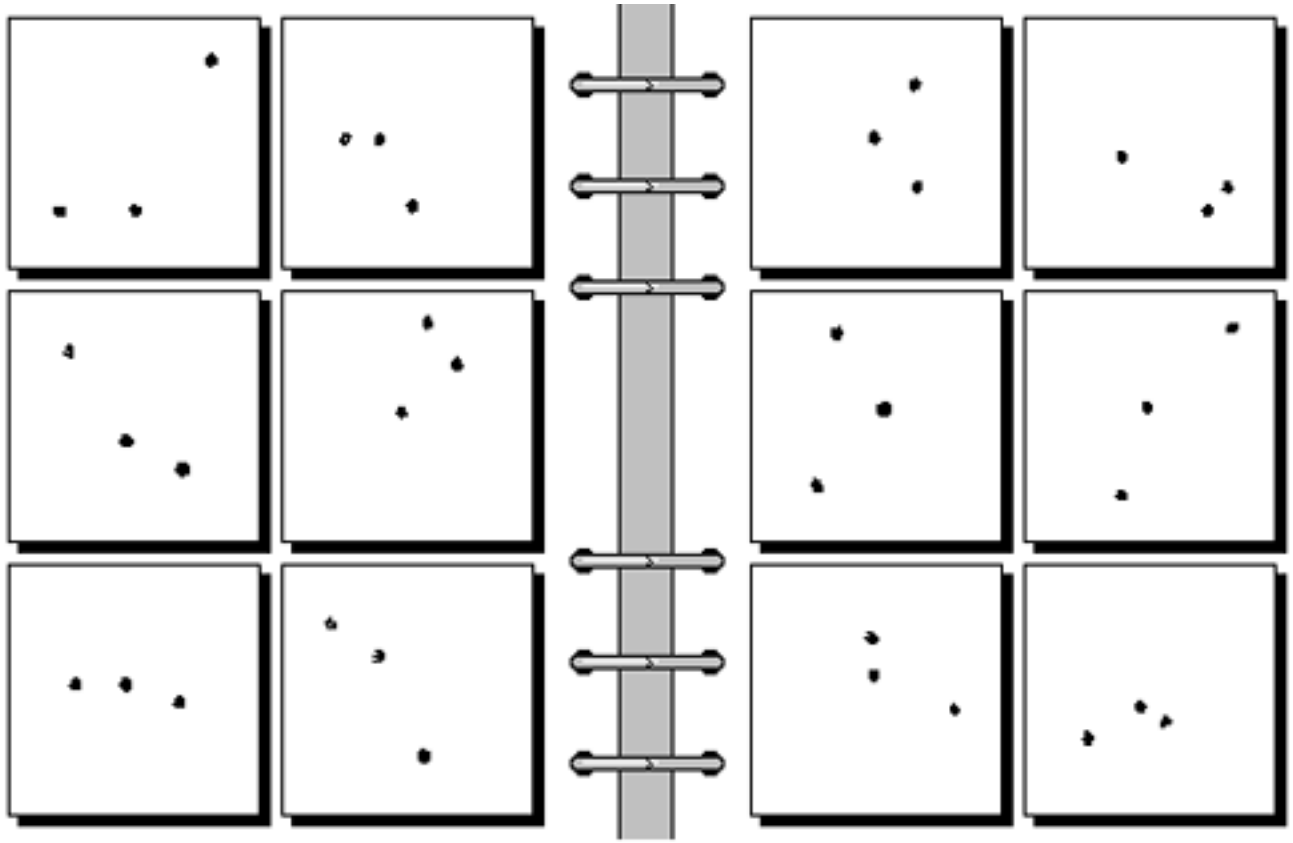
 # items is not two





A Master Thesis

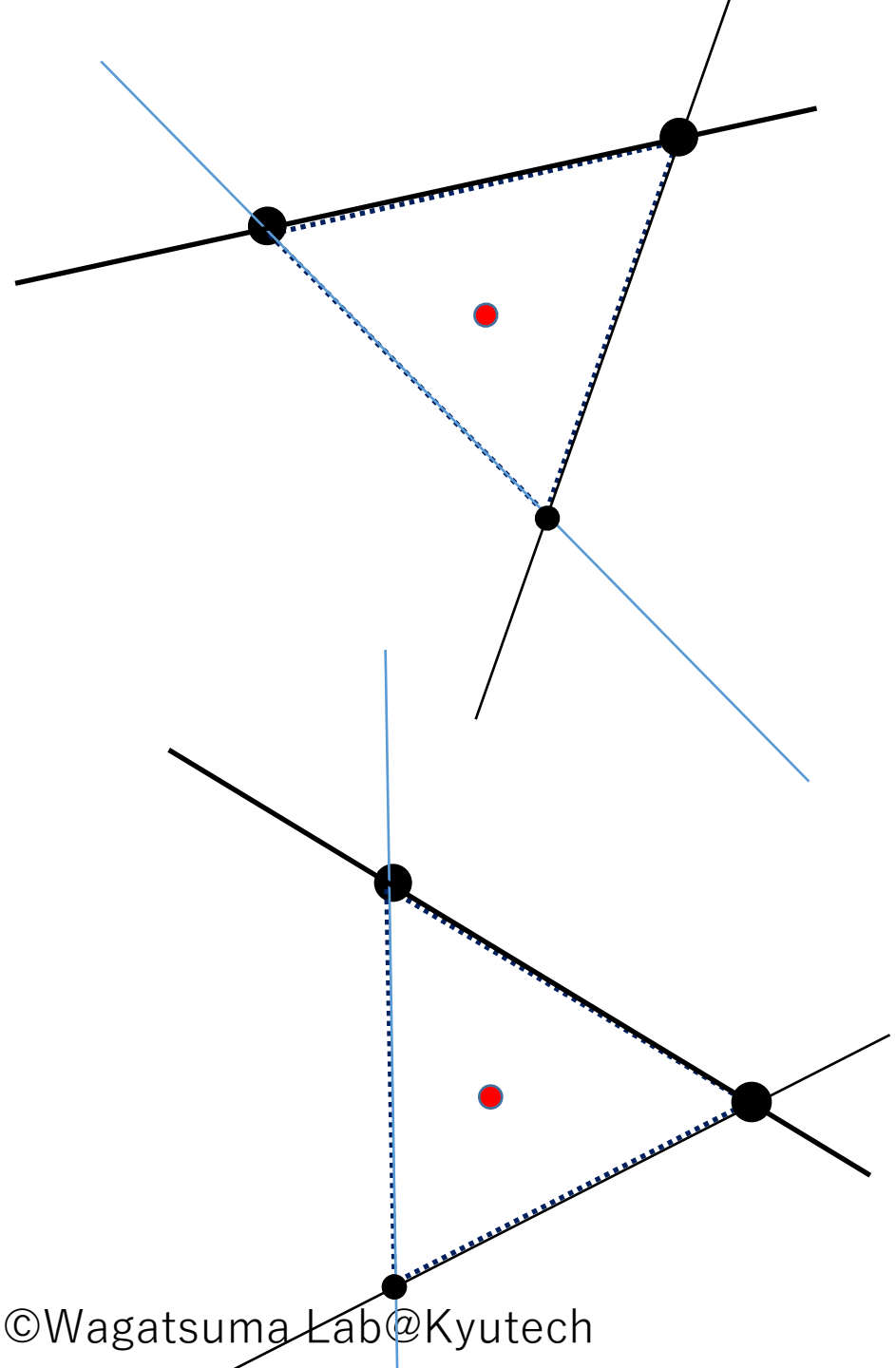
ボンガード（Bongard Problem）最難問といわれるもの

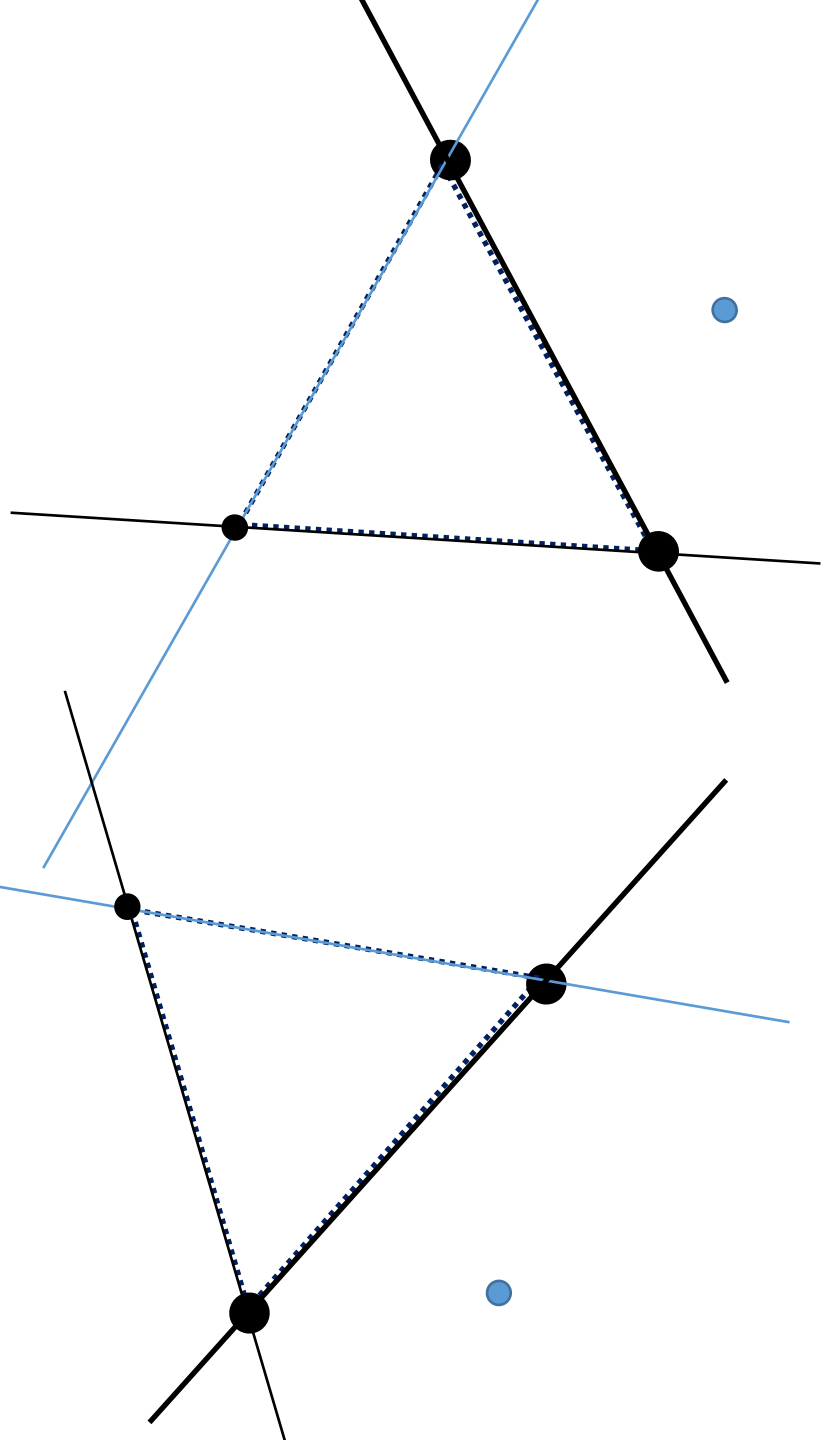


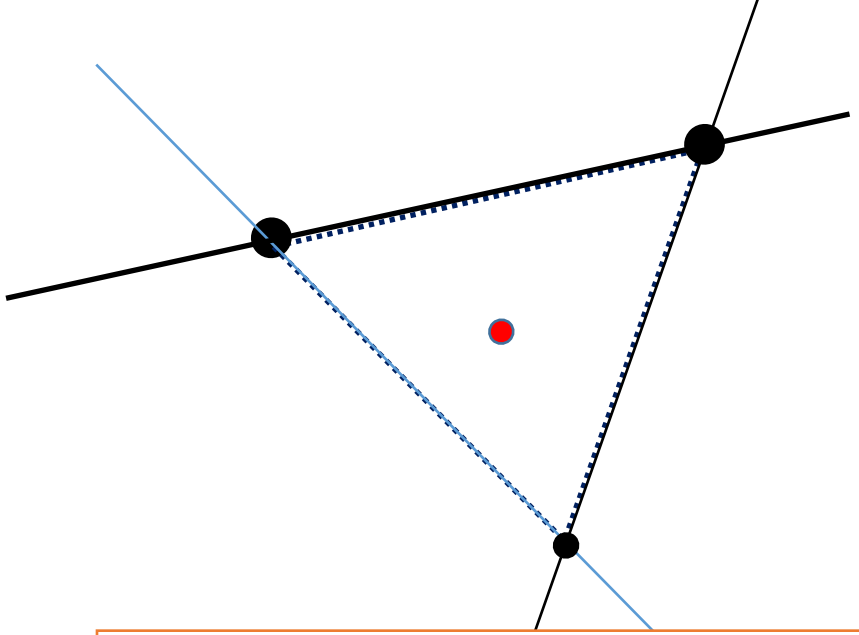
左6つの模様に共通するルール ⟺ 右6つの模様に共通するルール

ただし、二つは相反する

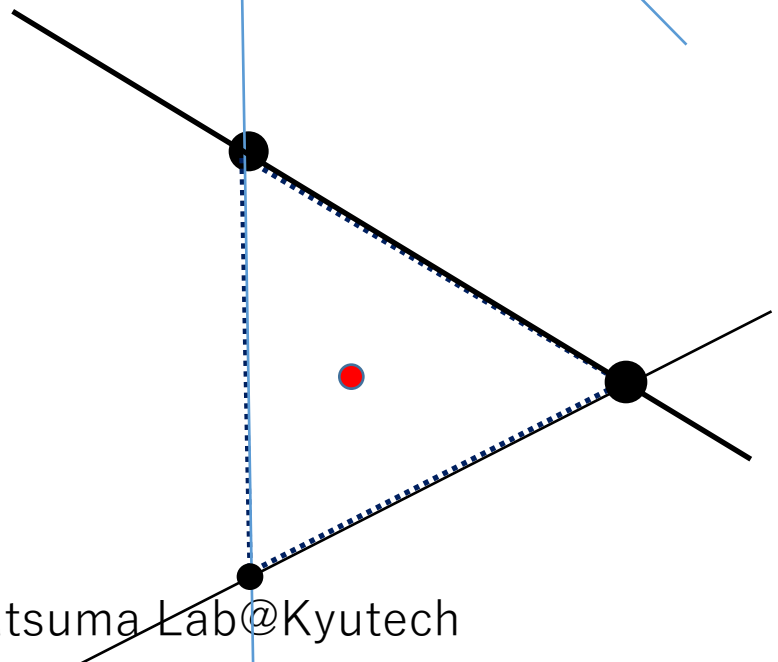
http://www.gizmodo.jp/2013/08/_so_you_th.html

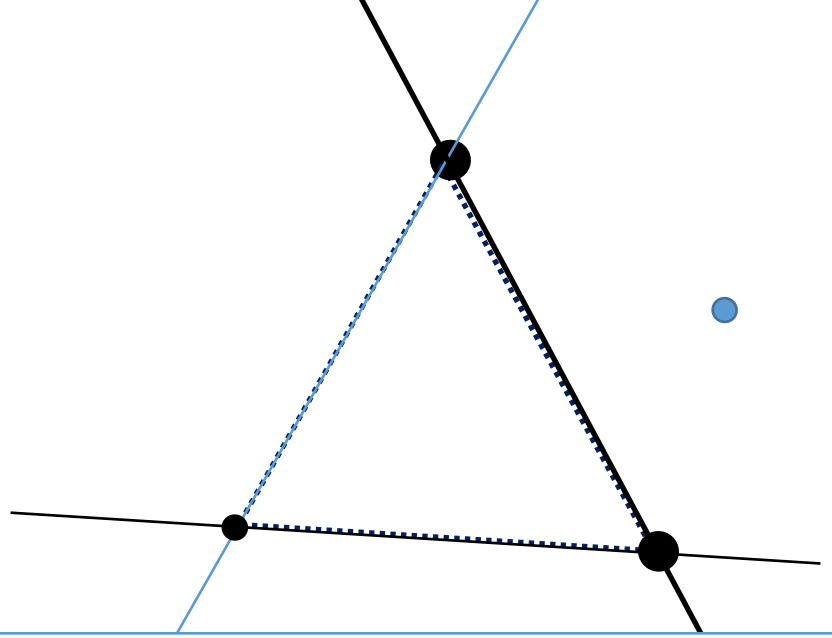




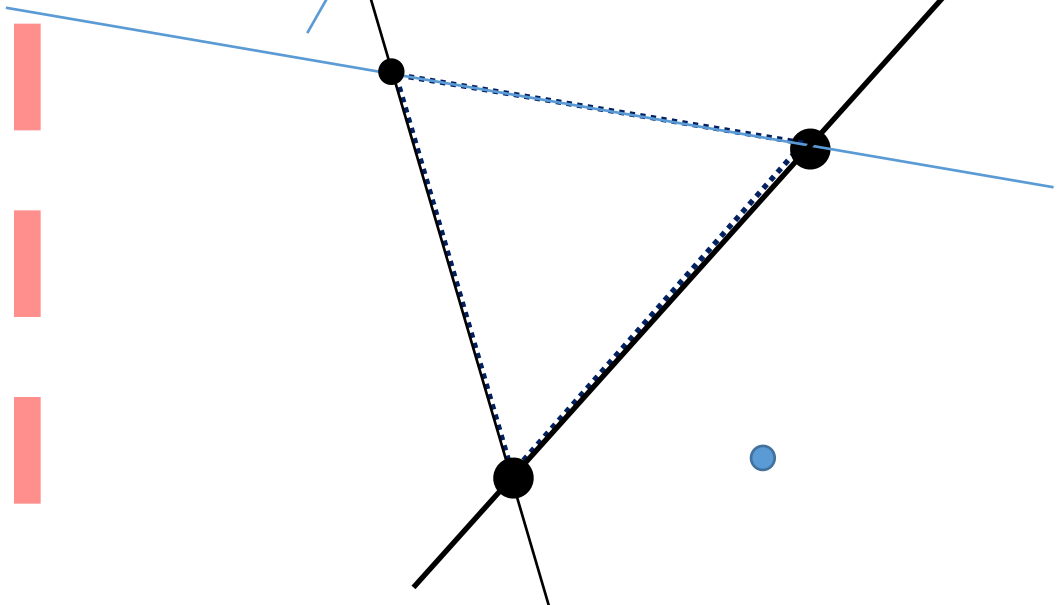


直線を構成する2点に対し、残った点が、**直線に対して同じ側にある**こと。

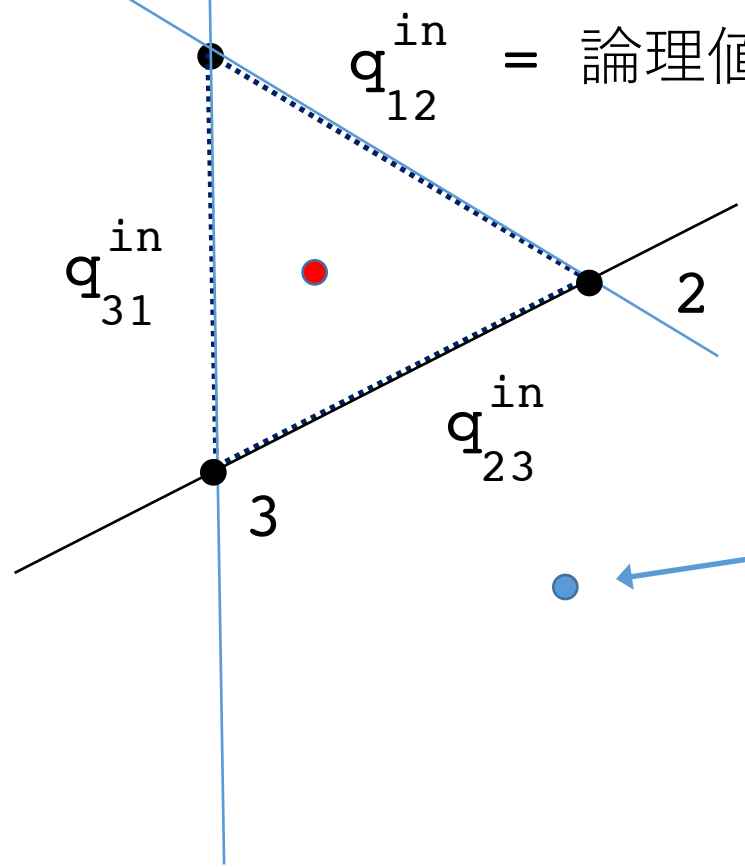




直線を構成する2点に対し、残った点が、**直線に対して逆側**にあること。



$y > kx+c$?



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

$1 \wedge 1 \wedge 1 = 1$

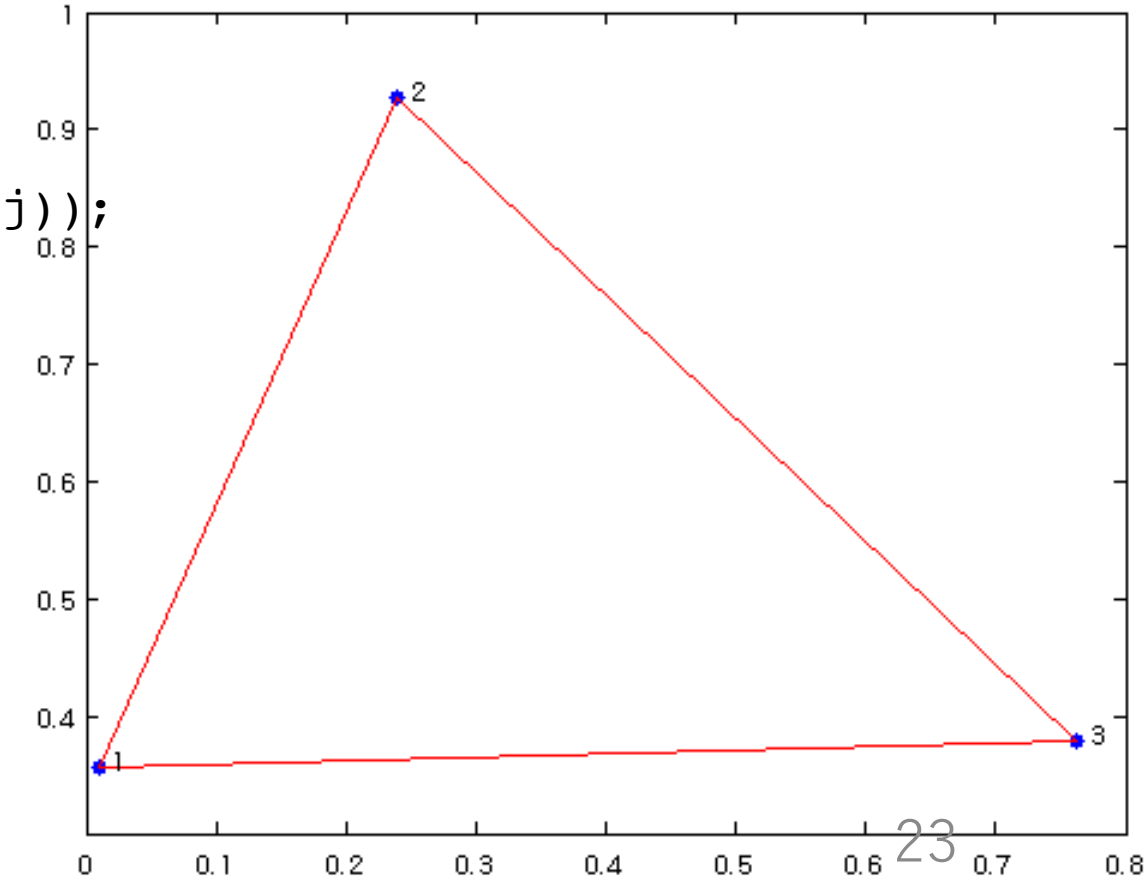
$1 \wedge 0 \wedge 1 = 0$

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)

```
% frame data generation in the first place
FPnum=3;

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

figure(4); clf
plot(FPxy(1,:),FPxy(2,:),'.','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
```



[Polygon_Detect_Normal.m（三角形・内部外部識別・基本）]

```
DPnum=13;
DPxy=rand(2,DPnum);

% y-y1=(y2-y1)/(x2-x1)*(x-x1) % This is formula

% 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));


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

```
inq(DPxy(:,j),FPxy2(:,1),FPxy2(:,2))
```

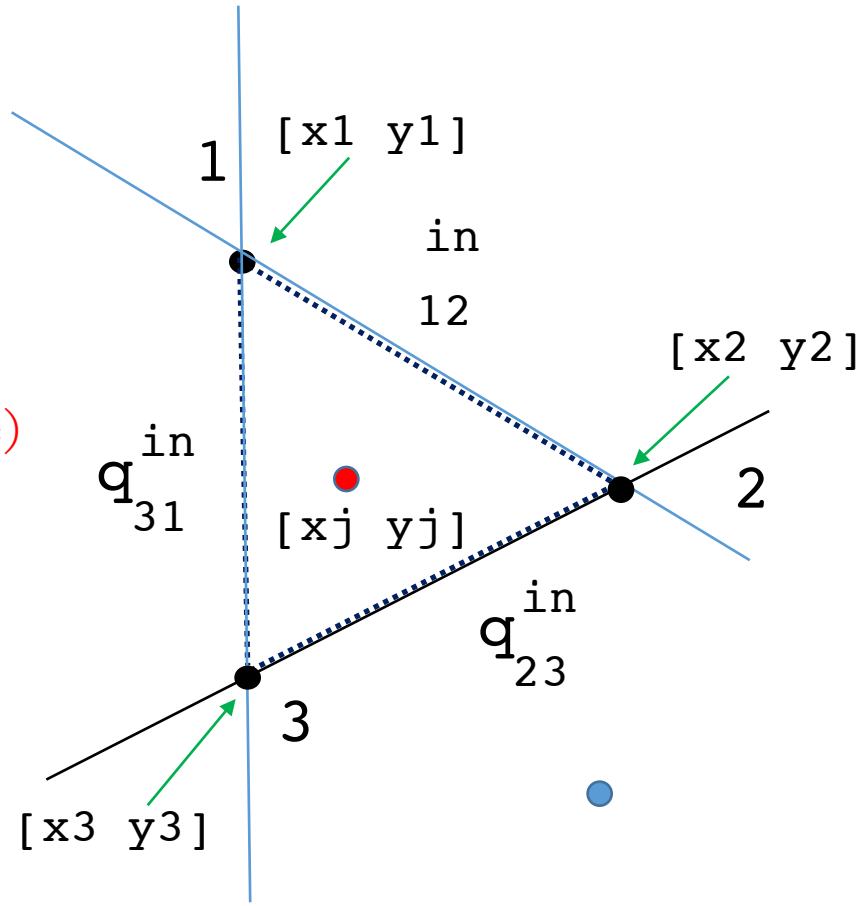
```
FPxy2(:,1)=[x1 y1]
FPxy2(:,2)=[x2 y2]

DPxy(:,j)=[xj yj]
```

[x1 y1]と[x2 y2]を結ぶ直線に対して、[xj yj]が上（y方向で正）にあれば1、そうでなければ0

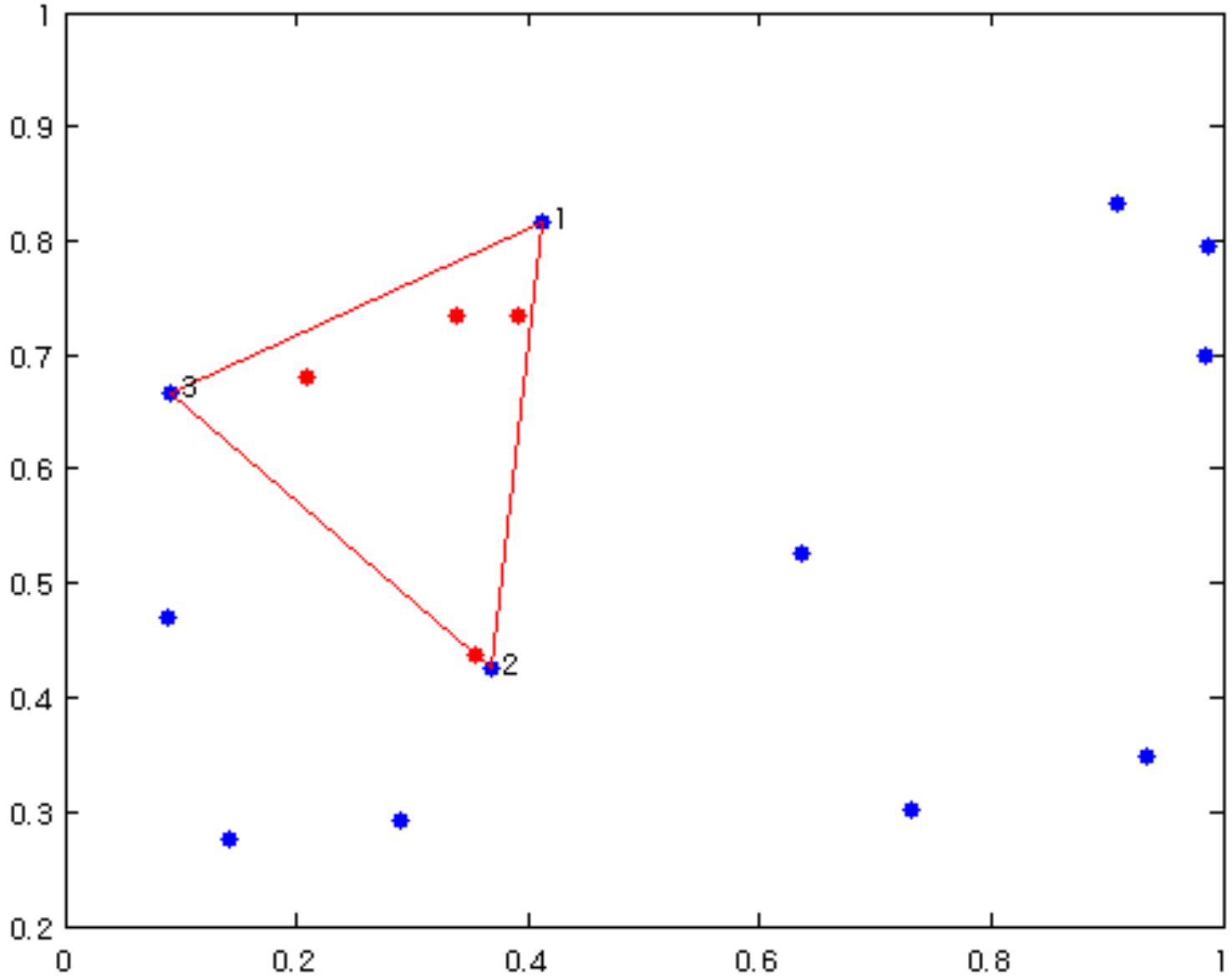
両者が同じ側にあるか(一致==)　⇕　逆側か(不一致~=)

```
inq(FPxy2(:,3),FPxy2(:,1),FPxy2(:,2))
```

[x1 y1]と[x2 y2]を結ぶ直線に対して、[x3 y3]が上（y方向で正）にあれば1、そうでなければ0




```
% 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));
```



提案したアルゴリズムが目的とした機能を果たすことがわかったら
⇒ 見通しのよい構造に最適化する（それによってさらに一般化が図れる可能性がある）

```
% 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）

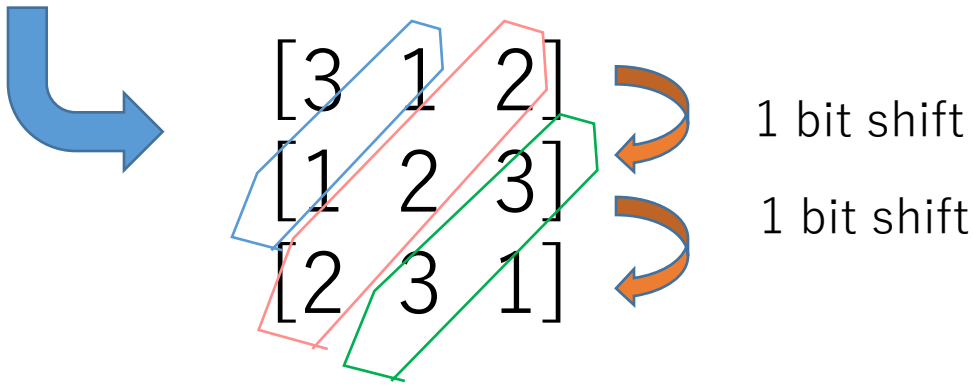
一般化とは何か？

個別の違いを超えて共通のルールを抽出すること

⇒ **形式的な記述に置き換え、その「構造」に注目**する

```
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))
```

同じ



さらなる一般化

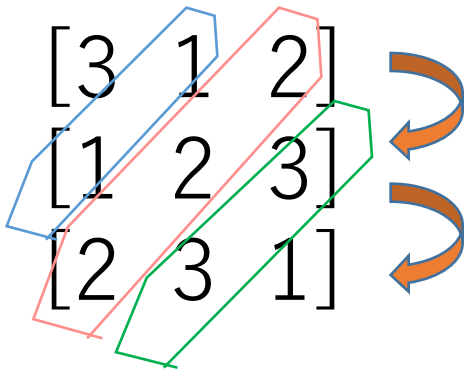
⇒ 行列のシフト演算（ビットシフト）：並列化やハード化を意識した設計

```
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);
Cinq2=@(cPa,cB) inq(cPa,cB(:,2),cB(:,3))==inq(cB(:,1),cB(:,2),cB(:,3));
Cinq3=@(cPa,cB3) Cinq2(cPa,cB3(1:2,:)) && Cinq2(cPa,cB3(3:4,:)) && Cinq2(cPa,cB3(5:6,:));

for j=1:DPnum
    FPelement=[FPxy; circshift(FPxy,[0 1]); circshift(FPxy,[0 2])];
    if Cinq3(DPxy(:,j),FPelement)
      pcol=[1 0 0];
    else
      pcol=[0 0 1];
    end
    plot(DPxy(1,j),DPxy(2,j),'.','MarkerSize',22,'color',pcol), hold on;
end
```

Polygon_Detect_Advanced.m（三角形・内部外部識別・シフト演算）

```
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))
```

この場合は、{1,2,3}という集合のすべての順列が揃っていれば良い



1 bit shift

1 bit shift

[1 2 3]

1 bit shift rightward

[3 1 2]

2 bit shift rightward

[2 3 1]

1 bit shift leftward

```
[circshift(FPxy,[0 -1]);
FPxy;
circshift(FPxy,[0 1])];
```

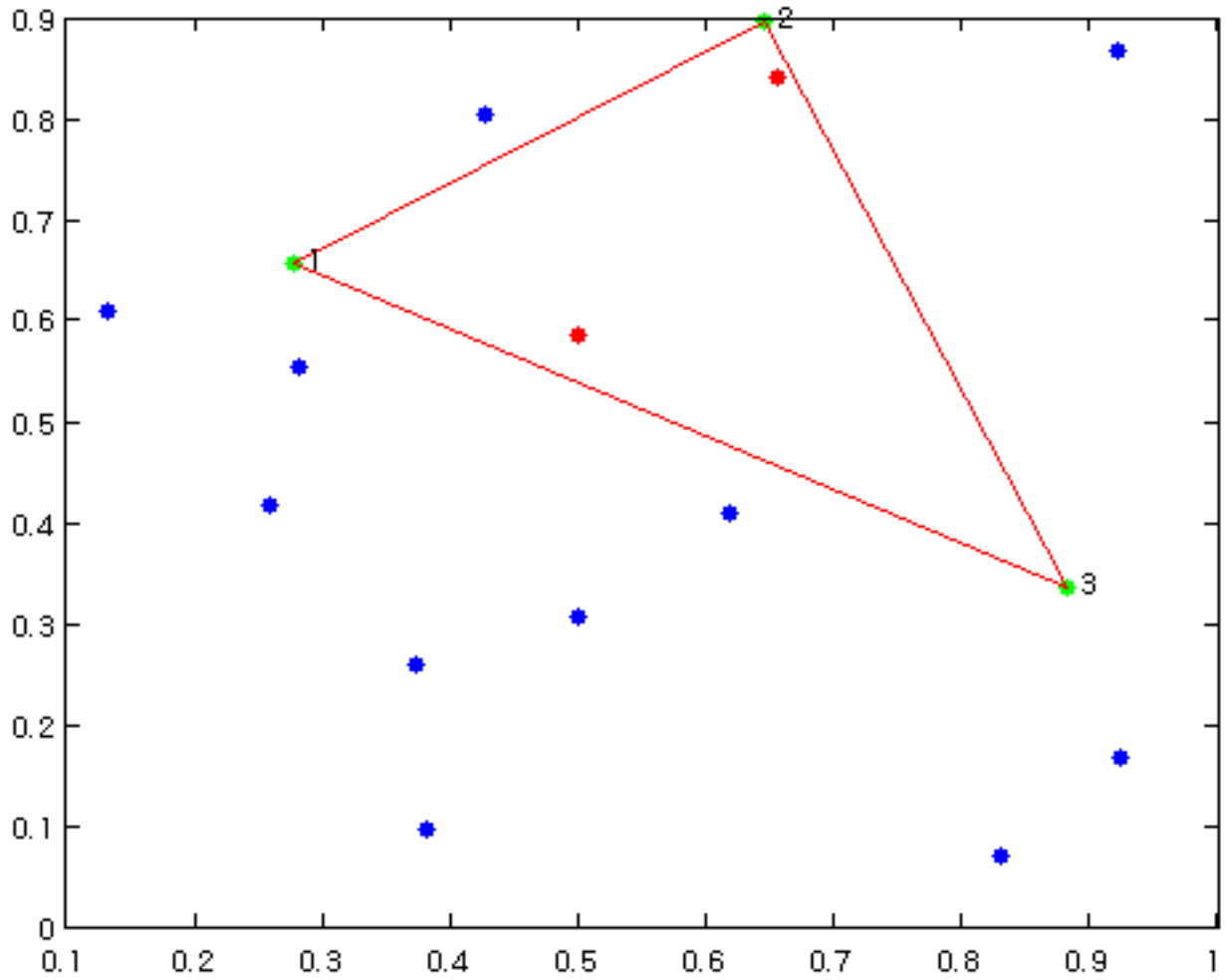
1 bit shift rightward

```
[FPxy;
circshift(FPxy,[0 1]);
circshift(FPxy,[0 2])];
```

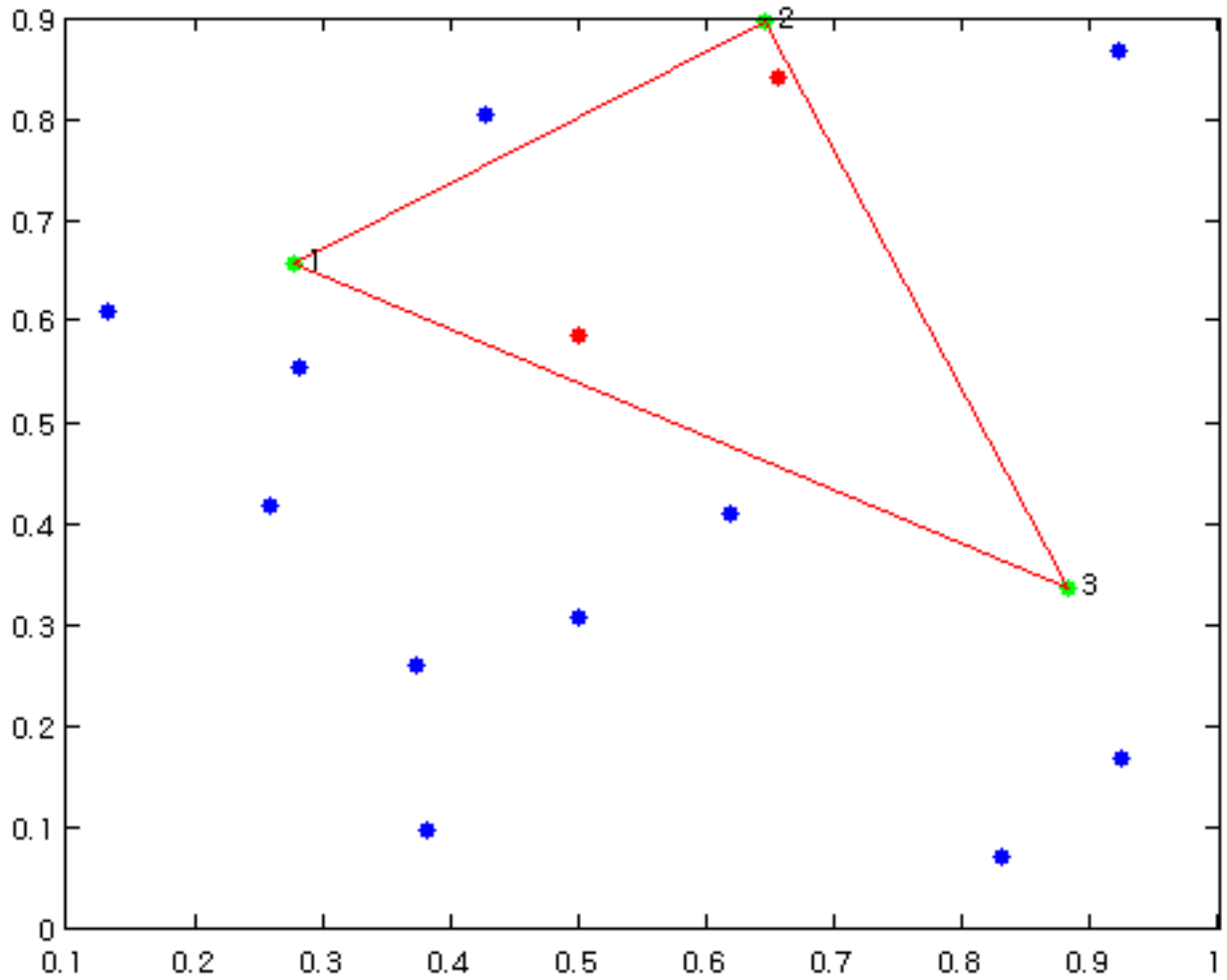
関数も、ビットシフトを操作しやすい引数（関数に与える変数）構造にした方が、のちのち便利（一般化しやすい）。

```
Cinq2=@(cPa,cB) inq(cPa,cB(:,2),cB(:,3))==inq(cB(:,1),cB(:,2),cB(:,3));
Cinq3=@(cPa,cB3) Cinq2(cPa,cB3(1:2,:)) && Cinq2(cPa,cB3(3:4,:)) &&
Cinq2(cPa,cB3(5:6,:));
```

というわけで、結果が得られる。アルゴリズムは変わっていないが、関数の演算の方式を見直したことで、より見通しのよいプログラムになっている。



では、皆さん自分で実行して、
動作確認をしてみましょう。

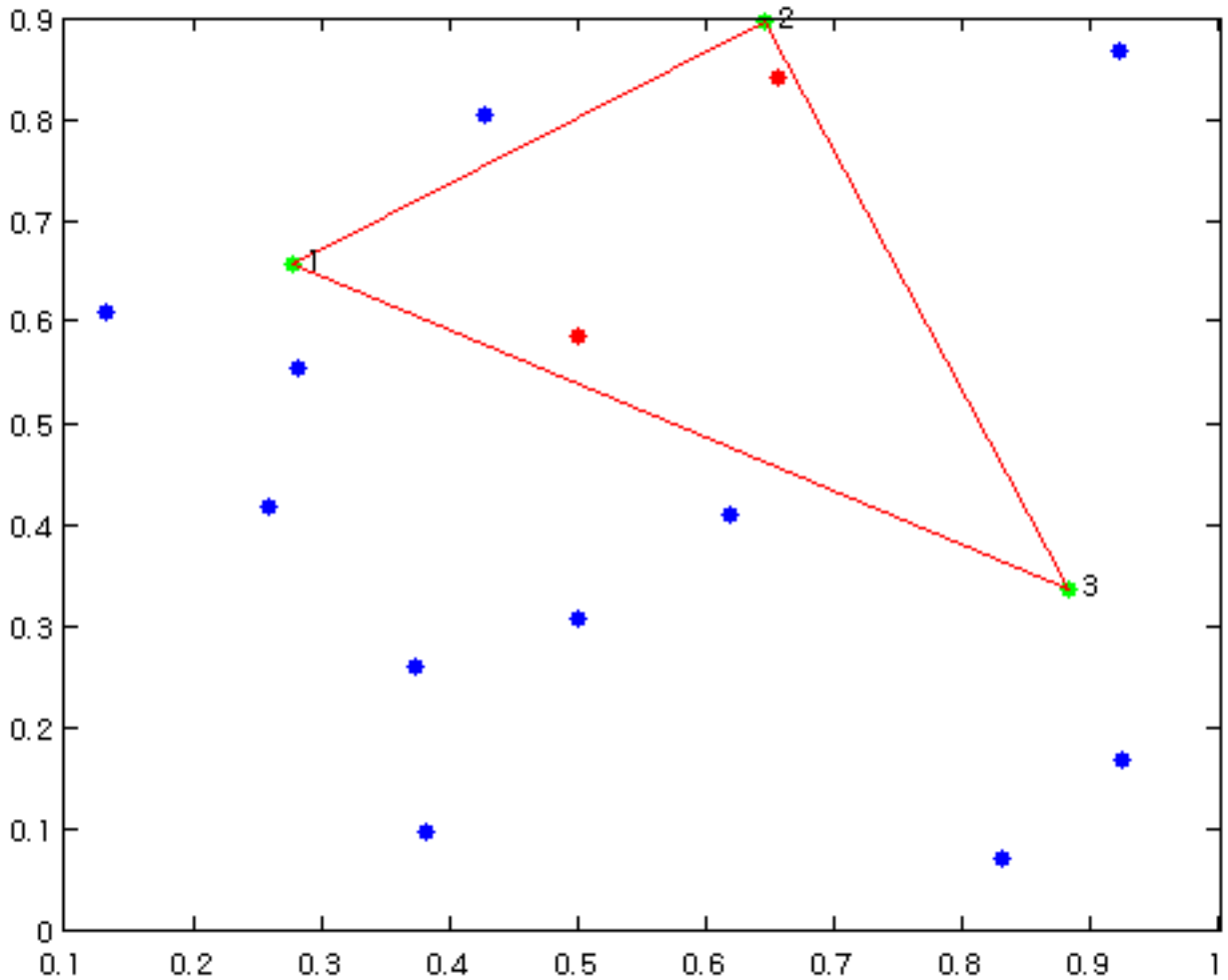


今日の課題はここまで。

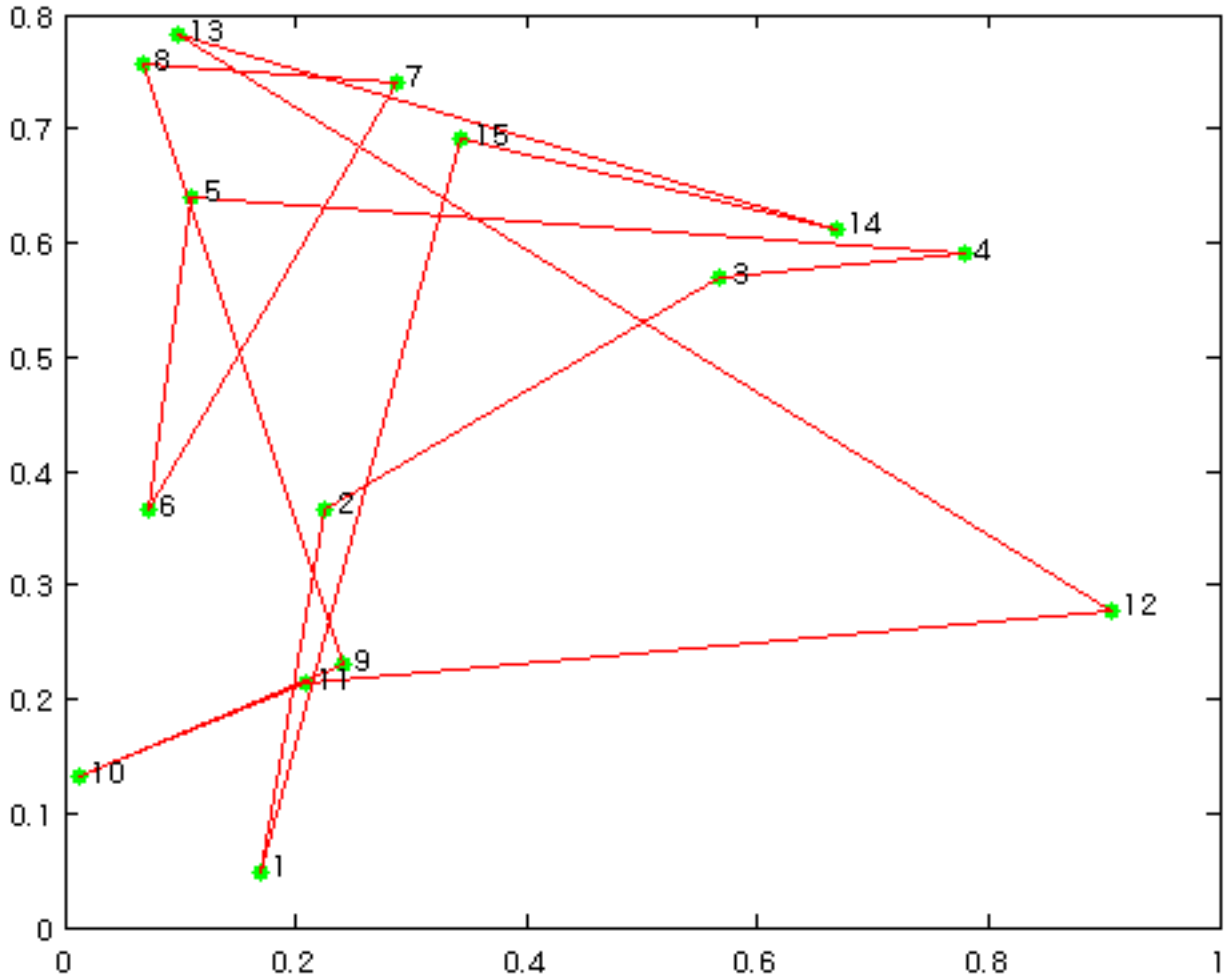
ですが、次回の予告と、

宿題をちょっと…

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。

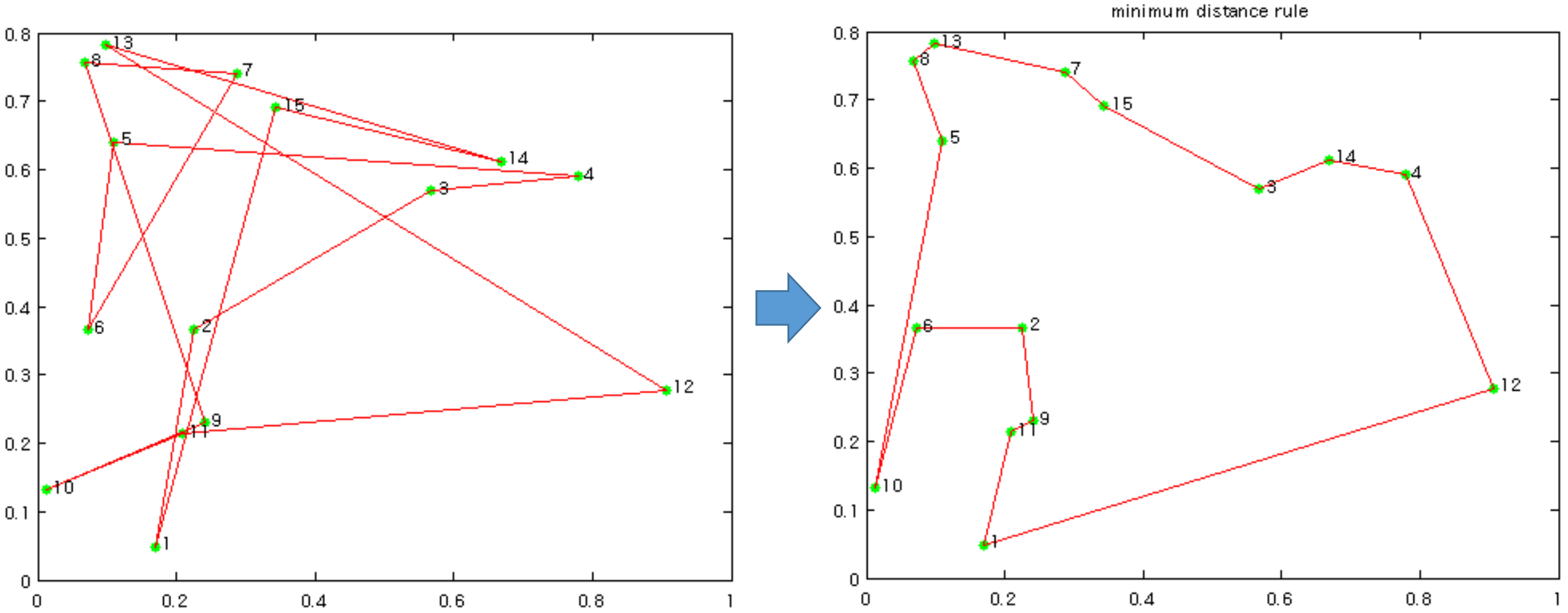


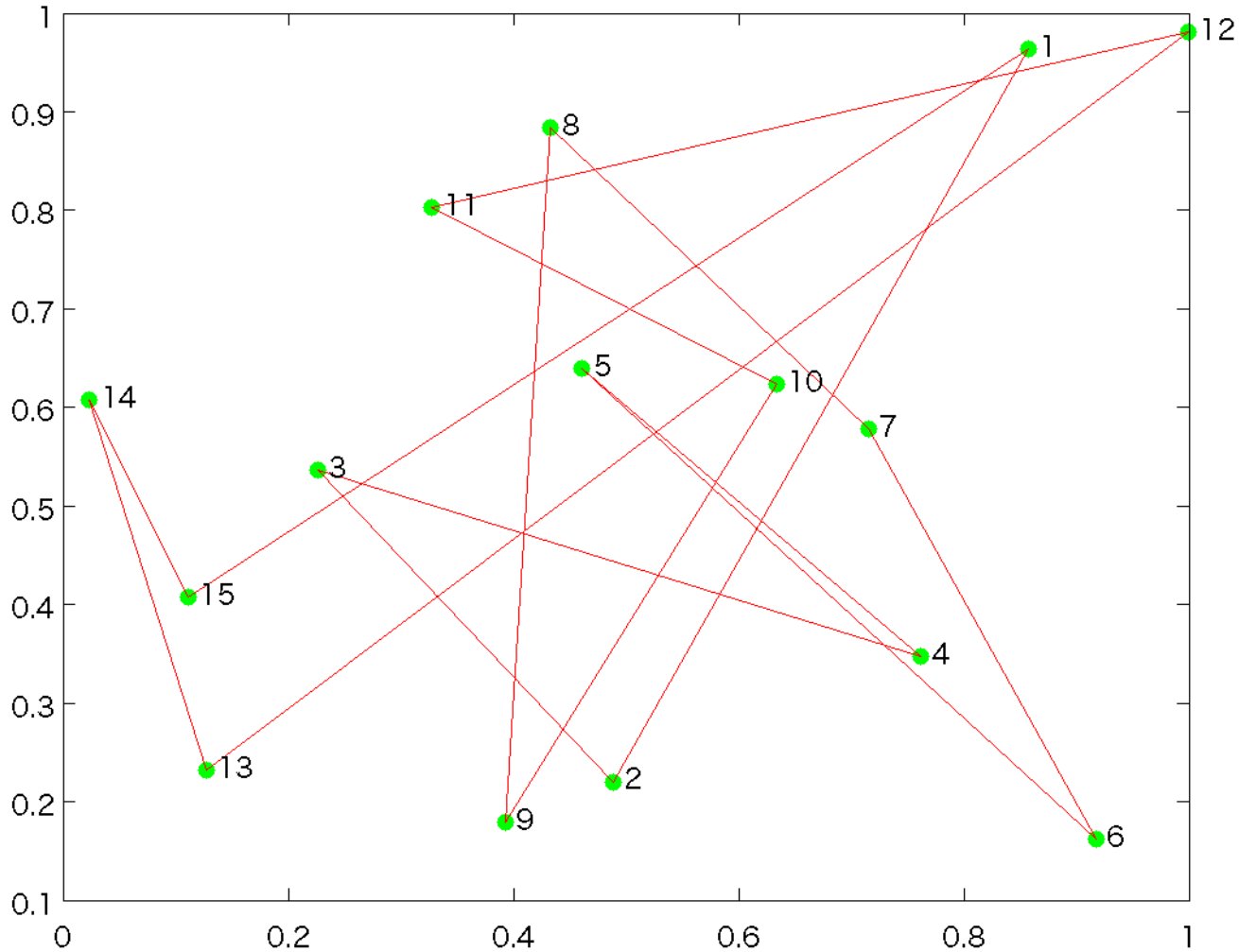
さて、点が３つ以上のときには
i) 一意性
ii) 最短経路
はどうなるでしょう？



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





The End – 1st lecture