プログラミングによる数学演習

勝山高非常勤(「情報1」担当) 原口

1： 方眼紙と作図

連立方程式(2D 版)とベクトル

基本図形と三角関数

代入、関数などイロハのイ

2. 数列と極限

反復:イロハのロ

3. 連立方程式(3D版)

解がない場合の近似解

4. データ分析

データ分析は「数学1」で扱います．多次元のデータを低次元で可視化するのが普通です

1： 方眼紙と作図

Python プログラミングは難しいとの感想をよく耳にする

考えられる理由： 変数、代入文、等式の理解不足、もしくは混同

そもそも変数とは？ 数学との関係性は？

中学数学でも「変数」は使っている： 連立方程式、一次関数

Contents:

連立方程式の図形的（幾何学的）な意味

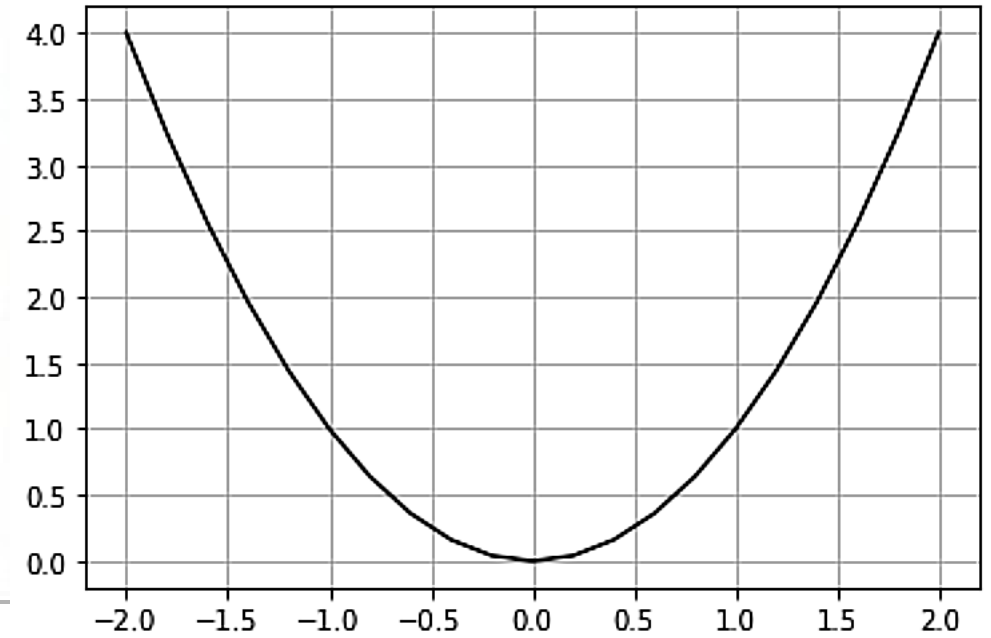
Point: 作図を主とした説明・解き方

数値計算・線形代数ライブラリを使うと簡単にプログラミングできるが

数学的理解なしの適用は盲目的

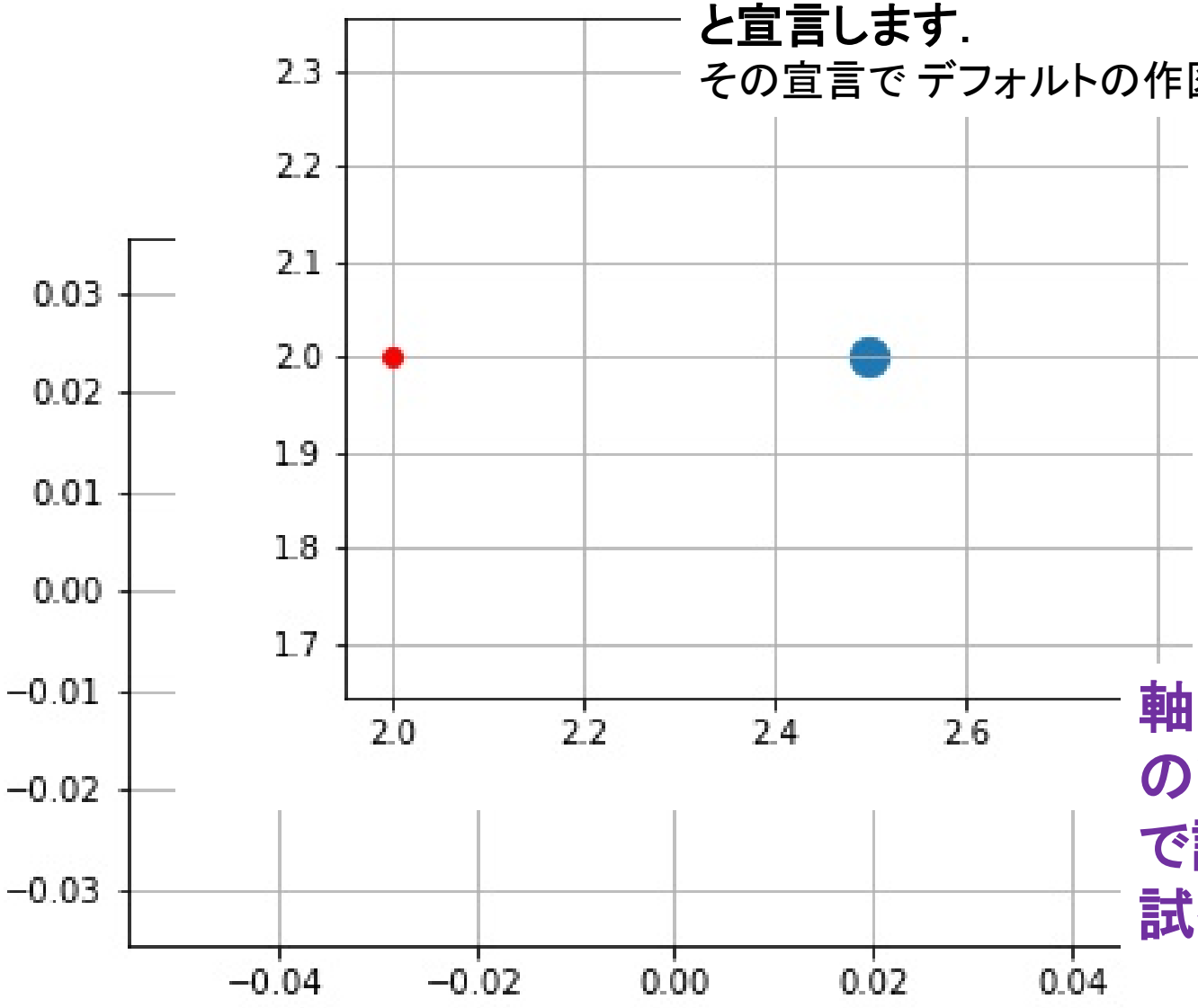
できるだけ図を．そのために方眼紙の話から

中学校のとき、『方眼紙を使いできるだけきれいな図を描きなさい』と
数学の先生から言われたことを思い出す．
綺麗に書くのは時間がかかるし、大変だし、
その後はサボり続けているが、綺麗な図示（可視化）は
今の世の中ではツールを使って当たり前？



方眼紙

最初は import 文で pyplot なる
『グラフ作成用の道具箱を使います』
と宣言します.
その宣言で デフォルトの作図場所が用意され，その道具が使えるようになります




```
import matplotlib.pyplot as plt
plplt.grid();plt.axis('equal');
```

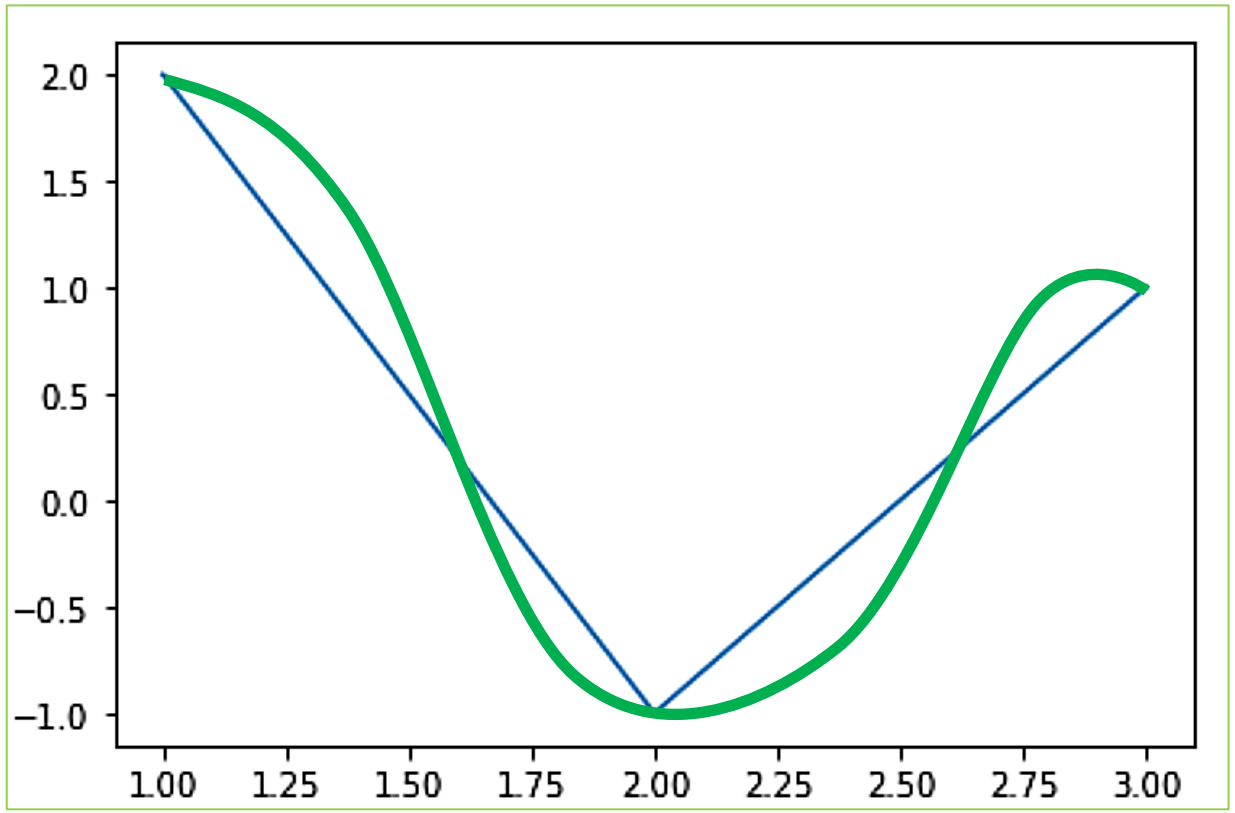
```
plt.plot(2,2,marker='o',color='red')
plt.scatter(3,2,marker='*',c='blue',s=600)
plt.scatter(2.5,2,s=150)
 #scatter は散布図作成によく用いられる
```

軸の自動調整は pyplot の良い点でもあるので，pyplot に最初はまかせて描画し，後で調整するのが一般に良い方法だろう．．．試行錯誤であまり気にしないこと

方眼紙 pyplot.plot(X座標,Y座標) から

線分の組 としての plot([1,2,3],[2,-1,1])

x の値が １，２，３ と変化したとき、
y の値が ２，－１、１ と変化した



いろいろな変化の仕方があるが
最も単純なものは線分の組み合わせ
（直線的変化）

構造を持つデータとしての点

変数 x, y, z, … ある値やデータを保管・保持する「もの」 変数名

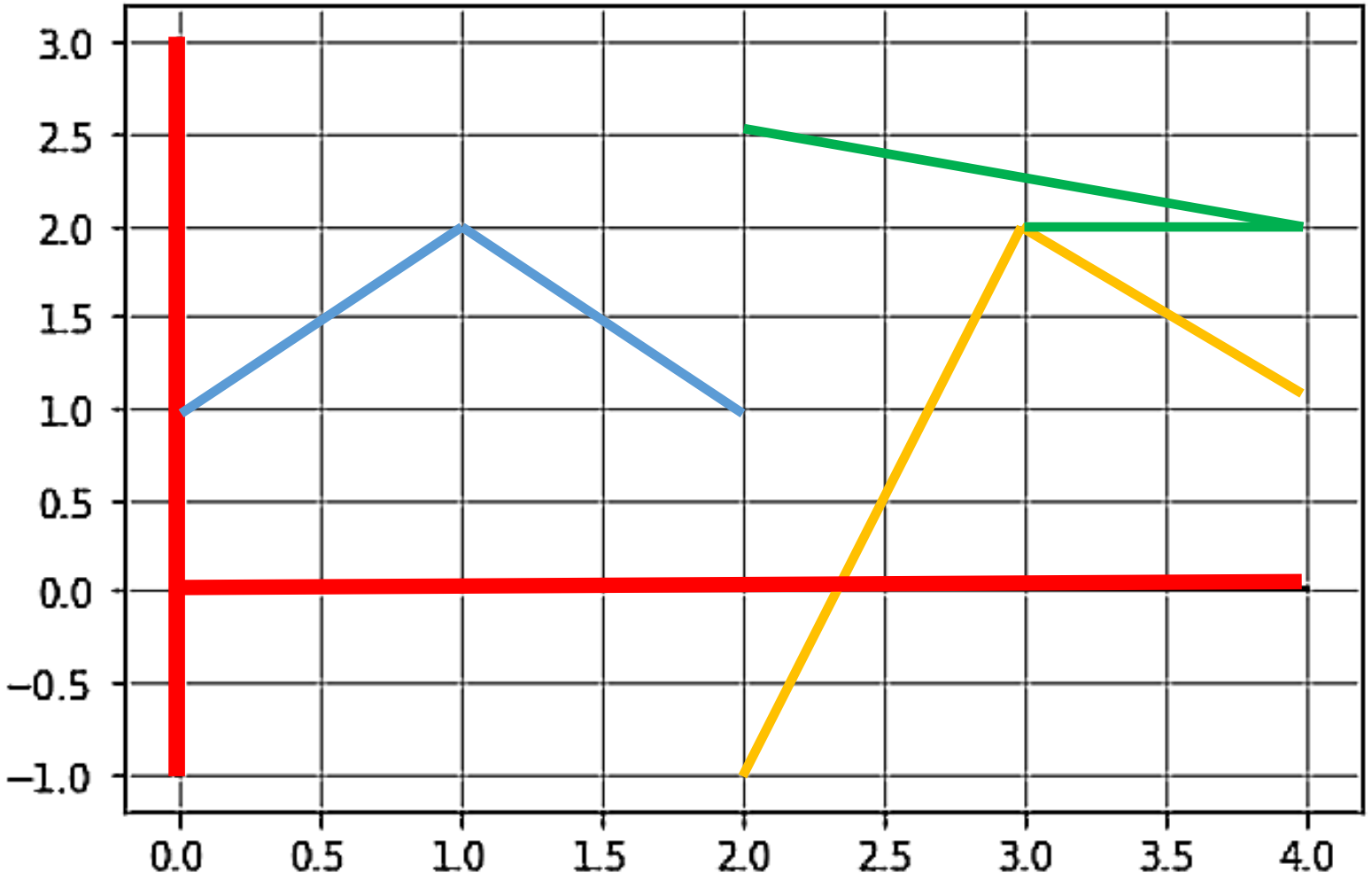
- ごく普通の代入文 x＝１＋２， x＝y． 代入操作により結果的に等号がなりたつ
- 状態変化を引き起こす代入文 x ＝ x+1 代入の前後で値が変わる
- 構造を持つデータに対するマッチングも代入の拡張と見做せる

点 (ax,ay) ． 対になったデータ． 点 a, b, c など，点を変数で保持したい．．．

使い方

```
a = (2,3) # 左辺の変数 a に 対 (2,3) を保持させる
          #「対 a」とか「点 a」とか呼べる
(a1,a2) = a  # 右辺が持つ対データのうち
             最初のデータ(この場合は2)を 変数 a1 に
             もう一方(この場合は3)を 変数 a2 に保持させる
             a が対を保持する変数でない場合は，マッチングできないのでエラー
```

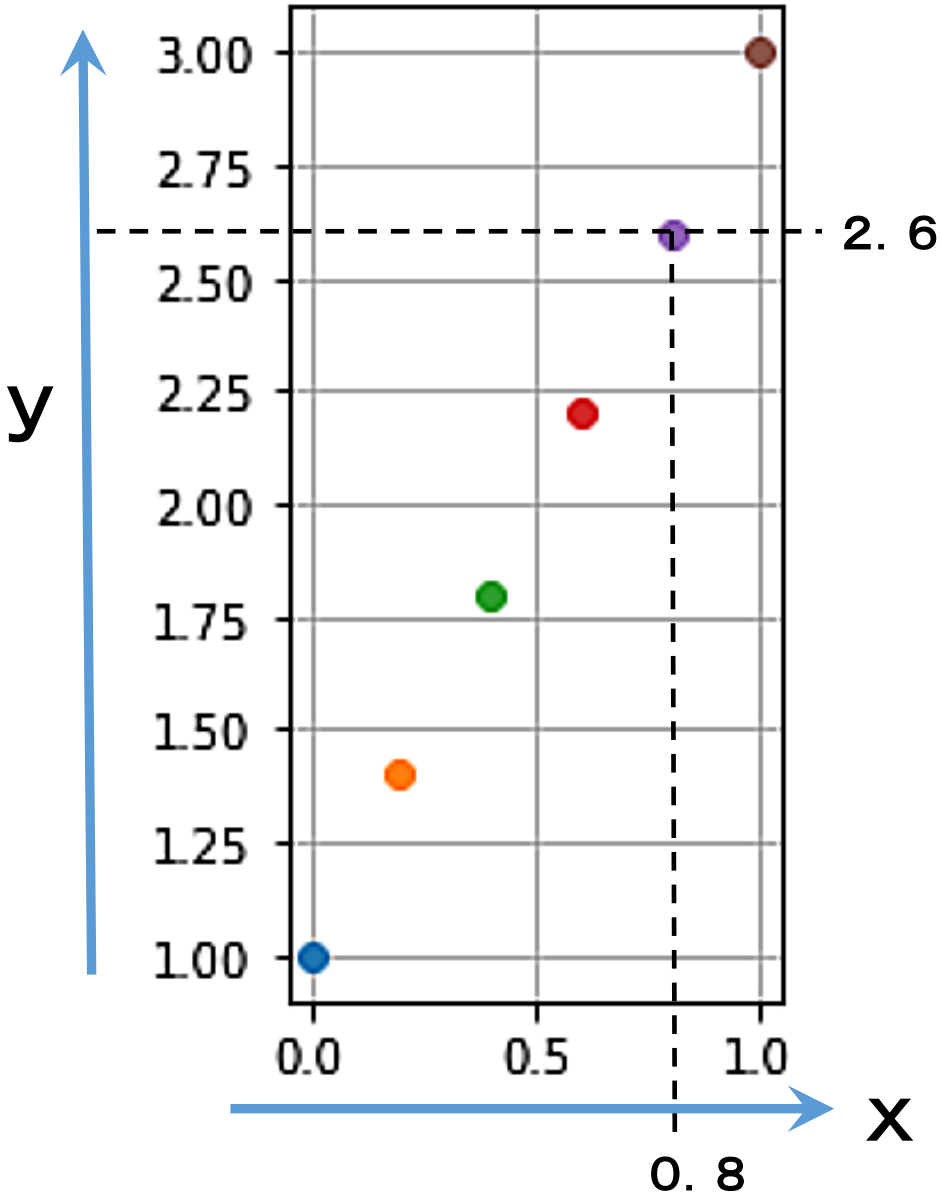
点と折線




```
import matplotlib.pyplot as plt
plt.grid()
plt.plot([1,2,1]) # plot([0,1,2],[1,2,1]) と同じ
plt.plot([2,3,4],[-1,2,1])
plt.plot([2,4,3],[2.5,2,2])# 順序に注意

def line(a,b,color):
 #点間に線分を引く関数
   (ax,ay)=a; (bx,by)=b
   plt.plot([ax,bx],[ay,by],color=color)

line((0,0),(4,0),'black')
line((0,-1),(0,3),‘red')
```

- ✓ もとい，「関数」をまだ説明していなかった．
- ✓ この場合，「２点間に線を引く」動作を定めた手順書のことです．
- ✓ その手順に従い，線を引かせます

一次関数（中学）　y = 2*x+1



- xの値に応じてｙの値も変化.
- 左辺 ＝ 右辺
  右辺の値で左辺の変数の値を決める
  結果的に、左辺と右辺の値は等しくなる

プログラミング言語における代入文と同じ働き

```
import matplotlib.pyplot as plt; import numpy as np
ax = plt.figure().add_subplot(aspect='equal')
ax.set_xlim(-0.2, 1.2) #ax.set_ylim(0.5, 3.2)
plt.grid()
def y(x) :
   return 2*x+1   # 関数 y(x)=2*x+1 の定義
xs = np.arange(0.0,1.1,0.2) # [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
for x in xs :
    plt.plot(x,y(x),marker='o') # 点を描く
```

普通の関数は return「数」だが
return「構造物」もOK

マグロとシャリがあるから握って

計算とは調理なり

return マグロの寿司

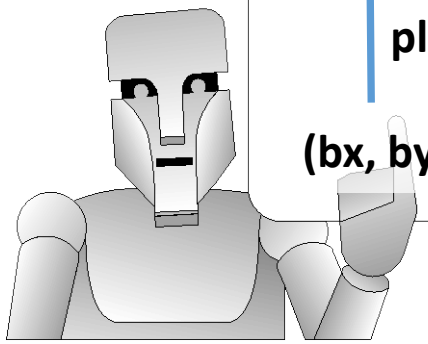
はいよ、これ

いきなり関数？

最初から使った方が良いのです

- 手続き、手順を述べたもの.
- 値を計算する「関数」を含む
- 処理対象は 変数で参照し、こうしてああしてこうやってを記述する

関数名 line



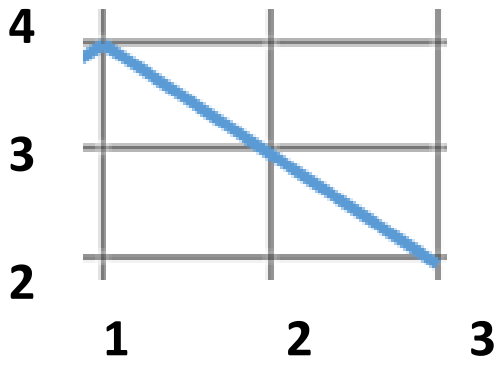
```
変数  点 a, 点 b, …
 (ax, ay) = a   # 点a が持つ座標 X=ax, Y=ay を取り出す

 plot([ax,bx],[ay,by])  # X を ax, bx  と動かし、
                        # それに応じて Y の値 ay,by をプロット
(bx, by) = b
```

具体的な点を念頭に書いて良いが、他の点にも適用できるように書く

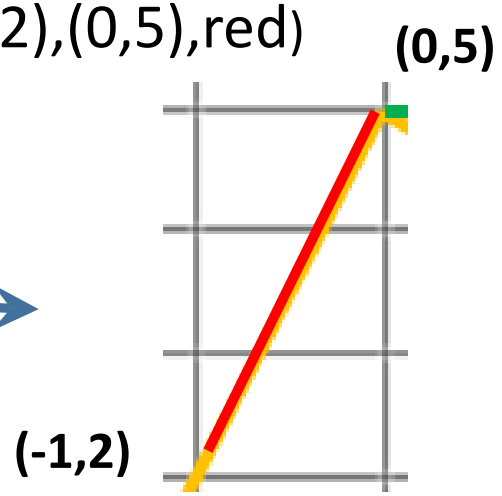
一般化、抽象化

手順書の具体化・実行
他の点に適用

line((-1,2),(0,5),red)



Plot([1,3],[4,2]

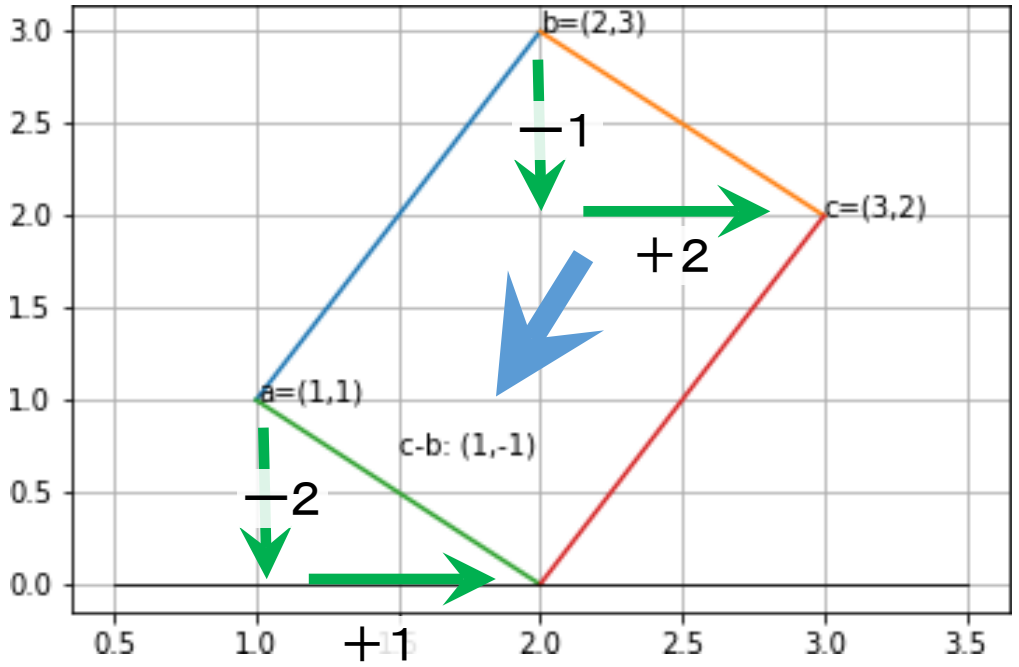


演習1．1

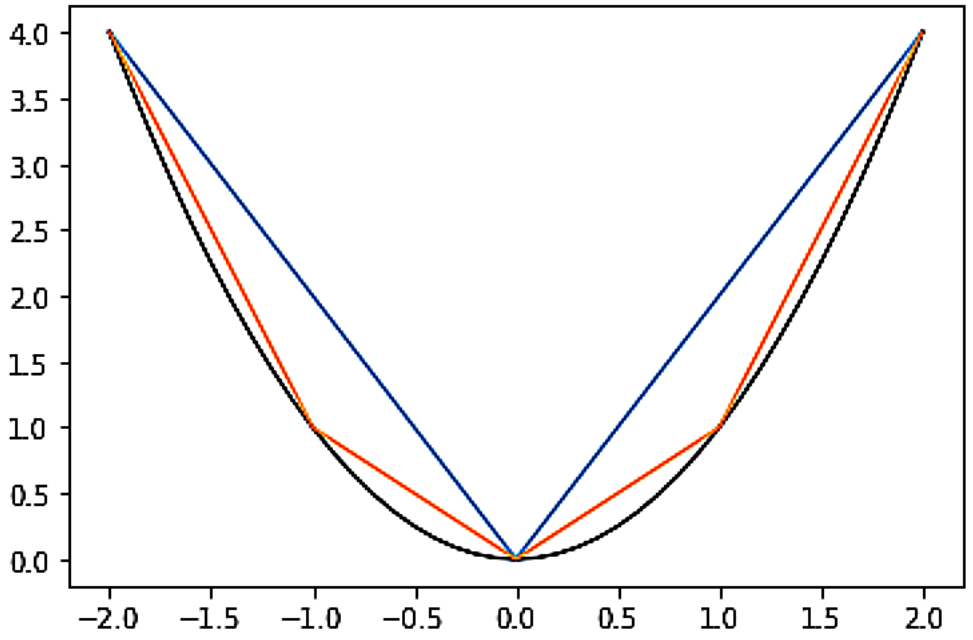
- **3つの点 a, b, c を与え、3角形を描く関数 triangle(a,b,c) を与えよ.**
  line を使う場合、line を使わない場合の2つのコードを与え、
  読みやすさの点から比較せよ. さらに
  平行四辺形を描くものに拡張せよ

```
def triangle(a,b,c):
  line(a,b); line(b,c); line(c,a)
```

```
import matplotlib.pyplot as plt
def parallelogram(a,b,c):
   d = add(a, minus(c,b))
   line(a,b);line(b,c);line(a,d);line(d,c)
def add(a,b): # a+b
    (ax,ay)=a; (bx,by)=b; return (ax+bx,ay+by)
def minus(a,b): # a-b
    (ax,ay)=a; (bx,by)=b; return (ax-bx,ay-by)
parallelogram((1,1),(2,3),(3,2))
```



曲線の折れ線近似




```
import matplotlib.pyplot as plt, numpy
x= numpy.arange(-2,2.1,0.1)
plt.plot(x,x**2,color='black')
for i in range(1,3): #[-2,2] の2等分から4等分まで
  delta = 4/2**i
  x = numpy.arange(-2, 2+delta, delta)
  plt.plot(x,x**2)
```

上記のコードでは、[-2,2] を最大 2^2=4 等分したときの折れ線近似を図示している.

.

演習：円に対し同様な図示を行い、2^n 等分や n 等分をしたときの曲線の長さなどの近似を確かめよ.

この問題は，３角関数を使った方が直観的にもわかりやすいので，しばらく三角関数の話をします．.
後で再び考えましょう

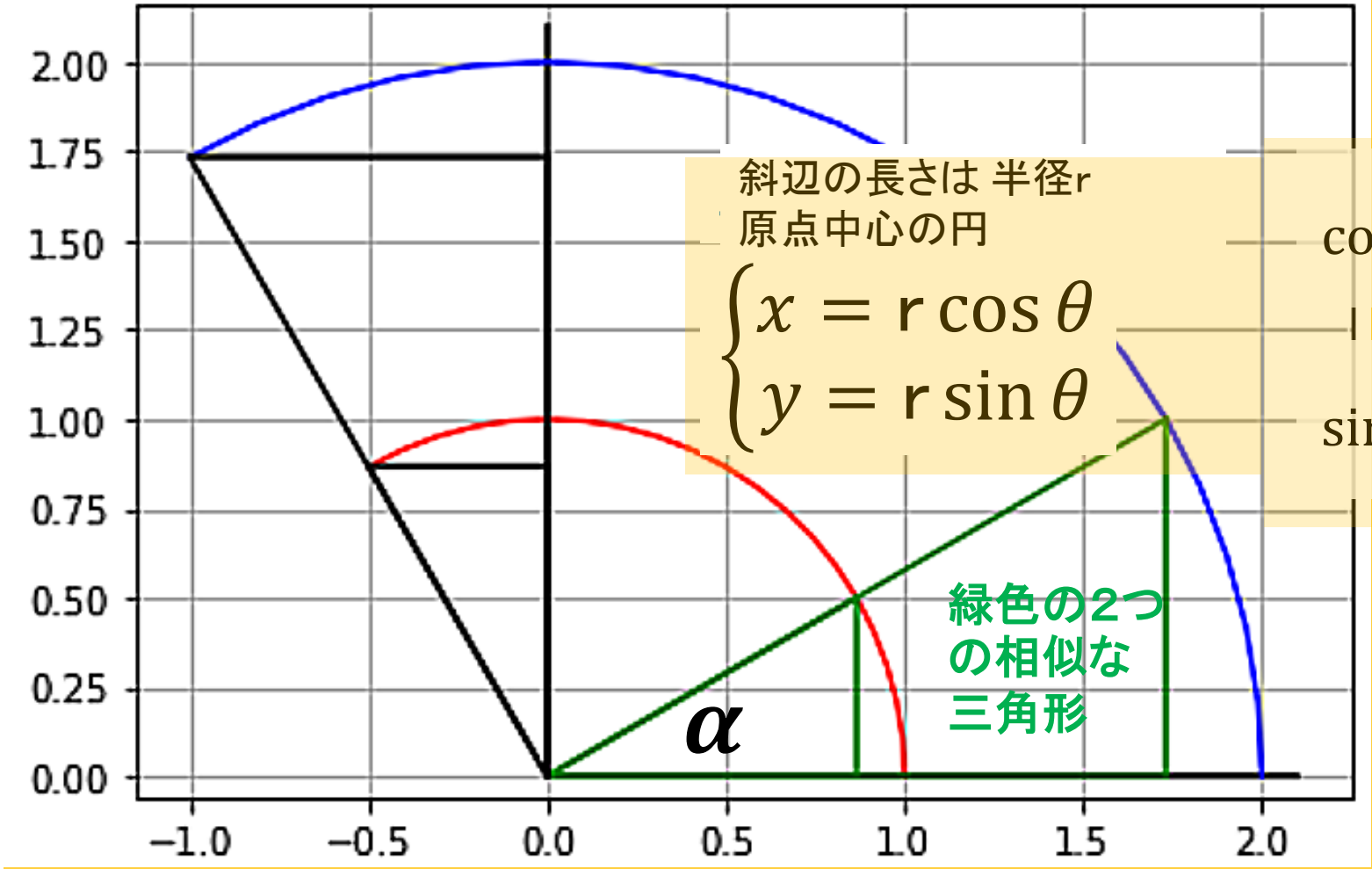
```
import matplotlib.pyplot as plt, numpy
plt.grid()
x = numpy.arange(-2, 2.1, 0.2)
# [-2.0,--1.8,-1.6,...,0.0,0.2,...,2.0,2.2,2.4,...]
plt.plot(x,x**2,color='black')
```

曲線中で最も完璧？な円 $x^2 + y^2 = r^2$ と三角関数

おまけでわかること

$$\begin{pmatrix} \cos\left(\alpha + \frac{\pi}{2}\right) \\ \sin\left(\alpha + \frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} -\sin\alpha \\ \cos\alpha \end{pmatrix}$$

緑色と黒色の三角形は，丁度90度回転させた関係（合同）.



斜辺の長さは 半径r
原点中心の円

$$\begin{cases} x = \mathrm{r}\cos\theta \\ y = \mathrm{r}\sin\theta \end{cases}$$

$$\cos\alpha = \frac{\text{底辺}}{\text{斜辺}}$$

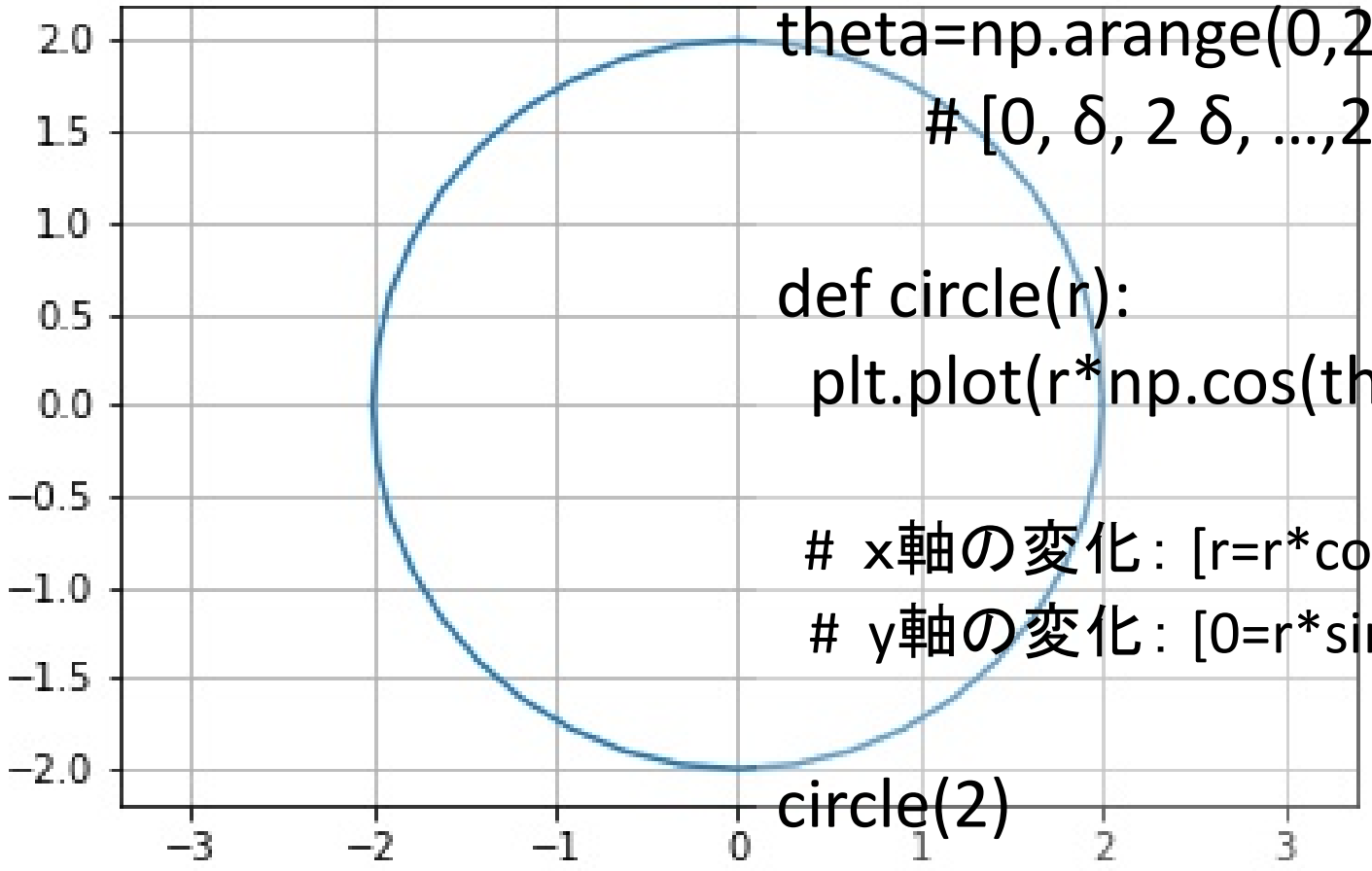
$$\sin\alpha = \frac{\text{高さ}}{\text{斜辺}}$$

相似な三角形ではこの比率は変わらない

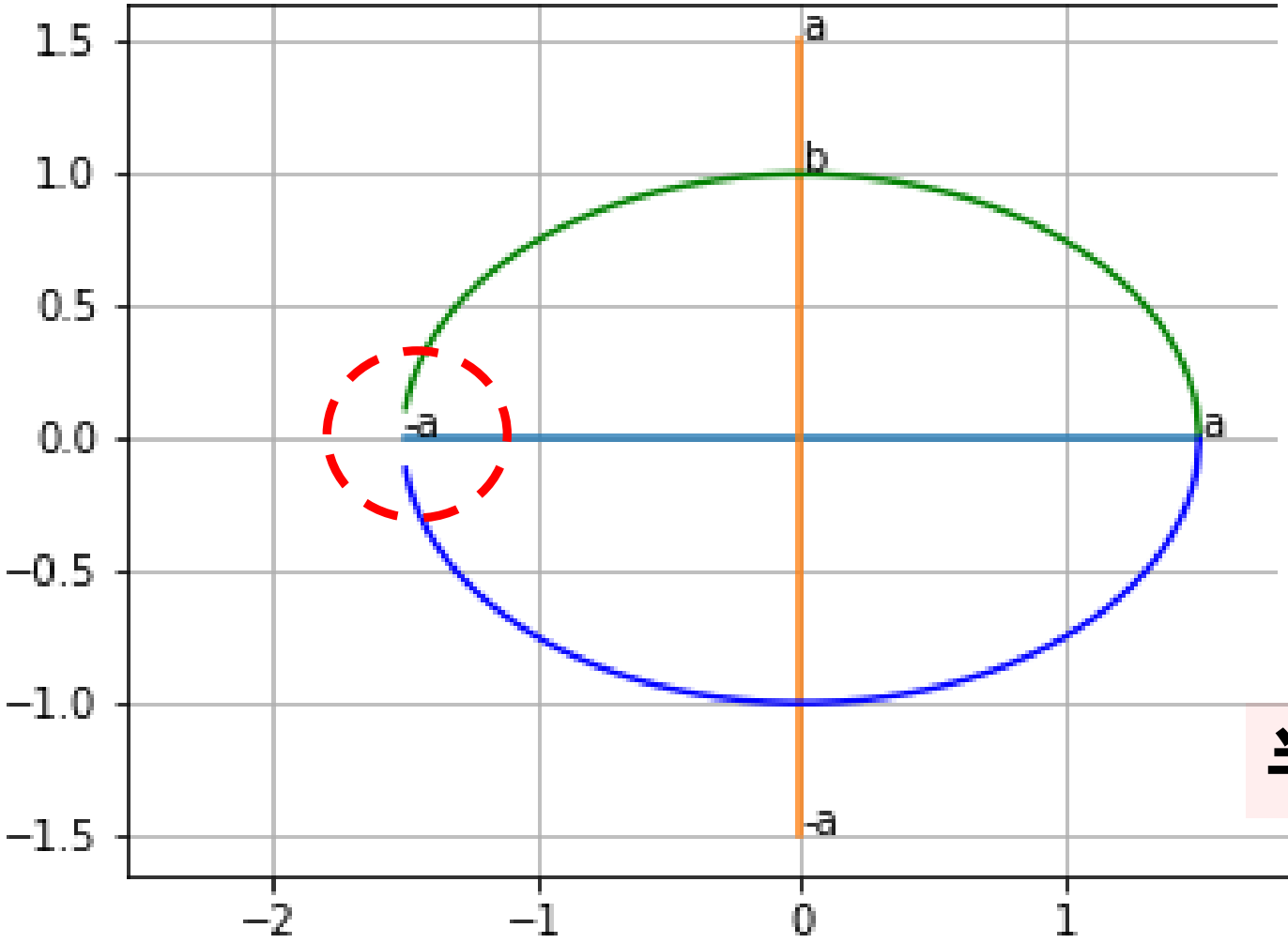
三角関数を使って円を描いてみましょう

$$\begin{cases} x = \mathrm{r}\cos\theta \\ y = \mathrm{r}\sin\theta \end{cases}$$

```
import matplotlib.pyplot as plt, numpy as np
plt.grid(); plt.axis('equal')

delta=2*np.pi/40; # 角度の増分 δ =2π/40
theta=np.arange(0,2*np.pi+delta,delta)
        # [0, δ, 2 δ, ...,2π] : 角度の変化を示す numpy 配列

def circle(r):
  plt.plot(r*np.cos(theta),r*np.sin(theta))

 #  x軸の変化 : [r=r*cos 0, r*cos δ, r*cos 2 δ,  ..., r*cos 2π =r]
 #  y軸の変化 : [0=r*sin 0, r*sin δ,  r*sin 2 δ,  ..., r*sin 2π =0]

circle(2)
```



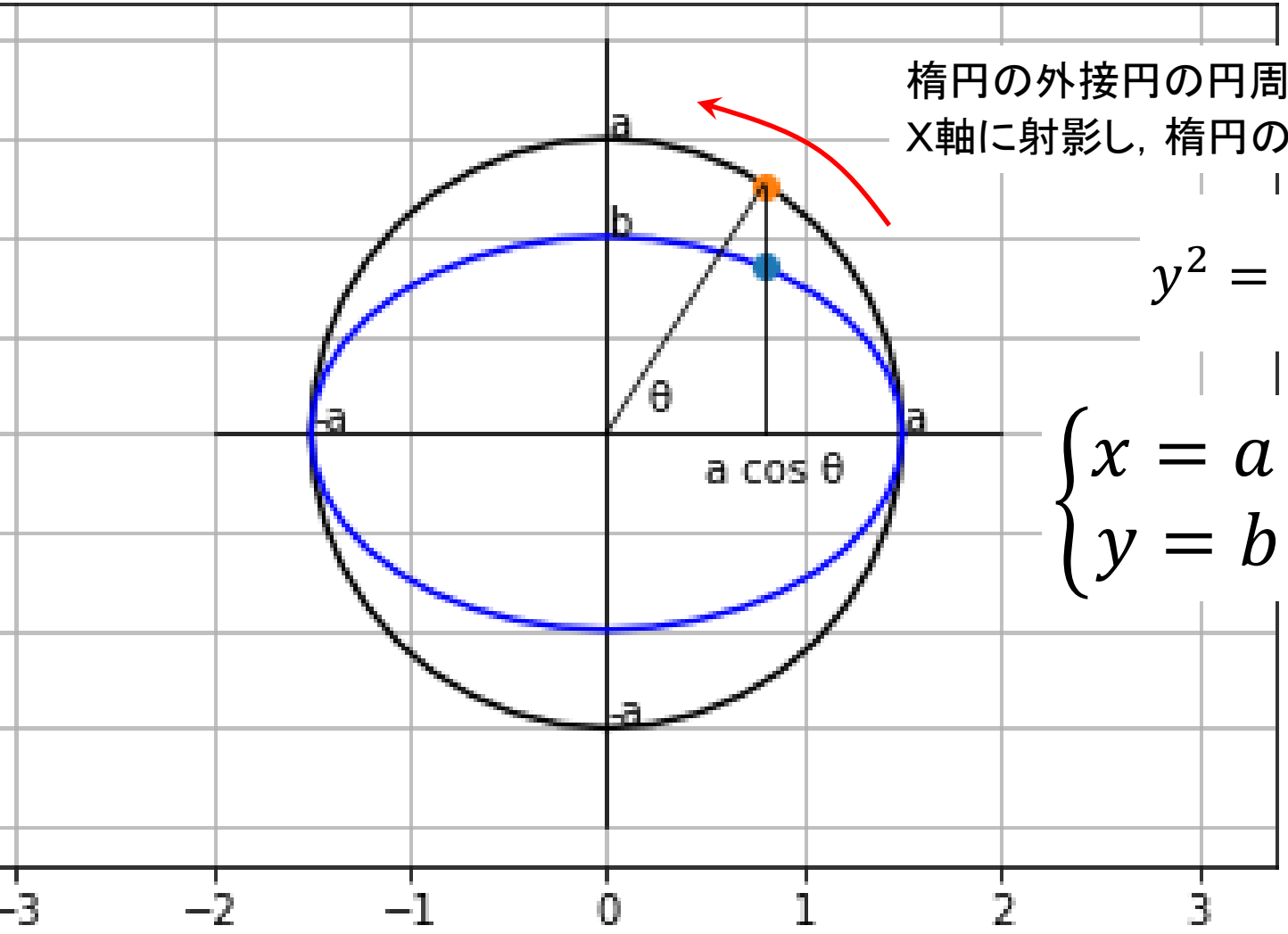
では楕円は？ $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$




```
a=1.5; b=1
xrange=np.arange(a,-a-0.01,-0.01)
def ellipseByExp():
    plt.plot(xrange,ely(xrange))
def ely(x):
    return b*np.sqrt(1-(x/a)**2)
#分割を細かくすると(-a,0)の近傍で
#　平方根計算に関する警告が！
plt.plot(xrange,ely(xrange))# X軸の上側
plt.plot(xrange,-ely(xrange));# X軸の下側
```

半径 a=1.5 の円をつぶした形

演習：$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$ は円を軸方向に縮小・拡大したもの



楕円の外接円の円周上の点を動かしながら，
X軸に射影し，楕円のX 座標 $a\cos\theta$ を得る

$$y^2=b^2\left(1-\frac{a^2cos^2\theta}{a^2}\right)=b^2sin^2\theta$$

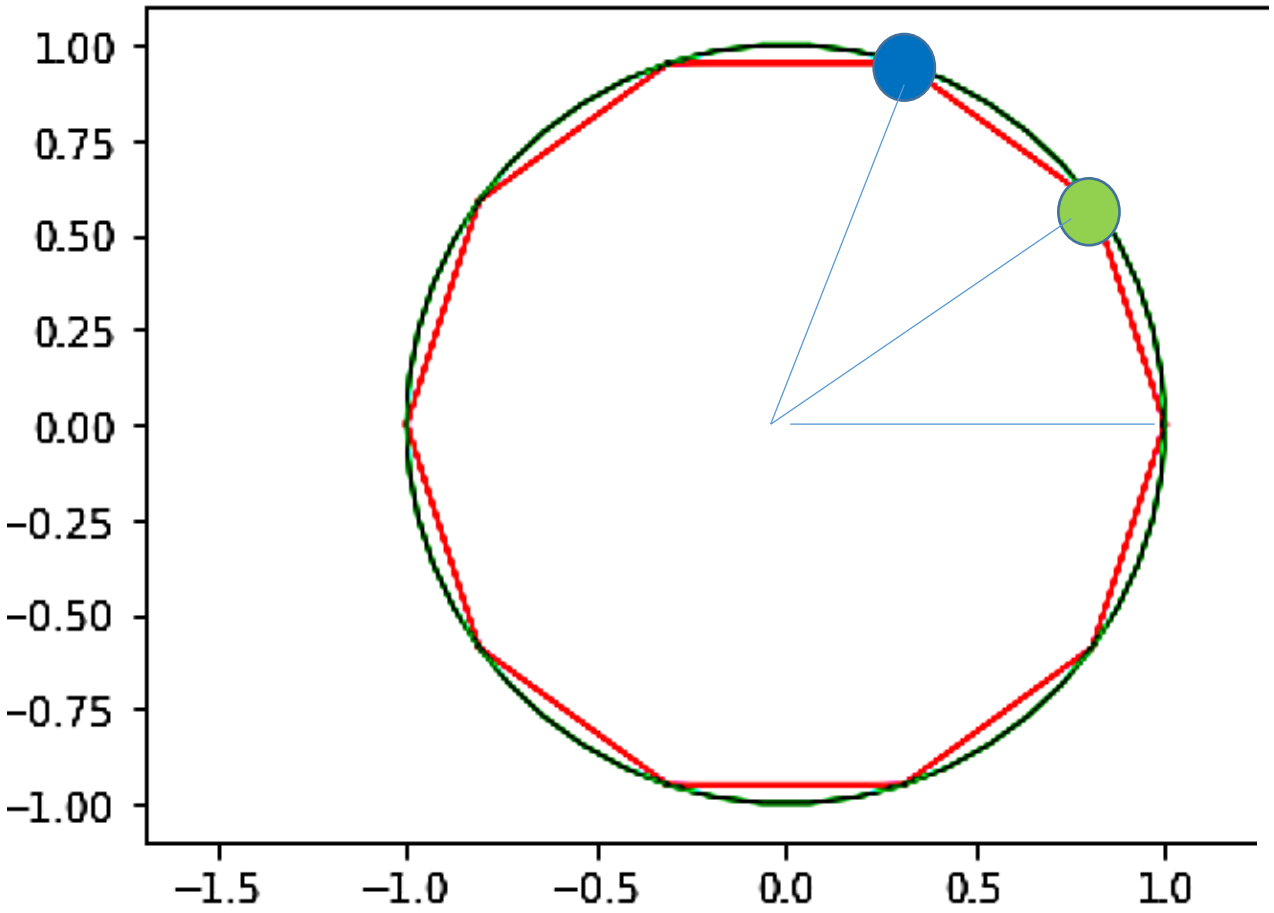
$$\begin{cases}x=a\cos\theta\\y=b\sin\theta\end{cases}$$

演習：
角度の分割により楕円を描く コードを完成させよ.
さらに，外接円と内接円も同時に描き，軸方向の拡大・縮小を実感しましょう

演習１．３ 円の近似と円周の長さ（円周率）

np.cos(theta) = […., cos θ_i, cos θ_{i+1}, …]

np.sin(theta) = […., sin θ_i, sin θ_{i+1}, …]




```
import matplotlib.pyplot as plt, numpy as np
plt.axis('equal')
def drawCircle(delta,color):
    theta =
        np.arange(0,2*np.pi+delta,delta)
    plt.plot(np.cos(theta),np.sin(theta),color=color)
drawCircle(2*np.pi/10,'red')
drawCircle(2*np.pi/50,'green')
```

- ✓ 青●と緑●の距離（円弧を近似）の総和は 2π を近似.
- ✓ このプログラムは角度の等分割なので、単純に線分長を分割数倍すれば良い（円周率の近似だけが欲しければ，図示の必要はない）
- ✓ numpy は 平方根 sqrt も実装しているので、距離計算ではこれを使う