Henon Maps

背景知识:

In mathematics, the **Hénon map** is a discrete-time dynamical system. It is one of the moststudied examples of dynamical systems that exhibit chaotic behavior. the Hénon map is given by:

$x_{n+1} = 1 - ax_n^2 + y_n$
$y_{n+1} = bx_n$

This map produces an orbit iteratively, i.e. starting from $u_0 = (x_0, y_0)$ one applies the above rule to get $u_1 = (x_1, y_1)$, and then applies the rule again on u_1 to get u_2, and so on. The orbit consists of the sequence of states $[u_1, u_2, u_3, \dots]$.

Solution:

> **Q1:** 输入任意函数系数a, b、初始值u_0以及轨迹长度N，输出**Hénon map**的轨迹$[u_1, u_2, u_3, \dots, u_n]$。
>
> - Step1:读入对应的a, b, u_0, N
> 对应的a, b, u_0在这里应当使用*float*类型进行读入
> 我们使用下面的代码进行读入:

```
def getin_float(tempstr):
    return float(tempstr)
```

类似地，对于需要使用*int*类型读入的N，我们使用 `getin_int` 函数进行读入

```
def getin_int(tempstr):
   return int(tempstr)
```

在使用时，使用 `input` 函数替换 `tempstr` ,进行读入

> - Step2: 生成储存x_n, y_n的list对
> `generate_henonmap_xy` 函数返回值为 `henon_setx` 和 `henon_sety` 两个list，用于储存x和y的值，方便后续的调用。
> `generate_henonmap_xy` 函数需要调用我们前面读入的五个输入值a, b, u_0, N,在函数体中，通过迭代的方法进行数列递归的计算，在计算后向已经初始化的两个list中写入x和y的值，这个过

> 程用到了 append 函数
> 代码如下：

```
def generate_henonmap_xy(a ,b ,x ,y ,N ):
    henon_setx = [x]
    henon_sety = [y]
    for i in range(0 ,N + 1):
        x ,y = 1 - a*x**2 + y ,b*x
        henon_setx.append(x)
        henon_sety.append(y)
    return henon_setx,henon_sety
#产生HenonMap
#为便于后续画图，此处输出的为上述形式
```

> - Step3：将上述生成的 henon_setx 和 henon_sety 使用 zip() 和 list() 操作，并将最终得到的HenonMap输出
>
> generate_henonmap 函数直接返回 list(zip(henon_setx ,henon_sety)) 完成了两个list的结合
>
> output_orbit 函数读入henonmap和N，将其中的元素按索引逐一输出
>
> 需要注意的是，这里直接输出henonmap会达到 print 函数的输出上限，无法正常输出
> 代码如下：

```
def generate_henonmap(henon_setx ,henon_sety):
    return list(zip(henon_setx ,henon_sety))
#这里生成正式的Henonmap

def output_orbit(henonmap, N):
    temp_list = henonmap
    for i in range(0 ,N + 1):
        print(temp_list[i])
#输出轨迹
```

> - Step4:最终整合
>
> 为了保证代码的可读性，我们分别读入a, b, x_0, y_0, N。对应的整合代码如下:

```
a = getin_float(input("a:"))
b = getin_float(input("b:"))
x = getin_float(input("x:"))
y = getin_float(input("y:"))
N = getin_int(input("N:"))
henon_x,henon_y = generate_henonma_xy(a ,b ,x ,y ,N )
plt.plot(henon_x,henon_y,"*")
```

把上面的代码合并就可以完成第一题的任务

当运行出现报错时,一般是 `OverflowError: (34, 'Result too large')` ,这时发生了溢出,需要调整输入值。

上述报错也体现出这段代码的局限性

一次测试的运行结果为

a: 1.3

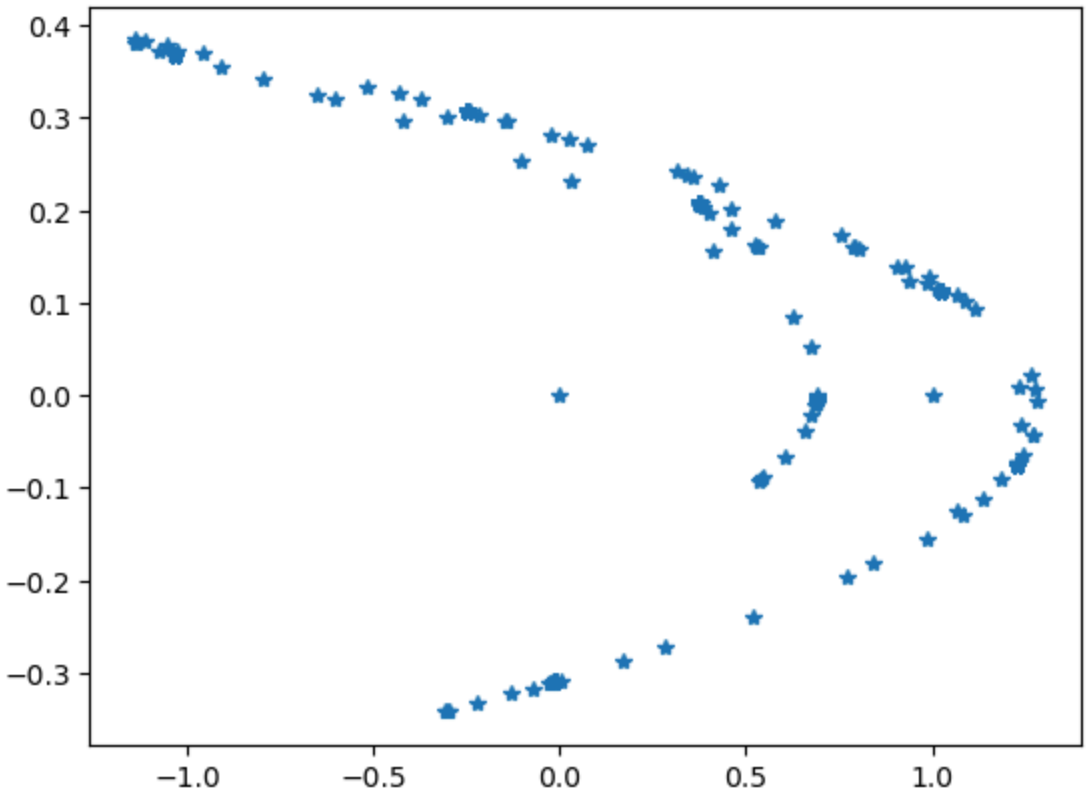
b: 0.3

x: 0

y: 0

N: 10000

[<matplotlib.lines.Line2D at 0x24a68e0c0e0>]



这里由于输出太长没有输出轨迹

在上面的代码中加上 `output_orbit(generate_henonmap(henon_x,henon_y), N)` 指令就可完成要求。对应

的部分输出如下

a: 1.4

b: 0.3

x: 0

y: 0

N = 1000

```
(-0.4734108699638832, -0.34417403604254554)
(0.3420609714375094, -0.14202326098916496)
(0.6941687475577534, 0.10261829143125283)
(0.4279999413109929, 0.20825062426732602)
(0.9517930946002273, 0.12839998239329786)
(-0.13987415050685026, 0.2855379283800682)
(1.25814723920805, -0.04196224515205508)
(-1.2580705108896282, 0.37744417176241496)
(-0.8383938027557108, -0.37742115326688846)
(-0.3614869891657429, -0.25151814082671325)
(0.5655398785027272, -0.10844609674972287)
(0.44378440740264585, 0.16966196355081817)
(0.8939395231956137, 0.13313532222079374)
(0.014356302637112212, 0.2681818569586841)
(1.2678933121631124, 0.004306890791133664)
(-1.246267940647993, 0.3803679936489337)
(-0.7940892981928516, -0.3738803821943979)
(-0.2566893211005796, -0.23822678945785547)
(0.6695280399482375, -0.07700679633017389)
(0.295418288882126, 0.20085841198447124)
PS D:\code (1)>
```

> **Q2:** 利用编写的函数计算经典Hénon map的轨迹,参数取值为$a=1.4, b=0.3, u_0=(0,0)$探索N的取值，求解得到的轨迹，并用绘制轨迹图（x为横坐标，y为纵坐标）。
> 利用 **Q1** 中的函数,只要稍加修改最终的整合部分,就可以完成要求对应的代码如下:

```
N = getin_int(input("N:"))
henon_x,henon_y = generate_henonmap_xy(1.4 ,0.3 ,0 ,0 ,N )
plt.plot(henon_x,henon_y,"*")
```

对轨迹的输出与 **Q1** 同理，加上 `output_orbit(generate_henonmap(henon_x,henon_y), N)` 指令

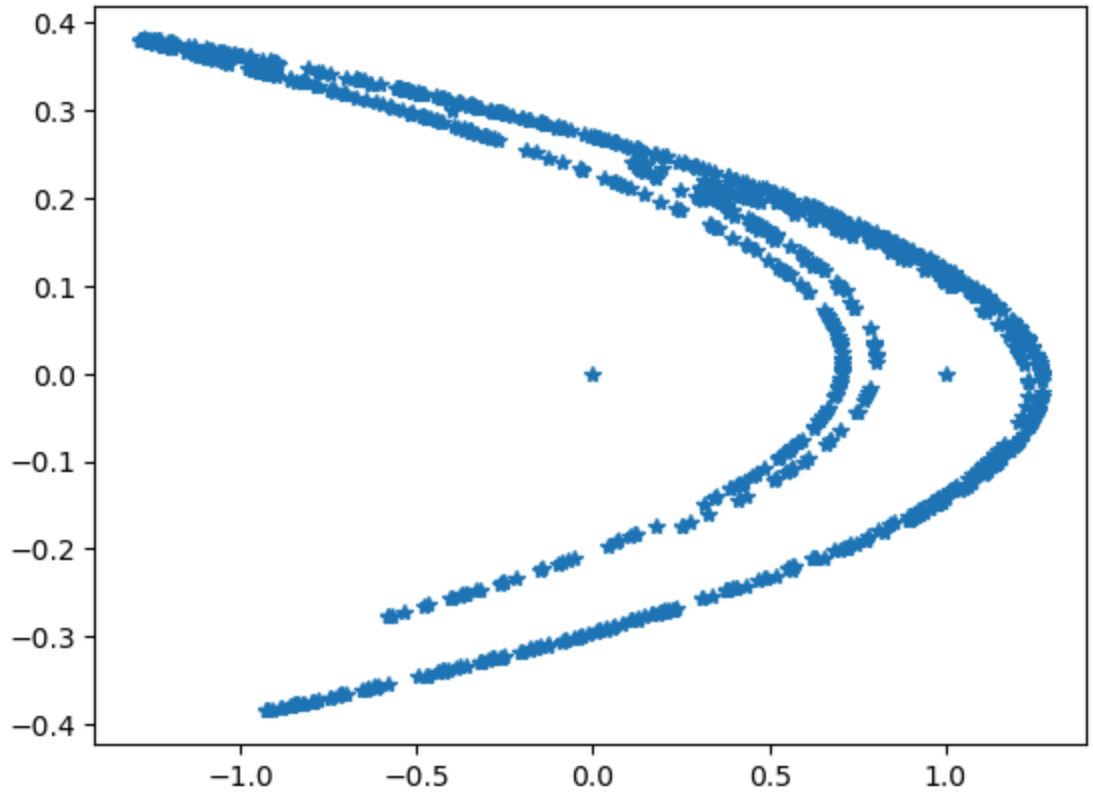
轨迹输出只截取部分

```
(-0.4734108699638832, -0.34417403604254554)
(0.3420609714375094, -0.14202326098916496)
(0.6941687475577534, 0.10261829143125283)
(0.4279999413109929, 0.20825062426732602)
(0.9517930946002273, 0.12839998239329786)
(-0.13987415050685026, 0.2855379283800682)
(1.25814723920805, -0.04196224515205508)
(-1.2580705108896282, 0.37744417176241496)
(-0.8383938027557108, -0.37742115326688846)
(-0.3614869891657429, -0.25151814082671325)
(0.5655398785027272, -0.10844609674972287)
(0.44378440740264585, 0.16966196355081817)
(0.8939395231956137, 0.13313532222079374)
(0.014356302637112212, 0.2681818569586841)
(1.2678933121631124, 0.004306890791133664)
(-1.246267940647993, 0.3803679936489337)
(-0.7940892981928516, -0.3738803821943979)
(-0.2566893211005796, -0.23822678945785547)
(0.6695280399482375, -0.07700679633017389)
(0.295418288882126, 0.20085841198447124)
PS D:\code (1)>
```

轨迹图的几次实验输出如下

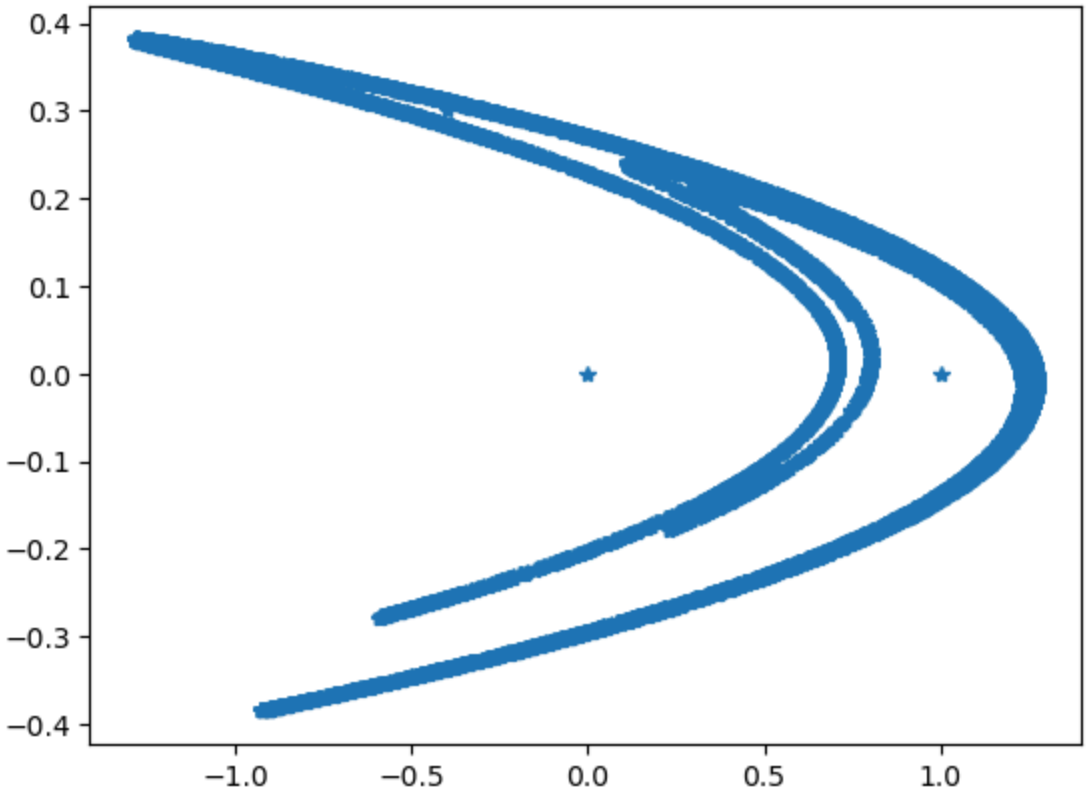
N: 1000

[<matplotlib.lines.Line2D at 0x24a68d2a180>]



N: 10000

[<matplotlib.lines.Line2D at 0x24a68ebc1a0>]



在$N = 10000$后轨迹图的形状大致不变,后面将以$N = 10000$作为标准的N输入值

> **Q3:** 固定$b = 0.3$,改变a后获得一系列**Hénon map**的轨迹,然后以a为横轴,x为纵轴绘制orbit digram图.
> 使用plot函数要求list大小相同，对应每一个a都应当有一个与 `henon_setx` 相同 `size` 的 `A` ， 需要在对 `henon_setx` 进行 `append` 时候同时为 `A` 添加相同的 `a` 我们使用循环的嵌套,
> 在 `generate_henonmap_xy` 的基础上进行改写,得
> 到 `ploting_orbit_diagram(start,end,step,b,N)` ，`start` ，`end` 为始末点, `step` 表示我们设置的测试步长.函数代码如下:

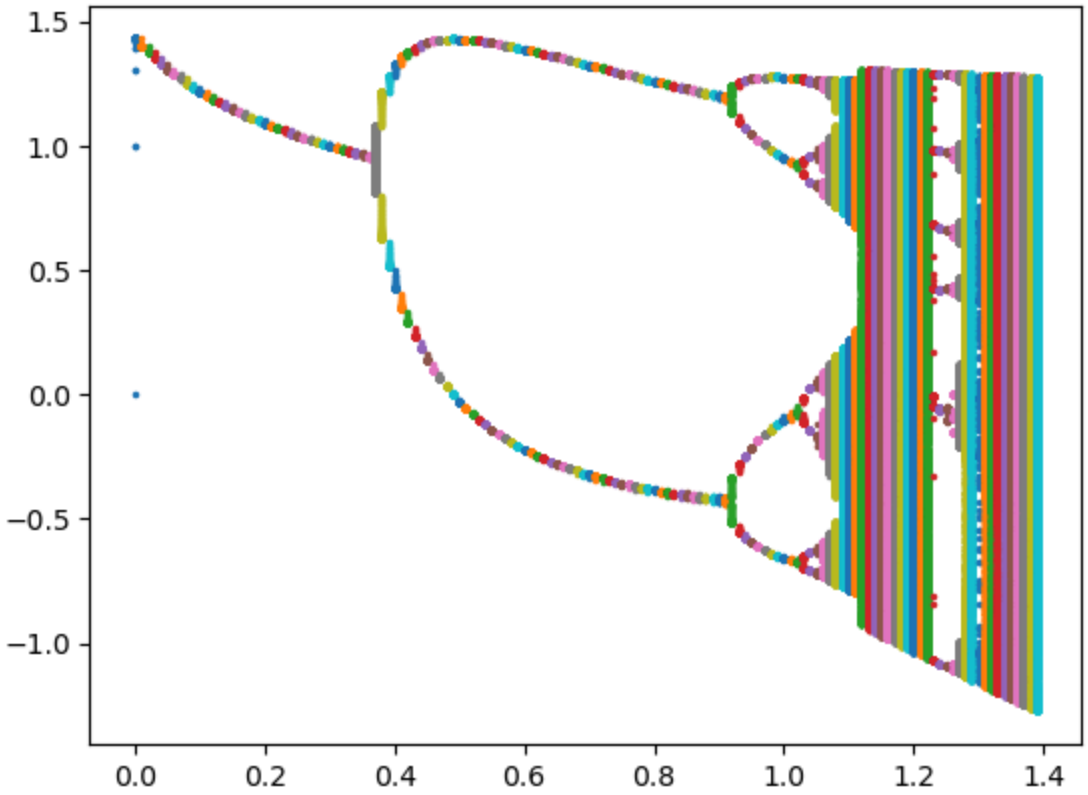
```
def ploting_orbit_diagram(start,end,step,b,N):
    a = start
    while a <= end:
        A = [a]
        henon_setx = [x]
        for i in range(0,N+1):
            x ,y = 1 - a*x**2 + y ,b*x
            henon_setx.append(x)
            A.append(a)
        plt.plot(A,henon_setx,'o',markersize=1)
        a += step
```

我们设置为 `start = 0,end = 1.4,step = 0.01,b = 0.3` 进行下面的测试.需要注意的是, `end > 1.4` 会很容易超过上限, `end` 最好不要超过 `1.4`

测试代码指令为:

```
ploting_orbit_diagram(0,1.4,0.01,0.3,10000)
```

输出结果如下:



> **Q4:** 分析上述画出的orbit digram，找到**Hénon map**可以收敛到一条周期性轨道的a值，计算该a值对应的**Hénon map**的轨迹并绘图.

在上图中,我们不难知道:当某个a值对应的x在orbit diagram中表现出明显的连续时,其对应的轨迹图就应当同我们在**Q1**中得到的相似,此时**Hénon map**收敛到周期性轨道,对应的$[u_n]$在轨道图上来回移动.

因此,我们选择$a = 1.2$作为我们找到的a值,对于轨道的输出,使用下面的代码指令

```
henon_x,henon_y = generate_henonmap_xy(1.2,0.3,0,0,10000)
output_orbit(generate_henonmap(henon_x,henon_y), 10000)
```

输出的部分结果为:

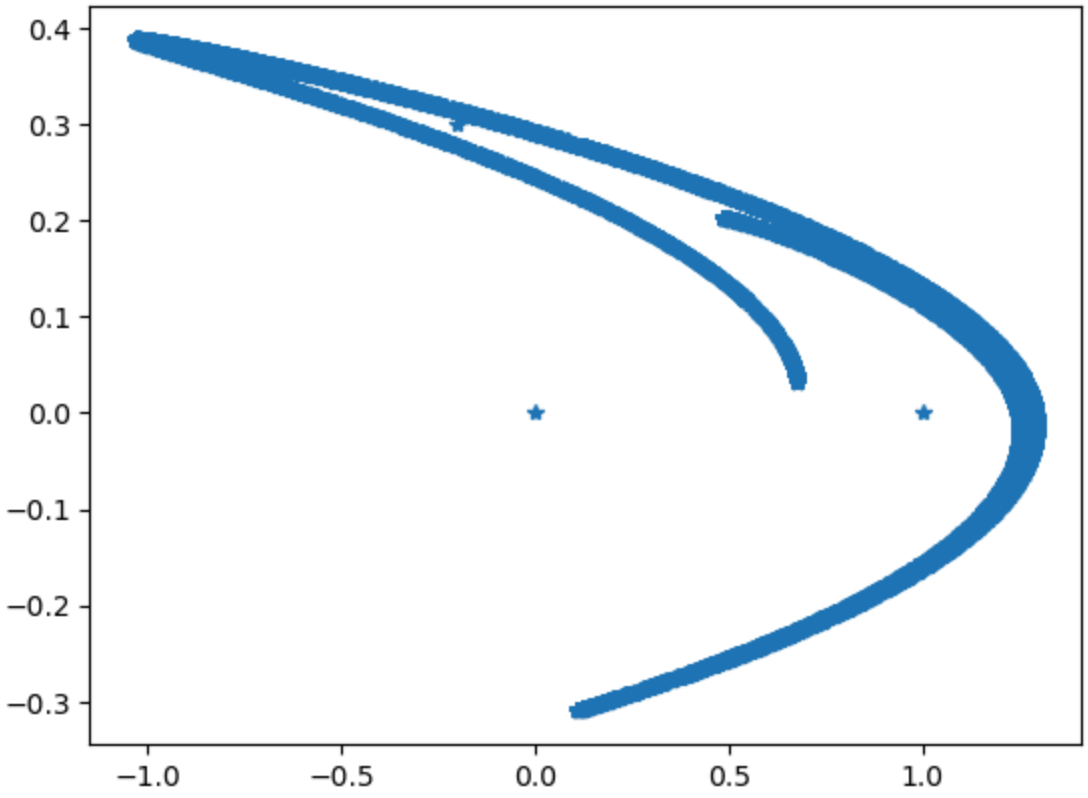
```
(0.5079471464354843, 0.20060670738535022)
(0.8909943430990086, 0.15238414393064528)
(0.1997390406093247, 0.2672983029297026)
(1.2194234817174623, 0.059921712182797 41)
(-0.7244706411339285, 0.3658270445152387)
(0.7359977926772323, -0.21734119234017854)
(0.13262750666891168, 0.22079933780316968)
(1.199691271172915, 0.039788252000673506)
(-0.6873227233535083, 0.3599073813518745)
(0.7930123501061745, -0.2061968170060525)
(0.039160878088845 97, 0.23790370503185235)
(1.236063415784625, 0.011748263426653 79)
(-0.821675057982732, 0.3708190247353875)
(0.5606391436422764, -0.2465025173948196)
(0.3763179833444464, 0.16819174309268292)
(0.9982534735865657, 0.11289539500333391)
(-0.08291660202983918, 0.2994760420759697)
(1.2912258466053599, -0.02487498060895175)
(-1.0255920049390255, 0.38736775398160794)
(0.12516100126778806, -0.30767760148170764
(0.6735240670322661, 0.037548300380336415)
(0.4931866977343151, 0.20205722010967983)
(0.9101774775231853, 0.14795600932029454)
(0.15384836061173227, 0.27305324325695557)
(1.2446500615814544, 0.046154508183519675)
(-0.8128300227701423, 0.3733950184744363)
(0.5805638433746243, -0.24384900683104266)
PS D:\code (1)>
```

删去指令，进行画图操作:
只需要输入:

```
henon_x,henon_y = generate_henonma_xy(1.2,0.3,0,0,10000)
plt.plot(henon_x,henon_y,"*")
```

输出轨道图为

[<matplotlib.lines.Line2D at 0x1ebdf171220>]



可以看到,在orbit diagram中的a值对应的x的表现,实际上为轨道图在x轴上的投影