基于蒲丰投针问题计算圆周率并衡量误差

1　蒲丰投针问题与蒙特卡洛

抛针问题主要内容是：在平面上以等距离重复地画出一些平行线，向这个平面抛出某一长度的针，求针与任一平行线相交的概率。

1.1　蒲丰投针问题的概率求解

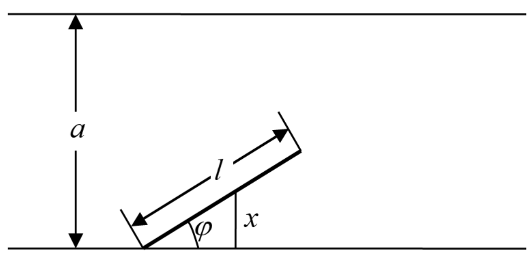

图 1: 试验方法

如图 1，假设平行线距离为 a, 针长度为 l, 落下后中点为 M, 则 x 表示中点 M 到最近的一条平行线的距离,φ 表示针与平行线的夹角。显然，可以得到自变量 x,φ 的取值范围:

$$\begin{cases} 0 \le x \le \frac{a}{2} \\ 0 \le \varphi \le \pi \end{cases} \tag{1}$$

相交的条件为:$0 < x < \frac{l}{2} sin\varphi$ 如图 2, 方程组 (1) 构成了一个矩形区域，面积为 $S = \frac{a\pi}{2}$;
满足相交条件的区域为弧线下方深色部分，面积为 $g = \int_0^{\pi} \frac{1}{2} l sin\varphi d\varphi = l$

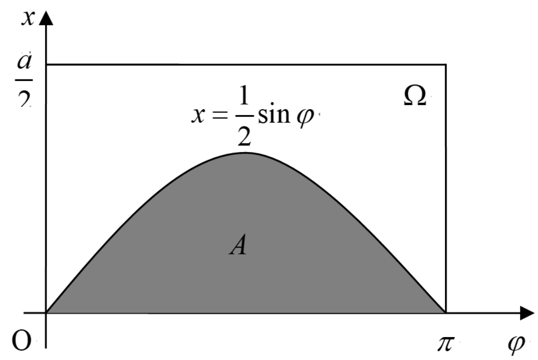

图 2: 试验原理

由概率的几何意义可知，事件 A: 针与平行线相交的概率 $P(A)=\frac{g}{s}=\frac{2l}{a\pi}$

1.2 蒙特卡洛

高中数学中曾提到，当事件发生的次数足够多时，便可以用频率作为概率的估计值。这就是蒙特卡洛的基本思想: 为了求解问题，首先建立一个概率模型或随机过程，使它的参数或数字特征等于问题的解：然后通过对模型或过程的观察或抽样试验来计算这些参数或数字特征，最后给出所求解的近似值。解的精确度用估计值的标准误差来表示。在蒲丰投针问题中，令 $a=2, l=1$, 此时概率 $P=\frac{1}{\pi}$

2 使用 python 进行模拟

用蒙特卡洛估算 π 的要领在于: 要用计算机生成足够随机的随机数，模拟投针的过程; 用足够精确的数据类型存储坐标，避免坐标分辨率不够引入的误差; 实验足够多次。

2.1 随机数的生成与绘制

所有随机数均由 numpy 库中的 numpy.random.random 生成。绘制由 matplotlib.pyplot 库中的函数完成 random.random 函数返回一个 [0.0,1.0) 区间内随机的浮点数（不同于 c 语言中的 srand 和 rand 函数，它已经封装

好，不需要重新播种即可生成完全无规律的随机数。matplotlib 函数画出三条平行线，设定坐标轴的范围，便于观察。

```
import numpy as np
import matplotlib.pyplot as plt
a=2;l=1
fig,ax=plt.subplots()
ax.plot([-4,4],[-a/2]*2,"b")
ax.plot([-4,4],[a/2]*2,"b")
ax.plot([-4,4],[3*a/2]*2,"b")
ax.set_ylim([-5,5])
```

设定 n 表示投针次数。y 为针中心的坐标，theta 为旋转的角度，ys,yx 为针上下两端点的 y 坐标。x 为针中心的横坐标，同样随机生成。

```
n=1000
y=np.random.random(size=(n))*a
theta=np.random.random(size=(n))*(np.pi)
ys=y+l/2*np.sin(theta)
yx=y-l/2*np.sin(theta)
x=np.random.uniform(low=-4,high=4,size=(n))
```

之后，判断针是否与平行线相交，用绿色表示相交，红色表示未相交，并绘出，如图 3、图 4所示。为了更直观地展现这一过程，我做了一个动画，放在附件中。

```
def flag(y,ys,yx):
    return (y>a/2 and yx<a/2) or (y<a/2 and ys>a/2)
flag=np.vectorize(flag)
f=flag(y,ys,yx)
def color(flag):
    if flag:
        return "green"
    return "red"
for i in range(n):
    ax.plot(x[i],y[i],"o",color=color(f[i]))
```

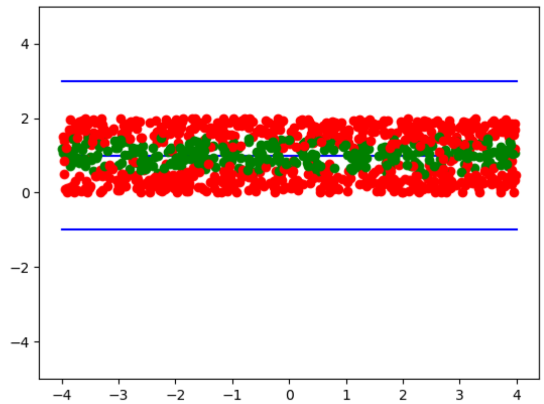

图 3: 随机投针 1000 次的结果

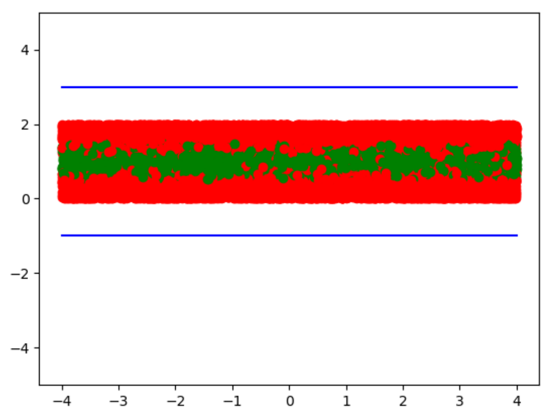

图 4: 随机投针 20000 次的结果

2.2 π 值的计算与误差估计

当 n 取值足够大时，概率 P 将接近 $\frac{1}{\pi}$. 计算相对误差:$err = \frac{|\frac{1}{P}-\pi|}{\pi}$
为了使相对误差更可感知，可以借用电子学中"信噪比"的概念，将它以 10 为底取对数，取绝对值再乘上 10，单位是 dB，称为"误差率". 这个数可以衡量有效位数。n,err,result 的关系记于表 1.

n	20	200	2000	$2*10^4$	$2*10^5$	$2*10^6$	$2*10^7$	$2*10^8$
err	5.635	8.027	18.167	22.361	28.369	35.214	37.774	39.759
result	3.3333	3.0769	3.2742	3.1113	3.1377	3.1423	3.1413	3.1415

表 1: 不同次数 n 下的误差率 (err)

从中可以看出，即使 n 取 2 亿之大, 相对误差仍有 $10^-3.9759$ 之大，即有效位数仅有 4 位，仅有 3.141 是可靠的。

3 浮点数位数对误差的影响

x,y 坐标以浮点数的形式存储于内存中。浮点数的位数是有限的，如 float 类型占 4 个字节，即 32 位；double 类型占 8 个字节，即 64 位。因此，即使是 double 类型，仍然存在 2^-63 的误差（第一位是符号位，用于存储数据的位数是 63 位）。那么，这种由"量化"引入的误差是否会对 err 产生影响呢？为此，我改写了算法（如下），它可以产生 32bit 以内指定位数的随机浮点数作为坐标。

```
cnt=0
bit=int(input())
for i in range(n):
    y=np.random.randint(0,np.power(2,bit))
    y=float(y/np.power(2,bit))
    theta=np.random.randint(0,np.power(2,bit))
    theta=float(theta/np.power(2,bit))*np.pi
    ys=y+1/2*np.sin(theta)
    yx=y-1/2*np.sin(theta)
    if (y> a/2 and yx<a/2) or (y<a/2 and ys> a/2) :
        cnt+=1
```

```
result=n/cnt
```

运行结果如图 5

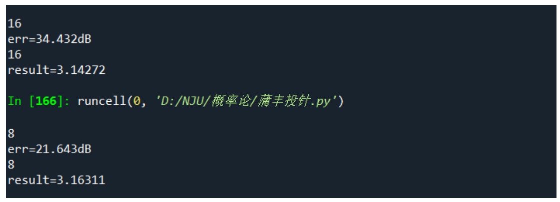

图 5: 不同位数浮点数投 2000 万次的结果

结果记录于表 2

bit	8	16	32	64
err	17.589	32.432	37.526	37.778

表 2: 不同浮点数位数投 2000 万次的结果

绘图于图 6

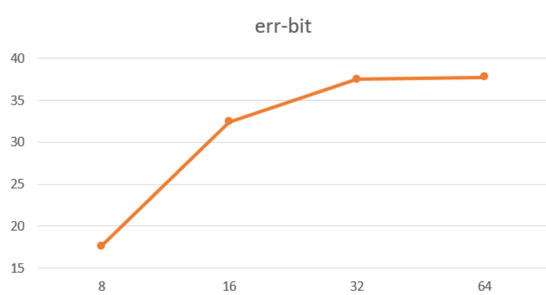

图 6: 不同浮点数位数（精度）与误差的关系

从这个结果中可以看出，当 bit 取大于等于 32 的值时，相对误差率 err 受精度的影响较小，不再继续增大，而是接近 37dB（具有 3 位半的有效位数）。但是当 bit 取小于 32 的值时，坐标产生的精度不足，不能足够稠密地覆盖这个几何空间，因此会显著地降低最终结果的精度。

4 总结

对于现代计算机而言，绝大多数 cpu 和编译器的位数都支持 32bit 模式和 64bit 模式。因此可以得出结论，用计算机对蒲丰投针进行蒙特卡洛是可靠的，只要次数足够多，那么结果的精度就不会受到浮点数位数的影响。换而言之，现代计算机的运算精度可以足够精确地覆盖这个几何空间，而可以近似地把连续问题当作离散问题求解（量子化），而不引入额外的误差。最终给出的 π 值为 3.142，此时投针次数为 2 亿次，相对误差接近万分之一,π 的计算值具有接近 4 位的有效位数。