第六次计算物理作业

零、信息

- 姓名：韩懿杰
- 学号：20213006413

一、题目：蒲丰投针

二、分析

该问题可视为针与线相交的概率积分结果含 π,下面简化模型进行模拟。

1. 设针长为1，直线间距为1。
2. 所谓平行线有一个维度是没有用的，可以取等距点简化。
3. 针本来也有三个自由度 (x,y,θ)，但是我们可以配合平行线去除y自由度。

此外边界条件很是问题。

1. 采取平行线上界为实线，下界为虚线。即上界线可以相交，下界线不可以。
2. 而随机投针的最下端位置在这上下界中间。
3. 这种边界规定平衡有界系统的边界效应，使其和无界等价。
4. 此外这种规定还有一个优点：我们可以直接规定只有一条实线，一条虚线。

综上，问题简化为

1. "实平行线"(当然现在只是个点了)取值为 $x_{line}=1$，虚线为 x=0。
2. 针的最下端取值范围是 $x_d=[0,1]$,角向取值范围是 $\theta=[0,\pi/2]$（其他情况重复，单调性有利于后续优化。） 则针的最高点坐标为 $x_m=x_d+|\sin\theta|$
3. 显然相交条件为 $x_m>1$
4. 理论概率期望是 $2/\pi$,由此计算 π 的数值

三、解答正文

（一）基础功能

其基础功能实行较为简单。

此次估计PI值结果为3.1418276781351495，可见其收敛性极差。

In [ ]:
```
import numpy as np
from scipy import stats

# self-definition variable
x_line=1
N=100000000 #选取N个样本点。

# calculate
## random of x_d and theta
low = 0
high = x_line
x_d = stats.uniform.rvs(loc=low, scale=high-low, size=(N,))# 一个均匀的连续随机变量。使用参数
low = 0
high = np.pi/2
theta = stats.uniform.rvs(loc=low, scale=high-low, size=(N,))
## calaulate x_m
x_m=x_d+np.abs(np.sin(theta))
```

```
## estimate pi
cross1=x_m>x_line
probability_posi=np.sum(cross1)/N
pi_est=2/probability_posi
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为3.1418276781351495

（二）提升收敛性

1.sobol序列（不好用）

一亿次只能精确到3.1415,可见MC方法在收敛性方面极差。

均匀分布也不大好用，因为这里有转角和位置两个自由度，平方倍复杂度。

Sobol序列是一种折中的分布，是一种低方差（相对于完全随机）的伪随机，可以尝试。

尝试完了，不好用，应该是高维才用的，不稳定。

In [ ]:
```
import numpy as np
from scipy.stats import qmc

# self-definition variable
x_line=1
N=2**27 #选取N个样本点。

# calculate
## random of x_d
low = 0
high = x_line
sampler1 = qmc.Sobol(d=1, scramble=True)
x_d = sampler1.random(n=N)
x_d=x_d*high
## random of theta
low = 0
high = np.pi/2
sampler2 = qmc.Sobol(d=1, scramble=True)
theta = sampler2.random(n=N)
theta=theta*high
## calaulate x_m
x_m=x_d+np.abs(np.sin(theta))
## estimate pi
cross=x_m>x_line
probability_posi=np.sum(cross)/N
pi_est=2/probability_posi
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为2.226086956521739

2.均匀取点

没啥能用的，均匀取点试一下。

均匀取10000个 x_c 和 θ $(10000^2 = 100000000)$,组合100000000种情况。

此次估计pi值为3.1417311419276883，优点是没有方差，稳定，但没啥用。

In [ ]:
```
import numpy as np
from scipy import stats

# self-definition variable
x_line=1
N=10000 #选取N个样本点。

# calculate
## random of x_d and theta
low = 0
high = x_line
```

```
x_d = np.linspace(low,high,N)
low = 0
high = np.pi/2
theta = np.linspace(low,high,N)
X_c,Theta=np.meshgrid(x_d,theta)

## calaulate x_m
X_m=X_c+np.abs(np.sin(Theta))
x_m=X_m.reshape(1,N**2)
## estimate pi
cross3=x_m>x_line
probability_posi=np.sum(cross3)/(N**2)
pi_est=2/probability_posi
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为3.1417311419276883

3.重要性抽样

此题用不了，因为很难给出 x_d 和 θ 的概率密度函数使 $x_d+\sin\theta$ 的概率密度函数在 $x_d+\sin\theta=1$ 时最大。而且就算强行计算，做卷积时大概率也没有解析解。

4.分层抽样

假如我们将随机区间分成子区间，按正常的随机抽样，投入点数理应正比于子空间占比，此时各子区间结果重要程度正比子区间占比。如果我们不想均匀取点，理应对子区间重要性加权。

$$\langle f(\vec{x})\rangle=\sum_i\frac{S_i}{S}\langle f(\vec{x})\rangle_i$$

其中 $\langle f(\vec{x})\rangle_i$ 为取点数正比于子区间占比时每个子区间的计算结果。$\langle g(\vec{x})\rangle_i$ 为取点数和子区间占比无关时每个子区间的计算结果。

$\frac{\rho_s}{\rho_i}$ 这i个区域权重需要归一

现在我们可以进一步简化，令子区间体积相同,那么计算公式简化为：

$$\langle f(\vec{x})\rangle=\propto\rho\sum_i\langle g(\vec{x})\rangle_i$$

这个操作在子区间不多时几乎不会增加数值计算复杂度，却令人心情愉悦。

- 只对 x_d 分层抽样。

对x_d 的范围 $[0,1]$ 取 5个等区间。提升敏感区间,资源分配比例如下为[1,3,2,1,1]

此次估计pi值为3.1411836530340005

In [ ]:

```
import numpy as np
from scipy import stats

# self-definition variable
N=100000000 #选取N个样本点。
x_line=1
## 空间分配
num_room=5 #空间数量
rho=1/num_room #空间占比
num_dim=1 #空间维度(划分的)
room_bound=np.array([[0,0.2],[0.2,0.4],[0.4,0.6],[0.6,0.8],[0.8,1]])
room_pro=0
## resource proportion
rescource_ratio=np.array([1,3,2,1,1])
rescource_proportion=rescource_ratio/np.sum(rescource_ratio)



# calculate
```

```
## sub_room
for i in range(num_room):
    ### 计算子区间资源
    sub_N=int(N*rescource_proportion[i])

    ### random of x_d
    low,high=room_bound[i,:]
    x_d = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))# 一个均匀的连续随机变量。
    ### random of theta
    low = 0
    high = np.pi/2
    theta = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))

    ### calaulate x_m in this room
    x_m=x_d+np.abs(np.sin(theta))
    ### estimate pi
    cross=(x_m>x_line)
    probability_posi=np.sum(cross)/sub_N
    room_pro+=rho*probability_posi

pi_est=2/room_pro
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为3.141729934440097

- 同理，可以对 θ 也空间划分，资源倾斜，即为5*5空间。

```
rescource_ratio=np.array([1*np.array([1,1,1,2,3]),\
                          3*np.array([1,1,2,3,1]),\
                          2*np.array([1,1,3,2,1]),\
                          1*np.array([1,3,2,1,1]),\
                          1*np.array([3,2,1,1,1])])
```

此次估计pi值为3.1415315137363873

In [ ]:

```
import numpy as np
from scipy import stats

# self-definition variable
N=100000000 #选取N个样本点。
x_line=1
## 空间分配
num_room=25 #空间数量
rho=1/num_room #空间占比
num_row=5 #空间维度(划分的)
num_column=5
room_pro=0
### 界限
class Bound():
    x_d=np.array([[0,0.2],[0.2,0.4],[0.4,0.6],[0.6,0.8],[0.8,1]])
    theta=np.array(np.array([[0,0.2],[0.2,0.4],[0.4,0.6],[0.6,0.8],[0.8,1]])*np.pi/2)
room_bound=Bound()
## 资源分配
rescource_ratio=np.array([1*np.array([1,1,1,2,3]),\
                          3*np.array([1,1,2,3,1]),\
                          2*np.array([1,1,3,2,1]),\
                          1*np.array([1,3,2,1,1]),\
                          1*np.array([3,2,1,1,1])])
rescource_proportion=rescource_ratio/np.sum(rescource_ratio)


# calculate
## sub_room
for i in range(num_row):
    for j in range(num_column):
        ### 计算子区间资源
        sub_N=int(N*rescource_proportion[i,j])

        ### random of x_d
```

```
        low,high=room_bound.x_d[i,:]
        x_d = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))# 一个均匀的连续随机变
        ### random of theta
        low,high=room_bound.theta[j,:]
        theta = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))

        ### calaulate x_m in this room
        x_m=x_d+np.abs(np.sin(theta))
        ### estimate pi
        cross=(x_m>x_line)
        probability_posi=np.sum(cross)/sub_N
        room_pro+=rho*probability_posi

pi_est=2/room_pro
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为3.141389234974111

- 线程优化

上面不同子区间的计算显然是平行关系，可进行线程优化。

In [ ]:
```
import numpy as np
from scipy import stats
import concurrent.futures

# self-definition variable
N=100000000 #选取N个样本点。
x_line=1
## 空间分配
num_room=25 #空间数量
rho=1/num_room #空间占比
num_row=5 #空间维度(划分的)
num_column=5
room_pro=0
### 界限
class Bound():
    x_d=np.array([[0,0.2],[0.2,0.4],[0.4,0.6],[0.6,0.8],[0.8,1]])
    theta=np.array(np.array([[0,0.2],[0.2,0.4],[0.4,0.6],[0.6,0.8],[0.8,1]])*np.pi/2)
room_bound=Bound()
## 资源分配
rescource_ratio=np.array([1*np.array([1,1,1,2,3]),\
                          3*np.array([1,1,2,3,1]),\
                          2*np.array([1,1,3,2,1]),\
                          1*np.array([1,3,2,1,1]),\
                          1*np.array([3,2,1,1,1])])
rescource_proportion=rescource_ratio/np.sum(rescource_ratio)

# calculate
## sub_room
def task(i,j):
    ### 计算子区间资源
    sub_N=int(N*rescource_proportion[i,j])

    ### random of x_d
    low,high=room_bound.x_d[i,:]
    x_d = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))# 一个均匀的连续随机变量。
    ### random of theta
    low,high=room_bound.theta[j,:]
    theta = stats.uniform.rvs(loc=low, scale=high-low, size=(sub_N,))

    ### calaulate x_m in this room
    x_m=x_d+np.abs(np.sin(theta))
    ### estimate pi
    cross=(x_m>x_line)
    probability_posi=np.sum(cross)/sub_N
    return rho*probability_posi
## 线程优化
### 创建一个线程池
with concurrent.futures.ThreadPoolExecutor() as executor:
```

```
    # 提交任务到线程池
    futures = []
    for i in range(num_row):
        for j in range(num_column):
            future = executor.submit(task, i,j)
            futures.append(future)

    # 等待所有任务完成
    results = np.array([])
    for future in concurrent.futures.as_completed(futures):
        result=future.result()  #获取任务结果
        results=np.append(results,result)
# Output
pi_est=2/np.sum(results)
print("此次估计pi值为{}".format(pi_est))
```

此次估计pi值为3.141627229666462

四、总结

可以发现费那么大劲不改变一亿次计算量的前提也就多收敛一位。可见MC方法先天就有收敛性问题，难以进行N量级以上的优化。

当然，一定有更有效资源配比，不过这是自适应这个配比是另一个问题。