使用蒙特卡洛方法计算积分的方差为$\sigma^2 = V/n$，其中V是被积函数的方差
蒙特卡洛方法的误差来自随机点分布不均匀，如果把抽样区域分层，并规定每一层抽样的点数，就可以减小随机点的不均匀性
另外，分层之后每一层的方差可能不同，在方差大的层分配更多的抽样点，更有利于减小方差
合理地划分积分区域和点数可以减小误差，但是这需要对被积函数有一定的了解，如果划分不合理，反而可能会加大误差
大多数情况下，如果对被积函数了解不够，做均匀分层抽样即可

计算积分$I = \int_0^\pi \sin^2 x dx$
先将积分区域变换到$[0, 1]$区间，计算最终结果时再恢复
原始蒙特卡洛方法

In [1]:

```
import numpy as np
from numpy.random import default_rng

rng = default_rng()
```

In [2]:

```
# 变换后的被积函数
def integrand(x):
    return np.sin(np.pi*x)**2
```

In [3]:

```
N = 10**6
X = rng.random(N)
Y = integrand(X)
I = np.pi*sum(Y)/N
print(f"结果：{I}")
I0 = np.pi/2
print(f"误差：{abs(I0-I)/I0:.4%}")
```

```
结果：1.5725494175681252
误差：0.1116%
```

均匀分层抽样

In [4]:

```
# 对积分区域进行划分
N_divide = 10
edges = np.linspace(0, 1, N_divide + 1)
# 设置每个区域的点数
N_list = [10**5 for _ in range(N_divide)]
# 分别计算每个区域的积分估计值
I_list = [0 for _ in range(N_divide)]
for i in range(N_divide):
    X = rng.uniform(edges[i], edges[i+1], N_list[i])
    Y = integrand(X)
    I_list[i] = np.pi/10*sum(Y)/N_list[i]
# 求和得到总的积分估计值
I = sum(I_list)
print(f"结果：{I}")
print(f"误差：{abs(I0-I)/I0:.4%}")
```

```
结果：1.5710738968834188
误差：0.0177%
```