蒙特卡洛求定积分的方法与讨论

1 定积分与蒙特卡洛

根据概率论中大数定理的知识可以知道，当试验次数 n 足够多时，便可以用这一系列随机数的平均值近似地估计数学期望。因此蒙特卡洛计算定积分的思路就是，先将定积分写成数学期望的形式，然后通过大量试验，根据大数定理就可以得到定积分的近似值。

1.1 方法描述・期望法

考察对一个任意连续函数的定积分:

$$\int_a^b f(x)dx \tag{1}$$

如果 R.V. X∼ $pd(x)$(pd 为 Probability Distribution 缩写,pd(x) 是区间 $[a,b]$ 上的概率分布函数), 那么有:

$$\int_a^b f(x)dx = \int_a^b \frac{f(x)}{pd(x)}pd(x)dx \tag{2}$$

$$E(\frac{f(x)}{pd(x)}) = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{pd(x)}pd(x)dx \tag{3}$$

利用大数定理，有

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \frac{f(x_i)}{pd(x_i)} = E(\frac{f(x)}{pd(x)}) \tag{4}$$

如果 $pd(x)$ 为 $[a,b]$ 上的均匀分布，即

$$pd(x) = \frac{1}{b-a}, x \in [a,b] \tag{5}$$

那么

$$\int_a^b f(x)dx = \lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} \frac{f(x_i)}{pd(x_i)} = \lim_{n\to\infty} \frac{b-a}{n}\sum_{i=1}^{n} f(x_i) \tag{6}$$

即可以在区间 $[a,b]$ 上随机均匀地取一些点 x_i，用它们的函数值估计期望，而期望即定积分的值。

1.2 方法描述・投点法

投点法的思路是用蒙特卡洛方法估算定积分的线下面积，即在包含待积函数的一块矩形矩形区域里随机投点，点的概率分布为均匀分布，这样当点数足够多时，待求定积分的线下面积

$$S = \frac{N_1}{N_0} \cdot S_0 \tag{7}$$

其中 N_1 是被积函数线下的点数,N_0 是整个矩形区域里的点数，S_0 是矩形区域的面积。当 N_0 足够大时，根据大数定理，就可以近似地估计待求定积分的线下面积。

2 使用 python 进行试验

与蒲丰投针实验类似，试验的要领在于: 使用足够好的随机数算法产生符合均匀分布的随机数; 使用足够位数的数据类型存储、运算以避免”有限字长效应”; 试验次数足够大。本文选择使用 numpy 进行随机数生成及科学计算，使用 matplotlib 作为绘图工具。

2.1 计算实例

计算这样的定积分:

$$\int_{-\infty}^{+\infty} e^{-x^2} dx \tag{8}$$

它的值是 $\sqrt{\pi}$. 为了便于计算，将它的形式做一点改变:

$$\int_{-\infty}^{+\infty} e^{-x^2} dx = 2 \cdot \int_{0}^{+\infty} e^{-x^2} dx = \lim_{b \to \infty} 2 \cdot \int_{0}^{b} e^{-x^2} dx \tag{9}$$

图 1是它的图像:

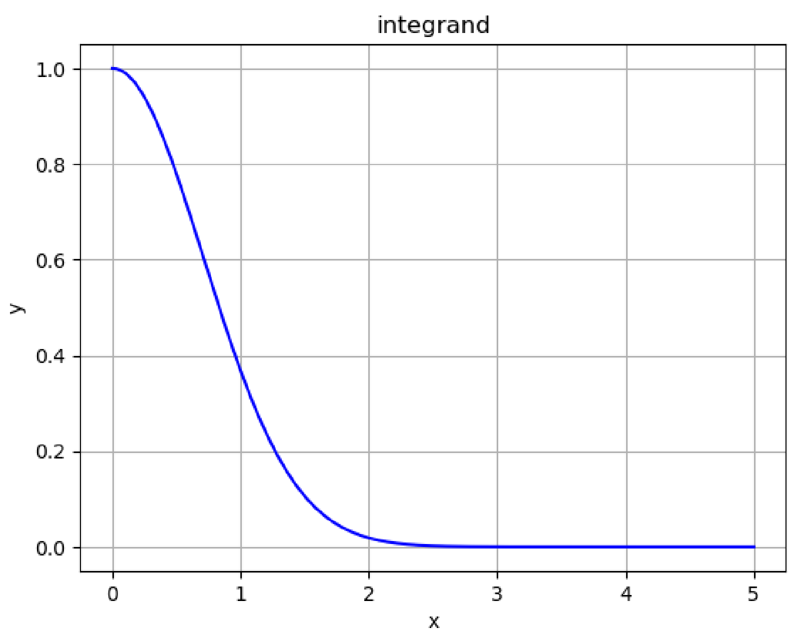


图 1: 待积函数图像

2.2 期望法的结果

代码如下:

```
b=500
n=20000
x=np.random.uniform(0,b,size=n)
x_0=np.random.uniform(0,b)
y_0=np.exp(-x_0**2)
y=np.exp(-x**2)
res=2*b/n*np.sum(y)
print(res)
```

注意到 x 较大时，y 非常接近 0. 那么不妨先让 b, n 取不同的值, 观察一下得出的结果。如表 1。注意到误差并不随试验次数 n 增加，如图 2。

表 1: 期望法-不同 n,b 的结果及误差，保持 $\frac{b}{n}$ 相同

b	5	50	500	5000
n	200	2000	$2*10^4$	$2*10^5$
result	1.5161609488	1.8117818826	2.1732717661	2.1717538931
误差	0.256292902	0.039328032	0.400817915	0.399300042

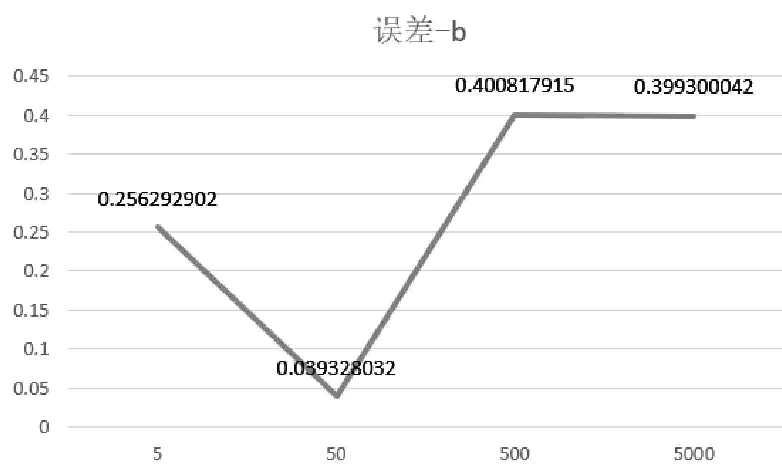


图 2: 误差-b 的关系，保持 $\frac{b}{n}$ 相同

那么分别固定 b 与 n, 再次进行蒙特卡洛

b=50 的结果如表 2。误差可视化为图 3。

可以发现 n 越大时，误差越小。这说明在固定区间长度 b 时，增加试验次数 n 可以增大最终结果的精度。

将区间长度 b 改为 500, 再次实验, 结果如表 3及图 4.

表 2: b=50, 不同 n 的结果及误差

n	2000	20000	$2*10^5$	$2*10^6$	$2*10^7$
result	1.8117818	1.7081665	1.7956457	1.7664646	1.7735477
误差	0.039328032	0.064287307	0.023191946	0.00598917	0.001093887

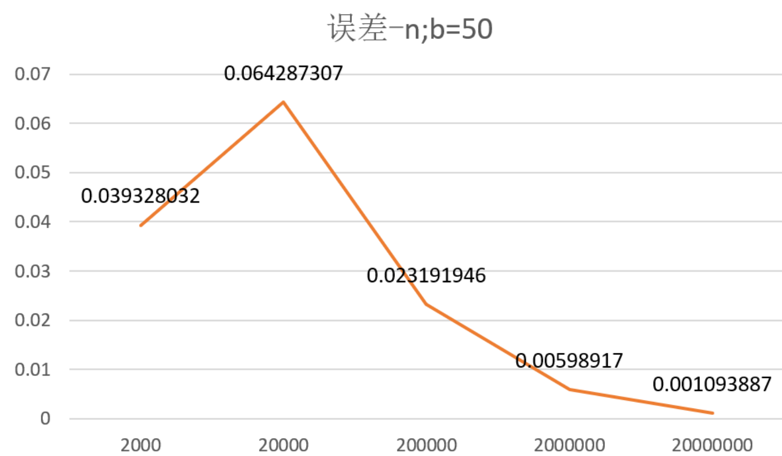


图 3: b=50; 误差-n

表 3: b=500, 不同 n 的结果及误差

n	$2*10^4$	$2*10^5$	$2*10^6$	$2*10^7$	$2*10^8$
result	1.879006581	1.691043252	1.758966654	1.773908719	1.77369733
误差	0.10655273	0.081410599	0.013487197	0.001454868	0.001243479

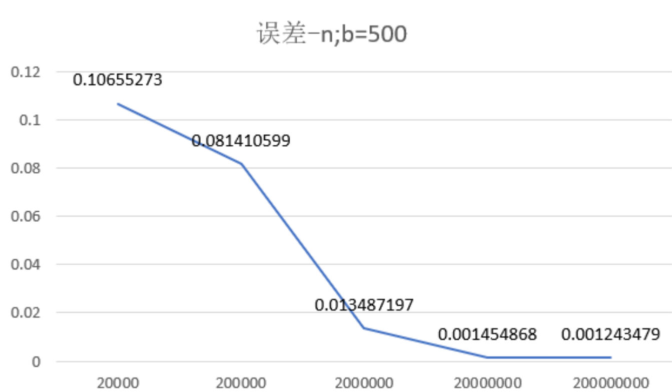


图 4: b=500, 误差-n

对比结果可以看出，当试验次数 n 相同时，增大区间长度 b 不能使误差减小，反而还会增大。这与图 2的结果一致："随机数密度" $\frac{b}{n}$，即单位长度上的试验次数保持不变时，增加试验次数或延长区间长度不会带来精度提升。并且，e^{-x^2} 在 x 取 500 时仅为 10^{-1085} 量级，此时区间 $[500, +\infty]$ 对定积分的贡献已经相当小，再增大 b 的边际收益已经非常有限。

2.3 期望法小结

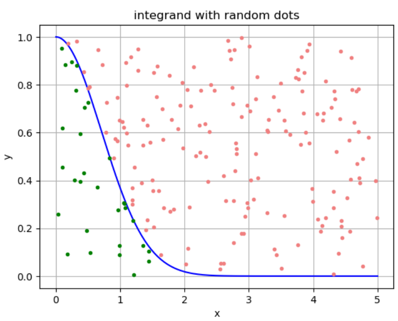
用蒙特卡洛在待积分的区间上随机生成均匀分布的 x_i, 当生成足够多数量 n 时, 就可以用它们的函数值估计期望值, 进而求得定积分的近似值。对于积分上下限为 ∞ 的定积分, 可以用足够大的实数 b 代替 ∞ 计算定积分的值。参数 $b, n, \frac{b}{n}$ 都会影响结果的精度。$\frac{b}{n}$ 越小, 即 b 不变，n 越大，随机数越"精细"，结果精度越高；同时增大 b 和 n，可以提高结果精度，但效应不及前者。

2.4 投点法结果

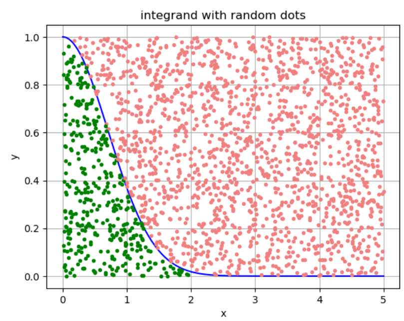
为方便起见，投点法取一个从 [0,0] 到 [b,1] 的矩形。投点法的代码如下:

```
    x=np.linspace(0,5,200)
    y=np.exp(-x**2)
    b=5#区间长度
    n=200#试验次数
    x1=np.random.uniform(0,b,size=n)
    y1=np.random.uniform(0,1,size=n)#随机投点
    def flag(x0,y0):
        return (y0<=np.exp(-x0**2))#判定点是否在曲线下
    flag=np.vectorize(flag)
    f=flag(x1,y1)
    def color(flag):
        if flag:
            return "green"
      return "lightcoral"#着色
    fig,ax=plt.subplots()
    ax.plot(x,y,color="blue",label="y(x)")
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('integrand with random dots')
    ax.grid('True')#绘制待积函数
    for i in range(n):
      ax.plot(x1[i],y1[i],".",color=color(f[i]))#绘制点
    n1=np.sum(f)
    result=2*n1/n*b
    print(result)
```

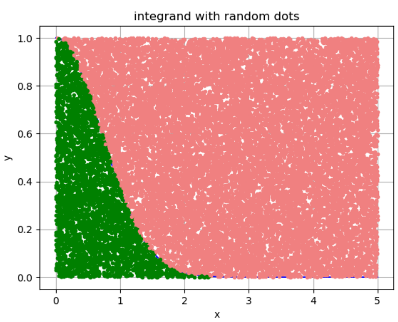
不同 n,b 取值的试验结果可视化如图 5, 图 6所示。与期望法类似，把不同 n,b 取值的结果汇总如图 7。



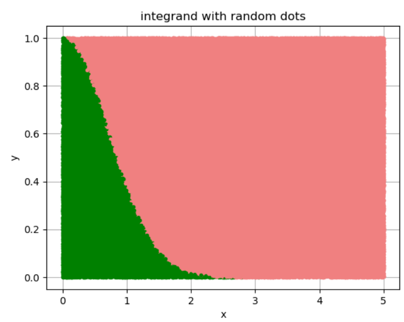
(a) b=5,n=200



(b) b=5,n=2k

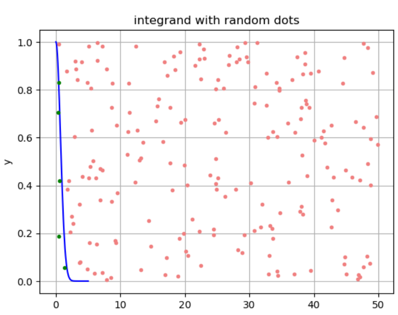


(c) b=5,n=20k

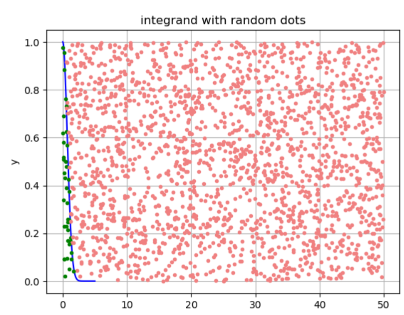


(d) b=5,n=200k

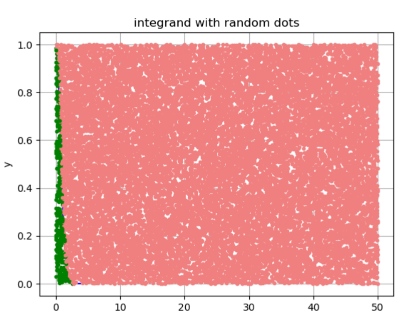
图 5: b=5, 不同 n



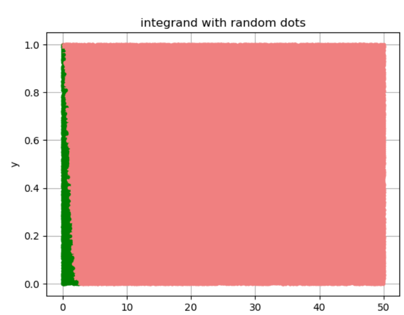
(a) b=50,n=200



(b) b=50,n=2k

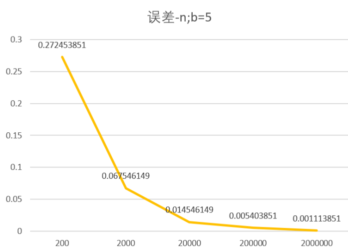


(c) b=50,n=20k

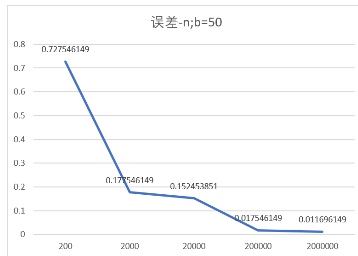


(d) b=50,n=200k

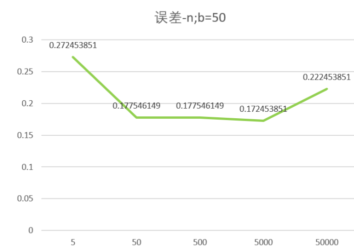
图 6: b=50, 不同 n



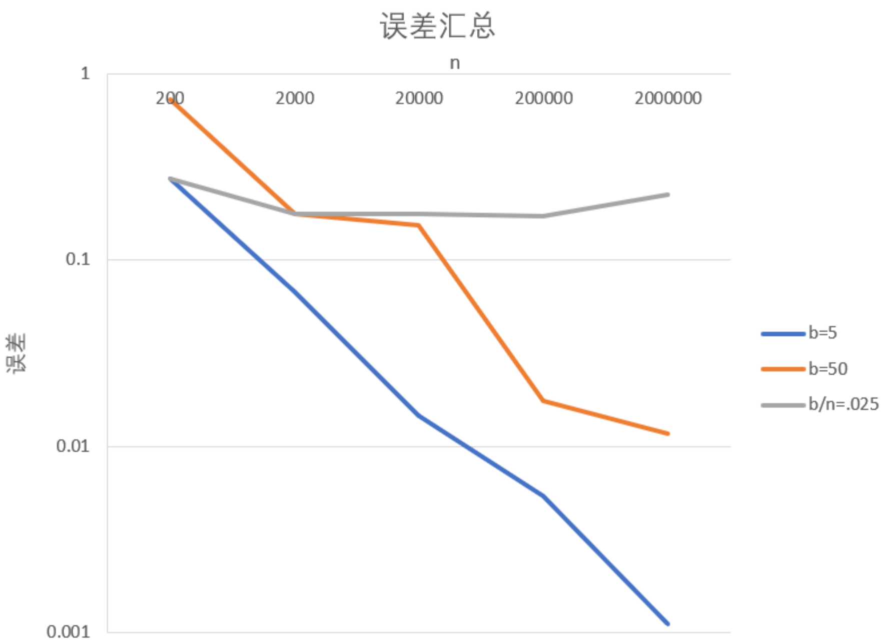
(a) b=5 时误差



(b) b=50 时误差



(c) $\frac{b}{n} = 0.025$ 时误差



(d) 结果汇总 (为便于观察采取对数坐标)

图 7: 误差结果汇总

表 4: b=5/50, 不同 n 的结果及误差

n	200	2000	20000	$2 * 10^5$	$2 * 10^6$
result(b=5)	1.5	1.84	1.787	1.76705	1.77134
误差	0.272453851	0.067546149	0.014546149	0.005403851	0.001113851
result(b=50)	2.5	1.95	1.62	1.79	1.78415
误差	0.727546149	0.177546149	0.152453851	0.017546149	0.011696149

表 5: 投点法保持 $\frac{b}{n}$ 相同的结果及误差

b	5	50	500	5000	50000
n	200	2000	20000	$2 * 10^5$	$2 * 10^6$
result	1.5	1.95	1.95	1.6	1.55
误差	0.272453851	0.177546149	0.177546149	0.172453851	0.222453851

2.5 投点法小结

从图 7(d)可以看出，与期望法的结论类似，参数 $b, n, \frac{b}{n}$ 均有不同作用。具体而言，$\frac{b}{n}$ 越小，随机数越“密集”, 结果精度越高; 同时增大 b 和 n, 提高精度的效果不显著;n 保持不变提高 b, 反而会降低精度 (不过, 值得注意的是, 此处 n 最小也达到了 5. 如果按照电路中“稳态响应”的标准, 当 x^2 超过 5 时即可认为达到“稳态”, 之后的函数积分对结果的影响已经很小);b 保持不变提高 n，可以提高结果的精度。具体数据见于表 4, 表 5

3 两种方法的比较

取 $b = 50, n \in \{200, 2000, 2*10^4, 2*10^5, 2*10^6\}$, 将误差绘图比较。如图 8所示。为便于观察，已采用对数坐标。

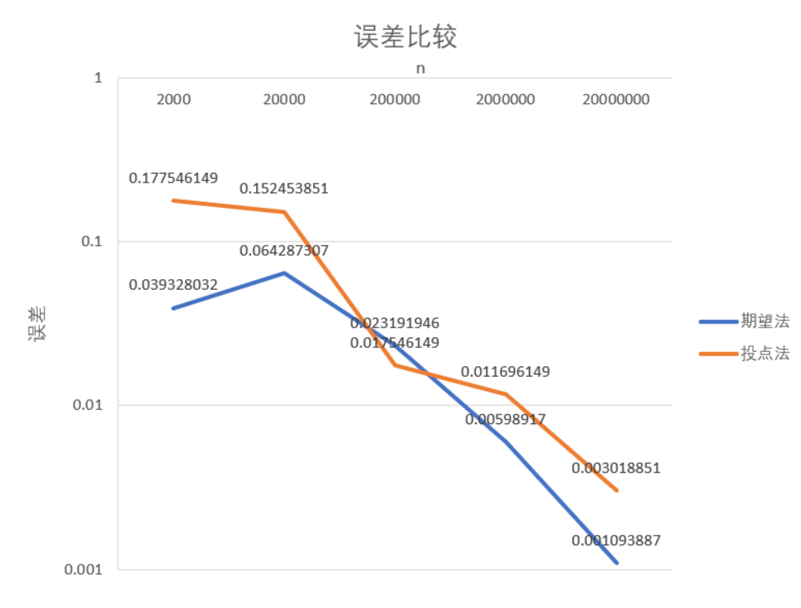


图 8: 期望法与投点法的误差比较

从中似乎可以得出结论: 期望法的误差比投点法小一些。但值得注意的是，蒙特卡洛方法得出的总体精度并不高，即便试验了 2 千万次，绝对误差仍大于 0.001，而同等时间复杂度的矩形法的精度要高的多, 如图 9. 它计算 2 百万次即得出具有 6 位有效数字的结果 1.77245. 因此, 这个结论并非总是成立, 它会受到试验次数 n 的影响. 尤其是 n 较小时. 如图 8前几个数据点。

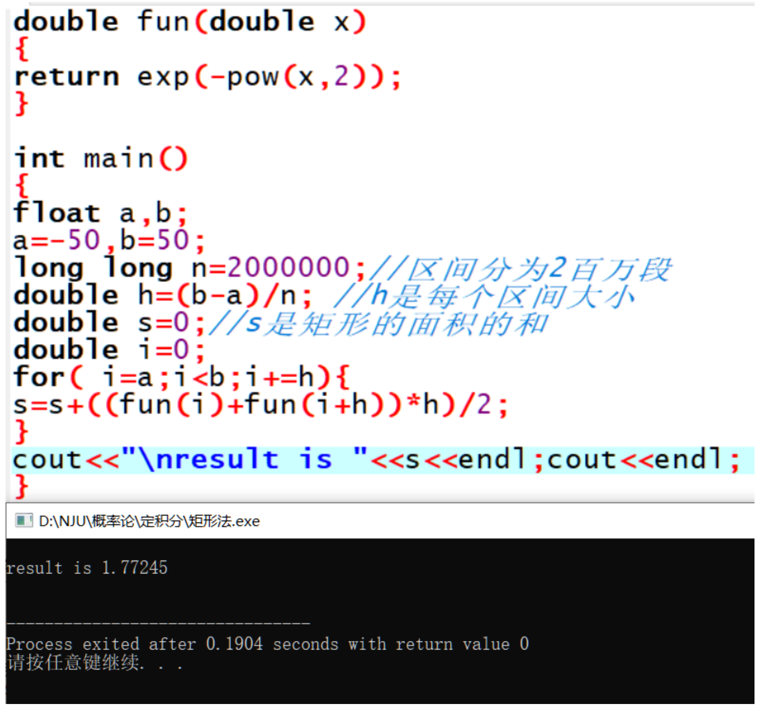


图 9: 矩形法的结果

4 计算其他定积分

4.1 三个定积分实例

待求的定积分一般不是 $(-\infty,+\infty)$, 而是上下限为常数的定积分. 此处将验证 3 个定积分的结果:

$$I_1=\int_{-5}^{5}\frac{sin(x)}{x}dx=2Si(5)\approx 3.09986 \tag{10}$$

$$I_2=\int_{-5}^{5}\frac{sin^2(x)}{x^2}dx\approx 2.94888 \tag{11}$$

$$I_3=\int_{1}^{5}\frac{e^x}{x}dx\approx 38.2902 \tag{12}$$

首先给出两种方法的代码, 仅以 I_1 为例. 它在待积区间内的函数值有正有负, 因此线下面积也有正有负.

```
#期望法
a=-5
b=5
n=2000000
x=np.random.uniform(a,b,size=n)
y=np.sin(x)/x#integrand
res=(b-a)/n*np.sum(y)
print(res)
```

```
#投点法
a=-5
b=5#区间长度
n=2000000#试验次数
maxy=1
miny=-1#矩形区域为(a,miny)到(b,maxy)
x1=np.random.uniform(a,b,size=n)
y1=np.random.uniform(miny,maxy,size=n)#随机投点
def flag(x0,y0):
  if (y0<=np.sin(x0)/x0 and y0>=0):#integrand
    return 1#面积为正
  elif(y0>np.sin(x0)/x0 and y0<0):
    return -1#面积为负
  else:
    return 0
flag=np.vectorize(flag)
f=flag(x1,y1)
n1=np.sum(f)
result=n1/n*(b-a)*(maxy-miny)
print(result)
```

保持 $\frac{b-a}{n}=\frac{1}{400000}$, 这三个积分的结果如表 6所示, 误差如表 7所示. 值得注意的是, 这里取相对误差代替绝对误差, 可视为一种归一化处理.

表 6: 蒙特卡洛结果

待积函数	I_1	I_2	I_3
期望法	3.098587	2.94715	38.307207
投点法	3.098845	2.94189	38.272637

表 7: 蒙特卡洛的相对误差

待积函数	I_1	I_2	I_3
期望法	0.00041066	0.00058652	0.00044416
投点法	0.00032743	0.00237039	0.00045868

4.2 实例小结

蒙特卡洛方法确实可以计算定积分, 但它的精度并不高. 它的相对误差的数量级为千分之一, 这表明在大多数情况下, 结果具有三位有效数字; 但也必须注意到, 它的误差并不”稳定”, 即方差较高, 例如表 5中 I_2 的投点法, 相对误差超过了 2‰, 而其他所有相对误差都在 4‱ 左右.

值得注意的是，在上一篇《基于蒲丰投针问题计算圆周率并衡量误差》中曾提到: 由于运算的所有变量均以 64bit 浮点数或整数存储，因此事实上有“台阶效应”，尽管这一效应在 32bit

及以下的影响最为显著；同时，试验次数虽然可以很大，但不能达到“无穷多”，因此也会影响结果的精度。这两点是用计算机进行蒙特卡洛的固有缺陷，也是精度不高的原因。

尽管对算法复杂度的讨论并不属于本课程的教学内容，但我认为小结是有必要的。在实验时我也注意到, 投点法的运算时间比期望法长. 它们代码量的差异可以解释这一不同: 投点法生成随机数、代入待机函数运算的次数比期望法多，而期望法仅需各运算一次。因此，试验相同次数，投点法会更慢。同时，由于投点法需要额外存储这些点的位置信息 (它所在的这块面积对积分的贡献是 1,-1, 还是 0?)，因此空间复杂度也更高。值得注意的是，期望法在使用中更加便利，因为投点法需要设定投点的区域即 ymin 与 ymax，这两个参数与函数在区间上的最大值、最小值有关，因此对于更加“陌生”的函数，使用起来不如期望法方便

5 总结

蒙特卡洛方法计算定积分可按照积分上下限是否为 ∞ 分为两类: 上下限为常数的定积分与上下限为 ∞ 的定积分；实现方法有期望法和投点法两种。对于前一类，需要用一个足够大的数 b 替代 ∞ 进行计算，影响结果精度的参数有 b, 试验次数 n,$\frac{b}{n}$。$\frac{b}{n}$ 越小，随机点越“密集”，结果精度越高：b 足够大时，同步增大 n 可以提高结果精度，但效应不如固定 b、提高 n 显著。对于后一类，提高 n 有助于提高结果的精度。

从函数本身出发，当 b 取到一定有限值，如 $e^{-5^2} \approx 1.4 * 10^{-11}$，函数随后快速趋于 0，积分值的贡献远远低于蒙特卡洛本身的误差，此时完全应该提高 n 以获得更高的精度。但 b 究竟该取多大又依赖于人对这个函数的认知与判断，因此研究 b 取值的自动化方法（如基于统计特征等），在实际应用中更有意义。因为超出课程内容，不做展开。

总体而言，期望法的精度比投点法高，尤其是当 n 非常大时。并且，期望法有复杂度更低、使用更方便（人工设定的参数更少，不必预先知道待机函数的最值信息）的优点。但它们的精度均不及同等复杂度的传统积分方法——矩形法。这也是矩形法在实际中被广泛用于计算定积分的原因.

References

[1] 知乎-浮光掠影之蒙特卡洛积分 https://zhuanlan.zhihu.com/p/586493653

[2] 李嘉伟-基于蒲丰投针问题计算圆周率并衡量误差