数据挖掘导论

第一次实验报告

——蒙特卡洛方法

学号：16340192

姓名：宋晓彤

方向：嵌入式软件与系统

目录

一、 实验一：落点法求 pi 近似值

1. 实验目的

使用 python 语言通过蒙特卡洛方法中的落点法求出 pi 值的近似值。

2. 实验原理

如下图可见，正方形的面积是 1，该 1/4 圆的面积为 pi/4。通过编程实现在这个正方形中产生均匀分布的点。落在圈内的点和总的投在正方形上的点的比率就是 pi/4 的近似值。

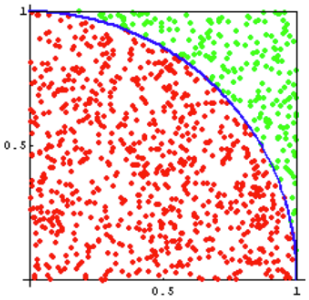

图 1. 蒙特卡罗求 pi 值实验方法示意图

3. 实现过程

首先，设定不同的落点数和重复试验次数，根据实验目标，我们设置为如下的数值，即落点数 N 分别设置为 20，50，100，200，300，500，1000，5000，每次修改落点数要重复 20 次试验

将上图定义为横纵坐标范围均为[0,1]的坐标平面，在每次试验中，产生 N 个 x 坐标和 y 坐标都随机分布在[0,1]之间的点，然后根据该点距离坐标原点（0,0）的距离是否大于 1 判断该点是否在该 1/4 圆内（<1 则在圆内，否则在圆外），统计落在 1/4 圆内的点数 inside，此时，通过计算可以得到 pi 的近似值

针对不同的落点数重复 20 次试验，记录每次实验的结果，然后得出其平均值与方差。（为了便于查看，在命令行中输出均值与方差的结果，将每次试验的统计数值存放在 excel 文件中，具体结果见附录）

4. 实验结果

在命令行中输出方差和均值，整理得到表 1

落点总数 N	均值μ	方差δ
20	3.11	9.1419
50	3.164	0.063664
100	3.132	0.020296

200	3.101000	0.009619
300	3.161333	0.004549
500	3.132	0.003670
1000	3.1614	0.001768
5000	3.14468	0.000424

表 1. 蒙特卡罗求 pi 值实验结果记录

具体每次试验产生的圆内落点数、圆外落点数、pi 值存放在 excel 文件中，示例见图 2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	落点总数	圆内落点	圆外落点	pi		落点总数	圆内落点	圆外落点	pi		落点总数	圆内落点	圆外落点	pi		落点总数	圆内落点	圆外落点	pi
2	20	15	5	3		50	37	13	2.96		100	78	22	3.12		200	154	46	3.08
3	20	13	7	2.6		50	38	12	3.04		100	79	21	3.16		200	157	43	3.14
4	20	15	5	3		50	34	16	2.72		100	72	28	2.88		200	160	40	3.2
5	20	12	8	2.4		50	38	12	3.04		100	79	21	3.16		200	153	47	3.06
6	20	18	2	3.6		50	42	8	3.36		100	80	20	3.2		200	153	47	3.06
7	20	13	7	2.6		50	39	11	3.12		100	81	19	3.24		200	156	44	3.12
8	20	19	1	3.8		50	41	9	3.28		100	78	22	3.12		200	155	45	3.1
9	20	16	4	3.2		50	41	9	3.28		100	83	17	3.32		200	151	49	3.02
10	20	18	2	3.6		50	43	7	3.44		100	70	30	2.8		200	166	34	3.32
11	20	17	3	3.4		50	39	11	3.12		100	70	30	2.8		200	150	50	3
12	20	15	5	3		50	44	6	3.52		100	80	20	3.2		200	145	55	2.9
13	20	14	6	2.8		50	34	16	2.72		100	84	16	3.36		200	159	41	3.18
14	20	14	6	2.8		50	38	12	3.04		100	73	27	2.92		200	155	45	3.1
15	20	15	5	3		50	36	14	2.88		100	82	18	3.28		200	164	36	3.28
16	20	15	5	3		50	43	7	3.44		100	79	21	3.16		200	159	41	3.18
17	20	18	2	3.6		50	41	9	3.28		100	80	20	3.2		200	149	51	2.98
18	20	14	6	2.8		50	35	15	2.8		100	79	21	3.16		200	152	48	3.04
19	20	17	3	3.4		50	41	9	3.28		100	73	27	2.92		200	153	47	3.06
20	20	16	4	3.2		50	43	7	3.44		100	84	16	3.36		200	152	48	3.04
21	20	17	3	3.4		50	44	6	3.52		100	82	18	3.28		200	158	42	3.16
22																			

图 2. 蒙特卡罗求 pi 值实验过程部分记录

二、 实验二：蒙特卡洛方法求简单积分

1. 实验目的

通过 python 语言使用蒙特卡洛方法求出$\int_0^1 x^3$的积分结果并进行分析，判断采样点的分布方法，评价积分结果的准确度。

2. 实验原理

当对形式是次方的基本函数求积分时，可以很简单地画出其函数轨迹，同时得到其在目标范围内的极值，也就是在 x=1 处取最大值为 1,x=0 处取最小值为 0，此时，可以类比上一道题通过落点法求出其积分。

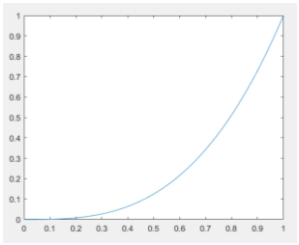

图 3. x^3 函数图像

除此之外，我们可以引入平均值方法，用均匀分布的方法对 x 坐标进行采样，对得到的相应的函数值求期望，再乘以 x 坐标轴的范围求出期望

的面积也就是期望的积分结果。

3. 实现过程

首先，设定不同的采样数和重复试验次数，根据实验目标，我们设置为如下的数值，即采样数 N 分别设置为 5，10，20，30，40，50，60，70，80，100，每次修改采样数要重复 100 次试验。

针对落点法，也就是已知函数图象形状时，具体方法同实验一，在[0,1]的横纵坐标范围内随机取点，判断此时的 y1 是否小于 x^3，此时积分结果=位于曲线下方的点的个数/总点数*1。

针对平均值法，对采样点的横坐标进行相应数量 N 均匀分布的采样，将得到的 x 代入函数求出对应纵坐标 y2，计算 y2 均值，乘上横坐标范围也就是 1，从而得到面积，也就是积分结果。

将两个方法得到的 s 分别存入 s_list，在重复 100 次试验后求出均值和方差，并用命令行输出

4. 实验结果

在命令行中输出方差和均值，将落点法的均值方差记为μ1、δ1，平均值法的均值方差记为μ2、δ2，整理下表

落点总数 N	均值μ1	方差δ1	均值μ2	方差δ2
5	0.231444	0.015387	0.238000	0.038156
10	0.258542	0.008632	0.259	0.020019
20	0.246642	0.003311	0.233499	0.009103
30	0.251701	0.002768	0.253667	0.005064
40	0.256482	0.002350	0.254250	0.004426
50	0.252423	0.001328	0.253399	0.003040
60	0.244805	0.001242	0.233333	0.002456
70	0.257572	0.001050	0.252857	0.001945
80	0.249075	0.000890	0.240125	0.001832
100	0.251764	0.000886	0.250700	0.001653

表 2. 蒙特卡罗落点法及平均值法求 x^3 积分实验结果记录

具体每次试验产生的 x 序列均值、y 序列均值、落点法结果 s1，平均值法结果 s2 存放在 excel 文件中，部分截图如下

序号	采样数	采样点x均值	采样点y均	积分结果1	积分结果2		序号	采样数	采样点x均	采样点y均	积分结果1	积分结果2
0	5	0.48082258	0.504909	0.1879101	0		0	10	0.616463	0.274478	0.436612	0.6
1	5	0.45575726	0.605684	0.1906294	0		1	10	0.350656	0.443284	0.087167	0.2
2	5	0.48799075	0.460082	0.1888558	0.2		2	10	0.454446	0.367907	0.208985	0.2
3	5	0.50821257	0.480771	0.2247599	0.2		3	10	0.64252	0.511408	0.367524	0.2
3	5	0.38149471	0.610155	0.2235888	0.2		4	10	0.489282	0.476791	0.199586	0.4
3	5	0.29141508	0.293699	0.1315976	0.2		5	10	0.440365	0.404397	0.176991	0.2
3	5	0.40862906	0.445572	0.0831446	0.2		6	10	0.664037	0.421978	0.407695	0.5
3	5	0.52085292	0.662776	0.3833779	0.4		7	10	0.484267	0.354008	0.235561	0.4
3	5	0.18771414	0.595913	0.0325849	0		8	10	0.495794	0.4984	0.274715	0.2
3	5	0.53005294	0.353596	0.2602074	0.4		9	10	0.489068	0.630106	0.183321	0
3	5	0.73961629	0.467279	0.4491171	0.4		10	10	0.520059	0.529182	0.233212	0.2
3	5	0.39596134	0.550062	0.1150888	0		11	10	0.513306	0.280856	0.211668	0.4
3	5	0.49490745	0.537084	0.2633074	0		12	10	0.527811	0.591516	0.231751	0.2
3	5	0.27396511	0.676247	0.0775875	0		13	10	0.437168	0.398376	0.234898	0.2
3	5	0.44914427	0.487735	0.1436022	0.2		14	10	0.710947	0.463919	0.465522	0.5

图 4. 落点法及平均值法求 x^3 积分实验过程部分记录

5. 实验结论

（1）在本次实验中，对采样点的选取使用的是**均匀分布**的方法。

（2）从最终的均值结果来看，实验设计和实验结果都是**比较合理**的，不同采样点数的实验结果大体在理论值 1/4 附近浮动，且浮动范围不大。当采样点数越多时，方差越小，也就是最终的结果越来越稳定，理论上估计可以在样本数量足够大的时候得到 1/4 的理论值。

三、 实验三：平均值法求二重积分

1. 实验目的

$$\int_{x=2}^{4}\int_{y=-1}^{1} f(x,y) = \frac{y^2 * e^{-y^2} + x^4 * e^{-x^2}}{x * e^{-x^2}}$$

（1） 判断是否能通过公式直接求出上述函数的积分结果

（2） 通过 python 语言使用蒙特卡洛方法求出上述函数的积分结果并进行分析。判断采样的分布、评价积分结果的准确度。

2. 实验原理

（1）公式求解难度很大。

（2）平均值法：将选取范围定为 x∈[2,4]，y∈[-1，1]，先在其中得到**均匀分布**的样本点（算法代码见附录），然后代入函数得到函数值，将多次试验得到的函数值取平均后就可以认为这是当前函数的理论期望，将其乘以面积 4，就可以得到体积也就是最终的积分结果

3. 实现过程

首先，设定不同的采样数和重复试验次数，根据实验目标，我们设置为如下的数值，即采样数 N 分别设置为 10，20，30，40，50，60，70，80，100，200,500 每次修改采样数要重复 100 次试验。

然后跟实验二的平均值法同理，但是并不适用直接产生随机数的方法得到采样点，而是使用自己编写的 points 函数生成均匀分布的点。使用直线向量形式的理论，得出在三角形中得到均匀分布的点的方法，然后将当前的范围划分为两个三角形，分别均匀采样得到采样点。

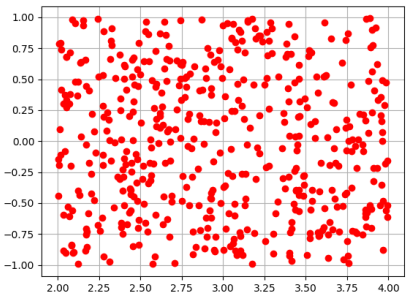

图 5. N=200 时采样点分布

将其 x，y 数值代入函数得到一系列 z 值，将 z 值取平均，乘以底面积 2*2=4，得到的结果就是积分结果

对生成点的查看我们可以使用 matplotlib 模块生成，代码部分未展示，生成图见附录

4. 实验结果

落点总数 N	均值μ	方差δ
10	114449.5713	10956781991.7
20	112667.7853	5398996748.1
30	112415.2937	4396699576.4
40	120708.5847	3387066245.7
50	120096.8235	2584450309.4
60	113541.1594	2119501404.9
70	119058.2010	2141038441.3
80	119004.8501	1431552251.0
100	111644.8087	1137657077.3
200	114608.9487	478795866.1
500	114859.5597	267906511.5

表 3. 蒙特卡罗平均值法求二重积分实验结果记录

具体每次试验产生的 x 序列均值、y 序列均值、积分结果存放在 excel 文件中，部分截图如下

10	3.172447	-0.05134	78176.15	20	3.045802	-0.06401	474188.4	30	3.168573	-0.05005	226994.8	40	3.063345	-0.03003	71697.94
10	3.002875	0.059591	49096.83	20	3.00718	0.038113	35187.21	30	2.872346	0.100114	40982.87	40	3.072406	-0.03794	113789.6
10	3.109899	0.040376	332690.5	20	3.006451	-0.00537	197286.2	30	2.919333	0.074139	10850.63	40	3.087042	0.084448	108556.7
10	2.938396	-0.14169	16441.57	20	3.022475	-0.14822	47607.26	30	3.013969	0.039577	56154.03	40	2.992392	0.052604	92689.98
10	3.06384	-0.18065	9018.754	20	2.856626	-0.00645	25809.02	30	2.940805	0.074893	76255.72	40	3.112578	-0.17578	180132.2
10	2.993136	0.179617	105931.6	20	3.079119	0.078652	71422.34	30	2.901243	0.055668	162805	40	2.849124	0.000633	71845.8
10	3.042862	-0.32865	254615.5	20	3.176686	-0.08108	156575	30	3.063478	-0.06611	198169.6	40	3.058154	-0.03587	88697.24
10	2.969231	-0.14099	1968.695	20	3.036861	0.07028	118595.6	30	2.983115	-0.02533	57319.74	40	2.947553	-0.05073	117948.7
10	2.980197	0.092503	22802.94	20	3.08629	-0.08122	161543.4	30	3.09523	-0.05496	194587.5	40	2.973255	0.090843	62122.67
10	2.965758	0.077678	22733.4	20	2.913202	-0.06627	74633.42	30	2.946487	0.071477	30568.32	40	2.943536	0.043888	31798.42
10	3.104562	-0.1249	172663.6	20	2.916147	0.194466	43971.07	30	3.00344	0.034172	120260.7	40	3.066471	-0.05202	66699.57
10	2.96513	-0.05592	14318.54	20	2.903886	0.137331	33185.72	30	2.991436	-0.04305	150245	40	3.112058	-0.06521	106250.2
10	2.864209	-0.02128	9605.407	20	3.016676	0.115155	55292.21	30	3.035071	0.101847	91369.18	40	2.973992	-0.00143	83144.21
10	3.093123	0.062918	647629	20	2.867427	-0.05729	17072.64	30	3.001311	0.023461	102063.4	40	2.984951	-0.00163	87363.64
10	3.186522	-0.06762	358309.1	20	2.952616	-0.03761	21495.43	30	3.058967	-0.07765	129100	40	2.967857	-0.00021	84133.52
10	2.959686	-0.21805	3685.252	20	2.946001	0.02616	59237.53	30	3.10097	0.034382	81138.37	40	2.966982	0.060287	109310.8
10	2.971467	-0.07677	11717.06	20	2.832417	0.041893	45228.91	30	2.976909	0.025639	66122.71	40	2.886833	0.072652	52875.71
10	2.997062	0.080338	73293.49	20	3.078026	-0.17506	85097.91	30	2.958766	0.111914	140111.8	40	2.888467	0.11859	75772.49
10	2.8116	0.283777	17063.63	20	3.045466	0.10001	77957.35	30	3.051578	-0.01808	133077.5	40	2.974316	0.116433	109167.1
10	2.969562	0.166678	216830.9	20	3.011801	0.053301	45293.37	30	3.080991	-0.02435	238601.6	40	2.957411	-0.07401	86343.26
10	3.059524	-0.07789	61274.14	20	2.835573	0.108613	112495.8	30	2.891094	0.043705	70088.47	40	3.040736	-0.04974	50064.75
10	2.983264	0.046903	30323.57	20	2.9911	0.009015	15391.55	30	2.946935	-0.0299	166078.9	40	3.069005	-0.05954	172058.6
10	3.093225	0.0364	449136.3	20	2.93555	0.207054	161610.4	30	2.961077	-0.1147	170016.8	40	2.958459	-0.10176	235760.5
10	2.90196	0.125349	62654.91	20	2.986034	0.020138	54058.63	30	2.9355	-0.00565	92218.45	40	2.996257	-0.03765	84197.08

5. 实验结论

（1）用公式法计算积分是很难的。

（2）对采样点的选取使用的是**均匀分布**的方法。

（3）从最终的均值结果来看，实验结果集中在 11 万-12 万的范围内，但是由于目标理论值很大，所以每次实验结果上下浮动的范围也以万计数，从而导致每次实验得到的方差十分巨大。也就是说，就当前的采样点数设计来看，积分结果并不是十分理想。

四、 附录

1. 实验一源码

```
# 省略 import
rb = open_workbook('res.xls')
wb = copy(rb)
sheet = wb.get_sheet(0);

time = 20;
N = [20, 50, 100, 200, 300, 500, 1000, 5000];
l = -1;

for num in N:
    pi_list = [];
    l += 1;
    for i in range(time):
        inside = 0;
        for j in range(num):
            x = random.random();
            y = random.random();
            dis = math.sqrt(x**2 + y**2);
            if dis < 1:
                inside += 1;
        pi = 4*inside/num;
        sheet.write(i+1, l*5+0, num);
        sheet.write(i+1, l*5+1, inside);
        sheet.write(i+1, l*5+2, num-inside);
        sheet.write(i+1, l*5+3, round(pi, 6));
        wb.save('res.xls')
        # print('第'+str(i+1)+'次试验: ', inside, outside, round(pi, 3));
        pi_list.append(pi);
```

```
    pi_mean = np.mean(pi_list);
    pi_var = np.var(pi_list);
    print('均值为', pi_mean, '方差为', pi_var);
```

2. 实验二源码

```
# 省略部分内容
for num in N:
    s_list1 = [];
    s_list2 = [];
    l += 1;
    for i in range(time):
        cur_row = 1;
        inside = 0;
        x = np.random.uniform(0, 1, num);
        y1 = np.random.uniform(0, 1, num);
        y2 = x*x*x;
        s1 = np.mean(y2)*1;
        s_list1.append(s1);
        for j in range(len(y1)):
            if y1[j]<y2[j]:
                inside += 1;
        s2 = inside / num * 1;
        s_list2.append(s2);
    s_mean = np.mean(s_list1);
    s_var = np.var(s_list1);
    s_mean = np.mean(s_list2);
    s_var = np.var(s_list2);
```

3. 实验三源码

```
# 选取采样点函数
def points(x1, x2, y1, y2, sample_size):
    half = math.floor(sample_size/2)
    rnd1 = np.random.random(size = half)
    rnd2 = np.random.random(size = half)
    rnd2 = np.sqrt(rnd2)
    x = (rnd2 * (rnd1 * x1 + (1 - rnd1) * x1) + (1 - rnd2) *
x2).tolist()
    y = (rnd2 * (rnd1 * y1 + (1 - rnd1) * y2) + (1 - rnd2) *
y1).tolist()
   # plt.plot(x,y,'ro')
    rnd1 = np.random.random(size = sample_size - half)
    rnd2 = np.random.random(size = sample_size - half)
    rnd2 = np.sqrt(rnd2)
```

```
    x1 = ((rnd2 * (rnd1 * x1 + (1 - rnd1) * x2) + (1 - rnd2) *
x2)).tolist()
    y1 = ((rnd2 * (rnd1 * y2 + (1 - rnd1) * y1) + (1 - rnd2) *
y2)).tolist()
    x = x+x1
    y = y+y1
    return x, y

# 省略部分代码
for n in N:
    l += 1
    v_list = []
    for j in range(times):
        x, y = points(2, 4, -1, 1, n)
        if i < 2:
            image(x, y, n, i)
        z_list = []
        for i in range(len(x)):
            z = (y[i]*y[i]*math.exp(-y[i]*y[i]) + math.pow(x[i],
4)*math.exp(-x[i]*x[i]))/(x[i]*math.exp(-x[i]*x[i]))
            z_list.append(z)
        v_list.append(np.mean(z_list)*4)
    print(n, np.mean(v_list), np.var(v_list))
```

4. 实验三采样点绘制结果示例