

# 计算物理 A 第九题

杨旭鹏 PB17000234

2019 年秋季

## 1 题目描述

自设若干个随机分布（相同或不同分布，它们有相同或不同的  $\mu$  和  $\sigma$ ）通过 Monte Carlo 模拟，验证中心极限定理成立 ( $N = 2, 5, 10$ )。

## 2 算法

### 2.1 中心极限定理

设随机变量  $x$  满足概率密度分布函数  $p(x)$ ，则对于任意  $p(x)$  都有：

$$\left\{ \begin{array}{l} p\left(\left|\frac{\langle x \rangle - \mu}{\sigma/\sqrt{N}}\right| < \beta\right) \rightarrow \Phi(\beta) \\ \langle x \rangle = \frac{1}{N} \sum_{i=1}^N x_i \\ \mu = \int_{-\infty}^{\infty} xp(x)dx \\ \sigma = \sqrt{E(x^2) - (E(x))^2} = \sqrt{\int_{-\infty}^{\infty} x^2 p(x)dx - \left(\int_{-\infty}^{\infty} xp(x)dx\right)^2} \end{array} \right. \quad (1)$$

其中  $\mu$  为随机变量  $x$  的期望值， $\sigma$  为  $x$  的标准差。则若想验证此条定理，可对满足某一分布的随机变量  $x$  进行  $N$  次抽样，计算统计量  $\frac{\langle x \rangle - \mu}{\sigma/\sqrt{N}}$ ，重复上述过程  $n$  次，则可得到对应抽样  $N$  次对应的统计量的近似概率分布 ( $n$  越大时此近似分布约接近该统计量的理论分布)。将得到的统计量的近似概率分布与标准正态分布比对，若当  $n$  越大时，两者越接近，即可验证此条定理。具体实现见代码。

## 2.2 分布 1

设随机变量  $x$  满足概率分布：

$$p(x) = \begin{cases} x + 1 & , if x \in [-1, 0] \\ -x + 1 & , if x \in [0, 1] \\ 0 & , otherwise \end{cases} \quad (2)$$

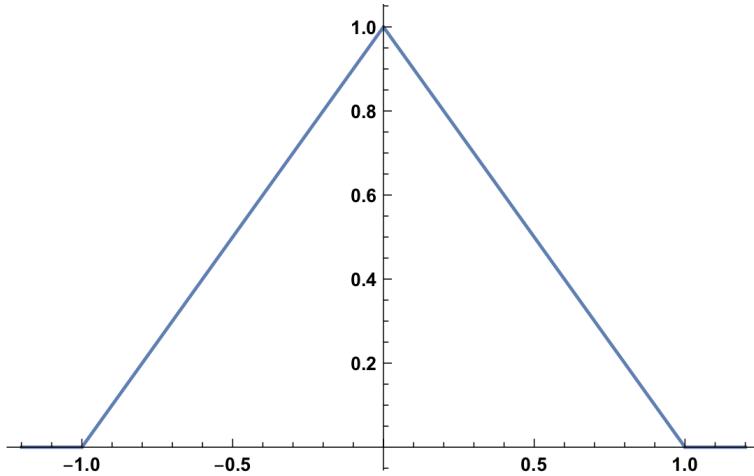


图 1：分布 1 的概率分布函数示意图

则可得到其期望值和标准差为： $\mu = 0, \sigma = \frac{1}{\sqrt{6}}$ ，则按上述方法进行数值模拟即可。每次计算时所采取的抽样方法为直接抽样法：

$$\xi(x) = \begin{cases} \frac{x^2+1}{2} + x & , if x \in [-1, 0] \\ -\frac{x^2-1}{2} + x & , if x \in [0, 1] \end{cases} \quad (3)$$

其中  $\xi(x)$  为累积函数。容易求得其反函数：

$$x(\xi) = \begin{cases} \sqrt{2\xi} - 1 & , if \xi \in [0, 0.5] \\ 1 - \sqrt{2(1-\xi)} & , if \xi \in [0.5, 1] \end{cases} \quad (4)$$

$\xi$  可在  $[0, 1]$  区间内均匀抽取得到，即可进行直接抽样。

## 2.3 分布 2

设随机变量  $x$  满足概率分布：

$$p(x) = \begin{cases} 1 & , if x \in [1, 2] \\ 0 & , otherwise \end{cases} \quad (5)$$

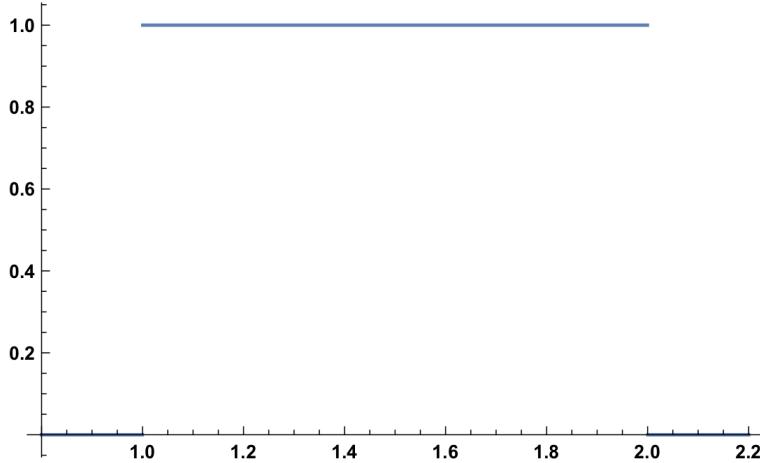


图 2: 分布 2 的概率分布函数示意图

则可得到其期望值和标准差为:  $\mu = 0.5, \sigma = \frac{1}{\sqrt{12}}$ , 则按上述方法进行数值模拟即可。其每次计算时的抽样可利用  $[0, 1]$  区间内均与分布的抽样点向正方向平移 1 得到。

## 2.4 分布 3

设随机变量  $x$  满足概率分布:

$$p(x) = \begin{cases} x + 1 & , if x \in [-1, 0] \\ 0.5 & , if x \in [0, 1] \\ 0 & , otherwise \end{cases} \quad (6)$$

则可得到其期望值和标准差为:  $\mu = \frac{1}{12}, \sigma = \frac{\sqrt{35}}{12}$ , 则按上述方法进行数值模拟即可。每次计算时所采取的抽样方法为直接抽样法:

$$\xi(x) = \begin{cases} \frac{x^2+1}{2} + x & , if x \in [-1, 0] \\ \frac{x+1}{2} & , if x \in [0, 1] \end{cases} \quad (7)$$

其中  $\xi(x)$  为累积函数。容易求得其反函数:

$$x(\xi) = \begin{cases} \sqrt{2\xi} - 1 & , if \xi \in [0, 0.5] \\ 2\xi - 1 & , if \xi \in [0.5, 1] \end{cases} \quad (8)$$

其中  $\xi$  可在  $[0, 1]$  区间内均匀抽取得到, 即可进行直接抽样。

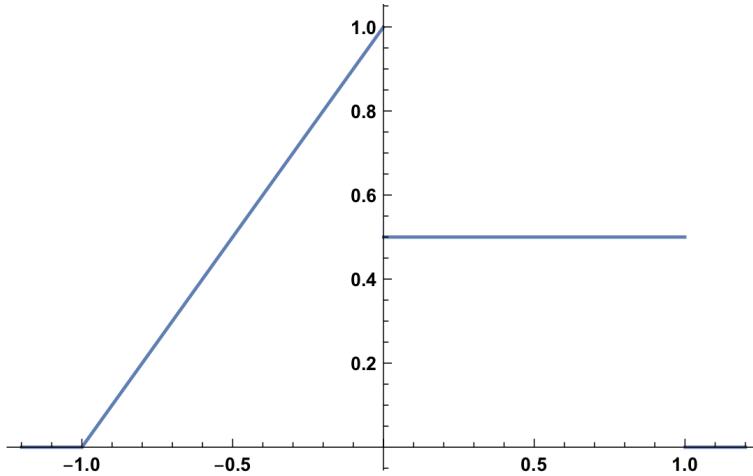


图 3: 分布 3 的概率分布函数示意图

## 2.5 16807 产生器

16807 产生器属于线性同余法产生器的特例。而线性同余法方法为:

$$\begin{aligned} I_{n+1} &= (aI_n + b) \bmod m \\ x_n &= I_n / m \end{aligned} \tag{9}$$

其中整数  $I_i \in [0, m - 1]$ ,  $a, b, m$  为算法中的可调参数, 其选取直接影响产生器的质量。选取参数:

$$\begin{cases} a = 7^5 = 16807 \\ b = 0 \\ m = 2^{31} - 1 = 2147483647 \end{cases} \tag{10}$$

即为所谓的 16807 产生器。由于直接利用 9 编写程序时计算  $(aI_n \bmod m)$  时很容易造成数据溢出, 故采取 Schrage 方法进行具体编程的实现:

$$aI_n \bmod m = \begin{cases} a(I_n \bmod q) - r[I_n/q], & if \geq 0 \\ a(I_n \bmod q) - r[I_n/q] + m, & otherwise \end{cases} \tag{11}$$

其中  $m = aq + r$ , 即  $q = m/a = 127773$ ,  $r = m \bmod a = 2836$ 。即可利用此方法产生伪随机数序列。

### 3 程序使用方法

在运行程序后，会看到请求输入所需计算的统计量个数的提示，按照提示在后面输入所需要的总计算次数，摁回车继续。屏幕请求输入每次计算统计量所用的点数，输入后摁回车继续。然后经过计算统计出 3 个分布在  $[-3, 3]$  上剖分的  $o$  个小区间上计算的统计量出现的概率到数据文件。<sup>1</sup>程序输出完这些后会自动退出。

```
请输入您所需的计算次数: 100000000
请输入您每次计算所用的点数: 10
您输入的参数已接受, 正在计算请稍等片刻~
Program ended with exit code: 0
```

图 4：一个典型程序的运行示例

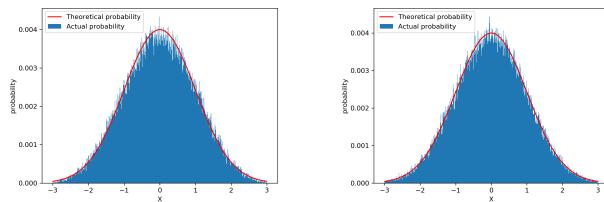
### 4 程序结果与讨论

当输入一些不同的计算次数时，得到如下结果<sup>2</sup>：

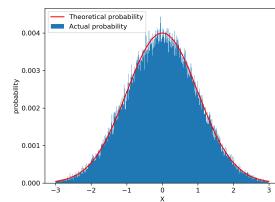
---

<sup>1</sup>  $o$  可在程序宏定义中更改，默认值为 600，即每个小区间长度为 0.01

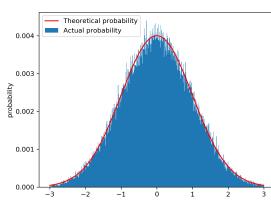
<sup>2</sup> 以下的结果是将  $[-3, 3]$  剖分为 600 个小区间得到的（即每个小区间的长度为 0.01）其中概率为在  $[-3, 3]$  之间归一化后的结果，例如对于在  $[x_i, x_{i+1}]$  中出现的概率，对于抽样点来说即为： $p_i = \frac{n_i}{N}$  ( $n_i$  为统计量落在此区间内的个数， $N$  为计算总次数数)；对于理论频率（标准正态分布）来说，即为： $p_i = \frac{\int_{x_i}^{x_{i+1}} p(x) dx}{\int_{-3}^3 p(x) dx} \doteq \frac{p(\frac{x_2+x_i+1}{2}) \Delta x}{\int_{-3}^3 p(x) dx}$  (其中  $\Delta x$  为区间长度,  $p(x)$  为标准正态分布的密度函数)



(a) 每次抽样 2 个点

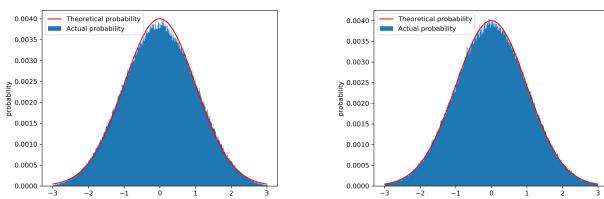


(b) 每次抽样 5 个点

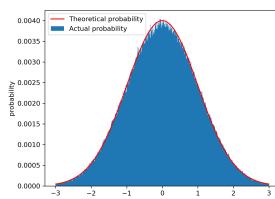


(c) 每次抽样 10 个点

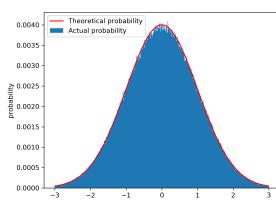
图 5: 1 号分布计算  $10^5$  次后的结果



(a) 每次抽样 2 个点

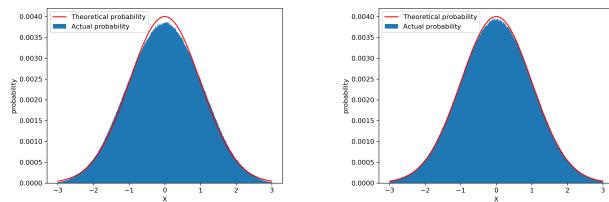


(b) 每次抽样 5 个点

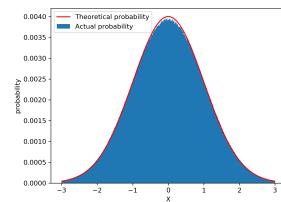


(c) 每次抽样 10 个点

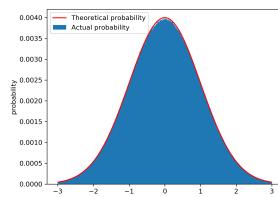
图 6: 1 号分布计算  $10^6$  次后的结果



(a) 每次抽样 2 个点

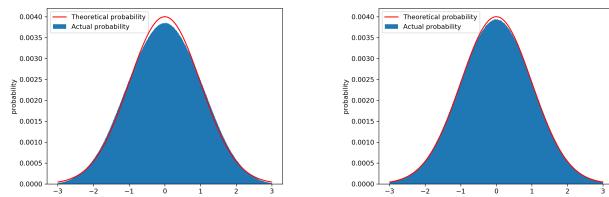


(b) 每次抽样 5 个点

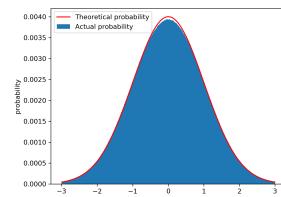


(c) 每次抽样 10 个点

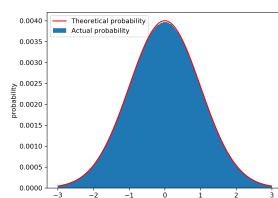
图 7: 1 号分布计算  $10^7$  次后的结果



(a) 每次抽样 2 个点



(b) 每次抽样 5 个点



(c) 每次抽样 10 个点

图 8: 1 号分布计算  $10^8$  次后的结果

可以看出对于一号分布，随着计算次数和每次抽样个数的增加，其统计量的分布与标准正态分布越来越接近。<sup>3</sup>

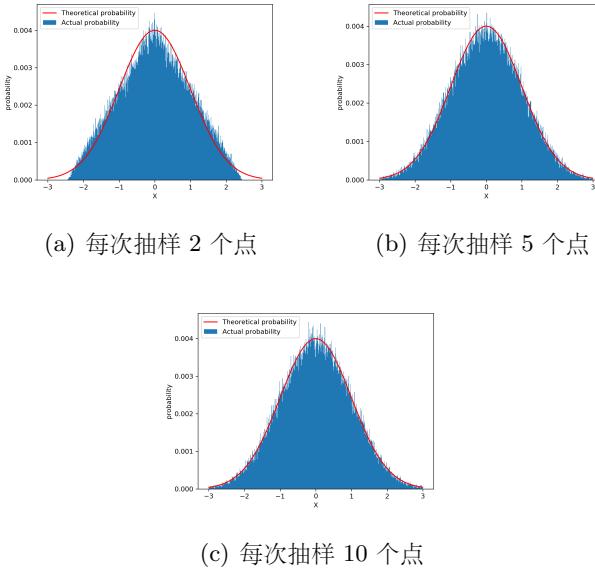
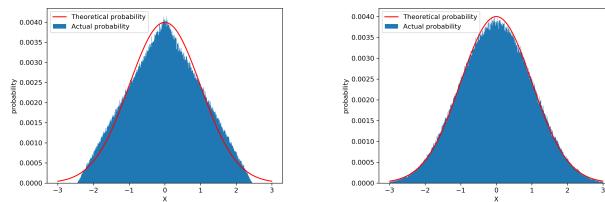
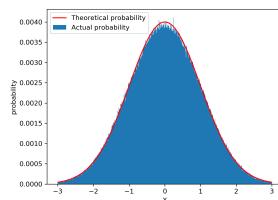


图 9: 2 号分布计算  $10^5$  次后的结果

<sup>3</sup>计算次数控制的为对某每次抽样个数确定的统计量的概率分布精确度，次数越多，其与统计量的理论概率分布约接近，下同。

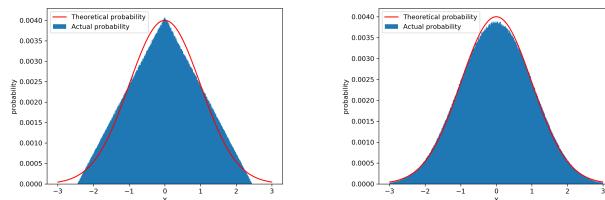


(a) 每次抽样 2 个点 (b) 每次抽样 5 个点

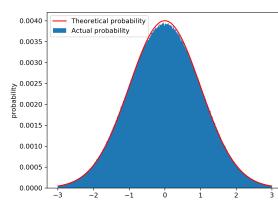


(c) 每次抽样 10 个点

图 10: 2 号分布计算  $10^6$  次后的结果



(a) 每次抽样 2 个点 (b) 每次抽样 5 个点



(c) 每次抽样 10 个点

图 11: 2 号分布计算  $10^7$  次后的结果

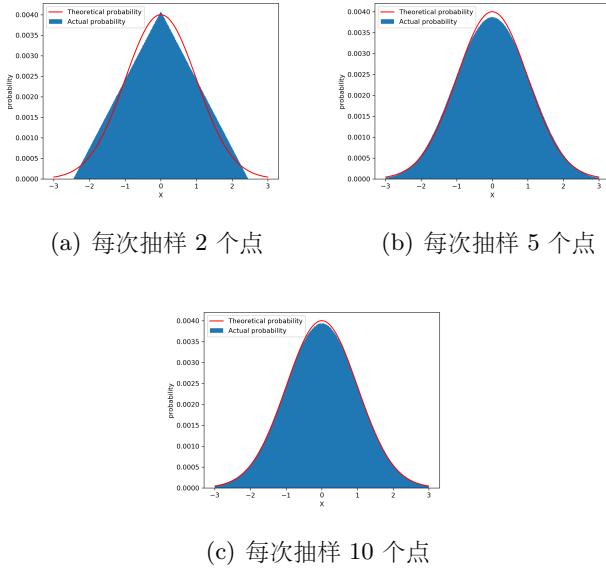


图 12: 2 号分布计算  $10^8$  次后的结果

可以看出对于 2 号分布随着计算次数和每次抽样个数的增加，其统计量的分布与标准正态分布越来越接近。

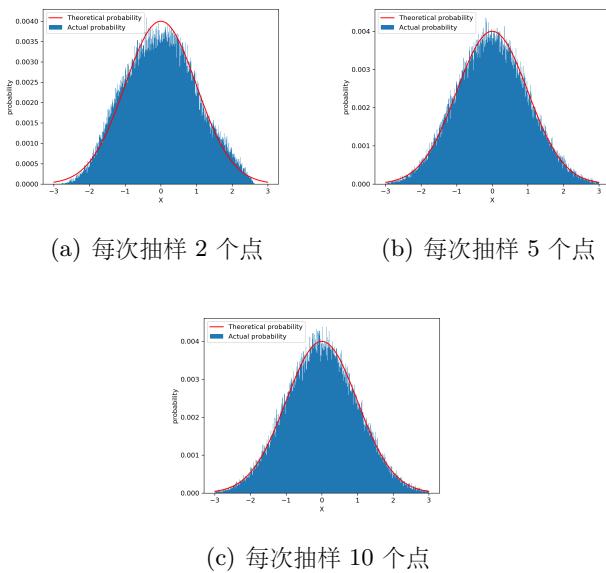
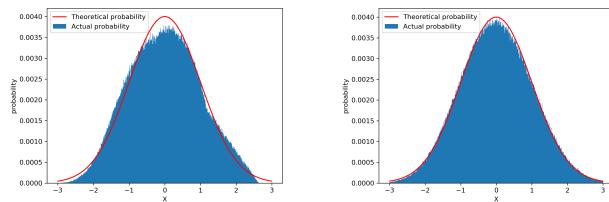
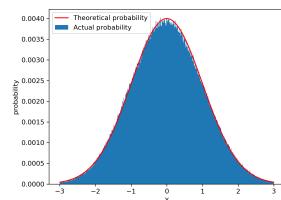


图 13: 3 号分布计算  $10^5$  次后的结果

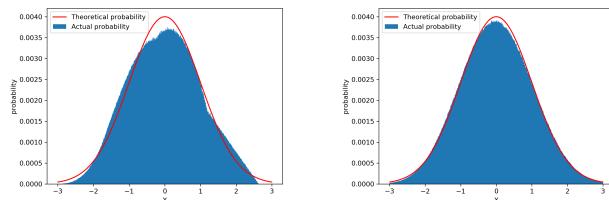


(a) 每次抽样 2 个点 (b) 每次抽样 5 个点

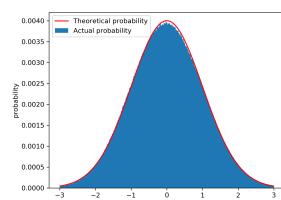


(c) 每次抽样 10 个点

图 14: 3 号分布计算  $10^6$  次后的结果



(a) 每次抽样 2 个点 (b) 每次抽样 5 个点



(c) 每次抽样 10 个点

图 15: 3 号分布计算  $10^7$  次后的结果

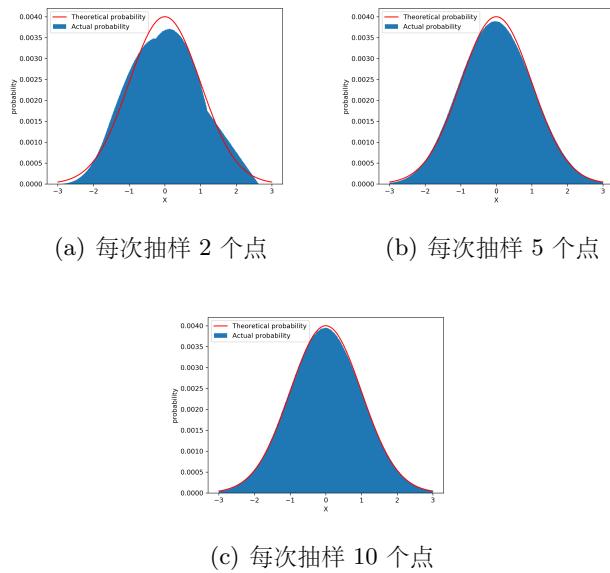


图 16: 3 号分布计算  $10^8$  次后的结果

可以看出对于 3 号分布随着计算次数和每次抽样个数的增加，其统计量的分布与标准正态分布越来越接近。

通过对此 3 个分布的计算结果，看出每次抽样个数越多时，统计量的概率分布越接近于标准正态分布，中心极限定理得到了验证。

## 5 心得与体会

通过此次作业，对中心极限定理有了更深刻的认识。

通过编程作业，也更加熟悉了一些 C 语言和 LATEX。

## 6 附录

### A C 语言源程序

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include <math.h>
5 #define a 16807
6 #define b 0
7 #define m 2147483647
8 #define r (m%a)
9 #define q (m/a)
10 #define Pi 3.1415926
11 #define k 600
12 #define GETMATELEM(base,i,j,imax) ((*(base + i * imax + j)))
    //取二维数组元素
13
14
15
16 int my_filewriter_double(char str[],double num[],int n){
17     FILE * fp;
18     fp = fopen(str,"w+");
19
20     for(int i=0;i<(n-1);i++)
21     {
22         fprintf(fp,"%lf,",num[i]);
23
24     }
25     fprintf(fp,"%lf",num[n-1]); //最后一个数据后不加 ","
26     fclose(fp);
27     return 0;
28 }
```

```

32
33 // Schrage 方法产生随机数
34 int my_schrage(int seed){
35     int x;
36     if(seed == m-1){
37         if(a >= b){ //由于Schrage方法只对z in
38             (0,m-1)成立，故这里要讨论z == m-1的情况
39             x = m + (b-a) % m;
40         }
41     } else x = (b-a) % m;
42 }
43 else x = ((a * (seed % q) - r * (seed / q)) + b % m) % m;
        //递推式
44 return x;
45 }
46
47
48 int my_function1(double p[],int N,int n){
49     double beta; //存放统计量计算结果
50     int x = 1; //16807方法递推数列，初始种子值为1
51     double y; //y为(0,1)均匀抽取的点
52     int flag = 0;
53     for(int i = 0;i<k;i++){ //概率分布数组初始化
54         p[i] = 0;
55     }
56     for(;flag<N;){ //控制计算次数
57         beta = 0;
58         for(int i = 0;i<n;i++){ //控制每次计算时抽取的点数
59             x = my_schrage(x); //随机数递推数列
60             if(x >= 0) y = (double)x/m;
61             else y = (double)(x+m)/m; //y为(0,1)均匀抽取的点
62             if(y <= 0.5){
63                 beta += (sqrt(2*y)-1)/n;
64             }
65             else beta += (1-sqrt(2-2*y))/n;
                //直接抽样法并计算得到第i次抽样结果的平均值
66     }

```

```

67     beta = beta*sqrt(6*n); //计算统计量
68
69     if(beta <= 3 && beta >= -3){ //统计统计量分布
70         p[(int) floor( (beta+3)/6*k) ] += (double)1/N ;
71         flag++;
72     }
73 }
74 return 0;
75 }

76
77 int my_function2(double p[],int N,int n){
78     double beta; //存放统计量计算结果
79     int x = 1; //16807方法递推数列，初始种子值为1
80     double y; //y为(0,1)均匀抽取的点
81     int flag = 0;
82     for(int i = 0;i<k;i++){ //概率分布数组初始化
83         p[i] = 0;
84     }
85     for(;flag<N;){ //控制计算次数
86         beta = 0;
87         for(int i = 0;i<n;i++){ //控制每次计算时抽取的点数
88             x = my_schrage(x); //随机数递推数列
89             if(x >= 0) y = (double)x/m;
90             else y = (double)(x+m)/m; //y为(0,1)均匀抽取的点
91             beta += (y+1)/n;
92             //直接抽样法并计算得到第i次抽样结果的平均值
93         }
94         beta = (beta-1.5)*sqrt(12*n); //计算统计量
95         if(beta <= 3 && beta >= -3){ //统计统计量分布
96             p[(int) floor( (beta+3)/6*k) ] += (double)1/N ;
97             flag++;
98         }
99     }
100    return 0;
101
102
103

```

```

104 int my_function3(double p[],int N,int n){
105     double beta; //存放统计量计算结果
106     int x = 1; //16807方法递推数列，初始种子值为1
107     double y; //y为(0,1)均匀抽取的点
108     int flag = 0;
109     for(int i = 0;i<k;i++){ //概率分布数组初始化
110         p[i] = 0;
111     }
112     for(;flag<N;){ //控制计算次数
113         beta = 0;
114         for(int i = 0;i<n;i++){ //控制每次计算时抽取的点数
115             x = my_schrage(x); //随机数递推数列
116             if(x >= 0) y = (double)x/m;
117             else y = (double)(x+m)/m; //y为(0,1)均匀抽取的点
118             if(y <= 0.5){ //直接抽样法并计算得到第i次抽样结果的平均值
119                 beta += (sqrt(2*y)-1)/n;
120             }
121             else beta += (2*y-1)/n;
122         }
123         beta = 12*(beta-(double)1/12)*sqrt(n)/sqrt(35); //计算统计量
124
125         if(beta <= 3 && beta >= -3){ //统计统计量分布
126             p[(int) floor((beta+3)/6*k)] += (double)1/N ;
127             flag++;
128         }
129     }
130     return 0;
131 }
132
133
134
135 int main(int argc, const char * argv[]) {
136     int N; //总计算次数
137     int n; //每此计算所选点数
138     double p1[k];
139     double p2[k];
140     double p3[k];
141     char str[50];

```

```
142     printf("请输入您所需的计算次数: ");
143     while (!scanf("%d",&N)){ //简单的输入检查
144         gets(str);
145         printf("\nInput error,please try again\n");
146         printf("请输入您所需的计算次数: ");
147     }
148
149     printf("请输入您每次计算所用的点数: ");
150     while (!scanf("%d",&n)){ //简单的输入检查
151         gets(str);
152         printf("\nInput error,please try again\n");
153         printf("请输入您每次计算所用的点数: ");
154     }
155
156     if(N*n >1000000)
157         printf("您输入的参数已接受，正在计算请稍等片刻~\n");
158
159     my_function1(p1,N,n);
160     my_function2(p2, N, n);
161     my_function3(p3, N, n);
162
163     my_filewriter_double("function1.txt", p1, k);
164     my_filewriter_double("function2.txt", p2, k);
165     my_filewriter_double("function3.txt", p3, k);
166
167     return 0;
168 }
```

## B 可视化绘图 python 程序源码

```
1
2 import matplotlib.pyplot as plt
3 from mpl_toolkits.mplot3d import Axes3D
4 import numpy as np
5 #from IPython.core.pylabtools import figsize # import figsize
6 #figsize(12.5, 4) # 设置 figsize
7 plt.rcParams['savefig.dpi'] = 300 #图片像素
8 plt.rcParams['figure.dpi'] = 300 #分辨率
9 # 默认的像素: [6.0,4.0], 分辨率为100, 图片尺寸为 600&400
10 fig = plt.figure()
11 ax1 = fig.add_subplot(111)
12 X = []
13 Y = []
14
15 with open('problem 9/3-8-10.txt', 'r') as f:
16     while True:
17         lines = f.readline() # 整行读取数据
18         if not lines:
19             break
20         Y = [float(i) for i in lines.split(',')]] # 将整行数据分割处理
21         Y = np.array(Y) # 将数据从list类型转换为array类型。
22
23
24 X = np.arange(-2.995, 3.005, 0.01)
25
26 plt.bar(x=X, height=Y, width=0.01, label='Actual probability')
27 ax1.legend(loc=1)
28 ax1.set_ylabel('probability')
29 #plt.savefig("1.png")
30
31
32 oY = np.exp(-X**2/2)*0.01/2.499860889
33
34
35 ax1.plot(X, oY, 'r', label='Theoretical probability')
```

```
36 ax1.legend(loc=2)
37 #plt.ylim([0,0.0055])
38 plt.xlabel('X')
39
40
41 plt.savefig("2.png")
```