计算物理 A 第七题

杨旭鹏 PB17000234

2019 年秋季

1 题目描述

对一个实验谱数值曲线 $p(x)$ ，自设 $F(x)$，分别用直接抽样和舍选法对 $p(x)$ 抽样。比较原曲线和抽样得到的曲线以验证。讨论抽样效率。

2 算法及程序思路

2.1 算法

由于指定的数据文件“data.txt”为离散分布的，故我们这里采用离散抽样。

2.2 直接抽样

由“data.txt”文件我们看到，横坐标能量 (eV)$\in [2900, 3013]$ 共 114 个取值。对于其中某一数据点 x_i, 其对应纵坐标值（即数目）为 n_i，则对于某一次随机抽样，其为 x_i 的概率为 $p_i = \frac{n_i}{\sum_{i=1}^{n-1} n_i}$。

其直接抽样方法为：在 $[0, 1]$ 间由均匀抽样抽取一值 ξ，若 $\sum_{i=1}^{k} p_i < \xi \leq \sum_{i=1}^{k+1} p_i$，则其横坐标能量 x 的取值为 x_k。

2.3 舍选法抽样

观察“data.txt”中给出的数据，我们发先在 $x \leq 2994$ 时计数量相对平稳，其内的概率最大值为在 $x = 2965$ 处取得的约 0.0144(对应原数据计数 5672)。而在 $x \in [2995, 3005]$ 时有一个峰，其概率峰值在 3001 处取得约 0.0951(对应原数据计数 37630)。而在之后 x 对应的概率比较小 (概率

< 0.00253，对应原数据计数 < 1000)。故我们可采取较为简单的分段函数来作为比较函数进行舍选法抽样。

在此题中所使用的比较函数 $F(x)$ 为：

$$F(x) = \begin{cases} 0.0144 & ,x \in [2900, 2994) \\ 0.0951 & ,x \in [2994, 3005) \\ 0.00253 & ,x \in [3005, 3013] \end{cases} \tag{1}$$

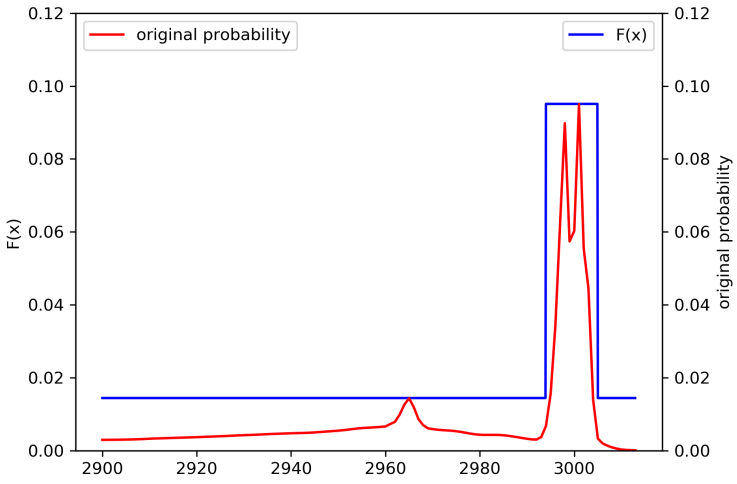

图 1: 比较函数与给定概率分布的比较

其累积函数 $\xi(x)$ 为：

$$\xi(x) = \begin{cases} 0.0144(x - 2900) & ,x \in [2900, 2994) \\ 0.0951(x - 2994) + 1.3536 & ,x \in [2994, 3005) \\ 0.00253(x - 3005) + 1.0461 + 1.3536 & ,x \in [3005, 3013] \end{cases} \tag{2}$$

可知 $\xi(x) \in [0, 2.41994]$, 则可取反函数 $x(\xi)$:[1]

$$x(\xi) = \begin{cases} floor(\frac{\xi}{0.0144} + 2900) & ,\xi \in [0, 1.3536] \\ floor(\frac{\xi - 1.3536}{0.0951} + 2994) & ,\xi \in [1.3536, 1.0461 + 1.3536] \\ floor(\frac{\xi - 1.3536 - 1.0461}{0.00253} + 3005) & ,\xi \in [1.0461 + 1.3536, 0.02024 + 1.0461 + 1.3536] \end{cases} \tag{3}$$

[1]其中 $floor()$ 表示向下取整的函数

化简即：

$$x(\xi)=\begin{cases} floor(\frac{\xi}{0.0144}+2900) & ,\xi\in[0,1.3536] \\ floor(\frac{\xi-1.3536}{0.0951}+2994) & ,\xi\in[1.3536,2.3997] \\ floor(\frac{\xi-2.3997}{0.00253}+3005) & ,\xi\in[2.3997,2.41994] \end{cases} \tag{4}$$

其中，ξ 为 $[0,2.41994]$ 间均匀抽取的随机数。

对于属于不同区间的 ξ 值，我们均匀抽取在不同区间的随机数 η:

$$\begin{cases} \eta\in[0,0.0144] & ,\xi\in[0,1.3536] \\ \eta\in[0,0.0951] & ,\xi\in[1.3536,2.3997] \\ \eta\in[0,0.00253] & ,\xi\in[2.3997,2.41994] \end{cases} \tag{5}$$

则判断 η 是否小于对应 $x(\xi)$ 数据的给定概率 p_i。若是，则取此 $x(\xi)$ 为抽取结果；不是则舍。

2.4 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：

$$\begin{aligned} I_{n+1} &= (aI_n+b)\ mod\ m \\ x_n &= I_n/m \end{aligned} \tag{6}$$

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：

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

即为所谓的 16807 产生器。由于直接利用6编写程序时计算 $(aI_n\ mod\ m)$ 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：

$$aI_n\ mod\ m=\begin{cases} a(I_n\ mod\ q)-r[I_n/q], & if\geq 0 \\ a(I_n\ mod\ q)-r[I_n/q]+m, & otherwise \end{cases} \tag{8}$$

其中 $m=aq+r$，即 $q=m/a=127773$，$r=m\ mod\ a=2836$。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，会自动输出“data.txt”中每一能量值对应的出现概率到“p.txt”。会看到请求输入所需总随机点数的提示，按照提示在后面输入所需要的总随机点数，摁回车继续。然后经过计算会分别给出直接抽样法和舍选法得到的数据到相应的文件。并在屏幕上输出舍选法的抽样效率。程序输出完这些后会自动退出。

```
请输入您所需要的总点数：100
舍选法的抽样效率约为0.384615
Program ended with exit code: 0
```

图 2: 一个典型程序的运行过程示例

4 程序结果与讨论

对于直接抽样法，我们有：

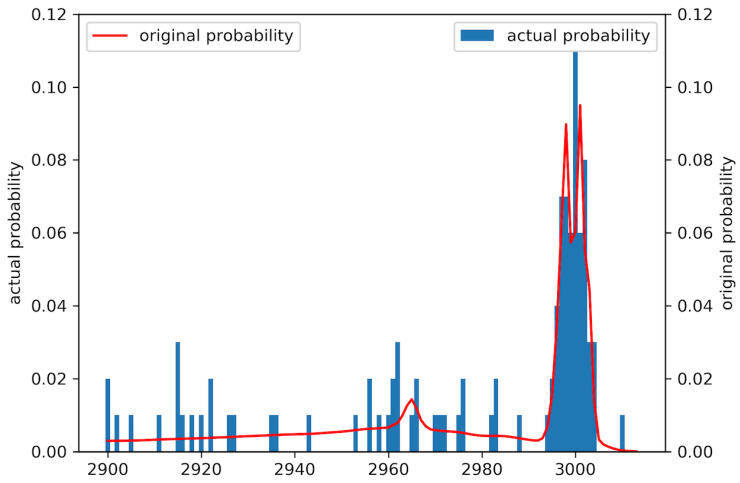

图 3: 直接抽样 10^2 个点得到的概率分布与给定分布的比较

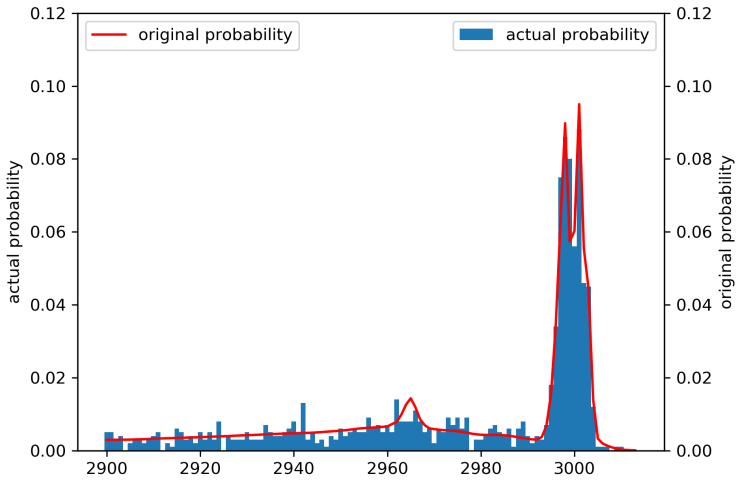

图 4: 直接抽样 10^3 个点得到的概率分布与给定分布的比较

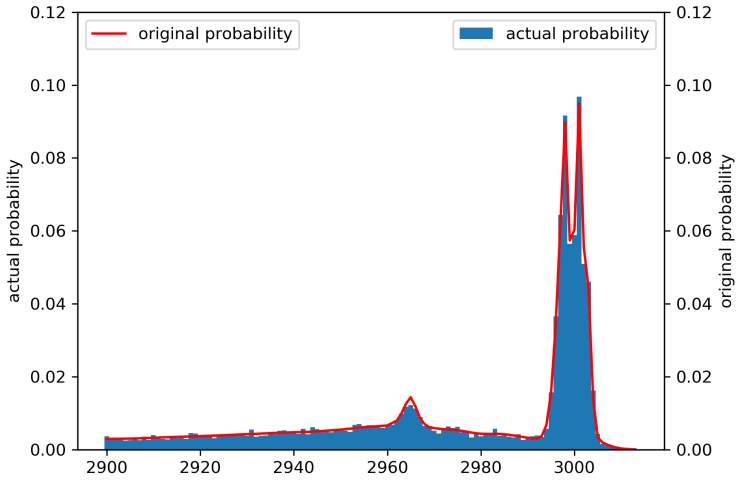

图 5: 直接抽样 10^4 个点得到的概率分布与给定分布的比较

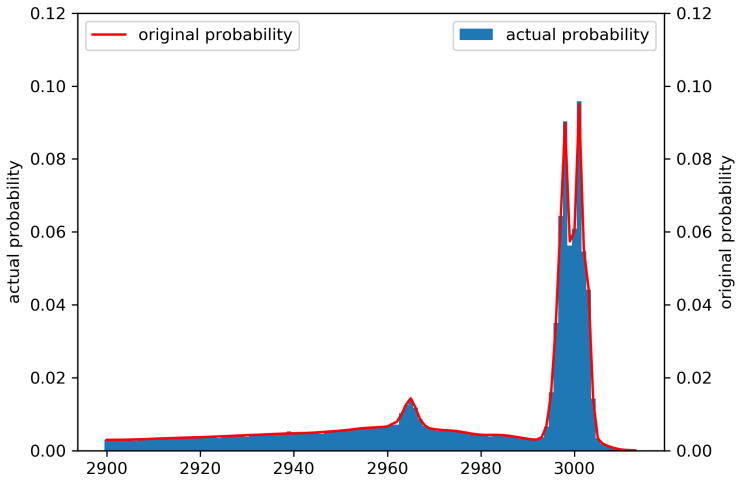

图 6: 直接抽样 10^5 个点得到的概率分布与给定分布的比较

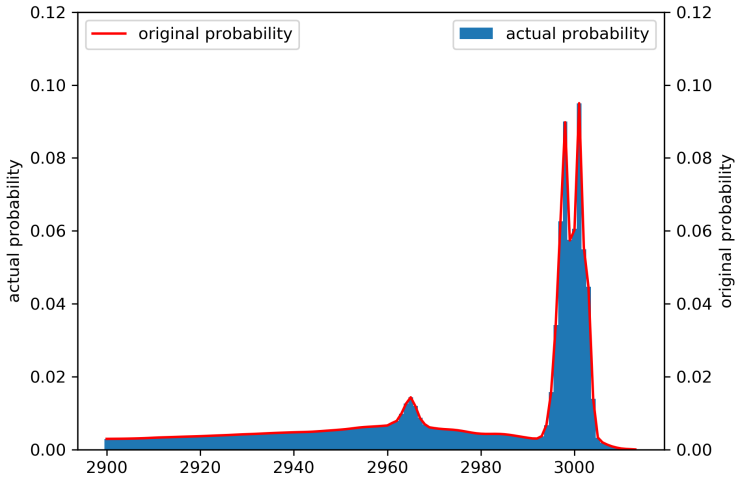

图 7: 直接抽样 10^6 个点得到的概率分布与给定分布的比较

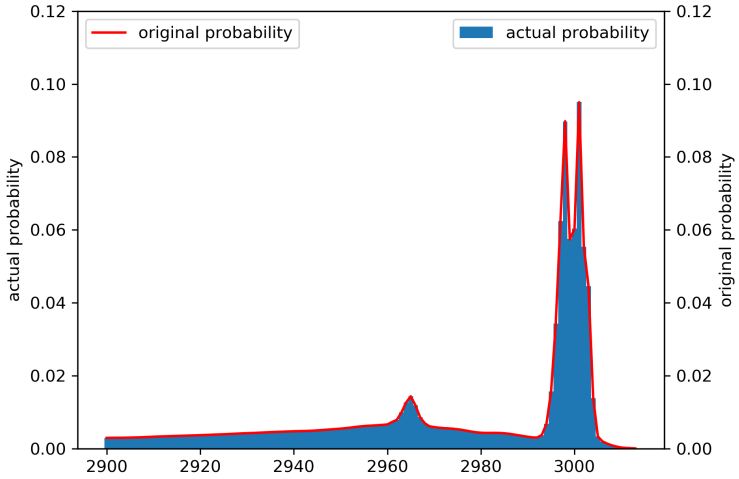

图 8: 直接抽样 10^7 个点得到的概率分布与给定分布的比较

对于舍选法，得到：

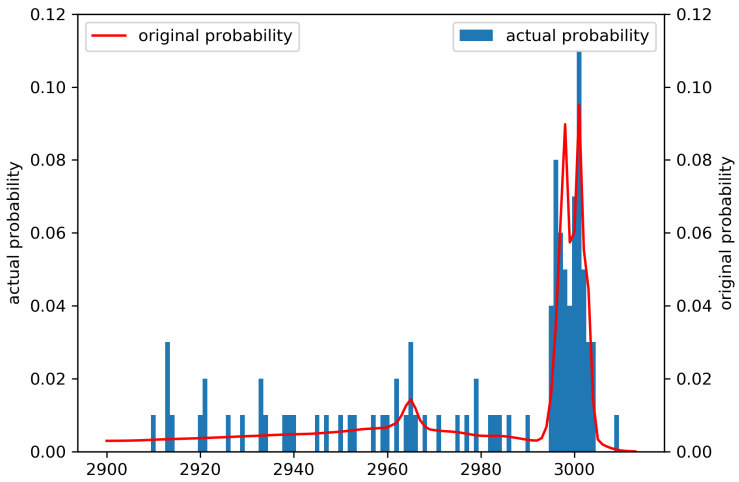

图 9: 舍选法 10^2 个点得到的概率分布与给定分布的比较

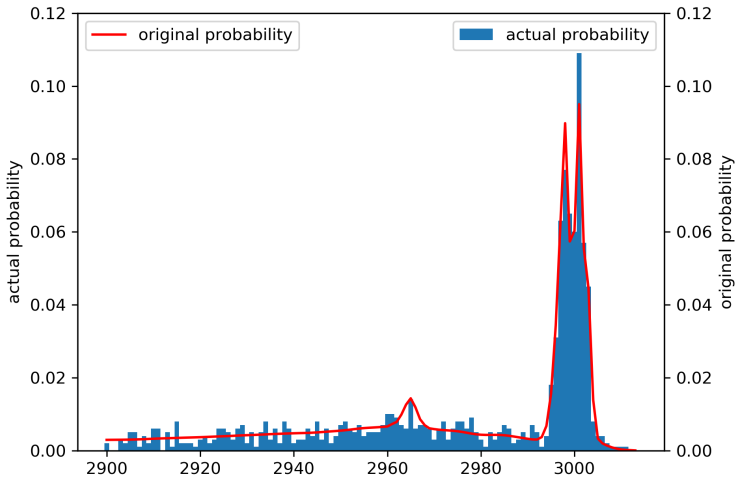

图 10: 舍选法 10^3 个点得到的概率分布与给定分布的比较

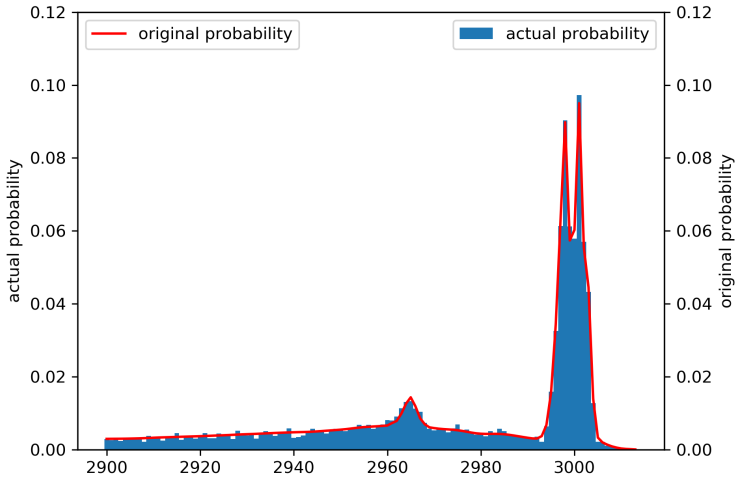

图 11: 舍选法 10^4 个点得到的概率分布与给定分布的比较

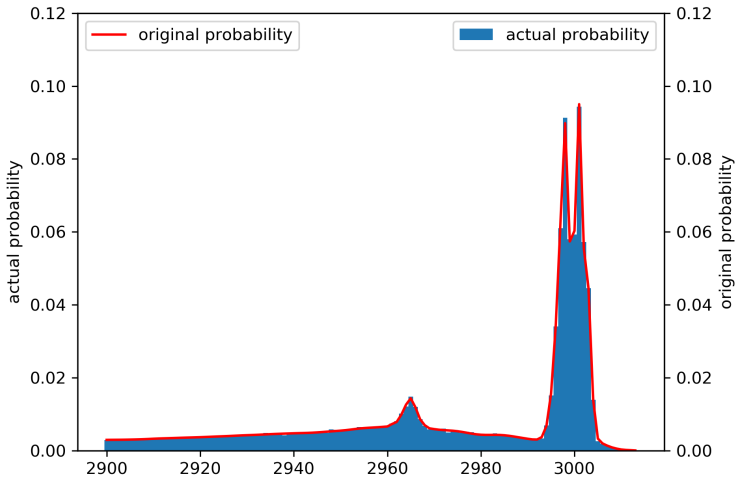

图 12: 舍选法 10^5 个点得到的概率分布与给定分布的比较

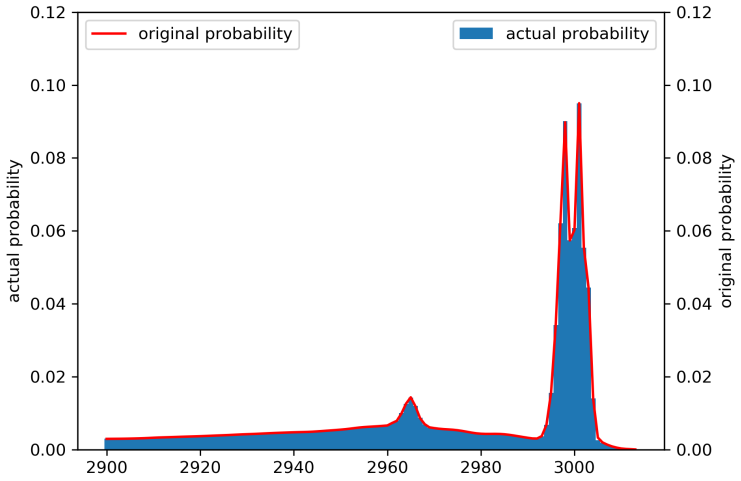

图 13: 舍选法 10^6 个点得到的概率分布与给定分布的比较

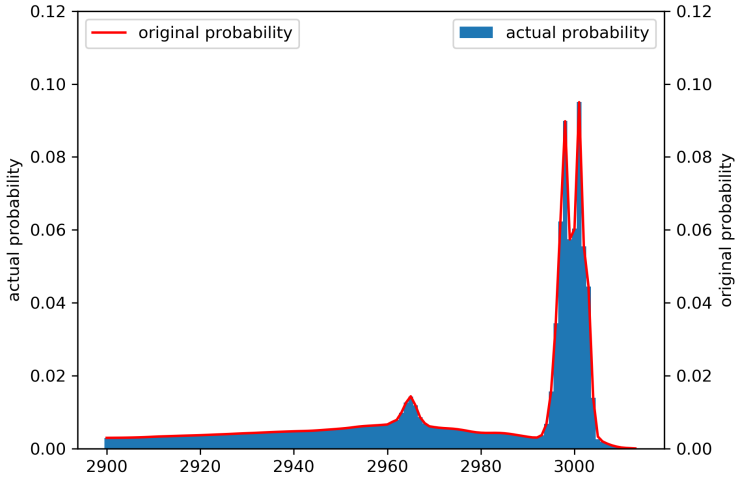

图 14: 舍选法 10^7 个点得到的概率分布与给定分布的比较

舍选法的抽样效率：

	$N = 10^2$	$N = 10^3$	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 10^7$
抽样效率	0.384615	0.417711	0.421799	0.412528	0.413363	0.412934

表 1: 抽样效率一览表

从图中可以看出，不管是哪种抽样方法，在抽样点数变多时，抽样出来的点的概率分布都会收敛到给定概率分布。而直接抽样法的抽样效率永远为 1；舍选抽样法的抽样效率永远都小于 1，换取不同的比较函数可能会提高抽样效率，但相应的比较函数的抽样可能会变得很复杂，浪费编程的时间，在点数不是很多时，利用简单的分段函数作为比较函数会比较有效率。

5 心得与体会

通过此次作业结果，发现不管是直接抽样还是舍选法都能抽出以给定概率分布的点。也联系了对于离散点的抽样方法。通过编程作业，也更加熟悉了一些 C 语言和 LaTeX。

6 附录

A C 语言程序源码

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define a 16807
#define m 2147483647
#define b 0
#define r (m%a)
#define q (m/a)
#define k 114 //能量取值个数

//写文件子程序，输入写成文件名称字符串str，数据来源于数组num，数据总数n
int my_filewriter_double(char str[],double num[],int n){
    FILE * fp;
    fp = fopen(str,"w+");

    for(int i=0;i<(n-1);i++)
    {
        fprintf(fp,"%lf,",num[i]);

    }
    fprintf(fp,"%lf",num[n-1]); //最后一个数据后不加 ","
    fclose(fp);
    return 0;
}


//写文件子程序，输入写成文件名称字符串str，数据来源于数组num，数据总数n
int my_filewriter_int(char str[],int num[],int n){
    FILE * fp;
    fp = fopen(str,"w+");

    for(int i=0;i<(n-1);i++)
```

```
33      {
34          fprintf(fp,"%d,",num[i]);
35
36      }
37      fprintf(fp,"%d",num[n-1]); //最后一个数据后不加 ","
38      fclose(fp);
39      return 0;
40  }
41
42
43
44
45
46  int my_filereader_int(char str[],int num[],int n){
47      FILE * fp;
48      int N = 0;
49
50      if ((fp = fopen(str, "r")) == NULL) {
51          printf("Have problem with opening the data file!\n");
52          return 1;
53      }
54
55      int x;
56      char buff[20];
57      fgets(buff,20, fp); //先读取第一行表头
58
59      for(int i=0;i<n;i++)
60      {
61          fscanf(fp,"%d",&x);
62          fscanf(fp,"%d",&num[i]);
63          N += num[i];
64      }
65      fclose(fp);
66      return N;
67  }
68
69
70
```

```
71 // 16807 之 Schrage 方法产生随机数
72 int my_schrage(double ran[],int seed,int n){
73     for (int i = 0; i < n - 1; i++) {
74         if (seed >= 0) {
75             ran[i] = seed / (double) m;
76         } else {
77             ran[i] = (seed + m) / (double) m;
78         }
79         if(seed == m-1){
80             if(a >= b){   //由于Schrage方法只对z in
                    (0,m-1)成立，故这里要讨论z == m-1的情况
81                 seed = m + (b-a) % m;
82             }
83             else  seed = (b-a) % m;
84 
85         }
86         else seed = ((a * (seed % q) - r * (seed / q)) + b % m ) %
                m; //递推式
87     }
88     if (seed >= 0) {
89         ran[n-1] = seed / (double) m;
90     } else {
91         ran[n-1] = (seed + m) / (double) m;
92     }
93     return 0;
94 }
95 
96 
97 
98 
99 // 舍选法抽样
100 int my_choose(int seed[], int ran[],double orip[], int n){
101     double x;
102     double y;
103     int flag = 0; //记录总抽样次数
104     for (int j = 0; j <= n; ) { //j记录舍选成功的次数
105         flag++;
106 
```

```
107         if (seed[0] >= 0) { //产生在[0,2,4199]间均匀抽取的\xi
108             x = 2.41994*(seed[0] / (double)m) ;
109         }
110         else{
111             x = 2.41994*((seed[0] + m) / (double)m) ;
112         }
113 
114         if (seed[1] >= 0) { //产生在[0,1]间均匀抽取的\eta
115             y = (seed[1] / (double)m);
116         }
117         else{
118             y = ((seed[1] + m) / (double)m);
119         }
120         if(x <= 1.3536) { //对于不同区级的\xi对x,y分别赋值
121             x = x/0.0144 + 2900;
122             y = y*0.0144;
123         }
124         else if(x <= 2.3997 && x > 1.3536) {
125             x = (x-1.3536)/0.0951 + 2994;
126             y = y*0.0951;
127         }
128         else {
129             x = (x-2.3997)/0.00253 + 3005;
130             y = y*0.00253;
131         }
132 
133         if( y <= orip[(int)floor(x-2900)] ){ //舍选条件判断
134            j++;
135            ran[j] = (int)floor(x);
136         }//满足舍选法条件后写入数据
137         seed[0] = ((a * (seed[0] % q) - r * (seed[0] / q)) + b % m )
                % m;
138         seed[1] = ((a * (seed[1] % q) - r * (seed[1] / q)) + b % m )
                % m;
139         if(seed[0] == m-1){
140             if(a >= b){   //由于Schrage方法只对z in
                    (0,m-1)成立，故这里要讨论z == m-1的情况
141                 seed[0] = m + (b-a) % m;
```

```
142             }
143             else  seed[0] = (b-a) % m;
144 
145         }
146         if(seed[1] == m-1){
147             if(a >= b){   //由于Schrage方法只对z in
                    (0,m-1)成立，故这里要讨论z == m-1的情况
148                 seed[1] = m + (b-a) % m;
149             }
150             else  seed[1] = (b-a) % m;
151 
152         }
153     }
154     return flag;
155 }
156 
157 
158 
159 
160 
161 
162 int main(int argc, const char * argv[]) {
163     int x[k];
164     double p[k]; //存放给定数据的的不同x对应的抽到的概率
165     double E[k]; //存放抽样得到的数据的不同x对应的抽到的概率
166     int flag; //舍选法总抽样次数
167     int seed[2] = {34028207,1677078722};
        //舍选法利用的16807方法所需的初始2种子
168     for(int i=0;i<k;i++){ //初始化
169         E[i] = 0;
170     }
171     int N;  //给定数据的计数总数
172     N = my_filereader_int("data.txt", x, k);
173     for(int i=0;i<k;i++){
174         p[i] = (double)x[i]/N;
175     }
176     my_filewriter_double("orip.txt", p, k);
        //输出给定数据的的不同x对应的抽到的概率
```

```
177
178
179     char str[50];
180     printf("请输入您所需要的总点数: ");
181     while (!scanf("%d",&N)) {
            //简单的输入检查，此时N变为需要产生的随机点的总数
182         gets(str);
183         printf("\nInput error,please try again\n");
184         printf("请输入您所需要的总点数: ");
185     }
186     if(N >1000000)
            printf("您输入的参数已接受，正在计算请稍等片刻~\n");
187
188     int *ran1 = malloc(sizeof(int) * N); //用来存放舍选法产生的随机数
189
190     flag = my_choose(seed, ran1, p, N);
191     //my_filewriter_double("ran1,dat",ran1,N);
            //可以输出舍选法抽样得到的数据，此处只计算得到数据的概率分布，故输出略去。
192
193     for(int i = 0;i < N;i++){
194         E[(int)floor(ran1[i]-2900)] += (double)1/N;
                //得到离散的x的取值
195     }
196
197     my_filewriter_double("p-1.txt", E, k);
            //输出舍选法得到的数据的概率分布至文件
198     printf("舍选法的抽样效率约为%lf\n",(double)N/flag);
199
200
201     double * ran2 = malloc(sizeof(double) * N);
            //直接抽样法所用的数据存放数组
202
203     my_schrage(ran2, 1, N); //种子值设为1 //得到[0, 1]均匀抽样的数据点
204
205     for(int i=0;i<k;i++){ //初始化
206         E[i] = 0;
207     }
208
```

```
    for(int i=0;i<N;i++){ //离散数据的直接抽样法
        double sump = 0;
        for (int j=0;j<k;j++){
            sump += p[j];
            if(ran2[i] < sump) {
                ran2[i] = j + 2900;
                E[j] += (double)1/N;
                break;
            }
        }
    }

    //my_filewriter_double("ran2.dat",ran2,N);
        //可以输出直接抽样法抽样得到的数据，此处只计算得到数据的概率分布，故输出略去。
    my_filewriter_double("p-2.txt", E, k);

    return 0;
}
```

B 可视化 python 源程序

[language=python]

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
#from IPython.core.pylabtools import figsize # import figsize
#figsize(12.5, 4) # 设置 figsize
plt.rcParams['savefig.dpi'] = 300 #图片像素
plt.rcParams['figure.dpi'] = 300 #分辨率
# 默认的像素：[6.0,4.0]，分辨率为100，图片尺寸为 600&400
fig = plt.figure()
ax1 = fig.add_subplot(111)

with open('problem 7/F.txt', 'r') as f:
```

```
    while True:
        lines = f.readline() # 整行读取数据
        if not lines:
            break
        Y = [float(i) for i in lines.split(',')] # 将整行数据分割处理
    Y = np.array(Y) # 将数据从list类型转换为array类型。


with open('problem 7/orip.txt', 'r') as f:
    while True:
        lines = f.readline() # 整行读取数据
        if not lines:
            break
        oY = [float(i) for i in lines.split(',')] # 将整行数据分割处理
    oY = np.array(oY) # 将数据从list类型转换为array类型。


X = np.arange(2900, 3014)


plt.bar(x=X, height=Y, width=1, label='actual probability')
ax1.legend(loc=1)
ax1.set_ylabel('actual probability')
ax1.set_ylim([0, 0.12])

ax2 = ax1.twinx()

ax2.plot(X, oY, 'r', label='original probability')
ax2.legend(loc=2)
ax2.set_ylabel('original probability')
ax2.set_ylim([0, 0.12])

plt.xlabel('Energy')


plt.savefig("1.png")
```