计算物理 A 第二题

杨旭鹏 PB17000234

2019 年秋季

1 题目描述

用 16807 产生器测试随机数序列中满足关系 $X_{n-1} < X_{n+1} < X_n$ 的比重。讨论 Fibonacci 延迟产生器中出现这种关系的比重。

2 算法及程序思路

2.1 算法

2.1.1 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:

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

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:

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

即为所谓的 16807 产生器。由于直接利用1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:

$$aI_n\ mod\ m = \begin{cases} a(I_n\ mod\ q) - r[I_n/q], & if \geq 0 \\ a(I_n\ mod\ q) - r[I_n/q] + m, & otherwise \end{cases} \tag{3}$$

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。

2.1.2 Fibonacci 延迟产生器：Marsaglia 1 号产生器

其思想是用序列中的两个整数进行操作得到后续的整数，较线性同余法的优势在于它的周期非常长：

$$I_n = I_{n-p} \otimes I_{n-q} \ mod \ m \tag{4}$$

其中操作符 $\otimes$ 可以是："+"，"−"，"×"，"XOR"。整数对 $[p, q]$ 表示延迟，取值并非按 Fibonacci 数序列，而是根据统计验证后确认。

在此程序中，我们使用的是 Fibonacci 延迟产生器的一个特例——Marsaglia 1 号产生器：

$$I_n = \begin{cases} I_{n-p} - I_{n-q}, & if \geq 0 \\ I_{n-p} - I_{n-q} + 1, & otherwise \end{cases} \tag{5}$$

其中的 $[p, q]$ 整数对的值选为 $[97, 33]$，因此其算法要求存储所有前面的 97 个随机数值。在本程序中，前 97 个随机数值由在 Linux 系统下访问"/dev/random"一次性来得到[1]，[2]并存储为文件以便后期读取使用 (此文件的原始数据见附录 A)。

[1]对于 Linux 下的"/dev/random"文件在 Linux manual page 上有如下解释：The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bits of noise in the entropy pool. From this entropy pool, random numbers are created. 可以看出此种方法产生的随机数并非来自于算法，而来自于热噪声。

[2]便于结果的可重复性，方便调试程序

2.2 程序思路

对于 16807 方法，程序需要一个种子，在本程序中便简单地指定该种子为 1。[3] 产生随机种子后，利用递推公式：

$$I_{n+1} = aI_n \ mod \ m = \begin{cases} a(I_n \ mod \ q) - r[I_n/q], & if \geq 0 \\ a(I_n \ mod \ q) - r[I_n/q] + m, & otherwise \\ a = 7^5 = 16807 \\ q = m/a = 127773 \\ r = m \ mod \ a = 2836 \end{cases} \tag{6}$$

计算随机数列。对于 Marsaglia 1 号产生器，所需要 97 个初始随机数，得到方式在上文已经提及。

得到随机数列后，我们即可通过循环的方式，找出满足 $X_{n-1} < X_{n+1} < X_n$ 关系的个数 n，然后和总序列值 $(N-2)$（N 为随机数列所含随机数的总个数）的比值即为出现这种关系的比重。例如对于随机数的数列 $[1,3,2,5,6]$，满足此种关系的只有序列 $[1,3,2]$, 则比重计算式即为

$$\eta = \frac{n}{N-2} = \frac{1}{3} \doteq 0.33333 \tag{7}$$

详情请见代码。

3 程序使用方法

在运行程序后，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。然后经过计算给出 16807 方法产生的随机数中满足关系 $X_{n-1} < X_{n+1} < X_n$ 的比重。然后按照程序提示，输入 Marsaglia 1 号产生器所需延迟整数值 $[p,q]$（最大值不可超过 97，Marsaglia 1 号产生器简易使用默认延迟整数值为 $[97,33]$，建议按此输入），按回车继续程序。按照屏幕提示进一步确定计算比例时是否排除初始的随机数 (是输入 1，不是输入 0)，回车继续，程序即会自动在屏幕上显示 Marsaglia 1 号产生器产生的随机数中满足关系 $X_{n-1} < X_{n+1} < X_n$ 的比重。程序输出完这些后会自动退出。

[3] 当然在 Linux 下也可以访问“/dev/random”来获取相对“真”的随机数种子，在这里我们这么做是为了结果的可重复性

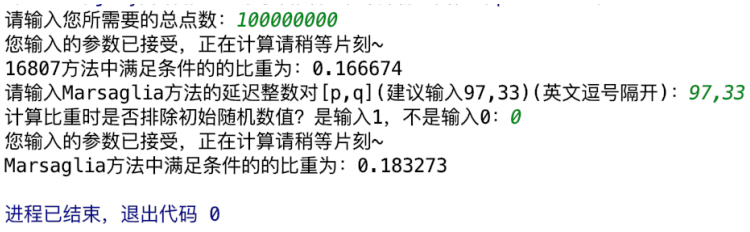

图 1: 一个典型程序的运行过程示例

4 程序结果与讨论

当 Marsaglia 1 号产生器的前 97 个随机数按照附录 A 给出的数值进行计算时，我们给出以下结果：

	$N=10^2$	$N=10^3$	$N=10^4$	$N=10^5$	$N=10^6$	$N=10^7$	$N=10^8$
16807 产生器满足关系比重	0.153061	0.160321	0.165833	0.165303	0.166620	0.166592	0.166674
Marsaglia 1 号产生器满足关系比重	0.183673	0.178357	0.183337	0.184034	0.184024	0.183076	0.183273

表 1: 满足 $X_{n-1} < X_{n+1} < X_n$ 关系比重一览表

由上表看出，对于 16807 产生器，满足关系比重在 $N < 10^4$ 时随其增大而增大，当 $N > 10^4$ 时没有明显关系，并最终趋于稳定值。[4]而对于 Marsaglia 1 号产生器，与 N 的关系不明显，不过这一点更有可能是由前 97 个随机数是由读取“/dev/random”文件得到的原因。为了排除这种可能，我们计算 Marsaglia 1 号产生器产生随机数列中满足 $X_{n-1} < X_{n+1} < X_n$ 关系比重时排除了前 97 个值，得到如下结果：

	$N=10^3-97$	$N=10^4-97$	$N=10^5-97$	$N=10^6-97$	$N=10^7-97$	$N=10^8-97$
Marsaglia 1 号产生器满足关系比重	0.186459	0.183416	0.184042	0.184024	0.183076	0.183273

表 2: Marsaglia 1 号产生器刨除前 97 个随机数后满足 $X_{n-1} < X_{n+1} < X_n$ 关系比重一览表

由上表看出对于 Marsaglia 1 号产生器，刨除前 97 个初始随机数后，但其满足 $X_{n-1} < X_{n+1} < X_n$ 关系比重与 N 的关系仍然不明显。通过两个随机数产生器产生的结果比较，Marsaglia 1 号产生器中满足关系的比例要大于 16807 产生器的。产生如此结果的具体原因不明。由于对于理想的均匀抽取的随机数列，其满足此种关系的比例为 $\frac{1}{6} \doteq 16.67\%$，可看出在点数很多的时候，16807 产生器产生的随机数列中满足此种关系的比例趋近于此值，而

[4]由表中看出 $N > 10^7$ 时其变化量 $< 10^{-4}$

Marsaglia 1 号产生器产生的随机数列中满足此种关系的比例比此值要大。可以看出 16807 产生器产生的随机数列在某种程度上均匀性比 Marsaglia 1 号产生器产生的要好。

5 心得与体会

通过此次作业结果，发现 Marsaglia 1 号产生器产生的随机数列值并不是很均匀。

通过编程作业，也更加熟悉了一些 C 语言和 LaTeX。

6 附录

A Marsaglia 1 号产生器前 97 个随机数值

34028207,1677078722,340727950,236401562,1063000778,-1889330694,-362720814,
1001156461,773939007,591817669,-1289745081,-1014809404,-461409850,761086960,
790966442, 440866592,449400572,-955282719,1292837573,1502388563,-1368094447,
-1113617160, -1377400795,123704243,-618974543,731701529,61406536,-716590537,
-1498636901,-1873427357,-127500922,591237362,-1194899970,-1880293183,-894137448,
1367301471,-1692689541,-1374988845,1365918001,1115567034,2027089295,-1433358734,
1707969962,-1539357866,1554870558,-2870946,694356600,-209537731,1101478399,
-1017918007,576659320,-619898998,-1127461630,2131790151,972605678,1648439350,
2053601537,-766916429,380626540,-1370106699,301497204,-658164181,1390028910,
-1051418226,-1667813809,-1624802196,1768820847,374599957,736737858,433591492,
-2093954975,-1264640940,453354730,2104986002,644165957,711527726,93928571,
2132331337,1355671981,-30921655,-65350020,-1824844232,-1348059645,1327131520,
1647560514,58434354,-831533038,203200978,-1828958168,497737622,-1761196784,
731382657,-1856004511,792328231,609589382,-120097207,32766

B 产生上述随机数的 C 语言程序

```c
#include <stdio.h>
#include <stdlib.h>



 //利用/dev/random 产生随机种子
int my_randomseed(int seed[],int n){
    //seed为存储随机种子的数据，n为所需要种子的个数
    FILE * fp1 = fopen("/dev/random","r");
    for(int i=0;i<n;i++){
        fread(&seed[i], 1, sizeof(int), fp1);
    }
    fclose(fp1);
```

```
    return 0;
}



int main(){
int ranI[97];
   my_randomseed(ranI,97);
   FILE * fp;
   fp = fopen("ranI.txt","w+");

   for(int i=0;i<96;i++)
   {
       fprintf(fp,"%d,",ranI[i]);

   }
   fprintf(fp,"%d",ranI[96]); //最后一个数据后不加 ","
   fclose(fp);
   return 0;
   }
```

C C 语言程序源码

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define a 16807
#define m 2147483647
#define b 0
#define r (m % a)
#define q (m / a)


int my_filereader_int(char str[],int num[],int n){
```

```
13      FILE * fp;
14      fp = fopen(str,"r");
15
16      for(int i=0;i<n;i++)
17      {
18          fscanf(fp,"%d,",&num[i]);
19
20      }
21      fscanf(fp,"%d",&num[n]); //最后一个数据后不加 ","
22      fclose(fp);
23      return 0;
24  }
25
26  // 16807 之 Schrange 方法产生随机数
27  int my_schrage(double ran[],int ranI[],int n){
28      for (int i = 0; i < n - 1; i++) {
29          if (ranI[i] >= 0) {
30              ran[i] = ranI[i] / (double) m;
31          } else {
32              ran[i] = (ranI[i] + m) / (double) m;
33          }
34          if(ranI[i] == m-1){
35              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
36                  ranI[i+1] = m + (b-a) % m;
37              }
38              else  ranI[i+1] = (b-a) % m;
39
40          }
41          else ranI[i+1] = ((a * (ranI[i] % q) - r *( ranI[i] / q) ) +
                b % m ) % m; //递推式
42      }
43      if (ranI[n-1] >= 0) {
44          ran[n-1] = ranI[n-1] / (double) m;
45      } else {
46          ran[n-1] = (ranI[n-1] + m) / (double) m;
47      }
48      return 0;
```

```
}

//Fibonacci延迟器
int my_fibonacci(double ran[],int ranI[],int n,int o,int p){
    int temp;
    for(int i=1;i < p;i++){ //先把ran的前p-1项计算出来
        if (ranI[i] >= 0) {
            ran[i-1] = ranI[i] / (double) m;
        } else {
            ran[i-1] = (ranI[i] + m) / (double) m;
        }
    }
    for (int i = p; i <= n; i++) { //ranI[i % p]存放的为第i项

        temp = (ranI[i % p] - ranI[i % p+(p-o)]) ; //递推式

        if(temp >= 0) ranI[i % p] = temp ;
        else ranI[i % p] = temp + 1;
            //递推得到的新结果放入ranI中i%p一项

        if (ranI[i % p] >= 0) { //计算ran
            ran[i-1] = ranI[i % p] / (double) m;
        } else {
            ran[i-1] = (ranI[i % p] + m) / (double) m;
        }

    }
    return 0;
}


int main() {
    int N; //总随机数个数
    double x,y,z; //用来存放X_{n-1},X_{n},X_{n+1}
    int flag = 0; //记录满足条件的个数
    double *ran = malloc(sizeof(double) * N); //用来存放随机数
    int *ranI = malloc(sizeof(int) * (N+1)); //用来存放递推数列
    int o,p;  //Fibonacci方法的延迟整数对
```

```
86
87      char str[50]; //用来进行输入检查时用来存放的临时数组
88      printf("请输入您所需要的总点数: ");
89      while (!scanf("%d",&N)) { //简单的输入检查
90          gets(str);
91          printf("\nInput error,please try again\n");
92          printf("请输入您所需要的总点数: ");
93      }
94      if(N >1000000)
            printf("您输入的参数已接受, 正在计算请稍等片刻~\n");
95
96
97    ranI[0]= 1; //手动指定16807方法的种子值为1
98
99      my_schrage(ran,ranI,N); //16807方法产生随机数组
100
101     for(int i = 0;i < (N-2);i++){ //计算满足关系式的比重
102         x = ran[i];
103         y = ran[i+1];
104         z = ran[i+2];
105         if(x < z && z < y) flag ++;
106     }
107     //my_filewriter_double("16807_ran.dat",ran,N);
            //将产生的随机数组存储为文件
108     printf("16807方法中满足条件的的比重为: %lf\n",flag/(double)(N-2));
109
110     inputpq:printf("请输入Marsaglia方法的延迟整数对[p,q]
111                    (建议输入97,33)(英文逗号隔开): ");
112     while (!scanf("%d,%d",&o,&p)) { //相对简单的输入检查
113         gets(str);
114         printf("\nInput error,please try again\n");
115         printf("请输入Marsaglia方法的延迟整数对[p,q]
116                (建议值97,33)(英文逗号隔开): ");
117     }
118
119     int temp;
120     if(o > p){  //使 p > o
121         temp = o;
```

```
122         o = p;
123         p=temp;
124     }
125     if( N-p < 3 ) { //排除N相对于p,q太小的情况发生
126         printf("输入的p,q值相对于总点数太大了，请重试!\n");
127         goto inputpq;
128     }
129
130     printf("计算比重时是否排除初始随机数值? 是输入1，不是输入0: ");
131     int u; //是否排除初始随机数值得flag
132     while (!scanf("%d",&u)||(u != 0 && u != 1) ) {
            //相对简单的输入检查
133         gets(str);
134         printf("\nInput error,please try again\n");
135         printf("计算比重时是否排除初始随机数值? 是输入1，不是输入0: ");
136     }
137
138     if(N >1000000)
            printf("您输入的参数已接受，正在计算请稍等片刻~\n");
139     free(ranI);
140     free(ran);  //清除上面16807方法产生的数值
141     ran = malloc(sizeof(double) * N); //用来存放随机数
142     ranI = malloc(sizeof(int) * (N+1)); //用来存放递推数列
143
144
145
146     my_filereader_int("ranI.txt",ranI,p);
            //读取之前利用"/dev/random"产生的初始随机数组
147     my_fibonacci(ran,ranI,N,o,p); //Fibonacci方法产生随机数组
148     //my_filewriter_double("Marsaglia_ran.dat",ran,N);
149
150     flag = 0;
151     if(u == 1) {
152         for (int i = p; i < (N-2); i++) {
153             x = ran[i];
154             y = ran[i + 1];
155             z = ran[i + 2];
156             if (x < z && z < y) flag++;
```

```
157         }
158         printf("Marsaglia方法中满足条件的的比重为: %lf\n",
159                 flag/(double)(N-p-2));
160     }
161     if(u == 0) {
162         for (int i = 0; i < (N-2); i++) {
163             x = ran[i];
164             y = ran[i + 1];
165             z = ran[i + 2];
166             if (x < z && z < y) flag++;
167         }
168         printf("Marsaglia方法中满足条件的的比重为: %lf\n",
169                 flag/(double)(N-2));
170     }
171 
172     return 0;
173 }
```