2022 年秋计算物理——Homework1

中国科学技术大学物理学院 2020 级
曾郅琛 PB20071431

2022 年 10 月 5 日

摘要：利用 C++ 及 Python 解决以下问题：用 Schrage 方法编写随机数子程序，用指定间隔（非连续 $l > 1$）两个随机数作为点的坐标值绘出若干点的平面分布图。再用 $< x^k >$ 测试均匀性（取不同量级的 N 值，讨论偏差与 N 的关系）、$C(l)$ 测试其 2 维独立性（总点数 $N > 10^7$）

关键字：Schrage 方法；均匀性；独立性

1 算法及实现

1.1 16807 产生器 Schrage 方法

利用线性同余法：$I_{n+1} = (aI_n + b) \bmod m$, $x_n = I_n/m$, 取 $a = 16807, b = 0, m = 2^{31} - 1$, 在此基础上，通过 Schrage 方法，得到 $az \bmod m$ 值大小，即可产生不超过计算机 32 位机的最大数据范围。

算法实现如下：

- 利用教材上 1.1.1.4 种子值方法生成关于系统时间的种子值 $Seed(\,)$ 函数，作为起始随机数值；
- 接着利用生成的种子值，在 Sch 16807$(int\ z)$ 中可以返回第二个随机数，作为递归子函数；
- 之后，通过 $Save\ Random(int\ seed, int\ N, char\ const\ *url)$ 函数，在生成的种子基础上不断生成随机数 z, 并每次计算 $z/(2^{31} - 1)$, 得到指定数目 N 个 $[0, 1]$ 范围内随机数，并存入 $*url$ 文件内。
- 最后，将以上函数封装成类 $class\ Sch$ ，并组织成 $main$ 函数的库函数。

```
class Sch {
public:
    int Seed();
    int Sch_16807(int z);
    void Save_Random(int seed, int N, char const url[]);
};
```

图 1: 类封装函数

1.2 均匀性测试检验

在均匀性检验过程中读文件按行读取数据，并将读取 string 转化为 double 类型：

```cpp
double xk_average(int k, int N, char const p[]){
    ifstream fp;
    fp.open( s: p, mode: ios::in);
    string line;
    double s=0;
    if(fp) // 有该文件
    {
        while (getline( &: fp, &: line))
        {
            istringstream sin( str: line);
            string Waypoints;
            getline( &: sin, &: Waypoints);
//          cout << "Waypoints:" << Waypoints << endl;
            double x;
            stringstream sx; //transform string to double
            sx << Waypoints;
            sx >> x;
            s = s + pow(x, y: k);
        }
    }
    else cout <<"no such file" << endl;
    fp.close();
    return s/N;
}
```

图 2: $< x^k >$ 计算函数

1.3 独立性测试检验

由概率论知识，独立性检验标准中间距为 l 的自相关函数 $C(l)$:

$$C(l) =< x_n x_{n+1} > - < x_n >^2 / < x_n^2 > - < x_n >^2$$

在 $< x^k >$ 计算函数基础上，只需计算 $< x_n x_{n+1} > - < x_n >^2$ 大小，在算法实现中，通过创建一个动态数组，先读出 l 个数据，此后的数据在读取时更新动态数组，如此持续下去，最后释放空间，使用动态数组避免占用缓存空间：

```cpp
    auto* tempArray = (double*)malloc(l * sizeof(double));
    if(fp) // 有该文件
    {
        for(i=0;i<l;i++)
        {
            getline( &: fp, &: line);
            istringstream sin( str: line);
            string Waypoints;
            getline( &: sin, &: Waypoints);
            stringstream sx; //transform string to double
            sx << Waypoints;
            sx >> x;
            tempArray[i]=x;
        }
        for (int j = l; j < N; j++) {
            getline( &: fp, &: line);
            istringstream sin( str: line);
            string Waypoints;
            getline( &: sin, &: Waypoints);
            stringstream sx; //transform string to double
            sx << Waypoints;
            sx >> x;
            s += x * tempArray[(i - l)%l];
            tempArray[i%l] = x;
        }
    }
    else cout <<"no such file" << endl;
    free(tempArray);
    fp.close();
```

图 3: 自相关函数 $C(l)$ 核心代码

2 结果分析讨论

2.1 随机数平面分布图

用指定间隔（非连续 l >1），本实验取 l=4，两个随机数作为点的坐标值绘出若干点的平面分布图。利用 python 中 matplolib 库函数作图如下：

```python
import matplotlib.pyplot as plt
f=open(".\\cmake-build-debug\\Random_data.txt")
a = f.read()
p = list(float(i) for i in a.split())
plt.scatter(p[0:100000:4],p[4:100001:4],marker="*",s=0.1, c='blue')
plt.xlabel("x random")
plt.ylabel("y random")
plt.title("when l = 4")
plt.savefig("Scatter_plot.png")
plt.show()
```

图 4: 随机数分布图代码

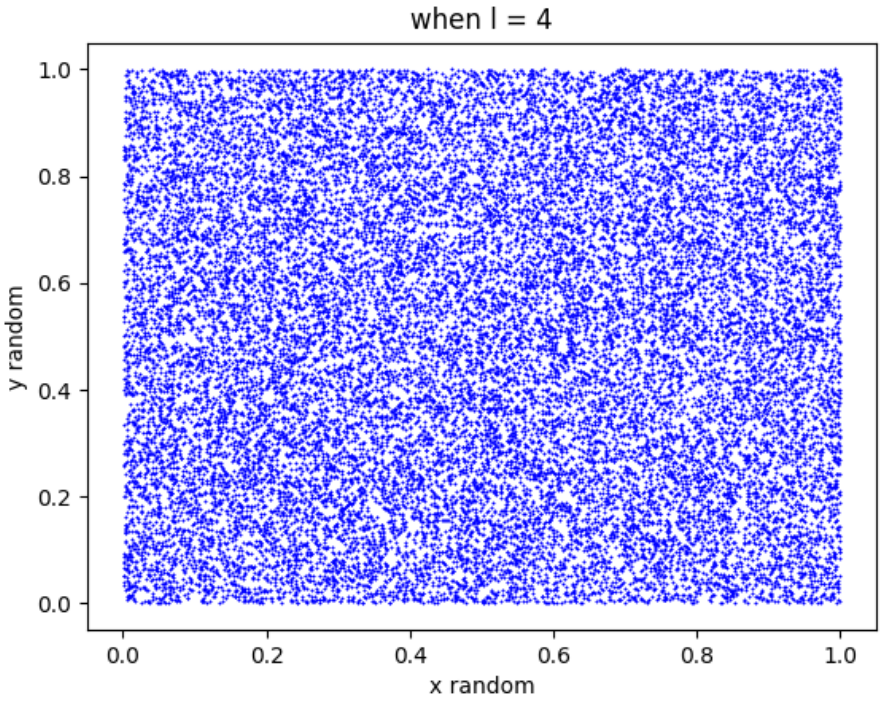

图 5: 随机数二维分布图

2.2 均匀性检验结果

```
均匀性验证：
when k=1,N=100, <x^k>: 0.506734
   <x^k>-1/k+1: 0.00673353
when k=2,N=100, <x^k>: 0.331729
   <x^k>-1/k+1: -0.00160453
when k=4,N=100, <x^k>: 0.196136
   <x^k>-1/k+1: -0.00386395
when k=7,N=100, <x^k>: 0.121761
   <x^k>-1/k+1: -0.00323869
when k=1,N=1000, <x^k>: 0.484923
   <x^k>-1/k+1: -0.0150768
when k=2,N=1000, <x^k>: 0.319647
   <x^k>-1/k+1: -0.0136864
when k=4,N=1000, <x^k>: 0.190282
   <x^k>-1/k+1: -0.0097179
when k=7,N=1000, <x^k>: 0.117883
   <x^k>-1/k+1: -0.00711707
when k=1,N=10000, <x^k>: 0.501177
   <x^k>-1/k+1: 0.00117709
```

图 6: result-1

```
when k=2,N=10000, <x^k>: 0.33386
   <x^k>-1/k+1: 0.000526554
when k=4,N=10000, <x^k>: 0.199661
   <x^k>-1/k+1: -0.000338954
when k=7,N=10000, <x^k>: 0.123848
   <x^k>-1/k+1: -0.00115184
when k=1,N=100000, <x^k>: 0.500033
   <x^k>-1/k+1: 3.32115e-05
when k=2,N=100000, <x^k>: 0.333182
   <x^k>-1/k+1: -0.000150974
when k=4,N=100000, <x^k>: 0.199583
   <x^k>-1/k+1: -0.000416541
when k=7,N=100000, <x^k>: 0.124389
   <x^k>-1/k+1: -0.000611338
when k=1,N=500000, <x^k>: 0.499751
   <x^k>-1/k+1: -0.000248535
when k=2,N=500000, <x^k>: 0.333092
   <x^k>-1/k+1: -0.00024129
```

图 7: result-2

```
when k=4,N=500000, <x^k>: 0.199845
   <x^k>-1/k+1: -0.000154801
when k=7,N=500000, <x^k>: 0.124929
   <x^k>-1/k+1: -7.11791e-05
when k=1,N=1000000, <x^k>: 0.500386
   <x^k>-1/k+1: 0.000385892
when k=2,N=1000000, <x^k>: 0.333644
   <x^k>-1/k+1: 0.000310187
when k=4,N=1000000, <x^k>: 0.200177
   <x^k>-1/k+1: 0.000176829
when k=7,N=1000000, <x^k>: 0.125088
   <x^k>-1/k+1: 8.77116e-05
when k=1,N=10000000, <x^k>: 0.500056
   <x^k>-1/k+1: 5.55248e-05
when k=2,N=10000000, <x^k>: 0.333423
   <x^k>-1/k+1: 8.95177e-05
when k=4,N=10000000, <x^k>: 0.200116
   <x^k>-1/k+1: 0.000115719
when k=7,N=10000000, <x^k>: 0.125122
   <x^k>-1/k+1: 0.0001221
```

图 8: result-3

由结果可以看出:

- $< x^k >$ 与 $1/(k+1)$ 间距在 0.001 0.000001 范围内，差距较小;
- 随着 N 的增大，$< x^k >$ 与 $1/(k+1)$ 间距呈现下降趋势;
- $< x^k > -1/(k+1)$ 大小与 $1/\sqrt{N}$ 呈现一定的正比关系;

N/k	100	1000	10000	100000	500000	1000000	10000000
1	0.0067335	0.0150768	0.00117709	0.0000332	0.0002485	0.00038589	0.0000555
2	0.0016045	0.0136864	0.00052655	0.0001510	0.0002413	0.00031019	0.0000895
4	0.0038640	0.0097179	0.00033895	0.0004165	0.0001548	0.00017683	0.0001157
7	0.0032387	0.0071171	0.00115184	0.0006113	0.0000712	0.00008771	0.0001221

图 9: $| < x^k > -1/(k+1)|$ 与 N、k 关系

2.3 独立性检验结果

```
独立性验证：
when l=1, N=1e7, C(l) = -0.000113138
when l=2, N=1e7, C(l) = -0.000113952
when l=3, N=1e7, C(l) = -0.000114723
when l=4, N=1e7, C(l) = -0.000115467
when l=5, N=1e7, C(l) = -0.000116139
```

图 10: $C(l)$ 在 $N = 10^7$ 结果

理想情况下，$C(l)=0$，在实验中由于随机数的产生并非真随机数，故而存在一定的相关性，所以即使当 $N=10^7$ 时，$C(l)$ 仍然在 0.0001 量级，无法做到完全为 0。

3 总结与心得

在此次实验中自我编写了 16807 随机数生成器，并通过 Schrage 方法解决了溢出问题，最后通过取点画图以及均匀性验证，证明了随机数较为均匀；同时也在验证独立性过程中发现该随机数产生并非每个数都是独立的，存在一定的相关性，也间接证明了此方法为伪随机数产生方法。

从个人而言，此次实验学会 C++ 编写以及 matplotlib 画图，进一步提高编程能力，也为后面蒙特卡罗方法奠定基础。