计算物理作业 1

刘畅, PB09203226

2012 年 9 月 30 日

[作业 1] 用 Schrage 方法编写随机数子程序, 用连续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 $\langle x^k \rangle$ 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 $N > 10^7$)。与前面的 `randomz` 子程序进行比较 (采用同样的常数以及单精度或双精度实数运算), 总结和评价 2 个随机数产生器的随机性质量。

1 用 Schrage 方法编写随机数发生器

按照书上的公式

$$az \bmod m = \begin{cases} a(z \bmod q) - r\lfloor z/q \rfloor & \text{if } \geq 0 \\ a(z \bmod q) - r\lfloor z/q \rfloor + m & \text{otherwise} \end{cases}$$

编程, 其中最主要的代码是 (见 `schrage.c`).

```
double rand_schrage(double x)
{
    int32_t z = x * CONST_M;
    int32_t q = CONST_M / CONST_A;
    int32_t r = CONST_M % CONST_A;
    int32_t retval = CONST_A * (z % q) - r * (z / q);
    if (retval < 0)
        retval += CONST_M;
    return (double) retval / (double) CONST_M;
}
```

其中的 CONST_* 是各种常数.

将给出的 randomz 例程转换成 C 语言, 主要的代码是 (见 randomz.c)

```
double randomz(double x)
{
    return fmod(CONST_A * x * 1e+8, CONST_M) * 1e-8;
}
```

同样, 其中的 CONST_* 是各种常数.

2 做 (x_i, x_{i+1}) 的平面分布图

对两种随机数发生器, 取定种子值后生成 (x_i, x_{i+1}) 对, 打印到屏幕上. 主要代码如下, 见 generate_x_y_dataset.c.

```
void test_rand(int nsteps, FILE *fout, double (*prand)(double))
{
    ...
    for (i = 0; i < nsteps; i++) {
        z_next = prand(z); /* generate random number */
        fprintf(fout, "%.12f\t%.12f\n", z, z_next);
        z = z_next;
    }
}
```

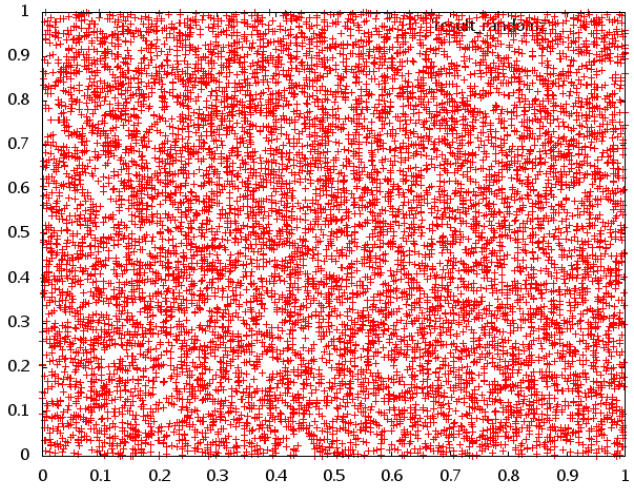
函数声明中的 double (*prand)(double) 是函数指针的声明方式, 这样将比如 randomz 作为参数传给 test_rand 就可以生成 randomz 的 (x_i, x_{i+1}) 对.

将屏幕上打印出的数据重定向到文件, 就可以用 gnuplot 做图, 比如

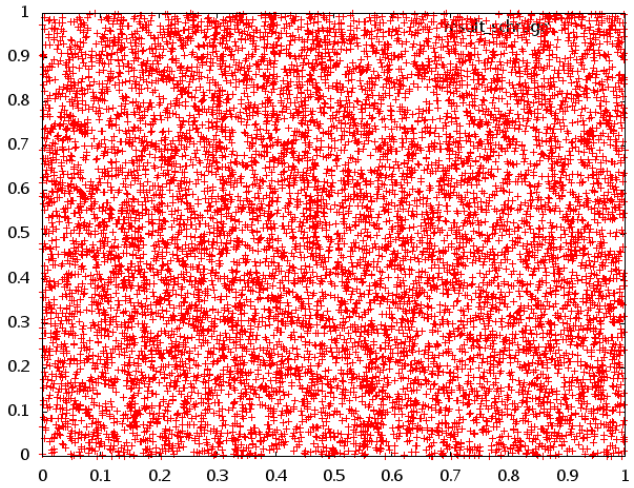
```
./generate_x_y_dataset 10000 > fout
gnuplot -e "plot 'fout'; pause 10"
```

见 Makefile 文件.

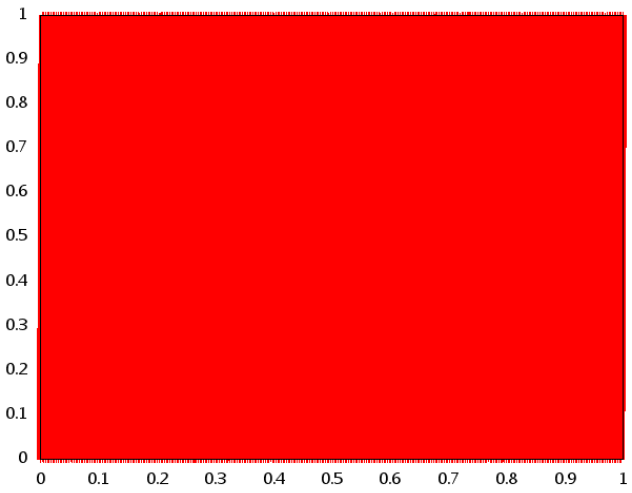
首先取数据点个数为 10000, 对 randomz 例程, 结果为:



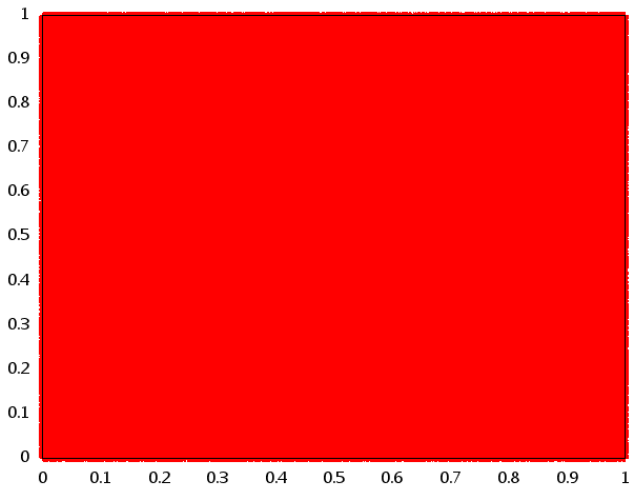
对 `schrage` 方法, 结果为:



然后将数据点数增加到 1000000, 对 `randomz` 例程, 结果为:


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

对 schrage 方法, 结果为:



可以看出对两种算法, x_i 和 x_{i+1} 的关联性都很弱.

3 用 $\langle x^k \rangle$ 测试均匀性

按照书上的公式

$$\langle x_n^k \rangle = \sum_{n=1} \frac{x_n^k}{N} \tag{1}$$

编写计算 $\langle x^k \rangle$ 的程序, 主要代码为:

```
double test_x_k(int k, int nsteps, double (*prand)(double))
{
    ...
    accum_x_k = 0.0;
    for (i = 0; i < nsteps; i++) {
        x = prand(x);     /* generate random number */
        x_k = 1.0;        /* compute $x_i^k$ */
        for (j = 0; j < k; j++) {
            x_k *= x;
        }
        accum_x_k += x_k; /* compute $\sum_i x_i^k$ */
    }
```

```
    return accum_x_k / nsteps;
}
```

详见 `test_x_k_c_l.c`. 注意这个程序每次运行结果都不同.

对 `randomz` 例程, 运行结果为:

Steps	$\lvert\langle x\rangle - \frac{1}{2}\rvert$	$\lvert\langle x^2\rangle - \frac{1}{3}\rvert$	$\lvert\langle x^3\rangle - \frac{1}{4}\rvert$	$\lvert\langle x^4\rangle - \frac{1}{5}\rvert$
10	0.0318369	0.0061295	0.0130491	0.0265782
100	0.0134899	0.0252084	0.0292341	0.0298540
1000	0.0034047	0.0023630	0.0018616	0.0015572
10000	0.0036694	0.0035312	0.0029775	0.0024888
100000	0.0000882	0.0001825	0.0002090	0.0002105
1000000	0.0001414	0.0001890	0.0001798	0.0001556
10000000	0.0001381	0.0001328	0.0000020	0.0000011

对 `rand_schrage` 例程, 运行结果为:

Steps	$\lvert\langle x\rangle - \frac{1}{2}\rvert$	$\lvert\langle x^2\rangle - \frac{1}{3}\rvert$	$\lvert\langle x^3\rangle - \frac{1}{4}\rvert$	$\lvert\langle x^4\rangle - \frac{1}{5}\rvert$
10	0.2139290	0.2298819	0.2203814	0.2058830
100	0.0203078	0.0210274	0.0203174	0.0185691
1000	0.0163075	0.0170248	0.0146589	0.0121269
10000	0.0022703	0.0012091	0.0002325	0.0004635
100000	0.0002978	0.0002813	0.0005853	0.0007172
1000000	0.0003324	0.0003617	0.0003349	0.0002953
10000000	0.0000166	0.0000017	0.0000063	0.0000106

分析: 注意到对 $[0,1]$ 内的均匀分布, 有 $\langle x^k\rangle = \frac{1}{k+1}$, 这样 $|\langle x^k\rangle - \frac{1}{k+1}| \to 0$, 当 $n \to \infty$. 这与程序得出的结果一致, 反映在表上竖着看每一列的值都在减小. 另外注意到每一行也有减小的规律, 这表明可能有 $\langle x^k\rangle - \frac{1}{k+1}| \to 0$ 当 $k \to \infty$. 但我没法证明这一点.

4 计算 $C(l)$ 检验独立性

按照书上的公式

$$C(l) = \frac{\langle x_n x_{n+1}\rangle - \langle x_n\rangle^2}{\langle x_n^2\rangle - \langle x_n\rangle^2} \tag{2}$$

计算 $C(l)$, 主要代码如下:

```
double test_c_l(int l, int nsteps, double (*prand)(double))
{
    ...
    accum_x_nl = 0.0;
    accum_x_n = 0.0;
    accum_x_n2 = 0.0;
    for (i = 0; i < nsteps; i++) {
        accum_x_nl += x * x_l;
        accum_x_n += x;
        accum_x_n2 += x*x;
        x = prand(x);
        x_l = prand(x_l);
    }
    return (accum_x_nl - accum_x_n * accum_x_n / nsteps)
           / (accum_x_n2 - accum_x_n * accum_x_n / nsteps);
}
```

详见 `test_x_k_c_l.c`.

对 `randomz` 例程, 运行结果为:

Steps	$\|C(1)\|$	$\|C(2)\|$	$\|C(3)\|$	$\|C(4)\|$
10	0.3865996	0.3304132	0.9369573	0.9437896
100	0.0363420	0.0575383	0.0381607	0.0103210
1000	0.0283273	0.0420643	0.0386733	0.0087944
10000	0.0092249	0.0157128	0.0098716	0.0074826
100000	0.0003122	0.0051709	0.0038947	0.0014403
1000000	0.0003657	0.0000330	0.0010533	0.0009559
10000000	0.0000487	0.0000949	0.0003670	0.0000421

对 `rand_schrage` 例程, 运行结果为:

Steps	$\|C(1)\|$	$\|C(2)\|$	$\|C(3)\|$	$\|C(4)\|$
10	0.3489163	0.5155943	0.4333424	0.0781640
100	0.1601396	0.0211235	0.1586193	0.1272870
1000	0.0366447	0.0175990	0.0360489	0.0153560
10000	0.0188222	0.0077633	0.0109835	0.0055603
100000	0.0010679	0.0005384	0.0017460	0.0030404
1000000	0.0008213	0.0000468	0.0009403	0.0017251
10000000	0.0004282	0.0002680	0.0000584	0.0000205

分析: 假设 x_n 和 x_{n+l} 独立, 这样 $\langle x_n x_{n+l}\rangle = \langle x_n\rangle\langle x_{n+l}\rangle = \langle x_n\rangle^2$, 这样 $C(l) = 0$. 程序的输出反映了这样的特性, 即随着 N 增大, 计算出的 $C(l)$ 减小, 直至趋于 0.

5 比较两种算法的优劣

从计算结果可以看出, 无论是独立性, 均匀性还是相邻两点的关联性, 两种算法都没有明显的区别.