标准库 rand()函数的缺陷以及 Blitz++随机数生成的简介

（newsuppy，转载请注明出处）

当我们需要在某个任务中使用随机数，通常我们习惯于使用标准库的 rand 函数。像这样：srand(time(0)); // 时间种子

rand() % MAX_RAND;

标准库的 rand 函数使用线性同余算法，是生成速度相当快的一种随机数生成算法。在多数情况下也确实能满足我们的要求，但是对于一些特殊目的应用这个算法生成的随机数是不行的，比如某些加密算法，蒙特卡罗积分等（在.NET 中创建随机密码的加密安全随机数就不能使用 Random 类的线性同余随机数，而要使用 System.Security.Cryptography 命名空间中的相关随机数生成类）。

这个线性同余算法的实现可以在很多书籍中找到。下面我给出一个 The C Programming Langurage 2nd 中的一个实现，这也是普遍使用的标准库随机数算法的实现：

```
unsigned long int next = 1;



/* rand:  return pseudo-random integer on 0..32767 */

int rand(void)

{

   next = next * 1103515245 + 12345;

   return (unsigned int)(next/65536) % 32768;

}



/* srand:  set seed for rand() */

void srand(unsigned int seed)

{

   next = seed;
```

```
}
```

这个实现的问题在于 rand 函数 return 行中的那个 32768，在标准库中这个数字定义为 RAND_MAX 宏，在 VisualC++和 Mingw32 编译器的 stdlib.h 头文件（或者 cstdlib)中你都可以发现 RAND_MAX 的值为 32768。也就是说这个算法的随机数分布在 0--RAND_MAX 中，而在一般编译器中就是 0--32768。假设你的算法需要的是 300000 多个的随机数，那么使用 rand 函数会产生重负次数近 30 次的随机数!

所以在这里我将简单介绍一下如何使用 Blitz++库中的随机数生成类。不过在这里我不能够证明 Blitz++随机数算法相对于标准库有什么优越性。Blitz++的源代码是开放的，你可以完全了解它的随机数算法的实现。

在 Blitz++中随机数类组织在 ranlib namespace 中，使用方法也非常简单，seed 成员函数种入种子后，再用 random 成员函数就可以了。Blitz++中包括了产生各种概率分布情况的随机数,列举如下:均匀分布(Uniform),正态分布(Normal),指数分布(Exponential),Beta 分布，Gamma 分布， X 方分布，F 分布，离散均匀分布。具体地可以参考 Blitz++文档的第九章。本文将会演示正态分布的随机数类。

NormalUnit<>()　标准正态分布，μ = 0, σ = 1;
Normal<>(T mean, T standardDeviation) 正态分布，N(μ, σ^2)，其中 mean 就是 μ,standardDeviation 就是 σ。

Blitz++中随机数类的 seed 是共享的，调用一个类的 seed 后，其他类也同样种入了相同的种子。对于种子的来源，Blitz++文档中有这样一个注意事项:

Note: you may be tempted to seed the random number generator from a static initializer. **Don't do it!** Due to an oddity of C++, there is no guarantee on the order of static initialization when templates are involved. Hence, you may seed the RNG before its constructor is invoked, in which case your program will crash. If you don't know what a static initializer is, don't worry -- you're safe!

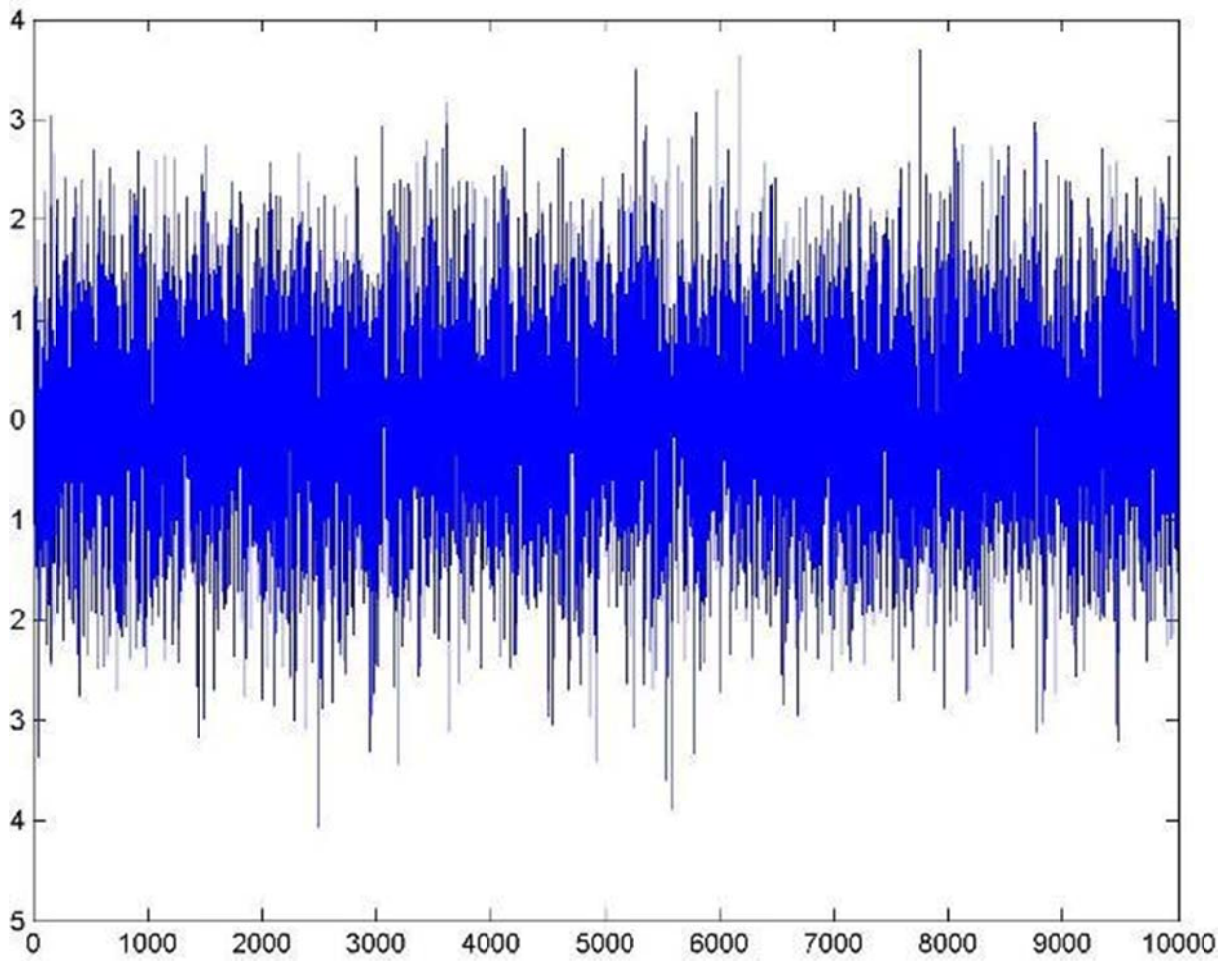
（幸运的是我也不太清楚 static initializer 具体指什么,:-) ）

1，先来看一个标准正态分布的例子，根据 3σ 法则(正态分布的随机数几乎全部落在(μ-3σ, μ+3σ)中),这些随机数应该大部分落在(-3, 3)之间。

下面的程序产生 10000 个正态分布随机数，然后导入 Matlab 生成图形，横坐标是随机数的序数，纵坐标是随机数值。很显然，大部分点在 3σ 区间内。在区间(μ-σ, μ+σ)中数据点最密。

```
#include <random/uniform.h>
```

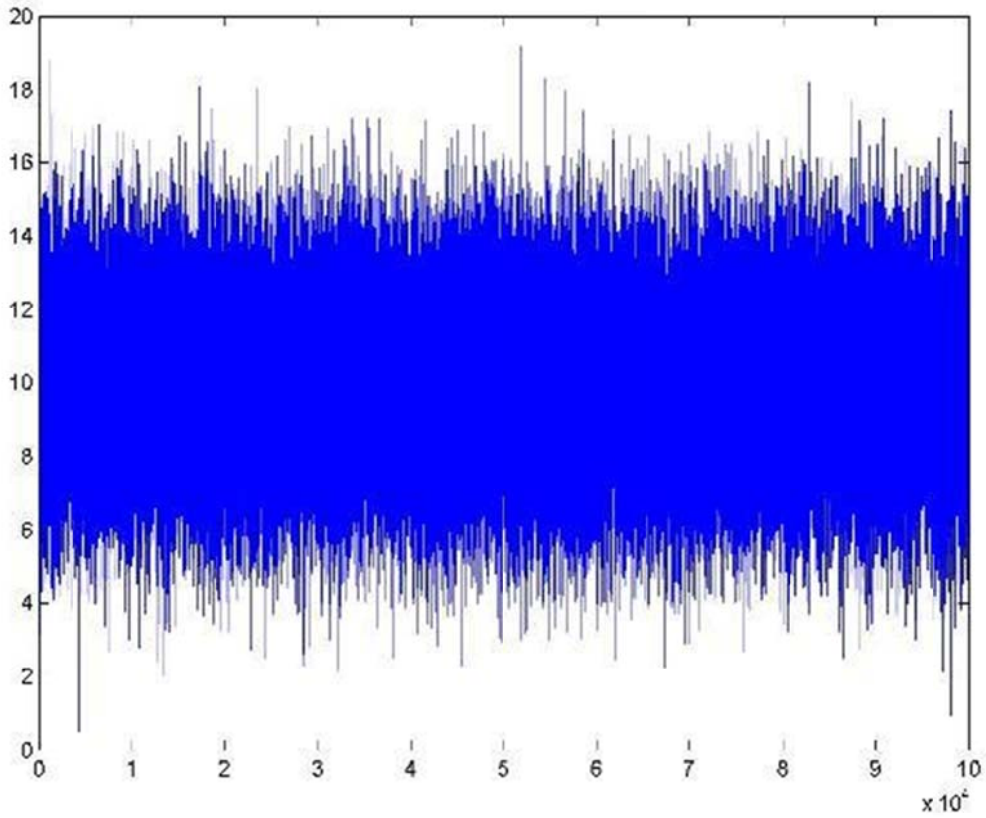
```
#include <random/normal.h>

#include <iostream>

#include <fstream>

#include <ctime>

#include <cstdlib>

using namespace std;

using namespace ranlib;



int main()

{

    const size_t MAXGEN = 10000;

    NormalUnit<float> rnd;



    rnd.seed(time(0));



    ofstream file("c:\\file.txt");



    // generator Normal distribution random number

    for (size_t i=0; i<MAXGEN; ++i)

    {

        file << rnd.random() << " ";

    }

}
```



2，下面是一个服从于 μ = 10， σ = 2 的正态分布，根据 3 σ 法则，大部分点应该落在(4, 16)之间。

```
#include <random/uniform.h>

#include <random/normal.h>

#include <iostream>

#include <fstream>

#include <ctime>

#include <cstdlib>

using namespace std;

using namespace ranlib;
```

```
int main()

{

    const size_t MAXGEN = 1000000;

    const int mu = 10;

    const int sigma = 2;

    Normal<float> rnd(mu,sigma);



    rnd.seed(time(0));



    ofstream file("c:\\file.txt");



    // generator Normal distribution random number

    for (size_t i=0; i<MAXGEN; ++i)

    {

        file << rnd.random() << " ";

    }

}
```



3，生成前述正态分布的钟型曲线（PDF）。

这里产生 1M 的 float 随机数，然后乘以 10 转型为整数，统计在区间 0-170 之间随机数出现的概率，这样再导入 Matlab 生成图形就是一个近似的正态分布的钟型曲线了。

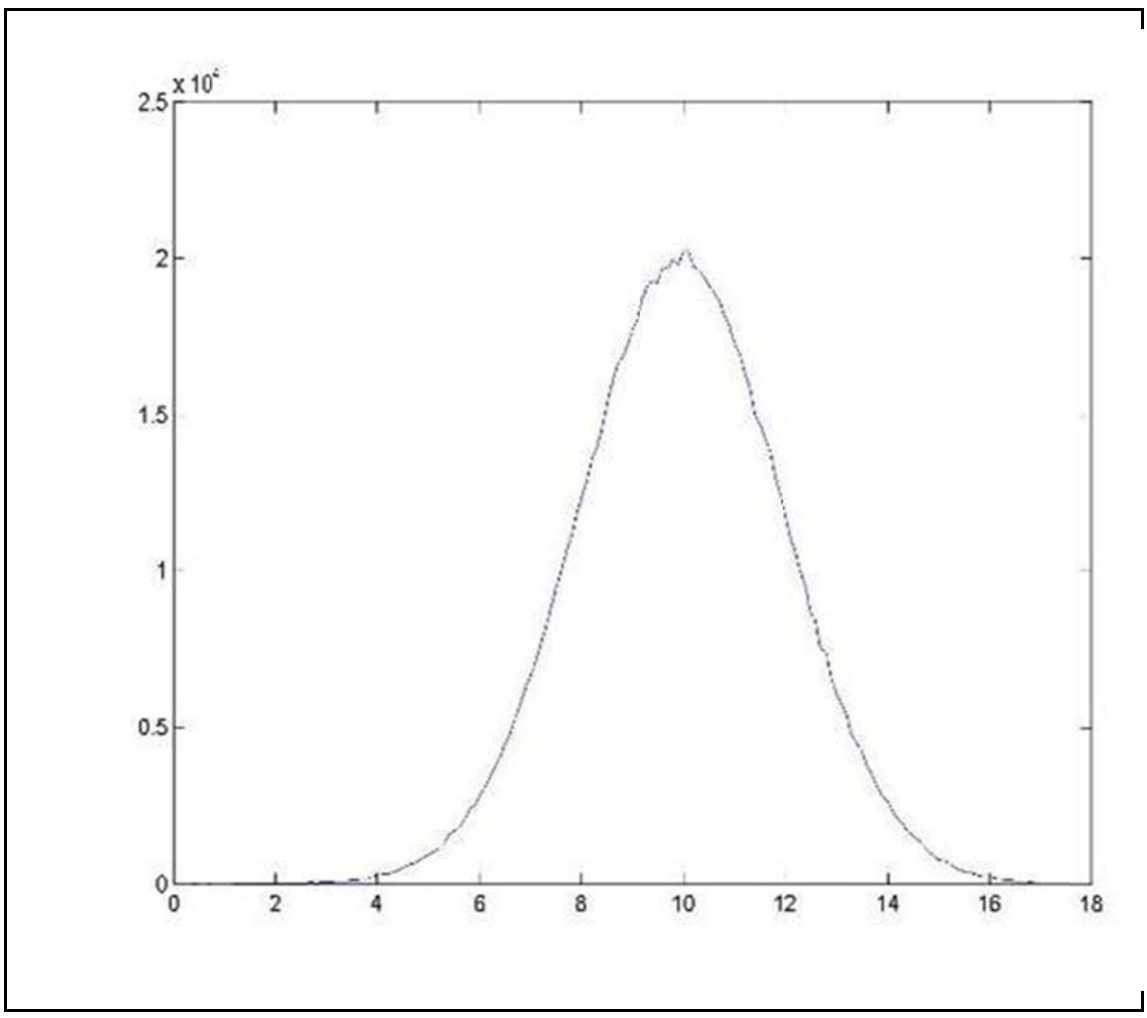
```
#include <random/uniform.h>

#include <random/normal.h>

#include <iostream>

#include <fstream>

#include <ctime>
```

```
#include <cstdlib>

using namespace std;

using namespace ranlib;



int main()

{

   const size_t MAXGEN = 1000000;

   const int mu = 10;

   const int sigma = 2;

   Normal<float> rnd(mu,sigma);



   rnd.seed(time(0));



   ofstream file("c:\\file.txt");



   float *rndArray = new float[MAXGEN];

   // generator Normal distribution random number

   for (size_t i=0; i<MAXGEN; ++i)

   {

      rndArray[i] = rnd.random();

   }



   int *rndConvertIntArray = new int[MAXGEN];
```

```
int multiple = 10;

// convert float random number to integer

for (size_t i=0; i<MAXGEN; ++i)

{

   rndConvertIntArray[i] = int(rndArray[i] * multiple);

}



const size_t PDFSIZE = (mu + 3 * sigma) * multiple;

const size_t redundancy = 10;

int *pdf = new int[PDFSIZE+redundancy];

for (size_t i=0; i<PDFSIZE+redundancy; ++i)

{

   pdf[i] = 0;

}



// generator PDF(Probability Distribution Function), Normal distribution is a "bell" shape

for (size_t i=0; i<MAXGEN; ++i)

{

   if (rndConvertIntArray[i] <= PDFSIZE+redundancy-1)

   {

      ++pdf[rndConvertIntArray[i]];

   }

}
```

```
    for (size_t i=0; i<PDFSIZE+redundancy; ++i)
    {
       file << pdf[i]   << " ";
    }

    delete[] rndArray;
    delete[] rndConvertIntArray;
    delete[] pdf;
}
```

钟型曲线，当然不是特别光滑(☺),大部分随机数都出现在（4，16）区间内。



总结，Blitz++的随机数生成类应该说是值得信任的。

[参考文献]

1， 概率论与数理统计（3rd）， 浙江大学 盛骤 谢式千 潘承毅 编 高等教育出版社

2， C 数值算法（2nd）， [美] William H. Press, Saul A. Teukolsky, William T. Vetterling,

Brian P. Flannery 著

傅祖芸 赵梅娜 丁岩石 等译，傅祖芸 审校

3，Matlab 6.5 的文档 The MathWorks, Inc.