信息网络建模与仿真

实验 1 实验报告

班级：2019211123

姓名：李雨霖

学号：2019210195

2021 年 10 月

1. 实验目标

1、深入理解均匀分布随机变量产生方法；

2、掌握由均匀分布随机变量产生其他分布随机变量的方法；

3、掌握两种常用的高斯分布随机变量的产生方法；

4、掌握泊松过程的产生方法；

2. 实验内容

1、均匀分布随机数的产生

（1） 利用 $x_{i+1}=[16807x_i]\bmod(2147483647)$ 递推关系，编写程序，用线性同余法生成(0, 1)区间上的随机数，计算其期望、方差和概率密度与理论值的吻合程度。

（2） 分析 Mersenne Twister 产生均匀分布随机数的代码，并用该代码产生一组(0,1)区间上的随机数，计算其期望、方差和概率密度与理论值的吻合程度。

（3） 编写程序，实现 NS-3 中采用的 MRG32k3a 算法生成随机数，并计算其期望、方差和概率密度与理论值的吻合程度。

2、互联网中流量分析建模中经常会用到 Pareto 分布，其概率密度函数为：

$$p(x)=\frac{ab^a}{x^{a+1}},\quad b\le x<\infty$$

以 Mersenne Twister 随机数产生器为基础产生服从 Pareto 分布的随机变量，数据导入到 Matlab 中画出其概率密度函数分布图和累积分布图。（自行尝试 a 和 b 的不同取值）

3、用中心极限定理方法和 Box-Muller 算法分别产生标准高斯分布随机数，在 Matlab 中观察数据分布。

4、用组合法产生概率密度函数为 $f(x)=\sum_{i=1}^{3}p_i\frac{1}{b_i}\varphi\left(\frac{x-a_i}{b_i}\right)$ 的混合高斯随机变量，画出其数据分布图。其中：$p_1=1/2$，$p_2=1/3$，$p_3=1/6$，$a_1=-1$，$a_2=0$，$a_3=1$，$b_1=1/4$，$b_2=1$，$b_3=1/2$。

5、产生一个参数 λ 等于 2 的泊松过程，存储事件发生的时间，统计单位时间内事件发生的次数，判断其与泊松分布是否吻合。

3. 实验步骤与结果

3.1 均匀分布随机数的产生

3.1.1 线性同余法产生均匀分布随机数

（一）算法分析

线性同余法通过以下递归式产生均匀分布的随机序列$\{X\}$：

$$X_{n+1} = (aX_n + b) mod\, m$$

其基本思想是通过对前一个数进行线性运算并取模从而得到下一个数。

在本实验中，$a = 16807$，$b = 0$，$m = 2147483647$

（二）程序设计

使用 `int *LinerCongruential(int seed)`函数生成随机数。

在 `void Test_1_1()`函数计算均值和方差，并将生成的数据写入 `data.txt`。

在 `test1_1.m` 中，读取 `data.txt`，使用 `ksdensity` 核心平滑密度估计算法绘制出概率密度函数，与均匀分布的概率密度函数做对比。

(1) 头文件 LinearCongruential.h:

```
#define LCGLength 100000

///\brief 线性同余法
///\param seed 种子
///\return 随机数数组
int *LinerCongruential(int seed);
```

(2)源文件 LinearCongruential.cpp:

```
#include"LinearCongruential.h"

int *LinerCongruential(int seed)
{
    int a = 16807;
    int b = 2147483647;
    int *RandomNumber= new int[LCGLength];
    RandomNumber[0] = seed;
    for (int i = 1; i < LCGLength; i++)
    {
        RandomNumber[i] = (RandomNumber[i - 1] * a) % b;
    }
```

```
    return RandomNumber;
}
```

(3)C++验证 main.cpp> Test_1_1():

```
#pragma warning(disable:6262)

void Test_1_1()
{
    int *RamdonNumber;
    double Rand[LCGLength];
    double expect = 0;//期望
    double variance = 0;
    RamdonNumber = LinerCongruential((int)time(NULL));
    cout << "计算中" << endl;
    fstream data;
    data.open("data.txt", ios::out | ios::trunc);
    for(int i=0;i< LCGLength;i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2. + 0.5;
        data << Rand[i] << endl;
        expect += Rand[i];
    }
    cout << endl;
    data.close();
    delete[] RamdonNumber;
    expect /= LCGLength;
    cout << "期望: " << expect << endl;
    for (int i = 0; i < LCGLength; i++)
    {
        variance += pow(Rand[i] - expect, 2);
    }
    variance /= LCGLength;
    cout << "方差: " << variance << endl;
    cout << "概率密度与理论值的吻合程度在 matlab 验证\n";
}
```

(4)Matlab 验证 test1_1.m

```
x=1/100:1/100:1;
y=ones(1,100);

plot(x,y);
hold on
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");
%disp(data);
```

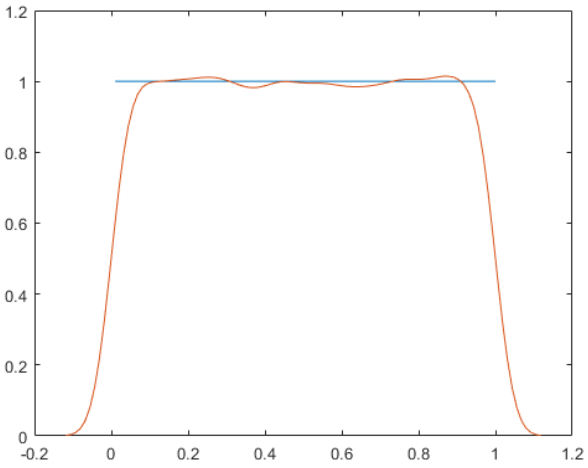
```
[f,xi]=ksdensity(data);
plot(xi,f)
```

（三）测试结果

```
计算中

期望：0.500047
方差：0.0837198
概率密度与理论值的吻合程度在matlab验证
```

期望的理论值：0.5

方差的理论值：0.8333333



由图片可以看出，概率密度与理论值的吻合程度较高。

3.1.2 Mersenne Twister 方法产生均匀分布随机数

（一）算法分析

- 算法中用到的变量如下所示：

·w：长度（以 bit 为单位）

·n：递归长度

·m：周期参数，用作第三阶段的偏移量

·r：低位掩码/低位要提取的位数

·a：旋转矩阵的参数

·f：初始化梅森旋转链所需参数

·b,c：TGFSR 的掩码

·s,t：TGFSR 的位移量

·u,d,l：额外梅森旋转所需的掩码和位移量

- MT19937-32 的参数列表如下：

·(w, n, m, r) = (32, 624, 397, 31)

·a = 9908B0DF（16）

·f = 1812433253

·(u, d) = (11, FFFFFFFF16)

·(s, b) = (7, 9D2C568016)

·(t, c) = (15, EFC6000016)

·l = 18

- 整个算法分为三个阶段：

(1)初始化，获得基础的梅森旋转链；

首先将传入的 seed 赋给$MT[0]$作为初值，然后根据递推式：

$$MT[i] = f \times (MT[i-1] \oplus (MT[i-1] >> (w-2))) + i$$

递推求出梅森旋转链。

(2)对于旋转链进行旋转算法；

遍历旋转链，对每个$MT[i]$，根据递推式：

$$MT[i] = MT[i+m] \oplus ((upper\_mask(MT[i]) \,||\, lower\_mask(MT[i+1]))A)$$

进行旋转链处理。

其中，“||”代表连接的意思，即组合$MT[i]$的高 $w-r$ 位和$MT[i+1]$的低 r 位，设组合后的数字为x，则xA的运算规则为（x_0是最低位）：

$$xA = \begin{cases} x >> 1 & x_0 = 0 \\ (x >> 1) \oplus a & x_0 = 1 \end{cases}$$

(3)对于旋转算法所得的结果进行处理；

设x是当前序列的下一个值，y是一个临时中间变量，z是算法的返回值。则处理过程如下：

$$y := x \oplus ((x >> u) \,\&\, d)$$

$$y := y \oplus ((y << s) \,\&\, b)$$

$$y := y \oplus ((y << t) \,\&\, c)$$

$$z := y \oplus (y >> l)$$

（二）程序代码分析

使用 uint32_t *MersenneTwister(uint32_t seed)函数生成随机数。

在 void Test_1_2()函数计算均值和方差，并将生成的数据写入 data.txt。

在 test1_1.m 中，读取 data.txt，使用 ksdensity 核心平滑密度估计算法绘制出概率密度函数，与均匀分布的概率密度函数做对比。

(1)头文件 MersenneTwister.h:

```
#include <stdint.h>
#define MTLength 100000
#pragma warning(disable:6385)

///\brief 梅森旋转法
///\param seed 种子
///\return 随机数数组
uint32_t *MersenneTwister(uint32_t seed);
```

(2)源文件 LinearCongruential.cpp:

```
#include "MersenneTwister.h"

// 定义 MT19937-32 的常数
enum
{
    N = 624,
    M = 397,
    R = 31,
    A = 0x9908B0DF,
    F = 1812433253,
    U = 11,
    S = 7,
    B = 0x9D2C5680,
    T = 15,
    C = 0xEFC60000,
    L = 18,
    MASK_LOWER = (1ull << R) - 1,
    MASK_UPPER = (1ull << R)
};

static uint32_t  mt[N];
static uint16_t  index;

// 根据给定的 seed 初始化旋转链
void Initialize(uint32_t  seed)
{
```

```
    uint32_t  i;
    mt[0] = seed;
    for (i = 1; i < N; i++)
    {
        mt[i] = (F * (mt[i - 1] ^ (mt[i - 1] >> 30)) + i);
    }
    index = N;
}

static void Twist()
{
    uint32_t  i, x, xA;
    for (i = 0; i < N; i++)
    {
        x = (mt[i] & MASK_UPPER) + (mt[(i + 1) % N] & MASK_LOWER);
        xA = x >> 1;
        if (x & 0x1)
        {
            xA ^= A;
        }
        mt[i] = mt[(i + M) % N] ^ xA;
    }

    index = 0;
}

// 产生一个 32 位随机数
uint32_t ExtractU32()
{
    uint32_t  y;
    int  i = index;
    if (index >= N)
    {
        Twist();
        i = index;
    }
    y = mt[i];
    index = i + 1;
    y ^= (y >> U);
    y ^= (y << S) & B;
    y ^= (y << T) & C;
    y ^= (y >> L);
    return y;
}
```

```
uint32_t *MersenneTwister(uint32_t seed)
{
    Initialize(seed);
    uint32_t* RandomNumber = new uint32_t[MTLength];
    for (int i = 0; i < MTLength; i++)
    {
        Twist();
        RandomNumber[i] = ExtractU32();
    }
    return RandomNumber;
}
```

(3)C++验证 main.cpp> Test_1_2():

```
void Test_1_2()
{
    uint32_t* RamdonNumber;
    double Rand[MTLength];
    double expect = 0;//期望
    double variance = 0;
    RamdonNumber = MersenneTwister((uint32_t)time(NULL));
    cout << "计算中" << endl;
    fstream data;
    data.open("data.txt", ios::out | ios::trunc);
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
        data << Rand[i] << endl;
        expect += Rand[i];
    }
    cout << endl;
    data.close();
    delete[] RamdonNumber;
    expect /= MTLength;
    cout << "期望: " << expect << endl;
    for (int i = 0; i < LCGLength; i++)
    {
        variance += pow(Rand[i] - expect, 2);
    }
    variance /= MTLength;
    cout << "方差: " << variance << endl;
    cout << "概率密度与理论值的吻合程度在 matlab 验证\n";
}
```

(4)Matlab 验证 test1_1.m

```
x=1/100:1/100:1;
```

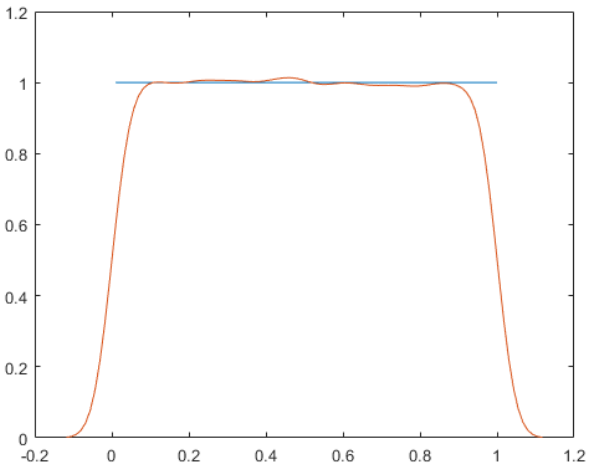
```
y=ones(1,100);

plot(x,y);
hold on
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");
%disp(data);

[f,xi]=ksdensity(data);
plot(xi,f)
```

（三）测试结果与分析

```
计算中

期望：0.499306
方差：0.0832515
概率密度与理论值的吻合程度在matlab验证
```

期望的理论值：0.5

方差的理论值：0.8333333



由图片可以看出，概率密度与理论值的吻合程度较高。

3.1.3 MRG32k3a 方法产生均匀分布随机数

（一）算法分析

MRG32k3a 方法有两个分量，每个分量都是 3 阶。在第 n 步，它的状态是一对向量 $s_{1,n} = (x_{1,n}, x_{1,n+1}, x_{1,n+2})$和$s_{2,n} = (x_{2,n}, x_{2,n+1}, x_{2,n+2})$，它们根据线性递归进化

$$x_n = (1403580 \times x_{n-2} - 810728 \times x_{n-3}) \mathrm{mod}(2^{32} - 209)$$

$$y_n = (527612 \times y_{n-1} - 1370589 \times y_{n-3}) \mathrm{mod}(2^{32} - 22853)$$

输出u_n定义为

$$z_n = (x_n - y_n)\mathrm{mod}(4294967087)$$

$$u_n = \begin{cases} z_n/4294967088 & z_n > 0 \\ 4294967087/4294967088 & z_n = 0 \end{cases}$$

这个随机数发生器的优点是算法相对简单，周期长达2^{191}， 并可以很容易地切分成长度为2^{127}的不同段，每一段的种子容易确定，这样，在并行计算时，每个工人节点可以从不同段的种子出发进行模拟。

将初始种子s_0设置为向量种子中的六个整数。种子中的前三个整数必须都小于$2^{32} - 209 = 4294967087$，并且不能全为 0；并且最后三个整数必须都小于$2^{32} - 22853 = 4294944443$，并且不能全为 0。

（二）程序代码分析

使用 double* MRG32k3a (uint32_t seed)函数生成随机数。

在 void Test_1_3()函数计算均值和方差，并将生成的数据写入 data.txt。

在 test1_1.m 中，读取 data.txt，使用 ksdensity 核心平滑密度估计算法绘制出概率密度函数，与均匀分布的概率密度函数做对比。

(1)头文件 MRG32k3a.h:

```
#include <stdint.h>
#define CMRGLength 100000

///\brief L'Ecuyer-CMRG
///\param seed 种子
///\return 随机数数组
double* MRG32k3a(uint32_t *seed);
```

(2)源文件 MRG32k3a.cpp:

```
#include"MRG32k3a.h"

static uint32_t  x[3];
static uint32_t  y[3];

void Initialize(uint32_t  *seed)
{
    for (int i = 0; i < 3; i++)
    {
        x[i] = seed[i];
        y[i] = seed[i + 3];
    }
}
```

```
double ExtractU32()
{
    uint32_t thisX = ((long long)1403580 * x[1] - (long long)810728
* x[0]) % 4294967087;
    uint32_t thisY = ((long long)527612 * y[2] - (long long)1370589
* y[0]) % 4294944443;
    for (int i = 0; i < 2; i++)
    {
        x[i] = x[i + 1];
        y[i] = y[i + 1];
    }
    x[2] = thisX;
    y[2] = thisY;
    uint32_t z = (thisX - thisY) % 4294967087;
    if (z > 0)
    {
        return (double)z / 4294967088.;
    }
    else
    {
        return 4294967087. / 4294967088.;
    }
}

double * MRG32k3a(uint32_t* seed)
{
    Initialize(seed);
    double* RandomNumber = new double[CMRGLength];
    for (int i = 0; i < CMRGLength; i++)
    {
        RandomNumber[i] = ExtractU32();
    }
    return RandomNumber;
}
```

(3)C++验证 main.cpp> Test_1_3():

```
void Test_1_3()
{
    double* RamdonNumber;
    double Rand[CMRGLength];
    double expect = 0;//期望
    double variance = 0;
    uint32_t seed[6] = { 12345,12345,12345,12345,12345,12345 };
    RamdonNumber = MRG32k3a(seed);
    cout << "计算中" << endl;
```

```
    fstream data;
    data.open("data.txt", ios::out | ios::trunc);
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i];
        data << Rand[i] << endl;
        expect += Rand[i];
    }
    cout << endl;
    data.close();
    delete[] RamdonNumber;
    expect /= CMRGLength;
    cout << "期望: " << expect << endl;
    for (int i = 0; i < LCGLength; i++)
    {
        variance += pow(Rand[i] - expect, 2);
    }
    variance /= CMRGLength;
    cout << "方差: " << variance << endl;
    cout << "概率密度与理论值的吻合程度在 matlab 验证\n";
}
```

(4)Matlab 验证 test1_1.m

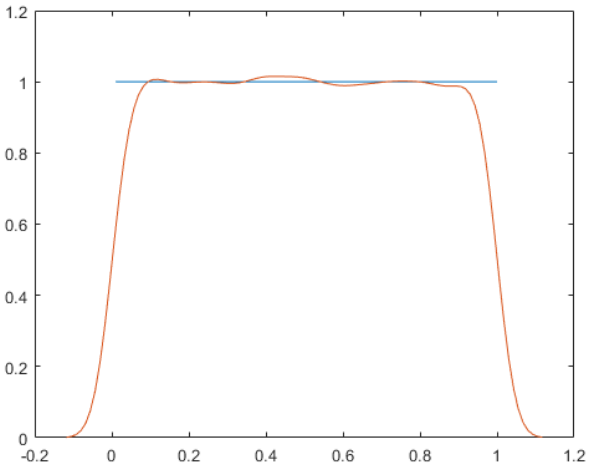
```
x=1/100:1/100:1;
y=ones(1,100);

plot(x,y);
hold on
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");
%disp(data);

[f,xi]=ksdensity(data);
plot(xi,f)
```

（三）测试结果与分析

```
计算中

期望: 0.500195
方差: 0.0831628
概率密度与理论值的吻合程度在matlab验证
```

期望的理论值：0.5

方差的理论值：0.8333333



由图片可以看出，概率密度与理论值的吻合程度较高。

3.2 Pareto 分布随机变量的产生

（一） 算法分析

Pareto 分布概率密度函数为：

$$p(x)=\begin{cases}0, & x<b\\ \dfrac{ab^a}{x^{a+1}}, & x\geq b\end{cases}$$

Pareto 分布的分布函数为：

$$F(x)=\begin{cases}0, & x<b\\ 1-\left(\dfrac{b}{x}\right)^a, & x\geq b\end{cases}$$

Pareto 分布的生成函数为：

$$F^{-1}(x)=b\times(1-x)^{-\frac{1}{a}}\quad 0\leq x\leq 1$$

即x服从(0, 1)区间上的均匀分布

Pareto 分布的数学期望为：

$$E(X)=\frac{a\times b}{a-1}\quad (a>1)$$

Pareto 分布的方差为：

$$D(X)=\frac{a\times b^2}{(a-1)^2\times(a-2)}\quad (a>2)$$

（二） 程序代码分析

（1）C++生成 Pareto 分布> Pareto():

```
void Pareto()
{
```

```
    int times = 5;
    double a[5] = {4,5,6,4,4};
    double b[5] = {4,4,4,5,6};

    cout << "计算中" << endl;
    for (int j = 0; j < times; j++)
    {
        cout << "第" << j + 1 << "次计算,a=" << a[j] << ",b=" << b[j]
<< endl;
        uint32_t* RamdonNumber;
        double Rand[MTLength];
        double expect = 0;//期望
        double variance = 0;
        RamdonNumber = MersenneTwister((uint32_t)time(NULL));

        fstream data;
        data.open("Pareto"+ to_string(j)+".txt", ios::out |
ios::trunc);
        for (int i = 0; i < MTLength; i++)
        {
            Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
            Rand[i] = b[j] * pow((1 - Rand[i]), (-1.0) / a[j]);
            data << Rand[i] << endl;
            expect += Rand[i];
        }
        data.close();
        delete[] RamdonNumber;
        expect /= MTLength;
        if(a[j]>1)
            cout << "期望: " << expect << endl;
        for (int i = 0; i < MTLength; i++)
        {
            variance += pow(Rand[i] - expect, 2);
        }
        variance /= MTLength;
        if (a[j] > 2)
            cout << "方差: " << variance << endl;
    }
    cout << endl << "计算完成";

}
```

(2)matlab 画概率密度函数和分布函数

```
clear;
clc;
```

```
close all;

path="C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\";

data1=load(path+"Pareto0.txt");
data1(data1>20) = [];
[f1,x1]=ksdensity(data1);
data2=load(path+"Pareto1.txt");
data2(data2>20) = [];
[f2,x2]=ksdensity(data2);
data3=load(path+"Pareto2.txt");
data3(data3>20) = [];
[f3,x3]=ksdensity(data3);
data4=load(path+"Pareto3.txt");
data4(data4>30) = [];
[f4,x4]=ksdensity(data4);
data5=load(path+"Pareto4.txt");
data5(data5>30) = [];
[f5,x5]=ksdensity(data5);

figure(1)
subplot(2,3,1);
plot(x1,f1);
xlim([0 20]);ylim([0 1.2]);title("a=4,b=4");

subplot(2,3,2);
plot(x2,f2);
xlim([0 20]);ylim([0 1.2]);title("a=5,b=4");

subplot(2,3,3);
plot(x3,f3);
xlim([0 20]);ylim([0 1.2]);title("a=6,b=4");

subplot(2,3,4);
plot(x1,f1);
xlim([0 30]);ylim([0 0.8]);title("a=4,b=4");

subplot(2,3,5);
plot(x4,f4);
xlim([0 30]);ylim([0 0.8]);title("a=4,b=5");

subplot(2,3,6);
plot(x5,f5);
xlim([0 30]);ylim([0 0.8]);title("a=4,b=6");
```

```
figure(2)
subplot(2,3,1);cdfplot(data1);xlim([0 20])
subplot(2,3,2);cdfplot(data2);xlim([0 20])
subplot(2,3,3);cdfplot(data3);xlim([0 20])
subplot(2,3,4);cdfplot(data1);xlim([0 20])
subplot(2,3,5);cdfplot(data4);xlim([0 20])
subplot(2,3,6);cdfplot(data5);xlim([0 20])
```

（三） 测试结果与分析

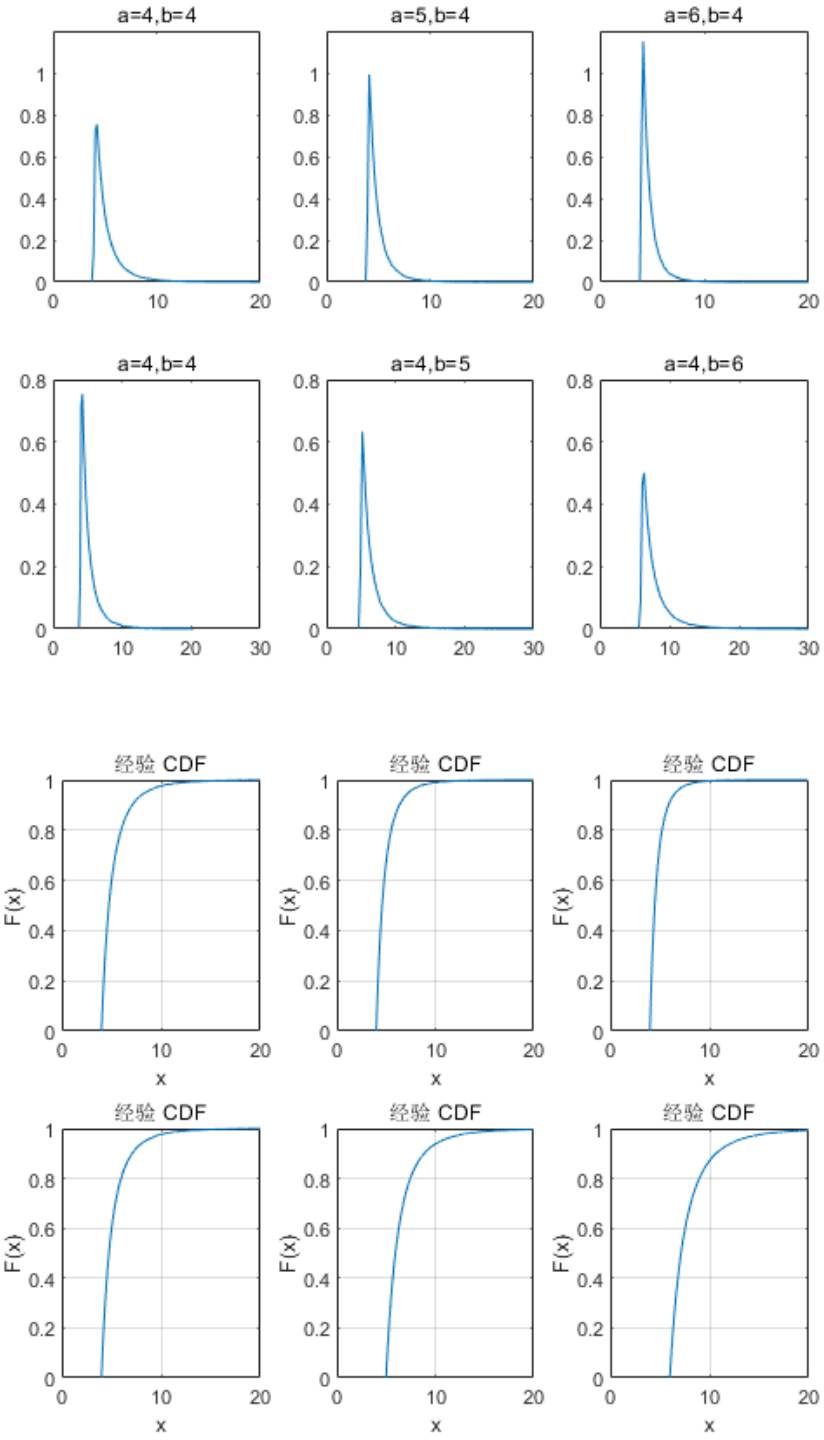
```
计算中
第1次计算,a=4,b=4
期望：5.33685
方差：3.73298
第2次计算,a=5,b=4
期望：5.00531
方差：1.66392
第3次计算,a=6,b=4
期望：4.79531
方差：0.935589
第4次计算,a=4,b=5
期望：6.67152
方差：5.70146
第5次计算,a=4,b=6
期望：8.02619
方差：8.84477

计算完成
```

期望和方差的理论值如下表：

	期望	方差
a=4,b=4	5.3333	3.5556
a=4,b=5	5.0000	1.6667
a=6,b=4	4.8000	0.9600
a=4,b=5	6.6667	5.5556
a=4,b=6	8.0000	8.0000

由表可知，生成的 Pareto 分布在误差允许范围内。

画出的 PDF 图和 CDF 图如下。



3. 3 高斯分布随机变量的产生

3.3.1　中心极限定理法产生高斯分布随机变量

（一）算法分析

中心极限定理：

设随机变量序列$\{X_i\}$相互独立，具有相同的期望和方差，即

$$E(X_i) = \mu, D(X_i) = \sigma^2。$$

令

$$Y_n = X_1 + \cdots + X_n$$

$$Z_n = \frac{Y_n - E(Y_n)}{\sqrt{D(Y_n)}} = \frac{Y_n - n\mu}{\sqrt{n}\sigma}$$

则

$$Z_n \to N(0,1)$$

（二）程序代码分析

(1)C++生成标准正态分布> CenterLimitTheorem():

```
void CenterLimitTheorem()
{
    cout << "计算中" << endl;
    uint32_t* RamdonNumber;
    double Rand[MTLength];
    RamdonNumber = MersenneTwister((uint32_t)time(NULL));
    fstream data;
    double expect = 0.5;
    double variance = 1. / 12;
    data.open("data.txt", ios::out | ios::trunc);
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
    }
    for (int i = 0; i < 10000; i++)
    {
        double sum = 0;
        for (int j = 0; j < 10; j++)
        {
            sum += Rand[10 * i + j];
        }
        Rand[i] = (sum - 10. * expect) / (sqrt(10 * variance));
        data << (sum-10.* expect)/(sqrt(10* variance)) << endl;
    }
    expect = 0;
    variance = 0;
    for (int i = 0; i < 10000; i++)
    {
        expect += Rand[i];
    }
    expect /= 10000;
    cout << "期望: " << expect << endl;
```

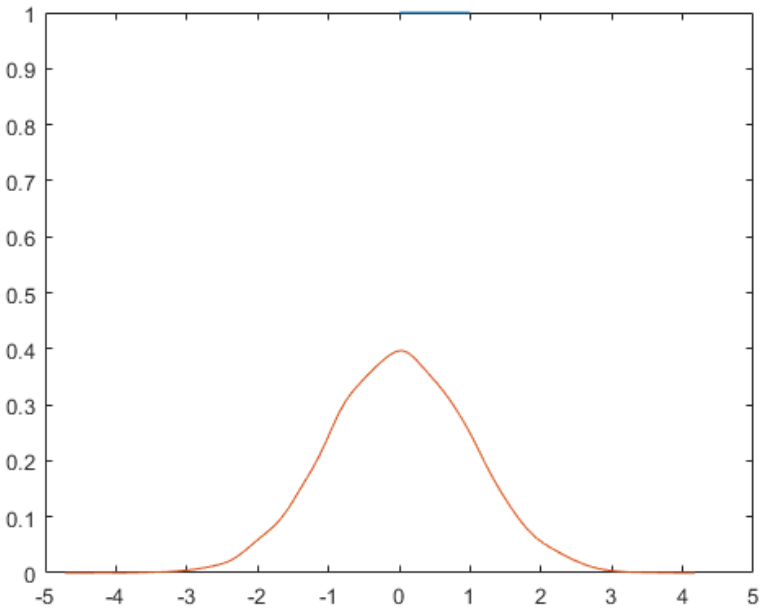
```
    for (int i = 0; i < 10000; i++)
    {
        variance += pow(Rand[i] - expect, 2);
    }
    variance /= 10000;
    cout << "方差: " << variance << endl;
    cout << endl << "计算完成";

}
```

(2)Matlab 验证 test1_1.m

```
x=1/100:1/100:1;
y=ones(1,100);

plot(x,y);
hold on
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");
%disp(data);

[f,xi]=ksdensity(data);
plot(xi,f)
```

（三）测试结果与分析

```
计算中
期望: 0.00115852
方差: 1.01676
```



3.3.2 Box-Muller 法产生高斯分布随机变量

（一）算法分析

Box-Muller 变换是通过服从均匀分布的随机变量，来构建服从高斯分布的随机变量的一种方法。

具体的描述为：选取两个服从[0,1]上均匀分布的随机变U_1、U_2，

若X、Y满足

$$X = cos(2\pi U_1)\sqrt{-2\ln U_2}$$

$$Y = sin(2\pi U_1)\sqrt{-2\ln U_2}$$

则 X 与 Y 服从均值为 0，方差为 1 的高斯分布。

（二）程序代码分析

(1)C++生成标准正态分布> BoxMuller():

```
void BoxMuller()
{
    cout << "计算中" << endl;
    uint32_t* RamdonNumber;
    double Rand[MTLength];
    RamdonNumber = MersenneTwister((uint32_t)time(NULL));
    fstream data;
    double expect = 0;
    double variance = 0;
    data.open("data.txt", ios::out | ios::trunc);
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
    }
    for (int i = 0; i < MTLength/2; i++)
    {

        Rand[i]= cos(2 * pi * Rand[2 * i + 1]) * sqrt(-2 *
log(Rand[2 * i]));
        data << Rand[i] << endl;
    }
    data.close();
    delete[] RamdonNumber;
    for (int i = 0; i < 50000; i++)
    {
        expect += Rand[i];
    }
    expect /= 50000;
```

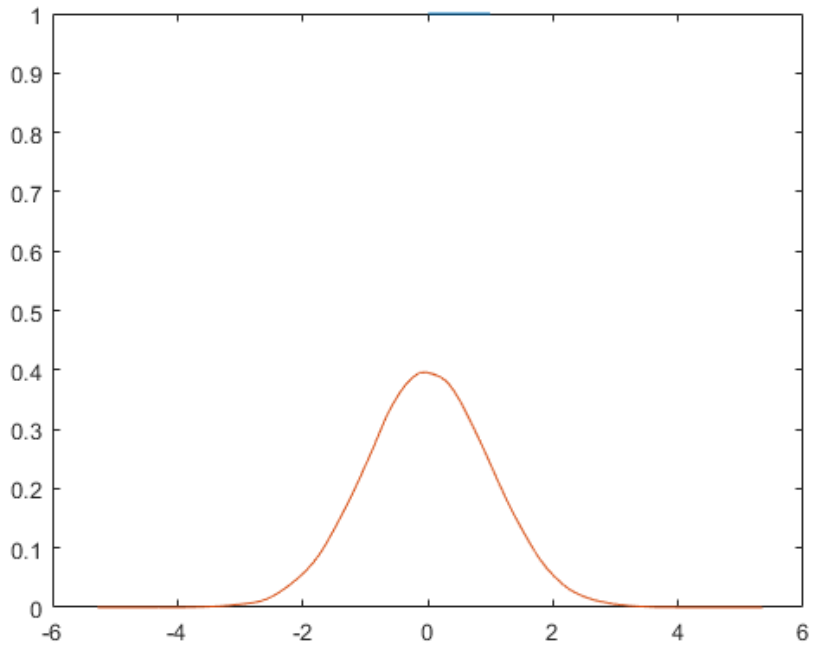
```
        cout << "期望: " << expect << endl;
        for (int i = 0; i < 50000; i++)
        {
            variance += pow(Rand[i] - expect, 2);
        }
        variance /= 50000;
        cout << "方差: " << variance << endl;
        cout << endl << "计算完成";

    }
```

(2)Matlab 验证 test1_1.m

```
x=1/100:1/100:1;
y=ones(1,100);

plot(x,y);
hold on
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");
%disp(data);

[f,xi]=ksdensity(data);
```

（三）测试结果与分析

```
计算中
期望: 2.44037e-05
方差: 1.00298
```



3.4 组合法产生混合高斯分布随机变量

(一)算法分析

设随机变量序列$\{X_i\}$相互独立，具有相同的期望和方差，即

$$E(X_i)=\mu, D(X_i)=\sigma^2。$$

令

$$Y_n=X_1+\cdots+X_n$$

$$\varphi(t)=\frac{Y_n-E(Y_n)}{\sqrt{D(Y_n)}}=\frac{Y_n-n\mu}{\sqrt{n}\sigma}$$

则

$$\varphi(t)\to N(0,1)$$

即

$$\varphi(t)=\frac{1}{\sqrt{2\pi}}e^{-\frac{t^2}{2}}$$

令$y=t\times b+a$,即$t=\frac{y-a}{b}$,可得

$$f(y)=\frac{1}{b}\varphi\left(\frac{y-a}{b}\right)=\frac{1}{\sqrt{2\pi}b}e^{-\frac{(y-a)^2}{2b^2}}$$

即

$$f(y)=\frac{1}{b}\varphi\left(\frac{y-a}{b}\right)\to N(a,b)$$

组合法基本流程:

(1)产生随机整数I，使$P\{I=i\}=p_i$

(2)产生具有分布函数$F_i(x)$的随机变量 x_i

(3)令$x=x_i$

(二)程序代码分析

(1)C++生成混合正态分布> CombinatorialMethod():

```
void CombinatorialMethod()
{
    double p[3] = { 0.5,1. / 3.,1. / 6. };
    double a[3] = { -1,0,1 };
    double b[3] = { 0.25,1,0.5 };

    cout << "计算中" << endl;
    double Rand[MTLength];//均匀分布
    double Randn[MTLength / 10];//标准正态分布
    double Result[MTLength / 10];//混合高斯分布
```

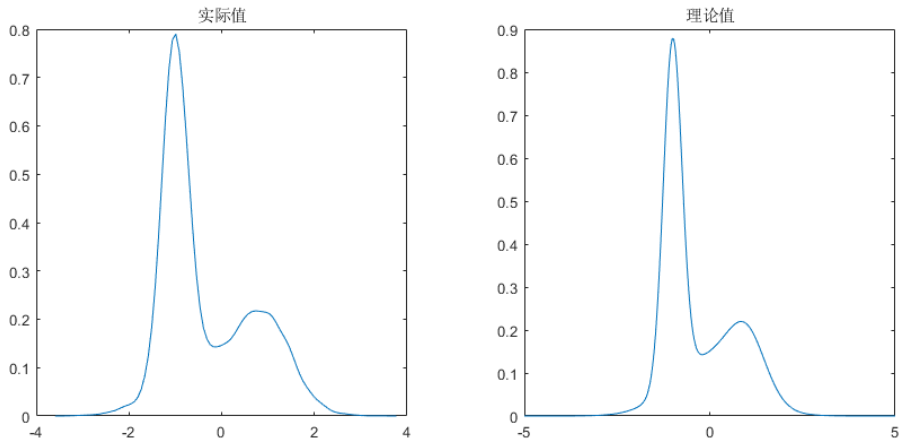
```
    uint32_t* RamdonNumber;
    RamdonNumber = MersenneTwister((uint32_t)time(NULL));
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
    }
    for (int i = 0; i < MTLength/10; i++)
    {
        double sum = 0;
        for (int j = 0; j < 10; j++)
        {
            sum += Rand[10 * i + j];
        }
        Randn[i] = (sum - 10. * 0.5) / (sqrt(10./12.));
    }
    for (int i = 0; i < MTLength / 10; i++)
    {
        if (Rand[i] < p[0])
        {
            Result[i] = Randn[i] * b[0] + a[0];
        }
        else if (Rand[i] < p[0] + p[1])
        {
            Result[i] = Randn[i] * b[1] + a[1];
        }
        else
        {
            Result[i] = Randn[i] * b[2] + a[2];
        }
    }

    delete[] RamdonNumber;
    fstream data;
    data.open("data.txt", ios::out | ios::trunc);
    for (int i = 0; i < MTLength / 10; i++)
    {
        data << Result[i] << endl;
    }
    data.close();

    cout << endl << "计算完成";
}
```

(2)Matlab 验证 test4_1.m

```
clear;
```

```
clc;
close all;

data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");

[f,xi]=ksdensity(data);
subplot(1,2,1)
plot(xi,f)
title("实际值");

x=-5:0.01:5;

p1=0.5;
p2=1/3;
p3=1/6;
a1=-1;
a2=0;
a3=1;
b1=1/4;
b2=1;
b3=1/2;

y1=p1./b1./sqrt(2*pi)*exp(-(x-a1).^2./2./b1^2);
y2=p2./b2./sqrt(2*pi)*exp(-(x-a2).^2./2./b2^2);
y3=p3./b3./sqrt(2*pi)*exp(-(x-a3).^2./2./b3^2);
y=y1+y2+y3;
subplot(1,2,2)
plot(x,y);
title("理论值");
```

(三)测试结果与分析



3.5 泊松过程的产生

(一) 算法分析

泊松过程的两种解释：

(1)某一时间内的事件发生次数服从泊松分布；

(2)相邻两次事件发生的时间间隔服从指数分布

利用第二种解释，可以产生泊松过程

(1)产生均匀分布随机数

(2)利用反变换法产生指数分布随机数

(3)把指数分布随机数作为时间间隔，可以得到一个时间序列

(二) 程序代码分析

(1)C++生成泊松过程> Poisson():

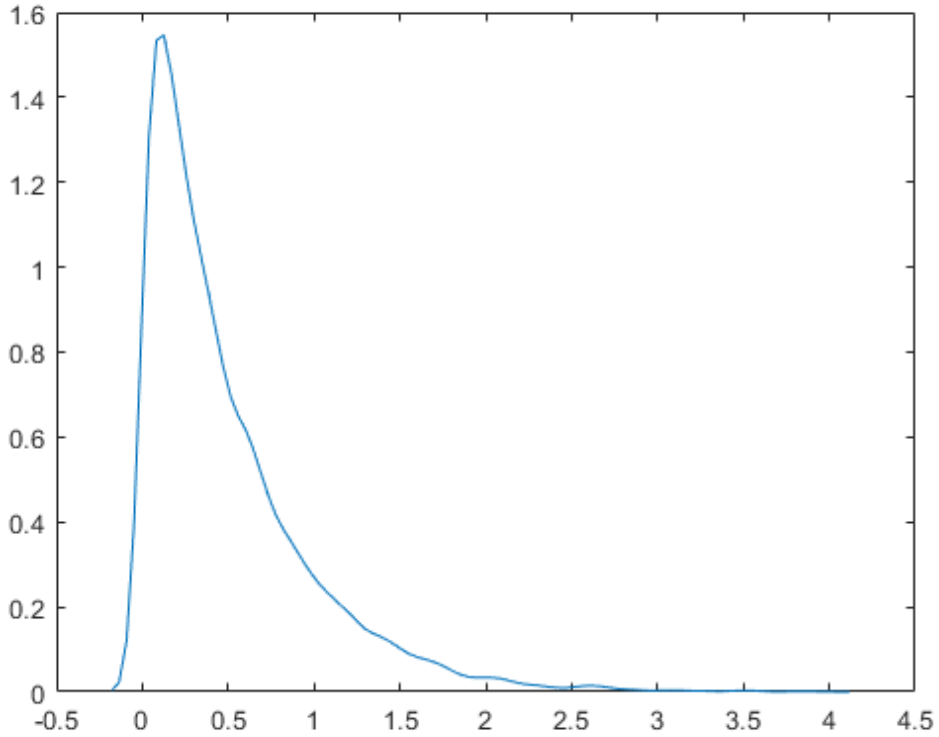
```
void Poisson()
{
    cout << "计算中" << endl;
    double lambda = 2;
    double Rand[MTLength];
    double Result[MTLength];
    uint32_t* RamdonNumber;
    RamdonNumber = MersenneTwister((uint32_t)time(NULL));
    for (int i = 0; i < MTLength; i++)
    {
        Rand[i] = RamdonNumber[i] / 2147483647. / 2.;
        Result[i] = -1. / lambda * log(Rand[i]);
    }
    delete[] RamdonNumber;
    fstream data;
    data.open("data.txt", ios::out | ios::trunc);
    double sum = 0;
    for (int i = 0; i < MTLength ; i++)
    {
        sum += Result[i];
        data << Result[i] << endl;
    }
    data.close();
    cout << endl << "计算完成";

}
```

(2)Matlab 验证 test5_1.m

```
clear;
```

```
clc;
close all;

data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\da
ta.txt");
data(data>4)=[];

[f,xi]=ksdensity(data);
plot(xi,f)
```

(三) 测试结果与分析



4. 实验结论与心得体会

深入理解了均匀分布随机变量产生方法；

掌握了由均匀分布随机变量产生其他分布随机变量的方法；

掌握了两种常用的高斯分布随机变量的产生方法；

掌握了泊松过程的产生方法；