

**信息网络建模与仿真**

**期末综合实验**

**实验报告**

班级：2019211123

姓名：李雨霖

学号：2019210195

2021年12月

# 实验题目

不依赖NS-3和其他常用仿真工具，用C++/Python或Java编写一个离散事件驱动的仿真器，仿真器应当包含对于事件列表和仿真时钟的管理以及各种随机数和随机过程生成函数。在所设计的仿真器上实现一个简单的网络仿真。

1、如果选择自己设计一个离散事件仿真器，需要包含事件类型定义、事件列表管理、仿真时钟管理和随机数生成等关键模块，并对一个简化的网络模型进行仿真，例如呼叫处理、简单MAC机制（Aloha，Slotted Aloha）等。

2、实验结束安排验收，验收时需要讲明所设计的仿真器的功能，并对仿真程序进行演示。

3、实验报告需要完整呈现仿真器的设计思路，包括类、方法等。

# 实验内容

本实验由Python编写。离散事件驱动的仿真器中包含事件类型定义、事件列表管理、仿真时钟管理和随机数生成等关键模块。在设计的仿真器上实现了Slotted Aloha的仿真。

# 实验步骤与结果

## 均匀分布随机数的产生

### 线性同余法产生均匀分布随机数

1. 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：

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，，，

1. 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);

1. 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;

}

1. 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";

}

1. Matlab验证test1\_1.m

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

y=ones(1,100);

plot(x,y);

hold on

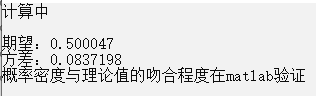
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");

%disp(data);

[f,xi]=ksdensity(data);

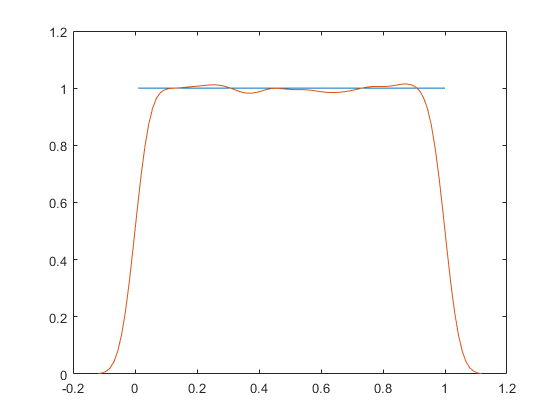
plot(xi,f)

1. 测试结果



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。

### Mersenne Twister方法产生均匀分布随机数

1. 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赋给作为初值，然后根据递推式：

递推求出梅森旋转链。

1. 对于旋转链进行旋转算法；

遍历旋转链，对每个，根据递推式：

进行旋转链处理。

其中，“”代表连接的意思，即组合的高 位和的低 位，设组合后的数字为，则的运算规则为（是最低位）：

1. 对于旋转算法所得的结果进行处理；

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

1. 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);

1. 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;

}

1. 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";

}

1. Matlab验证test1\_1.m

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

y=ones(1,100);

plot(x,y);

hold on

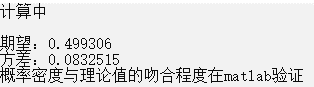
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");

%disp(data);

[f,xi]=ksdensity(data);

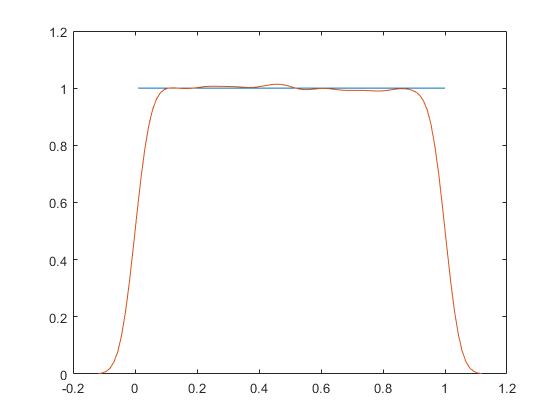
plot(xi,f)

1. 测试结果与分析



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。

### MRG32k3a方法产生均匀分布随机数

1. 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和，它们根据线性递归进化

输出定义为

这个随机数发生器的优点是算法相对简单，周期长达， 并可以很容易地切分成长度为的不同段，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，并且不能全为0；并且最后三个整数必须都小于，并且不能全为0。

1. 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);

1. 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;

}

1. 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";

}

1. Matlab验证test1\_1.m

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

y=ones(1,100);

plot(x,y);

hold on

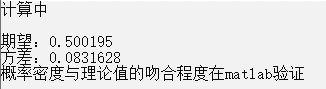
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");

%disp(data);

[f,xi]=ksdensity(data);

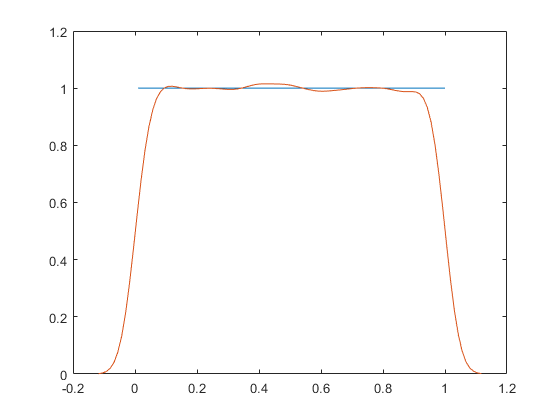
plot(xi,f)

1. 测试结果与分析



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。

## Pareto分布随机变量的产生

1. 算法分析

Pareto分布概率密度函数为：

Pareto分布的分布函数为：

Pareto分布的生成函数为：

即服从(0, 1)区间上的均匀分布

Pareto分布的数学期望为：

Pareto分布的方差为：

1. 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])

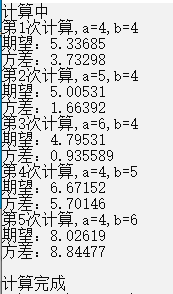
subplot(2,3,3);cdfplot(data3);xlim([0 20])

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])

1. 测试结果与分析

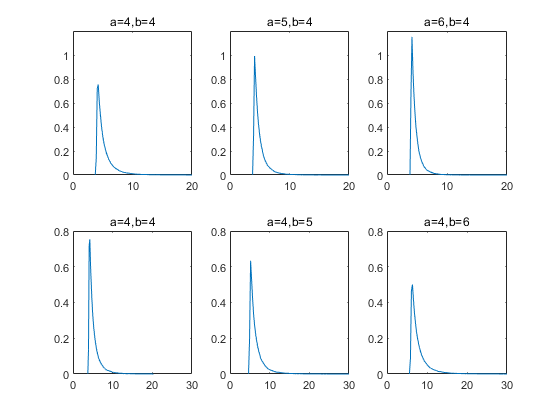


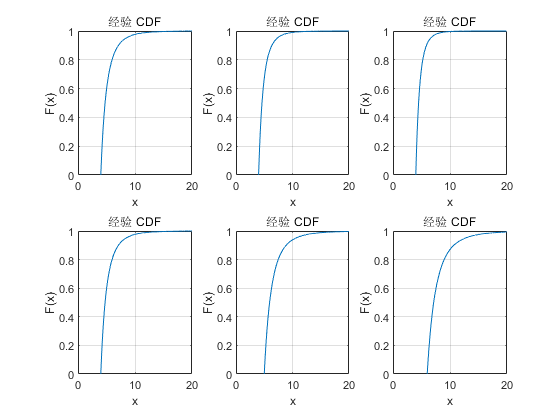
期望和方差的理论值如下表：

|  |  |  |
| --- | --- | --- |
|  | 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。





## 高斯分布随机变量的产生

### 中心极限定理法产生高斯分布随机变量

1. 算法分析

中心极限定理：

设随机变量序列相互独立，具有相同的期望和方差，即

。

令

则

1. 程序代码分析
2. 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";

}

1. 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");

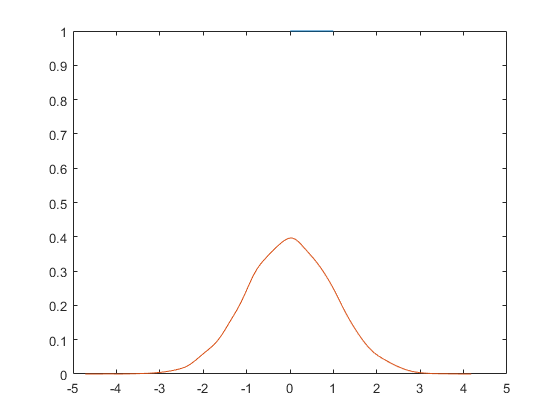
%disp(data);

[f,xi]=ksdensity(data);

plot(xi,f)

1. 测试结果与分析





### Box-Muller法产生高斯分布随机变量

1. 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：选取两个服从上均匀分布的随机变,

若满足

则X与Y服从均值为0，方差为1的高斯分布。

1. 程序代码分析
2. 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";

}

1. Matlab验证test1\_1.m

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

y=ones(1,100);

plot(x,y);

hold on

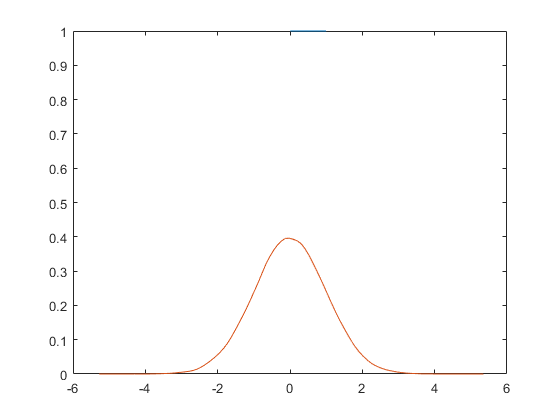
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");

%disp(data);

[f,xi]=ksdensity(data);

1. 测试结果与分析





## 组合法产生混合高斯分布随机变量

1. 算法分析

设随机变量序列相互独立，具有相同的期望和方差，即

。

令

则

即

令,即,可得

即

组合法基本流程:

1. 产生随机整数，使
2. 产生具有分布函数的随机变量
3. 令
4. 程序代码分析
5. 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";

}

1. Matlab验证test4\_1.m

clear;

clc;

close all;

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

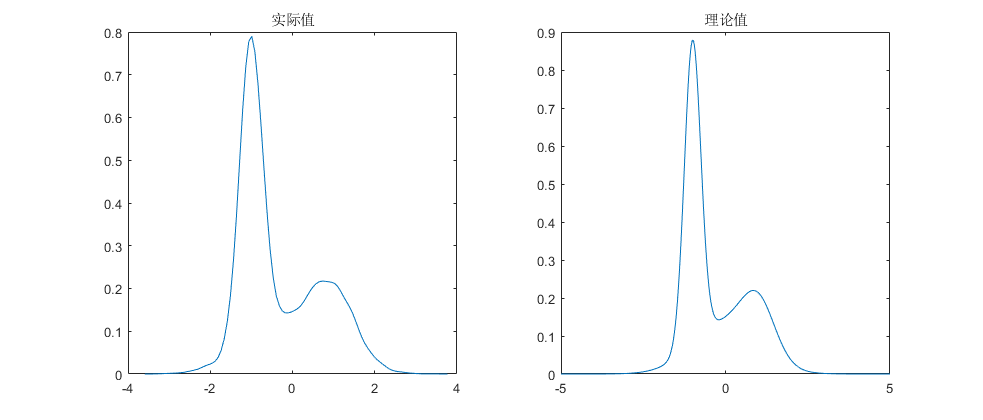
y=y1+y2+y3;

subplot(1,2,2)

plot(x,y);

title("理论值");

1. 测试结果与分析



## 泊松过程的产生

1. 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
4. 程序代码分析
5. 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";

}

1. Matlab验证test5\_1.m

clear;

clc;

close all;

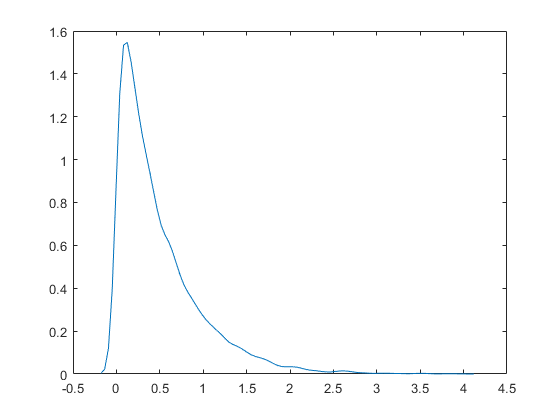
data=load("C:\Users\Scarlet\Desktop\MessageNetwork\MessageNetwork\data.txt");

data(data>4)=[];

[f,xi]=ksdensity(data);

plot(xi,f)

1. 测试结果与分析



# 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；