

• 0x00 前言

• 0x01 知识回顾

- 均匀分布
- 概率分布函数和概率数
- 中心极限定理
- 反变换法(Inverse tra sampling)
- 雅可比矩阵与雅可比
- 拒绝采样(Rejection Sampling)

• 0x02 暴力生成正态分布

• 0x03 反变换法生成正态

• 0x04 Box–Muller算法

• 0x06 总结

• 0x07 参考资料

• Similar Posts

• Comments

花式生成正态分布

📅 2015-05-20 👤 宾狗 📁 学术 🏷 数学

0x00 前言

“So much of life, it seems to me, is determined by pure randomness.” – Sidney Poitier

在之前的博客中，或多或少都提到过一些随机、伪随机、随机数等等，但基本上只是直接使用，没有探寻背后的一些原理，刚好最近偶然看到 `python` 标准库中如何生成服从正态分布随机数的源码，所以本文就简单聊聊如何生成正态分布

0x01 知识回顾

为了防止你对接下来的内容一头雾水，我觉得还是有必要回顾一下我们曾经学过的高数和概率统计知识

均匀分布

均匀分布是生成其他分布的基础，基本上只要是个编程语言，其标准函数库里面肯定有一个随机生成 $[0, 1)$ 之间浮点数的函数，原理很简单，使用的是**线性同余法**（linear congruential generator, LCG），依据下面这个递推公式

$$X_{n+1} = (aX_n + c) \pmod m$$

其中 X 就是伪随机数序列， X_n 是序列中第 N 个数， a, c, m 是常数， a 是乘数， c 是增量， m 是模，我们熟悉的随机种子 `seed` 是 X_0

LCG的周期最大为 m ，但大部分情况都会小于 m 。要令LCG达到最大周期，应符合以下条件：

- c, m 互素
- m 的所有质因数都能整除 $a - 1$
- 若 m 是4的倍数， $a - 1$ 也是
- a, c, x_0 都比 m 小
- a, c 是正整数

不过这些约束怎么来的本文就暂不讨论了

我们以 `Java` 中 `Random` 的实现为例，为了便于理解，这里对代码进行了部分调整，只截取了相关的片段

```
//这两个东西在本文后面讲正态分布的时候会涉及到
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

//随机数序列生成器种子
private final AtomicLong seed;

//线性同余发生器乘数a
private final static long multiplier = 0x5DEECE66DL;

//线性同余发生器加数c
private final static long addend = 0xBL;

//线性同余发生器模数m
private final static long mask = (1L << 48) - 1;

public Random(long seed) {
    this.seed = new AtomicLong(0L);
    setSeed(seed);
}
```

```
}

synchronized public void setSeed(long seed) {
    //实现线性同余算法
    seed = (seed ^ multiplier) & mask;
    //atomic set具有写入（分配）volatile 变量的内存效果（即具有内存可见性）
    this.seed.set(seed);
    //暂且忽略。。
    haveNextNextGaussian = false;
}

protected int next(int bits) {
    long oldseed, nextseed;
    AtomicLong seed = this.seed;
    //compareAndSet 如果当前值==预期值，则以原子方式将该值设置为给定的更新值（利用了CPU的硬件原语CAS指令）
    do {
        //atomic get具有读取volatile 变量的内存效果
        oldseed = seed.get();
        //实现线性同余算法
        nextseed = (oldseed * multiplier + addend) & mask;
    } while (!seed.compareAndSet(oldseed, nextseed));
    //截取bits位整型
    return (int)(nextseed >>> (48 - bits));
}
```

有人可能会有疑惑，这个代码中的实现 `nextseed = (oldseed * multiplier + addend) & mask` 好像和递推公式不一样啊？那个模运算为什么变成了与运算？

注意，`x & [(1L << 48)-1]` 与 `x(mod 2^48)` 是等价的。为什么呢？从二进制的角度来考虑这个问题就很清楚了。一个数 `x` 除以 `2^n`，在二进制中相当于将 `x` 右移 `n` 位，商和余数分别在小数点左侧和右侧。如 `23/8=2`，`23%8=7`，`23` 的二进制表示为 `10111`，除以 `2^3=8` 相当于右移3位，得到 `10.111`，左侧为商 `10` 也就是2，右侧为余数 `111` 也就是7。也就是说如果一个数对 `2^n` 取余，那么只需要得到该数的低 `n` 位即可，很自然的想到如果能得到这样 `0...01...1` 一个二进制数，与原来的数做与运算即可，而 `2^n-1` 恰好可以得到这样的一个二进制数。如 `2^3-1=7` → `111`，`2^7-1=127` → `1111111`。

现在回头看看 `nextseed = (oldseed * multiplier + addend) & mask` 这行代码可以理解了吧？

关于均匀分布的更多细节可以参考JDK源码分析——从java.util.Random源码分析线性同余算法

概率分布函数和概率密度函数

直接引用教材上的黑体字吧

设随机变量 X 的分布函数为 $F(x)$ ，若存在非负实函数 $f(x)$ ，使对任意实数 x ，有 $F(x) = \int_{-\infty}^x f(x) dx$ ，则称 X 为连续型随机变量， $f(x)$ 称为 X 的概率密度函数.简称概率密度或密度函数

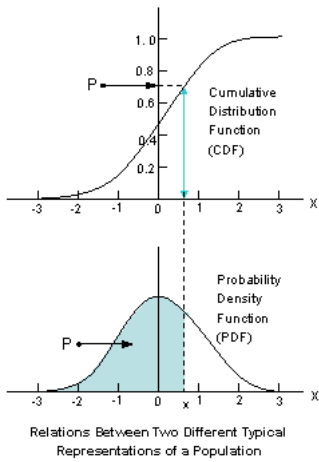
看下面这张图也非常清楚，概率分布函数 $F(x)$ 是 **cumulative distribution function(CDF)**，概率密度函数 $f(x)$ 是 **probability density function(PDF)**

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box–Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

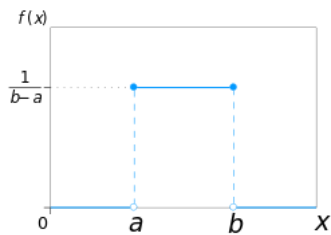
Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments



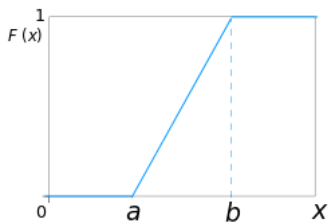
刚才说到的均匀分布概率密度函数 $f(x)$ 如下

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{for } a \leq x \leq b, \\ 0 & \text{for } x < a \text{ or } x > b \end{cases}$$



概率分布函数 $F(x)$ 如下

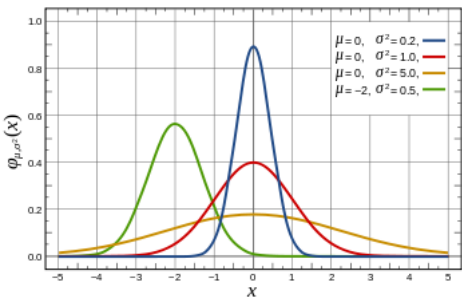
$$F(x) = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } x \in [a, b] \\ 1 & \text{for } x \geq b \end{cases}$$



期望为 $\frac{1}{2}(a + b)$, 方差为 $\frac{1}{12}(b - a)^2$

正态分布的概率密度函数 $f(x)$ 如下

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



标准正态分布中, $\mu = 0, \sigma = 1$, $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

正态分布的概率分布函数不太好求, 不信自己去积分试试...

中心极限定理

设 X_1, X_2, \dots, X_n 为独立同分布的随机变量序列, 均值为 μ , 方差为 σ^2 , 则

$$Z_n = \frac{X_1 + X_2 + \dots + X_n - n\mu}{\sigma\sqrt{n}}$$

具有渐近分布 $N(0, 1)$, 也就是说当 $n \rightarrow \infty$ 时,

$$P\left\{\frac{X_1 + X_2 + \dots + X_n - n\mu}{\sigma\sqrt{n}} \leq x\right\} \rightarrow \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt$$

说人话就是, n 个相互独立同分布的随机变量之和的分布近似于正态分布, n 越大, 近似程度越好

反变换法(Inverse transform sampling)

假设 $u = F(x)$ 是一个概率分布函数(CDF), F^{-1} 是它的反函数, 若 U 是一个服从 $(0, 1)$ 均匀分布的随机变量, 则 $F^{-1}(U)$ 服从函数 F 给出的分布

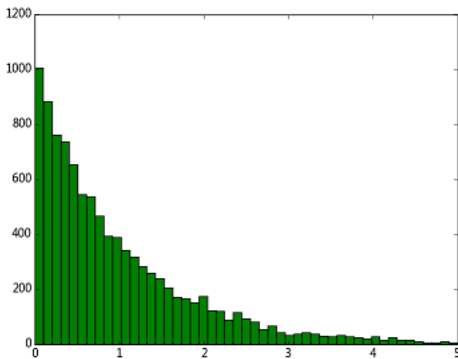
例如要生成一个服从指数分布的随机变量, 我们知道指数分布的概率分布函数(CDF)为 $F(x) = 1 - e^{-\lambda x}$, 其反函数为 $F^{-1}(x) = -\frac{\ln(1-x)}{\lambda}$, 写程序实现一下

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np

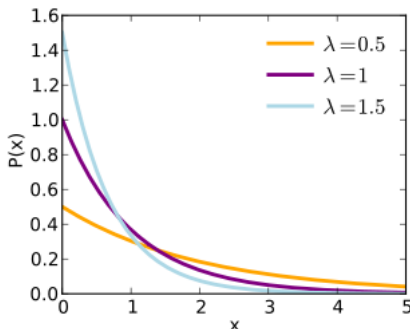
def getExponential(SampleSize, p_lambda):
    result = -np.log(1-np.random.uniform(0,1,SampleSize))/p_lambda
    return result

# 生成10000个数, 观察它们的分布情况
SampleSize = 10000
es = getExponential(SampleSize, 1)
plt.hist(es, np.linspace(0, 5, 50), facecolor="green")
plt.show()
```

得到如下结果



对比维基百科里面标准的指数分布



那么为什么 $F^{-1}(U)$ 会服从 F 给出的分布呢? 其实很好证明, $P(F^{-1}(U) \leq x)$, 两边同时取 F 得到 $P(F^{-1}(U) \leq x) = P(U \leq F(x))$, 根据均匀分布的定义 $P(U < y) = y$, 所以 $P(U \leq F(x)) = F(x)$, 即 $P(F^{-1}(U) \leq x) = F(x)$, 刚好是随机变量服从某个分布的定义, 证毕~

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse transform sampling)
 - 雅可比矩阵与雅可比数
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态分布
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

$$\begin{aligned} P(F^{-1}(U) \leq x) &= P(U \leq F(x)) \\ &= F(x) \end{aligned}$$

雅可比矩阵与雅可比行列式

这个东西在高数课本中有，只怪当初学习不用功.....

设 $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 是一个向量函数，输入为 $\mathbf{x} \in \mathbb{R}^n$ ，输出为 $\mathbf{f}(\mathbf{x}) \in \mathbb{R}^m$ ，雅可比矩阵可以写成如下形式：

$$J = \frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

当 $m = n$ 时，雅可比矩阵为一个方阵，我们可以取它的行列式

$$|J| = \begin{vmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{vmatrix}$$

以咱们熟悉的平面直角坐标与极坐标转换为例吧，

$$\begin{cases} x = r \cos \theta \\ y = r \sin \theta \end{cases}$$

求雅可比矩阵，

$$J = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & -r \sin \theta \\ \sin \theta & r \cos \theta \end{bmatrix}$$

取行列式

$$|J| = r \cos^2 \theta + r \sin^2 \theta = r$$

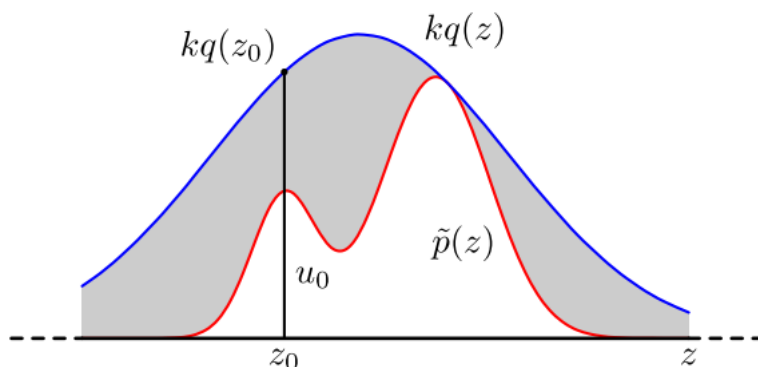
回顾当初学习二重积分时，利用极坐标变换时的式子如下

$$\iint_D f(x, y) dx dy = \iint_D f(r \cos \theta, r \sin \theta) r dr d\theta$$

现在知道那个多出来的 r 是怎么回事了吧？雅可比行列式相当于两个不同坐标系中微小区域面积的缩放倍数

拒绝采样(Rejection Sampling)

这个方法有的时候也称接收-拒绝采样，使用场景是有些函数 $p(x)$ 太复杂在程序中没法直接采样，那么可以设定一个程序可抽样的分布 $q(x)$ 比如正态分布等等，然后按照一定的方法拒绝某些样本，达到接近 $p(x)$ 分布的目的：



具体操作如下，设定一个方便抽样的函数 $q(x)$ ，以及一个常量 k ，使得 $p(x)$ 总在 $kq(x)$ 的下方。（参考上图）

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse transformation)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

- x 轴方向：从 $q(x)$ 分布抽样得到 a
- y 轴方向：从均匀分布 $(0, kq(a))$ 中抽样得到 u
- 如果刚好落到灰色区域： $u > p(a)$ ，拒绝；否则接受这次抽样
- 重复以上过程

证明过程就不细说了，知道怎么用就行了，感兴趣的可以看看这个文档

- Acceptance-Rejection Method

不过在高维的情况下，拒绝采样会出现两个问题，第一是合适的 q 分布比较难以找到，第二是很难确定一个合理的 k 值。这两个问题会造成图中灰色区域的面积变大，从而**导致拒绝率很高，无用计算增加**。

0x02 暴力生成正态分布

根据中心极限定理，生成正态分布就非常简单粗暴了，直接生成 n 个独立同分布的随机变量，求和即可。注意，**无论**你使用什么分布，当 n 趋近于无穷大时，它们和的分布都会趋近正态分布！

以最简单的均匀分布为例，看代码

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np

def getNormal(SampleSize,n):
    xsum = []
    for i in range(SampleSize):
        # 利用中心极限定理，[0,1)均匀分布期望为0.5，方差为1/12
        tsum = (np.mean(np.random.uniform(0,1,n))-0.5)*np.sqrt(12*n)
        xsum.append(tsum)
    return xsum

# 生成10000个数，观察它们的分布情况
SampleSize = 10000

# 观察n选不同值时，对最终结果的影响
N = [1,2,10,1000]

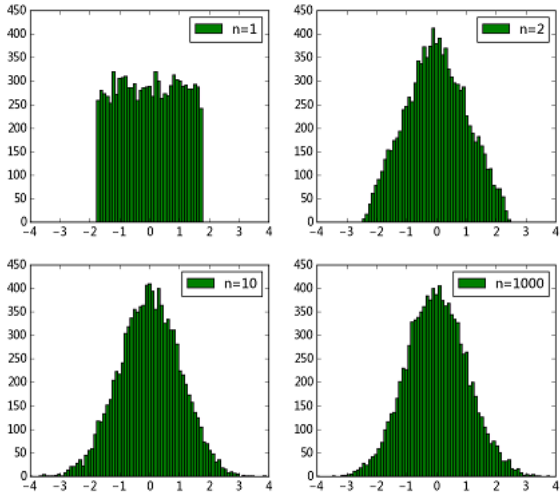
plt.figure(figsize=(10,20))
subi = 220
for index,n in enumerate(N):
    subi += 1
    plt.subplot(subi)
    normalsum = getNormal(SampleSize, n)
    # 绘制直方图
    plt.hist(normalsum,np.linspace(-4,4,80),facecolor="green",label="n={0}".format(n))
    plt.ylim([0,450])
    plt.legend()

plt.show()
```

得到结果如下图所示

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box–Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments



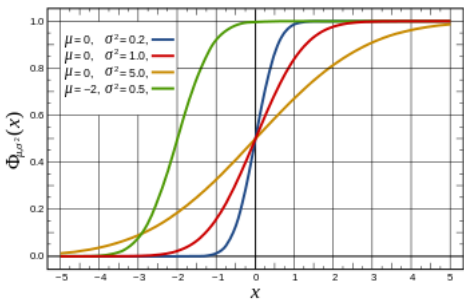
可以看到， $n=1$ 时其实就是均匀分布， $n=2$ 时有正态分布的样子了，但不够平滑，随着 n 逐渐增大，直方图轮廓越来越接近正态分布了~因此利用中心极限定理暴力生成服从正态分布的随机数是可行的

但是这样生成正态分布速度是非常慢的，因为要生成若干个同分布随机变量，然后求和、计算，效率非常低。那有没有其他办法呢？

当然有！利用反变换法

0x03 反变换法生成正态分布

正态分布的概率分布函数(CDF)如下图所示，



在 y 轴上产生服从(0,1)均匀分布的随机数，水平向右投影到曲线上，然后垂直向下投影到 x 轴，这样在 x 轴上就得到了正态分布。

当然正态分布的概率分布函数不方便直接用数学形式写出，求反函数也无从说起，不过好在 `scipy` 中有相应的函数，我们直接使用即可

```
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm

def getNormal(SampleSize):
    iid = np.random.uniform(0,1,SampleSize)
    result = norm.ppf(iid)
    return result

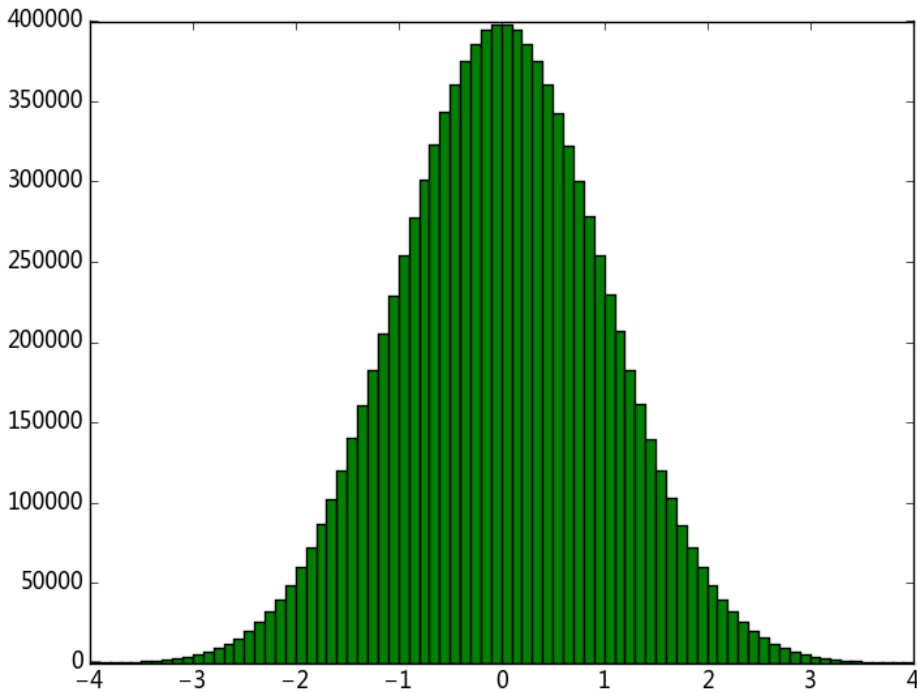
SampleSize = 1000000
normal = getNormal(SampleSize)

plt.hist(normal,np.linspace(-4,4,81),facecolor="green")
plt.show()
```

结果如下图所示，

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments



以上两个方法虽然方便也容易理解，但是效率实在太低，并不实用，那么在实际中到底是如何生成正态分布的呢？

0x04 Box-Muller算法

说来也巧，某天闲的无聊突然很好奇 `python` 是如何生成服从正态分布的随机数的，于是就看了看 `random.py` 的代码，具体实现的代码就不贴了，大家自己去看，注释中有下面几行

```
# When x and y are two variables from [0, 1), uniformly
# distributed, then
#
#   cos(2*pi*x)*sqrt(-2*log(1-y))
#   sin(2*pi*x)*sqrt(-2*log(1-y))
#
# are two *independent* variables with normal distribution
```

顿时感觉非常科幻，也就是说当 x 和 y 是两个独立且服从 $[0,1)$ 均匀分布的随机变量时， $\cos(2\pi x) \cdot \sqrt{-2 \ln(1-y)}$ 和 $\sin(2\pi x) \cdot \sqrt{-2 \ln(1-y)}$ 是两个独立且服从正态分布的随机变量！

后来查了查这个公式，发现这个方法叫做 `Box-Muller`，其实本质上也是应用了反变换法，证明方法比较多，这里我们选取一种比较好理解的

我们把反变换法推广到二维的情况，设 U_1, U_2 为 $(0, 1)$ 上的均匀分布随机变量， (U_1, U_2) 的联合概率密度函数为 $f(u_1, u_2) = 1 (0 \leq u_1, u_2 \leq 1)$ ，若有：

$$\begin{cases} U_1 = g_1(X, Y) \\ U_2 = g_2(X, Y) \end{cases}$$

其中， g_1, g_2 的逆变换存在，记为

$$\begin{cases} x = h_1(u_1, u_2) \\ y = h_2(u_1, u_2) \end{cases}$$

且存在一阶偏导数，设 J 为Jacobian矩阵的行列式

$$|J| = \begin{vmatrix} \frac{\partial x}{\partial u_1} & \frac{\partial x}{\partial u_2} \\ \frac{\partial y}{\partial u_1} & \frac{\partial y}{\partial u_2} \end{vmatrix} \neq 0$$

则随机变量 (X, Y) 的二维联合密度为（回顾直角坐标和极坐标变换）：

$$f[h_1(u_1, u_2), h_2(u_1, u_2)] \cdot |J| = |J|$$

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

根据这个定理我们来证明一下,

$$\begin{cases} Y_1 = \sqrt{-2 \ln X_1} \cos(2\pi X_2) \\ Y_2 = \sqrt{-2 \ln X_1} \sin(2\pi X_2) \end{cases}$$

求反函数得

$$\begin{cases} X_1 = e^{-\frac{Y_1^2 + Y_2^2}{2}} \\ X_2 = \frac{1}{2\pi} \arctan \frac{Y_2}{Y_1} \end{cases}$$

计算Jacobian行列式

$$\begin{aligned} |J| &= \begin{vmatrix} \frac{\partial X_1}{\partial Y_1} & \frac{\partial X_1}{\partial Y_2} \\ \frac{\partial X_2}{\partial Y_1} & \frac{\partial X_2}{\partial Y_2} \end{vmatrix} = \begin{vmatrix} -Y_1 \cdot e^{-\frac{1}{2}(Y_1^2 + Y_2^2)} & -Y_2 \cdot e^{-\frac{1}{2}(Y_1^2 + Y_2^2)} \\ -\frac{Y_2}{2\pi(Y_1^2 + Y_2^2)} & \frac{Y_1}{2\pi(Y_1^2 + Y_2^2)} \end{vmatrix} \\ &= e^{-\frac{1}{2}(Y_1^2 + Y_2^2)} \left[\frac{-Y_1^2}{2\pi(Y_1^2 + Y_2^2)} - \frac{Y_2^2}{2\pi(Y_1^2 + Y_2^2)} \right] \\ &= -\frac{1}{2\pi} e^{-\frac{1}{2}(Y_1^2 + Y_2^2)} \\ &= -\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}Y_1^2} \right) \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}Y_2^2} \right) \end{aligned}$$

由于 X_1, X_2 为 $(0, 1)$ 上的均匀分布, 概率密度函数均为 1, 所以 Y_1, Y_2 的联合概率密度函数为 $-\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}Y_1^2}\right)\left(\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}Y_2^2}\right)$, 熟悉二维正态分布的就知道是两个独立的正态分布, 所以 Y_1, Y_2 是两个独立且服从正态分布的随机变量~

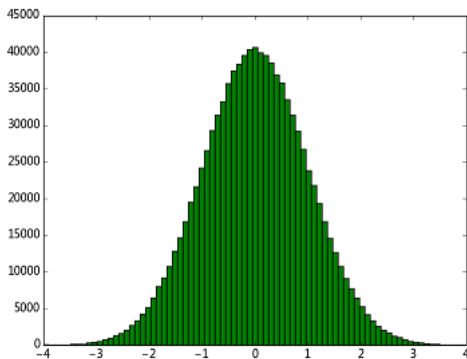
写程序实现一下

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np

def getNormal(SampleSize):
    iid = np.random.uniform(0,1,SampleSize)
    normal1 = np.cos(2*np.pi*iid[0:SampleSize/2-1])*np.sqrt(-2*np.log(iid[SampleSize/2:SampleSize-1]))
    normal2 = np.sin(2*np.pi*iid[0:SampleSize/2-1])*np.sqrt(-2*np.log(iid[SampleSize/2:SampleSize-1]))
    return np.hstack((normal1,normal2))

# 生成10000000个数, 观察它们的分布情况
SampleSize = 10000000
es = getNormal(SampleSize)
plt.hist(es,np.linspace(-4,4,80),facecolor="green")
plt.show()
```

得到的结果如下图所示,



这里抽样次数达到1千万次, 1秒左右就完成了, 速度比暴力生成正态分布要快的多~

ps: 由于 **Box-Muller** 算法一次性生成了两个独立且服从正态分布的随机数, 所以可以把其中一个保存起来, 下次直接使用即可。本文刚开始的那段代码中 `nextNextGaussian` 就是用来保存它的~

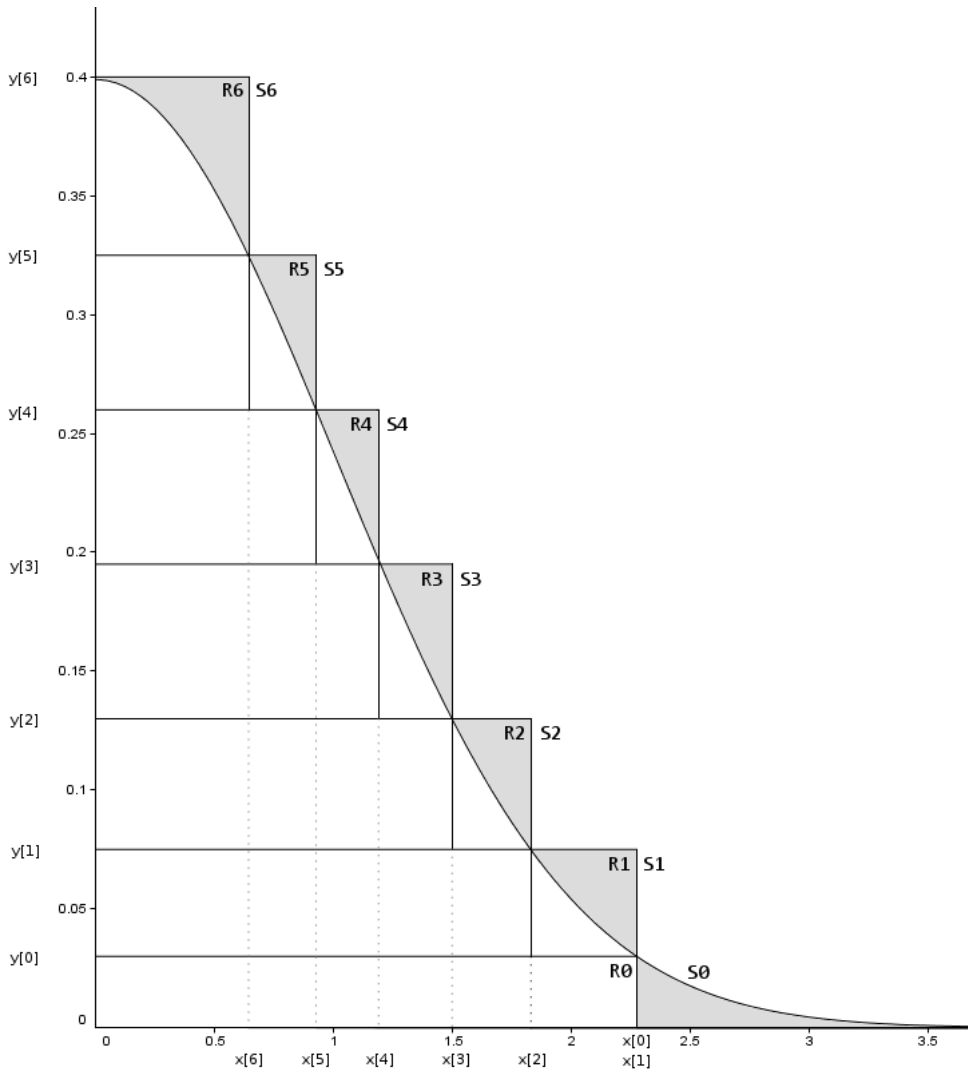
#0x05 Ziggurat Algorithm

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

`Box-Muller` 算法虽然快了许多，但是由于用到了三角函数和对数函数，相对来说还是比较耗时的，如果想要更快一点有没有办法呢？

当然有，这就是 `Ziggurat` 算法，不仅可以用于快速生成正态分布，还可以生成指数分布等等。其基本思想就是利用拒绝采样，其高效的秘密在于构造了一个非常精妙的 $q(x)$ ，看下面这张图



如果为了方便，我们当然可以直接使用一个均匀分布，也就是一个矩形，但是这样的话，矩形与正态分布曲线间的距离很大，容易造成拒绝率很高，无用计算增加，高效也就无从谈起了

再看看上面那张图，我们用多个堆叠在一起的矩形，这样保证阴影部分（被拒绝部分）的始终较小，这样就非常高效了

简单对图作一个解释：

- 我们用 $R[i]$ 来表示一个矩形， $R[i]$ 的右边界为 $x[i]$ ，上边界为 $y[i]$ 。 $S[i]$ 表示一个分割，当 $i=0$ 时， $S[0]=R[0]+tail$ ，其他情况 $S[i]=R[i]$
- 除了 $R[0]$ 以外，其他每个矩形面积相等，设为定值 A 。 $R[0]$ 的面积 = $A - tail$ 的面积。这样保证从任何一个分割中抽样 (x,y) 的概率相等
- 当任意选定一个 $R[i]$ 在其中抽样 (x,y) ，若 $x < x[i+1]$ ， y 必然在曲线下方，满足条件，接受 x ；若 $x[i+1] < x < x[i]$ ，则还需要进一步判断。同样这里 $R[0]$ 和 $tail$ 中的样本需要进行特殊处理
- 这里为了方便解释，只用了几个矩形，在程序实现的时候，一般使用 128 或 256 个矩形

可以看出，为了提高效率，`Ziggurat` 算法中使用了许多技巧性的东西，这在其 `C` 代码实现中更加明显，使用了与运算和字节等各种小技巧，代码就不在这里贴了，感兴趣可以看看下面几个版本，`C` 版本的追求的是极致的速度，每个矩形的边界已经提前计算好了。`C#` 版本中的注释非常详细，`Java` 版的基本与 `C#` 一致，但是效率一般。

- `C`
- `C#`
- `Java`

最后对比一下 `Ziggurat` 算法与 `Box-muller` 算法的效率

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse transformation)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments

Hardware
CPU: Intel Core i7 920
4 cores, 8 with hyperthreading (enabled).
Bloomfield / 45nm / 1.248V
Core Speed 2.798 GHz
Bus Speed 133.2 MHz
Multiplier 21x
RAM: DDR3, 3 channels. 533 MHz

Cores	Box Muller (Million samples/sec)	Ziggurat (Million samples/sec)	Speedup
1	23.2	36.7	1.58x
2	42	73.3	1.74x
3	66	108	1.63x
4	83	133	1.60x
8 (hyperthreading)	138	204	1.47

0x06 总结

本文介绍了多种生成正态分布的方法，其中 `Box-muller` 算法应对一般的需求足够了，但是要生成大量服从正态分布的随机数时，`Ziggurat` 算法效率会更高一点~

再说点题外话，作为一名普通的程序员，对于很多东西往往不需要了解的非常深入，说白了“会用就行了”。但是有的时候探寻其背后的原理往往能发现别人领会不到的数学之美，这也是写程序之余的一点乐趣吧~

0x07 参考资料

- 大数定律与中心极限定理
- Jacobian 矩陣與行列式
- 从随机过程到马尔科夫链蒙特卡洛方法
- 随机数产生原理
- Transformed Random Variables
- Box-Muller Transform Normality
- The Ziggurat Method for Generating Random Variables
- The Ziggurat Algorithm for Random Gaussian Sampling

上一篇 另类新浪微博基本数据采集方法

下一篇 利用机器学习进行恶意代码分类

Comments

Content

- 0x00 前言
- 0x01 知识回顾
 - 均匀分布
 - 概率分布函数和概率数
 - 中心极限定理
 - 反变换法(Inverse tra sampling)
 - 雅可比矩阵与雅可比
 - 拒绝采样(Rejection Sampling)
- 0x02 暴力生成正态分布
- 0x03 反变换法生成正态
- 0x04 Box-Muller算法
- 0x06 总结
- 0x07 参考资料
- Similar Posts
- Comments