18070100017钟保明

# 第一部分

# 产生正态分布的随机数

1. 简要引出问题：

要在编程中得到服从均匀分布的伪随机数是容易的，C、Python、Java语言等都提供了相应的函数。但是要想生成服从正态分布的随机数就没那么容易了，生成服从正态分布的随机数的基本思想是先得到服从均匀分布的随机数，再将服从均匀分布的随机数转变为服从正态分布。目前的主要方法有：

1. 利用分布函数的反函数

使用反函数，先随机抽出一个服从[0,1]均匀分布的数字u，然后

z是正态分布的。

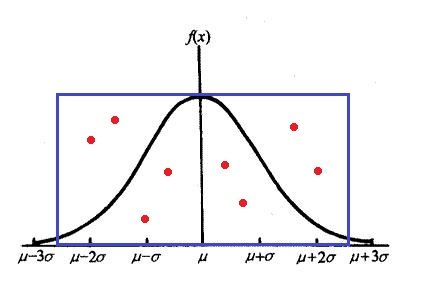
1. Box-Muller方法
2. 利用中心极限定理：

利用林德伯格-莱维（Lindeberg—Levi）中心极限定理, 独立同分布的事件，具有相同的期望和方差，则事件服从中心极限定理。他表示了对于抽取样本，n足够大的时候，样本分布符合x~N(μ,σ^2)。

中心极限定理告诉我们，当样本量足够大时，样本均值的分布慢慢变成正态分布。

1. zigurat 方法
2. zigurat算法介绍：

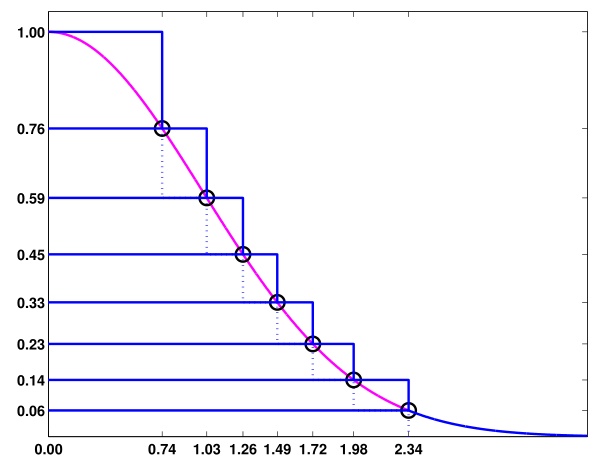
先考虑最简单的舍弃，其方法如下图所示，首先生成在蓝色矩形框中均匀分布的随机点，然后舍弃所有未落在正态分布钟形曲线以下的点，剩余点的横坐标就构成一个服从相应正态分布的随机数序列。rejection sampling的效率之所以低下，就是由于很多生成的随机点因为落在钟形曲线以上而被白白舍弃。



如果有一种方法能够直接产生在钟形曲线下方面积内均匀分布的随机点，那么就不存在舍弃随机点的情况，也就能够避免上面所说的“浪费”问题，这正是ziggurat算法的基本思路。具体来说，ziggurat算法的内容如下：

% 以下内容均遵从MATLAB语言的规定，数组及向量的下标索引从1开始编号

正态分布的钟形曲线左右对称，这里只取右半边考虑，如下图所示。首先将钟形曲线以下的面积等分为n份，包括n-1个水平矩形和最下方一直延伸至无限大的尾巴，即n-1个矩形及最下方尾巴的面积都相等。图中取n=8，实践中可能达到200以上。设各水平矩形右边界的横坐标（即各圆圈标记的横坐标）为，其中。当n确定后，也就确定了，可以通过数值方法求解超越方程得到的值并制表记录。这里称长度方向上与上一层矩形重叠的部分为当前矩形的核心区，下图中各核心区的右边界就是虚线边，显见最上面的矩形没有核心区。



定义，即相邻两层矩形水平方向长度的比值，规定。显然，依照所要求的参数（）生成正态分布时，所有的都只需要在初始化时计算一次并制表记录即可，之后不需要再重复计算，除非所要求生成的正态分布的参（）发生改变。

初始化结束后就可以开始生成正态分布随机数了：首先生成一个区间[1,n]内均匀分布的随机整数和一个(-1,1)区间内均匀分布的随机浮点数。然后检查是否成立，若成立则就在第j段核心区内，显然也就在钟形曲线下方，故可将作为正态分布的一个采样点返回。

1. 特点和与其它方法相比的优势：

用ziggurat算法时只需要生成一个随机整数和一个随机浮点数，做一次条件判断和一次乘法运算，外加两次查表即可得到一个服从正态分布的随机数，整个过程中没有开根号、求对数、算三角函数等复杂运算。

与方法的比较：

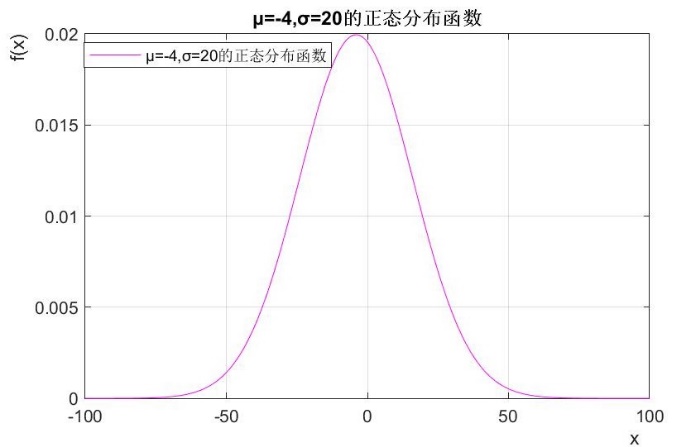
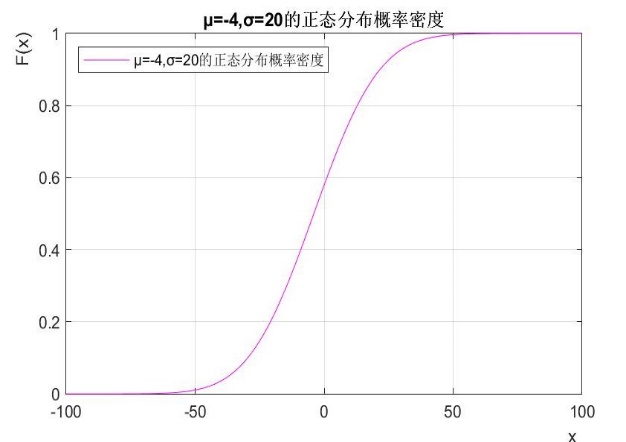
1. 最简单的：rejection sampling，思路很简单，也很容易实现，但效率较差
2. 较复杂的：inverse CDF，直接利用累积分布函数（CDF）的反函数生成随机数，但计算中牵扯到比较复杂的误差函数erf（非初等函数）
3. 更好的：Box-Muller算法，在很长时间内都是生成正态分布随机数的"标准"算法。Box-Muller算法的特点是效率高，并且计算过程比较简单（只用到了初等函数）
4. zigurat算法：效率远高于Box-Muller算法，适用于连续生成大量服从同一正态分布的随机数的情形。并不局限于生成正态分布随机数，只要一个连续分布的概率密度函数（PDF）图象是一条下降的曲线或是像正态分布这样的钟形曲线，理论上都可以使用ziggurat算法生成。

参考文献：<https://github.com/miloyip/normaldist-benchmark>

# 第二部分：

# 正态分布的分布函数及其概率密度图：

# 

1. T分布的分布函数及其概率密度：  
    