Lecture 4: 随机数产生方法

张伟平

Sunday 11$^{\text{th}}$ October, 2009

Contents

Chapter 1

Methods for Generating Random Variables

统计计算中一个基础问题就是从特定的概率分布中产生随机变量(随机数). 在最简单的情形, 从一个有限总体中产生一个随机观测, 就需要一种从离散均匀总体中产生随机观测的方法. 从而, 一个合适的均匀(伪)随机数产生器是从根本上所需要的, 从其他概率分布中产生随机数都依赖于均匀随机数产生器.

1.1 Generating Uniform(0,1) random number

线性同余发生器:

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

这里:

(1) m: 模$= 2^{31} - 1$(32位计算机可以直接表示的最大素数).

(2) a: 乘数, 要小心选择

(3) c: 增量(可以)=0.

(4) X_0: 初始值(种子).

- $X_n \in \{0, 1, \ldots, m-1\}$, $U_n = X_n/m$.
- 使用奇数作为种子.
- 代数和群理论有助于a的选择.
- 希望生成器的周期很大.
- 不要产生多与$m/1000$个数.

在R中, 使用帮助主题来了解**.Random.seed**或者**RNGkind**关于缺省均匀随机数 产生器的详细信息. 各种不同类型的随机数产生器及其性质可以参考一些数值计算方面的资料.

1.2 Random Generators of Common Probability Distribution in R

在 R 中均匀伪随机数的产生器是**runif**. 产生伪均匀随机数方式为

↑Example

```
runif{n}       #产生0到1上的长度为n的向量
runif{n,a,b}   #产生a到b上的长度为n的向量
matrix(runif(n*m),nrow=n,ncol=m) #产生0到1上的nxm的矩阵
```

↓Example

常用的一元概率分布的概率质量函数(**pmf**)或者概率密度函数(**pdf**), 累积分布函数(**cdf**), 分位数函数(**quantile function**)以及随机数 产生器函数在R中已经集成, 比如对二项分布, 可以参看帮助文档

↑Code

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
```

```
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)
```

↓Code

R中常见的一元分布函数

分布	cdf	随机数产生器	参数
beta	pbeta	rbeta	shape1, shape2
二项分布	pbinom	rbinom	size, prob
χ^2分布	pchisq	rchisq	df
指数分布	pexp	rexp	rate
F分布	pf	rf	df1,df2
gamma	pgamma	rgamma	shape,rate or scale
几何分布	pgeom	rgeom	prob
对数正态分布	plnorm	rlnorm	meanlog,sdlog
负二项分布	pnbinom	rnbinom	size, prob
正态分布	pnorm	rnorm	mean,sd
Poisson分布	ppois	rpois	lambda
t分布	pt	rt	df
均匀分布	punif	runif	min,max

使用**sample**函数从一个有限离散总体中抽样:

sample 可以实现从有限总体中有放回或者不放回两种抽样方式进行抽样.

↑Example

```
#掷硬币
sample(0:1,size=10,replace=TRUE)
#字母a-z的一个置换
sample(letters)
#多项分布抽样
x<-sample(1:3,size=100,replace=TRUE,prob=c(0.2,0.3,0.5))
#> table(x)
#x
# 1  2  3
#12 30 58
```

↓Example

1.2.1 The Inverse Transform Method

❑ 连续型场合

生成随机数的逆变换方法是基于以下熟知的定理

Theorem 1 (Probability Integral Transformation). 若X为连续型随机变量, 其*cdf* 为F_X, 则 $U = F_X^{-1}(X) \sim U(0,1)$.

Proof. 定义

$$F_X^{-1}(u) = \inf\{x : F_X(x) = u\}, \quad 0 < u < 1.$$

若随机变量$U \sim U(0,1)$, 则对所有$x \in \mathcal{R}$, 有

$$\begin{aligned} P(F_X^{-1}(U) \le x) &= P(\inf\{t : F_X(t) = U\} \le x) \\ &= P(U \le F_X(x)) = F_U(F_X(x)) = F_X(x). \end{aligned}$$

因此$F_X^{-1}(U)$和随机变量X同分布. □

从而, 若要产生X的一个随机观测x, 可以

1. 导出逆变换函数 $F_X^{-1}(u)$.
2. 从均匀分布$U(0,1)$中产生一个随机数u, 令$x = F_X^{-1}(u)$.

这种方法要求F的逆函数要容易求出.

例: 使用逆变换方法产生连续型密度$f(x) = 3x^2 I(0 < x < 1)$的随机观测值.

此处$F_X(x) = x^3, (0 < x < 1)$, 因此 $F_X^{-1}(u) = u^{1/3}$, 因此

↑Code

```
n<-1000
u<-runif(n)
x<-u^(1/3)
hist(x,prob=TRUE,main=expression(f(x)==3*x^2))
y<-seq(0,1,0.01)
```

```
lines(y,3*y^2)
```

↓Code

绘图中数学符号的表示更多内容参看帮助文档 **?plotmath**.

例: 使用逆变换方法产生指数分布的随机数.

$X \sim Exp(\lambda)$, 因此对$x > 0$, $F_X(x) = 1 - e^{-\lambda x}$, 从而 $F_X^{-1}(u) = -\frac{1}{\lambda} log(1-u)$. 注意到$U$和$1-U$同分布, 因此产生参数是$\lambda$的指数分布的长度为$n$的随机数命令为

```
-log(runif(n))/lmbda
```

↑Code

↓Code

在R中也可以使用**rexp**来产生.

❑离散型场合

设X为一离散型随机变量, 其可能取值记为

$$\cdots < x_{i-1} < x_i < x_{i+1} < \cdots$$

定义

$$F_X^{-1}(u) = \inf\{x : F_X(x) \geq u\}$$

则逆变换为$F_X^{-1}(u) = x_i$, 其中$F_X(x_{i-1}) < u \leq F_X(x_i)$. 从而产生一个随机数$x$的方式为

1. 从均匀分布$U(0,1)$中产生一个随机数u
2. 取$x = x_i$, 若$F_X(x_{i-1}) < u \leq F_X(x_i)$.

对某些分布来说, 计算$F_X(x_{i-1}) < u \leq F_X(x_i)$可能比较困难. 在离散型场合应用逆变换方法产生随机数的不同方式可以参考**Devroye**第三章.

例: 使用逆变换方法产生0-1分布的随机数.

此时$F_X(0) = f_X(0) = 1-p$以及$F_X(1) = 1$. 因此若$u > 1-p$, 则$F_X^{-1}(u) = 1$; 若$u \leq 1-p$, 则 $F_X^{-1}(u) = 0$. 因此

↑Code

```
n<-1000
p<-0.4
u<-runif(n)
x<-as.integer(u>1-p)
mean(x) #理论值 p
var(x)  #理论值p(1-p)
```

↓Code

在R中,我们使用**rbinom(n,1,p)**来产生0-1分布长为n的一个随机观测向量.

例: 使用逆变换方法产生几何分布的随机数.

由于几何分布的分布函数$F_X(x) = 1 - q^x, \quad x = 1, 2, \cdots$. 因此每个随机

数 都要计算

$$1 - q^{x-1} < u \leq 1 - q^{x}.$$

这个不等式等价于$x - 1 < log(1 - u)/log(q) \leq x$, 其解为$x = \lceil log(1 - u)/log(q) \rceil$, 这里$\lceil t \rceil$表示不小于$t$的最小整数(ceiling). 因此

↑Code

```
n<-1000
p<-0.4
u<-runif(n)
x<-ceiling(log(1-u)/log(1-p))
```

↓Code

注意到U和$1 - U$同分布, 以及$log(1 - U)/log(q)$取整数的概率为0, 从而上述代码最后一步可以等价为

↑Code

↓Code

```
k<-floor(log(u)/log(1-p))+1
```

在R中,我们使用**rgeom(n,p)**来产生参数为p的几何分布的长为n的一个随机观测向量.

例: 使用逆变换方法产生Poisson分布的随机数.

对Poisson分布来说, 求解$F(x-1) < u \leq F(x)$比起几何分布情形时要复杂的多. 产生Poisson分布 的基本方法是通过如下递推关系产生和存储分布函数:

$$f(x+1) = \frac{\lambda f(x)}{x+1}; \quad F(x+1) = F(x) + f(x+1).$$

从而可以通过存储下的分布函数序列中寻求$F(x-1) < u \leq F(x)$的解. R中可以通过**rpois(n,lambda)**来产生Poisson分布的长为n的一组观测向量.

例: 使用逆变换方法产生如下对数分布的随机数.

$$f(x) = P(X = x) = \frac{a\theta^x}{x}, \quad x = 1, 2, \cdots$$

其中$0 < \theta < 1$以及$a = (-log(1-\theta))^{-1}$.

显然, 递推关系为

$$f(x+1) = \frac{\theta^x}{x+1} f(x), \quad x = 1, 2, \cdots$$

理论上讲, $f(x)$可以通过如上递推关系求值, 但是对很大的x值这种计算方法的精度不够, 最终 会导致$f(x) = 0, F(x) < 1$. 因此我们通过$f(x)$的表达式$exp(log(a) + xlog(\theta) - log(x))$来 计算$f(x)$的值. 为求解不等式, 将$F(x)$的值存储下来, 初始可以选择一个大数$N$, 设置向量$F(x), x = 1, \cdots, N$, 然后在有必要的时候再增大$N$. 另外, 对特定的$u$求解$F(x-1) < u \leq F(x)$, 计算满足$F(x-1) < u$的$x$的个数. 当$F$存为向量时, $F < u_i$为一逻辑向量, 其值为TRUE或者FALSE, 而$u_i > F$的和即为$x - 1$. 所以

↑Code

```
rlogarithmic<-function(n,theta){
  #return a random logarithmic(theta) sample size n
  u<-runif(n)
  #set the initial length of cdf vector
  N<-ceiling(-16/log10(theta))
```

```
  k<-1:N
  a<--1/log(1-theta)
  fk<-exp(log(a)+k*log(theta)-log(k))
  Fk<-cumsum(fk)
  x<-integer(n)
  for(i in 1:n){
    x[i]<-as.integer(sum(u[i]>Fk))
    while(x[i]==N){
    logf<-log(a)+(N+1)*log(theta)-log(N+1)
    fk<-c(fk,exp(logf))
    Fk<-c(Fk,Fk[N]+fk[N+1])
    N<-N+1
    x[i]<-as.integer(sum(u[i]>Fk))
   }
  }
  x+1
}
```

↓Code

从而产生 logarithmic(0.5) 分布的随机数

↑Example

```
n<-1000
theta<-0.5
x<-rlogarithmic(n,theta)
#计算对数分布的分布律来对比
k<-sort(unique(x))
p<--1/log(1-theta)*theta^k/k
se<-sqrt(p*(1-p)/n)  #标准差
```

↓Example

下面的结果表明样本的频率分布符合理论分布:

↑Result

```
> round(rbind(table(x)/n,p,se),3)
       1     2     3     4     5     6     7     8     9
   0.720 0.185 0.058 0.018 0.012 0.003 0.002 0.001 0.001
p  0.721 0.180 0.060 0.023 0.009 0.004 0.002 0.001 0.000
se 0.014 0.012 0.008 0.005 0.003 0.002 0.001 0.001 0.001
```

↓Result

1.2.2 The Acceptance-Rejection Method

假设X和Y是随机变量, 其概率函数分别为f和g. 满足

$$\frac{f(t)}{g(t)} \leq c, \quad \forall \ t \ \ s.t. \ \ f(t) > 0$$

则舍选法(Acceptance-Rejection Method)可以用来生成X的随机数.

1. 找一个可以方便生成随机数的随机变量Y, 其概率函数g满足$f(t)/g(t) \leq c, \forall \ t \ s.t. \ f(t) > 0$.
2. 从g中产生一个随机数y.
3. 从均匀分布$U(0,1)$中产生一个随机数u.
4. 若$u < f(y)/(cg(y))$, 则接受$x = y$, 否则拒绝y. 重复2-4, 直至产生给定个数的x.

在离散型场合, 对每个使得$f(k) > 0$的k有

$$P(k|A) = \frac{P(A|k)g(k)}{P(A)} = \frac{[f(k)/(cg(k))]g(k)}{1/c} = f(k)$$

对连续型随机变量场合, 即需证明$P(Y \le y|U \le \frac{f(Y)}{cg(Y)}) = F_X(y)$. 事实上

$$\begin{aligned} P(Y \le y|U \le \frac{f(Y)}{cg(Y)}) &= \frac{P(U \le \frac{f(Y)}{cg(Y)}, Y \le y)}{1/c} \\ &= \int_{-\infty}^{y} \frac{P(U \le \frac{f(Y)}{cg(Y)}|Y = \omega \le y)}{1/c} g(\omega)d\omega \\ &= c\int_{-\infty}^{y} \frac{f(\omega)}{cg(\omega)} g(\omega)d\omega \\ &= F_X(y) \end{aligned}$$

舍选法的特点:

- 优点是计算时间不随着x增加;

- 缺点是寻求一个容易产生随机数且逼近f很好的分布g比较困难, 因此一个不好的g会导致接受概率很低,造成 运行时间过长.

例: 使用舍选法从如下分布产生随机数.

$$f(x) = 6x(1-x), \quad 0 < x < 1$$

从此分布中产生n个随机数需要循环的总数依赖于接受概率$1/c$的大小. 取$g(x)$为$U(0,1)$的密度, 则$c = 6$, 从而从g中产生的一个随机数x被接受, 除非

$$\frac{f(x)}{cg(x)} = x(1-x) > u$$

因此,平均来看需要cn次循环, 即$2cn$个随机数需要产生.

↑Code

```
n<-1000
j<-k<-0
y<-numeric(n)
while(k<n){
  u<-runif(1)
```

```
  j<-j+1
  x<-runif(1) #从g中产生一个随机数
  if(x*(1-x)>u) {
    k<-k+1
    y[k]<-x
    }
  }
#>j
#[1] 6153
```

↓Code

在这次模拟中, 需要6153次循环来产生$n = 1000$个需要的随机数. 比较其经验分位数和理论分位数

↑Example

```
p<-seq(.1,.9,.1)
Qhat<-quantile(y,p)
Q<-qbeta(p,2,2)
se<-sqrt(p*(1-p)/(n*dbeta(Q,2,2)))
round(rbind(Qhat,Q,se),3)
```

↓Example

当密度靠近0时需要大量的重复来估计分位数.

1.2.3 Transformation Methods

除前面介绍的方法外, 许多变换类型可以用来生成随机数. 比如

1. 若$Z \sim N(0,1)$, 则$Z^2 \sim \chi^2(1)$.
2. 若$U \sim \chi^2(m)$ 以及 $V \sim \chi^2(n)$, 则$F = \frac{U/m}{V/n} \sim F(m,n)$.
3. 若$Z \sim N(0,1)$ 以及 $V \sim \chi^2(n)$ 且相互独立, 则$T = \frac{Z}{\sqrt{V/n}} \sim t(n)$.
4. 若$U, V \sim U(0,1)$且相互独立, 则$Z_1 = \sqrt{-2logU}cos(2\pi V)$与$Z_2 = \sqrt{-2logV}sin(2\pi U)$ 相互独立的标准正态随机变量.

例: 使用变换法从产生对数分布的随机数.

注意到若$U, V \sim U(0,1)$且相互独立, 则

$$X = \left\lfloor 1 + \frac{logV}{log(1-(1-\theta)^U)} \right\rfloor$$

服从参数为θ的对数分布. 从而

1. 从$U(0,1)$中产生u
2. 从$U(0,1)$中产生v
3. 取$x = \lfloor 1 + log(v)/log(1-(1-\theta)^u) \rfloor$

实现代码为

↑Code

```
n<-1000
theta<-0.5
u<-runif(n)
v<-runif(n)
x<-floor(1+log(v)/log(1-(1-theta)^u))
```

```
k<-1:max(x)#计算对数分布的概率
p<--1/log(1-theta)*theta^k/k
se<-sqrt(p*(1-p)/n)
p.hat<-tabulate(x)/n
round(rbind(p.hat,p,se),3)
```

↓Code

相比于逆变换方法, 变换法更有效率(为什么?)

↑Code

```
rlogarithmic<-function(n,theta){
  stopifnot(all(theta>0 & theta <1))
  th<-rep(theta,length=n)
  u<-runif(n)
  v<-runif(n)
  x<-floor(1+log(v)/log(1-(1-theta)^u))
  return(x)
  }
```

↓Code

1.2.4 Sums and Mixtures

随机变量的和或者混合是一种特殊类型的变换. 比如

例: 使用随机变量的和产生χ^2分布的随机数.

ν个标准正态随机变量的平方和为$\chi^2(\nu)$分布随机变量, 因此

↑Code

```
n<-1000
nu<-2
X<-matrix(rnorm(n*nu),n,nu)^2
#方法1
y<-rowSum(X)
#方法2
y<-apply(X,MARGIN=1,FUN=sum)
```

↓Code

对随机变量的混合, 我们先看定义:

称一个随机变量为离散混合，如果其分布为某些随机变量$X_1, X_2, \cdots$分布的加权：

$$F_X(x) = \sum \theta_i F_{X_i}(x).$$

Definition

$\theta_i > 0$且$\sum \theta_i = 1$为权重.

而对连续型随机变量, 类似的有

称一个随机变量为连续混合，如果其分布为某个分布族$X|Y = y$的加权：

$$F_X(x) = \int F_{X|Y=y}(x) f_Y(y) dy.$$

Definition

其中$\int f_Y(y)dy = 1$.

例: 产生如下混合分布的随机数.

$$F_X(x) = pF_{X_1}(x) + (1-p)F_{X_2}(x)$$

产生此随机数的方法显然

1. 产生一个整数$k \in \{1, 2\}$, 这里$P(1) = p, P(2) = 1 - p$.

2. 若$k = 1$, 则从F_{X_1}中产生x_1, 并令$x = x_1$.

3. 若$k = 2$, 则从F_{X_2}中产生x_2, 并令$x = x_2$.

例: Γ分布的混合.

$$F_X = \sum_{i=1}^{5} \theta_j F_{X_j}$$

其中$X_j \sim Gamma(r = 3, \lambda_j = 1/j)$相互独立, $\theta_j = j/15,\ j = 1, \cdots, 5.$

```
n<-1000
k<-sample(1:5,size=n,replace=TRUE,prob=(1:5)/15)
rate<-1/k
x<-rgamma(n,shape=3,rate=rate)
#画混合的密度以及分量的密度
plot(density(x),xlim=c(0,40),ylim=c(0,.3),lwd=3,xlab="x",main="")
for(i in 1:5)
lines(density(rgamma(n,3,1/i)))
```

↓Code

例: Γ密度的混合.

$$f(x) = \sum_{j=1}^{5} \theta_j f_j(x), x > 0$$

其中f_j为$Gamma(3, \lambda_j)$的密度. 为画此混合密度的图形, 需要计算其值

↑Code

```
f<-function(x,lambda,theta){
 sum(dgamma(x,3,lambda)*theta)
 }
```

↓Code

这里$dgamma(x,3,lambda)*theta$是向量$(\theta_1 f_1(x),\cdots,\theta_5 f_5(x))$.

↑Code

```
x<-seq(0,8,length=200)
dim(x)<-length(x) #使用apply函数需要
p<-c(.1,.2,.2,.3,.2)
lambda<-c(1,1.5,2,2.5,3)
#计算混合密度在x处的值
y<-apply(x,1,f,lambda=lambda,theta=p)
plot(x,y,type="l",ylim=c(0,.85),lwd=3,ylab="Density")
for(j in 1:5){
 y<-apply(x,1,dgamma,shape=3,rate=lambda[j])
 lines(x,y)
 }
```

↓Code

例: $Poisson - Gamma$混合.

若$X|\Lambda = \lambda \sim Pois(\lambda), \Lambda \sim Gamma(r, \beta)$, 则 $X \sim NB(r, p = \beta/(1 + \beta))$. 本例说明一个Poisson-Gamma混合,并和负二项 分布的样本相比较.

↑Code

```
#Poisson-Gamma Mixture
n<-1000
r<-4
beta<-3
lambda<-rgamma(n,r,beta) #lambda是随机的
x<-rpois(n,lambda)
#compare with negative binomial
mix<-tabulate(x+1)/n
negbin<-round(dnbinom(0:max(x),r,beta/(1+beta)),3)
se<-sqrt(negbin*(1-negbin)/n)
round(rbind(mix,negbin,se),3)
```

↓Code

1.3 Multivariate Distribution

1.3.1 Multivariate Normal Distribution

一个随机向量$X = (X_1, \cdots, X_d)$服从d元正态分布$N_d(\mu, \Sigma)$,如果其联合密度有形式

$$f(x) = (2\pi)^{-d/2}|\Sigma|^{-1/2}exp\{-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\}, \quad x \in \mathcal{R}^d.$$

产生$N_d(\mu, \Sigma)$的一个随机数, 一般通过两步实现:

1. 产生$i.i.d.$的标准正态分布随机变量$Z_1, \cdots, Z_d$.

2. 将$Z = (Z_1, \cdots, Z_d)$变换为$X = (X_1, \cdots, X_d)$使其期望为μ,协方差为Σ:

$$X = CZ + \mu,$$

这里$CC' = \Sigma$. 分解Σ可以是谱分解方法(**eigen**), Choleski分解(**chol**)或者奇异值分解(**svd**).

一般不会对随机向量的一个样本进行一次线性变换, 经常的是要对所有的样本组成的矩阵进行变换. 假设$Z = (Z_{ij})$是一个$n \times d$的矩阵, 其中Z_{ij} iid $N(0,1)$. 则Z的行是n个d元标准正态随机变量的观测. 则此时需要进行的变换是

$$X = ZQ + J\mu^T,$$

其中$Q^TQ = \Sigma$, $J = J_{n\times 1} = (1, \cdots, 1)^T$. 则$X$的行是$N_d(\mu, \Sigma)$的随机数. 从而总结如下

1. 产生一个有标准正态分布随机数构成的$n \times d$矩阵Z.

2. 计算分解$\Sigma = Q^TQ$.

3. 应用变换 $X = ZQ + J\mu^T$.

其中$X = ZQ + J\mu^T$在R中可以如下实现

↑Code

```
Z<-matrix(rnorm(n*d),nrow=n,ncol=d)
X<-Z%*%Q+matrix(mu,n,d,byrow=TRUE)
```

↓Code

生成$N_d(\mu, \Sigma)$随机数的谱分解方法 由于

$$\Sigma^{1/2} = P\Lambda^{1/2}P^T$$

所以

↑Code

```
mu <- c(0, 0)
Sigma <- matrix(c(1, .9, .9, 1), nrow = 2, ncol = 2)
rmvn.eigen <-
function(n, mu, Sigma) {
    # generate n random vectors from MVN(mu, Sigma)
    # dimension is inferred from mu and Sigma
    d <- length(mu)
    ev <- eigen(Sigma, symmetric = TRUE)
    lambda <- ev$values
```

```
    V <- ev$vectors
    R <- V %*% diag(sqrt(lambda)) %*% t(V)
    Z <- matrix(rnorm(n*d), nrow = n, ncol = d)
    X <- Z %*% R + matrix(mu, n, d, byrow = TRUE)
    X
}
# generate the sample
X <- rmvn.eigen(1000, mu, Sigma)
plot(X, xlab = "x", ylab = "y", pch = 20)
print(colMeans(X))
print(cor(X))
```

↓Code

生成$N_d(\mu, \Sigma)$随机数的奇异值分解方法 根据奇异值分解易知$\Sigma^{1/2} = UD^{1/2}V^T$. 所以

↑Code

```
rmvn.svd <-
   function(n, mu, Sigma) {
       # generate n random vectors from MVN(mu, Sigma)
       # dimension is inferred from mu and Sigma
```

```
    d <- length(mu)
    S <- svd(Sigma)
    R <- S$u %*% diag(sqrt(S$d)) %*% t(S$v) #sq. root Sigma
    Z <- matrix(rnorm(n*d), nrow=n, ncol=d)
    X <- Z %*% R + matrix(mu, n, d, byrow=TRUE)
    X
}
```

↓Code

生成$N_d(\mu, \Sigma)$随机数的**Choleski分解方法**

↑Code

```
rmvn.Choleski <-
   function(n, mu, Sigma) {
       # generate n random vectors from MVN(mu, Sigma)
       # dimension is inferred from mu and Sigma
       d <- length(mu)
       Q <- chol(Sigma) # Choleski factorization of Sigma
       Z <- matrix(rnorm(n*d), nrow=n, ncol=d)
       X <- Z %*% Q + matrix(mu, n, d, byrow=TRUE)
       X
   }
```

↓Code

比较各种生成器的性能

我们已经讨论了产生随机数的不同方法, 那么哪种方法更好呢? 一种考虑可以时间复杂性, 别的方面的考虑可以是根据模拟的目的是估计一个或多个参数, 或者是估计量的方差等, 进行 评估(在以后的课程中将涉及到).

这里我们使用**system.time**来评估各个生成器的时间复杂性.

↑Code

```
library(MASS)
library(mvtnorm)
n <- 100          #sample size
d <- 30           #dimension
N <- 2000         #iterations
mu <- numeric(d)

set.seed(100)
system.time(for (i in 1:N)
```

```
    rmvn.eigen(n, mu, cov(matrix(rnorm(n*d), n, d))))
set.seed(100)
system.time(for (i in 1:N)
    rmvn.svd(n, mu, cov(matrix(rnorm(n*d), n, d))))
set.seed(100)
system.time(for (i in 1:N)
    rmvn.Choleski(n, mu, cov(matrix(rnorm(n*d), n, d))))
set.seed(100)
system.time(for (i in 1:N)
    mvrnorm(n, mu, cov(matrix(rnorm(n*d), n, d))))
set.seed(100)
system.time(for (i in 1:N)
    rmvnorm(n, mu, cov(matrix(rnorm(n*d), n, d))))
set.seed(100)
system.time(for (i in 1:N)
    cov(matrix(rnorm(n*d), n, d)))

detach(package:MASS)
detach(package:mvtnorm)
```

↓Code

在多元正态分布随机数生成过程中, 大部分是工作是分解协方差矩阵. 这里使用 的协方差矩阵是单位阵的样本协方差矩阵, 因此, 随机产生的协方差矩阵Σ在每次循环时是变化的, 但是Σ非常接近单位阵. 为了在同一个协方差矩阵下比较各种不同的方法, 每次运行后都将随机数种子恢复. 最后一次运行仅仅是产生 协方差矩阵, 以和总时间比较.

1.3.2 Mixtures of Multivariate Normals

多元正态的混合为

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2)$$

产生此混合分布n个随机向量:

↑Code

```
library(MASS)  #for mvrnorm
loc.mix <- function(n, p, mu1, mu2, Sigma) {
        #generate sample from BVN location mixture
        n1 <- rbinom(1, size = n, prob = p)
```

```
        n2 <- n - n1
        x1 <- mvrnorm(n1, mu = mu1, Sigma)
        x2 <- mvrnorm(n2, mu = mu2, Sigma)
        X <- rbind(x1, x2)              #combine the samples
        return(X[sample(1:n), ])        #mix them
    }
```

↓Code

用此程序产生1000个4维正态分布随机向量:

↑Example

```
x <- loc.mix(1000, .5, rep(0, 4), 2:5, Sigma = diag(4))
    r <- range(x) * 1.2
    par(mfrow = c(2, 2))
    for (i in 1:4)
        hist(x[ , i], xlim = r, ylim = c(0, .3), freq = FALSE,
        main = "", breaks = seq(-5, 10, .5))
```

↓Example

1.3.3 Wishart Distribution

若$M = X^T X$, X为从$N_d(0, \Sigma)$中抽取的$n \times d$随机矩阵, 则M 服从Wishart分布$W_d(\Sigma, n)$. 当$d = 1$时, $W_1(\sigma^2, n) = \sigma^2 \chi^2(n)$.

显然, 从Wishart分布中产生随机数可以通过如下方式:

1. 从$N_d(0, \Sigma)$中生成X
2. 令$W = X^T X$.

这种方法是种低效率的方法. 这是由于必须产生nd个随机数来决定$d(d+1)/2$个元素值. 一种效率更高的方法是基于$Bartlett$分解: 令$T = (T_{ij})$为一$d \times d$的下三角矩阵, 其元素满足

1. T_{ij} $i.i.d \sim N(0, 1)$, $i > j$.
2. $T_{ii} \sim \sqrt{\chi^2(n - i + 1)}, i = 1, \cdots, d.$

则矩阵$A = TT^T$服从$W_d(I_d, n)$. 因此生产$W_d(\Sigma, n)$可以通过Σ的Choleski分解$\Sigma = LL^T$, 则$LAL^T \sim W_d(\Sigma, n)$.

1.3.4 Uniform Distribution on the $d-$Sphere

$d-$球面即集合$\{x \in \mathcal{R}^d : \|x\|^2 = 1\}$. 在$d-$球面上均匀分布的随机向量有相同的可能方向. 产生此随机数可以基于如下性质:

若$X_1, \cdots, X_d$ $i.i.d$ $N(0,1)$, 则$(U_1, \cdots, U_d)$服从$\mathcal{R}^d$中单位球面上的均匀分布, 其中

$$U_j = \frac{X_j}{\|x\|}, \ j = 1, \cdots, d. \tag{1.1}$$

算法如下: 对每个u_i, 重复

1. 从$N(0,1)$中产生随机数$x_{i1}, \cdots, x_{id}$
2. 计算欧式模$\|x\| = (x_{i1}^2 + \cdots + x_{id}^2)^{1/2}$.
3. 令$u_{ij} = x_{ij}/\|x\|$, $u_i = (u_{i1}, \cdots, u_{id})$.

在R中可以通过如下方式提高效率:

1. 产生nd个正态随机数构成$n \times d$维矩阵M, 其第i行对应u的第i个随机数向量.

2. 对每一行计算(1.1),把此n个模值存在向量L中.

3. 对每个数$M[i,j]$除以$L[i]$, 得到U.

实现代码如下

Code

```
runif.sphere <- function(n, d) {
       # return a random sample uniformly distributed
       # on the unit sphere in R ^d
       M <- matrix(rnorm(n*d), nrow = n, ncol = d)
       L <- apply(M, MARGIN = 1,
                  FUN = function(x){sqrt(sum(x*x))})
       D <- diag(1 / L)
       U <- D %*% M
```

```
        U
    }
```

↓Code

画出200个2维圆周上的均匀分布随机数散点图:

↑Example

```
X <- runif.sphere(200, 2)
par(pty = "s")
plot(X, xlab = bquote(x[1]), ylab = bquote(x[2]))
par(pty = "m")
```

↓Example

1.4 Stochastic Process

齐次Poisson过程

为通过模拟来研究计数过程, 我们可以产生 过程的一个有限时间的路径.

一个计数过程$\{N(t), t \geq 0\}$为速率是λ的齐次Poisson过程, 如果

1. $N(t)$有独立增量性
2. $P(N(s+t) - N(s) = n) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$.

对齐次Poisson过程, 事件的来到间隔$T_1, T_2, \cdots$是$i.i.d$的参数为λ 的指数分布随机变量. 因此, 产生齐次Poisson过程的方法产生来到间隔时间$T_1, \cdots, T_n, \cdots$, 则第n个时间的来到时刻为$S_n = T_1 + \cdots + T_n$, 序列

$\{T_n\}_{n=1}^{\infty}$ 或者 $\{S_n\}_{n=1}^{\infty}$ 即可以表示过程的一个路径.

另外一种方式是利用$S_1, \cdots, S_n | N(t) = n$同分布于从$U(0,1)$中抽取的$n$个简单样本的次序统计量. 过程在给定时刻$t$处的状态等于在$[0,t]$内到达的数目,$\min\{k : S_k > t\} - 1$. 即 若$S_n$是最小的到达时间超过$t$, 则$N(t) = n - 1$. 模拟一个齐次Poisson过程在$[0, t_0]$区间上状态算法如下

1. 设置$S_1 = 0$
2. 对满足$S_j \leq t_0$的 $j = 1, 2, \ldots,$
   (a) 产生$T_j \sim exp(\lambda)$.
   (b) 令$S_j = T_1 + \cdots + T_j$.
3. $N(t_0) = \min_j(S_j > t0) - 1$.

比如, 要产生速率为2的齐次Poisson过程在$[0,3]$内的状态,

↑Code

```
lambda <- 2
t0 <- 3
```

```
Tn <- rexp(100, lambda)        #interarrival times
Sn <- cumsum(Tn)               #arrival times
n <- min(which(Sn > t0))-1     #arrivals in [0, t0]
```

↓Code

使用另外一种方式就是

↑Code

```
lambda <- 2
t0 <- 3
upper <- 100
pp <- numeric(10000)
for (i in 1:10000) {
    N <- rpois(1, lambda * upper)
    Un <- runif(N, 0, upper)      #unordered arrival times
    Sn <- sort(Un)                #arrival times
    n <- min(which(Sn > t0))      #arrivals+1 in [0, t0]
    pp[i] <- n - 1                #arrivals in [0, t0]
    }
```

↓Code

由Poisson分布的性质, $N(3)$的期望和方差都应该是6:

```
c(mean(pp), var(pp))
```

非齐次 Poisson 过程

一个计数过程$\{N(t), t \geq 0\}$为强度是$\lambda(t)$的Poisson过程, 如果

1. $N(t)$有独立增量性
2. $P(N(s+t) - N(s) = n) = \frac{(\tilde{m}(s,t))^n}{n!} e^{-\tilde{m}(s,t)}$, 其中$\tilde{m}(s,t) = \int_s^{s+t} \lambda(y)dy$.

如果一个非齐次Poisson过程的强度函数有界, 即$\lambda(t) \leq \lambda_0 < \infty$,, 则可以通过 从一个齐次Poisson过程抽样产生. 这是因为从强度为λ的齐次Poisson过程中 抽样, 使得在时刻t发生的事件以$\lambda(t)/\lambda$被接受(计数), 最后得到的计数过程就服从强度为$\lambda(t)$的非齐次 Poisson过程. 产生$[0, t_0]$的非齐次Poisson过程可以通过此想法实现. 先找一个$\lambda_0 < \infty$, 使得$\lambda(t) \leq \lambda_0$, $\forall 0 \leq t \leq t_0$. [为什么?]

然后从 强度为λ_0的齐次Poisson过程中产生事件的来到时刻$\{S_j\}$, 然后以概率$\lambda(S_j)/\lambda_0$ 接受每个S_j. 算法如下

1. 设置$S_1 = 0$
2. 对满足$S_j \le t_0$的 $j = 1, 2, \ldots,$
   (a) 产生$T_j \sim exp(\lambda_0)$, 令$S_j = T_1 + \cdots + T_j$.
   (b) 产生均匀随机数$U_j \sim U(0, 1)$.
   (c) 若$U_j \le \lambda(S_j)/\lambda_0$, 则接受$S_j$, 令$I_j = 1$, 否则令$I_j = 0$.
3. 返回达到时间 $\{S_j : I_j = 1\}$.

比如$\lambda(t) = 3cos^2(t)$, 则$\lambda(t) \le 3 = \lambda_0$. 因此第$j$个到达时间当$U_j \le 3cos^2(S_j)/3 = cos^2(S_j)$时 被接受.

↑Example

```
lambda <- 3
upper <- 100
```

```
N <- rpois(1, lambda * upper)
Tn <- rexp(N, lambda)
Sn <- cumsum(Tn)
Un <- runif(N)
keep <- (Un <= cos(Sn)^2)    #indicator, as logical vector
Sn[keep]
round(Sn[keep], 4)
```

↓Example

而过程在时刻t处的状态即为

```
sum(Sn[keep]<=t)
```

更新过程

一个计数过程$\{N(t), t \geq 0\}$为更新过程, 如果 到达时间间隔$T_1, T_2, \ldots$是$i.i.d$的.

例: 假设到达时间间隔服从参数为p的几何分布, 即达到时间间隔为非负整数, 因此 $S_j = T_1 + \cdots + T_j$服从负二项分布,参数为j和p, 因此

↑Example

```
t0 <- 5
Tn <- rgeom(100, prob = .2)    #interarrival times
Sn <- cumsum(Tn)               #arrival times
n <- min(which(Sn > t0))       #arrivals+1 in [0, t0]
```

↓Example

$N(t_0)$的分布可以通过重复如上过程来估计

↑Example

```
Nt0 <- replicate(1000, expr = {
        Sn <- cumsum(rgeom(100, prob = .2))
        min(which(Sn > t0)) - 1
```

```
    })
table(Nt0)/1000
Nt0
```

↓Example

而均值函数$EN(t)$可以通过如下估计

↑Example

```
t0 <- seq(0.1, 30, .1)
mt <- numeric(length(t0))
for (i in 1:length(t0)) {
    mt[i] <- mean(replicate(1000,
    {
    Sn <- cumsum(rgeom(100, prob = .2))
    min(which(Sn > t0[i])) - 1
    }))
```

```
}
plot(t0, mt, type = "l", xlab = "t", ylab = "mean")
abline(0, .25)
```

↓Example

对称的随机游动

设$X_1, X_2, \cdots$是一列$i.i.d$的$B(1, 0.5)$随机变量，令$S_n = \sum_{i=1}^{n} X_i$，以及$S_0 = 0$，则过程$\{S_n, n \geq 0\}$称为对称的随机游动.

产生对称随机游动过程很简单

```
n <- 400
incr <- sample(c(-1, 1), size = n, replace = TRUE)
S <- as.integer(c(0, cumsum(incr)))
```

```
plot(0:n, S, type = "l", main = "", xlab = "i")
```

若初始$S_0 = 0$, 过程在时间$[1, 400]$内返回0的次数为`which(S==0)`.

如果需要的是过程S_n在时间n时的状态, 而不是历史值, 对比较大的n, 可以通过如下方式来 高效率的产生:

假设$S_0 = 0$为过程的初始状态, 若过程在时间n之前返回到0, 则在产生S_n的时候, 我们可以 忽略过程在最近一次返回到0的时刻之前的时间. 令T为过程首次返回到0的时刻, 则为产生S_n, 可以简化此问题为先产生等待时间T, 直至总时间(T之和)首次超过n, 然后从n时刻之前的最近一次返回到0的时刻开始, 产生增量X_i, 然后将它们相加.

T 的分布为

$$p_{2n} = P(T = 2n) = C_{2n-2}^{n-1} \frac{1}{n2^{2n-1}} = \frac{\Gamma(2n-1)}{n2^{2n-1}\Gamma^2(n)}, \quad n \geq 1.$$

$$P(T = 2n+1) = 0, \quad n \geq 0.$$

则算法如下:

令W_j为第j次返回初始位置需要的等待时间,

1. 设置$W_1 = 0$
2. 对满足$W_j \leq n$的 $j = 1, 2, \ldots,$
   (a) 从时间的分布中产生T_j, 直至首次返回到0.
   (b) 令$W_j = T_1 + \cdots + T_j$.
3. 令$t_0 = W_j - T_j$
4. 设置$s_1 = 0$.
5. 产生从$t_0 + 1$到时刻n的增量. 对$i = 1, 2, \ldots, n - t_0$
   (a) 产生随机增量$x_i \sim P(X = \pm 1) = 1/2$.
   (b) 设置$s_i = x_1 + \cdots + x_i$.
   (c) 若$s_i = 0$, 设置计数器为$i = 1$.
6. 返回s_i.

从T的分布中高效率的产生随机数的方法参见 Devroye, p.754. 这里我们使用一个简单易行但效率不高的方法. 注意到

$$p_{2n} = \frac{1}{2n} P(X = n-1), \quad X \sim B(2n-2, 1/2)$$

因此直接计算

↑Example

```
 #compute the probabilities directly
n <- 1:10000
p2n <- exp(lgamma(2*n-1)
          - log(n) - (2*n-1)*log(2) - 2*lgamma(n))
```

↓Example

或者使用二项分布以及逆变换方法计算

↑Example

```
P2n <- (.5/n) * dbinom(n-1, size = 2*n-2, prob = 0.5)
 pP2n <- cumsum(P2n)
#利用逆变换方法产生一个T
# u<-runif(1)
# Tj<-2*(1+sum(u>pP2n))
```

↓Example

现在假设给定n,我们要计算在$(0, n]$内最后一次返回到0的时间, 则

↑Example

```
#given n compute the time of the last return to 0 in (0,n]
n <- 200
sumT <- 0
while (sumT <= n) {
    u <- runif(1)
```

```
    s <- sum(u > pP2n)
    if (s == length(pP2n))
        warning("T is truncated")
    Tj <- 2 * (1 + s)
    #print(c(Tj, sumT))
    sumT <- sumT + Tj
    }
sumT - Tj
```

↓Example