统计计算 张楠 2019秋: Methods for Generating Random Vectors

(返回 统计计算 张楠 2019秋)

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

目录

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

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

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

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

```
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))
```

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

```
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
    }
```

生成$N_d(\mu, \Sigma)$随机数的Choleski分解方法

```
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
    }
```

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).

这里我们使用来评估各个生成器的时间复杂性.

```
    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)
```

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.

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:

```
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
    }
```

用此程序产生1000个4维正态分布随机向量:

```
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))
```

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$LAL^T\sim W_d(\Sigma,n)$.

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$U_j=\frac{Xj}{\|x\|}$, $j=1,\cdots,d.$
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]),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

```
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
   }
```

画出200个2维圆周上的均匀分布随机数散点图:

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