Chapter 3: Methods for Generating Random Variables

Lecturer: Zhao Jianhua

Department of Statistics
Yunnan University of Finance and Economics

Outline

3.1 Introduction

Random Generators of Common Probability Distributions in R

3.2 The Inverse Transform Method

3.2.1 Inverse Transform Method, Continuous Case

3.2.2 Inverse Transform Method, Discrete Case

3.3 The Acceptance-Rejection Method

The Acceptance-Rejection Method

3.4 Transformation Methods

3.5 Sums and Mixtures

3.6 Multivariate Distributions

3.6.1 Multivariate Normal Distribution

3.6.2 Mixtures of Multivariate Normals

3.6.3 Wishart Distribution

3.6.4 Uniform Dist. on the d-Sphere

Introduction

- One of the fundamental tools required in computational statistics is the ability to simulate random variables (r.v.) from specified probability (prob.) distributions (dist.).
- A suitable generator of uniform pseudo random numbers is essential. Methods for generating r.v. from other prob. dist. all depend on the uniform random number generator.

`runif`: Generating Uniform random number.

```
runif(n) #generate a vector of size n in [0,1]
runif(n,a,b) #generate n Uniform(a, b) numbers
matrix(runif(n*m),n,m) #generate n by m matrix
```

The `sample` function can be used to sample from a finite population, with or without replacement.

Example 3.1 (Sampling from a finite population)

```
> #toss some coins
> sample(0:1, size = 10, replace = TRUE)
> #choose some lottery numbers
> sample(1:100, size = 6, replace = FALSE)
> #permuation of letters a-z
> sample(letters)
> #sample from a multinomial dist.
> x=sample(1:3,size=100,replace=TRUE,prob=c(.2,.3,.5))
> table(x)
x
 1  2  3
17 35 48
```

Random Generators of Common Prob. Dist. in R

The prob. mass function (pmf) or density (pdf), cumulative dist. function (cdf), quantile function, and random generator of many commonly used prob. dist. are available. For example:

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

Table: Selected Univariate Prob. Functions Available in R

Distribution	cdf	Generator	Parameters
beta	pbeta	rbeta	shape1, shape2
binomial	pbinom	rbinom	size, prob
chi-squared	pchisq	rchisq	df
exponential	pexp	rexp	rate
F	pf	rf	df1, df2
gamma	pgamma	rgamma	shape, rate or scale
geometric	pgeom	rgeom	prob
lognormal	plnorm	rlnorm	meanlog, sdlog
negative binomial	pnbinom	rnbinom	size, prob
normal	pnorm	rnorm	mean, sd
Poisson	ppois	rpois	lambda
Students t	pt	rt	df
uniform	punif	runif	min, max

The Inverse Transform Method

Theorem: If X is a continuous r.v. with CDF $F_X(x)$, then $U = F_X(X) \sim U(0,1)$.

Proof: $F_X(x) = P(X \leq x) = P(F_X(X) \leq F_X(x)) = P(U \leq F_X(x))$. Second $=$ is because $F_X(x)$ is an increasing function. By the definition of uniform CDF, we have $U \sim U(0,1)$.

Define: the inverse transformation $F_X^{-1}(u) = \inf\{x : F_X(x) = u\}$, $0 < u < 1$. If $U \sim U(0,1)$, then for all $x \in \mathbf{R}$,

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

and therefore $F_X^{-1}(U)$ has the same dist. as X.

To generate a random observation X,

1. Dreive the inverse function $F_X^{-1}(u)$
2. generate a random u from $U(0,1)$.
3. deliver $x = F_X^{-1}(u)$.

The method easy to apply, provided that F_X^{-1} is easy to compute.

Inverse Transform Method, Continuous Case

Example 3.2 (Inverse transform method, continuous case)

Simulate a random sample from the dist. with density $f_X(x) = 3x^2, 0 < x < 1$.

Here $F_X(x) = x^3$ for $0 < x < 1$, and $F_X^{-1}(u) = u^{1/3}$, then

```
n <- 1000 ; u <- runif(n) ; x <- u^(1/3)
hist(x, prob = TRUE) #density histogram of sample
y <- seq(0,1,.01); lines(y,3*y^2) #density curve f(x)
```

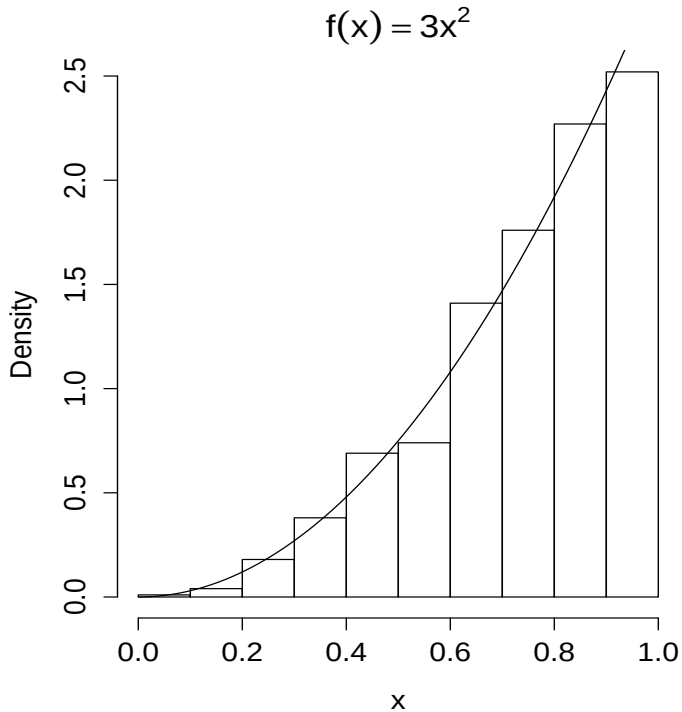


Fig.3.1: Prob. density histogram with theoretical density $f(x) = 3x^2$ superimposed.

R note 3.1

In Figure 3.1, the title includes a math expression, obtained by

```
hist(x, prob = TRUE, main = expression(f(x)==3*x^2)).
```

Alternately, `main = bquote(f(x)==3*x^2))` produces the same title. Math annotation is covered in the help topic for `plotmath`. Also see the help topics for `text` and `axis`.

Example 3.3 (Exponential dist.)

Generate a random sample from the exponential dist.
$X \sim \text{Exp}(\lambda)$, for $x > 0$, the cdf of X is $F_X(x) = 1 - e^{-\lambda x}$, then $F^{-1}(u) = -\frac{1}{\lambda}\log(1-u)$. Note that U and $1-U$ have the same dist. and it is simpler to set $x = -\frac{1}{\lambda}\log(u)$. To generate a random sample of size `n` with parameter `lambda`:

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

A generator `rexp` is available in R.

Inverse Transform Method, Discrete Case

If X is a discrete r.v. and

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

are the points of discontinuity of $F_X(x)$.
Define:

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

then the inverse transformation is $F_X^{-1}(u) = x_i$, where $F_X(x_{i-1}) < u \leq F_X(x_i)$.

For each random variate required:

1. Generate a random u from $U(0, 1)$.
2. Deliver x_i where $F(x_{i-1}) < u \leq F(x_i)$.

The solution of $F_X(x_{i-1}) < u \leq F_X(x_i)$ may be difficult for some dist..

Example 3.4 (Two point distribution)

Generate a random sample of Bernoulli($p = 0.4$) variates.
In this example, $F_X(0) = f_X(0) = 1 - p$ and $F_X(1) = 1$. Thus, $F_X^{-1}(u) = 1$, if $u > 0.6$ and $F_X(u) = 0$ if $u \leq 0.6$,

```
n <- 1000 ; p<-0.4 ; u <- runif(n)
x <- as.integer(u > 0.6) #(u > 0.6) is a logical vector
> mean(x);var(x)
[1] 0.41   0.2421421
```

Sample mean and variance should be approximately $p = 0.4$ and $p(1 - p) = 0.24$.

R note 3.2

`rbinom` (random binomial) function with `size=1` generates a Bernoulli sample. Another method is to sample from the vector $(0, 1)$ with prob. $(1 - p, p)$.

```
rbinom(n, size = 1, prob = p)
sample(c(0,1),size=n,replace=TRUE,prob = c(.6,.4))
```

Example 3.5 (Geometric distribution)

Generate a random geometric sample with parameter $p = \frac{1}{4}$.
The pmf is $f(x) = pq^x$, $x = 0, 1, 2, \cdots$, where $q = 1 - p$. At the points of discontinuity $x = 0, 1, 2, \cdots$, cdf is $F(x) = 1 - q^{x+1}$. For each sample element, we generate a random uniform u and solve

$$1 - q^x < u \leq 1 - q^x,$$

which simplifies to $x < \log(1-u)/\log(q) \leq x + 1$. Thus $x + 1 = \lceil \log(1-u)/\log(q) \rceil$, where $\lceil t \rceil$ denotes ceiling function.

```
n <- 1000
p <- 0.25
u <- runif(n)
k <- ceiling(log(1-u) / log(1-p)) - 1
```

U and $1 - U$ have the same dist. and the prob. that $\log(1-u)/\log(1-p)$ equals an integer is zero, and thus the last step is

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

Poisson distribution

The basic method to generate a Poisson(λ) variate is to generate and store the cdf via the recursive formula

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

For each Poisson variate, a random uniform u is generated, and the cdf vector is searched for the solution to $F(x-1) < u \leq F(x)$.

Note: R function `rpois` generates random Poisson samples.

Example 3.6 (Logarithmic distribution)

Simulate a Logarithmic(θ) random sample

$$f(x) = P(X = x) = \frac{a\theta^x}{x}, x = 1, 2, \cdots \tag{3.1}$$

where $0 < \theta < 1$ and $a = (-\log(1-\theta))^{-1}$. A recursive formula for $f(x)$ is

$$f(x+1) = \frac{\theta x}{x+1} f(x), x = 1, 2, \cdots \tag{3.2}$$

Generate random samples from Logarithmic(0.5) dist.

```
n <- 1000 ; theta <- 0.5 ; x <- rlogarithmic(n, theta)
#compute density of logarithmic(theta) for comparison
k<-sort(unique(x)); p=-1/log(1-theta)*theta^k/k
se<-sqrt(p*(1-p)/n) #standard error
```

Sample relative frequencies (line 1) match theoretical dist. (line 2) of Logarithmic(0.5) dist. within two standard errors.

```
> round(rbind(table(x)/n, p, se),3)
       1     2     3     4     5     6     7
   0.741 0.169 0.049 0.026 0.008 0.003 0.004
p  0.721 0.180 0.060 0.023 0.009 0.004 0.002
se 0.014 0.012 0.008 0.005 0.003 0.002 0.001
```

`rlogarithmic`

- Initially choose a length N for the cdf vector, and compute $F(x), x = 1, 2, \cdots, N$. If necessary, N will be increased.

```
rlogarithmic <- function(n, theta) {
  #returns 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 <- a*theta^k/k
  Fk <- cumsum(fk); x <- integer(n)
  for (i in 1:n) {
    x[i] <- as.integer(sum(u[i] > Fk)) #F^{-1}(u)-1
    while (x[i] == N) {
      #if x==N we need to extend the cdf
      #very unlikely because N is large
      f <- a*theta^(N+1)/N
      fk <- c(fk, f)
      Fk <- c(Fk, Fk[N] + fk[N+1])
      N <- N + 1
      x[i] <- as.integer(sum(u[i] > Fk))}}
  x + 1 }
```

The Acceptance-Rejection Method

Suppose that X and Y are r.v. with density or pmf f and g respectively, and there exists a constant c such that $\frac{f(t)}{g(t)} \leq c$ for all t such that $f(t) > 0$. Then the acceptance-rejection method can be applied to generate the r.v. X.

The acceptance-rejection method

1. Find a r.v. Y with density g satisfying $f(t)/g(t) \leq c$, for all t such that $f(t) > 0$.
2. Generate a random y from the dist. with density g
3. Generate a random u from the $U(0,1)$ dist.
4. If $u < f(y)/(cg(y))$ accept y and deliver $x = y$; otherwise reject y and repeat step 2-4.

Note that in step 4, $P(\text{accept}|Y) = P(U < \frac{f(Y)}{cg(Y)}|Y) = \frac{f(Y)}{cg(Y)}$.
The total prob. of acceptance for any iteration is

$$\sum_y P(\text{accept}|y)P(Y = y) = \sum_y \frac{f(y)}{cg(y)} g(y) = \frac{1}{c}.$$

In the discrete case, for each k such that $f(k) > 0$,

$$P(Y = k \mid A) = \frac{P(A \mid k)g(k)}{P(A)} = \frac{(\frac{f(k)}{cg(k)})g(k)}{1/c} = f(k)$$

In the continuous case, we need to prove $P(Y \leq y \mid U \leq \frac{f(Y)}{cg(Y)}) = F_X(y)$. In fact

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

Example 3.7 (Acceptance-rejection method), Beta($\alpha = 2, \beta = 2$) dist.

The Beta(2,2) density is $f(x) = 6x(1-x), 0 < x < 1$. Let $g(x)$ be the $U(0,1)$ density. Then $f(x)/g(x) \leq 6$ for all $0 < x < 1$, so $c = 6$. A random x from $g(x)$ is accepted if

$$f(x)/cg(x) = x(1-x) > u.$$

Averagely, $cn = 6000$ iterations will be required for sample size 1000.

```
n <- 1000; k <- 0 #counter for accepted
y <- numeric(n); j <- 0 #iterations
while (k < n) {u <- runif(1); j<-j+1;
x <- runif(1) #random variate from g
if (x * (1-x) > u) { k<-k+1; y[k] <- x }} #we accept x
>j
[1] 5873
```

This simulation required 5873 iterations (11746 random numbers).

```
#compare empirical and theoretical percentiles
p<-seq(.1,.9,.1); Qhat<-quantile(y,p) #quantiles of sample
Q <- qbeta(p, 2, 2) #theoretical quantiles
se <- sqrt(p * (1-p) / (n * dbeta(Q, 2, 2))) #see Ch. 1
```

Sample percentiles (line 1) approximately match the Beta(2,2) percentiles computed by qbeta (line 2).

```
> round(rbind(Qhat, Q, se), 3)
        10%   20%   30%   40%   50%   60%   70%   80%   90%
Qhat  0.189 0.293 0.365 0.449 0.519 0.589 0.665 0.741 0.830
Q     0.196 0.287 0.363 0.433 0.500 0.567 0.637 0.713 0.804
se    0.010 0.011 0.012 0.013 0.013 0.013 0.012 0.011 0.010
```

$n = 10000$ replicates produces more precise estimates.

```
> round(rbind(Qhat, Q, se), 3)
        10%   20%   30%   40%   50%   60%   70%   80%   90%
Qhat  0.194 0.292 0.368 0.436 0.504 0.572 0.643 0.716 0.804
Q     0.196 0.287 0.363 0.433 0.500 0.567 0.637 0.713 0.804
se    0.003 0.004 0.004 0.004 0.004 0.004 0.004 0.004 0.003
```

Large numbers of replicates are required for estimation of percentiles where the density is close to zero.

Remark 3.2

See Example 3.8 for a more efficient beta generator based on the ratio of gammas method.

Transformation Methods

Many types of transformations other than the prob. inverse transformation can be applied to simulate r.v..

1. If $Z \sim N(0,1)$, then $V = Z^2 \sim \chi^2(1)$.
2. If $U \sim \chi^2(m)$ and $V \sim \chi^2(n)$ are independent, then $F = \frac{V/m}{U/m} \sim F(m,n)$.
3. If $Z \sim N(0,1)$ and $V \sim \chi^2(n)$ are independent, then $T = \frac{Z}{\sqrt{V/n}} \sim t(n)$
4. If $U, V \sim U(0,1)$ are independent, then $Z_1 = \sqrt{-2\log U}\cos(2\pi V)$ $Z_2 = \sqrt{-2\log V}\sin(2\pi U)$ are independent standard normal variables.
5. If $U \sim \text{Gamma}(r,\lambda)$ and $V \sim \text{Gamma}(s,\lambda)$ are independent, then $X = U/(U+V)$ has the Beta(r,s) dist.
6. If $U, V \sim U(0,1)$ are independent, then $X = \lfloor 1 + \frac{\log(V)}{\log(1-(1-\theta)^U)} \rfloor$ has the Logarithmic() dist., where $\lfloor x \rfloor$ denotes the integer part of x.

Example 3.8 (Beta distribution)

If $U \sim \text{Gamma}(r, \lambda)$ and $V \sim \text{Gamma}(s, \lambda)$ are independent, then $X = U/(U+V)$ has the Beta(r, s) dist.

1. Generate u from Gamma$(a, 1)$.
2. Generate v from Gamma$(b, 1)$.
3. Deliver $x = u/(u+v)$.

Beta(3,2) sample

```
n <- 1000; a <- 3; b <- 2;
u <- rgamma(n, shape=a, rate=1);
v <- rgamma(n, shape=b, rate=1);
x <- u/(u + v)
```

```
q <- qbeta(ppoints(n), a, b)
qqplot(q, x, cex=0.25, xlab="Beta(3, 2)", ylab="Sample")
abline(0, 1) #line x = q added for reference.
```

The sample data can be compared with the Beta(3, 2) dist. using a quantile-quantile (QQ) plot. If the sampled dist. is Beta(3, 2), the QQ plot should be nearly linear.

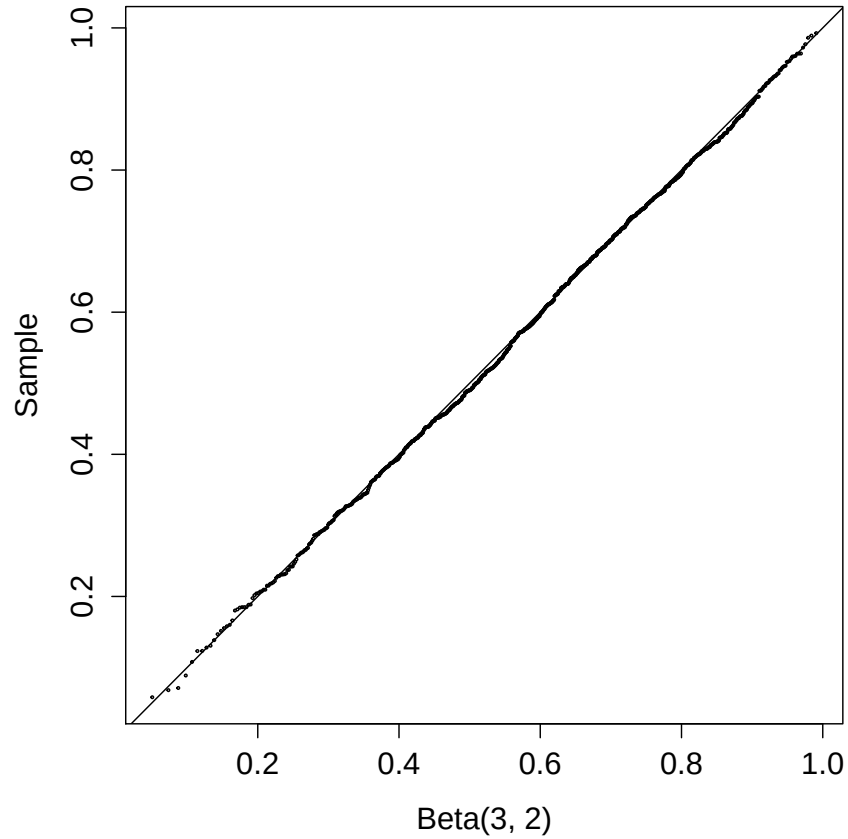


Fig.3.2: QQ Plot comparing the Beta(3, 2) distribution with a simulated data in Example 3.8.

Remark: The QQ plot of the ordered sample vs the Beta(3, 2) quantiles is very nearly linear.

Example 3.9 (Logarithmic dist., more efficient generator)

If U, V are independent $U(0,1)$ r.v., then $X = \lfloor 1 + \frac{\log(V)}{\log(1-(1-\theta)^U)} \rfloor$ has the Logarithmic(θ) dist.

1. Generate u from $U(0,1)$.
2. Generate v from $U(0,1)$.
3. Deliver $x = \lfloor 1 + \log(v)/\log\left(1-(1-theta)^u\right) \rfloor$.

```
# compare the Logarithmic(0.5) dist. with
# a sample using the transformation
n <- 1000; theta <- 0.5
u <- runif(n) #generate logarithmic sample
v <- runif(n);
x <- floor(1 + log(v) / log(1 - (1 - theta)^u))
k <- 1:max(x) #calc. logarithmic probs.
p <- -1 / log(1 - theta) * theta^k / k
se <- sqrt(p*(1-p)/n); p.hat <- tabulate(x)/n
```

```
> print(round(rbind(p.hat, p, se), 3))
        [,1] [,2] [,3] [,4] [,5] [,6] [,7]
p.hat  0.740 0.171 0.052 0.018 0.010 0.006 0.003
p      0.721 0.180 0.060 0.023 0.009 0.004 0.002
se     0.014 0.012 0.008 0.005 0.003 0.002 0.001
```

Below is a simple replacement for `rlogarithmic` in Example 3.6

```
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 - th)^u))
return(x)
}
```

Sums and Mixtures

Sums and mixtures of r.v. are special types of transformations.

Convolutions

Let $X_1, \ldots, X_n$ be i.i.d. with dist. $X_j \sim X$, and let $S = X_1 + \ldots + X_n$. The dist. function of the sum S is called the n-fold convolution of X. It is straightforward to simulate a convolution by directly generating $X_1, \ldots, X_n$ and computing the sum.

Examples:

- If $\nu > 0$ is an integer, chisquare dist. with ν degrees of freedom is the convolution of ν iid squared standard normal variables.
- Negative binomial dist. NegBin(r, p) is the convolution of r iid Geom(p) r.v.
- Convolution of r independent Exp(λ) r.v. has the Gamma(r, λ) dist.

Note: the functions `rchisq`, `rgeom` and `rnbinom` to generate chisquare, geometric and negative binomial random samples.

Example 3.10 (Chisquare)

Generate chisquare $\chi^2(\nu)$ random sample. If $Z_1, \cdots, Z_\nu$ are iid $N(0,1)$ r.v., then $V = Z_1^2 + \cdots + Z_\nu^2$ has the $\chi^2(\nu)$ dist.

1. Fill an $n \times \nu$ matrix with ν r.v. from $N(0,1)$.
2. Compute the row sums of the squared normals.
3. Deliver the vector of row sums.

```
n <- 1000; nu <- 2
X <- matrix(rnorm(n*nu), n, nu)^2 #matrix of sq. normals
#sum the squared normals across each row: method 1
y <- rowSums(X)
y<-apply(X,MARGIN=1,FUN=sum) #a vector length n, method 2
> mean(y); mean(y^2)
[1] 2.027334 7.835872
```

Sample statistics agree with theoretical moments $\mathrm{E}[Y] = \nu = 2$ and $\mathrm{E}[Y^2] = 2\nu + \nu^2 = 8$. std. of sample moments are 0.063 and 0.089.

R note 3.5

- `apply`: apply a function to the margins of an array. The function (FUN=sum) is applied to the rows (MARGIN=1)
- For efficient programming in R, avoid unnecessary loops.

Discrete mixture X

dist. of X is a weighted sum $F_X(x) = \sum \theta_i F_{X_i}(x)$ for some sequence of r.v. $X_1, X_2, \cdots$ and $\theta_i > 0$ such that $\sum_i \theta_i = 1$.

Continuous mixture X

dist. of X is $F_X(x) = \int_{-\infty}^{+\infty} F_{X|Y=y}(x) f_Y(y) dy$ for a family $X|Y = y$ indexed by the real numbers y and weighting function f_Y such that $\int_{-\infty}^{+\infty} f_Y(y) dy = 1$.

Example: 50% Normal mixture

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

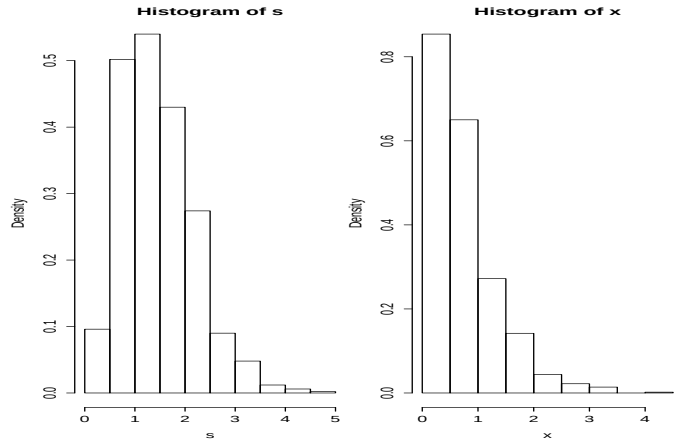
To simulate the mixture:

1. Generate an integer $k \in \{1, 2\}$,where $P(1) = P(2) = 0.5$
2. if $k = 1$ deliver random x from $N(0, 1)$;
3. if $k = 2$ deliver random x from $N(3, 1)$.

Example 3.11 (Convolutions and mixtures)

Let $X_1 \sim \text{Gamma}(2,2)$, $X_2 \sim \text{Gamma}(2,4)$ be independent. Compare the histograms of the samples generated by the convolution $S = X_1 + X_2$ and the mixture $F_X = 0.5F_{X_1} + 0.5F_{X_2}$.

Remark: Histograms of the convolution S and mixture X are different.




```
n <- 1000; x1 <- rgamma(n, 2, 2)
x2 <- rgamma(n, 2, 4); s <- x1 + x2 #the convolution
u <- runif(n)
k <- as.integer(u > 0.5) #vector of 0 s and 1 s
x <- k * x1 + (1-k) * x2 #the mixture
par(mfcol=c(1,2)) #two graphs per page
hist(s, prob=TRUE); hist(x, prob=TRUE)
par(mfcol=c(1,1)) #restore display
```

R note 3.6: `par`: set (or query) certain graphical parameters. `par()`: return a list of all graphical parameters. `par(mfcol=c(n,m))`: configure graphical device to display nm graphs per screen (n rows and m columns).

Example 3.12 (Mixture of several gamma dist.)

There are several components to the mixture and the mixing weights are not uniform. The mixture is $F_X = \sum_{i=1}^{5} \theta_j F_{X_j}$ where $X_j \sim$ Gamma$(r = 3, \lambda_j = 1/j)$ are independent and the mixing prob. are $\theta_j = j/15$, $j = 1, \cdots, 5$.

To simulate the mixture F_X:

1. Generate an integer $k \in \{1, 2, 3, 4, 5\}$, $P(k) = \theta_k, k = 1, \ldots, 5$.
2. Deliver a random Gamma(r, λ_k) variate.

which suggests using a `for` loop to generate a sample size n, but `for` loops are really inefficient in R.

Efficient vectorized algorithm:

1. Generate a random sample $k_1, \ldots, k_n$ of integers in vector `k`, where $P(k) = \theta_k, k = 1, \ldots, 5$. `k[i]` indicates which of the five gamma distributions will be sampled to get the ith element of sample (use sample).
2. Set rate equal to the length n vector $\lambda = (\lambda_k)$.
3. Generate a gamma sample size n, with shape parameter r and rate vector rate (use `rgamma`).

```r
n <- 5000
k <- sample(1:5, size=n, replace=TRUE, prob=(1:5)/15)
rate <- 1/k; x <- rgamma(n, shape=3, rate=rate)
#plot the density of the mixture
#with the densities of the components
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)))
```

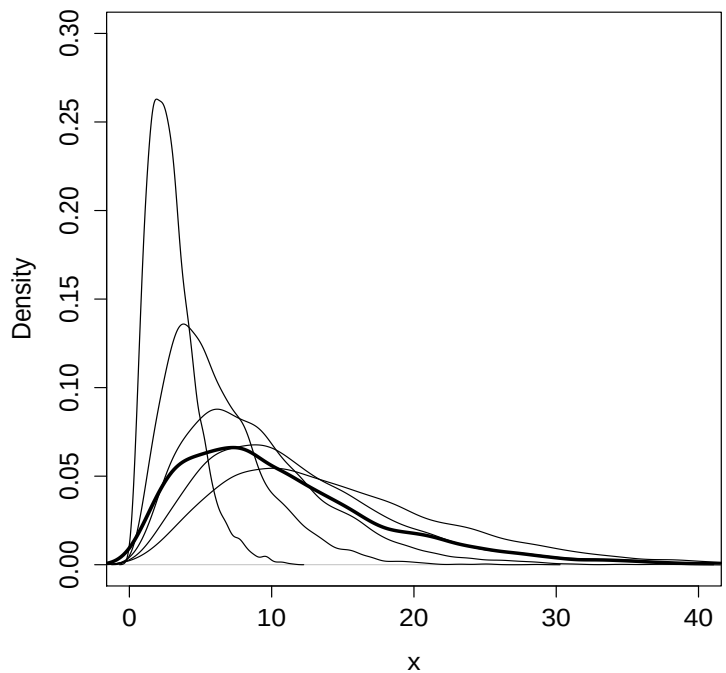


Fig.3.4: Density estimates from Example 3.12: A mixture (thick line) of several gamma densities (thin lines).

Example 3.14 (Plot density of mixture)

The density is $f(x) = \sum_{j=1}^{5} \theta_j f_j(x), x > 0$, where f_j is the Gamma(3,λ_j) density, with rates $\lambda = (1, 1.5, 2, 2.5, 3)$ and mixing prob. $\theta = (0.1, 0.2, 0.2, 0.3, 0.2)$.

function to compute the density $f(x)$

```
f <- function(x, lambda, theta) {
#density of the mixture at the point x
sum(dgamma(x, 3, lambda) * theta)}
```

dgamma(x,3,lambda)*theta is vector $(\theta_1 f_1(x), \cdots, \theta_5 f_5(x))$.

```
x<-seq(0,8,length=200); dim(x)<-length(x)#need for apply
y<-apply(x,1,f,lambda=lambda,theta=p)#compute density f(x)
#plot the density of the mixture
plot(x,y,type="l",ylim=c(0,.85),lwd=3,ylab="Density")
for (j in 1:5) {
#add the j-th gamma density to the plot
y <- apply(x, 1, dgamma, shape=3, rate=lambda[j])
lines(x, y)}
```

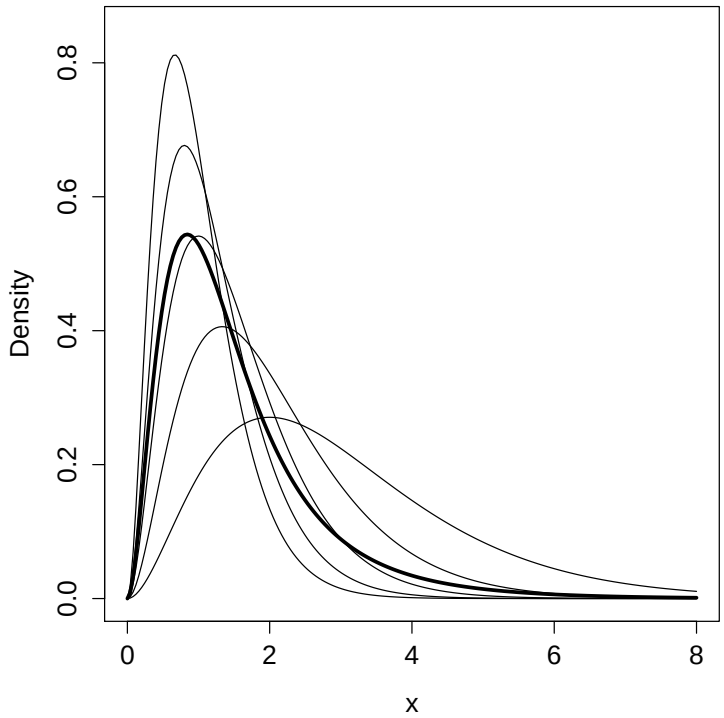


Fig.3.5: Densities from Example 3.14: A mixture (thick line) of several gamma densities (thin lines).

R note 3.7

- `apply` function requires a dimension attribute for x.
- Vector x does not have a dimension attribute, which is thus assigned by `dim(x) <- length(x)`. Alternately, `x <- as.matrix(x)` converts x to a matrix (a column vector), which has a dimension attribute.

Example 3.15 (Poisson-Gamma mixture, continuous mixture)

If $(X|\Lambda = \lambda) \sim \text{Poisson}(\lambda)$ and $\Lambda \sim \text{Gamma}(r, \beta)$, then X has the negative binomial dist. with parameters r and $p = \beta/(1+\beta)$.

```
#generate a Poisson-Gamma mixture
n <- 1000; r<-4; beta <- 3
lambda <- rgamma(n, r, beta) #lambda is random
#now supply the sample of lambda s as the Poisson mean
x <- rpois(n, lambda) #the mixture
#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)
        [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
mix    0.334 0.305 0.201 0.091 0.042 0.018 0.005 0.003 0.001
negbin 0.316 0.316 0.198 0.099 0.043 0.017 0.006 0.002 0.001
se     0.015 0.015 0.013 0.009 0.006 0.004 0.002 0.001 0.001
```

3.6.1 Multivariate Normal Dist.

A random vector $X = (X_1, \cdots, X_d)$ has a d-dimensional mutivariate normal (MVN) dist. denoted $N_d(\mu, \Sigma)$ if the density of X is

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

Suppose that Σ can be factored as $\Sigma = CC^T$ for some matrix C. Let $z \sim N(0, I_d)$, then $Cz + \mu = N_d(\mu, \Sigma)$. Factorization of Σ can be obtained by the spectral decomposition method (eigenvector decomposition, `eigen`), Choleski factorization (`chol`), or singular value decomposition (`svd`).

To generate a random sample of size n from $N_d(\mu, \Sigma)$:

1. Generate an nd matrix Z containing nd random $N(0, 1)$ variates
2. Compute a factorization $\Sigma = Q^T Q$.
3. Apply the transformation $X = ZQ + J\mu^T$ and deliver X.

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

Spectral decomposition method for generating $N_d(\mu, \Sigma)$ samples

Spectral decomposition: $\Sigma^{1/2} = P\Lambda^{1/2}P^{-1}$, $Q = \Sigma^{1/2}$, $Q^TQ = \Sigma$, where Λ is the diagonal matrix with the eigenvalues of Σ along the diagonal and P is the matrix whose columns are the corresponding eigenvectors. This method also called eigen-decomposition, in which, $P^{-1} = P^T$ and $\Sigma^{1/2} = P\Lambda^{1/2}P^T$.

Generate a random sample from $N_d(\mu, \Sigma)$ with $\mu = (0, 0)^T$ and

$$\Sigma = \begin{bmatrix} 1.0 & 0.9 \\ 0.9 & 1.0 \end{bmatrix}$$

```
# mean and covariance parameters
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}
```

Check results: Print summary statistics and display a scatterplot.

```
# generate the sample
X <- rmvn.eigen(1000, mu, Sigma)
plot(X, xlab = "x", ylab = "y", pch = 20)
> print(colMeans(X))
[1] -0.001628189 0.023474775
> print(cor(X))
[,1] [,2]
[1,] 1.0000000 0.8931007
[2,] 0.8931007 1.0000000
```

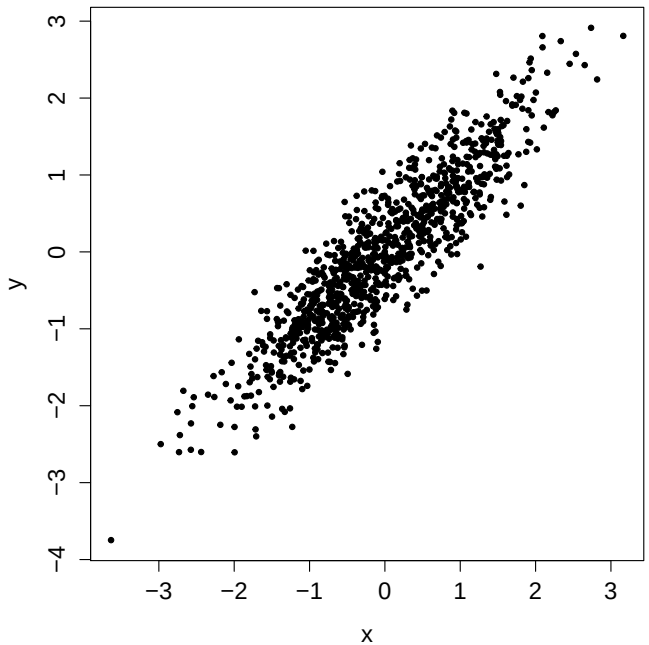


Fig.3.6: Scatterplot of a random bivariate normal sample from Example 3.16.

Remark: The scatter plot exhibits the elliptical symmetry of multivariate normal dist.

SVD Method of generating $N_d(\mu, \Sigma)$ samples

SVD generalizes the idea of eigenvectors to rectangular matrices.

- svd: $X = UDV^T$, where D is a vector containing the singular values of X, U is a matrix whose columns contain the left singular vectors of X, and V is a matrix whose columns contain the right singular vectors of X.
- Since $\Sigma \succ 0$ (positive definite), $UV^T = I$, thus $\Sigma^{1/2} = U\Lambda^{1/2}U^T$ and svd is equivalent to spectral decomposition.
- svd is less efficient because it does not take advantage of the fact that the matrix Σ is square symmetric.

```
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}
```

Choleski factorization method

Choleski factorization: $X = Q^T Q (X \succ 0)$, where Q is an upper triangular matrix. The R syntax: `Q=chol(X)`.

```
rmvn.Choleski <- function(n, mu, 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}
```

Generate 200 random observations from a 4-D normal using `chol`

```
y <- subset(x=iris, Species=="virginica")[, 1:4]
mu <- colMeans(y); Sigma <- cov(y)
#now generate MVN data with this mean and covariance
X <- rmvn.Choleski(200, mu, Sigma); pairs(X)
```

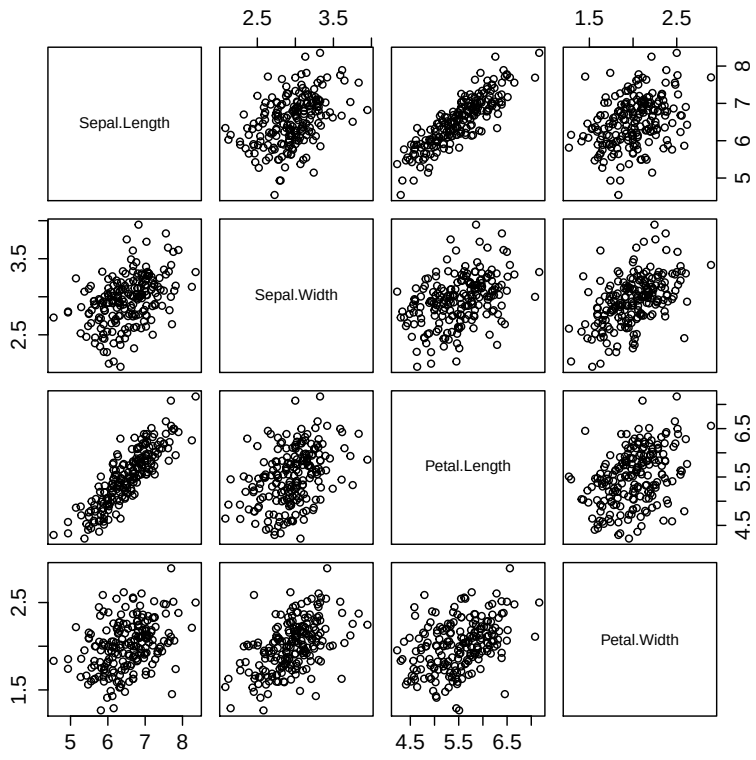


Fig.3.7: Pairs plot of the bivariate marginal distributions of a simulated multivariate normal random sample in Example 3.18.

Remark: The joint distribution of each pair of marginal distributions is theoretically bivariate normal. The plot can be compared with Figure 4.1, which displays the iris virginica data.

Comparing Performance of Generators

- Compares the timing of the above three generators and two other generators (`rmvnorm` and `rmvnorm`).
- Generate data in $\mathbb{R}^{30}$. `system.time` function is used.

```
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)))
```

3.6.2 Mixtures of Multivariate Normals

A multivariate normal mixture is denoted

$$pN_d(\mu_1, \Sigma_1) + (1-p)N_d(\mu_2, \Sigma_2) \tag{3.7}$$

1. Generate $U \sim U(0,1)$ ($N \sim$ Bernoulli(p)).
2. If $U \leq p$ $(N = 1)$, generate X from $N_d(\mu_1, \Sigma_1)$; otherwise generate X from $N_d(\mu_2, \Sigma_2)$.

Generate a multivariate normal mixture with two components

```
library(MASS)  #for mvrnorm
#ineffecient version loc.mix.0 with loops
loc.mix.0 <- function(n, p, mu1, mu2, Sigma) {
  #generate sample from BVN location mixture
  X <- matrix(0, n, 2)
  for (i in 1:n) {
    k <- rbinom(1, size = 1, prob = p)
    if (k)
      X[i,] <- mvrnorm(1, mu = mu1, Sigma) else
        X[i,] <- mvrnorm(1, mu = mu2, Sigma)  }
  return(X) }
```

More efficient version

```
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
}
```

Generate a sample of $n = 1000$ from a 50% 4-dimensional normal mixture with $\mu_1 = (0, 0, 0, 0)$ and $\mu_2 = (2, 3, 4, 5)$ and $\Sigma = I_4$.

```
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))
par(mfrow = c(1, 1))
```

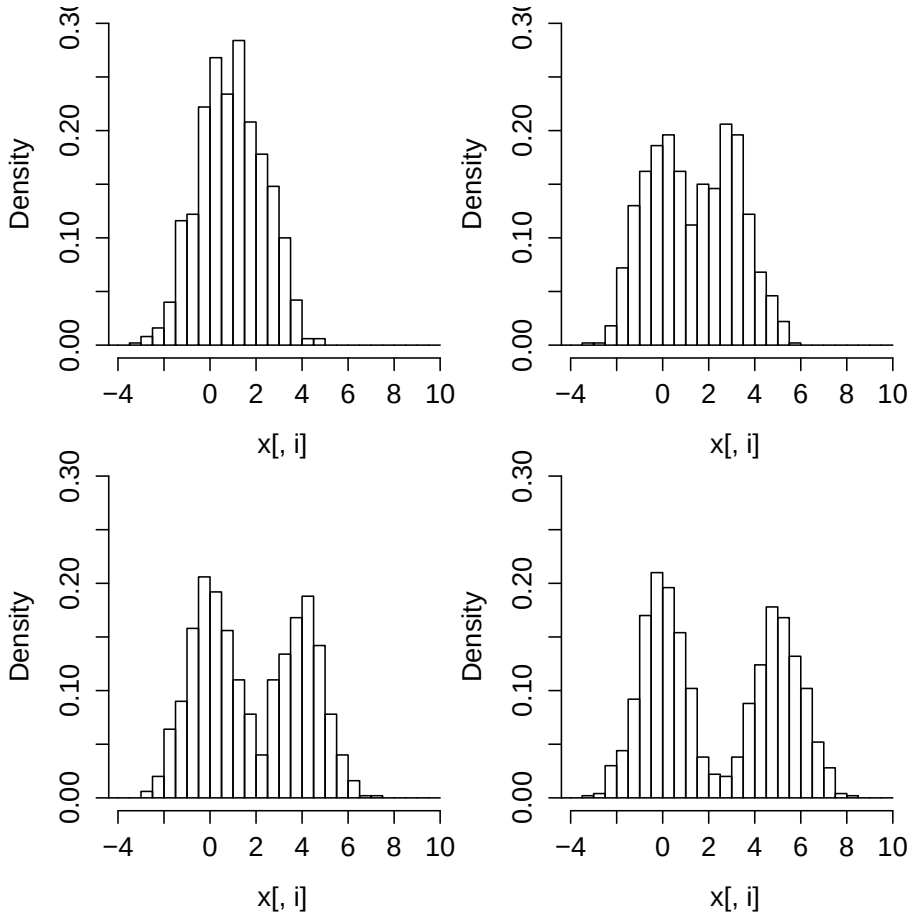


Fig.3.8: Histograms of the marginal distributions of MVN data generated in Example 3.20.

Remark: It is difficult to visualize data in $\mathbf{R}^4$, so display only the histograms of the marginal distributions. All of the 1-D marginal distributions are univariate normal location mixtures. Methods for visualization of multivariate data are covered in Chapter 4.

3.6.3 Wishart Distribution

Suppose $= X^T X$, where X is nd data matrix from $N_d(\mu, \Sigma)$ dist. Then M has a Wishart dist. with scale matrix Σ and n degrees of freedom, denoted $M \sim W_d(\Sigma, n)$.

When $d = 1$, X is from $N(\mu, \alpha^2)$, so $W_1(\alpha, n) \stackrel{D}{=} \sigma^2 \chi^2(n)$.

To generate Wishart r.v.

Generate MVN sample X and compute the matrix product $X^T X$. Computationally expensive because nd r.v. must be generated to determine the $d(d+1)/2$ distinct entries in M.

More efficient method based on Bartletts decomposition

Let $T = (T_{ij})$ be a lower triangular $d \times d$ random matrix with independent entries satisfying

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

Then the matrix $A = TT^T$ has a $W_d(I_d, n)$ dist.

To generate $W_d(\Sigma, n)$ r.v.: Obtain the Choleski factorization $\Sigma = LL^T$, where L is lower triangular. Then $LAL^T \sim W_d(\Sigma, n)$.

3.6.4 Uniform Dist. on the d-Sphere

The d-sphere is the set of all points $x \in R^d$ such that $\| x \| = (x^T x)^{1/2} = 1$. If $X_1, ..., X_d$ are iid $N(0,1)$, then $U = (U_1, ..., U_d)$ is uniformly distributed on the unit sphere in $\mathbb{R}^d$, where

$$U_j = \frac{X_j}{(X_1^2 + \ + X_d^2)^{1/2}}, j = 1, \cdots, d. \tag{3.8}$$

To generate uniform r.v. on the d-Sphere, for each variate $u_i, i = 1, \cdots, n$, repeat

1. Generate a random sample $x_{i1}, \ldots, x_{id}$ from $N(0,1)$
2. Compute the Euclidean norm $\| x \| = (x_{i1}^2 + \ + x_{id}^2)^{1/2}$
3. Set $u_{ij} = x_{ij} / \| x \|, j = 1, \cdots, d.$
4. Deliver $u_i = (u_{i1}, \ldots, u_{id})$.

```r
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}
```

Generate a sample of 200 points uniformly distributed on the circle.

```r
#generate a sample in d=2 and plot
X <- runif.sphere(200, 2)
par(pty = "s")
plot(X, xlab = bquote(x[1]), ylab = bquote(x[2]))
par(pty = "m")
```

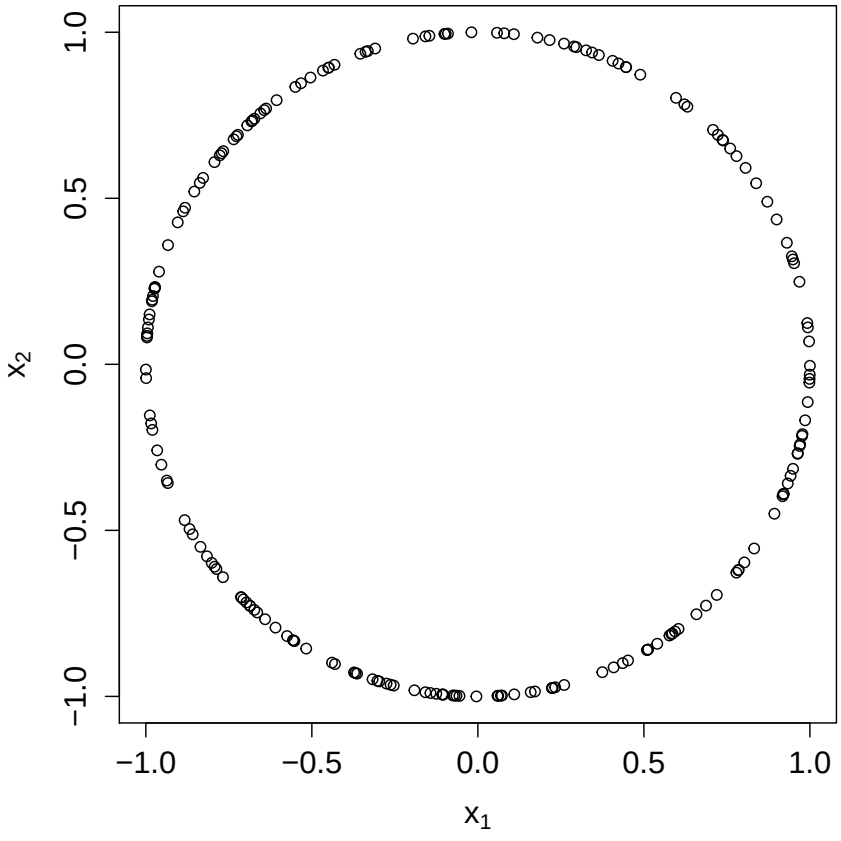


Fig.3.9: A random sample of 200 points that is uniformly distributed on the unit circle in Example 3.21.

R note 3.9

- `par(pty = "s")`: set the square plot type so the circle is round rather than elliptical;
- `par(pty = "m")` restore the type to maximal plotting region.