Introduction to the R software

Chapter 3. Distributions and Matrix Operations

Peng Zhang

School of Mathematical Science
Zhejiang University

Outline

1. Discrete distributions
   - Binomial distribution
   - Poisson distribution
2. Continuous distributions
   - Normal distribution
3. Simulation of random numbers
   - Sampling from discrete distributions
   - Sampling from the normal and other continuous distributions
   - Simulation of regression data
4. Sampling from finite populations
5. Matrix operations

Bernoulli trial

- Successive tosses of a fair coin come up tails or heads with probability 0.5, independently between tosses.
- If we let X take the value 1 for a head and 0 for a tail, then X is said to have a Bernoulli distribution with parameter $\pi = 0.5$.
- More generally, we might consider an experiment or test with an uncertain outcome, but where the possibilities are "success" (or "1") and "failure" (or "0"). Success may occur with probability π, where $0 \leq \pi \leq 1$.

Binomial distribution

- The sum of a number of independent Bernoulli random variables is called a binomial random variable.
- The number of successes in n independent Bernoulli trials (where success at each trial occurs with probability π) has a binomial distribution with parameters n and π.
- We can use the function *dbinom*() to determine probabilities of having 0, 1 or 2 heads in two coin tosses:

```
probs <- dbinom(0:2, size=2, prob=0.5)
names(probs) <- 0:2
probs
   0     1     2
0.25  0.50  0.25
```

Cumulative distribution function

- To calculate the probability of no more than two heads, add up the probabilities of 0, 1, and 2 ($0.0625 + 0.2500 + 0.3750 = 0.6875$).
- The function *pbinom*() can be used to determine such cumulative probabilities, thus:

```
pbinom(q=2, size=4, prob=0.5)
```

- For another example, suppose a sample of 50 manufactured items is taken from an assembly line that produces 20% defective items, on average. To find the probability of observing no more than four defectives in a sample, use:

```
> pbinom(q=4, size=50, prob=0.2)
[1] 0.0185
```

Poisson distribution

- The Poisson distribution is often used to model the number of events that occur in a certain time interval or for the numbers of defects that are observed in items such as manufactured products.
- The distribution depends on a parameter λ, which happens to coincide with the mean or expected value.
- Consider a population of raisin buns for which there are an average of three raisins per bun, i.e., $\lambda = 3$. Under the Poisson model, we have the following probabilities for 0, 1, 2, 3, or 4 raisins in a bun:

```
## Probabilities of 0, 1, 2, 3, 4 raisins
## mean number of raisins per bun = 3
## dpois(x = 0:4, lambda = 3)
     0      1      2      3      4
0.0498 0.1494 0.2240 0.2240 0.1680
```

Cumulative distribution function

The cumulative probabilities are:

```
## ppois(q = 0:4, lambda = 3)
     0      1      2      3      4
0.0498 0.1991 0.4232 0.6472 0.8153
```

Thus, for example, the probability of finding two or fewer raisins in a bun is 0.4232.

Probability density function

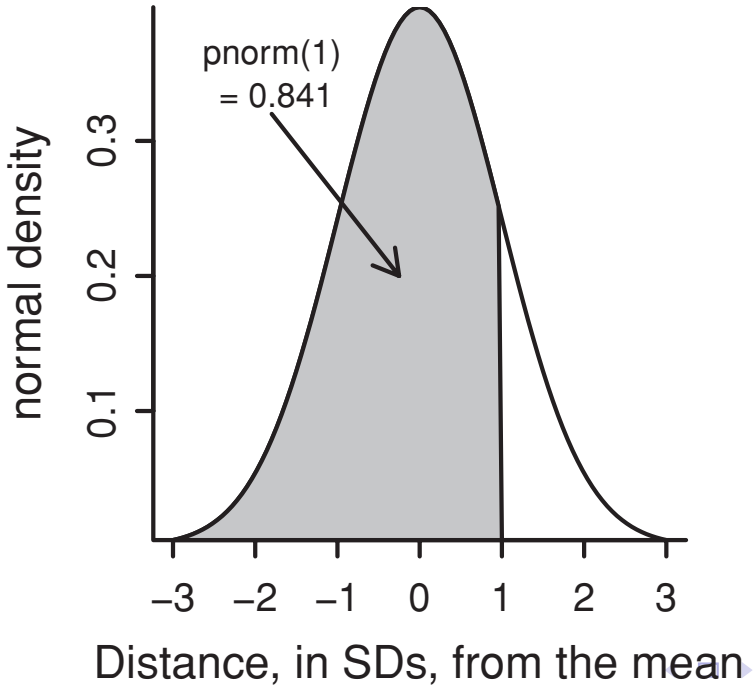
- The normal distribution, which has the bell-shaped density curve pictured in the figure, is often used as a model for continuous measurement data.
- It corresponds to a normal distribution with a mean of 0 and standard deviation 1.

$$\phi(z) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right).$$

- The height of the curve is a function of the distance from the mean. The area under the density curve is 1.
- Multiplying a fixed value σ by a population of such normal variates changes the standard deviation to σ. By adding a fixed value μ, we can change the mean to μ, leaving the standard deviation unchanged.

Normal density curve

```
## Plot the normal density, in the range -3 to 3
z <- pretty(c(-3,3), 30) # Find 30 equally spaced points
ht <- dnorm(z) # By default: mean=0, standard deviation=1
plot(z, ht, type="l", xlab="Normal deviate",
                              ylab="Density", yaxs="i")
# yaxs="i" locates the axes at the limits of the data
```



Cumulative probability and quantile

- The function *pnorm*() calculates the cumulative probability, i.e., the area under the curve up to the specified ordinate or x-value. For example, there is a probability of 0.841 that a normal deviate is less than 1.
- This corresponds to the area of the shaded region in the above figure.

```
> pnorm(1.0) # by default, mean=0 and SD=1
[1] 0.841
```

- The function *qnorm*() can be used to compute the normal quantiles. For example, the 90th percentile is 1.28:

```
> qnorm(.9) # 90th percentile; mean=0 and SD=1
[1] 1.28
```

Other continuous distributions

- The uniform distribution, for which an observation is equally likely to take any value in a given interval; the probability density of values is constant on a fixed interval.
- The exponential distribution that gives high probability density to positive values lying near 0; the density decays exponentially as the values increase.
- χ^2 distribution.
- t distribution.
- F distribution.

Simulation

- In a simulation, repeated random samples are taken from a specified distribution. Statistics, perhaps estimates that are derived from one or other model, can then be calculated for each successive sample.
- The following uses *set.seed*() to make the call below to *rbinom*(10, *size* = 1, *p* = .5) thus reproducible:

```
set.seed(23286) # Use to reproduce the sample below
rbinom(10, size=1, p=.5)
```

Simulation

- Simulate a Bernoulli process with 10 trials.

```
> rbinom(10, size=1, p=.5) # 10 Bernoulli trials, prob=0.5
1 0 0 0 1 1 1 0 1 0
```

- To generate the numbers of daughters in a simulated sample of 25 four-child families, assuming that males and females are equally likely,

```
# For the sequence that follows, set.seed(9388)
> rbinom(25, size=4, prob=0.5)
[1] 3 1 2 4 1 2 0 3 2 1 2 3 2 4 2 1 1 1 2 2 3 2 0 2 2
```

- Now simulate the number of raisins in 20 raisin buns, where the expected number of raisins per bun is 3:

```
> set.seed(9388)
> rpois(20, 3)
[1] 3 3 4 1 2 2 3 1 1 4 3 0 1 1 3 1 0 4 5 2
```

Simulation

- To generate 10 random variables from a standard normal distribution,

```
> options(digits=2) # Suggest number of digits to display
> rnorm(10) # 10 from the standard normal distribution
          # For our sequence, precede with set.seed(3663)
 [1] -0.599 -1.876 1.441 -1.025 0.612 -1.669 0.138 -0.099
1.010 0.013
```

- Use *runif*() to generate uniform random numbers or *rexp*() to generate exponential random numbers, follow the same pattern.

```
runif(n = 20, min=0, max=1) # 20 uniform numbers on (0, 1)
rexp(n=10, rate=3) # 10 numbers, exponential, mean 1/3.
```

Simulation of regression data

Simulate a sample of n observations from the model:

$$y = b_0 + b_1 x + \varepsilon.$$

where ε is normally distributed with standard deviation σ . We take $n = 8$, the intercept to be 2, the slope to be 3, and σ to be 2.5 in our simulation, and we use a fixed equally spaced design for the predictor values:

```
> options(digits=3)
> n<-8; x<-seq(1,n); sigma<-2.5; b0<-2; b1<-3
> error <- rnorm(n, sd=sigma)
> y <- b0 + b1*x + error
> t(data.frame(x,y))
   [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]
x  1.00  2.00  3.00   4.0   5.0     6   7.0   8.0
y  6.85  8.96  9.49  12.2  19.1    20  26.2  28.5
```

Use *simulate*()

- The function *simulate*() can be used to repeatedly simulate data from a fitted model, then re-fitting to each new set of simulated data.
- Thus to do 10 simulations based on the model that was fitted to the *roller* data, do:

```
roller.lm <- lm(depression ~ weight, data=roller)
roller.sim <- simulate(roller.lm, nsim=20) # 20
simulations
```

- The object *roller.sim* is a data frame with 20 columns, i.e., one column for each of the 20 simulations. Each column has values of depression, simulated from the fitted model at each level of weight. To visualize this output, enter

```
with(roller, matplot(weight, roller.sim, pch=1, ylim=range
(depression)) points(roller, pch=16)
```

Use *sample*()

- We can use the *sample*() function to generate a simple random sample from a given set of numbers.

```
> ## For the sequence below, set.seed(3676)
> sample(1:9384, 15, replace=FALSE)
[1] 9178 2408 8724 173 106 4664 3787 6381 5098 3228 8321
165 7332 9036 540
```

- To randomly assign 10 plants (labeled from 1 to 10, inclusive) to one of two equal-sized groups, control and treatment,

```
> ## For the sequence below, set.seed(366)
>split(sample(seq(1:10)),rep(c("Control","Treatment"),5))
> # sample(1:10) gives a random re-arrangement
$Control
[1] 6 8 3 7 9
$Treatment
[1] 5 4 2 110
```

With-replacement samples

- We can randomly sample from the set $\{1, 2, \ldots, 10\}$, allowing for repeated observations,

```
> sample(1:10, replace=TRUE)
[1] 7 5 2 1 2 3 1 5 7 6
```

Matrix operations

Several basic matrix operations are included in the base version of **R**. Before we introduce them, let λ be a scalar, let **A** and **B** be two real matrices.

```
> lambda <- 2 # Creating scalar λ.
> A <- matrix(c(2,3,5,4),nrow=2,ncol=2) # Real matrix.
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
> B <- matrix(c(1,2,2,7),nrow=2,ncol=2) # Symmetric real matrix.
> B
     [,1] [,2]
[1,]    1    2
[2,]    2    7
```

Matrix operations

Several basic matrix operations are included in the base version of **R**. Before we introduce them, let λ be a scalar, let **A** and **B** be two real matrices.

```
> lambda <- 2 # Creating scalar λ.
> A <- matrix(c(2,3,5,4),nrow=2,ncol=2) # Real matrix.
> A
     [,1] [,2]
[1,]    2    5
[2,]    3    4
> B <- matrix(c(1,2,2,7),nrow=2,ncol=2) # Symmetric real matrix.
> B
     [,1] [,2]
[1,]    1    2
[2,]    2    7
```

Matrix operations

- Adding a scalar: $\lambda + \mathcal{A}$

```
> lambda+A
     [,1] [,2]
[1,]    4    7
[2,]    5    6
```

- Addition (entry-wise): $\mathcal{A} + \mathcal{B}$

```
> A+B
     [,1] [,2]
[1,]    3    7
[2,]    5   11
```

- Substraction (entry-wise): $\mathcal{A} - \mathcal{B}$

```
> A-B
     [,1] [,2]
[1,]    1    3
[2,]    1   -3
```

- Multiplying by a scalar: $\lambda\mathcal{A}$

```
> lambda*A
     [,1] [,2]
[1,]    4   10
[2,]    6    8
```

- Transposition: $\mathcal{A}^{\mathsf{T}}$

```
> t(A)
     [,1] [,2]
[1,]    2    3
[2,]    5    4
```

Matrix operations

- Entry-wise multiplication:

```
> A*B
     [,1] [,2]
[1,]    2   10
[2,]    6   28
```

- Dot product: $\mathcal{AB}$

```
> A%*%B
     [,1] [,2]
[1,]   12   39
[2,]   11   34
```

- Entry-wise division:

```
> A/B
     [,1]      [,2]
[1,]  2.0 2.5000000
[2,]  1.5 0.5714286
```

- Matrix inversion: $\mathcal{B}^{-1}$

```
> solve(B)
           [,1]       [,2]
[1,]  2.3333333 -0.6666667
[2,] -0.6666667  0.3333333
```

- Matrix division: $\mathcal{A}^{-1}\mathcal{B}$

```
> solve(A)%*%B # Identical to: solve(A,B)
           [,1]      [,2]
[1,]  0.8571429  3.857143
[2,] -0.1428571 -1.142857
```

- Cross product: $\mathcal{A}^{\mathsf{T}}\mathcal{B}$

```
> crossprod(A,B)   # t(A)%*%B
     [,1] [,2]
[1,]    8   25
[2,]   13   38
```

Outer product

The outer product of column vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is the matrix $\boldsymbol{x}\,\boldsymbol{y}^{\mathsf{T}}$ of general element $x_i\,y_j$.

```
> x <- seq(1,4)
> y <- seq(4,7)
> outer(x,y,FUN="*")
     [,1] [,2] [,3] [,4]
[1,]    4    5    6    7
[2,]    8   10   12   14
[3,]   12   15   18   21
[4,]   16   20   24   28
```

Kronecker product

If $\boldsymbol{\mathcal{A}}$ is an $m \times n$ matrix and $\boldsymbol{\mathcal{B}}$ is a $p \times q$ matrix, then the Kronecker product of matrix $\boldsymbol{\mathcal{A}}$ by matrix $\boldsymbol{\mathcal{B}}$ is the matrix $\boldsymbol{\mathcal{A}} \otimes \boldsymbol{\mathcal{B}} = \begin{bmatrix} a_{11}\boldsymbol{\mathcal{B}} & \cdots & a_{1n}\boldsymbol{\mathcal{B}} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{\mathcal{B}} & \cdots & a_{mn}\boldsymbol{\mathcal{B}} \end{bmatrix}$ of dimensions $mp \times nq$.

```
> kronecker(A,B)
     [,1] [,2] [,3] [,4]
[1,]    2    4    5   10
[2,]    4   14   10   35
[3,]    3    6    4    8
[4,]    6   21    8   28
```

Triangular matrices

It can be useful to get the lower and upper parts of a matrix. This can be done with the functions *lower.tri()* and *upper.tri()*.

```
> M <- matrix(1:16,nrow=4)
> lower.tri(M)
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,]  TRUE FALSE FALSE FALSE
[3,]  TRUE  TRUE FALSE FALSE
[4,]  TRUE  TRUE  TRUE FALSE
> upper.tri(M,diag=TRUE)
      [,1]  [,2]  [,3] [,4]
[1,]  TRUE  TRUE  TRUE TRUE
[2,] FALSE  TRUE  TRUE TRUE
[3,] FALSE FALSE  TRUE TRUE
[4,] FALSE FALSE FALSE TRUE
```

```
> M[lower.tri(M)] <- 0
> M
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    0    6   10   14
[3,]    0    0   11   15
[4,]    0    0    0   16
```

Determinant and trace

The function `det()` computes the determinant of a matrix.

```
> det(A)
[1] -7
```

There is no R function to compute the trace of a matrix directly, but it is very easy to calculate:

```
> sum(diag(A))
[1] 6
```

Eigenvalues and eigenvectors

The eigenvalues and eigenvectors of a matrix are returned by the function *eigen()*.

```
> eigen(A)
$values
[1]  7 -1
$vectors
           [,1]       [,2]
[1,] -0.7071068 -0.8574929
[2,] -0.7071068  0.5144958
```