Statistics 1 Unit 3 Team 8

Nikolaos Kornilakis Rodrigo Viale Jakub Trnan Aleksandra Daneva
Luis Diego Pena Monge

2023-12-20

Exercise 41

Let $X \sim N(\vec{\mu}_x, \Sigma_x)$. Then $Y = AX + b \sim N(A\vec{\mu}_x + b, A\Sigma_x A')$. If we draw n independent points from an univariate standard normal distribution, then we are able to create a vector that has a multivariate standard normal distribution, i.e. $(x_1, ..., x_n) \sim N(\vec{0}, I_{n\times n})$. Note we have that $\vec{\mu}_x = \vec{0}$ since every draw has a mean value of 0, and $\Sigma_x = I_{n\times n}$ since each draw has variance 1 and we assume independence between them.

Assume we want to generate observations from $Y \sim N(\vec{\mu}_y, \Sigma_y)$ through X. Then the mean vector $\vec{\mu}_y = A\vec{\mu}_x + b$ is trivial to find since $\vec{\mu}_x = 0$. We just put $\mu_y = b$. For Σ_y, however, need to find A such that $\Sigma_y = A\Sigma_x A' = AI_{n\times n}A' = AA'$.

a)

Note that is we use the eigendecomposition $\Sigma_y = UDU'$, where D is diagonal with Σ_y's eigenvalues as diagonal elements and U has its eigenvectors as columns, then we have that since D is diagonal:

$$D = \begin{pmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_n \end{pmatrix} = \begin{pmatrix} \sqrt{\lambda_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda_n} \end{pmatrix} \begin{pmatrix} \sqrt{\lambda_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda_n} \end{pmatrix} = \sqrt{D}\sqrt{D}$$

Also, since $\sqrt{D}$ is a diagonal matrix, then $(\sqrt{D})' = \sqrt{D}$. Also, note that $\Sigma_y = UDU' = U\sqrt{D}\sqrt{D}U' = U\sqrt{D}\Sigma_x\sqrt{D}U' = (U\sqrt{D})\Sigma_x(U\sqrt{D})'$. Then, take $A = (U\sqrt{D})$

```
rmvnorm <- function(n, m, Sigma){
  X <- matrix(rnorm(n*nrow(Sigma)), nrow = nrow(Sigma), ncol = n)

  tmp <- eigen(Sigma)
  U <- tmp$vectors
  sqrtD <- diag(sqrt(tmp$values))

  A <- U %*% sqrtD

  t(A %*% X + m)

}


m <- c(5,0)
Sigma <- matrix(c(1,0.5,0.5,2),2)
mv_norm <- rmvnorm(1000, m, Sigma)
```

```r
apply(mv_norm, 2, mean)
```

```
## [1]  4.997734342 -0.002861611
```

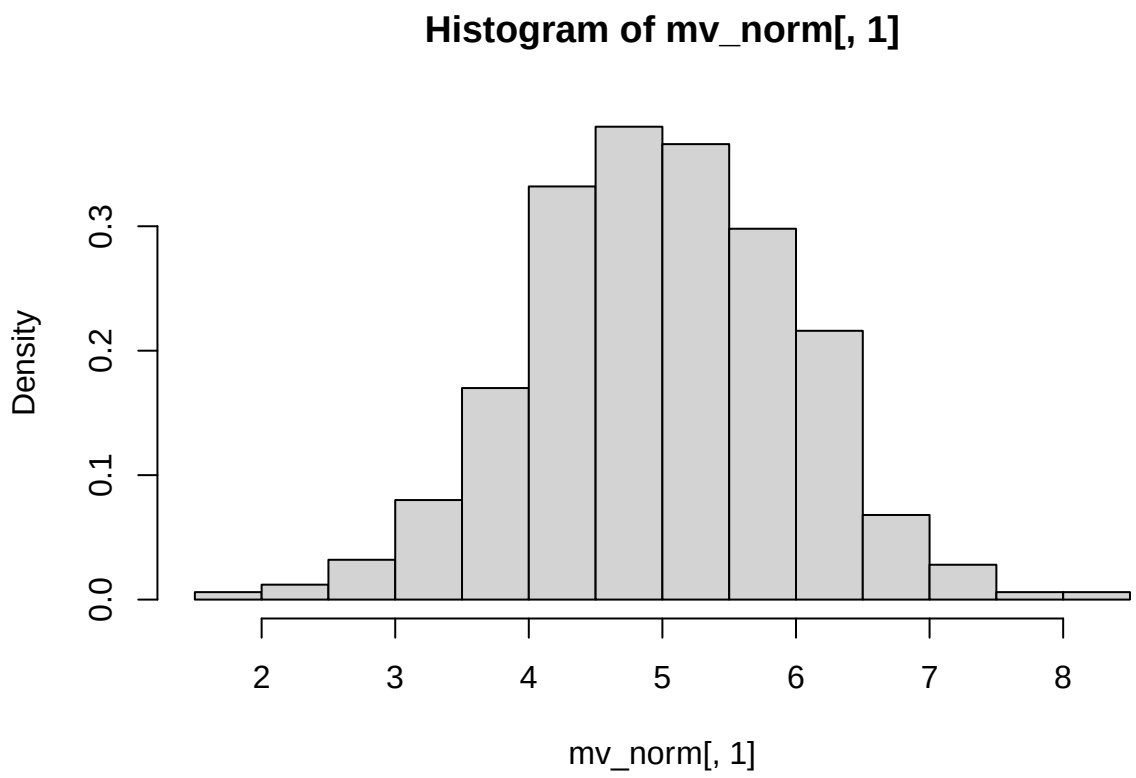
```r
apply(mv_norm, 2, var)
```

```
## [1] 0.9848105 1.8845571
```

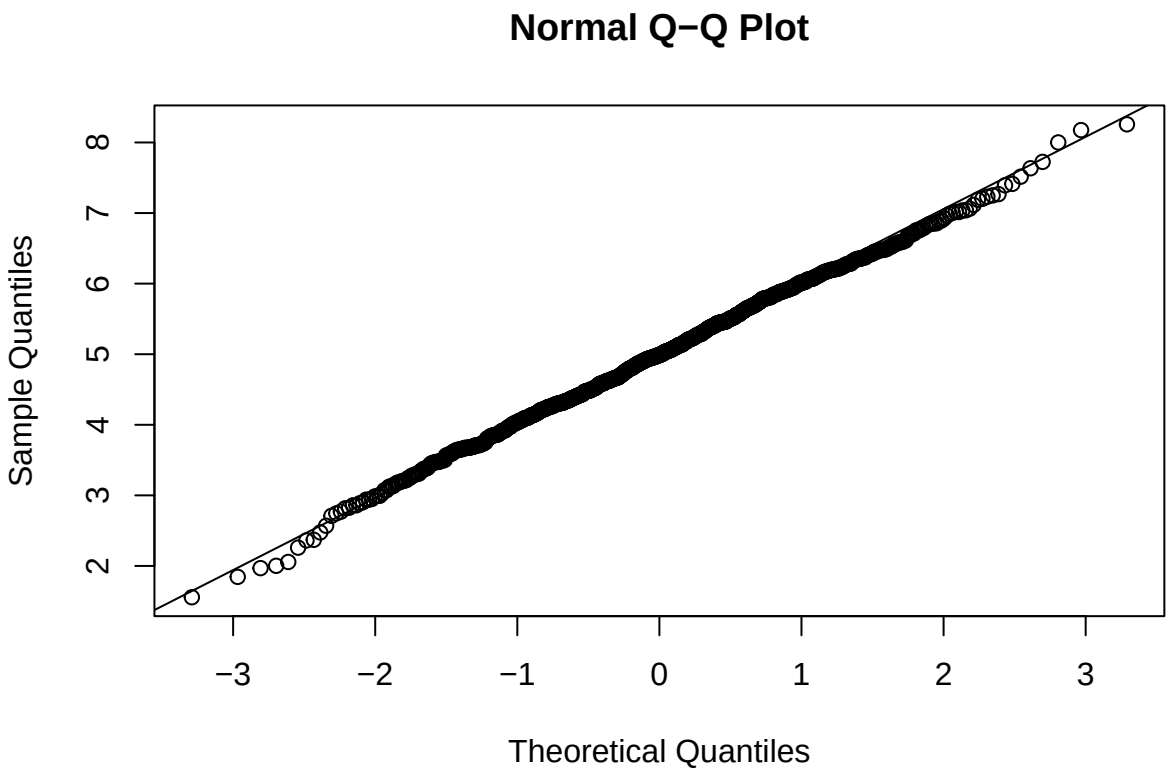
```r
cov(mv_norm[,1], mv_norm[,2])
```

```
## [1] 0.4666726
```

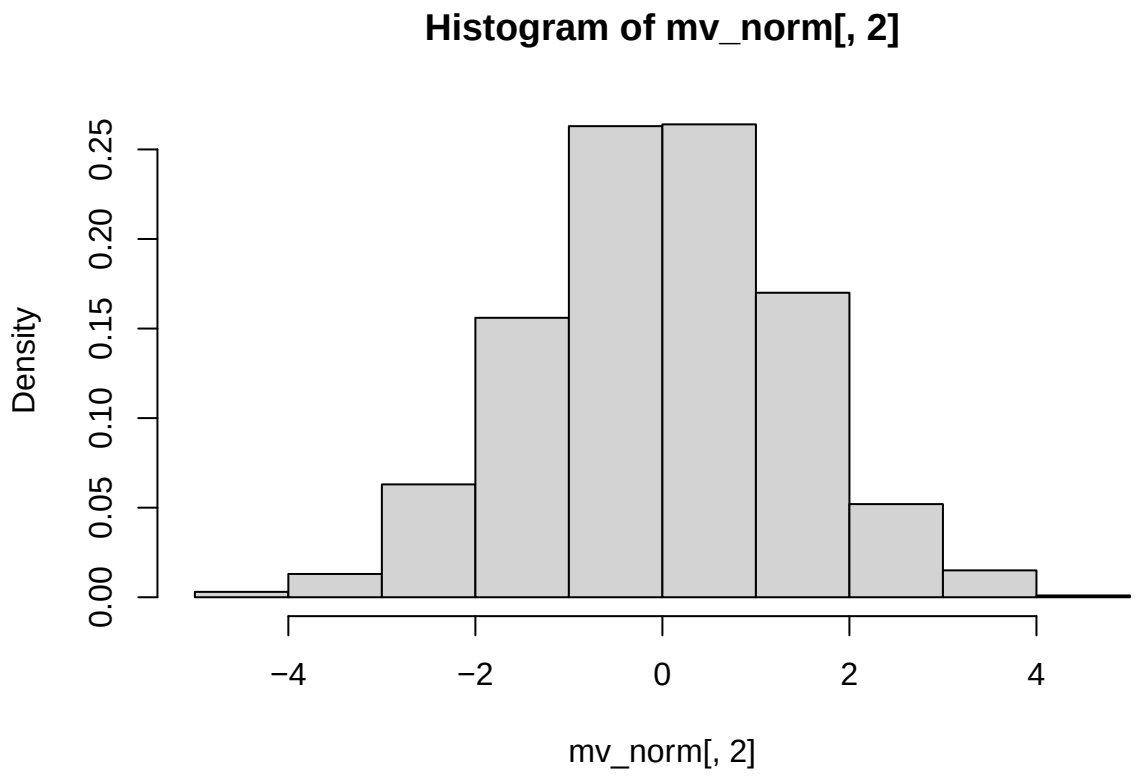
```r
hist(mv_norm[,1], probability = T)
```



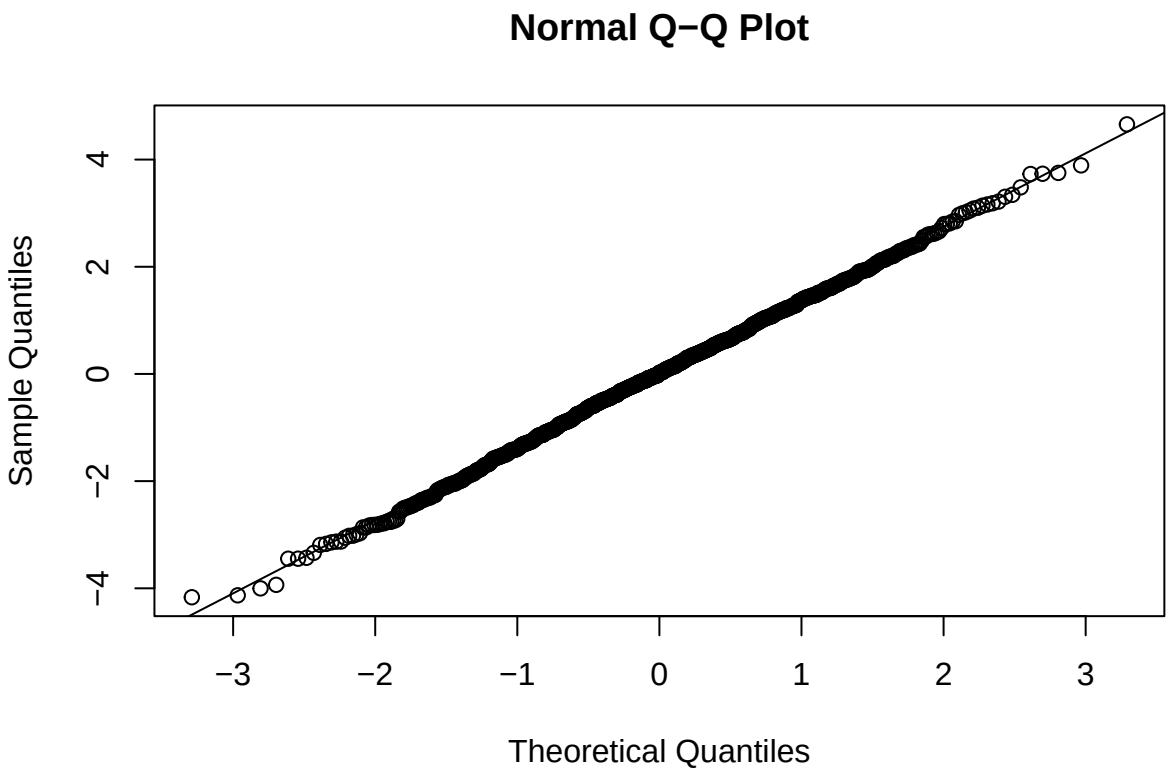

```r
qqnorm(mv_norm[,1])
qqline(mv_norm[,1])
```




```
hist(mv_norm[,2], probability = T)
```




```
qqnorm(mv_norm[,2])
qqline(mv_norm[,2])
```



b)

Recall we want to find A such that $\Sigma_y = A\Sigma_x A' = AI_{n\times n}A' = AA'$. This formula immediately indicates we can use Σ_y's Cholesky decomposition. In this case, it if the matrix has the Cholesky decomposition $\Sigma_y = LL'$, then it is enough to take $A = L$. Note that in R, the chol() function returns the upper triangular matrix, so we transpose it.

```
rmvnorm <- function(n, m, Sigma){
  X <- matrix(rnorm(n*nrow(Sigma)), nrow = nrow(Sigma), ncol = n)

  A <- t(chol(Sigma))

  t(A %*% X + m)

}


m <- c(5,0)
Sigma <- matrix(c(1,0.5,0.5,2),2)
mv_norm <- rmvnorm(1000, m, Sigma)

apply(mv_norm, 2, mean)
```

```
## [1] 5.03572055 0.03632609
```

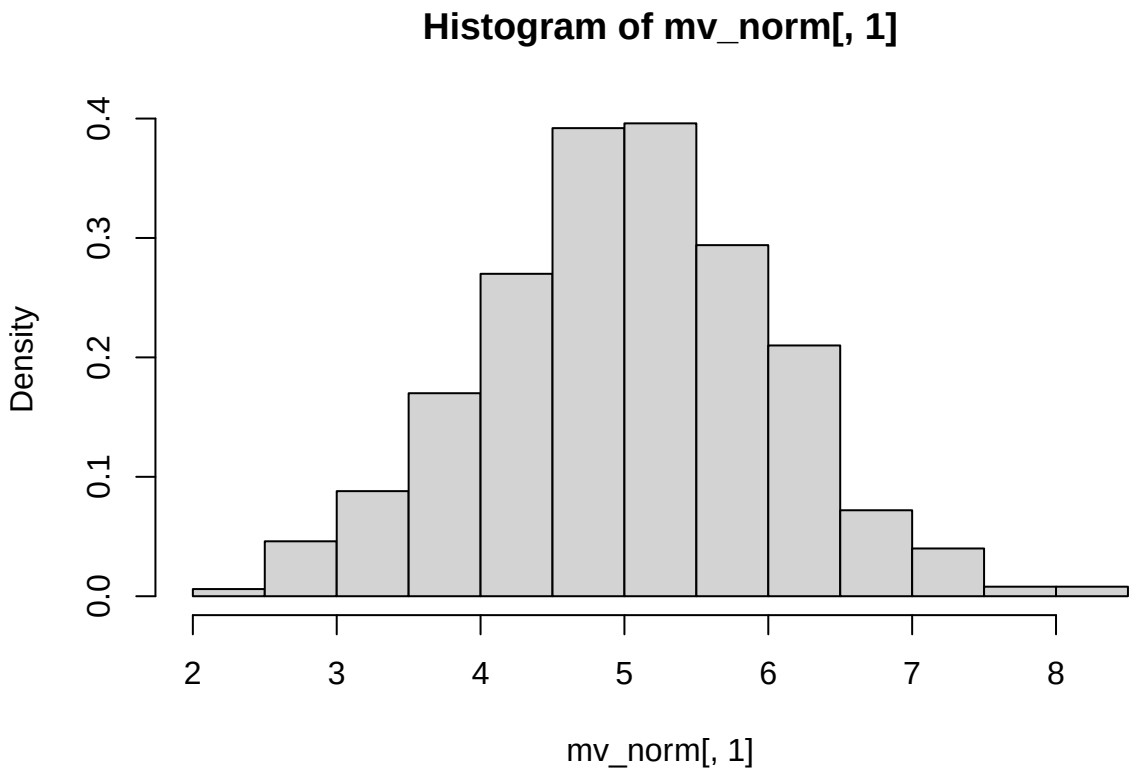
```
apply(mv_norm, 2, var)
```

```
## [1] 1.023554 2.032559
```

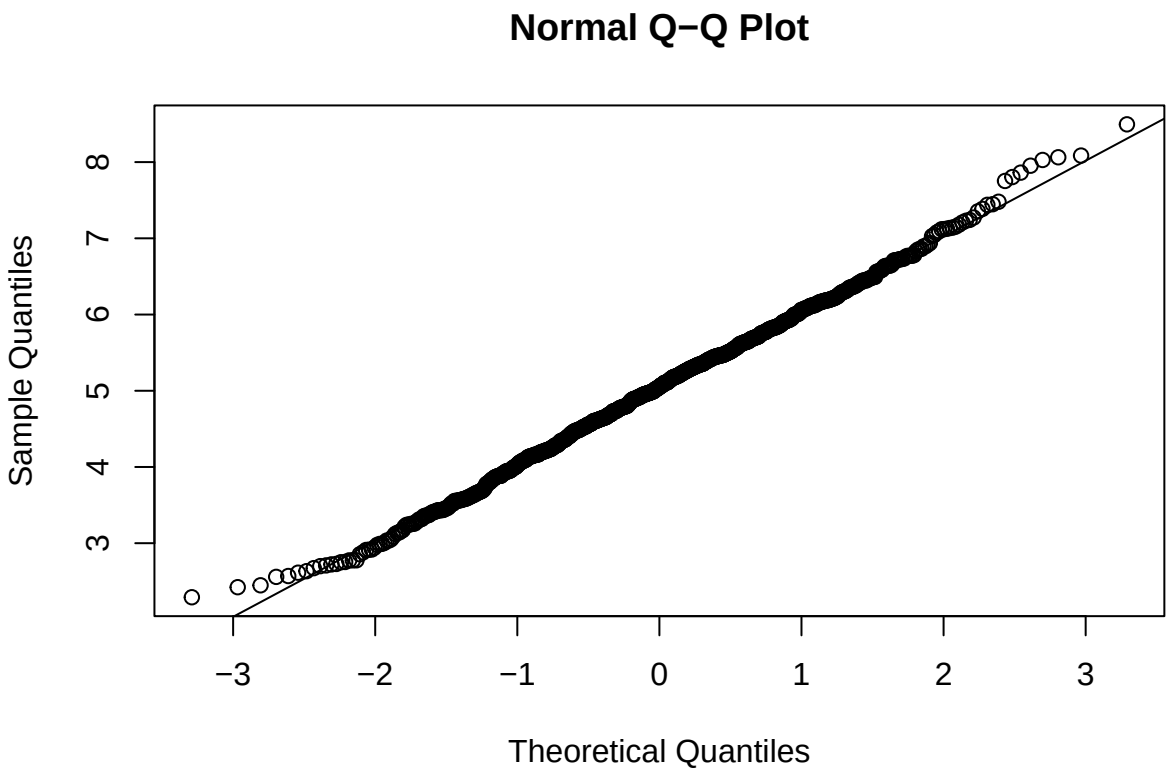
```
cov(mv_norm[,1], mv_norm[,2])
```

```
## [1] 0.4824294
```

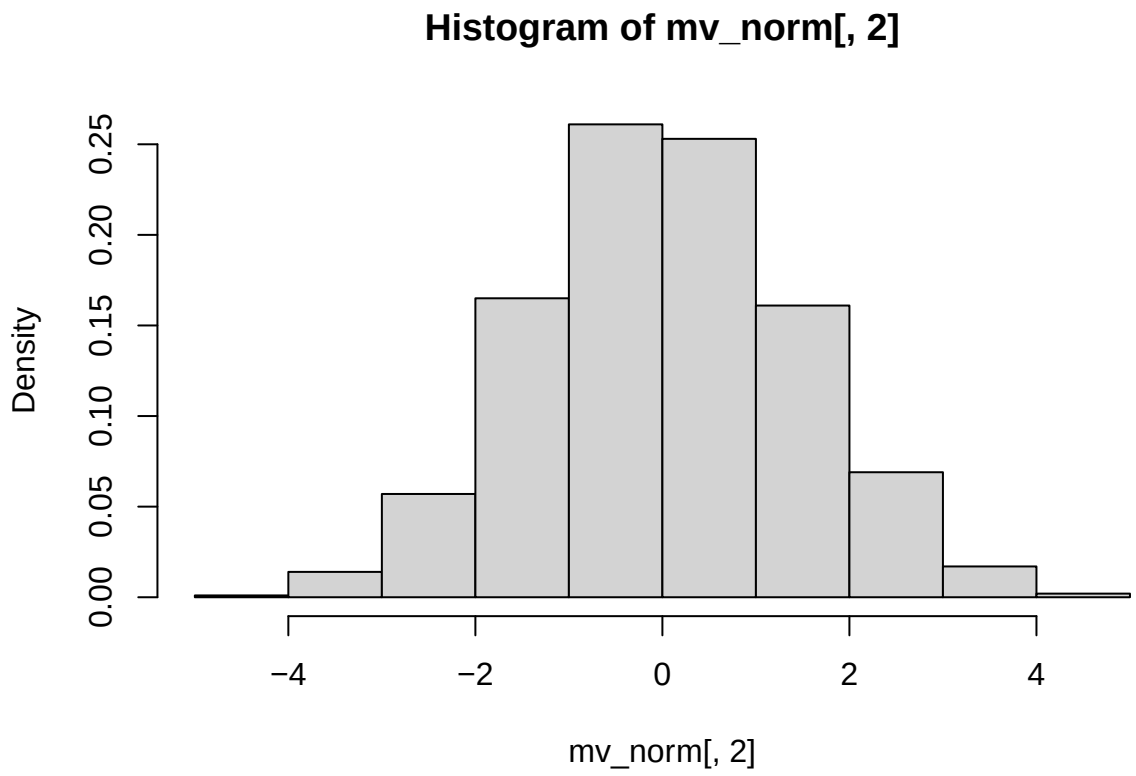
```
hist(mv_norm[,1], probability = T)
```



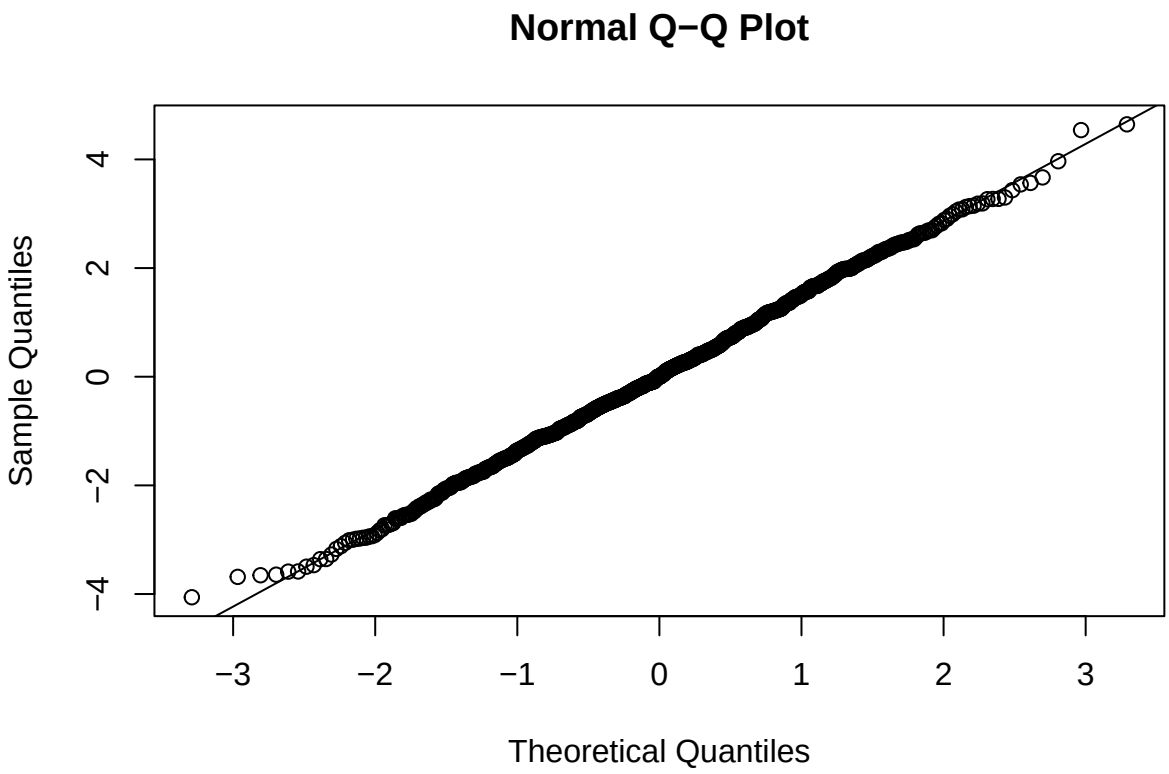

```
qqnorm(mv_norm[,1])
qqline(mv_norm[,1])
```




```
hist(mv_norm[,2], probability = T)
```




```
qqnorm(mv_norm[,2])
qqline(mv_norm[,2])
```



Exercise 42

a)

lET

$$F = \frac{\sqrt{\rho}}{1+\rho(d-1)} \sum_{j=1}^{d} X_j + \sqrt{\frac{1-\rho}{1+\rho(d-1)}} Y$$

Then, because of linearity of expected value:

$$\begin{aligned} E[F] &= E\left[\frac{\sqrt{\rho}}{1+\rho(d-1)} \sum_{j=1}^{d} X_j + \sqrt{\frac{1-\rho}{1+\rho(d-1)}} Y \right] \\ &= \frac{\sqrt{\rho}}{1+\rho(d-1)} \sum_{j=1}^{d} E[X_j] + \sqrt{\frac{1-\rho}{1+\rho(d-1)}} E[Y] \\ &= 0 + 0 \\ &= 0 \end{aligned}$$

Now, note that $E[\varepsilon_i] = E[X_i - \sqrt{\rho}F] = E[X_i] - \sqrt{\rho}E[F] = 0$

Also,

$$
\begin{aligned}
Cov(F, \varepsilon_i) &= E[(F - E[F])(\varepsilon_i - E[\varepsilon_i])] \\
&= E[F\varepsilon_i] \\
&= E[FX_i - \sqrt{\rho}F^2] \\
&= E[FX_i] - \sqrt{\rho}E[F^2]
\end{aligned}
$$

Now, because of independence between Y and X:

$$
\begin{aligned}
E[FX_i] &= E\left[\left(\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d} X_j + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y\right)X_i\right] \\
&= \frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d} E[X_iX_j] + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}E[YX_i] \\
&= \frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d} E[X_iX_j] + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}E[Y]E[X_i] \\
&= \frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d} E[X_iX_j]
\end{aligned}
$$

Recall for $i \neq j$ we have $\rho = Cor(X_i, X_j) = \frac{E[(X_i - E[X_i])(X_j - E[X_j])]}{\sigma_i\sigma_j} = E(X_iX_j)$, since X has standardized margins. Also, $Cov(X_i, X_i) = Var(X_i) = 1$. Therefore:

$$
\begin{aligned}
E[FX_i] &= \frac{\sqrt{\rho}}{1+\rho(d-1)}\left(\sum_{j=1}^{d-1}\rho + 1\right) \\
&= \frac{\sqrt{\rho}((d-1)\rho + 1)}{1+\rho(d-1)} \\
&= \sqrt{\rho}
\end{aligned}
$$

Also, notice that since $E[F] = 0$, then $Var[F] = E[F^2] - E[F]^2 = E[F^2]$, then:

$$
\begin{aligned}
E[F^2] &= Var\left[\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}X_j + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y\right] \\
&= Var\left[\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}X_j\right] + Var\left[\sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y\right] + 2Cov\left[\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}X_j, \sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y\right] \\
&\text{Because of independence between } X, Y \\
&= Var\left[\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}X_j\right] + Var\left[\sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y\right] \\
&\text{Because of standardized margin of } Y \\
&= \frac{\rho}{(1+\rho(d-1))^2}Var\left[\sum_{j=1}^{d}X_j\right] + \frac{1-\rho}{1+\rho(d-1)} \\
&= \frac{\rho}{(1+\rho(d-1))^2}\left(\sum_{i=1}^{d}Var[X_i] + 2\cdot\sum_{1\le i<j\le d}Cov[X_i,X_j]\right) + \frac{1-\rho}{1+\rho(d-1)} \\
&= \frac{\rho}{(1+\rho(d-1))^2}\left(\sum_{i=1}^{d}1 + 2\cdot\sum_{1\le i<j\le d}\rho\right) + \frac{1-\rho}{1+\rho(d-1)} \\
&= \frac{\rho}{(1+\rho(d-1))^2}\left(d + 2\rho\frac{d(d-1)}{2}\right) + \frac{1-\rho}{1+\rho(d-1)} \\
&= \frac{\rho}{(1+\rho(d-1))^2}\left(d + \rho d^2 - \rho d\right) + \frac{1-\rho}{1+\rho(d-1)} \\
&= \frac{2\rho d + \rho^2 d^2 - 2\rho^2 d + 1 - 2\rho + \rho^2}{(1+\rho(d-1))^2} \\
&= \frac{2\rho d + \rho^2 d^2 - 2\rho^2 d + 1 - 2\rho + \rho^2}{2\rho d + \rho^2 d^2 - 2\rho^2 d + 1 - 2\rho + \rho^2} \\
&= 1
\end{aligned}
$$

Therefore, $Var[F] = E[F^2] = 1$, so $Cov(F, \varepsilon_i) = E[FX_i] - \sqrt{\rho}E[F^2] = \rho - \rho = 0$. Also, $Var[X_i] = Var[\sqrt{\rho}F + \varepsilon_i] = \rho Var[F] + Var[\varepsilon_i] + 2\sqrt{\rho}Cov(F, \varepsilon_i)$, therefore, $1 = \rho + Var[\varepsilon_i] \Rightarrow Var[\varepsilon_i] = 1 - \rho$.

Finally:

$$
\begin{aligned}
Cov(\varepsilon_i, \varepsilon_j) &= E[(\varepsilon_i - E[\varepsilon_j])(\varepsilon_j - E[\varepsilon_j])] \\
&= E[\varepsilon_i\varepsilon_j] \\
&= E[(X_i - \sqrt{\rho}F)(X_j - \sqrt{\rho}F)] \\
&= E[X_iX_j] - \sqrt{\rho}E[X_iF] - \sqrt{\rho}E[FX_j] + \rho E[F^2] \\
&= \rho - (\sqrt{\rho})^2 - (\sqrt{\rho})^2 + \rho \\
&= 0
\end{aligned}
$$

b)

To see that this is a one-dimensional factor model, notice that by taking $A = 0$, $B = \sqrt{\rho}$ and $\epsilon_i = \sqrt{1-\rho}Z_i$, we get that $X_i = A + BF + \epsilon_i$, where F, ϵ_i are iid random variables with mean 0 and variance 1. Also, notice

that $cov(F, \epsilon_i) = 0$. Therefore, we have a one-dimensional factor model where we take F as a factor and the error term fulfills the criteria of having 0 expected value and being uncorrelated to the factor (i.e. it can be seen as some sort of noise). We use this idea to generate the realizations through the following code:

```
rmvStNorm <- function(n, d, rho){
  F_n <- rnorm(n)
  Z <- matrix(rnorm(n*d), nrow = n, ncol = d)

  apply(Z, 2, function(Zi) sqrt(rho)*F_n + sqrt(1-rho)*Zi)

}

A <- rmvStNorm(1000, 2, 0.5)
cor(A[,1], A[,2])
```

```
## [1] 0.5233351
```

Exercise 43

Let $X \overset{d}{=} m + \sqrt{W}AZ$, such that $Z \sim N_k(0, I_k)$, $W \geq 0$, $A \in \mathbb{R}^{d\times k}$ and $m \in \mathbb{R}^d$. Assume $E[|W|] < \infty$. Then:

$$\begin{aligned} E[X] &= E(m + \sqrt{W}AZ) \\ &= E(m) + E(\sqrt{W}AZ) \\ &\text{Due to independence between } Z, W \\ &= m + E(\sqrt{W})AE(Z) \\ &\text{Since } Z \sim N_k(0, I_k) \\ &= m \end{aligned}$$

Now, for the covariance, notice that:

$$\begin{aligned} Cov(X) &= E[(X - E[X])(X - E[X])'] \\ &= E[(X - m)(X - m)'] \\ &= E[(m + \sqrt{W}AZ - m)(m + \sqrt{W}AZ - m)'] \\ &= E[(\sqrt{W}AZ)(\sqrt{W}AZ)'] \\ &= E[WAZZ'A'] \\ &\text{Due to linearity and independence between } W, Z \\ &= E[W]AE[ZZ']A' \\ &= E[W]AI_kA' \\ &= E[W]AA' \\ &= E[W]\Sigma \end{aligned}$$

Now, suppose W has an inverse gamma distribution with parameters $\frac{\nu}{2}$ and $\frac{\nu}{2}$, and assume $\nu > 2$. If $X \sim Gamma^{-1}(\alpha, \beta)$ with $\alpha > 1$ and $\beta > 0$, then $E[X] = \frac{\beta}{\alpha - 1}$. Note that since $\nu > 2$, we can use this formula. Then:

$$Cov(X) = \frac{\frac{\nu}{2}}{\frac{\nu}{2} - 1}\Sigma = \frac{\nu}{\nu - 2}\Sigma$$

```r
rmvt <- function(n, nu, m, Sigma){

  AZ <- rmvnorm(n, 0, Sigma)

  W <- nu/rchisq(n, df = nu)

  sim <- t( apply(sqrt(W)*AZ, 1, function(Xi) Xi+m))

  list(Simulation = sim,
       T.mean = m,
       E.mean = apply(sim, 2, mean),
       T.var = nu/(nu-2) * Sigma,
       E.var = cov(sim))
}

# Case nu = 4
nu <- 4
d <- 2
m <- rep(0, d)
Sigma <- matrix(0.3,d,d)
diag(Sigma) <- 1
n <- 50000
sim <- rmvt(n,nu,m,Sigma)

sim$T.mean
```

```
## [1] 0 0
```

```r
sim$E.mean
```

```
## [1] -0.011020001 -0.004035186
```

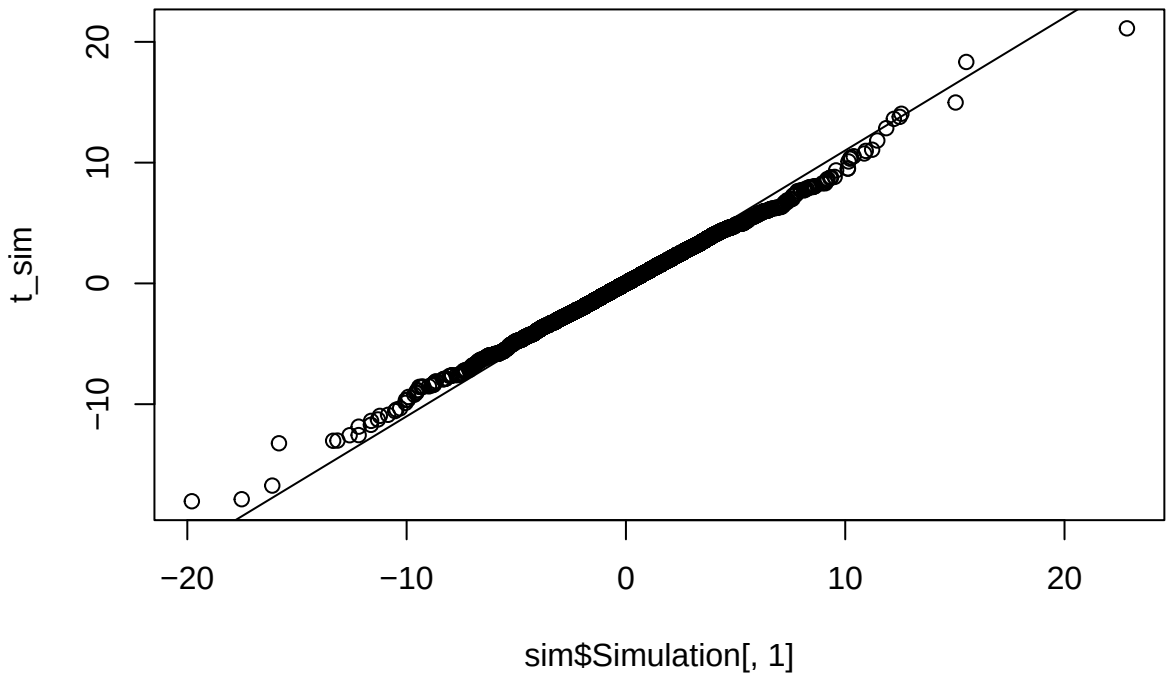
```r
sim$T.var
```

```
##      [,1] [,2]
## [1,]  2.0  0.6
## [2,]  0.6  2.0
```

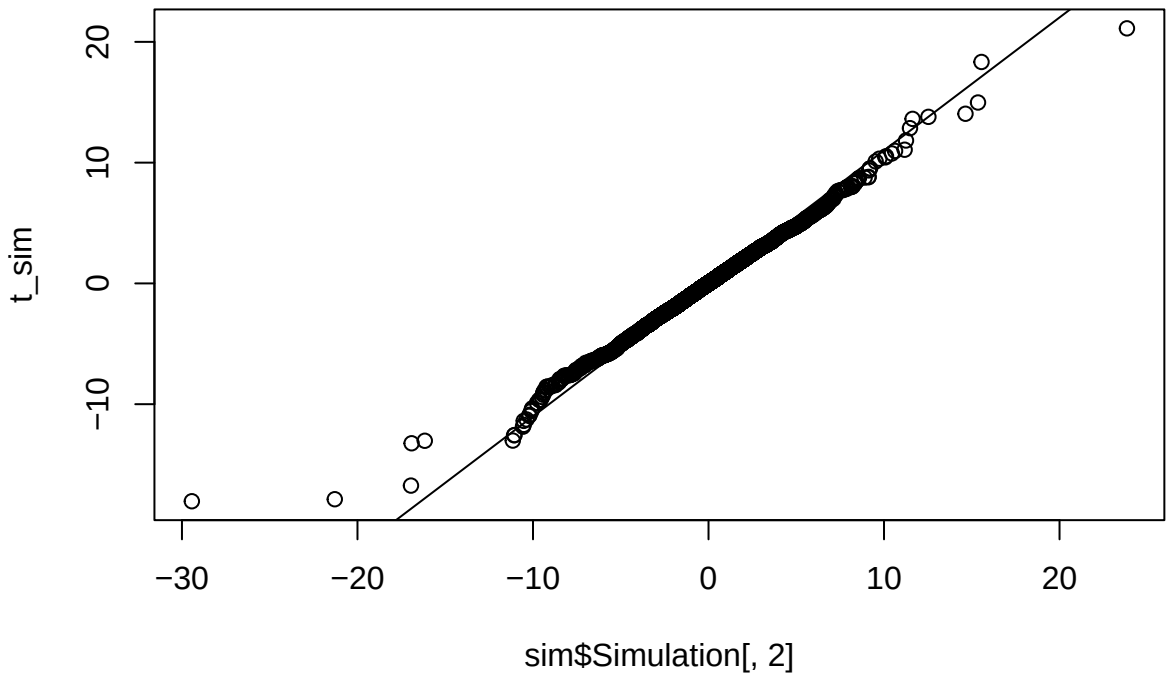
```r
sim$E.var
```

```
##           [,1]      [,2]
## [1,] 2.0349261 0.6227787
## [2,] 0.6227787 2.0276928
```

```r
t_sim <- rt(n, df = nu)
qqplot(sim$Simulation[,1], t_sim)
qqline(t_sim)
```




```
qqplot(sim$Simulation[,2], t_sim)
qqline(t_sim)
```




```
# Case nu = 4
nu <- 13
d <- 5
m <- rep(0, d)
Sigma <- matrix(0.3,d,d)
diag(Sigma) <- 1
n <- 50000
sim <- rmvt(n,nu,m,Sigma)

sim$T.mean
```

```
## [1] 0 0 0 0 0
```

```
sim$E.mean
```

```
## [1]  2.754992e-03  9.973924e-05  1.650072e-03 -1.921888e-03 -1.174239e-02
```

```
sim$T.var
```

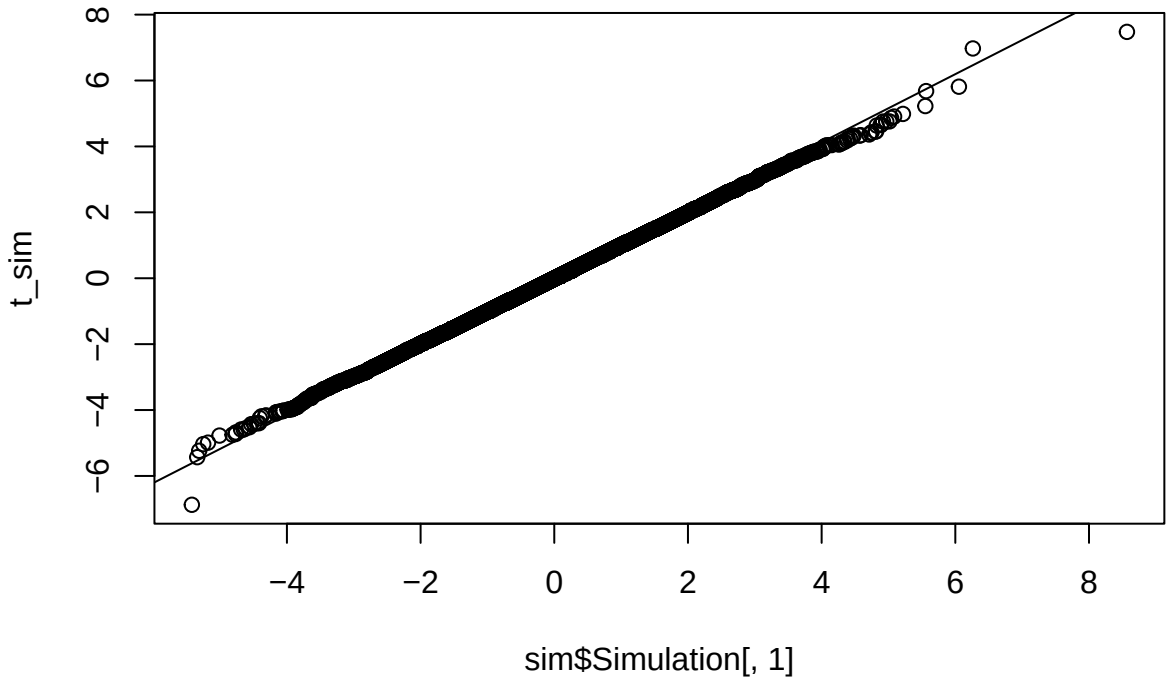
```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.1818182 0.3545455 0.3545455 0.3545455 0.3545455
## [2,] 0.3545455 1.1818182 0.3545455 0.3545455 0.3545455
## [3,] 0.3545455 0.3545455 1.1818182 0.3545455 0.3545455
## [4,] 0.3545455 0.3545455 0.3545455 1.1818182 0.3545455
## [5,] 0.3545455 0.3545455 0.3545455 0.3545455 1.1818182
```

```
sim$E.var
```

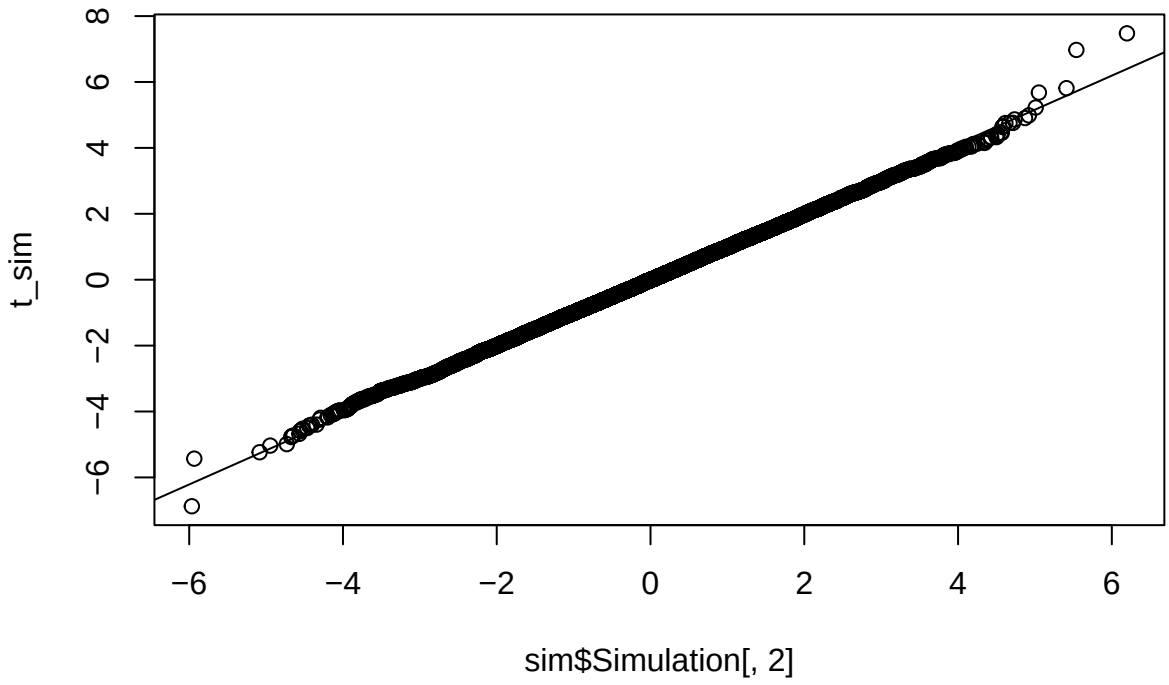
```
##           [,1]      [,2]      [,3]      [,4]      [,5]
```

```
## [1,] 1.1836127 0.3545785 0.3477795 0.3547922 0.3592822
## [2,] 0.3545785 1.1911948 0.3547761 0.3685591 0.3645185
## [3,] 0.3477795 0.3547761 1.1762693 0.3498674 0.3462915
## [4,] 0.3547922 0.3685591 0.3498674 1.1760587 0.3555388
## [5,] 0.3592822 0.3645185 0.3462915 0.3555388 1.1736485
```

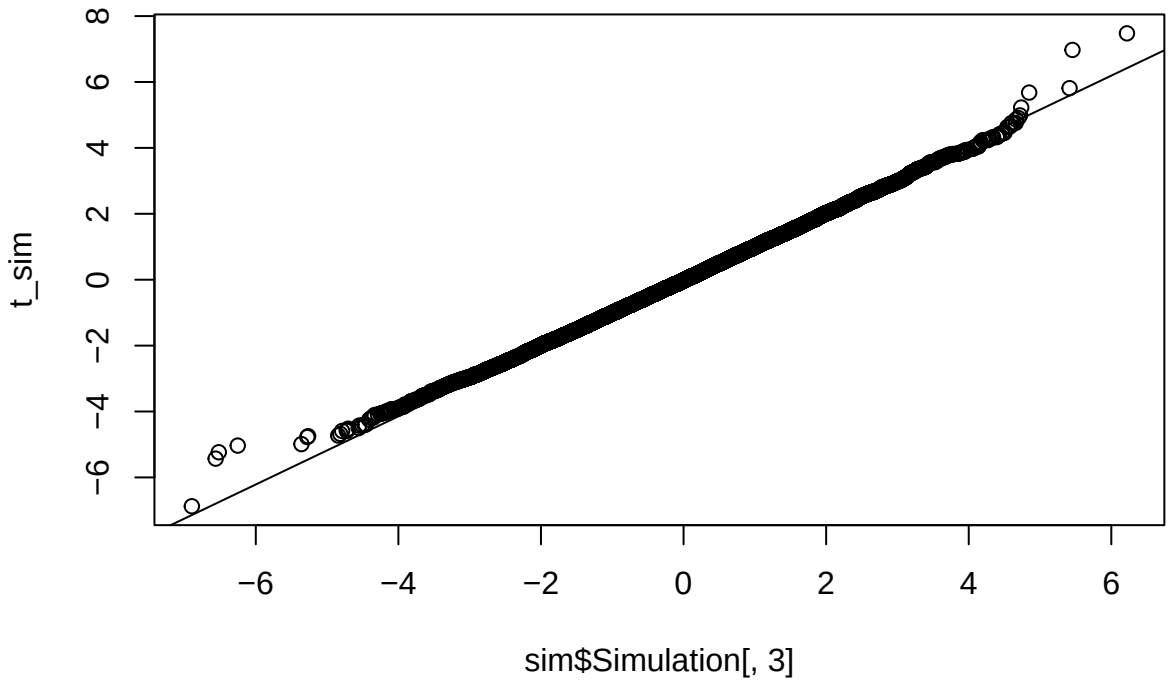
```
t_sim <- rt(n, df = nu)
qqplot(sim$Simulation[,1], t_sim)
qqline(t_sim)
```



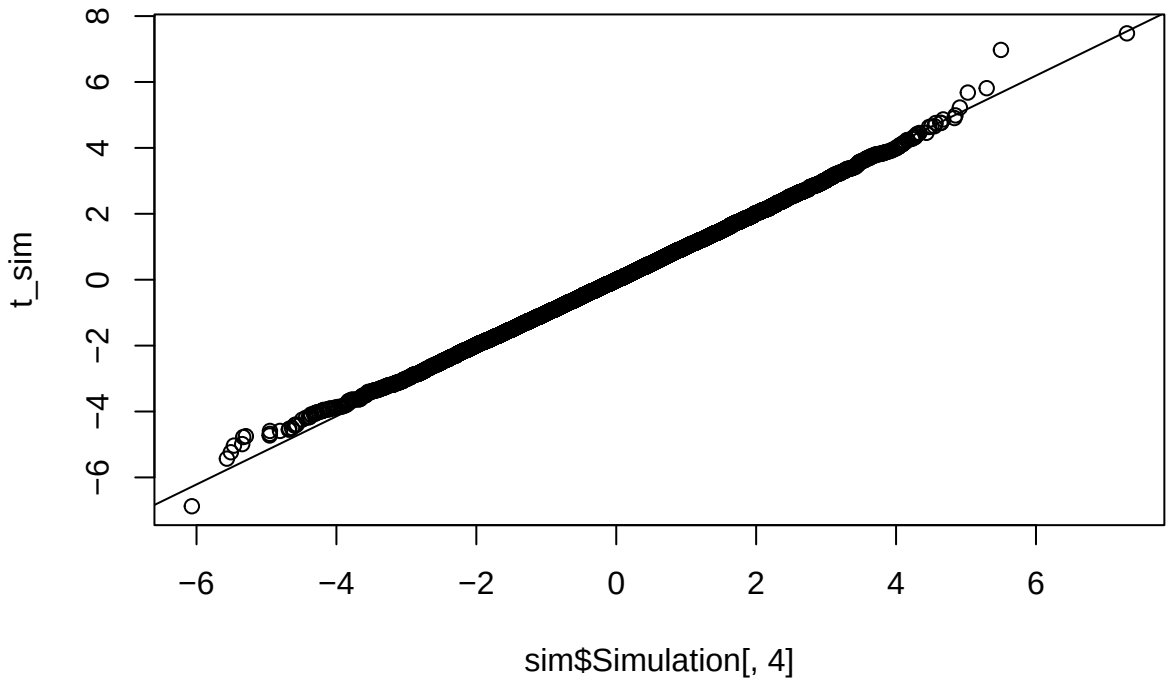

```
qqplot(sim$Simulation[,2], t_sim)
qqline(t_sim)
```



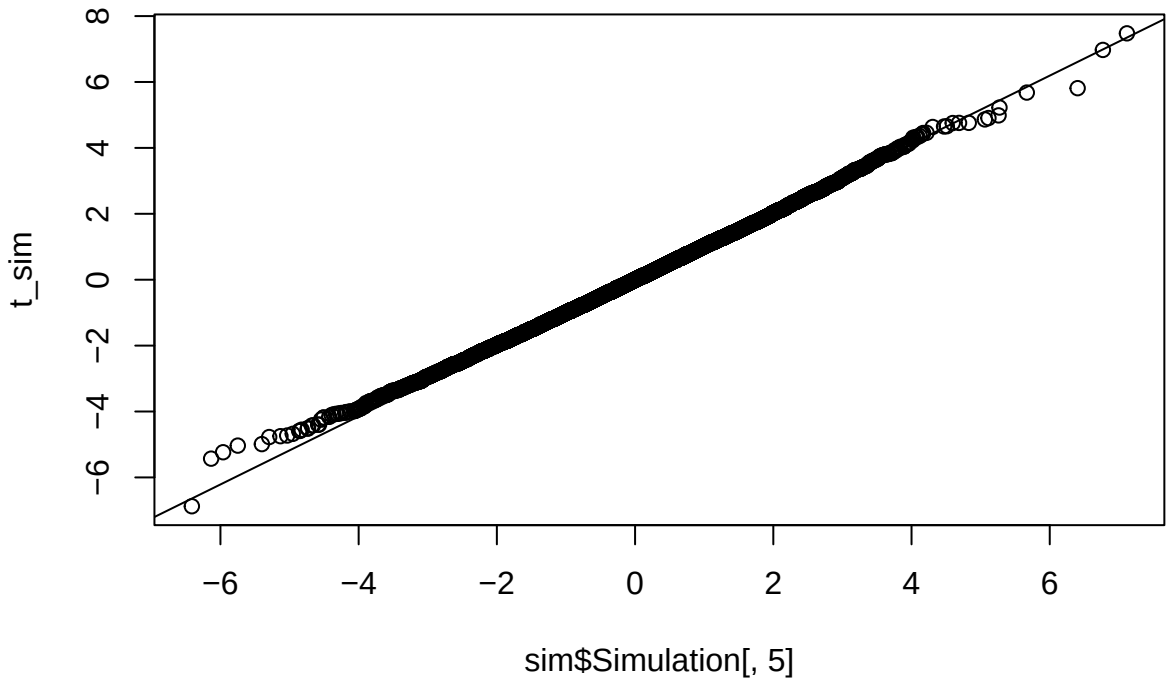

```
qqplot(sim$Simulation[,3], t_sim)
qqline(t_sim)
```




```
qqplot(sim$Simulation[,4], t_sim)
qqline(t_sim)
```




```
qqplot(sim$Simulation[,5], t_sim)
qqline(t_sim)
```



Exercise 44

Let $-1 < \rho < 1$, and suppose (X_1, X_2) has a bivariate normal distribution with correlation ρ.

Recall $f_{X|Y=y}(x, y) = \frac{f(x,y)}{f_y(y)}$. Then, $f_{X_2|X_1=x} = \frac{f(x_1,x_2)}{f_{X_1}(x_1)}$. Note that in our case:

$$f(x_1, x_2) = \frac{1}{2\pi\sqrt{1-\rho^2}} exp\left[-\frac{x_1^2 - 2\rho x_1 x_2 + x_2^2}{2(1-\rho^2)}\right]$$

$$f_{X_1}(x_1) = \frac{1}{\sqrt{2\pi}} exp\left[-\frac{x_1^2}{2}\right]$$

Then:

$$
\begin{aligned}
f_{X_2|X_1=x}(x_2) &= \frac{\frac{1}{2\pi\sqrt{1-\rho^2}} exp\left[-\frac{x^2-2\rho x x_2+x_2^2}{2(1-\rho^2)}\right]}{\frac{1}{\sqrt{2\pi}} exp\left[-\frac{x^2}{2}\right]} \\
&= \frac{\sqrt{2\pi}}{2\pi\sqrt{1-\rho^2}} exp\left[-\frac{x^2-2\rho x x_2+x_2^2}{2(1-\rho^2)}+\frac{x^2}{2}\right] \\
&= \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}} exp\left[\frac{-x^2-2\rho x x_2-x_2^2+(1-\rho^2)x^2}{2(1-\rho^2)}\right] \\
&= \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}} exp\left[\frac{-x_2^2+2\rho x x_2-\rho^2 x^2}{2(1-\rho^2)}\right] \\
&= \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}} exp\left[-\frac{(x_2-x\rho)^2}{2(1-\rho^2)}\right]
\end{aligned}
$$

Which corresponds to the density function of a normal distribution of the form $N(\rho x, 1-\rho^2)$. This means that its distribution function is of the form $P(X_2 \leq x|X_1 = x) = \Phi\left(\frac{x-\rho x}{\sqrt{1-\rho^2}}\right)$, where Φ is the distribution function for a scalar standard normal random variable. Therefore:

$$
\begin{aligned}
\lim_{x\to-\infty} P(X_2 \leq x|X_1 = x) &= \lim_{x\to-\infty} \Phi\left(\frac{x-\rho x}{\sqrt{1-\rho^2}}\right) \\
&= \lim_{x\to-\infty} \Phi\left(\frac{(1-\rho)x}{\sqrt{1-\rho^2}}\right) \\
&\text{Note } \Phi \text{ is a distribution function, and } \rho \text{ is a constant, therefore:} \\
&= 0
\end{aligned}
$$

Finally, suppose (X_1, X_2) has a bivariate t distribution $t_2(\nu, 0, P)$, where P is the correlation matrix with off-diagonal elements ρ. As given by the exercise:

$$
\sqrt{\frac{\nu+1}{\nu+x^2}}\frac{X_2-\rho X_1}{\sqrt{1-\rho^2}}\Big| X_1 = x \sim t_{\nu+1}
$$

Then:

$$
\begin{aligned}
\lim_{x\to-\infty} P\left(\sqrt{\frac{\nu+1}{\nu+x^2}}\frac{X_2-\rho X_1}{\sqrt{1-\rho^2}} \leq x\Big| X_1 = x\right) &= \lim_{x\to-\infty} F_{t_{\nu+1}}\left(\sqrt{\frac{\nu+1}{\nu+x^2}}\frac{x-\rho x}{\sqrt{1-\rho^2}}\right) \\
&= \lim_{x\to-\infty} F_{t_{\nu+1}}\left(\frac{\sqrt{\nu+1}}{x\sqrt{\frac{\nu}{x^2}+1}}\frac{x(1-\rho)}{\sqrt{1-\rho^2}}\right) \\
&= \lim_{x\to-\infty} F_{t_{\nu+1}}\left(\frac{\sqrt{\nu+1}}{\sqrt{\frac{\nu}{x^2}+1}}\frac{1-\rho}{\sqrt{1-\rho^2}}\right) \\
&= F_{t_{\nu+1}}\left(\sqrt{\nu+1}\frac{1-\rho}{\sqrt{1-\rho^2}}\right) \neq 0
\end{aligned}
$$

Exercise 47

```r
set.seed(19908)
U <- runif(1000)
```

a)

```r
mean(U)
```

```
## [1] 0.496057
```

```r
var(U)
```

```
## [1] 0.08148008
```

```r
sd(U)
```

```
## [1] 0.2854472
```

b)

We know that if $U \sim \mathcal{U}(0,1)$, then $E[U] = 0.5$, $Var(U) = \frac{1}{12}$, and $sd(U) = \frac{1}{\sqrt{12}}$

```r
real <- c(0.5, 1/12, 1/sqrt(12))
sample <- c(mean(U), var(U), sd(U))

data.frame(Real = real,
           Sample = sample,
           Abs.Diff = abs(real - sample),
           Rel.Diff = abs(real-sample)/real,
           row.names = c("Mean", "Variance", "Std.Dev."))
```

```
##                Real     Sample    Abs.Diff   Rel.Diff
## Mean     0.50000000 0.49605698 0.003943025 0.00788605
## Variance 0.08333333 0.08148008 0.001853252 0.02223903
## Std.Dev. 0.28867513 0.28544716 0.003227975 0.01118203
```

It is possible to observe our sample does not match the values completely. However, this is expected given the (pseudo-)randomness of sampling from the distribution. It is also important to note that, while 1000 points sounds like a lot, it might not be so. However, due to Law of Large Numbers, the values of the sample mean, variance and standard deviation converge to the theoretical ones when $n \to \infty$.

c)

Recall since $U \sim \mathcal{U}(0,1)$, then $P(U \leq 0.6) = 0.6$. Let's test this with our sample:

```r
sum(U <= 0.6)/length(U)
```

```
## [1] 0.61
```

```r
sum(U <= 0.6)/length(U) - 0.6
```

```
## [1] 0.01
```

```r
(sum(U <= 0.6)/length(U) - 0.6)/0.6
```

```
## [1] 0.01666667
```

Again, this value is not exact due to the same reasons given above. However, the estimated value does not differ significantly from the real value, as seen by both the absolute and relative errors.

Exercise 48

```
U1 <- runif(10000)
U2 <- runif(10000)
```

a)

Due to linearity, $E[U_1 + U_2] = E[U_1] + E[U_2] = 0.5 + 0.5 = 1$

```
data.frame(Real = 1,
           Mean.of.sum = mean(U1+U2),
           Sum.of.means = mean(U1)+mean(U2),
           Diff= abs(1-mean(U1+U2)),
           row.names = c("Mean"))
```

```
##      Real Mean.of.sum Sum.of.means        Diff
## Mean    1   0.9981552    0.9981552 0.001844755
```

b)

Recall $Var(U_1+U_2) = Var(U_1)+Var(U_2)+2cov(U_1, U_2)$. Since we are assuming U_1 and U_2 are independent, then their covariance is zero and the variance of the sum should equal the sum of the variances. This might not be the case if the sampling is not really independent. Let's verify this:

```
sum.var <- var(U1 + U2)
var.sum <- var(U1) + var(U2)

data.frame(Var.of.sum = sum.var,
           Sum.of.vars = var.sum,
           row.names = c("Value"))
```

```
##       Var.of.sum Sum.of.vars
## Value   0.163099   0.1647748
```

As we can see, they are not the same. Let's check what happens if we include the covariance in the sum of variances:

```
data.frame(Var.of.sum = sum.var,
           Sum.of.vars = var.sum,
           Cov = cov(U1, U2),
           Theoretical.var = var.sum + 2*cov(U1, U2),
           row.names = c("Value"))
```

```
##       Var.of.sum Sum.of.vars           Cov Theoretical.var
## Value   0.163099   0.1647748 -0.0008379225        0.163099
```

c)

```
sum((U1+U2) <= 1.5)/length(U1+U2)
```

```
## [1] 0.8792
```

d)

```
sum((sqrt(U1)+sqrt(U2)) <= 1.5)/length(U1+U2)
```

```
## [1] 0.6582
```

Exercise 49

```
U1 <- runif(10000)
U2 <- runif(10000)
U3 <- runif(10000)
```

a)

```
mean(U1+U2+U3)
```

```
## [1] 1.497473
```

b)

```
var(U1+U2+U3)
```

```
## [1] 0.2448316
```

```
var(U1) + var(U2) + var(U3)
```

```
## [1] 0.2456247
```

c)

```
mean(sqrt(U1+U2+U3))
```

```
## [1] 1.204957
```

d)

```
sum( sqrt(U1)+sqrt(U2)+sqrt(U3) >= 0.8 ) / length(sqrt(U1)+sqrt(U2)+sqrt(U3))
```

```
## [1] 0.9964
```

Exercise 50

Note that if a student guesses one question, there is a probability of 0.5 for being right and the same probability for being wrong. We can assume two things: for any individual student the answer to each question is independent to the answer to previous or following questions, and for every student their answers are independent to the answers of other students. Therefore we can simulate these exam results by sampling from a binomial distribution with $p = 0.5$, where we assume 1 corresponds to a correct answer and 0 to an incorrect answer. We need to do this for every student and for every question, so 100×20 times. Let's store this simulation in a matrix where the rows represent each student and the columns each question's mark:

```
test <- matrix(sample(c(0,1),100*20,replace = T), nrow = 100, ncol = 20)
head(test)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    0    1    1    0    1    1    1    0    0     1     1     1     1     0
## [2,]    1    1    0    0    1    0    0    1    0     1     1     0     1     0
## [3,]    1    0    0    1    0    1    1    1    1     0     0     0     0     1
## [4,]    1    1    0    1    0    0    0    1    1     0     1     1     0     1
## [5,]    1    1    1    1    0    1    0    1    1     1     0     0     1     0
## [6,]    1    1    1    0    1    0    0    1    0     0     0     1     1     1
```

```
##      [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     1     1     1     0     0     1
## [2,]     1     0     0     0     1     0
## [3,]     0     1     0     1     1     1
## [4,]     1     0     1     1     0     0
## [5,]     1     0     1     0     1     1
## [6,]     0     1     1     0     0     0
```

a)

We calculate the average score and standard deviation (on a scale from 0-20):

```
mean(apply(test, 1, sum))
```

```
## [1] 10.3
```

```
sd(apply(test, 1, sum))
```

```
## [1] 2.057261
```

If we want to convert it to percentages, we do this in the following way:

```
mean(apply(test, 1, sum)/20)
```

```
## [1] 0.515
```

```
sd(apply(test, 1, sum)/20)
```

```
## [1] 0.1028631
```

b)

Using the percentage version of part a), we can estimate how many students score 30% or higher:

```
sum(apply(test, 1, sum)/20 >= 0.3)/100
```

```
## [1] 0.99
```

Exercise 51

```
X <- rbinom(10000, 20, 0.3)
```

a)

```
estim <- sum(X <= 5)/length(X)
real <- pbinom(5, 20, 0.3)

data.frame(Estimated = estim,
           Real = real,
           row.names = "P(X <= 5)")
```

```
##           Estimated      Real
## P(X <= 5)    0.4195 0.4163708
```

b)

```
estim <- sum(X == 5)/length(X)
real <- choose(20,5)*(0.3)^(5)*(1-0.3)^(20-5)

data.frame(Estimated = estim,
           Real = real,
           row.names = "P(X = 5)")
```

```
##          Estimated      Real
## P(X = 5)    0.1816 0.1788631
```

c)

```
data.frame(Estimated = mean(X),
           Real = 20*0.3,
           row.names = "E[X]")
```

```
##      Estimated Real
## E[X]    6.0141    6
```

d)

```
data.frame(Estimated = var(X),
           Real = 20*0.3*0.7,
           row.names = "Var[X]")
```

```
##        Estimated Real
## Var[X]   4.29293  4.2
```

e, f, g)

```
estim <- quantile(X, probs = c(0.95, 0.99, 0.999999))
real <- qbinom(c(0.95, 0.99, 0.999999), 20, 0.3)

data.frame(Estimated = estim,
           Real = real,
           row.names = c("Q95", "Q99", "Q99.9999"))
```

```
##          Estimated Real
## Q95          10.00    9
## Q99          11.00   11
## Q99.9999     13.99   16
```

It is possible to observe a significant difference in the estimated and real 99.9999% quantile. Extreme quantiles (close to 100% or close to 0% in our case due to the shape of the binomial distribution) are hard to compute through simulation due to the fact that extreme realizations are, as their name implies, rare occurrences. Therefore, it would be necessary to have a (considerably) larger sample size to be able to improve accuracy, but even in these cases it is almost impossible to guarantee the occurrence of extreme values on either tail, let alone both in a way that would accurately estimate extreme quantiles.

Exercise 52

a)

```
ranbin1 <- function(n, size, prob) {
  cumbins <- pbinom(0 : (size - 1), size, prob)
  singlenumber <- function() {
    x <- runif(1)
    sum(x > cumbins)
  }
  replicate(n, singlenumber())
}
```

The function ranbin1 simulates binomial pseudorandom variates using inversion method. It takes three inputs:

1. **n:** The number of realizations to generate.
2. **size:** Size parameter for the binomial distribution, i.e. the number of independent experiments to be executed.
3. **prob:** The probability of success in each independent experiment.

With the parameters given above, the code does the following:

1. Generates value of the cumulative distribution function for all possible outcomes except the one where all experiments are successful. I.e. it calculates $P(X \leq n)$ for $n = 0, ..., (size - 1)$
2. Singlenumber generates a single realization of a pseudorandom variable distributed as $Y \sim \mathcal{U}(0, 1)$, and then it calculates for the previously mentioned $P(X \leq n)$, for how many it holds that $Y > P(X \leq n)$, for $n = 0, ..., (size - 1)$. As will be explained below, this is essentially calculating the quantile function of the distribution evaluated in a standard uniformly distributed realization.
3. The process of calculating a new Y as described above and comparing it to the cdf values is repeated n times, and each of these is a realization of a binomial distribution with the given parameters. This is done via the replicate() function.

This works because the inversion method exploits the fact that if $X \sim F$, then we can generate realizations of X by calculating $Q_F(U)$, where U is uniformly distributed in $(0, 1)$. The singlenumber function is essentially brute forcing the calculation of the quantile for this case (recall $Q_F(u) = inf\{x : F(x) \geq u\}$ which is exactly what singlenumber calculates), given that the binomial random variable is discrete. This is not particularly efficient.

b)

```
# 1000 realizations
t_ranbin1_1 <-system.time(ranbin1(1000,10,0.4))[3]
t_rbinom_1 <-system.time(rbinom(1000,10,0.4))[3]

# 10,000 realizations
t_ranbin1_10 <-system.time(ranbin1(10000,10,0.4))[3]
t_rbinom_10 <-system.time(rbinom(10000,10,0.4))[3]

# 100,000 realizations
t_ranbin1_100 <-system.time(ranbin1(100000,10,0.4))[3]
t_rbinom_100 <-system.time(rbinom(100000,10,0.4))[3]

# Results
data.frame(`1,000` = c(t_ranbin1_1, t_rbinom_1),
           `10,000` = c(t_ranbin1_10, t_rbinom_10),
```

```
           `100,000` = c(t_ranbin1_100, t_rbinom_100),
           row.names = c("ranbin1", "rbinom"))
```

```
##         X1.000 X10.000 X100.000
## ranbin1      0    0.03     0.35
## rbinom       0    0.00     0.00
```

It is possible to see that in all cases, the rbinom function is faster than the given implementation. Especially for more realizations, the difference is very large. As stated above, the way ranbin1 calculates the quantile function is not particularly efficient, and it must also be taken into account that the calculation and storing of the cdf values takes up time and space.

Exercise 53

a)

```
ranbin2 <- function(n, size, prob) {
  singlenumber <- function(size, prob) {
    x <- runif(size)
    sum(x < prob)
  }
  replicate(n, singlenumber(size, prob))
}
```

Same as in the previous exercise, the function ranbin2 simulates binomial pseudorandom variates. It takes three inputs:

1. **n:** The number of realizations to generate.
2. **size:** Size parameter for the binomial distribution, i.e. the number of independent experiments to be executed.
3. **prob:** The probability of success in each independent experiment.

In this case, however, as opposed to the previous exercise, note that there is no brute-forcing of the cdf being calculated. Rather, an amount of standard uniformly distributed numbers equal to the number of independent experiments are generated, and it is counted how many of these are less than the given probability of success. I.e. it counts how many times the generated value is lower than the given success probability, which given the characteristics of the uniform distribution (i.e. $F(x) = x$ for $x \in [0, 1]$), this event has exactly the same probability of occuring as an experiment success. That is, we are essentially running a sequence of Bernoulli experiments every time singlenumber() is run, and this counts how many were a success. This is then repeated n times through the replicate() call.

b)

```
# Size 10
t_ranbin2_10 <-system.time(ranbin2(10000,10,0.4))[3]
t_rbinom_10 <-system.time(rbinom(10000,10,0.4))[3]

# Size 100
t_ranbin2_100 <-system.time(ranbin2(10000,100,0.4))[3]
t_rbinom_100 <-system.time(rbinom(10000,100,0.4))[3]

# Size 1000
t_ranbin2_1000 <-system.time(ranbin2(10000,1000,0.4))[3]
t_rbinom_1000 <-system.time(rbinom(10000,1000,0.4))[3]
```

```r
# Results
data.frame(`10` = c(t_ranbin2_10, t_rbinom_10),
           `100` = c(t_ranbin2_100, t_rbinom_100),
           `1,000` = c(t_ranbin2_1000, t_rbinom_1000),
           row.names = c("ranbin2", "rbinom"))
```

```
##          X10 X100 X1.000
## ranbin2 0.03 0.06   0.16
## rbinom  0.00 0.00   0.00
```

Again, as in the previous exercise, it is possible to see that the rbinom implementation is much quicker than the ranbin2, especially when dealing with a higher value of *size* parameter. This might be due to the fact that each time the experiment is run, many realizations of the uniform random variable need to be generated and stored, in particular for n realizations of a size s run, $n \times s$ uniformly distributed random variable realizations must be generated and stored. This might take up time.

Exercise 54

a)

```r
ranbin3 <- function(n, size, prob) {
  singlenumber <- function(size, prob) {
    k <- 0
    U <- runif(1)
    X <- numeric(size)
    while(k < size) {
      k <- k + 1
      if(U <= prob) {
        X[k] <- 1
        U <- U / prob
      } else {
        X[k] <- 0
        U <- (U - prob) / (1 - prob)
      }
    }
    sum(X)
  }
  replicate(n, singlenumber(size, prob))
}

# 100 pseudorandom numbers from the binomial(20, 0.4)
bin20_40 <- ranbin3(100, 20, 0.4)
bin20_40
```

```
##   [1]  7 11  8 10 10 10  5 11 11 14  8  8  9  8  9  8  4  8  4  9 10  7 10 10  4
##  [26] 11  8  6  6  8  8  7  9  6 11  7  6  6  7 10  7  9  9 12  9  5 10 11  8  5
##  [51]  7 11  8 10  7  9  5  6  5 10  5 10  6 13  7  9  9 10 12  6  8 10  6  7  8
##  [76]  7  6  9  3  7  8  8  7  8  9  9 11  7 11  9  8  7  9  7 11  7 12  5  8  9
```

```r
# 100 pseudorandom numbers from the binomial(500, 0.7)
bin500_70 <- ranbin3(100, 500, 0.7)
bin500_70
```

```
##   [1] 350 353 340 359 345 352 348 337 350 355 347 369 368 352 353 355 347 350
```

```
##  [19] 350 352 352 347 361 352 354 351 351 338 359 363 343 360 355 347 347 360
##  [37] 364 363 347 359 362 345 352 338 342 355 357 343 354 348 356 355 348 363
##  [55] 365 360 356 353 354 338 341 361 355 347 372 344 349 348 361 347 356 351
##  [73] 340 368 367 354 348 346 369 349 352 349 345 369 341 364 351 362 346 331
##  [91] 335 336 356 373 342 354 356 369 350 343
```

b)

Note that if U is a standard uniformly distributed random variable, and assume $p \in (0, 1)$, then:

$$P\left(\frac{U}{p} \leq x \mid U < p\right) = P\left(U \leq px \mid U < p\right) = \begin{cases} 1 & \text{if } px \geq p \\ px & \text{if } 0 \leq px < p \\ 0 & \text{if } px < 0 \end{cases} = \begin{cases} 1 & \text{if } x \geq 1 \\ x & \text{if } 0 \leq x < 1 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Which means the conditional distribution of $\frac{U}{p}$ given $U < p$ is standard uniform $\mathcal{U}(0, 1)$.

c)

Similar to point b), note that:

$$P\left(\frac{U-p}{1-p} \leq x \mid U > p\right) = P\left(U \leq x(1-p)+p \mid U > p\right) = \begin{cases} 1 & \text{if } x(1-p)+1 \geq 1 \\ x(1-p)+p & \text{if } p < x(1-p)+p < 1 \\ 0 & \text{if } x(1-p)+p < p \end{cases} = \begin{cases} 1 & \text{if } x \geq 1 \\ x & \text{if } 0 \leq x < 1 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Which, again, corresponds to a standard uniform $(U)(0, 1)$

d)

Again, the function ranbin3 simulates binomial pseudorandom variates. It takes three inputs:

1. **n:** The number of realizations to generate.
2. **size:** Size parameter for the binomial distribution, i.e. the number of independent experiments to be executed.
3. **prob:** The probability of success in each independent experiment.

However, in this case, contrary to the previous exercise, not many realizations of a standard uniform random variable are generated, but rather only one for each generation of a single binomial realization. Same as in the previous exercise, if this uniform random value is less than the probability of success, this is stored as a success (i.e. stored as a 1) (see previous exercise to see why this works), while if it's not, then it is not stored as a success (i.e. stored as a 0). However, instead of generating a new uniform random realization for each check, the function exploits the results in b) and c) to create a new realization out of the previous one, which, in turn, will also have a standard uniform distribution. As such, this can be done iteratively and we will always get a uniform random distributed number to which we can apply the logic from exercise 53. This process is then repeated n times through the replicate() call.

Exercise 55

Let's do it by random trials:

```r
numPeople <- function(p, nsim = 10000){

  estim_p <- 0
```

```r
  n <- 1

  while(estim_p < p){
    n <- n+1

    bdays <- replicate(nsim, sample(1:365, n, replace = T))

    repeated_bdays <- apply(bdays, 2, function(sim) length(sim) > length(unique(sim)))

    estim_p <- sum(repeated_bdays)/nsim
  }

  list(n.people = n,
       prob = estim_p)
}

numPeople(1/2)
```

```
## $n.people
## [1] 23
##
## $prob
## [1] 0.508
```

This means we need 23 people for the probability of at least two of them sharing a birthday to be greater than 0.5. To justify this, let's rather look at the probability of the complement of this, that is the probability that everyone has a different birthday. We assume all years have exactly 365 days and that every birth date has an equal probability, and that every person's birth date is independent to everyone else. Due to pigeonhole principle, we have an upper bound for the number of people being 365. This is not interesting, however. For the first person, the probability of them having a unique birthday is 1. If you add a second person, the probability is 364/365. If you introduce a third one, he cannot have either of the two dates so the ratio of favorable to all scenarios is 363/365. We can repeat this logic iteratively to get that for the n-th introduced person, the ratio of favorable to all scenarios (i.e. them having a different birthday to all previous ones) is $(365-(n-1))/365$. Due to independence, the probability of none of them sharing a birthday is:

$$P(n \text{ people not sharing a birthday}) = \prod_{i=1}^{n} \frac{366-i}{365}$$

Note that this is clearly decreasing in n as for $i \geq 2$ we have that $366-(i+1) < 366-i < 365$. This means that, conversely, the probability of is complement (which is exactly what we are interested in and given by $1 - P(n \text{ people not sharing a birthday})$), is increasing in n. Therefore, we just need to find the first value of n for which either $P(n \text{ people not sharing a birthday}) < 0.5$. Note that:

$$\prod_{i=1}^{22} \frac{366-i}{365} \approx 0.5243 > 0.5$$

$$\prod_{i=1}^{23} \frac{366-i}{365} \approx 0.4927 < 0.5 \Rightarrow 1 - P(n \text{ people not sharing a birthday}) = P(\text{At least two people share a birthday}) > 0.5$$

Exercise 56

Let's do it by random trials:

```r
pRealRoots <- function(nsim = 10000){

  sim_coeffs <- replicate(nsim, runif(3, min = -1, max = 1))

  real_roots <- apply(sim_coeffs, 2, function(coeffs)
    coeffs[2]^2 - 4*coeffs[1]*coeffs[3] >= 0 )

  estim_p <- sum(real_roots)/nsim

  return(estim_p)
}

pRealRoots()
```

```
## [1] 0.6263
```

To conclude this analytically:

Firstnote that $F_{b^2}(x) = P(b^2 \leq x) = P(-\sqrt{x} \leq b \leq \sqrt{x}) = \sqrt{x}$. Therefore, $f_{b^2}(x) = \frac{1}{2\sqrt{z}}$.

Due to assumed independence, to get the density of $4ac$ we can use convolutions, taking $V = ac, Y = c$, through the random variable $(V/Y, Y)$ for which the Jacobian determinant is $\frac{1}{|Y|}$. Thus, if $v/y \in (-1, 1)$ we have

$$f_{(V,Y)}(v, y) = \frac{1}{|y|} \cdot \frac{1}{2} \cdot \left(\frac{v}{y}\right) \cdot \frac{1}{|y|}$$

And if $v/y \notin (-1, 1)$, then $f_{(V,Y)}(v, y) = 0$. Notice we are interested in the distribution of $V = ac$, so we need to solve the integral

$$f_V(v) = \int_{-1}^{1} \frac{1}{4}\frac{1}{|y|} \mathbb{1}_{\{v/y \in (-1,1)\}} dy = -\frac{1}{2} log(|v|) \mathbb{1}_{\{v \in (-1,1)\}}$$

Therefore,$f_{ac}(x) = -\frac{1}{2} log(|v|)$ if $x \in (-1, 1)$ and 0 otherwise. Finally, combining this information we can calculate:

$$P(b^2 \geq 4ac) = \int_0^1 \int_{-1}^{y/4} f_{ac}(x) f_{b^2}(y) dx dy \approx 0.6272$$

Exercise 57

Let's start by proving the equivalence:

($\Rightarrow$) Suppose $F^{-1}(u) \leq x$. Then $inf(x|F(X) \geq u) \leq x$. And since F is a cdf, it is non-decreasing, therefore $F(x) \geq u$.

($\Leftarrow$) Suppose $F(x) \geq u$. then by definition of infimum $x \geq inf(x|F(x) \geq u) = F^{-1}(u)$.

Now, this means that $F(F^{-1}(u)) = F(inf(x|F(x) \geq u)) \geq u$ by definition of the infimum. For a strict inequality, consider a non-continuous F, such that there is a discontinuity at $\hat{x}$. Assume that $F(\hat{x}) = u_2 > u_1 = F(\hat{x}-)$ (this makes sense due to right-continuity). Take u such that $u_2 > u > u_1$, and notice that $inf(x|F(x) \geq u) = \hat{x}$, but $F(\hat{x}) = u_2 > u$.

Also, $F^{-1}(F(x)) = inf(y|F(y) \geq F(x)) \leq x$ again since x holds the necessary property but it could happen that there exists a $y < x$ such that $F(x) = F(y)$. This already tells us what a strict inequality

case looks like. Consider F such that there exists an interval (a, b) in its domain such that F is constant (for the sake of making this interesting assume this region is not the trivial cases when $F(x) = 1$ or $F(x) = 0$). Then, take $x \in (a, b)$. Note that $F(x) = F(a)$ due to right continuity, $x > a$, and therefore $F^{-1}(F(x)) = inf(y|F(y) \geq F(x)) \leq a < x$.

Exercise 58

Note that:

$$\begin{aligned} F_{F(X)}(x) &= P(F(X) \leq x) \\ &= P(X \leq F^{-1}(x)) \\ &= F(F^{-1}(x)) \\ &= x \end{aligned}$$

This is exactly the cdf of a standard uniform distribution, and given that a random variable is completely characterized by its cdf, $F(X) \sim \mathcal{U}(0, 1)$

Exercise 59

Let X be a random variable such that the value x_i is attained with probability p_i for $i = 1, ..., n$, where $x_i < x_j$ if $i < j$. Then, notice that:

$$F(X) = P(X \leq x) = \begin{cases} 0 & x < x_1 \\ \sum_{i=1}^{k} p_i & x_k \leq x < x_{k+1} \text{ for } k = 1, ..., n-1 \\ 1 & x \geq x_n \end{cases}$$

Now, notice that for $P(F(X) \leq x)$ we have three cases:

Case 1: If $x < p_1$ then $P(F(X) \leq x) = 0$

Case 2: If $x \geq 1$ then $P(F(X) \leq x) = 1$

Case 3: If $x \in [p_1, 1)$, assume $x \in \left[\sum_{i=1}^{m} p_i, \sum_{i=1}^{m+1} p_i\right)$, then:

$$\begin{aligned} P(F(x) \leq x) &= P\left(F(X) \leq \sum_{i=1}^{m} p_i\right) \\ &= P\left(\bigcup_{j=1}^{m} F(X) = \sum_{i=1}^{j} p_i\right) \\ &= \sum_{j=1}^{m} P(X \in [x_j, x_{j+1})) \\ &= \sum_{j=1}^{m} P(X = x_j) \\ &= \sum_{j=1}^{m} p_j \end{aligned}$$

This is a piece-wise constant function with jumps on the points where we have a new probability p_j. We see therefore that we get a discrete approximation of a standard uniform distribution, where if the values that X takes approaches infinity, then we have exactly a standard uniform distribution.

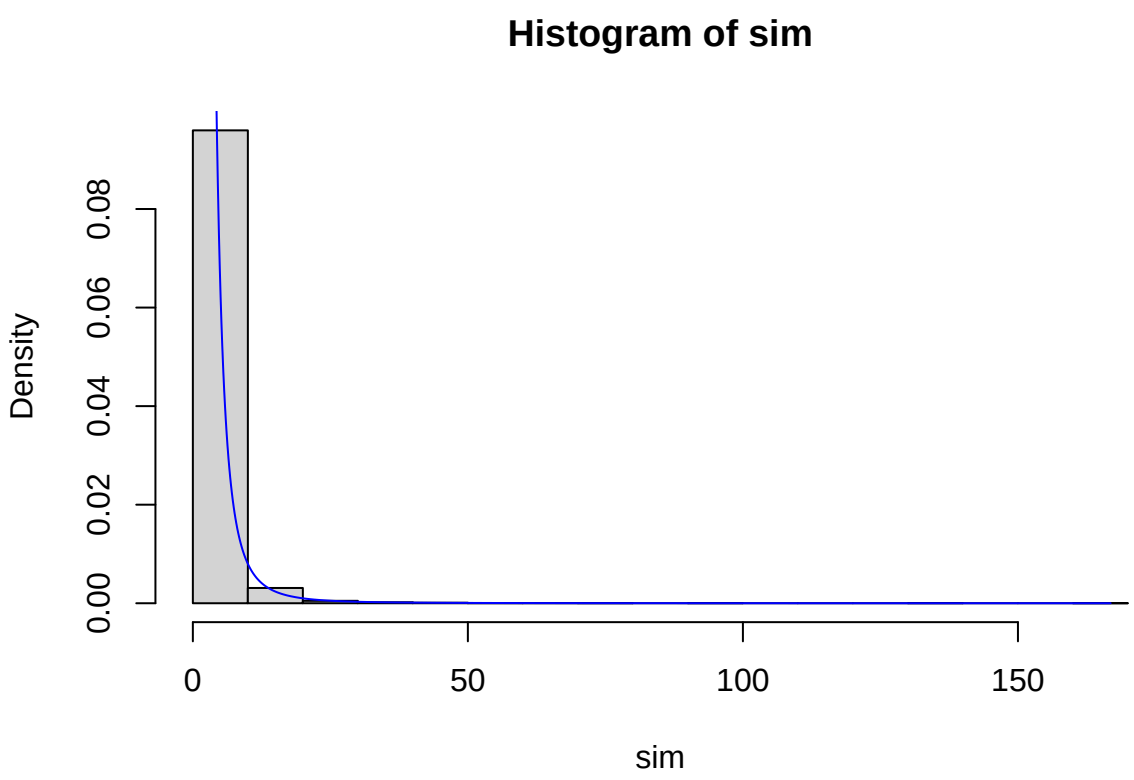
Exercise 60

Let $F(x) = u = 1 - \left(\frac{b}{x}\right)^a$. Then:

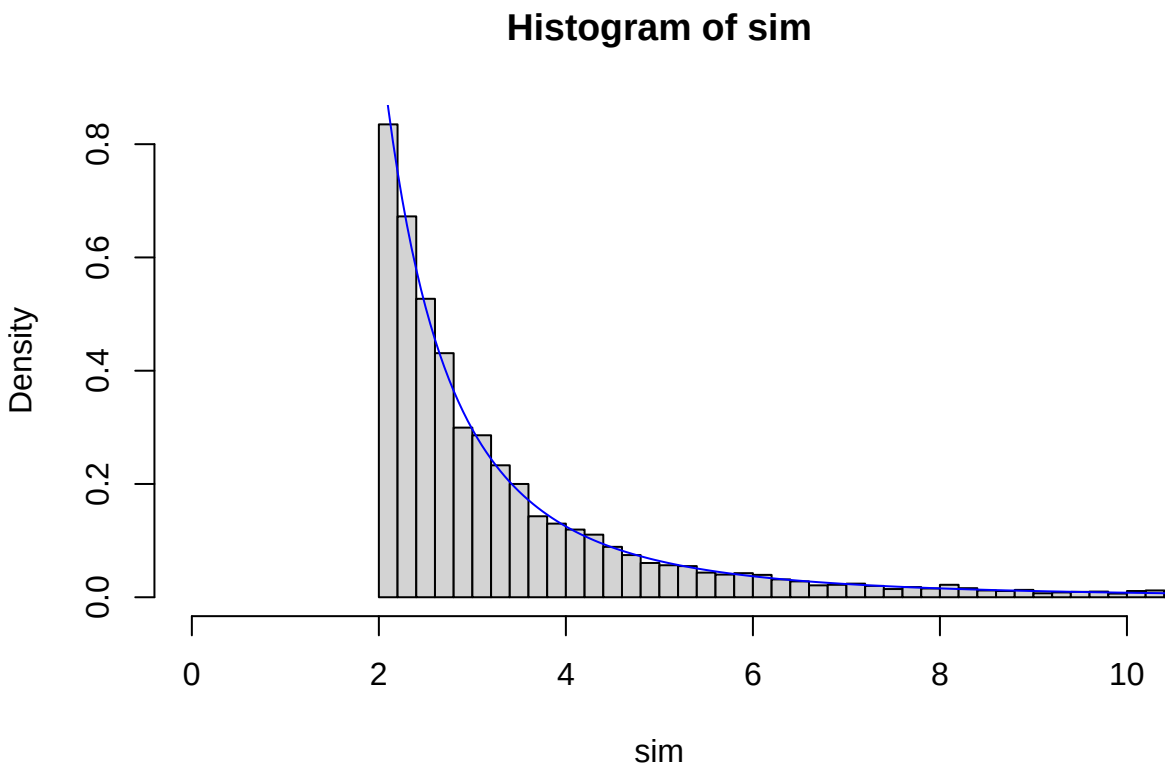
$$u = 1 - \left(\frac{b}{x}\right)^a \Leftrightarrow x = \frac{b}{(1-u)^{1/a}}$$

We use this to generate samples of a pareto distribution through inversion method:

```
rparetoInv <- function(n,a,b){
  u <- runif(n)
  b/((1-u)^(1/a))
}

paretoDensity <- function(a,b,x) a*b^a/x^(a+1)

sim <- rparetoInv(10000, 2,2)

x <- seq(0, max(sim), by = 0.05)
rpareto <- vapply(x, function(i) paretoDensity(2,2,i), 0)

hist(sim, probability = TRUE)
lines(x, rpareto, col="blue")
```




```
hist(sim, probability = TRUE, xlim = c(0,10), breaks = 1000)
lines(x, rpareto, col="blue")
```



Exercise 61

To start, recall that the definition of the exponential is given by $e^x = \lim_{n\to\infty}(1+x/n)^n$, which is equivalent to $\lim_{n\to 0}(1+xn)^{1/n}$.

Notice that when taking the limit, we have:

$$\lim_{\xi\to 0} 1-(1+\xi(x-\mu)/\sigma)^{-1/\xi} = 1-\lim_{\xi\to 0}(1+\xi(x-\mu)/\sigma)^{-1/\xi}$$

$$= 1-\left(\lim_{\xi\to 0}(1+\xi(x-\mu)/\sigma)^{1/\xi}\right)^{-1} = 1-exp((x-\mu)/\sigma)^{-1} = 1-exp(-(x-\mu)/\sigma)$$

Therefore if $\xi = 0$ we can take $F(x) = 1-exp(-(x-\mu)/\sigma)$

To check the support of F, we can use the fact that since F is a cdf then it is non-decreasing, and at its limits it has to be 0 and 1. Let's check for which values of x we have $F(x) = 0$ and $F(x) = 1$. Let's start with the lower bound.

For $\xi \neq 0$:

$$\begin{aligned} F(x) = 0 &\Leftrightarrow 1 - (1 + \xi(x-\mu)/\sigma)^{-1/\xi} = 0 \\ &\Leftrightarrow (1 + \xi(x-\mu)/\sigma)^{-1/\xi} = 1 \\ &\Leftrightarrow 1 + \xi(x-\mu)/\sigma = 1 \\ &\Leftrightarrow \xi(x-\mu)/\sigma = 1 \\ &\Leftrightarrow x = \mu \end{aligned}$$

For $\xi = 0$:

$$\begin{aligned} F(x) = 0 &\Leftrightarrow 1 - exp(-(x-\mu)/\sigma) = 0 \\ &\Leftrightarrow 1 = exp(-(x-\mu)/\sigma) \\ &\Leftrightarrow -(x-\mu)/\sigma = 0 \\ &\Leftrightarrow x - \mu = 0 \qquad\qquad \Leftrightarrow x = mu \end{aligned}$$

Therefore we see that in all cases the lower bound for the support is $x = \mu$. Let's move on now to the upper bound:

For $\xi \neq 0$:

$$\begin{aligned} F(x) = 1 &\Leftrightarrow 1 - (1 + \xi(x-\mu)/\sigma)^{-1/\xi} = 1 \\ &\Leftrightarrow (1 + \xi(x-\mu)/\sigma)^{-1/\xi} = 0 \\ &\Leftrightarrow 1 + \xi(x-\mu)/\sigma = 0 \\ &\Leftrightarrow \xi(x-\mu)/\sigma = -1 \\ &\Leftrightarrow x = \mu - \sigma/\xi \end{aligned}$$

However, notice that this only makes sense if $\xi < 0$, given that if $\xi > 0$ we would get a value of x such that $F(x) = 1$ that is smaller than μ and this is not possible due to the fact that F is a cdf, and therefore non-decreasing. In the case $\xi > 0$, ther eis no upper bound. For $\xi = 0$, notice that $1 - exp(-(x-\mu)/\sigma) = 1$ has no solution, but $\lim_{x\to\infty} 1 - exp(-(x-\mu)/\sigma) = 1$.

Therefore, we get that the support of F is $[\mu, \infty)$ if $\xi \geq 0$ and $[\mu, \mu - \sigma/\xi]$ if $\xi < 0$.

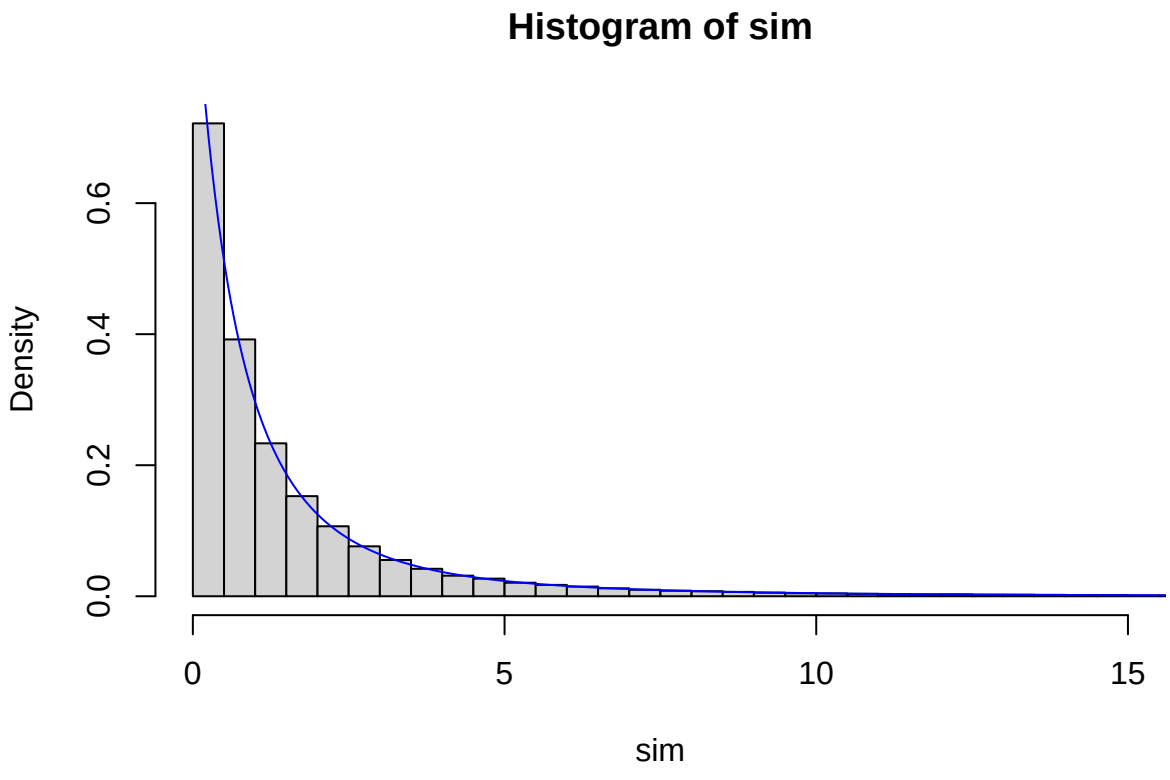
Exercise 62

Notice that we have two cases, as seen in the previous exercise. If $\xi \neq 0$ we can solve for x in $u = 1 - (1 + \xi(x-\mu)/\sigma)^{-1/\xi}$ to get that $x = ((1-u)^{-\xi} - 1)(\sigma/\xi) + \mu$. For the case $\xi = 0$, we can solve $u = 1 - exp(-(x-\mu)/\sigma)$ to find $x = -\sigma ln(1-u) + \mu$.

Recall, as seen in class, that if $U \sim \mathcal{U}(0,1)$ then $1 - U \sim \mathcal{U}(0,1)$. Therefore we can use U instead of $1 - U$ for the inversion method.

```
pareto_inv <- function(n, xi, mu, sigma){
  if(xi == 0){
    x <- mu-sigma*log(runif(n))
  }else{
    x <- (runif(n)^(-xi)-1)*(sigma/xi) + mu
  }
```

```
  if(xi >= 0){
    x[x >= mu]
  }else{
    x[ (x >= mu) & (x <= mu - sigma/xi) ]
  }
}

pareto_pdf <- function(x, xi, mu, sigma){
  if(xi != 0){
    1/sigma*(1+xi*(x-mu)/sigma)^(-(1/xi + 1))
  }else{
    1/sigma * exp((mu - x)/sigma)
  }
}

sim <- pareto_inv(1e5, xi = 0.5, mu = 0, sigma = 1)

x <- seq(0, max(sim), by = 0.05)
rpareto <- vapply(x, function(i) pareto_pdf(i,0.5,0,1), 0)

hist(sim, probability = T, xlim = c(0,15), breaks = 1000)
lines(x, rpareto, col="blue")
```



Exercise 63

The cumulative distribution function for this variable is:

$$F(x) = \begin{cases} 0, & x < 0; \\ 0.1, & 0 \leq x < 1; \\ 0.3, & 1 \leq x < 2; \\ 0.5, & 2 \leq x < 3; \\ 0.7, & 3 \leq x < 4; \\ 1, & x \geq 4; \end{cases}$$

Applying the inverse transform method, and using the sample function, we write two functions and simulate the results for a sample size 1000.

```
e63_sample <- function(n){ # Same function, but using sample()
  x <- sample(0:4, n, replace = T, prob = c(.1, .2, .2, .2, .3))
  table(x) / n # Relative freq table
}

inverse_transform <- function(n) {
  # cdf
  cumbins <- c(0.1, 0.3, 0.5, 0.7)

  singlenumber <- function() {
    x <- runif(1)
    sum(x > cumbins) # For each n_i check in which intervals the x "fits"
  }

  table(replicate(n, singlenumber())) / n # Relative freq table
}

set.seed(88)
e63_sample(10^3)
```

```
## x
##     0     1     2     3     4
## 0.097 0.204 0.182 0.216 0.301
```

```
inverse_transform(10^3)
```

```
##
##     0     1     2     3     4
## 0.109 0.204 0.188 0.219 0.280
```

Both methods give similar results, quite close to the real distribution, and could be improved by increasing the sample size.