QFin Statistics I - All Exercises

Julian Amon, Bence Kovacs, Martin Sevcik, Lin Wei

January 28, 2018

1 Exercise 1

a) The Hilbert matrix function with respect to variable n, can be constructed as below:

```
H <- function(n) {
      x <- 1 : n
      outer(x, x, function(i, j) 1 / (i + j - 1))
}
H(5)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000
## [2,] 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
## [3,] 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
## [4,] 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
## [5,] 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111
```

b) The Hilbert matrix is invertible, since its determinant is always positive, which is clear from the following formula:

$$\det H_n = \left(\prod_{k=1}^{n-1} (2k+1) \binom{2k}{k}^2 \right)^{-1} \tag{1}$$

c) Run solve() and qr.solve() function from H(1) to H(10).

```
options(width = 100)
solve(H(1))
```

```
##      [,1]
## [1,]    1
```

```
solve(H(2))
```

```
##      [,1] [,2]
## [1,]    4   -6
## [2,]   -6   12
```

```
solve(H(3))
```

```
##      [,1] [,2] [,3]
## [1,]    9  -36   30
## [2,]  -36  192 -180
## [3,]   30 -180  180
```

```
solve(H(4))
```

```
##      [,1]  [,2]  [,3]  [,4]
## [1,]   16  -120   240  -140
## [2,] -120  1200 -2700  1680
## [3,]  240 -2700  6480 -4200
## [4,] -140  1680 -4200  2800
```

```
solve(H(5))
```

```
##       [,1]   [,2]    [,3]    [,4]   [,5]
## [1,]    25   -300    1050   -1400    630
## [2,]  -300   4800  -18900   26880 -12600
## [3,]  1050 -18900   79380 -117600  56700
## [4,] -1400  26880 -117600  179200 -88200
## [5,]   630 -12600   56700  -88200  44100
```

```
solve(H(6))
```

```
##       [,1]    [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]    36    -630     3360    -7560     7560    -2772
## [2,]  -630   14700   -88200   211680  -220500    83160
## [3,]  3360  -88200   564480 -1411200  1512000  -582120
## [4,] -7560  211680 -1411200  3628800 -3969000  1552320
## [5,]  7560 -220500  1512000 -3969000  4410000 -1746360
## [6,] -2772   83160  -582120  1552320 -1746360   698544
```

```
solve(H(7))
```

```
##        [,1]     [,2]      [,3]      [,4]       [,5]       [,6]      [,7]
## [1,]     49    -1176      8820    -29400      48510     -38808     12012
## [2,]  -1176    37632   -317520   1128960   -1940400    1596672   -504504
## [3,]   8820  -317520   2857680 -10584000   18711000  -15717240   5045040
## [4,] -29400  1128960 -10584000  40320000  -72765000   62092800 -20180160
## [5,]  48510 -1940400  18711000 -72765000  133402500 -115259760  37837800
## [6,] -38808  1596672 -15717240  62092800 -115259760  100590336 -33297264
## [7,]  12012  -504504   5045040 -20180160   37837800  -33297264  11099088
```

```
solve(H(8))
```

```
##         [,1]      [,2]       [,3]       [,4]        [,5]        [,6]        [,7]       [,8]
## [1,]      64     -2016      20160     -92400      221760     -288288      192192     -51480
## [2,]   -2016     84672    -952560    4656960   -11642400    15567552   -10594584    2882880
## [3,]   20160   -952560   11430720  -58212000   149688000  -204324119   141261119  -38918880
## [4,]  -92400   4656960  -58212000  304919999  -800414996  1109908794  -776936155  216215998
## [5,]  221760 -11642400  149688000 -800414996 2134439987 -2996753738 2118916783 -594593995
## [6,] -288288  15567552 -204324119 1109908793 -2996753738  4249941661 -3030050996  856215352
## [7,]  192192 -10594584  141261119 -776936154 2118916782 -3030050996 2175421226 -618377753
## [8,]  -51480   2882880  -38918880  216215998 -594593995   856215351  -618377753  176679358
```

```
solve(H(9))
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
##  [1,]  8.099993e+01 -3.239995e+03  4.157992e+04 -2.494794e+05  8.108078e+05      -1513508
##  [2,] -3.239995e+03  1.727997e+05 -2.494794e+06  1.596668e+07 -5.405385e+07     103783367
##  [3,]  4.157992e+04 -2.494794e+06  3.841982e+07 -2.561321e+08  8.918884e+08   -1748100981
##  [4,] -2.494794e+05  1.596668e+07 -2.561321e+08  1.756334e+09 -6.243218e+09   12430939701
##  [5,]  8.108078e+05 -5.405385e+07  8.918884e+08 -6.243218e+09  2.254495e+10  -45450621475
##  [6,] -1.513508e+06  1.037834e+08 -1.748101e+09  1.243094e+10 -4.545062e+10   92553989760
##  [7,]  1.621615e+06 -1.135130e+08  1.942334e+09 -1.398481e+10  5.164843e+10 -106051443630
##  [8,] -9.266368e+05  6.589418e+07 -1.141617e+09  8.302667e+09 -3.091879e+10   63930539052
##  [9,]  2.187892e+05 -1.575283e+07  2.756745e+08 -2.021613e+09  7.581048e+09  -15768581291
##                [,7]          [,8]          [,9]
##  [1,]       1621615 -9.266369e+05  2.187892e+05
##  [2,]    -113513038  6.589418e+07 -1.575283e+07
##  [3,]    1942334186 -1.141617e+09  2.756745e+08
##  [4,]  -13984805997  8.302667e+09 -2.021613e+09
##  [5,]   51648430832 -3.091879e+10  7.581048e+09
##  [6,] -106051443959  6.393054e+10 -1.576858e+10
##  [7,]  122367050066 -7.420509e+10  1.839668e+10
##  [8,]  -74205091248  4.522977e+10 -1.126327e+10
##  [9,]   18396678800 -1.126327e+10  2.815818e+09
```

```
solve(H(10))
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
##  [1,]  9.999719e+01 -4.949757e+03  7.919482e+04 -6.005529e+05       2522295 -6.305682e+06
##  [2,] -4.949756e+03  3.266790e+05 -5.880152e+06  4.756344e+07    -208088462  5.350812e+08
##  [3,]  7.919480e+04 -5.880151e+06  1.128980e+08 -9.512635e+08    4280662450 -1.123669e+10
##  [4,] -6.005527e+05  4.756343e+07 -9.512634e+08  8.244246e+09  -37871868827  1.009913e+11
##  [5,]  2.522294e+06 -2.080884e+08  4.280662e+09 -3.787187e+10  176734991839 -4.771836e+11
##  [6,] -6.305679e+06  5.350810e+08 -1.123668e+10  1.009913e+11 -477183582308  1.301409e+12
##  [7,]  9.608580e+06 -8.323436e+08  1.775667e+10 -1.615857e+11  771204559101 -2.120813e+12
##  [8,] -8.750614e+06  7.700557e+08 -1.663323e+10  1.528915e+11 -735791094422  2.037577e+12
##  [9,]  4.375282e+06 -3.898391e+08  8.505604e+09 -7.883452e+10  382044919568 -1.064270e+12
## [10,] -9.236661e+05  8.313037e+07 -1.828875e+09  1.706955e+10  -83214282335  2.330005e+11
##                [,7]          [,8]          [,9]         [,10]
##  [1,]  9.608586e+06 -8.750620e+06  4.375286e+06 -9.236669e+05
##  [2,] -8.323439e+08  7.700561e+08 -3.898393e+08  8.313042e+07
##  [3,]  1.775667e+10 -1.663324e+10  8.505608e+09 -1.828876e+09
##  [4,] -1.615857e+11  1.528915e+11 -7.883455e+10  1.706956e+10
##  [5,]  7.712047e+11 -7.357912e+11  3.820450e+11 -8.321430e+10
##  [6,] -2.120813e+12  2.037577e+12 -1.064270e+12  2.330005e+11
##  [7,]  3.480308e+12 -3.363622e+12  1.765901e+12 -3.883348e+11
##  [8,] -3.363622e+12  3.267520e+12 -1.723107e+12  3.804102e+11
##  [9,]  1.765901e+12 -1.723107e+12  9.122340e+11 -2.020931e+11
## [10,] -3.883348e+11  3.804101e+11 -2.020931e+11  4.490964e+10
```

```
qr.solve(H(1))
```

```
##      [,1]
## [1,]    1
```

```
qr.solve(H(2))
```

```
##      [,1] [,2]
## [1,]    4   -6
## [2,]   -6   12
```

```
qr.solve(H(3))
```

```
##      [,1] [,2] [,3]
## [1,]    9  -36   30
## [2,]  -36  192 -180
## [3,]   30 -180  180
```

```
qr.solve(H(4))
```

```
##      [,1]  [,2]  [,3]  [,4]
## [1,]   16  -120   240  -140
## [2,] -120  1200 -2700  1680
## [3,]  240 -2700  6480 -4200
## [4,] -140  1680 -4200  2800
```

```
qr.solve(H(5))
```

```
##       [,1]   [,2]    [,3]    [,4]   [,5]
## [1,]    25   -300    1050   -1400    630
## [2,]  -300   4800  -18900   26880 -12600
## [3,]  1050 -18900   79380 -117600  56700
## [4,] -1400  26880 -117600  179200 -88200
## [5,]   630 -12600   56700  -88200  44100
```

```
qr.solve(H(6))
```

```
##       [,1]    [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]    36    -630     3360    -7560     7560    -2772
## [2,]  -630   14700   -88200   211680  -220500    83160
## [3,]  3360  -88200   564480 -1411200  1512000  -582120
## [4,] -7560  211680 -1411200  3628800 -3969000  1552320
## [5,]  7560 -220500  1512000 -3969000  4410000 -1746360
## [6,] -2772   83160  -582120  1552320 -1746360   698544
```

```
qr.solve(H(7))
```

```
## Error in qr.solve(H(7)):  singular matrix 'a' in solve
```

```
qr.solve(H(8))
```

```
## Error in qr.solve(H(8)):  singular matrix 'a' in solve
```

```
qr.solve(H(9))
```

```
## Error in qr.solve(H(9)):  singular matrix 'a' in solve
```

```
qr.solve(H(10))
```

```
## Error in qr.solve(H(10)):  singular matrix 'a' in solve
```

```
qr.solve(H(9), tol = 1e-17)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
##  [1,]  8.099992e+01 -3.239995e+03  4.157991e+04 -2.494793e+05  8.108075e+05      -1513507
##  [2,] -3.239995e+03  1.727996e+05 -2.494794e+06  1.596667e+07 -5.405383e+07     103783326
##  [3,]  4.157991e+04 -2.494794e+06  3.841981e+07 -2.561320e+08  8.918881e+08   -1748100312
##  [4,] -2.494793e+05  1.596667e+07 -2.561320e+08  1.756333e+09 -6.243216e+09   12430935058
##  [5,]  8.108076e+05 -5.405383e+07  8.918881e+08 -6.243216e+09  2.254494e+10  -45450604843
##  [6,] -1.513507e+06  1.037833e+08 -1.748100e+09  1.243094e+10 -4.545061e+10   92553956494
##  [7,]  1.621614e+06 -1.135130e+08  1.942333e+09 -1.398480e+10  5.164841e+10 -106051406118
##  [8,] -9.266365e+05  6.589415e+07 -1.141616e+09  8.302664e+09 -3.091878e+10   63930516758
##  [9,]  2.187891e+05 -1.575282e+07  2.756744e+08 -2.021612e+09  7.581046e+09  -15768575861
##                [,7]          [,8]          [,9]
##  [1,]       1621614 -9.266365e+05  2.187891e+05
##  [2,]    -113512992  6.589415e+07 -1.575282e+07
##  [3,]    1942333418 -1.141616e+09  2.756743e+08
##  [4,]  -13984800661  8.302664e+09 -2.021612e+09
##  [5,]   51648411712 -3.091878e+10  7.581046e+09
##  [6,] -106051405713  6.393052e+10 -1.576858e+10
##  [7,]  122367006931 -7.420507e+10  1.839667e+10
##  [8,]  -74205065609  4.522976e+10 -1.126327e+10
##  [9,]   18396672555 -1.126327e+10  2.815817e+09
```

```
qr.solve(H(10), tol = 1e-17)
```

```
##                [,1]          [,2]          [,3]          [,4]          [,5]          [,6]
##  [1,]  9.999763e+01 -4.949796e+03  7.919566e+04 -6.005606e+05       2522332 -6.305784e+06
##  [2,] -4.949794e+03  3.266822e+05 -5.880222e+06  4.756409e+07    -208091541  5.350897e+08
##  [3,]  7.919557e+04 -5.880218e+06  1.128994e+08 -9.512768e+08    4280726081 -1.123686e+10
##  [4,] -6.005595e+05  4.756403e+07 -9.512762e+08  8.244364e+09  -37872432943  1.009929e+11
##  [5,]  2.522326e+06 -2.080912e+08  4.280722e+09 -3.787241e+10  176737626224 -4.771909e+11
##  [6,] -6.305764e+06  5.350885e+08 -1.123685e+10  1.009928e+11 -477190694526  1.301428e+12
##  [7,]  9.608718e+06 -8.323557e+08  1.775693e+10 -1.615881e+11  771216047252 -2.120844e+12
##  [8,] -8.750745e+06  7.700672e+08 -1.663348e+10  1.528938e+11 -735802046037  2.037607e+12
##  [9,]  4.375350e+06 -3.898451e+08  8.505733e+09 -7.883571e+10  382050600435 -1.064285e+12
## [10,] -9.236809e+05  8.313167e+07 -1.828903e+09  1.706981e+10  -83215518397  2.330039e+11
##                [,7]          [,8]          [,9]         [,10]
##  [1,]  9.608755e+06 -8.750784e+06  4.375372e+06 -9.236858e+05
##  [2,] -8.323580e+08  7.700697e+08 -3.898465e+08  8.313200e+07
##  [3,]  1.775696e+10 -1.663352e+10  8.505756e+09 -1.828909e+09
##  [4,] -1.615883e+11  1.528940e+11 -7.883586e+10  1.706985e+10
##  [5,]  7.712167e+11 -7.358029e+11  3.820511e+11 -8.321566e+10
##  [6,] -2.120845e+12  2.037609e+12 -1.064286e+12  2.330042e+11
##  [7,]  3.480361e+12 -3.363673e+12  1.765928e+12 -3.883407e+11
##  [8,] -3.363671e+12  3.267568e+12 -1.723133e+12  3.804158e+11
##  [9,]  1.765927e+12 -1.723132e+12  9.122472e+11 -2.020961e+11
## [10,] -3.883404e+11  3.804156e+11 -2.020960e+11  4.491027e+10
```

For the solve() function, everything works fine. However, for qr.solve(), R is no longer able to solve the problem and it outputs "Error in qr.solve(): singular matrix 'a' in solve". According to the properties of Hilbert matrices, it is invertible, so the reason for this error output is that the matrix is very close to being singular. The function qr.solve() has a default tolerance setting of 10^{-7} for the QR decomposition it performs.

Therefore, manually enhancing the tolerance to 10^{-10}, qr.solve() will deliver solutions for all values greater than or equal to 7 (up to 10) as well, which are not completely identical to the solutions from solve() due to the different underlying numerical linear algebra computations performed.

2 Exercise 2

```
options(width = 75)
Quint<-function(x,p){
  y<-outer(x, 0:5,'^' )
  solve(y, p)
}
Quint(c(10, 11, 12, 13, 14, 15),c(25, 16, 26, 19, 21, 20))

## [1]  2.536100e+05 -1.025510e+05  1.650092e+04 -1.320667e+03  5.258333e+01
## [6] -8.333333e-01

a<-Quint(c(10, 11, 12, 13, 14, 15),c(25, 16, 26, 19, 21, 20))

Quint_inv<-function(x,a)
  {A<-outer(x,0:5,`^`)
  A%*%a}
Quint_inv(c(10, 11, 12, 13, 14, 15),a)

##      [,1]
## [1,]   25
## [2,]   16
## [3,]   26
## [4,]   19
## [5,]   21
## [6,]   20
```

3 Exercise 3

```
X<-matrix(runif(15),5,3)
#a)
H<-X%*%solve(t(X)%*%X)%*%t(X)
eigen(H)

## eigen() decomposition
## $values
## [1] 1.000000e+00 1.000000e+00 1.000000e+00 2.071089e-15 1.446596e-15
##
## $vectors
##             [,1]       [,2]       [,3]       [,4]        [,5]
## [1,] -0.23108293 -0.2308131  0.7924219 -0.1848135  0.48087169
## [2,]  0.01304771 -0.7720967 -0.1262309  0.6174178  0.08097861
## [3,] -0.08436258 -0.3510116 -0.5434068 -0.6076671  0.45290572
## [4,] -0.46457768 -0.3772129  0.1074671 -0.3462178 -0.71446629
## [5,] -0.85057685  0.2917077 -0.2220208  0.3090521  0.21591481

#b)
sum(diag(H)) #trace
```

```
## [1] 3

eigensum<-sum(eigen(H)$values)
sum(diag(H)) == eigensum

## [1] FALSE

sum(diag(H)) - eigensum

## [1] -3.996803e-15

#Difference due to floating-point arithmetic.

#c)
det(H)

## [1] 1.196864e-33

eigenprod<-prod(eigen(H)$values)
det(H) == eigenprod

## [1] FALSE

det(H) - eigenprod

## [1] -2.994832e-30

#Difference due to floating-point arithmetic.
```

A short proof of Part d) is the following: We know that X is a (5×3)-matrix, meaning it has 3 columns. Thus, it can be represented as $X = (x_1, x_2, x_3)$ where x_1, x_2 and x_3 are the column vectors of X. Now, let's suppose that the statement in Part d) is true: all 3 column vectors are eigenvectors of H. That means

$$Hx_i = \lambda_i x_i \quad for \quad i = \{1, 2, 3\} \quad \text{and} \quad (Hx_1, Hx_2, Hx_3) = (\lambda_1 x_1, \lambda_2 x_2, \lambda_3 x_3)$$

Due to the logic of matrix multiplication, the second equation can be rewritten as $HX = (\lambda_1 x_1, \lambda_2 x_2, \lambda_3 x_3)$ If we substitute H with its definition, we obtain $HX = (X(X^\intercal X)^{-1} X^\intercal)X = X(X^\intercal X)^{-1}(X^\intercal X) = XI = X$. Now, we can see that the matrix X behaves like an eigenvector to matrix H, since their product returns X itself. We also know from the logic of matrix multiplication, that if we apply the mapping H column-wise on X, we get the result column-wise, i.e. the products Hx_i return x_i. Therefore, all x_i are eigenvectors to H. The corresponding eigenvalues are $\lambda_i = 1$ for $i = \{1, 2, 3\}$.

4 Exercise 4

Having already defined the Hilbert function in exercise 1, we can now denote it by $H(n)$, so the (6×6) Hilbert matrix can be obtained from $H(6)$.

```
H <- function(n) {
      x <- 1 : n
      outer(x, x, function(i, j) 1 / (i + j - 1))
}
H(6)

##           [,1]      [,2]      [,3]      [,4]      [,5]       [,6]
## [1,] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.16666667
## [2,] 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.14285714
```

```
## [3,] 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571 0.12500000
## [4,] 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000 0.11111111
## [5,] 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111 0.10000000
## [6,] 0.1666667 0.1428571 0.1250000 0.1111111 0.1000000 0.09090909

#Eigenvalues and eigenvectors:
ev<-eigen(H(6))
ev

## eigen() decomposition
## $values
## [1] 1.618900e+00 2.423609e-01 1.632152e-02 6.157484e-04 1.257076e-05
## [6] 1.082799e-07
##
## $vectors
##            [,1]       [,2]       [,3]        [,4]        [,5]         [,6]
## [1,] -0.7487192  0.6145448 -0.2403254 -0.06222659  0.01114432 -0.001248194
## [2,] -0.4407175 -0.2110825  0.6976514  0.49083921 -0.17973276  0.035606643
## [3,] -0.3206969 -0.3658936  0.2313894 -0.53547692  0.60421221 -0.240679080
## [4,] -0.2543114 -0.3947068 -0.1328632 -0.41703769 -0.44357472  0.625460387
## [5,] -0.2115308 -0.3881904 -0.3627149  0.04703402 -0.44153664 -0.689807199
## [6,] -0.1814430 -0.3706959 -0.5027629  0.54068156  0.45911482  0.271605453

inv<-solve(H(6))
inv

##       [,1]    [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]    36    -630     3360    -7560     7560    -2772
## [2,]  -630   14700   -88200   211680  -220500    83160
## [3,]  3360  -88200   564480 -1411200  1512000  -582120
## [4,] -7560  211680 -1411200  3628800 -3969000  1552320
## [5,]  7560 -220500  1512000 -3969000  4410000 -1746360
## [6,] -2772   83160  -582120  1552320 -1746360   698544

eigen(inv)$values

## [1] 9.235320e+06 7.954970e+04 1.624040e+03 6.126880e+01 4.126079e+00
## [6] 6.177034e-01

sort(1/ev$values,decreasing=TRUE)

## [1] 9.235320e+06 7.954970e+04 1.624040e+03 6.126880e+01 4.126079e+00
## [6] 6.177034e-01
```

As we can see, the eigenvalues of the inverse are equal (neglecting rounding errors due to floating-point arithmetic) to the reciprocal of the eigenvalues of the original matrix. Since $H(6)$ has a basis of eigenvectors, it is diagnoalisable, and so applying the inverse to its diagonalised form, the theoretical relationship that is supposed to be applicable, is indeed that the eigenvalues of the inverse are supposed to be the reciprocals of the eigenvalues of the original matrix.

5 Exercise 5

Part (a) and (b):

```r
P<-matrix(c(0.1, 0.4, 0.3, 0.2, 0.2, 0.1, 0.4, 0.3, 0.3, 0.2, 0.1, 0.4, 0.4, 0.3, 0.2, 0.1),4,4)
rowSums(P) #returns 1 1 1 1
```

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

```r
P*P
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.01 0.04 0.09 0.16
## [2,] 0.16 0.01 0.04 0.09
## [3,] 0.09 0.16 0.01 0.04
## [4,] 0.04 0.09 0.16 0.01
```

```r
P%*%P
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.26 0.28 0.26 0.20
## [2,] 0.20 0.26 0.28 0.26
## [3,] 0.26 0.20 0.26 0.28
## [4,] 0.28 0.26 0.20 0.26
```

```r
#To avoid having to write the matrix product too many times,
#we use the package "expm".
library(expm)
```

```
## Loading required package:  Matrix
##
## Attaching package:  'expm'
## The following object is masked from 'package:Matrix':
##
##    expm
```

```r
P %^% 2
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.26 0.28 0.26 0.20
## [2,] 0.20 0.26 0.28 0.26
## [3,] 0.26 0.20 0.26 0.28
## [4,] 0.28 0.26 0.20 0.26
```

```r
P %^% 3
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,] 0.256 0.244 0.240 0.260
## [2,] 0.260 0.256 0.244 0.240
## [3,] 0.240 0.260 0.256 0.244
## [4,] 0.244 0.240 0.260 0.256
```

```r
P %^% 5
```

```
##         [,1]    [,2]    [,3]    [,4]
## [1,] 0.25056 0.25072 0.24928 0.24944
## [2,] 0.24944 0.25056 0.25072 0.24928
## [3,] 0.24928 0.24944 0.25056 0.25072
## [4,] 0.25072 0.24928 0.24944 0.25056
```

```r
P %^% 10
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.2500000 0.2500016 0.2500000 0.2499983
## [2,] 0.2499983 0.2500000 0.2500016 0.2500000
## [3,] 0.2500000 0.2499983 0.2500000 0.2500016
## [4,] 0.2500016 0.2500000 0.2499983 0.2500000
```

```
P %^% 20 #the matrix is converging to matrix(rep(0.25,16),4,4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,] 0.25 0.25 0.25 0.25
## [2,] 0.25 0.25 0.25 0.25
## [3,] 0.25 0.25 0.25 0.25
## [4,] 0.25 0.25 0.25 0.25
```

Part(c): The vector x we look for needs to be an eigenvector of P' with corresponding eigenvalue 1.

```
P_prime<-t(P)
eigen(P_prime)
```

```
## eigen() decomposition
## $values
## [1]  1.0+0.0i -0.2+0.2i -0.2-0.2i -0.2+0.0i
##
## $vectors
##         [,1]      [,2]      [,3]    [,4]
## [1,] -0.5+0i  0.0+0.5i  0.0-0.5i -0.5+0i
## [2,] -0.5+0i -0.5+0.0i -0.5-0.0i  0.5+0i
## [3,] -0.5+0i  0.0-0.5i  0.0+0.5i -0.5+0i
## [4,] -0.5+0i  0.5+0.0i  0.5+0.0i  0.5+0i
```

```
P_prime%*%rep(0.25,4) #returning a 4-element vector all of 0.25
```

```
##      [,1]
## [1,] 0.25
## [2,] 0.25
## [3,] 0.25
## [4,] 0.25
```

Indeed, P' has eigenvalue 1 with corresponding eigenvector $(-0.5, -0.5, -0.5, -0.5)'$. Since the additional condition imposed on this eigenvector is that the elements shall sum up to one, we multiply this eigenvector by -0.5, so $x = (0.25, 0.25, 0.25, 0.25)'$. As could be seen in the code, we also verified that this vector x is an eigenvector for eigenvalue 1.
The relationship between x and P^{10} is that x is the diagonal of P^{10} and it also equals the columns of the limit matrix P^n.

6 Exercise 8

It is easy to show that the specific matrix

$$\begin{bmatrix} L_1 & 0 \\ B & L_2 \end{bmatrix}$$

is also a lower triangular matrix. Its diagonal is the combination of the diagonals of L_1 and L_2, and over these values in the matrix, all elements are zeros. The partitioned linear system is then solvable in a trivial way.
More precisely, one should start the solution of

$$\begin{bmatrix} L_1 & 0 \\ B & L_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ c \end{bmatrix}$$

with the first row. It is easy to see that apart from the element in the diagonal, all the other elements in the first row of the matrix are zeros, thus the first linear equation of the system is

$$(L_1)_{11} x_1 = b_1$$

where $(L_1)_{11}$ is the element in the upper left corner of L_1, x_1 is the first element of the expanded x partition of the solution and b_1 is accordingly the first element of the expanded b partition of the RHS. Since this is a univariate equation, we get $x_1 = \frac{b_1}{(L_1)_{11}}$.
From here on, we can extract the consecutive element of the solution vector, using the consecutive rows in the matrix and the RHS, since the equations obtained by such an incremental step always includes one additional variable besides the already expressed ones.

7 Exercise 9

Part (a):
We basically need to found the assumption about the structure of the matrix, since if we already know it is a lower triangular matrix, its rows (and columns) are automatically linearly independent, thus regularity stands. The preliminary presumption is that M_k looks like an identity matrix of size $(n \times n)$ with substituting the elements under the k-th element of the diagonal with the corresponding $-\mu_i$ elements defined in the task. We expand the matrix product on the left-hand side to prove that this structure of the matrix provides exactly the desired product result.

$$M_k a = \begin{bmatrix} 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & \dots & -\mu_{k+1} & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & -\mu_n & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^n (M_k)_{1j} a_j \\ \vdots \\ \sum_{j=1}^n (M_k)_{kj} a_j \\ \sum_{j=1}^n (M_k)_{(k+1)j} a_j \\ \vdots \\ \sum_{j=1}^n (M_k)_{nj} a_j \end{bmatrix}$$

$$= \begin{bmatrix} 1a_1 \\ \vdots \\ 1a_k \\ -\mu_{k+1}a_k + a_{k+1} \\ \vdots \\ -\mu_n a_k + a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ -a_{k+1} + a_{k+1} \\ \vdots \\ -a_n + a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

The derivation above supports the original assumption of the structure of the elementary transformation matrix. Therefore, we can state that the matrix is non-singular.

Part (b):

$$I - m_k e_k' = \begin{bmatrix} 0, & \dots, & 0, & \mu_{k+1}, & \dots, & \mu_n \end{bmatrix}' \begin{bmatrix} 0, & \dots, & 1, & 0, & \dots, & 0 \end{bmatrix} = I - \begin{bmatrix} 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \dots & \mu_{k+1} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \mu_n & 0 & \dots & 0 \end{bmatrix} = M_k$$

Part (c):
Let's define $\hat{M}_k := I + m_k e_k'$

If we pre-multiply both sides of the equation $M_k a = \begin{bmatrix} a_1, & \ldots, & a_k, & 0, & \ldots, & 0 \end{bmatrix}'$ by $\hat{M}_k$ we get

$$\hat{M}_k M_k a = \begin{bmatrix} 1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 & \ldots & 0 \\ 0 & \ldots & \mu_{k+1} & 1 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & \mu_n & 0 & \ldots & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_k \\ a_{k+1} \\ \vdots \\ a_n \end{bmatrix} = a$$

$$\Rightarrow \hat{M}_k M_k a = Ia \Rightarrow \hat{M}_k M_k = I \Rightarrow \hat{M}_k = M_k^{-1}$$

Part (d):

$M_k M_l = (I - m_k e_k')(I - m_l e_l') = I - m_k e_k' - m_l e_l' + (m_k e_k')(m_l e_l')$

Regarding the last member of the sum:

$$(m_k e_k')(m_l e_l') = m_k (e_k' m_l) e_l' =$$

$$m_k \begin{bmatrix} 0, & \ldots, & 1, & 0, & \ldots, & 0, & 0, & \ldots, & 0 \end{bmatrix} \begin{bmatrix} 0, & \ldots, & 0, & 0, & \ldots, & 0, & \mu_{l+1}, & \ldots, & \mu_n \end{bmatrix}' e_l' =$$

$$m_k \, 0 \, e_l' = 0$$

8 Exercise 10

This proof is quite obvious. We use notations $L = \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix}, U = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix}, A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$,

if $A = LU$, then $a_{11} = 0 = l_{11} u_{11}$ should hold. That implied either $l_{11} = 0$ or $u_{11} = 0$ but there is no triangular matrix with a zero in the diagonal, thus the decomposition is impossible.

However, if we want to achieve the improved $PA = LU$ type of decomposition, where P is a permutation matrix, the decomposition is possible, and we can also provide it using the built-in function of R:

```
m_10<-matrix(c(0,1,1,0),2,2)
str(lu10 <- lu(m_10))

## Formal class 'denseLU' [package "Matrix"] with 4 slots
##   ..@ x       : num [1:4] 1 0 0 1
##   ..@ perm    : int [1:2] 2 2
##   ..@ Dimnames:List of 2
##   .. ..$ : NULL
##   .. ..$ : NULL
##   ..@ Dim     : int [1:2] 2 2

elu10 <- expand(lu10)
elu10 # three components: "L", "U", and "P", the permutation

## $L
## 2 x 2 Matrix of class "dtrMatrix" (unitriangular)
##      [,1] [,2]
## [1,]    1    .
## [2,]    0    1
##
## $U
## 2 x 2 Matrix of class "dtrMatrix"
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    .    1
##
## $P
## 2 x 2 sparse Matrix of class "pMatrix"
##
## [1,] . |
## [2,] | .
```

```
elu10$L %*% elu10$U
```

```
## 2 x 2 Matrix of class "dgeMatrix"
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
(m_10_2 <- with(elu10, P %*% L %*% U)) # the same as 'mm'
```

```
## 2 x 2 Matrix of class "dgeMatrix"
##      [,1] [,2]
## [1,]    0    1
## [2,]    1    0
```

```
m_10_2==m_10 #returns TRUE
```

```
## 2 x 2 Matrix of class "lgeMatrix"
##      [,1] [,2]
## [1,] TRUE TRUE
## [2,] TRUE TRUE
```

9 Exercise 11

Let A be a square $(n \times n)$-matrix. **Claim:** A has rank one $\iff \exists u, v \in \mathbb{R}^n \setminus \{0\} : A = uv'$.
" $\Rightarrow$ " Let A be an $(n \times n)$-matrix with rank one. Then the column space of A has dimension 1, i.e. it has only one linearly independent column vector. W.l.o.g. assume that the first column vector of A, denoted by $u \neq 0$, is this vector, then all other column vectors of A must be scalar multiples of this vector. Denoting the associated scalars by v_i with $i = 1, ..., n$ and also assigning $v_1 \neq 0$ as an (arbitrary) scalar to u, we can write A as $A = (v_1 u, v_2 u, ..., v_n u)$. This can matrix can thus also be expressed as the dyadic/outer product of vectors u and v, where $v = (v_1, ..., v_n)$, i.e.

$$A = uv'.$$

" $\Leftarrow$ " Let $u, v \in \mathbb{R}^n \setminus \{0\}$. Then the dyadic product uv' is

$$(u_1, ..., u_n)'(v_1, ..., v_n) = (v_1 u, ...v_n u) =: A.$$

Since the columns of the suchlike defined $(n \times n)$-matrix A are just scalar multiples of the same vector u, A has only one linearly independent column, so the dimension of the column space of A is 1. Consequently,

$$rank(A) = 1 \quad \text{q.e.d.}$$

10 Exercise 12

Part (a): Let A be an elementary matrix, i.e. $A = I - uv'$ for some non-zero n-vectors u and v. By the Sherman-Morrison formula (see exercise 13), it holds that

$$A^{-1} = (I - uv')^{-1} = I^{-1} + I^{-1}u(1 - v'I^{-1}u)^{-1}v'I^{-1} = I + u(1 - v'u)^{-1}v' = I + \frac{uv'}{1 - v'u}$$

Hence, we see that the inverse of A exists (i.e. A is non-singular), iff $v'u \neq 1$.

Part (b): From part (a), we got that

$$A^{-1} = (I - uv')^{-1} = I + \frac{uv'}{1 - v'u}$$

Let $\sigma \in \mathbb{R}$ and choose

$$\sigma := -\frac{1}{1 - v'u}$$

Thus, we get that indeed A^{-1} is an elementary matrix, since it can be written in the form of

$$A^{-1} = I - \sigma uv'$$

Part (c): By the results from exercise 9, we got that an elementary elimination matrix M_k allows us to annihilate all of the entries below the kth position of an n-vector, provided that the kth element of this vector is unequal to zero. As was shown in 9b), such a matrix M_k can be written as $M_k = I - m_k e_k'$, which shows that such a matrix is indeed elementary of the form $A = I - uv'$, where $u = m_k = (0, ..., 0, \mu_{k+1}, ..., \mu_n)'$ (for the definition of the μ_i, please see exercise 9) and v is the kth Cartesian unit vector e_k.

Finally, for σ, we get

$$\sigma = -\frac{1}{1 - v'u} = -\frac{1}{1 - e_k'm_k} = -1 \text{ since } e_k'm_k = 0$$

.

11 Exercise 13

The Sherman-Morrison formula is given by

$$(A - uv')^{-1} = A^{-1} + A^{-1}u(1 - v'A^{-1}u)^{-1}v'A^{-1}.$$

We pre-multiply both sides by $(A - uv')$, so the proof of this formula comes down to proving that the right-hand side of the equation pre-multiplied by $(A - uv')$ is equal to the identity. We get

$$(A - uv')(A^{-1} + A^{-1}u(1 - v'A^{-1}u)^{-1}v'A^{-1}) = AA^{-1} - uv'A^{-1} + \frac{(AA^{-1}uv'A^{-1}) - (uv'A^{-1}uv'A^{-1})}{1 - v'A^{-1}u} =$$

$$= I - uv'A^{-1} + \frac{uv'A^{-1} - uv'A^{-1}uv'A^{-1}}{1 - v'A^{-1}u} = I - uv'A^{-1} + \frac{u(1 - v'A^{-1}u)v'A^{-1}}{1 - v'A^{-1}u} =$$

$$= I - uv'A^{-1} + uv'A^{-1} = I \quad \text{q.e.d.}$$

12 Exercise 14

The Woodbury formula is given by

$$(A - UV')^{-1} = A^{-1} + A^{-1}U(I - V'A^{-1}U)^{-1}V'A^{-1}.$$

We pre-multiply both sides by $(A - UV')$, so the proof of this formula comes down to proving that the right-hand side of the equation pre-multiplied by $(A - UV')$ is equal to the identity. We get

$$(A - UV')(A^{-1} + A^{-1}U(I - V'A^{-1}U)^{-1}V'A^{-1}) =$$
$$= I - UV'A^{-1} + U(I - V'A^{-1}U)^{-1}V'A^{-1} - UV'A^{-1}U(I - V'A^{-1}U)^{-1}V'A^{-1} =$$
$$= I - U(I - (I - V'A^{-1}U)^{-1} + V'A^{-1}U(I - V'A^{-1}U)^{-1})V'A^{-1} =$$
$$= I - U(I - V'A^{-1}U + I + V'A^{-1}U)(I - V'A^{-1}U)^{-1}V'A^{-1} = I \quad \text{q.e.d.}$$

13 Exercise 22

Suppose the $(n \times n)$-matrix A has the block upper triangular form

$$A = \begin{bmatrix} A_{11} & A_{12} \\ O & A_{22} \end{bmatrix}$$

where A_{11} is $(k \times k)$ and A_{22} is $(n-k) \times (n-k)$.

Part (a): Let λ be an eigenvalue of A_{11} and let u be the corresponding eigenvector. Then, by the definition of eigenvalues and eigenvectors, it holds that $A_{11}u = \lambda u$. Similarly, for λ to be an eigenvalue of A, it must hold that $Aw = \lambda w$, where w is the corresponding eigenvector of A. Based on the hint in the exercise, we choose $w = (u', v')'$ and then try to determine, whether there exists a $(n-k)$-vector v in such a way that λ will be an eigenvalue of A with corresponding eigenvector $w = (u', v')'$. Writing the equation for the eigenvalue/-vector of A in matrix form, we get

$$\begin{bmatrix} A_{11} & A_{12} \\ O & A_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_k \\ v_1 \\ \vdots \\ v_{n-k} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ \vdots \\ u_k \\ v_1 \\ \vdots \\ v_{n-k} \end{bmatrix}$$

Using the fact that λ is an eigenvalue of A_{11}, we can also write this equation as $A(u', v')' = ((\lambda u + A_{12}v)', (A_{22}v)')' \stackrel{!}{=} \lambda(u', v')'$. If there exists a vector v, for which this equation holds, then λ will be an eigenvalue of A. From this stacked vector equation, we get the following two equations that must hold for v:

$$\lambda u + A_{12}v = \lambda u \quad \text{and} \quad A_{22}v = \lambda v \Rightarrow v = 0$$

Since the equation will therefore always hold for $v = 0$, v is the $(n-k)$-dimensional zero vector and λ will be an eigenvalue of A with the corresponding eigenvector $w = (u', v')' = (u', (0, 0, ..., 0)')'$.

Part (b): Let λ be an eigenvalue of A_{22} (but not of A_{11}) and let q denote the corresponding eigenvector of A_{22}. Using a similar ansatz as in part (a), we want to verify if λ is also an eigenvalue of A with corresponding eigenvector $x = (p', q')'$, where p is some k-vector. Thus, for λ to be an eigenvalue of A, it must hold that

$$Ax = A(p', q')' = ((A_{11}p + A_{12}q)', (A_{22}v_2)')' = ((A_{11}p + A_{12}q)', (\lambda v_2)')' \stackrel{!}{=} \lambda(p', q')'.$$

Since the lower part of this stacked vector equation holds automatically (as λ is an eigenvalue of A_{22}), the upper part delivers the decisive equation for p:

$$(A_{11}p + A_{12}q) = \lambda p \iff (A_{11} - I\lambda)p = -A_{12}q \iff p = (A_{11} - I\lambda)^{-1}(-A_{12}q).$$

Choosing vector p in this way, we see that λ will indeed be an eigenvalue of A with corresponding eigenvector $x = (p', q')'$ (Note that the fact that λ is an eigenvalue only of A_{22}, but not of A_{11}, ensures that $(A_{11} - I\lambda)$ will be invertible).

Part (c): Let λ be an eigenvalue of A with corresponding eigenvector $(u', v')'$ where u is a k-vector and v is an $(n-k)$-vector. Then, it holds that

$$A(u', v')' = ((A_{11}u + A_{12}v)', (A_{22}v)')' = \lambda(u', v')' = (\lambda u', \lambda v')'.$$

If this equation holds, we could either have (i) $A_{11}u = \lambda u$ and $v = 0$ or (ii) $A_{22}v = \lambda v$ and $u = (A_{11} - I\lambda)^{-1}(-A_{12}v)$ for non-zero matrices A_{11}, A_{12}, A_{22}. Thus, we see that either λ is an eigenvalue of A_{11} with corresponding eigenvector u or λ is an eigenvalue of A_{22} with corresponding eigenvector v. Note that λ cannot be an eigenvalue of both A_{11} and A_{22}, since then $(A_{11} - I\lambda)^{-1}$ would not exist.

Part (d): Claim: $Ax = \lambda x \iff (A_{11}u = \lambda u) \veebar (A_{22}v = \lambda v)$.
" $\Rightarrow$ " Shown in part (c): if λ is an eigenvalue of A, it will either be an eigenvalue of A_{11} or A_{22}.

" $\Leftarrow$ " Part (a) showed that if λ is an eigenvalue of A_{11}, then it will also be an eigenvalue of A. Part (b) showed that if λ is an eigenvalue of A_{22} (but not of A_{11}), then it will also be an eigenvalue of A. Since the statement says that λ should be an eigenvalue of EITHER A_{11} OR A_{22} (but not both, i.e. exclusive OR), we have therefore proved the above claim.

14 Exercise 24

Proof. Let $u, v \in \mathbb{R}^n$ and I be the $n \times n$ identity matrix then we use following equality:

$$\begin{bmatrix} I & 0 \\ v^\intercal & 1 \end{bmatrix} \begin{bmatrix} I + uv^\intercal & u \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ -v^\intercal & 1 \end{bmatrix} = \begin{bmatrix} I & u \\ 0 & 1 + v^\intercal u \end{bmatrix}$$

Taking determinant of both sides and using Laplace formula we get:

$$\det I \times \det(I + uv^\intercal) \times \det I = \det(1 + u^\intercal v) \tag{2a}$$
$$\det(I + uv^\intercal) = 1 + u^\intercal v \tag{2b}$$

□

15 Exercise 25

The eigenvalues of householder transformation $H = I - 2 \times \frac{vv^\intercal}{v^\intercal v}$ are ± 1. To see this we provide a proof.

Proof. Let $H = I - 2 \times \frac{vv^\intercal}{v^\intercal v}$ be a householder transformation, then

$$HH^\intercal = (I - 2\frac{vv^\intercal}{v^\intercal v})(I - 2\frac{vv^\intercal}{v^\intercal v}) = I - 4\frac{vv^\intercal}{v^\intercal v} + 4\frac{vv^\intercal vv^\intercal}{(v^\intercal v)^2} = I - 4\frac{vv^\intercal}{v^\intercal v} + 4\frac{vv^\intercal}{v^\intercal v} = I \tag{3}$$

Thus H is orthogonal. Let x be an eigenvector of H, then

$$|\lambda|^2 x^\intercal x = x^\intercal \lambda^\intercal \lambda x = (Hx)^\intercal Hx = x^\intercal H^\intercal Hx = x^\intercal x \tag{4}$$

Therefore λ must be ± 1. □

The household transformation presents a linear transformation, which provides reflection about a plane or a hyperplane containing the origin.

16 Exercise 27

To show that $(-1)^n p(z)$ is the characteristic polynomial of

$$p(z) = \gamma_0 + \gamma_1 z + \cdots + \gamma_{n-1} z^{n-1} + z^n$$

we use the induction:

Proof. Let $C(n)$ be the companion matrix of polynomial of order n, then the in initial step we have for $n = 1$:

$$\det(C(1) - Iz) = \det\left[-\gamma_0 - Iz\right] = -\gamma_0 - z = -1^n p(z)$$

Thus the statement holds for $n = 1$. In next step, we form the inductive hypothesis, which is $\det(C(n) - Iz) = (-1)^n p_n(z)$ and prove the statement for $n + 1$. That is

$$\det(C(n+1) - Iz) = \det \begin{bmatrix} -z & 0 & \dots & 0 & -\gamma_0 \\ 1 & -z & \dots & 0 & -\gamma_1 \\ 0 & 1 & \dots & 0 & -\gamma_2 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & -\gamma_n - z \end{bmatrix} =$$

$$= (-1)^{2n+1} \times 1 \times \det \begin{bmatrix} -z & 0 & 0 & \dots & -\gamma_0 \\ 1 & -z & 0 & \dots & -\gamma_1 \\ 0 & 1 & -z & \dots & \gamma_2 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & 1 & -\gamma_{n-1} \end{bmatrix} + (-1)^{2n+2}(-\gamma_n - z) \times \det \begin{bmatrix} -z & 0 & 0 & \dots & 0 \\ 1 & -z & 0 & \dots & 0 \\ 0 & 1 & -z & \dots & 0 \\ \vdots & & \ddots & & 0 \\ 0 & 0 & \dots & 1 & -z \end{bmatrix} =$$

$$= -1 \times \begin{bmatrix} -z & 0 & 0 & \dots & -\gamma_0 \\ 1 & -z & 0 & \dots & -\gamma_1 \\ 0 & 1 & -z & \dots & \gamma_2 \\ \vdots & & \ddots & & \vdots \\ 0 & \dots & 0 & 1 & -\gamma_{n-1} - z \end{bmatrix} + (-z)^n + (-\gamma_n - z) \times (-z)^n =$$

$$= -1 \times (-1)^n p_n(z) + (-1)^n \times z^n + (-1)^n \times (-\gamma_n) z^n + (-1)^{n+1} z^{n+1} =$$

$$= (-1)^{n+1} \Big(p_n(z) - z^n + \gamma_n \times z^n + z^{n+1} \Big) = (-1)^{n+1} p_{n+1}(z)$$

Note that the $(-z)^n$ on the 3rd line arises from adding and subtracting the same value to and from the last component of the shrunk matrix, thus linearly distributing the coefficient of the corresponding subdeterminant. By this alternation, on the one hand, we attain the determinant included in the inductive hypothesis that we wanted to use for the proof, on the other hand, we get a lower triangular subdeterminant. □

In next part of the exercise, we compute the roots of polynomial $p(z) = 24 - 40z + 35z^2 - 13z^3 + z^4$ using the companion matrix:

```
#First we form the companion matrix
A<-cbind(rbind(rep(0,3),diag(rep(1,3))),c(-24,40,-35,+13))
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    0    0    0  -24
## [2,]    1    0    0   40
## [3,]    0    1    0  -35
## [4,]    0    0    1   13
```

```r
#Next we compute the eigenvalues of companion matrix,
#which correspond to roots of the polynomial
eigen(A)$values
```

```
## [1] 9.8274224+0.0000000i 1.8047699+0.0000000i 0.6839038+0.9409769i
## [4] 0.6839038-0.9409769i
```

17 Exercise 29

```r
vec<-function(A) c(A)

#check
A1<-matrix(1:16,4,4)
A1
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

```r
vec(A1)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```r
A2<-matrix(1:16,4,4,byrow=TRUE)
A2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
```

```r
vec(A2)
```

```
##  [1]  1  5  9 13  2  6 10 14  3  7 11 15  4  8 12 16
```

18 Exercise 30

```r
vech<-function(A) c(A[lower.tri(A,diag = TRUE)])
#check
A1<-matrix(c(1,2,3,2,8,1,3,1,6),3,3)
A1
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    8    1
## [3,]    3    1    6
```

```r
vech(A1)
```

```
## [1] 1 2 3 8 1 6
```

19 Exercise 15

Part (a): Here we have

$$B = \begin{bmatrix} \alpha & a' \\ a & A \end{bmatrix} \tag{5}$$

Proof. Since the matrix B is positive definite, all leading principle minors of B should have a positive determinant. This immediately yields that α - being the first leading principle minor - must be positive.

Also, we have $\forall x \in \mathbb{R}^n \backslash \{0\} : x'Bx > 0$, since B is positive definite. Since this holds **for all** vectors x, we specifically choose

$$x = \begin{bmatrix} 0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{6}$$

We denote the sub-vector c in x by

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} \tag{7}$$

The quadratic form $x'Bx = c'Ac > 0 \quad \forall c \in \mathbb{R}^n \backslash \{0\}$. Hence, A is also positive definite. □

Part (b): The Cholesky-factorization of the symmetric block matrix leads to an iterative problem. One has to apply the so-called LDL' factorization first, where L is a lower-triangular block-matrix and D is a diagonal block-matrix. Then, one can figure out the components of the block-Cholesky factors.
So first, given the algorithm of LDL', we rewrite B as

$$B = \begin{bmatrix} \alpha & a' \\ a & A \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\alpha} & 1 \end{bmatrix} \begin{bmatrix} \alpha & 0 \\ 0 & A - \frac{1}{\alpha} aa' \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{\alpha} a' \\ 0 & 1 \end{bmatrix}$$

Note, that the 0 element at position $[1, 2]$ in the first matrix is a row-vector, the 1 element at position $[2, 2]$ in the first matrix is an identity matrix of $(n \times n)$. In the second matrix, the 0 at $[2, 1]$ is a column-vector, the 0 at $[1, 2]$ is a row-vector. The matrix $A - \frac{1}{\alpha} aa'$ is symmetric, since it is the difference of two symmetric matrices, thus it is Cholesky-decomposable. In the third matrix, the 0 element is a column-vector and the 1 element at $[2, 2]$ is again an identity matrix of size $(n \times n)$.
We denote the Cholesky-factors of α as $A = L_A L'_A$, here $L_A = L'_A = \sqrt{\alpha}$. We also denote the Cholesky-factors of $A - \frac{1}{\alpha} aa'$ as L_S and L'_S. Then, we can write the decomposition of B as follows:

$$B = \begin{bmatrix} \sqrt{\alpha} & 0 \\ \frac{1}{\sqrt{\alpha}} a & L_S \end{bmatrix} \begin{bmatrix} \sqrt{\alpha} & \frac{1}{\sqrt{\alpha}} a' \\ 0 & L'_S \end{bmatrix}$$

Thus, the Cholesky-decomposition of B can be traced back to the Cholesky-decomposition of the matrix $A - \frac{1}{\alpha} aa'$.

20 Exercise 16

Part (a): Here we have

$$B = \begin{bmatrix} A & a \\ a' & \alpha \end{bmatrix} \tag{8}$$

Proof. Since the matrix B is positive definite, all leading principle minors of B should have a positive determinant. A is a symmetric sub-matrix in the top left corner of B, hence all leading principle minors of A should have a positive determinant, which immediately yields that A must be positive definite.

Also, we have $\forall x \in \mathbb{R}^n \backslash \{0\} : x'Bx > 0$, since B is positive definite. Since this holds **for all** vectors x, we specifically choose

$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \tag{9}$$

The quadratic form of B then delivers that α must be postive, since $x'Bx = \alpha > 0$. □

Part (b): We use the same logic of Cholesky block-factorization as in part b of exercise 15.
First, the LDL' decomposition of our current B block matrix is

$$B = \begin{bmatrix} A & a \\ a' & \alpha \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ a'A^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & \alpha - a'A^{-1}a \end{bmatrix} \begin{bmatrix} 1 & A^{-1}a \\ 0 & 1 \end{bmatrix}$$

Here, that the 1 element at position $[1,1]$ in the first matrix is an identity matrix of $(n \times n)$, the 0 element at position $[1,2]$ in the first matrix is a column-vector. In the second matrix, the 0 at $[2,1]$ is a row-vector, the 0 at $[1,2]$ is a column-vector. $\alpha - a'A^{-1}a$ is a scalar. In the third matrix, the 1 element at $[1,1]$ is again an identity matrix of size $(n \times n)$ and the 0 element is a row-vector.
We denote the Cholesky-factors of A as $A = L_A L_A'$ and the Cholesky-factors of $\alpha - a'A^{-1}a$ as L_S and L_S'. Note that $L_S = L_S' = \sqrt{\alpha - a'A^{-1}a}$. Then, we can write the decomposition of B as follows:

$$B = \begin{bmatrix} L_A & 0 \\ a'L_A'^{-1} & \sqrt{\alpha - a'A^{-1}a} \end{bmatrix} \begin{bmatrix} L_A' & L_A^{-1}a \\ 0 & \sqrt{\alpha - a'A^{-1}a} \end{bmatrix}$$

Here, the Cholesky-decomposition of B can be traced back to the Cholesky-decomposition of the matrix A.

21 Exercise 17

```
v<-list()
for (k in 1:10){

Eps<-10^(-2*k)

A<- matrix(
  c(Eps,1,1,1),
  nrow=2,
  ncol=2)

B<- matrix(
  c(1+Eps,2),
  nrow=2,
  ncol=1)

M<- matrix(
  c(1,-1/Eps,0,1),
  nrow=2,
  ncol=2)

v[[k]]<-backsolve( M%*%A , M %*% B)
}
#List the vector of outcome regarding k from 1 to 10.
v
```

```
## [[1]]
##      [,1]
## [1,]    1
## [2,]    1
##
## [[2]]
##      [,1]
## [1,]    1
## [2,]    1
##
## [[3]]
##      [,1]
## [1,]    1
## [2,]    1
##
## [[4]]
##      [,1]
## [1,]    1
## [2,]    1
##
## [[5]]
##      [,1]
## [1,]    1
## [2,]    1
##
## [[6]]
##           [,1]
## [1,] 0.9998669
## [2,] 1.0000000
##
## [[7]]
##           [,1]
## [1,] 0.9992007
## [2,] 1.0000000
##
## [[8]]
##          [,1]
## [1,] 2.220446
## [2,] 1.000000
##
## [[9]]
##      [,1]
## [1,]    0
## [2,]    1
##
## [[10]]
##      [,1]
## [1,]    0
## [2,]    1
```

As we can see the error appears when $\epsilon \leq 2^{-16}$, i.e., for $k = 8, 9, 10$. (it already starts to appear at $k = 6$, albeit at a lower degree of significance). This problem is resulting from the rounding-error in floating point arithmetic, which eventually leads to a stable, yet completely erroneous solution of $x(\epsilon) = (0, 1)'$ for $k \geq 9$.

22 Exercise 18

First, we have to recognize that Ax and x are vectors and that the corresponding $\|.\|_2$ of a vector simply equals its Euclidean norm. Therefore, we can rewrite the 2-norm of A as:

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} \tag{10}$$

Let $B := A'A$, and rewrite $\|Ax\| = \sqrt{\langle Ax, Ax \rangle} = \sqrt{x'A'Ax} = \sqrt{x'Bx}$. B is obviously a symmetric matrix and it is possible to prove that all symmetric matrices have a set of eigenvectors that form an orthogonal basis (of the dimension of the matrix). If the dimension of B is n, let the normalized eigenvectors of B be $(e_1, e_2, ..., e_n)$.
Then, since x is an $(n \times 1)$ column vector, we can decompose

$$x = a_1 e_1 + ... + a_n e_n$$

where (a_i) are real numbers, the corresponding coordinates of x in the standard basis (e_i). We also know that $Be_i = \lambda_i e_i$ where λ_i is the corresponding eigenvalue of B. It is also possible to prove that all λ values are real and different. Then, we have

$$Bx = B(\sum_{i=1}^{n} a_i e_i) = \sum_{i=1}^{n} a_i B(e_i) = \sum_{i=1}^{n} \lambda_i a_i e_i \tag{11}$$

Let λ_j be such an eigenvalue of B that has the largest absolute value. Then,

$$\|Ax\| = \sqrt{\langle x, Bx \rangle} = \sqrt{\langle \sum_{i=1}^{n} a_i e_i, \sum_{i=1}^{n} \lambda_i a_i e_i \rangle} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} a_i e_i \lambda_j a_j e_j} = \sqrt{\sum_{i=1}^{n} a_i^2 \lambda_i}$$

$$\leq \sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{|\ \lambda_j\ |} = \|x\| \sqrt{|\ \lambda_j\ |}$$

$$\Rightarrow \frac{\|Ax\|}{\|x\|} \leq \max_j \sqrt{|\ \lambda_j\ |} \Rightarrow \|A\|_2 \leq \max_j \sqrt{|\ \lambda_j\ |}$$

which is an upper bound for the 2-norm of A.
Also, since the definition of the 2-norm holds for all x that are non-zero, it should also hold for such an x that is equal to $x_k = e_k$ which is a normalized eigenvector of B, having a length of 1 and belonging with the corresponding eigenvalue λ_k. The maximum value in the definition yields that

$$\|A\|_2 \geq \frac{\|Ax_k\|}{\|x_k\|} = \frac{\|Ae_k\|}{\|e_k\|}$$

Again, we can do the following rewriting of the numerator:

$$\|Ae_k\| = \sqrt{\langle e_k, Be_k \rangle} = \sqrt{\langle e_k, \lambda_k e_k \rangle} = \sqrt{\lambda_k} \quad \text{for any } \lambda_k, \text{ even } \max_j |\ \lambda_j\ |!$$

$$\Rightarrow \|A\|_2 \geq \max_j \sqrt{|\ \lambda_j\ |} \Rightarrow \|A\|_2 = \max_j \sqrt{|\ \lambda_j\ |} = \max_j \sqrt{|\ \sigma_j^2\ |} = \max_j \sigma_j, \quad \text{i.e. the largest singular value.}$$

The last conversion between λ_j and σ_j^2 comes from the fact that the squares of the singular values of A are the eigenvalues of $A'A$ and AA'. We also know that all singular values are non-negative. Therefore, the 2-norm of a matrix equals its largest singular value.

23 Exercise 19

$$\kappa(A) = \|A\|_2 \left\|A^{-1}\right\|_2$$

We already know that $\|A\|_2$ equals the largest singular value of A. Therefore, we have to look for the largest singular value of A^{-1}.
We derive the singular value decomposition of A^{-1}:

$$\text{Let } A = UDV', \text{then } A^{-1} = (V')^{-1}D^{-1}U^{-1} = VD^{-1}U' \quad \text{since } V \text{ and } U \text{ are orthogonal matrices.}$$

Since D^{-1} is diagonal, this is the singular value decomposition of A^{-1}, and its singular values are σ_i^{-1} where σ_i is the i-th singular value of A.
Therefore, the largest singular value of A^{-1} is the reciprocal of the smallest positive singular value of A.
$\kappa(A)$ then follows as:

$$\kappa(A) = \frac{\max_j \sigma_j}{\min_i \sigma_i}$$

where σ_j, σ_i are singular values of A and $\sigma_i \neq 0$.
The implementation is the following:

```
kappa<-function(A) {
  A_svd<-svd(A)
  A_norm2<-max(A_svd$d)
  A_inv_norm2<-min(A_svd$d[which(A_svd$d>0)])
  return(A_norm2/A_inv_norm2)
}

kappa(matrix(1:9,3,3))
```

```
## [1] 3.039469e+16
```

```
#returns 3.039469e+16
matrix(1:9,3,3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
svd(matrix(1:9,3,3)) #SVD-s are 1.684810e+01, 5.543107e-16
```

```
## $d
## [1] 1.684810e+01 1.068370e+00 5.543107e-16
##
## $u
##            [,1]        [,2]       [,3]
## [1,] -0.4796712  0.77669099  0.4082483
## [2,] -0.5723678  0.07568647 -0.8164966
## [3,] -0.6650644 -0.62531805  0.4082483
##
## $v
##            [,1]       [,2]       [,3]
## [1,] -0.2148372 -0.8872307  0.4082483
## [2,] -0.5205874 -0.2496440 -0.8164966
## [3,] -0.8263375  0.3879428  0.4082483
```

```
1.684810e+01/5.543107e-16 #returns the same value
```

```
## [1] 3.039469e+16
```

24 Exercise 20

$$\kappa(A) = \|A\|_2 \left\|A^{-1}\right\|_2$$

$$\kappa(A)\frac{\|\Delta b\|}{\|b\|} = \|A\|_2 \left\|A^{-1}\right\|_2 \frac{\|\Delta b\|}{\|Ax\|} \geq \|A\|_2 \left\|A^{-1}\right\|_2 \frac{\|\Delta b\|}{\|A\|_2 \|x\|} = \frac{\left\|A^{-1}\right\|_2 \|\Delta b\|}{\|x\|} \geq \frac{\|\Delta x\|}{\|x\|}$$

25 Exercise 21

When deciding on the small values ϵ should take we have to take the computer's precision into account. In the double-precision floating point system, the smallest significand of normalized numbers has the dimension of 2^{-52}. Since we also have ϵ^2 in our formulas, we should only let ϵ reduce until 2^{-26}, so as not to lose significant nonzero values due to underflow.

The $\kappa(A)$ function has the following formula: $\kappa = \sqrt{\frac{2+\epsilon^2+2\sqrt{1+\epsilon^2}}{2+\epsilon^2-2\sqrt{1+\epsilon^2}}}$. We can use our previously developped shortcut κ function though, which avoids underflow down to smaller ϵ values than the direct formula.

We take a 13-element length sequence as epsilons. We create the corresponding linear system matrices, right-hand sides and also the respective κ functions.

```
epsilon<-2^-seq(1,25,2)

matrices<-list()
b<-list()
for (j in 1:length(epsilon)) {
  matrices[[j]]<-matrix(c(1,1-epsilon[j],1+epsilon[j],1),2,2)
  b[[j]]<-c(1+epsilon[j]+epsilon[j]^2,1)
  }

matrices[[13]]==matrix(rep(1,4),2,2)
```

```
##       [,1]  [,2]
## [1,]  TRUE FALSE
## [2,] FALSE  TRUE
```

```
#returns FALSE in the off-diagonal, there is no underflow
b[[13]][1]==1
```

```
## [1] FALSE
```

```
#returns FALSE, there is no underflow

Kap<-numeric()
for(i in 1:length(epsilon)) {
  Kap[i]<-kappa(matrices[[i]])
}
```

25.1 Matrix inversion

We apply the first solution method which is simply inverting the matrix of the linear system. In our case (we will show this) this case is equivalent to calling the solve()-function of R, even if we give in the RHS as argument into it.

```
#simple solve(matrix) method
matrices_inv<-list()
sols1<-list()
```

```
for(i in 1:length(epsilon)) {
  matrices_inv[[i]]<-tryCatch({solve(matrices[[i]])}, error=function(err){
    print(paste("MY_ERROR:  ",i))
    return(matrix(rep(0,4),2,2))
  })
  max(abs(matrices_inv[[i]]%*%matrices[[i]]-diag(2)))
  sols1[[i]]<-c(matrices_inv[[i]]%*%b[[i]])
}

## [1] "MY_ERROR:   13"

#we eventually catch an error message since R is unable to invert the last matrix,
#which is numerically singular.

sols1[[10]][2]==0

## [1] TRUE

#returns TRUE, so R cannot solve the system with this method beyond epsilon<=2^-21
v<-numeric()
for (i in 1:length(epsilon)) {
  v<-c(v,sols1[[i]][2]-epsilon[i])
}
v #returns non-zero values from the 10th element

##  [1]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##  [6]  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00 -1.907349e-06
## [11] -4.768372e-07 -1.192093e-07 -2.980232e-08

#solve(system)
sols2<-list()
for(i in 1:length(epsilon)) {
  sols2[[i]]<-tryCatch({solve(matrices[[i]],b[[i]])}, error=function(err){
    print(paste("MY_ERROR:  ",i))
    return(matrix(rep(0,2)))
})
}

## [1] "MY_ERROR:   13"

v2<-numeric()
for (i in 1:length(epsilon)) {
  v2<-c(v2,sols2[[i]][2]-epsilon[i])
}

v==v2 #no difference here

##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

25.2 QR-decompositions

```
sols3<-list()
for(i in 1:length(epsilon)) {
  sols3[[i]]<-tryCatch({qr.solve(matrices[[i]],b[[i]])}, error=function(err){
```

```
    print(paste("MY_ERROR:  ",i))
    return(matrix(rep(0,2)))
  })
}
```

```
## [1] "MY_ERROR:   7"
## [1] "MY_ERROR:   8"
## [1] "MY_ERROR:   9"
## [1] "MY_ERROR:   10"
## [1] "MY_ERROR:   11"
## [1] "MY_ERROR:   12"
## [1] "MY_ERROR:   13"
```

```
#derived from the error messages, this R function already
#finds matrices numerically singular with epsilon<=2^-12.

v3<-numeric()
for (i in 1:length(epsilon)) {
  v3<-c(v3,sols3[[i]][2]-epsilon[i])
}

abs(v3)-abs(v2)
```

```
##  [1] 2.220446e-16 7.077672e-15 1.978556e-13 5.124821e-12 5.613688e-11
##  [6] 1.156994e-09 1.220703e-04 3.051758e-05 7.629395e-06 0.000000e+00
## [11] 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
```

```
#returns a positive sequence up to the point
#the 3rd method is not carried out anymore.
#We can almost surely say that method 3 is subordinate
#to method 2. Method 2 is on the one hand able to deal
#with more precise numbers, on the other hand, its
#solutions deviate from the expected values less.

#We can however apply the pure QR decomposition and see
#whether it helps us reach different solutions by computation:
matrices_qr<-list()
Q<-list()
R<-list()
errors<- numeric()
sols4<-list()
for(i in 1:length(epsilon)) {
  matrices_qr[[i]]<-qr(matrices[[i]])
  #the decompositions is always possible
  Q[[i]]<-qr.Q(matrices_qr[[i]])
  R[[i]]<-qr.R(matrices_qr[[i]])
  errors[i]<-max(abs(Q[[i]]%*%R[[i]] - matrices[[i]]))
  #this sequence tells us about very precise decomps.
  sols4[[i]]<-c(backsolve(R[[i]],crossprod(Q[[i]],b[[i]])))
}

v4<-numeric()
for (i in 1:length(epsilon)) {
  v4<-c(v4,sols4[[i]][2]-epsilon[i])
}
```

```
#The obtained x values look promising with relatively
#larger epsilons, but start to diverge from the real
#solutions as epsilon shrinks. These deviations become
#even larger than the ones with the first method (possibly LU),
#on the other hand, this solving method is applicable
#in all the cases. However, concerning these solutions,
#even the first element of the solution deviates from 1 as epsilon shrinks.
```

25.3 Singular Value Decomposition Method

```
#Singular Value Decomposition Method
cmult <- function(A, v) A * rep(v, each = nrow(A))
matrices_svd<-list()
sols5<-list()
for(i in 1:length(epsilon)) {
  matrices_svd[[i]]<-svd(matrices[[i]])
  sols5[[i]]<-cmult(matrices_svd[[i]]$v,
                    1 / matrices_svd[[i]]$d)  %*% crossprod(matrices_svd[[i]]$u, b[[i]])
}

v5<-numeric()
for (i in 1:length(epsilon)) {
  v5<-c(v5,sols5[[i]][2]-epsilon[i])
}
#In terms of the difference from the real solution,
#the SVD method provides similar results to the
#QR-decomposition method. As epsilon shrinks, the
#computed solution deviates astonishingly from the
#real solution, in both of its elements.
```

25.4 Conditional number probes

```
#To check the accuracy, we observe the condition number relationships

trivialsol<-list()
for(i in 1:length(epsilon)) {
  trivialsol[[i]]<-c(1,epsilon[i])
}
#this is the list of trivial solutions, for every epsilon

sol_diff_1<-mapply('-',sols1,trivialsol,SIMPLIFY=FALSE)
sol_diff_2<-mapply('-',sols2,trivialsol,SIMPLIFY=FALSE)
sol_diff_3<-mapply('-',sols3,trivialsol,SIMPLIFY=FALSE)
sol_diff_4<-mapply('-',sols4,trivialsol,SIMPLIFY=FALSE)
sol_diff_5<-mapply('-',sols5,trivialsol,SIMPLIFY=FALSE)
#here, solsn refers to the list of solutions obtained by using method n.
#The operation simply takes the difference between the obtained and the real solutions.

sol_reldiff_1<-sapply(sol_diff_1,function(x) sqrt(sum(x^2)))/sapply(trivialsol,function(x) sqrt(sum(x^2)))
sol_reldiff_2<-sapply(sol_diff_2,function(x) sqrt(sum(x^2)))/sapply(trivialsol,function(x) sqrt(sum(x^2)))
sol_reldiff_3<-sapply(sol_diff_3,function(x) sqrt(sum(x^2)))/sapply(trivialsol,function(x) sqrt(sum(x^2)))
sol_reldiff_4<-sapply(sol_diff_4,function(x) sqrt(sum(x^2)))/sapply(trivialsol,function(x) sqrt(sum(x^2)))
sol_reldiff_5<-sapply(sol_diff_5,function(x) sqrt(sum(x^2)))/sapply(trivialsol,function(x) sqrt(sum(x^2)))
#by these operations, we take the norm relative error of the solutions.
```

```r
b1<-mapply('%*%',matrices,sols1,SIMPLIFY = FALSE)
b2<-mapply('%*%',matrices,sols2,SIMPLIFY = FALSE)
b3<-mapply('%*%',matrices,sols3,SIMPLIFY = FALSE)
b4<-mapply('%*%',matrices,sols4,SIMPLIFY = FALSE)
b5<-mapply('%*%',matrices,sols5,SIMPLIFY = FALSE)
#the different b(i)-s are the different RHS vectors obtained by post-multiplying
#the original matrix by the obtained solution, for method (i) and for every epsilon.


rhs_diff_1<-mapply('-',b1,b,SIMPLIFY=FALSE)
rhs_diff_2<-mapply('-',b2,b,SIMPLIFY=FALSE)
rhs_diff_3<-mapply('-',b3,b,SIMPLIFY=FALSE)
rhs_diff_4<-mapply('-',b4,b,SIMPLIFY=FALSE)
rhs_diff_5<-mapply('-',b5,b,SIMPLIFY=FALSE)
#this operation helped us compute the absolute errors in the RHS-s

rhs_reldiff_1<-sapply(rhs_diff_1,function(x) sqrt(sum(x^2)))/sapply(b,function(x) sqrt(sum(x^2)))
rhs_reldiff_2<-sapply(rhs_diff_2,function(x) sqrt(sum(x^2)))/sapply(b,function(x) sqrt(sum(x^2)))
rhs_reldiff_3<-sapply(rhs_diff_3,function(x) sqrt(sum(x^2)))/sapply(b,function(x) sqrt(sum(x^2)))
rhs_reldiff_4<-sapply(rhs_diff_4,function(x) sqrt(sum(x^2)))/sapply(b,function(x) sqrt(sum(x^2)))
rhs_reldiff_5<-sapply(rhs_diff_5,function(x) sqrt(sum(x^2)))/sapply(b,function(x) sqrt(sum(x^2)))
#and again, we obtained the normed relative differences in RHS vectors.

Kap*rhs_reldiff_1>=sol_reldiff_1

##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

Kap*rhs_reldiff_2>=sol_reldiff_2

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
## [12] FALSE  TRUE

Kap*rhs_reldiff_3>=sol_reldiff_3

##  [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE

Kap*rhs_reldiff_4>=sol_reldiff_4

##  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE FALSE

Kap*rhs_reldiff_5>=sol_reldiff_5

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
## [12]  TRUE  TRUE

#If one runs these operations, the logical vectors one gets are almost everywhere TRUE.
#If we investigate the FALSE values, those are the cases where the RHS difference was
#eventually zero and there was some minor relative difference in the solutions.
#These examples can be however due to computational error.
#Here, one eventually discovers the difference between methods 1 and 2,
#since in case of method 2, some of such incidents come across, when the condition
#number theorem does not hold.
```

26 Exercise 23

Part (a):

$$Au = uv'u = u\langle v, u\rangle = u\langle u, v\rangle = u'vu$$

The equations above prove that $u'v$ is an eigenvalue of A, namely the one corresponding to the eigenspace on vector u.

Part (b): Take two eigenvectors x_1 and x_2 of A. Then, the followings hold:

$$Ax_1 = uv'x_1 = \lambda_1 x_1 = u\langle v, x_1\rangle \Leftrightarrow x_1 = \frac{\langle v, x_1\rangle}{\lambda_1}u$$

$$Ax_2 = uv'x_2 = \lambda_2 x_2 = u\langle v, x_2\rangle \Leftrightarrow x_2 = \frac{\langle v, x_2\rangle}{\lambda_2}u$$

We showed that all x eigenvectors are in the span of u unless the corresponding λ eigenvalue is 0 and x is orthogonal to v. (Then, we cannot carry out the division on the right side of the equivalence, but the left side still holds). This entails that the matrix A can only have eigenvectors with eigenvalue zero or eigenvectors with eigenvalue $u'v$ (i.e. eigenvectors in the span of u).

Part (c): The singular value decomposition (of the square matrix A) follows as: $A = UDV'$ where U and V' are orthogonal matrices, while D is diagonal. We say:

$$AA' = VDU'UDV = UD^2U' = uv'vu' = \langle v, v\rangle uu'$$

This matrix has eigenvectors that are the columns of uu':

$$\langle v, v\rangle uu' = \langle v, v\rangle \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \begin{bmatrix} u_1, & u_2, & \dots & u_n \end{bmatrix} = \langle v, v\rangle \begin{bmatrix} u_1u_1 & u_1u_2 & \dots & u_1u_n \\ u_2u_1 & u_2u_2 & \dots & u_2u_n \\ \vdots & \vdots & \ddots & \vdots \\ u_nu_1 & u_nu_2 & \dots & u_nu_n \end{bmatrix} = \begin{bmatrix} \langle v, v\rangle u_1u & \langle v, v\rangle u_2u & \dots & \langle v, v\rangle u_nu \end{bmatrix}$$

The block form of the matrix AA' shows that all its eigenvectors (the left singular vectors of A) are some scaled variants of u. Therefore, the matrix U will only have columns in the span of u, normalized.
The corresponding eigenvalues are $\langle v, v\rangle\langle u, u\rangle$, since the following holds:

$$\langle v, v\rangle(uu')u = \langle v, v\rangle u(u'u) = \langle v, v\rangle u\langle u, u\rangle$$

Without extending explanation, we state that the eigenvectors (the right singular vectors) of $A'A$ are in the span of v, with corresponding eigenvalues $\langle v, v\rangle\langle u, u\rangle$.
Finally, the singular value decomposition of A looks like the following:

$$A = \begin{bmatrix} \frac{u}{\|u\|} & \frac{u}{\|u\|} & \cdots & \frac{u}{\|u\|} \end{bmatrix} \begin{bmatrix} \|u\|\,\|v\| & & & 0 \\ & \|u\|\,\|v\| & & \\ & & \ddots & \\ 0 & & & \|u\|\,\|v\| \end{bmatrix} \begin{bmatrix} \frac{v'}{\|v\|} \\ \frac{v'}{\|v\|} \\ \vdots \\ \frac{v'}{\|v\|} \end{bmatrix}$$

Part (d): We provide a case-by-case solution to this section.
If the starting vector itself (let it be called x_{k-1}) is already a nonzero scalar multiple of u (which spans the corresponding eigenspace of the eigenvalue $u'v$), then we do not need any iterations to turn x_k into the desired eigenvector.
If after the first iteration, $Ax_{k-1} = uv'x_{k-1} = u\langle v, x_{k-1}\rangle$, the inner product $\langle v, x_{k-1}\rangle = 0$, i.e. v and x_{k-1} are orthogonal, then we can do the iteration infinitely many times, we would never get an eigenvector in the span of u, since the nullvector is not an eigenvector.
In any other case, only 1 iteration is enough, since we get $Ax_{k-1} = uv'x_{k-1} = u\langle v, x_{k-1}\rangle$ which is in the span of u.

27 Exercise 26

```r
options(width = 100)
gen_matrix<-function(n){
  output<-diag(n)
  output[upper.tri(output)]<--1
  output
}

svd(gen_matrix(4))
```

```
## $d
## [1] 2.2630774 1.5961547 1.5157216 0.1826443
##
## $u
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.7721586  0.5436987  0.2894381 -0.1561678
## [2,] -0.5247923 -0.4489780 -0.6837511 -0.2355762
## [3,] -0.1093053 -0.6219176  0.6420633 -0.4347707
## [4,]  0.3411985  0.3406303 -0.1909573 -0.8550379
##
## $v
##            [,1]       [,2]       [,3]       [,4]
## [1,] -0.3411985  0.3406303  0.1909573 -0.8550379
## [2,]  0.1093053 -0.6219176 -0.6420633 -0.4347707
## [3,]  0.5247923 -0.4489780  0.6837511 -0.2355762
## [4,]  0.7721586  0.5436987 -0.2894381 -0.1561678
```

```r
svd(gen_matrix(5))
```

```
## $d
## [1] 2.73632965 1.68194375 1.54805314 1.50945370 0.09298533
##
## $u
##             [,1]       [,2]        [,3]       [,4]       [,5]
## [1,] -0.69955243  0.5534327  0.39432272 -0.2059737 0.08014388
## [2,] -0.57207134 -0.1142105 -0.60026497  0.5337402 0.12038981
## [3,] -0.34034036 -0.6440739 -0.08082337 -0.6433686 0.22109200
## [4,] -0.04658835 -0.3969474  0.64247645  0.4900540 0.43282017
## [5,]  0.25565356  0.3290435 -0.25472170 -0.1364558 0.86189807
##
## $v
##             [,1]       [,2]        [,3]       [,4]       [,5]
## [1,] -0.25565356  0.3290435  0.25472170 -0.1364558 0.86189807
## [2,]  0.04658835 -0.3969474 -0.64247645  0.4900540 0.43282017
## [3,]  0.34034036 -0.6440739  0.08082337 -0.6433686 0.22109200
## [4,]  0.57207134 -0.1142105  0.60026497  0.5337402 0.12038981
## [5,]  0.69955243  0.5534327 -0.39432272 -0.2059737 0.08014388
```

```r
svd(gen_matrix(6))
```

```
## $d
## [1] 3.26606133 1.78656452 1.59057670 1.52965453 1.50634219 0.04676117
##
## $u
##             [,1]        [,2]       [,3]       [,4]       [,5]        [,6]
## [1,]  0.63736651 -0.53901099  0.4310585  0.3023944 -0.1560209 -0.04043257
## [2,]  0.56382838 -0.08674953 -0.3823603 -0.5874118  0.4238385 -0.06067098
## [3,]  0.42523680  0.43829982 -0.4742550  0.2512619 -0.5715188 -0.11127808
## [4,]  0.23758222  0.59559021  0.3287821  0.3505885  0.5572017 -0.21758512
## [5,]  0.02251585  0.25314589  0.5113986 -0.5817040 -0.3849454 -0.43280379
## [6,] -0.19514836 -0.30170250 -0.2710077  0.1976880  0.1035760 -0.86466122
##
## $v
##             [,1]        [,2]       [,3]       [,4]       [,5]        [,6]
## [1,]  0.19514836 -0.30170250  0.2710077  0.1976880 -0.1035760 -0.86466122
## [2,] -0.02251585  0.25314589 -0.5113986 -0.5817040  0.3849454 -0.43280379
## [3,] -0.23758222  0.59559021 -0.3287821  0.3505885 -0.5572017 -0.21758512
## [4,] -0.42523680  0.43829982  0.4742550  0.2512619  0.5715188 -0.11127808
## [5,] -0.56382838 -0.08674953  0.3823603 -0.5874118 -0.4238385 -0.06067098
## [6,] -0.63736651 -0.53901099 -0.4310585  0.3023944  0.1560209 -0.04043257
```

```r
svd(gen_matrix(10))
```

```
## $d
##  [1] 5.620481273 2.346585337 1.837194685 1.664578049 1.586555465 1.545597754 1.522399936 1.509102751
##  [9] 1.502162174 0.002929643
##
## $u
##              [,1]        [,2]        [,3]        [,4]       [,5]        [,6]        [,7]
##  [1,]  0.48034187 -0.45316213  0.41748850  0.37552437  0.3265013 -0.27057488 -0.20860085
```

```
##  [2,]  0.46410851 -0.32392587  0.11392327 -0.11165076 -0.3048360  0.42521857  0.44799286
##  [3,]  0.43219042 -0.10230994 -0.27247812 -0.45397917 -0.3467289  0.02754652 -0.30551935
##  [4,]  0.38566627  0.14848352 -0.46075457 -0.20735151  0.2818285 -0.44096244 -0.09737640
##  [5,]  0.32610836  0.35693132 -0.31400606  0.30827733  0.3654299  0.22448005  0.41726929
##  [6,]  0.25552949  0.46358669  0.06106328  0.42397195 -0.2575801  0.31266368 -0.38148519
##  [7,]  0.17631489  0.43803285  0.39173215 -0.01036056 -0.3825219 -0.40317906  0.02052618
##  [8,]  0.09114165  0.28755744  0.43756380 -0.43125211  0.2321975 -0.08223172  0.35792918
##  [9,]  0.00288825  0.05507418  0.16523302 -0.29267185  0.3979295  0.45017758 -0.43128858
## [10,] -0.08546276 -0.19311556 -0.22724238  0.22559734 -0.2057925 -0.17506164  0.13702106
##              [,8]        [,9]       [,10]
##  [1,] -0.14183311  0.07175886 0.002537115
##  [2,]  0.36947782 -0.20801238 0.003805678
##  [3,] -0.45118549  0.32320865 0.006977088
##  [4,]  0.35468308 -0.40568627 0.013637056
##  [5,] -0.11808729  0.44709598 0.027115582
##  [6,] -0.16515084 -0.44324583 0.054151958
##  [7,]  0.38315738  0.39452558 0.108264428
##  [8,] -0.44982371 -0.30586722 0.216509345
##  [9,]  0.33881783  0.18624569 0.433009398
## [10,] -0.09398506 -0.04777038 0.866015080
##
## $v
##              [,1]        [,2]        [,3]        [,4]       [,5]        [,6]        [,7]
##  [1,]  0.08546276 -0.19311556  0.22724238  0.22559734  0.2057925 -0.17506164 -0.13702106
##  [2,] -0.00288825  0.05507418 -0.16523302 -0.29267185 -0.3979295  0.45017758  0.43128858
##  [3,] -0.09114165  0.28755744 -0.43756380 -0.43125211 -0.2321975 -0.08223172 -0.35792918
##  [4,] -0.17631489  0.43803285 -0.39173215 -0.01036056  0.3825219 -0.40317906 -0.02052618
##  [5,] -0.25552949  0.46358669 -0.06106328  0.42397195  0.2575801  0.31266368  0.38148519
##  [6,] -0.32610836  0.35693132  0.31400606  0.30827733 -0.3654299  0.22448005 -0.41726929
##  [7,] -0.38566627  0.14848352  0.46075457 -0.20735151 -0.2818285 -0.44096244  0.09737640
##  [8,] -0.43219042 -0.10230994  0.27247812 -0.45397917  0.3467289  0.02754652  0.30551935
##  [9,] -0.46410851 -0.32392587 -0.11392327 -0.11165076  0.3048360  0.42521857 -0.44799286
## [10,] -0.48034187 -0.45316213 -0.41748850  0.37552437 -0.3265013 -0.27057488  0.20860085
##              [,8]        [,9]       [,10]
##  [1,] -0.09398506  0.04777038 0.866015080
##  [2,]  0.33881783 -0.18624569 0.433009398
##  [3,] -0.44982371  0.30586722 0.216509345
##  [4,]  0.38315738 -0.39452558 0.108264428
##  [5,] -0.16515084  0.44324583 0.054151958
##  [6,] -0.11808729 -0.44709598 0.027115582
##  [7,]  0.35468308  0.40568627 0.013637056
##  [8,] -0.45118549 -0.32320865 0.006977088
##  [9,]  0.36947782  0.20801238 0.003805678
## [10,] -0.14183311 -0.07175886 0.002537115

sv_ratio<-numeric(48)

for(i in 3:50){
  sv_ratio[i-2]<-max(svd(gen_matrix(i))$d)/min(svd(gen_matrix(i))$d)
}
sv_ratio

##  [1] 5.411474e+00 1.239063e+01 2.942754e+01 6.984559e+01 1.635252e+02 3.768062e+02 8.556269e+02
##  [8] 1.918487e+03 4.255786e+03 9.355367e+03 2.040740e+04 4.422189e+04 9.527923e+04 2.042630e+05
## [15] 4.359889e+05 9.269916e+05 1.964156e+06 4.148899e+06 8.739380e+06 1.836262e+07 3.849413e+07
## [22] 8.052813e+07 1.681395e+08 3.504519e+08 7.292603e+08 1.515253e+09 3.144006e+09 6.515074e+09
## [29] 1.348438e+10 2.787742e+10 5.757253e+10 1.187811e+11 2.448354e+11 5.042194e+11 1.037539e+12
## [36] 2.133291e+12 4.383008e+12 8.998890e+12 1.846376e+13 3.785890e+13 7.758058e+13 1.588997e+14
## [43] 3.252075e+14 6.657458e+14 1.283237e+15 1.283435e+15 1.281959e+15 1.284028e+15

multiples<-numeric(44)
for(i in 2:45){
  multiples[i-1]<-sv_ratio[i]/sv_ratio[i-1]
}
multiples

##  [1] 2.289695 2.374984 2.373477 2.341238 2.304271 2.270734 2.242200 2.218303 2.198270 2.181358
## [11] 2.166953 2.154572 2.143835 2.134449 2.126182 2.118850 2.112306 2.106434 2.101135 2.096332
## [21] 2.091958 2.087960 2.084292 2.080914 2.077795 2.074905 2.072220 2.069720 2.067386 2.065203
## [31] 2.063155 2.061232 2.059422 2.057713 2.056107 2.054576 2.053131 2.051781 2.050444 2.049203
## [41] 2.048190 2.046621 2.047142 1.927518
```

The ratio certainly seems to be increasing exponentially with an increasing factor of around 2, i.e. at each step, for a matrix of order n, the ratio is around twice as high as it was for the matrix of order $n-1$.

28 Exercise 28

Note that SVD delivers orthogonal matrices U and V, i.e. their columns constitute an orthonormal system. For the general $(m \times n)$ matrix A, the reduced form SVD will deliver a matrix U, for which all the columns that are associated with non-zero singular values, span the column space of A (proof was done in class). Since they are orthonormal vectors by construction, we therefore obtain an orthonormal basis for the range of A by extracting precisely these columns of U associated with non-zero singular values. If there are zero singular values, then this indicates linear dependency among the columns of A, so we do not extract all $min(m, n)$ left singular vectors of A as the basis, but only one less for each zero singular value.

```
orthorange<-function(A,tol=.Machine$double.eps){
  decomp<-svd(A)
  crit<-decomp$d[1]*max(nrow(A),ncol(A))*tol
  num_zero<-decomp$d>=crit
  cbind(decomp$u[,num_zero])
}
A<-matrix(1:15,5,3)
A[,2]<-2*A[,1] #creating an artificial linear dependence.
orthorange(A)

##            [,1]        [,2]
## [1,] -0.3219620  0.70451435
## [2,] -0.3806970  0.39378906
## [3,] -0.4394319  0.08306377
## [4,] -0.4981669 -0.22766152
## [5,] -0.5569018 -0.53838680

B<-matrix(runif(15),3,5)
orthorange(B)

##            [,1]       [,2]       [,3]
## [1,] -0.6147556  0.7233094 -0.3144823
## [2,] -0.5270432 -0.6733644 -0.5184649
## [3,] -0.5867717 -0.1529834  0.7951698
```

As regards the null space (or kernel) of A, we extract the right singular vectors, i.e. the columns of matrix V, for which we have a zero singular value. In this case, one additionally has to be aware of the case where $n > m$ because in this case, the reduced form of the SVD excludes orthonormal vectors, which do span the kernel of A, however. The underlying mathematical reason is the "loss" of dimensions due to the fact that the linear mapping represented by A maps from $\mathbb{R}^n$ into a lower-dimensional $\mathbb{R}^m$ in this case. We do have to obtain these additional right singular vectors, which in R, is done by specifying the number of singular vectors to be computed, which we do in the below function. To obtain an orthonormal basis of the kernel of A, we therefore extract the right singular vectors of A, for which the corresponding singular value is zero, and the additional ones resulting from the "loss" of dimensions in $\mathbb{R}^n \to \mathbb{R}^m$ if $n > m$.

```
orthokernel<-function(A,tol=.Machine$double.eps){
  m<-nrow(A)
  n<-ncol(A)
  decomp<-svd(A,m,n)
  crit<-decomp$d[1]*max(m,n)*tol
  selection<-if(n>m) c(decomp$d<crit,rep(TRUE,n-m)) else decomp$d<crit
  cbind(decomp$v[,selection])
  #function returns empty matrix in case of trivial kernel
}
orthokernel(A)
```

```
##               [,1]
## [1,] -8.944272e-01
## [2,]  4.472136e-01
## [3,] -5.551115e-17

orthokernel(B)

##            [,1]         [,2]
## [1,] -0.3680122 -0.379625061
## [2,]  0.5098996  0.353360070
## [3,] -0.5222956  0.001378727
## [4,]  0.5609881 -0.391970775
## [5,] -0.1306484  0.759854252
```

29 Exercise 31

```
vec<-function(A) matrix(c(A),ncol=1)
vech<-function(A) matrix(c(A[lower.tri(A,diag = TRUE)]),ncol=1)

dupli_n<-function(n){
        initial<-diag(n) #create an identity of dimension (n x n)
        #the duplication matrix will be of dimension (n^2 x (n^2+n)/2),
        #so we create the following sequence:
        ind<-seq(n*(n+1)/2)
        initial[lower.tri(initial,diag=TRUE)]<-ind
        initial[upper.tri(initial)]<-t(initial)[upper.tri(initial)]
        outer(c(initial),ind,function(n,x) ifelse(n==x,1,0))
}
A<-matrix(c(1,2,3,2,-1,0,3,0,4),3,3)
dupli_n(3)

##       [,1] [,2] [,3] [,4] [,5] [,6]
##  [1,]    1    0    0    0    0    0
##  [2,]    0    1    0    0    0    0
##  [3,]    0    0    1    0    0    0
##  [4,]    0    1    0    0    0    0
##  [5,]    0    0    0    1    0    0
##  [6,]    0    0    0    0    1    0
##  [7,]    0    0    1    0    0    0
##  [8,]    0    0    0    0    1    0
##  [9,]    0    0    0    0    0    1

dupli_n(3)%*%vech(A)==vec(A)

##       [,1]
##  [1,] TRUE
##  [2,] TRUE
##  [3,] TRUE
##  [4,] TRUE
##  [5,] TRUE
##  [6,] TRUE
##  [7,] TRUE
```

```
##  [8,] TRUE
##  [9,] TRUE
```

30 Exercise 32

Evidently, D_n has $(n^2+n)/2$ singular values. Note that $D_n'D_n$ is an $(m \times m)$ diagonal matrix, where $m = (n^2+n)/2$. The diagonal elements of this matrix are 1, then a sequence of $(n-1)$ 2's, then another 1, then a sequence of $(n-2)$ 2's, and so forth, where the last diagonal element is also a 1. The eigenvalues of this multiplied matrix will therefore be only 1's and 2's, from which we can conclude that n of the m singular values of D_n will be 1 and $(n^2-n)/2$ of them will be $\sqrt{2}$. This can also be verified computationally:

```
svd(dupli_n(4))$d

##  [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.000000 1.000000 1.000000 1.000000

svd(dupli_n(5))$d

##  [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [11] 1.000000 1.000000 1.000000 1.000000 1.000000

svd(dupli_n(6))$d

##  [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [11] 1.414214 1.414214 1.414214 1.414214 1.414214 1.000000 1.000000 1.000000 1.000000 1.000000
## [21] 1.000000

svd(dupli_n(7))$d

##  [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [11] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [21] 1.414214 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000

svd(dupli_n(8))$d

##  [1] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [11] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214
## [21] 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.414214 1.000000 1.000000
## [31] 1.000000 1.000000 1.000000 1.000000 1.000000 1.000000
```

31 Exercise 33

```
vec<-function(A) matrix(c(A),ncol=1)
vech<-function(A) matrix(c(A[lower.tri(A,diag = TRUE)]),ncol=1)

elim_n<-function(n){
  #Note that an elimination matrix will be ((n^2+n)/2 x n^2)
  result<-diag((n^2+n)/2)
  sequen<-(1:(n-1))*n
  j<-1
  #inserting all-zero rows into the identity based on the underlying
  #logic of elimination matrices:
  for(i in sequen){
    result<-cbind(result[,1:i],matrix(0,(n^2+n)/2,j),result[,(i+1):ncol(result)])
    j<-j+1
  }
```

```
  result
}

A<-matrix(c(1,2,3,2,-1,0,3,0,4),3,3)
elim_n(3)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
## [1,]    1    0    0    0    0    0    0    0    0
## [2,]    0    1    0    0    0    0    0    0    0
## [3,]    0    0    1    0    0    0    0    0    0
## [4,]    0    0    0    0    1    0    0    0    0
## [5,]    0    0    0    0    0    1    0    0    0
## [6,]    0    0    0    0    0    0    0    0    1
```

```
elim_n(3)%*%vec(A)==vech(A)
```

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

32 Exercise 34

Note that $L_n L_n' = I$, i.e. the elimination matrix L_n post-multiplied by its own transpose is equal to the $(m \times m)$ identity matrix, where $m = (n^2 + n)/2$. The identity obviously has eigenvalues all ones, so consequently, L_n has $(n^2 + n)/2$ singular values equal to 1. In reduced form, the matrix U in the SVD will then be equal to the identity matrix, while the matrix V will be the transpose of the elimination matrix L_n itself. To verify this computationally, consider the following:

```
svd(elim_n(4))
```

```
## $d
##  [1] 1 1 1 1 1 1 1 1 1 1
##
## $u
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0
##  [6,]    0    0    0    0    0    1    0    0    0     0
##  [7,]    0    0    0    0    0    0    1    0    0     0
##  [8,]    0    0    0    0    0    0    0    1    0     0
##  [9,]    0    0    0    0    0    0    0    0    1     0
## [10,]    0    0    0    0    0    0    0    0    0     1
##
## $v
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    1    0    0    0    0    0    0    0    0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0
##  [5,]    0    0    0    0    0    0    0    0    0     0
##  [6,]    0    0    0    0    1    0    0    0    0     0
##  [7,]    0    0    0    0    0    1    0    0    0     0
```

```
##  [8,]    0    0    0    0    0    0    1    0    0     0
##  [9,]    0    0    0    0    0    0    0    0    0     0
## [10,]    0    0    0    0    0    0    0    0    0     0
## [11,]    0    0    0    0    0    0    0    1    0     0
## [12,]    0    0    0    0    0    0    0    0    1     0
## [13,]    0    0    0    0    0    0    0    0    0     0
## [14,]    0    0    0    0    0    0    0    0    0     0
## [15,]    0    0    0    0    0    0    0    0    0     0
## [16,]    0    0    0    0    0    0    0    0    0     1
```

```
svd(elim_n(5))
```

```
## $d
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## $u
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
##  [1,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0     0     0     0     0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0     0     0     0     0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0     0     0     0     0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0     0     0     0     0     0
##  [6,]    0    0    0    0    0    1    0    0    0     0     0     0     0     0     0
##  [7,]    0    0    0    0    0    0    1    0    0     0     0     0     0     0     0
##  [8,]    0    0    0    0    0    0    0    1    0     0     0     0     0     0     0
##  [9,]    0    0    0    0    0    0    0    0    1     0     0     0     0     0     0
## [10,]    0    0    0    0    0    0    0    0    0     1     0     0     0     0     0
## [11,]    0    0    0    0    0    0    0    0    0     0     1     0     0     0     0
## [12,]    0    0    0    0    0    0    0    0    0     0     0     1     0     0     0
## [13,]    0    0    0    0    0    0    0    0    0     0     0     0     1     0     0
## [14,]    0    0    0    0    0    0    0    0    0     0     0     0     0     1     0
## [15,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     1
##
## $v
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15]
##  [1,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0
##  [2,]    0    1    0    0    0    0    0    0    0     0     0     0     0     0     0
##  [3,]    0    0    1    0    0    0    0    0    0     0     0     0     0     0     0
##  [4,]    0    0    0    1    0    0    0    0    0     0     0     0     0     0     0
##  [5,]    0    0    0    0    1    0    0    0    0     0     0     0     0     0     0
##  [6,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
##  [7,]    0    0    0    0    0    1    0    0    0     0     0     0     0     0     0
##  [8,]    0    0    0    0    0    0    1    0    0     0     0     0     0     0     0
##  [9,]    0    0    0    0    0    0    0    1    0     0     0     0     0     0     0
## [10,]    0    0    0    0    0    0    0    0    1     0     0     0     0     0     0
## [11,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [12,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [13,]    0    0    0    0    0    0    0    0    0     1     0     0     0     0     0
## [14,]    0    0    0    0    0    0    0    0    0     0     1     0     0     0     0
## [15,]    0    0    0    0    0    0    0    0    0     0     0     1     0     0     0
## [16,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [17,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [18,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [19,]    0    0    0    0    0    0    0    0    0     0     0     0     1     0     0
## [20,]    0    0    0    0    0    0    0    0    0     0     0     0     0     1     0
## [21,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [22,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [23,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [24,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0
## [25,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     1
```

33 Exercise 35

```r
vec<-function(A) matrix(c(A),ncol=1)

#Commutation matrix
m<-function(m,n){
  o<-matrix(0,m*n,m*n)
  k<-1 #counter for moving to the right within the commutation matrix
  c<-1 #counter for starting at previous column +1
  for(i in 1:(m*n)){
    o[i,k]<-1
    k<-k+m
    if(k>m*n) {
      c<-c+1
      k<-c}
  }
  o
}

#Check the case where m<n
A<-matrix(1:14,2,7)
m(2,7)%*%vec(A)==vec(t(A))
```

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

```r
#check the case where m>n
A<-matrix(1:10,5,2)
m(5,2)%*%vec(A)==vec(t(A))
```

```
##       [,1]
##  [1,] TRUE
##  [2,] TRUE
##  [3,] TRUE
##  [4,] TRUE
##  [5,] TRUE
##  [6,] TRUE
##  [7,] TRUE
##  [8,] TRUE
##  [9,] TRUE
## [10,] TRUE
```

```r
#check the case where m=n
A<-matrix(1:2500,50,50)
```

```r
sum(m(50,50)%*%vec(A)==vec(t(A)))==2500

## [1] TRUE
```

34 Exercise 36

First we check some examples in R to investigate the underlying patterns:

```r
#m defined in previous exercise
svd(m(3,2))$d

## [1] 1 1 1 1 1 1

svd(m(2,2))$d

## [1] 1 1 1 1

svd(m(3,5))$d

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

In all of our examples, we see that singular values are 1. This is not a coincidence of course. Instead, this results from the properties of commutation matrices. Note that permutation matrices are orthogonal (because its column vectors are just permutations of the corresponding unit vectors) and that a commutation matrix is just a special case of a permutation matrix. Consequently, we can apply the fact that the eigenvalues of orthogonal matrices are ± 1 (which was proved in the previous assignment). Then we know that singular values are $\sqrt{\lambda_i}$, where λ_i denotes eigenvalues of the associated square Gram matrix G ($G = KK^\intercal$), thus the singular values of a commutation matrix are 1.

35 Exercise 37

Let A be an arbitrary matrix, then, using the singular value decomposition, we have that $A = U\Sigma V^\intercal$, where U and V are orthogonal matrices and Σ is a (generalised) diagonal matrix (not necessarily square). Then the Moore-Penrose inverse (A^+) is as follows $A^+ = U\Sigma^+ V^\intercal$, where Σ^+ is the original Σ modified in the following way: take the reciprocal of each non-zero diagonal element, leaving the zeros in place, and then transpose the matrix. Computationally, additional complexity arises out of the question of how to decide whether or not a singular value is numerically zero. What our function does is, first, setting all singular values smaller or equal to a pre-set threshold (parameter tol in function below) to 0 and returning the reciprocal for the rest. Then those elements are put back into the a main diagonal of Σ, where previous singular values are replaced and the Σ^+ is obtained.

```r
mp_inv<-function(A,tol){
  dec<-svd(A)
  dec$v%*%t(diag(vapply(dec$d, FUN = function(x) ifelse(x<=tol,0,1/x),
                        numeric(1))))%*%t(dec$u)
}

#check
set.seed(1)
A<-matrix(runif(24),6,4)
round(mp_inv(A,10^-12)%*%A,12)

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

```
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

36 Exercise 38

```
wpctrace<-function(A,w){
  sum(w%*%A^2)
}

wpctrace_base<-function(A,w){
  sum(diag(t(A)%*%diag(w)%*%A))
}

#check
A<-matrix(1:100,10,10)
w<-1:10

wpctrace(A,w)

## [1] 1944250

wpctrace_base(A,w)

## [1] 1944250

#efficiency for small matrix:
system.time(for(i in 1:1000000) wpctrace(A,w))

##    user  system elapsed
##    1.96    0.04    2.02

system.time(for(i in 1:1000000) wpctrace_base(A,w))

##    user  system elapsed
##   25.64    0.04   25.83

#efficiency for larger matrices
A<-matrix(1:2500,50,50)
w<-1:50

system.time(for(i in 1:100000) wpctrace(A,w))

##    user  system elapsed
##    1.16    0.04    1.22

system.time(for(i in 1:100000) wpctrace_base(A,w))

##    user  system elapsed
##   13.65    0.00   13.66
```

To conclude, our wpctrace function provides a significantly more efficient computation that the simple case (wpctrace_base).

37 Exercise 39

```r
#benchmark - using just the formula:
b_dmvnorm<-function(x,m,V) {
  output<-numeric()
  inv<-solve(V)
  for(i in 1:nrow(x))
  {
    a<-(x[i,])
    output[i]<-(det(2*pi*V))^(-1/2)*exp((-t(a-m)%*%inv%*%(a-m))/2)
  }
  output
}

#out efficient function
dmvnorm<-function(x,m,V)
{
  d<-eigen(V)
  #scaling factor in front of exponential, where we use property of determinant and
  #eigenvalues which have already been computed
  scaling_factor<-(prod(2*pi*d$values))^(-1/2)
  #Using the fact that eigenvalues are non-zero for positive definite matrices we
  #decompose the V=BB^t and B is as follows:
  B<-d$vectors%*%diag(d$values^0.5)
  #Then we normalize X
  Z<-solve(B)%*%t((x-rep(1,nrow(x))%*%t(m)))
  #And extract the elements on the diagonal, since only those correspond to correspond to
  #quadratic form and corresponding of the covariance matrix and corresponding vector
  scaling_factor*diag(exp(-1/2*t(Z)%*%Z))
}

#check
x<-matrix(1:4,2,2)
m<-c(1,0.5)
V<-matrix(c(2,2,2,3),2,2)

dmvnorm(x,m,V)

## [1] 0.004944642 0.003850891

b_dmvnorm(x,m,V)

## [1] 0.004944642 0.003850891

#Functions work well

#performane check on large matrix X and running a loop to properly test performance
set.seed(15)
x<-matrix(runif(2000),ncol=10)
set.seed(1)
m<-runif(10)
set.seed(31)
V<-matrix(runif(100),10)
V<-t(V)%*%V

system.time(for(i in 1:1000) b_dmvnorm(x,m,V))

##    user  system elapsed
##    6.64    0.00    6.64
```

```r
system.time(for(i in 1:1000) dmvnorm(x,m,V))
```

```
##    user  system elapsed
##    2.24    0.01    2.25
```

```r
#Apparently our function is roughly two times faster than our benchmark
```

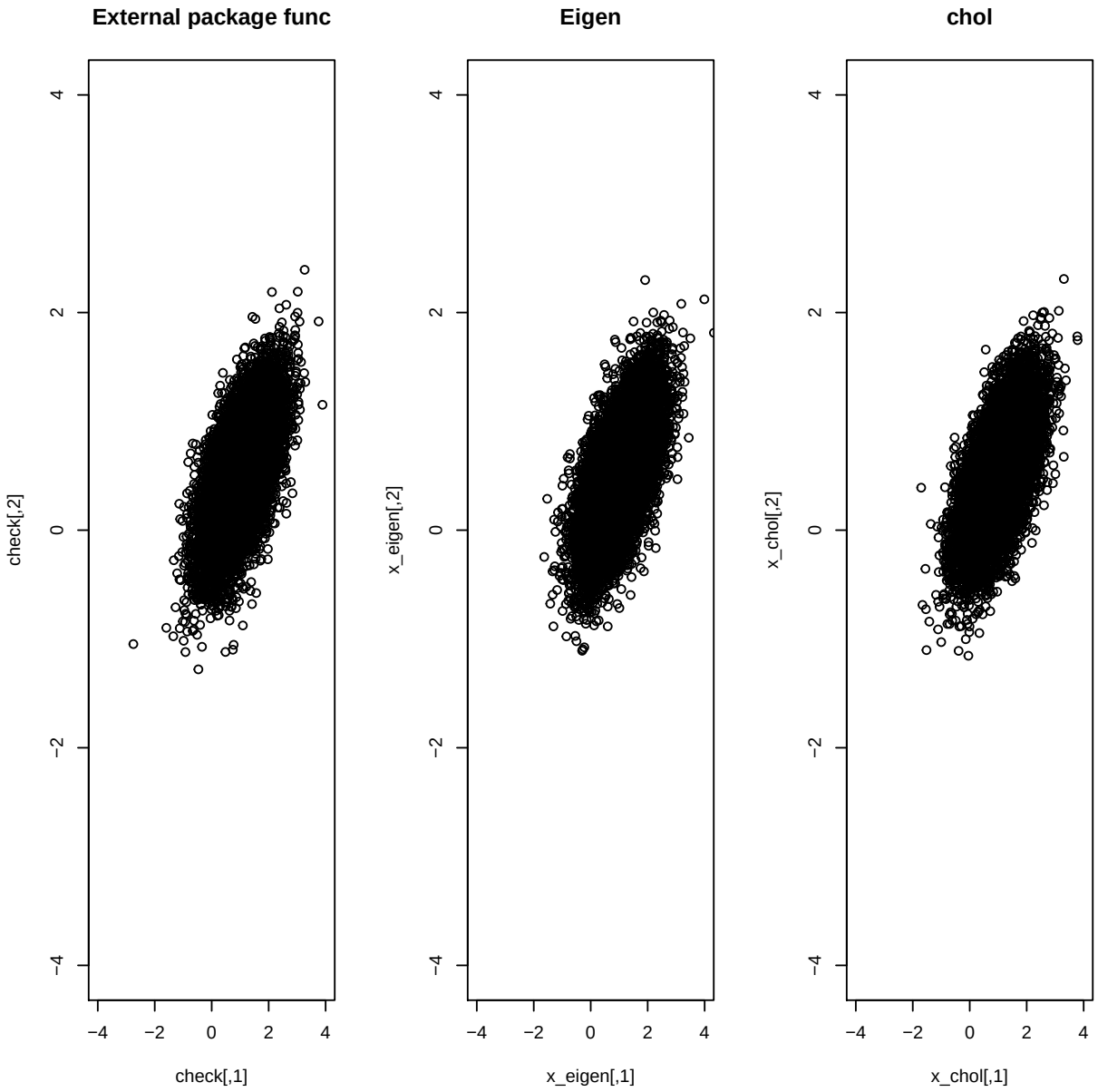
38 Exercise 41

```r
#Note that rmvnorm_my is the function which in the exercise correposponds to rmvnorm
rmvnorm_my<-function(n,m,Sigma,method="eigen")
{
  if(method=="eigen")
   {
  rows<-nrow(Sigma)
  univariate_normals<-matrix(0,n,rows)
  for(i in 1:rows)
  {
    univariate_normals[,i]<-rnorm(n)
  }
  # #These together make a multivariate normal N(0,I)
  # #now we apply the linear transformation from N(0,I) to N(m,Sigma)
  decom<-eigen(Sigma)
  A<-decom$vectors%*%sqrt(diag(decom$values))
  rep(1,nrow(univariate_normals))%*%t(m)+univariate_normals%*%t(A)}else{
    if(method=="chol")
  {
    rows<-nrow(Sigma)
  univariate_normals<-matrix(0,n,rows)
  for(i in 1:rows)
  {
    univariate_normals[,i]<-rnorm(n)
  }
  A<-chol(Sigma)
  Y<-univariate_normals%*%A+ rep(1,nrow(univariate_normals))%*%t(m)
  }else{
    "Wrong method"
  }}
}

#check
library(mvtnorm) #External package to compare results and check
```

```
##
## Attaching package:  'mvtnorm'
## The following object is masked _by_ '.GlobalEnv':
##
##     dmvnorm
```

```r
m<-c(1,0.5)
Sigma<-matrix(c(0.5,0.2,0.2,.2),2)
x_eigen<-rmvnorm_my(10000,m,Sigma)
x_chol<-rmvnorm_my(10000,m,Sigma,method="chol")
```

```
check<-rmvnorm(10000,m,Sigma)
par(mfrow=c(1,3))
plot(check,ylim=c(-4,4),xlim=c(-4,4),main="External package func")
plot(x_eigen,ylim=c(-4,4),xlim=c(-4,4),main="Eigen")
plot(x_chol,ylim=c(-4,4),xlim=c(-4,4),main="chol")
```




```
cov(x_eigen)

##           [,1]      [,2]
## [1,] 0.5025441 0.1985691
## [2,] 0.1985691 0.1981957

cov(x_chol)
```

```
##            [,1]      [,2]
## [1,] 0.4934177 0.1951151
## [2,] 0.1951151 0.1949786
```

```
cov(check)
```

```
##          [,1]      [,2]
## [1,] 0.483821 0.1962170
## [2,] 0.196217 0.2039657
```

```
colMeans(x_eigen)
```

```
## [1] 0.9998872 0.5051516
```

```
colMeans(x_chol)
```

```
## [1] 1.0128153 0.5019869
```

```
colMeans(check)
```

```
## [1] 0.9965546 0.5000767
```

```
library(MVN) #External package to test normality of the simulated data
```

```
## Warning:  package 'MVN' was built under R version 3.4.3
## sROC 0.1-2 loaded
```

```
mardiaTest(x_eigen)
```

```
##    Mardia's Multivariate Normality Test
## ---------------------------------------
##    data : x_eigen
##
##    g1p            : 0.001816979
##    chi.skew       : 3.028298
##    p.value.skew   : 0.553101
##
##    g2p            : 8.091449
##    z.kurtosis     : 1.143109
##    p.value.kurt   : 0.2529931
##
##    chi.small.skew : 3.029812
##    p.value.small  : 0.5528488
##
##    Result         : Data are multivariate normal.
## ---------------------------------------
```

```
mardiaTest(x_chol)
```

```
##    Mardia's Multivariate Normality Test
## ---------------------------------------
##    data : x_chol
##
##    g1p            : 0.003105369
##    chi.skew       : 5.175614
##    p.value.skew   : 0.2697483
##
```

```
##    g2p            : 7.932291
##    z.kurtosis     : -0.8463566
##    p.value.kurt   : 0.3973539
##
##    chi.small.skew : 5.178203
##    p.value.small  : 0.2694966
##
##    Result         : Data are multivariate normal.
## ---------------------------------------
```

39 Exercise 42

39.1 Part a

Based on discussion with professor Hornik we are supposed to show that $F, \epsilon_1, \ldots, \epsilon_d$ have zero mean and are mutually uncorrelated with $Var(F) = 1$ and $Var(\epsilon_i) = 1 - \rho$. Thus the mean follows as

$$\begin{aligned} \mathbb{E}(F) = \frac{\sqrt{\rho}}{1+\rho(d-1)}\,\mathbb{E}(\sum_{j=1}^{d} X_j) + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}\,\mathbb{E}(Y) = \\ = \frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}\mathbb{E}(X_j) + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}0 = \frac{\sqrt{\rho}}{1+\rho(d-1)}(\sum_{j=1}^{d}0) = 0 \end{aligned} \tag{12}$$

$$\mathbb{E}(\epsilon_i) = \mathbb{E}(X_j) - \sqrt{\rho}\,\mathbb{E}(F) = 0 \tag{13}$$

Next we compute the covariances:

$$\begin{aligned} Cov(F, \epsilon_j) = \mathbb{E}((F - \mathbb{E}(F))(\epsilon_j - \mathbb{E}(\epsilon_j))) = \mathbb{E}(F\epsilon_j) = \mathbb{E}(F(X_j - \sqrt{\rho}F)) = \mathbb{E}(FX_j) - \sqrt{\rho}\,\mathbb{E}(F^2) = \\ = \mathbb{E}(X_j(\frac{\sqrt{\rho}}{1+\rho(d-1)}\sum_{j=1}^{d}X_j + \sqrt{\frac{1-\rho}{1+\rho(d-1)}}Y)) - \sqrt{\rho} = \frac{\sqrt{\rho}}{1+\rho(d-1)} \times (1+\rho(d-1)) - \sqrt{\rho} = 0 \end{aligned} \tag{14}$$

Note here we use that the variance of F is 1, which is showed in Equation 16. We also use the property that X_j is independent of Y. We follow by computing:

$$\begin{aligned} Cov(\epsilon_i, \epsilon_j) = \mathbb{E}((\epsilon_i - \mathbb{E}(\epsilon_i))(\epsilon_j - \mathbb{E}(\epsilon_j))) = \mathbb{E}(\epsilon_i\epsilon_j) = \mathbb{E}((X_i - \sqrt{\rho}F)(X_j - \sqrt{\rho}F)) = \\ = \mathbb{E}(X_iX_j) - \sqrt{\rho}\,\mathbb{E}(X_iF) - \sqrt{\rho}\,\mathbb{E}(X_jF) + \rho\,\mathbb{E}(F^2) = 2\rho - 2 \times \frac{\sqrt{\rho}\sqrt{\rho}}{1+\rho(d-1)} \times (1+\rho(d-1)) = 0 \end{aligned} \tag{15}$$

Note here we again use that the variance of F is 1 and the property that X is independent of Y. Next we compute:

$$\begin{aligned} var(F) = \mathbb{E}(F^2) = \mathbb{E}((\frac{X_i - \epsilon_i}{\sqrt{\rho}})^2) = \frac{1}{\rho}\,\mathbb{E}(X_i^2 - 2X_i\epsilon_i + \epsilon_i^2) = \frac{1}{\rho}(1 - 2\,\mathbb{E}(X_i\epsilon_i) + \mathbb{E}(\epsilon_i^2)) = \\ = \frac{1}{\rho}(1 - 2\,\mathbb{E}(X_i^2) + 4\sqrt{\rho}\,\mathbb{E}(X_iF) - 2\rho\,\mathbb{E}(F^2) + \mathbb{E}(X_i^2) - 2\sqrt{\rho}\,\mathbb{E}(X_iF) + \rho\,\mathbb{E}(F^2)) = \\ = \frac{1}{\rho}(2\rho - \rho\,\mathbb{E}(F^2)) = 2 - \mathbb{E}(F^2) \end{aligned} \tag{16}$$

Next we follow by:

$$var(\epsilon_i) = \mathbb{E}((\epsilon_i - \mathbb{E}(\epsilon_i))^2) = \mathbb{E}(\epsilon_i^2) = \mathbb{E}(X_i^2) - 2\sqrt{\rho}\,\mathbb{E}(X_iF) + \rho\,\mathbb{E}(F^2) = 1 - 2\sqrt{\rho}\sqrt{\rho} + \rho = 1 - \rho \tag{17}$$

39.2 Part b

To show that $X_i = \sqrt{\rho}F + \sqrt{1-\rho}Z_i$ is a linear factor model we have to show that it satisfies the properties of the linear factor model. The random variable X follows a 1 -factor model if can be decomposed as $X = a + BF + \epsilon$. Thus in this case $a = 0$, $B = \sqrt{\rho}$ and $\epsilon = \sqrt{1-\rho}Z_i$. Then

- F is a random variable by the task itself
- ϵ_i are idiosyncratic with zero mean and variance 1 by the exercise itself (iid)
- $a \in \mathbb{R}$ and $B \in \mathbb{R}^{1\times 1}$
- $cov(F, \epsilon)$ should be equal to 0, thus

$$cov(F, \sqrt{1-\rho}Z_i) = \mathbb{E}(F\sqrt{1-\rho}Z_i) = 0 \tag{18}$$

Thus all the properties of the one-dimensional factor model are satisfied.

```
fun_42<-function(n,d,rho)
{
  output<-numeric()
  F<-rnorm(n)
  for(i in 1:n)
  {
    output<-rbind(output,sqrt(rho)*F[i]+sqrt(1-rho)*rnorm(d))
  }
    output
}

#check correlation
x<-fun_42(10000,5,0.3)
cor(x)

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.2861326 0.3023996 0.3055412 0.2920091
## [2,] 0.2861326 1.0000000 0.2928439 0.2896374 0.2905196
## [3,] 0.3023996 0.2928439 1.0000000 0.2937129 0.3154607
## [4,] 0.3055412 0.2896374 0.2937129 1.0000000 0.3023545
## [5,] 0.2920091 0.2905196 0.3154607 0.3023545 1.0000000

#check normality
library(MVN)
colMeans(x)

## [1]  0.011494304 -0.000748468 -0.002125047 -0.004483570  0.001853539

cov(x)

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0093257 0.2868756 0.3034035 0.3070035 0.2938085
## [2,] 0.2868756 0.9959128 0.2918573 0.2890834 0.2903611
## [3,] 0.3034035 0.2918573 0.9973495 0.2933625 0.3155159
## [4,] 0.3070035 0.2890834 0.2933625 1.0002664 0.3028493
## [5,] 0.2938085 0.2903611 0.3155159 0.3028493 1.0030086

mardiaTest(x)
```

```
##    Mardia's Multivariate Normality Test
## ---------------------------------------
##    data : x
##
##    g1p            : 0.01247435
##    chi.skew       : 20.79059
##    p.value.skew   : 0.9727083
##
##    g2p            : 35.3333
##    z.kurtosis     : 1.991844
##    p.value.kurt   : 0.04638816
##
##    chi.small.skew : 20.79891
##    p.value.small  : 0.9726192
##
##    Result          : Data are not multivariate normal.
## ---------------------------------------
```

40 Exercise 43

Let d-dimensional random vector X has a (multivariate) normal variance mixture distribution such that $X \overset{d}{=} m + \sqrt{W}AZ$, where for some k $Z \sim N_k(0; I_k)$, W is a non-negative, scalar-valued random variable which is independent of Z and $A \in R^{d\times k}$ and $m \in R^d$ are a matrix and a vector of constants, respectively. Next we assume that W has finite expectation, thus

$$\mathbb{E}(X) = \mathbb{E}(m + \sqrt{W}AZ) = m + \mathbb{E}(\sqrt{W})A\,\mathbb{E}(Z) = m \tag{19a}$$

$$cov(X) = \mathbb{E}((\sqrt{W}AZ)(\sqrt{W}AZ)^\intercal) = \mathbb{E}(W)A\,\mathbb{E}(ZZ^\intercal)A^\intercal = \mathbb{E}(W)\Sigma \tag{19b}$$

where $\Sigma = AA^\intercal$ by definition of multivariate mixture distribution.

Next let W have an inverse Gamma distribution with parameters $\nu/2$ and $\nu/2$, which is equivalent to saying that ν/W has a chi-squared distribution with ν degrees of freedom, then X has a multivariate t distribution $t_d \sim (\nu, m, \Sigma)$ with parameters ν (degrees of freedom), m (mean) and Σ (dispersion). Let $\nu > 2$ then

$$\mathbb{E}(X) = \mathbb{E}(m + \sqrt{W}AZ) = m + \mathbb{E}(\sqrt{W})A\,\mathbb{E}(Z) = m \tag{20a}$$

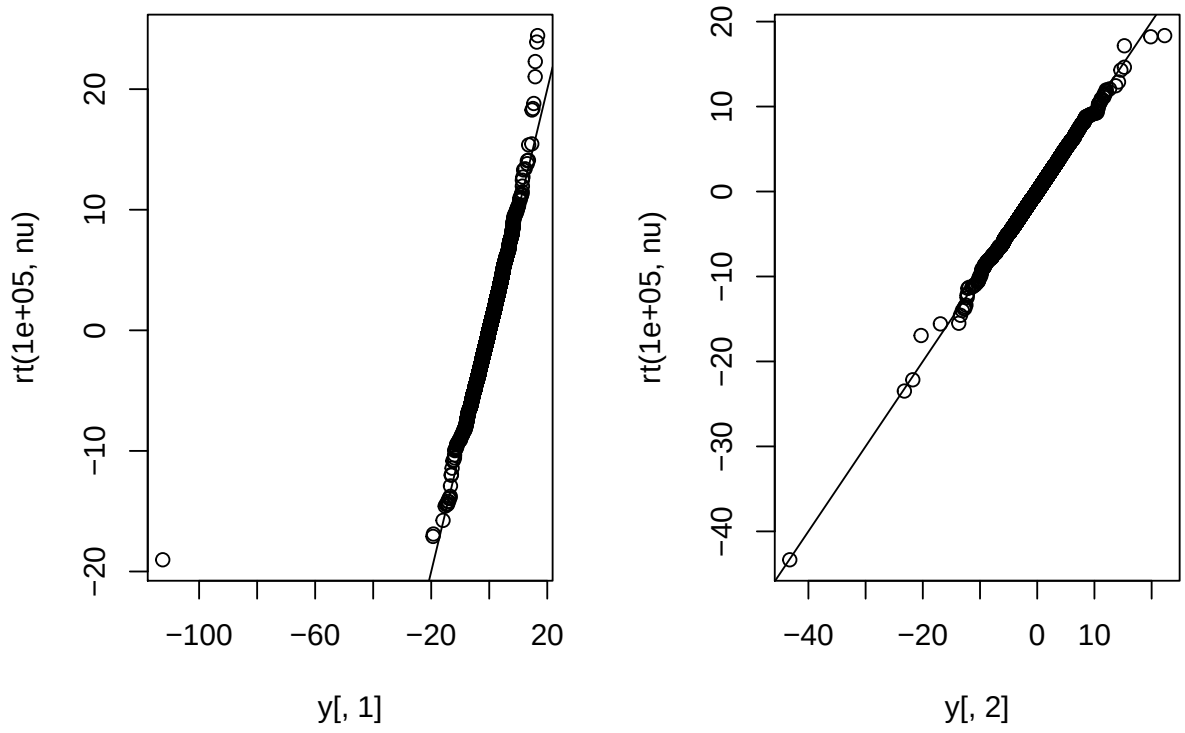
Since the expected value of the inverse gamma distribution is in our case $\mathbb{E}(W) = \frac{\nu}{\nu-2}$ we get that the covariance of X is as follows:

$$cov(X) = \mathbb{E}((\sqrt{W}AZ)(\sqrt{W}AZ)^\intercal) = \mathbb{E}(W)A\,\mathbb{E}(ZZ^\intercal)A^\intercal = \frac{\nu}{\nu-2}\Sigma \tag{21a}$$

```
rmvt_my<-function(n,nu,m,sigma)
{
  #First we generate the normal distribution with zero mean and
  #covariance matrix using function from exercise 41
  rmvnorm_my<-function(n,m,Sigma,method="eigen")
  {
  if(method=="eigen")
   {
  rows<-nrow(Sigma)
  univariate_normals<-matrix(0,n,rows)
  for(i in 1:rows)
```

```
  {
    univariate_normals[,i]<-rnorm(n)
  }
  decom<-eigen(Sigma)
  A<-decom$vectors%*%sqrt(diag(decom$values))
  rep(1,nrow(univariate_normals))%*%t(m)+univariate_normals%*%t(A)}else{
    if(method=="chol")
  {
    rows<-nrow(Sigma)
  univariate_normals<-matrix(0,n,rows)
  for(i in 1:rows)
  {
    univariate_normals[,i]<-rnorm(n)
  }
  A<-chol(Sigma)
  Y<-univariate_normals%*%A+ rep(1,nrow(univariate_normals))%*%t(m)
  }else{
    "Wrong method"
  }}
}
  #get the normal
   Z<-rmvnorm_my(n,rep(0,nrow(sigma)),sigma)
   #Next have to create the W
   W<-nu/rchisq(n,nu)
   #And finally we compute the random vectors from multivariate t-dist
   m+sqrt(W)*Z
}

#check for case when nu=4
nu=4
m<-c(0,0)
sigma<-matrix(c(1,0.3,0.3,1),2,2)

y<-rmvt_my(100000,nu,m,sigma)
colMeans(y)

## [1] -0.001676958 -0.001521260

cov(y)

##           [,1]      [,2]
## [1,] 2.1267898 0.6523838
## [2,] 0.6523838 2.0101552

cov(y)-sigma*nu/(nu-2)

##            [,1]       [,2]
## [1,] 0.12678983 0.05238376
## [2,] 0.05238376 0.01015516

#Note sigma is not the covariance matrix but scale matrix, thus it has
#to be modified to get the theoretical covariance

#qqplot
par(mfrow=c(1,2))
```

```
qqplot(y[,1],rt(100000,nu))
abline(0,1)
qqplot(y[,2],rt(100000,nu))
abline(0,1)
```

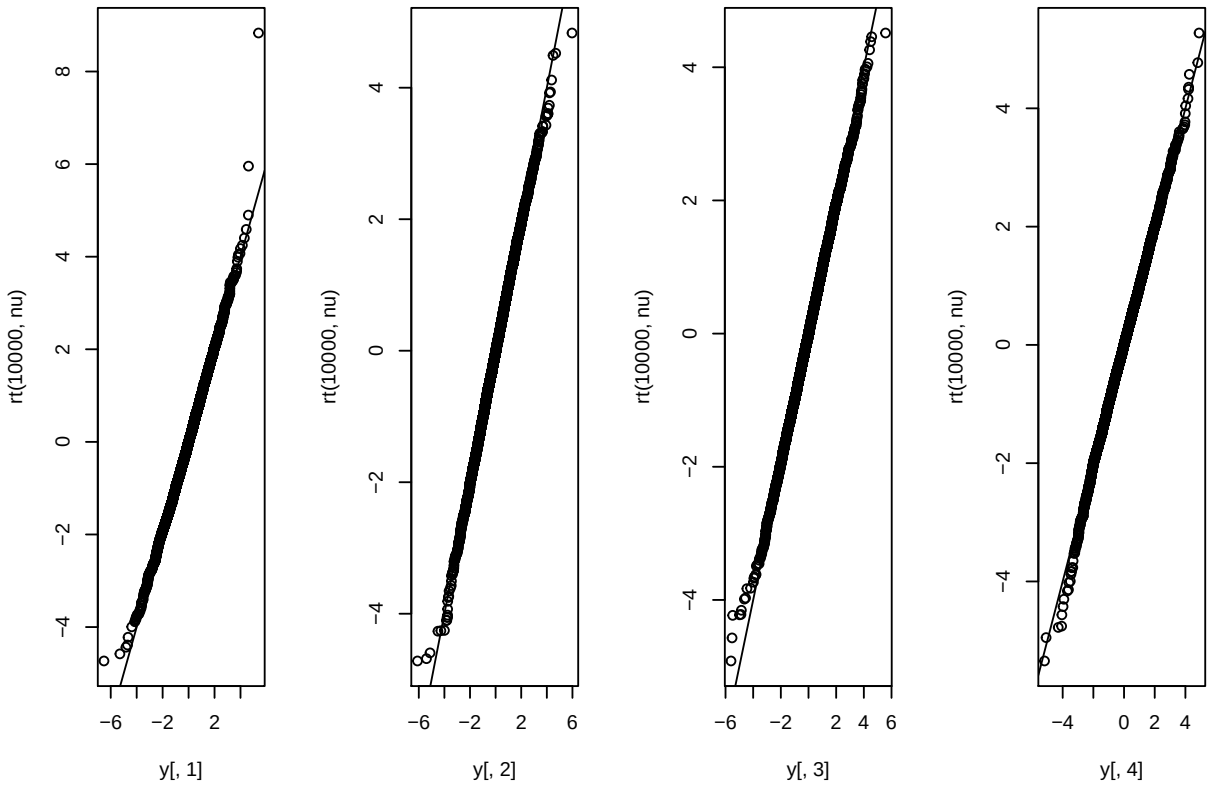



```
#Based on theory, both marginals should be t distributed with the same degrees of
#freedom, which is also suggested by the qq plots


#check for case when nun=13
nu=13
m<-c(0,0,0,0)
sigma<-matrix(c(1,0.3,0.3,0.3,0.3,1,0.3,0.3,0.3,0.3,1,0.3,0.3,0.3,0.3,1),4,4)

y<-rmvt_my(10000,nu,m,sigma)
colMeans(y)

## [1] -0.016827954  0.014476850  0.007639833  0.005002057

cov(y)

##           [,1]      [,2]      [,3]      [,4]
## [1,] 1.1608991 0.3232026 0.3663258 0.3488904
## [2,] 0.3232026 1.1630797 0.3480212 0.3331543
## [3,] 0.3663258 0.3480212 1.2353387 0.3579109
## [4,] 0.3488904 0.3331543 0.3579109 1.1418338
```

```r
cov(y)-sigma*nu/(nu-2)
```

```
##               [,1]          [,2]          [,3]          [,4]
## [1,] -0.020919117 -0.031342836  0.011780303 -0.005655032
## [2,] -0.031342836 -0.018738445 -0.006524209 -0.021391200
## [3,]  0.011780303 -0.006524209  0.053520476  0.003365494
## [4,] -0.005655032 -0.021391200  0.003365494 -0.039984399
```

```r
#qqplot
par(mfrow=c(1,4))
qqplot(y[,1],rt(10000,nu))
abline(0,1)
qqplot(y[,2],rt(10000,nu))
abline(0,1)
qqplot(y[,3],rt(10000,nu))
abline(0,1)
qqplot(y[,4],rt(10000,nu))
abline(0,1)
```



41 Exercise 44

Let $-1 < \rho < 1$ and let (X_1, X_2) have a bivariate standard normal distribution with correlation ρ. Then let $f(x_1, x_2)$ be the density function of this bivariate distribution, $f_{x_1}(x_1)$ be the marginal density of X_1 and $f_{X_2|X_1=x}(x_1, x_2)$ be the conditional density function of X_2 given $X_1 = x$ then:

$$f_{X_2|X_1=x}(x_1,x_2)=\frac{f(x_1,x_2)}{f_{x_1}(x_1)}=\frac{\frac{1}{\sqrt{2\pi V}}\exp\frac{-(x,x_2)V^{-1}(x,x_2)^\intercal}{2}}{\frac{1}{\sqrt{2\pi}}\exp\frac{-x^2}{2}}=\frac{\frac{1}{2\pi\sqrt{1-\rho^2}}\exp\frac{-(x^2-2\rho x x_2+x_2^2)}{2(1-\rho^2)}}{\frac{1}{\sqrt{2\pi}}\exp\frac{-x^2}{2}}= \\ =\frac{1}{\sqrt{2\pi(1-\rho^2)}}\exp\frac{-x^2+2\rho x x_2-x_2^2+x^2-\rho^2x^2}{2(1-\rho^2)}=\frac{1}{\sqrt{2\pi(1-\rho^2)}}\exp\frac{-(x_2-\rho x)^2}{2(1-\rho^2)} \tag{22}$$

Since the term above is a density of normal distribution with mean ρx and variance $1-\rho^2$, it is clear that $X_2|X_1=x\sim N(\rho x,1-\rho^2)$. Next we follow by

$$\lim_{x\to-\infty}\mathbb{P}(X_2\leq x|X_1=x)=\lim_{x\to-\infty}\phi(\frac{x-\rho x}{\sqrt{1-\rho^2}})=\lim_{x\to-\infty}\phi(\frac{x\sqrt{1-\rho}}{\sqrt{1+\rho}})=0 \tag{23}$$

Note here we use what had been proven before, i.e. the conditional probability is univariate normal. Next we follow by:

$$\lim_{x\to-\infty}\mathbb{P}(X_2\leq x|X_1=x)=t_{\nu+1}\left(-\sqrt{\frac{(\nu+1)(1-\rho)}{1+\rho}}\right) \tag{24}$$

which follows from:

$$\lim_{x\to-\infty}=\sqrt{\frac{\nu+1}{\nu+x^2}}\frac{x-\rho x}{\sqrt{1-\rho^2}}=\left(-\sqrt{\frac{(\nu+1)(1-\rho)}{1+\rho}}\right) \tag{25}$$

Thus unless $\rho=-1$ there is an asymptotic dependence in lower (but also in upper) tail.

42 Exercise 47

Part (a):

```
set.seed(19908)
U<-runif(1000)
#calculation of mean
mean(U)

## [1] 0.496057

#comparison to true mean
0.5-mean(U)

## [1] 0.003943025

#calculation of variance
var(U)

## [1] 0.08148008

#comparinson to true variance
1/12-var(U)

## [1] 0.001853252

#calculation of standard deviation
sd(U)

## [1] 0.2854472

#comparison to true standard deviation
sqrt(1/12)-sd(U)

## [1] 0.003227975
```

Part (b):

Comparison Table	R output	True(Theoretial)	Discrepancy
Mean	0.496057	$\frac{1}{2}$	0.003943025
Variance	0.08148008	$\frac{1}{12}$	0.001853252
Standard Deviation	0.2854472	$\sqrt{\frac{1}{12}}$	0.003227975

The discrepancy of the simulated parameters and the true values is clearly immaterial (based on the law of large numbers). However due to the inherent randomness of the experiment, there will of course remain a slight discrepancy.

Part (c):

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

## [1] 0.61
```

Here we have an outcome of 0.61 from the computation of proportion of random draws from the uniform distribution which are less than 0.6. The "true" probability that a uniform random variable U is smaller than 0.6 should be 0.6. Again, this difference results from the inherent randomness in the simulation experiment, and - on the basis of the law of large numbers - we expect the deviation to decrease for an increasing number of observations.

43 Exercise 48

Part (a):

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

## [1] 0.9981552

mean(U1)+mean(U2)

## [1] 0.9981552
```

Due to the linearity of the expectation operator, we have $\mathbb{E}(U_1 + U_2) = \mathbb{E}(U_1) + \mathbb{E}(U_2)$. The true value should be $0.5 + 0.5 = 1$. Obviously, mean$(U_1 + U_2)$ and mean(U_1)+mean(U_2) therefore deliver the same result, which is - in general - however slightly different to the true value of 1 due to the randomness of the simulation experiment.

Part (b):

```
var(U1+U2)

## [1] 0.163099

var(U1)+var(U2)

## [1] 0.1647748

var(U1+U2)==var(U1)+var(U2)

## [1] FALSE
```

As can be seen from this, our estimations of the underlying variance of the two independent random variables are not exactly equal, since the randomness of the simulation experiment will usually generate an ever so small covariance between the two theoretically independent random variables, so their covariance will not be exactly zero, which is why the sum of the variances does not exactly equal the variance of their sum. The true values, however, should be equal, since the two random variables are independent, which implies that the variance of their sum is equal to the sum of their individual variances.

Part (c):

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

## [1] 0.8792
```

Part (d):

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

## [1] 0.6582
```

44 Exercise 49

Here we set the observation number to 1,000,000 to get a large enough sample for the simulation.

```
U1<-runif(1000000)
U2<-runif(1000000)
U3<-runif(1000000)
```

Part (a):

```
mean(U1+U2+U3)

## [1] 1.499457
```

Part (b):

```
var(U1+U2+U3)

## [1] 0.2497903

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

## [1] 0.2499794
```

Part (c):

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

## [1] 1.205444
```

Part (d):

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

## [1] 0.997079
```

45 Exercise 50

Part (a):

```
#We create a (100 x 20) matrix, each row corresponds to one
#student, each column to one question in the test.
#0 indicates an incorrect answer, 1 indicates a correct answer.
testresponses<-replicate(20,sample(c(0,1),100,replace=TRUE))

#Average mark calculation, including standard deviation:
marks<-apply(testresponses,1,sum)/20
mean(marks) #logically around 50%

## [1] 0.5175

sd(marks)

## [1] 0.1128991
```

Part (b):

```
#Proportion of students with mark of >= 30%:
sum(marks>=0.3)/length(marks)

## [1] 0.99
```

46 Exercise 51

First, we set up a vector of draws from the binomial distribution:

```
set.seed(4711)
X<-rbinom(10000,20,0.3)
```

Part (a):

```
sum(X<=5)/length(X)

## [1] 0.4171

#True value:
binom_5<-numeric(6)
for(i in 0:5){
  binom_5[i+1]<-choose(20,i)*0.3^i*0.7^(20-i)
}
sum(binom_5)

## [1] 0.4163708

#Or just:
pbinom(5,20,0.3)

## [1] 0.4163708
```

Part (b):

```r
sum(X==5)/length(X)
```

```
## [1] 0.1766
```

```r
#True value:
choose(20,5)*0.3^5*0.7^15
```

```
## [1] 0.1788631
```

Part (c):

```r
mean(X)
```

```
## [1] 5.9843
```

```r
#True value:
20*0.3
```

```
## [1] 6
```

Part (d):

```r
var(X)
```

```
## [1] 4.127066
```

```r
#True value:
20*0.3*0.7
```

```
## [1] 4.2
```

Part (e):

```r
quantile(X,0.95)
```

```
## 95%
##   9
```

```r
#True value:
qbinom(0.95,20,0.3)
```

```
## [1] 9
```

Part (f):

```r
quantile(X,0.99)
```

```
## 99%
##  11
```

```r
#True value:
qbinom(0.99,20,0.3)
```

```
## [1] 11
```

Part (g):

```
quantile(X,0.999999)

## 99.9999%
##       13

#True value:
qbinom(0.999999,20,0.3)

## [1] 16
```

In order to estimate extreme quantities like the one in part (g) accurately, we need to have a large enough sample, so as to ensure that we can actually expect to encounter rare events like a 0.0001% upper tail event. In fact, for precisely this event to be expected to occur once in the sample, we would require a sample size of around 1,000,000. The effect of choosing a smaller sample can be seen directly in part (g), as there is already a notable deviation between estimated and true quantity: the estimate for the 99.9999th quantile based on our random sample is 13, yet the correct value is in fact 16. And this discrepancy increases for higher values of the quantiles.

47 Exercise 52

The inversion method uses the following theorem:
$\Pr(F^{-1}(U) \leq x) \Leftrightarrow \Pr(F(x) \geq U) \Leftrightarrow \Pr(U \leq F(x)) \Leftrightarrow F(x)$
where $U \sim \text{Uniform}(0,1)$ and F is the CDF of the corresponding distribution. Also, this implies that $F^{-1}(U) \sim F$.
The function *ranbin*1 takes n as the desired number of generated pseudorandom, i.i.d. binomially distributed numbers and *size* and *prob* as the parameters of the underlying distribution $Binom(n,p)$. The function first maps all the possible values from 0 to $size - 1$ of the random variables to the corresponding quantile of the cumulative distribution function and creates a vector of the cumulative probabilities. It then generates a pseudorandom number from the standard unifrom distribution, and maps it to a binomially distributed number by counting the number of probability mass values it dominates from the aforementioned vector. This algorithm conforms the general definition of the quantile function, such as $F^{-1}(q) = inf\{p : F(p) \geq q\}$, since the possible values of the binomial random variable are enumerated from zero (therefore, taking one more value into consideration) and thus sum up to the smallest possible value corresponding a higher cumulative probability than the uniformly distributed number. The function *ranbin*1 uses the built-in *replicate* function to repeat the procedure in the desired number n times.

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

system.time(ranbin1(1000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0

system.time(rbinom(1000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0
```

```
system.time(ranbin1(10000,10,0.4)) #0.05, 0, 0.05

##    user  system elapsed
##    0.02    0.00    0.02

system.time(rbinom(10000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0

system.time(ranbin1(100000,10,0.4)) #0.41, 0.01, 0.44

##    user  system elapsed
##    0.25    0.00    0.25

system.time(rbinom(100000,10,0.4)) #0.02, 0, 0.01

##    user  system elapsed
##       0       0       0
```

48 Exercise 53

The function *ranbin*2 takes n as the desired number of generated pseudorandom, i.i.d. binomially distributed numbers and *size* and *prob* as the parameters of the underlying distribution $Binom(n, p)$. The function simulates *size*-many independent pseudorandom, uniformly distributed numbers and enumerates those that remain below the *prob* parameter of the desired distribution. One can consider *prob* as the probability of a Bernoulli-experiment, that is attempted *size*-times in order to obtain a binomially distributed random variable. Here, the goal is practically the same by uniformly distributed numbers between 0 and 1 and thus simulating whether the experiment ws successful (in this case the generated number is between 0 and *prob*) or unsuccessful (if it is between *prob* and 1). In the end, the number of successful experiments are counted and a binomially distributed pseudorandom number is obtained. The *replicate* function is used to repeat this mechanism the number of desired times n.

```
ranbin2 <- function(n, size, prob) {
  singlenumber <- function(size, prob) {
    x <- runif(size)
    sum(x < prob)
    }
  replicate(n, singlenumber(size, prob))
  }
system.time(ranbin2(1000,10,0.4)) #0.02, 0, 0.02

##    user  system elapsed
##       0       0       0

system.time(rbinom(1000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0

system.time(ranbin2(10000,10,0.4)) #0.05, 0, 0.05

##    user  system elapsed
##    0.03    0.00    0.03
```

```
system.time(rbinom(10000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0

system.time(ranbin2(100000,10,0.4)) #0.45, 0, 0.45

##    user  system elapsed
##    0.32    0.00    0.31

system.time(rbinom(100000,10,0.4)) #0, 0, 0

##    user  system elapsed
##       0       0       0
```

49 Exercise 54

Part (a):

```
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))
}

ranbin3(100,20,0.4)

##   [1]  6  4 11  9  8  6  5 10  9 10  5  3  9 10 13  8 12  8  9  8 12  5 10  9  5  8  6  5 10 12 11
##  [32]  8 10 11  3  8  9  7  8  9 10  8  9  8  8 10  4 14  6  7  8  6  5  7  7 12  9  7  8  9  8  7
##  [63]  5  7  9 10 10  8 11  9 12  3  3  6  8 11 10 11  3 10  8  6  9  7 12  6  6  9 11  7  6  7 13
##  [94]  7 10  7  7 10  8  7

ranbin3(100,500,0.7)

##   [1] 360 346 354 350 349 354 341 344 361 356 348 355 352 362 360 360 357 356 368 345 342 358 346
##  [24] 343 330 341 357 347 341 340 356 349 344 344 354 349 361 340 360 356 336 363 333 345 356 354
##  [47] 353 365 354 343 335 340 323 358 368 363 337 355 352 342 347 348 353 364 348 353 360 343 353
##  [70] 352 338 356 345 352 371 326 351 368 351 345 355 341 347 357 367 342 348 365 349 346 346 347
##  [93] 347 355 338 368 351 360 353 358
```

Part (b) and (c): Here, U represents the pseudorandom, uniformly distributed number, while p is the *prob* parameter of the binomial distribution. If $U < p$, we know that U is a uniformly distributed random variable over $[0, prob)$, i.e. its density is

$$f_U(x) = \left\{ \begin{array}{ll} 1/p & \text{for } 0 \leq x < p \\ 0 & \text{else} \end{array} \right\}$$

Obviously, if we divide U by p we get back the standard uniform distribution over $[0, 1]$.
Using the previous explanation analogously, we can state that if $U \leq p$, than U has a density of 1 between p and 1 and zero everywhere else. Then, $U - p$ has a density of 1 over $[0, 1 - p]$, and $\frac{U-p}{1-p}$ has, again, standard uniform distribution.

Part (d): The function *ranbin*3 provides a parsimonious generation of i.i.d. pseudorandom binomially distributed numbers. takes n as the desired number of generated numbers and *size* and *prob* as the parameters of the underlying distribution $Binom(n, p)$. The function generates one uniformly distributed pseudorandom number and fills a *size*-length vector by using likewise-distributed numbers that are obtained using the logic explained above. Thus, the algorithm can generate *size*-many pseudorandom uniformly distributed numbers by using the built-in generator only once. The numbers which are larger than p imply a zero in the vector, while the rest imply a 1. The algorithm produces the results of *size*-many Bernoulli experiments. The number of successful experiments is stored and the *replicate* function is used to repeat the mechanism n times.

50 Exercise 55

To perform the simulation, we created a function that takes the desired probability that any two people share a birthday, the draw size of the simulation and the days (365 or 366) as argument. Its output is precisely the number of people it took in order to generate a probability of greater than "prob" of two people sharing the same birthday. We thus call the function with 0.5, 10,000 and both 365 and 366 days as arguments, and find that it usually gives 23 as an output, although there is of course some randomness in this simulation experiment.

```
ex_55<-function(prob,size,days){
  probab<-0
  result<-1
  while(probab<=prob){
  result<-result+1
  birthdays<-replicate(result,sample(days,size,replace=TRUE))
  duplicates<-apply(birthdays,1,function(x) ifelse(sum(duplicated(x))>0,1,0))
  probab<-sum(duplicates)/length(duplicates)
  }
  result
}
ex_55(0.5,10000,365)

## [1] 23

ex_55(0.5,10000,366)

## [1] 23
```

To calculate the probability that any two people in a group share a birthday analytically, we calculate the complementary probability, i.e. we find out how likely it is that none of them share a birthday. We make the assumption that there are only 365 days in a year, i.e. we ignore the possibility of any person having his birthday on the 29th February, mainly because the mathematical argument can be extended very easily to account for this case. The calculation is based on the following argument: the birthday of the first person does not matter, but the second person must have his birthday on one of the remaining 364 days, the third person on one of the remaining 363 days, and so on, so that none of them share a birthday. Since the

simulation yielded an answer of around 23, we calculate the probabilities for the cases of 21 to 24 people:

$$P(\text{"no share among 21"}) = \frac{364}{365} \cdot \frac{363}{365} \cdot \cdots \cdot \frac{345}{365} = \prod_{i=1}^{20} \frac{365-i}{365} \approx 0.556 \Rightarrow P(\text{"share among 21"}) = 1 - 0.556 = 0.444$$

$$P(\text{"no share among 22"}) = \frac{364}{365} \cdot \frac{363}{365} \cdot \cdots \cdot \frac{344}{365} = \prod_{i=1}^{21} \frac{365-i}{365} \approx 0.524 \Rightarrow P(\text{"share among 22"}) = 1 - 0.524 = 0.476$$

$$P(\text{"no share among 23"}) = \frac{364}{365} \cdot \frac{363}{365} \cdot \cdots \cdot \frac{343}{365} = \prod_{i=1}^{22} \frac{365-i}{365} \approx 0.493 \Rightarrow P(\text{"share among 23"}) = 1 - 0.493 = 0.507$$

$$P(\text{"no share among 24"}) = \frac{364}{365} \cdot \frac{363}{365} \cdot \cdots \cdot \frac{342}{365} = \prod_{i=1}^{23} \frac{365-i}{365} \approx 0.462 \Rightarrow P(\text{"share among 24"}) = 1 - 0.462 = 0.538$$

So, indeed, we confirm that the smallest number of persons required for the probability of two persons in a group to have the same birthday to be greater than one half is 23.

51 Exercise 56

Whether or not the quadratic equation $ax^2 + bx + c = 0$ has only real roots depends on the sign of the discriminant $b^2 - 4ac$. If this expression is non-negative, then the equation will have only real roots. Hence, we can use random sampling (sample size 1,000,000) to determine the probability that this will be the case. For this, we generate three vectors of coefficients (namely a, b and c), from the uniform distribution over the interval $[-1, 1]$ and determine the relative appearance of cases where $b^2 - 4ac \geq 0$.

```
coeff_a<-runif(1000000,-1,1)
coeff_b<-runif(1000000,-1,1)
coeff_c<-runif(1000000,-1,1)
discriminant<-coeff_b^2-4*coeff_a*coeff_c
sum(discriminant>=0)/length(discriminant)

## [1] 0.627127
```

Turning to the analytical justification, one needs to devise some densities in order to compute the probability $\Pr(4ac \leq b^2) = \Pr(ac \leq b^2/4)$. In particular, knowing the density of b^2 and ac is essential.
We derive the density of b^2 from its cumulative distribution function. Since we already know that the square of any number between $[-1, 1]$ is always between $[0, 1]$ we can define the distribution function of b^2, F_{b^2} over $z \in [0, 1]$.

$$F_{b^2}(z) = \Pr(b^2 \leq z) = \Pr(-\sqrt{z} \leq b \leq \sqrt{z}) = \frac{2\sqrt{z}}{2} = \sqrt{z} \tag{26}$$

In the previous equation, the probability of b being in an interval is exactly the ratio of the length of the interval and the length of $[-1, 1]$. Then, we obtain the density by

$$f_{b^2}(z) = \frac{d\sqrt{z}}{dz} = \frac{1}{2\sqrt{z}}.$$

For the derivation of the density of ac, we propose two different approaches. The first of them uses the convolution formula for the product of two independent random variables. Let $V = ac$ and $Y = c$. Then, the multivariate random variable $(a, c) = (\frac{V}{Y}, Y)$ with the Jacobian determinant $\frac{1}{|Y|}$. The density of the joint distribution (V, Y) is the following:

$$f_{(V,Y)}(v, y) = f_a(\frac{v}{y}) f_c(y) \frac{1}{|\, y \,|} = (\frac{1}{2} * \mathbb{1}_{(-1,1)}(\frac{v}{y}))(\frac{1}{2} * \mathbb{1}_{(-1,1)}(y)) * \frac{1}{|\, y \,|}$$

To obtain the density of V, we have to integrate out the y component from the joint probability distribution.

$$h_{ac}(v) = \frac{1}{4}\int_{-1}^{1} \frac{1}{\mid y \mid} * \mathbb{1}_{(-1,1)}(\frac{v}{y})dy =$$
$$\frac{1}{4}\int_{-1}^{0} \frac{1}{\mid y \mid} * \mathbb{1}_{-1\leq \frac{v}{y} \leq 1}dy + \frac{1}{4}\int_{0}^{1} \frac{1}{\mid y \mid} * \mathbb{1}_{-1\leq (\frac{v}{y}) \leq 1}dy =$$
$$\frac{1}{4}\int_{-1}^{0} \frac{1}{\mid y \mid} * \mathbb{1}_{-1\leq y\leq v\leq -y\leq 1}dy + \frac{1}{4}\int_{0}^{1} \frac{1}{\mid y \mid} * \mathbb{1}_{-1\leq -y\leq v\leq y\leq 1}dy =$$
$$\frac{1}{4}\mathbb{1}_{-1\leq v\leq 0}\int_{-1}^{v} \frac{1}{\mid y \mid}dy + \frac{1}{4}\mathbb{1}_{0\leq v\leq 1}\int_{-1}^{-v} \frac{1}{\mid y \mid}dy + \frac{1}{4}\mathbb{1}_{-1\leq v\leq 0}\int_{-v}^{1} \frac{1}{\mid y \mid}dy + \frac{1}{4}\mathbb{1}_{0\leq v\leq 1}\int_{v}^{1} \frac{1}{\mid y \mid}dy =$$
$$\frac{1}{2}\mathbb{1}_{-1\leq v\leq 0}log(\frac{1}{\mid v \mid}) + \frac{1}{2}\mathbb{1}_{0<v\leq 1}log(\frac{1}{\mid v \mid}) = \frac{1}{2}log(\frac{1}{\mid v \mid})\mathbb{1}_{-1<v\leq 1} = -\frac{1}{2}log(\mid v \mid)\mathbb{1}_{-1<v\leq 1}$$

Which means the density is defined over the interval $[-1, 1]$ which aligns with the intuitive expectations, since both a and c are defined over the same interval. One can also derive this density by avoiding the integrals. The distribution of ac is in fact the symmetrized version of the product of two i.i.d. standard uniform random variables over $[0, 1]$. If we call them u and v, then (by using the inverse transform method) we can also say that $-log(uv) = log(u) - log(v)$ is the sum of two exponentially distributed random variables, both having a scale parameter of 1. The exponentials are Gamma-distributed random variables at the same time, namely $exp(1) =^d \Gamma(1, 1)$ and $-log(uv) =^d exp(1) + exp(1) =^d \Gamma(2, 1)$. Then, we can find the density of ac by taking the exponential of the negative of a $\Gamma(2, 1)$ random variable and symmetrizing it over the x-axis.

$$f_{\Gamma(2,1)}(t)dt = \frac{1^2}{\Gamma(2)}t^{2-1}e^{-t}dt = te^{-t}dt \quad (t > 0)$$

Let z be the value of the random variable ac. Then $z = e^{-t}$ or $t = -log(z)$ where $0 < z < 1$. This implies that $dt = -\frac{dz}{z}$. Since this transformation also inverted the monotonicity of the function, we have to multiply by -1 to get back a positive density. The rewritten density will then be the following:

$$f_{\Gamma(2,1)}(t)dt = -\left(-log(z)e^{-(-log(z))}\left(-\frac{1}{z}\right)dz\right) = -log(z)dz \quad (0 < z < 1)$$

The symmetrization replaces z by $\mid z \mid$, thus allows its value to range from -1 to 1. It also divides the density by two:

$$f_{ac}(z)dz = \left\{ \begin{array}{ll} -\frac{1}{2}log(\mid z \mid)dz & \text{for } -1 < z < 1 \\ 0 & \text{else} \end{array} \right\}$$

The last part of the task is to determine the probability $\Pr(ac \leq b^2/4)$ which we can rewrite as the

following integral:

$$\int_{y=0}^{1}\int_{x=-1}^{y/4} f_a c(x) f_{b^2}(y) dx dy =$$
$$\int_{y=0}^{1} \frac{1}{2\sqrt{y}} \int_{x=-1}^{y/4} -\frac{1}{2} log(\mid x \mid) dx dy =$$
$$\int_{y=0}^{1} \frac{1}{4\sqrt{y}} (1 + \int_{x=-1}^{y/4} -log(\mid x \mid) dx) dy =$$
$$\int_{y=0}^{1} \frac{1}{4\sqrt{y}} (1 + (x - x * log(x))\Big|_0^{y/4}) dy =$$
$$\int_{y=0}^{1} \frac{1}{4\sqrt{y}} (1 + \frac{y}{4} - \frac{y}{4} * log(\frac{y}{4})) dy =$$
$$\int_{y=0}^{1} \frac{1}{4\sqrt{y}} dy + \int_{y=0}^{1} \frac{\sqrt{y}}{16} dy - \int_{y=0}^{1} \frac{\sqrt{y}}{16} * log(y) dy =$$
$$\int_{y=0}^{1} \frac{1}{4\sqrt{y}} dy + \int_{y=0}^{1} \frac{\sqrt{y}}{16} dy - \int_{y=0}^{1} \frac{\sqrt{y}}{16} * log(y) dy + \int_{y=0}^{1} \sqrt{y} * \frac{log4}{16} dy =$$
$$\frac{1}{2} + \frac{1}{24} + \frac{1}{36} + \frac{2}{3}\frac{1}{16} log(4) \approx 0.6272067$$

Which means that the simulation results almost exactly gave back the analytical solution.

52 Exercise 57

Let X be a random variable with cdf F and corresponding quantile function $F^{-1}(u) = \inf\{x : F(x) \geq u\}$. Note that the infimum will definitely be part of the set $\{x : F(x) \geq u\}$, since the distribution function is right-continuous and non-decreasing, so it could also be replaced by the minimum. We show that for $x \in \mathbb{R}$ and $0 \leq u \leq 1$, we have $F^{-1}(u) \leq x \iff F(x) \geq u$.
" $\Rightarrow$ " Let $F^{-1}(u) \leq x$. Then, by definition of the quantile function and the aforementioned fact of the replacibility of infimum and minimum for the quantile function, $x \in \{y : F(y) \geq u\}$. Consequently, we have that $F(x) \geq u$.
" $\Leftarrow$ " Let $F(x) \geq u$. This of course implies that $x \in \{y : F(y) \geq u\}$. Since $F^{-1}(u)$ is defined as the infimum (or minimum, see above) of this set, we hence must have that $F^{-1}(u) \leq x$.

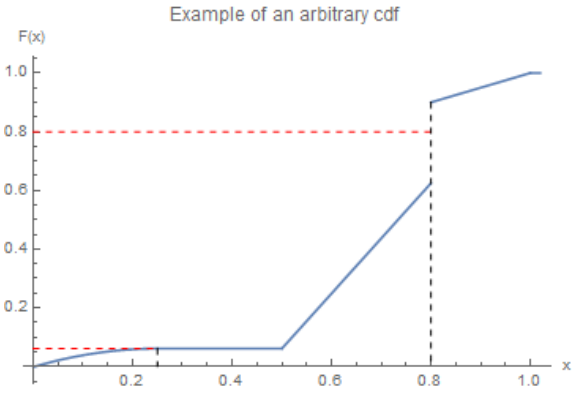
Furthermore, we can conclude on this basis that $F(F^{-1}(u)) \geq u$. Remember that the quantile function $F^{-1}(u)$ returns the minimum value of all x, for which the corresponding value of the cdf $F(x)$ is **greater than or equal to** u. Plugging the obtained x back into the cdf will thus logically deliver a value that is **at least** as big as the original u, so indeed $F(F^{-1}(u)) \geq u$. In this case, we have a strict inequality, if the cdf is discontinuous, i.e. if there is a jump in the cdf. Suppose the cdf jumps from u_0 to u_1 at x_0. Then for $u_0 < u' < u_1$, we have $F^{-1}(u') = x_0$. However $F(F^{-1}(u')) = F(x_0) = u_1 > u'$. This can be further illustrated by the below graph (upper horizontal red line), where $x_0 = 0.8, u_0 = 0.6$ and $u_1 = 0.9$.

Finally, we also conclude that $F^{-1}(F(x)) \leq x$ using the following simple line of reasoning:

$$F^{-1}(F(x)) = \inf\{y : F(y) \geq F(x)\} \quad \text{and } x \in \{y : F(y) \geq F(x)\} \Rightarrow F^{-1}(F(x)) \leq x.$$

In this case, we have strict inequality, in case of flat regions in the cdf, i.e. in case there are regions, which have density 0. Suppose the cdf is flat between x_0 and x_1, i.e. $F(x') = u_0$ for $x_0 < x' < x_1$. Consequently, we then have $F^{-1}(F(x)) = F^{-1}(u_0) = x_0 < x'$, since the quantile function is defined as the infimum of this interval between x_0 and x_1, which is x_0 of course. This can also be illustrated by the below graph (lower

horizontal red line), where $u_0 = 0.061, x_0 = 0.25$ and $x_1 = 0.5$.



53 Exercise 58

For the proof of the claim that the probability integral transform $F(X)$ of a random variable X with continuous cdf F is uniformly distributed on $[0, 1]$, we can use the result from exercise 57, where we showed that

$$(*) \quad F^{-1}(u) \leq x \iff F(x) \geq u.$$

We start by defining random variable Y by the probability integral transform, i.e. $Y := F(X)$. We compute the cdf of Y, denoted by $G(y)$, based on its definition:

$$\begin{aligned} G(y) &= P(Y \leq y) = P(F(X) \leq y) = \\ &\quad \text{applying result } (*) \text{ from above:} \\ &= P(X \leq F^{-1}(y)) = F(F^{-1}(y)) = y \end{aligned}$$

Hence, we have found that the cdf of random variable Y is given by $G(y) = y$, i.e. by the identity in y. Automatically, by the definition of probabilities and the cdf, we know that this cdf G is just the cdf of a uniform random variable on the interval $[0, 1]$, from which we conclude that Y is indeed uniformly distributed on [0,1].

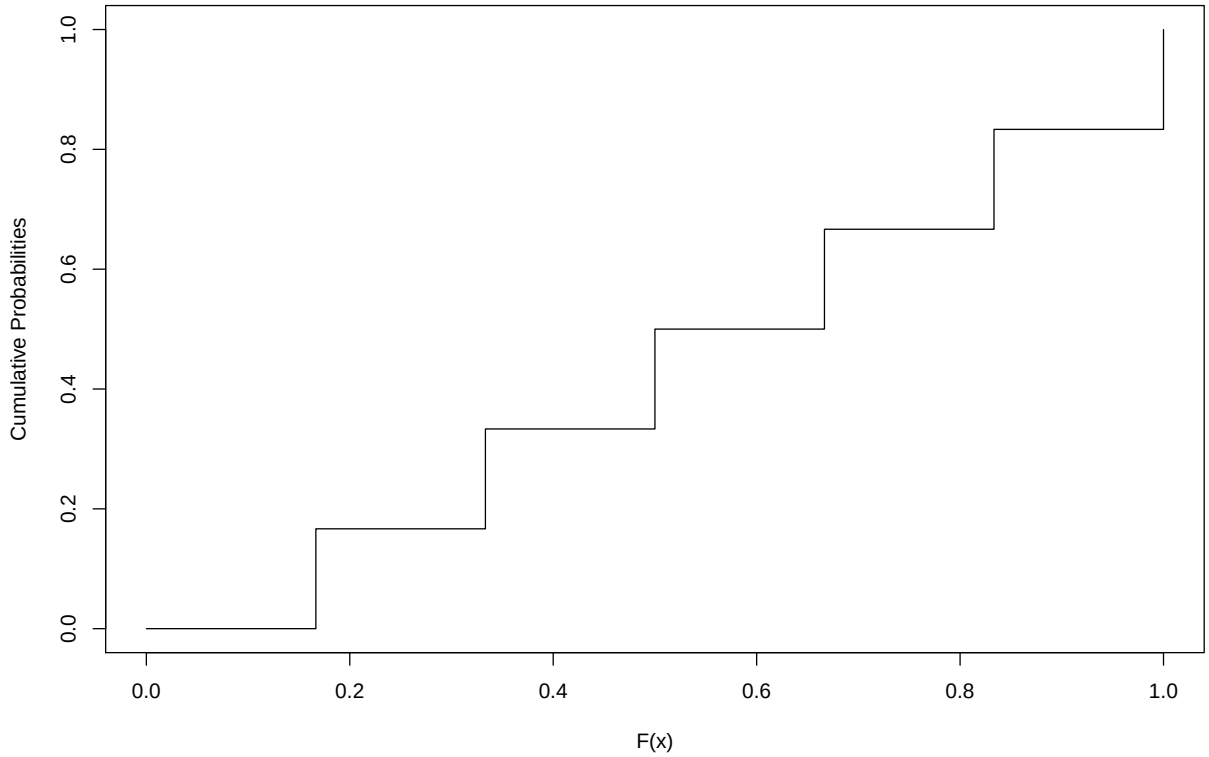
54 Exercise 59

Note that this task extends insights from the previous exercise to the discrete case, i.e. for a random variable X which attains the values $x_1 < ... < x_n$ with probabilities $p_1, ..., p_n$. First, let us characterise the cdf of X, given by $F(x)$:

$$F(x) = P(X \leq x) = \begin{cases} 0 & x < x_1 \\ p_1 & x_1 \leq x < x_2 \\ p_1 + p_2 & x_2 \leq x < x_3 \\ \vdots & \vdots \\ \sum_{i=1}^{n} p_i = 1 & x \geq x_n \end{cases}$$

As can be seen from this and from the general logic of discrete random variables, $F(x)$ takes at most countably different values in the interval $[0, 1]$ for a discrete random variable X. More specifically, these values are given by the **cumulative** discrete probabilities $p_1, p_2, ...$ Logically, the respective probability of each of these values attained by $F(x)$ are given by the relative lengths of the intervals between two x's (i.e. relative in relation to the whole range of attainable values $n = x_n - x_1 + 1$), in which $F(x)$ attains exactly this value. This is best illustrated with an easy example: Let X be the number in the roll of a fair Laplace die. Then $n = 6$ and $F(x)$ attains the values $1/6, 2/6, ..., 6/6$. The $x's$ are all equidistant from one another, so the relative length

of each interval is given by $1/n = 1/6$. The cdf of $F(x)$ will thus be a step function that has jumps of height $1/6$ at $x = i/6, \quad i = 1, 2, ..., 6$:



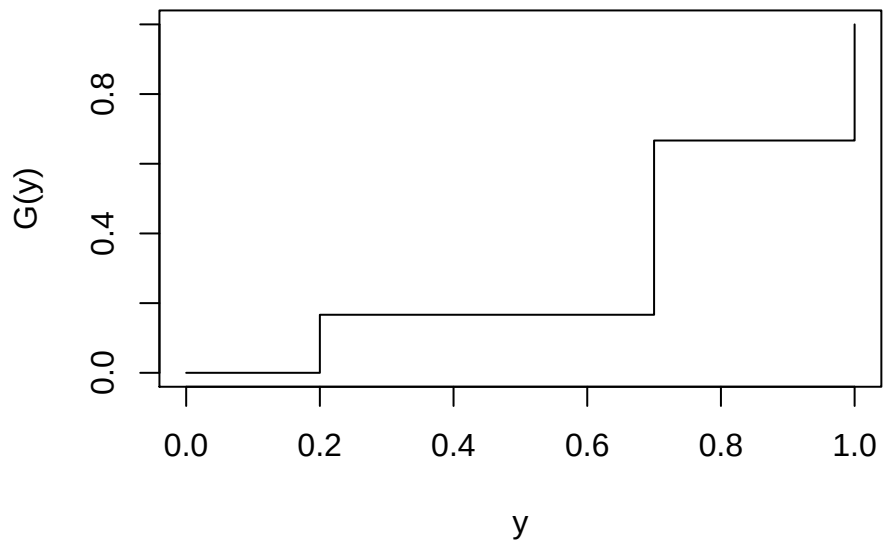
Let's generalise this. Let random variable Y be defined by $Y := F(x)$. By the above arguments, Y will also be discrete, since it will attain at most countably many different values. The cdf of Y, given by $G(y)$, can be characterised by the following step-wise function:

$$G(y) = P(Y \leq y) = \begin{cases} 0 & y < p_1 \\ \frac{x_2 - x_1}{n} & p_1 \leq y < p_1 + p_2 \\ \frac{x_2 - x_1}{n} + \frac{x_3 - x_2}{n} & p_1 + p_2 \leq y < p_1 + p_2 + p_3 \\ \vdots & \vdots \\ \sum_{i=1}^{n-1} \frac{x_{i+1} - x_i}{n} = \frac{x_n - x_1}{n} & \sum_{i=1}^{n-1} p_i = 1 \leq y < \sum_{i=1}^{n} p_i = 1 \\ \sum_{i=1}^{n-1} \left(\frac{x_{i+1} - x_i}{n} \right) + \frac{1}{n} = \frac{x_n - x_1 + 1}{n} = 1 & y \geq \sum_{i=1}^{n} p_i = 1 \end{cases}$$

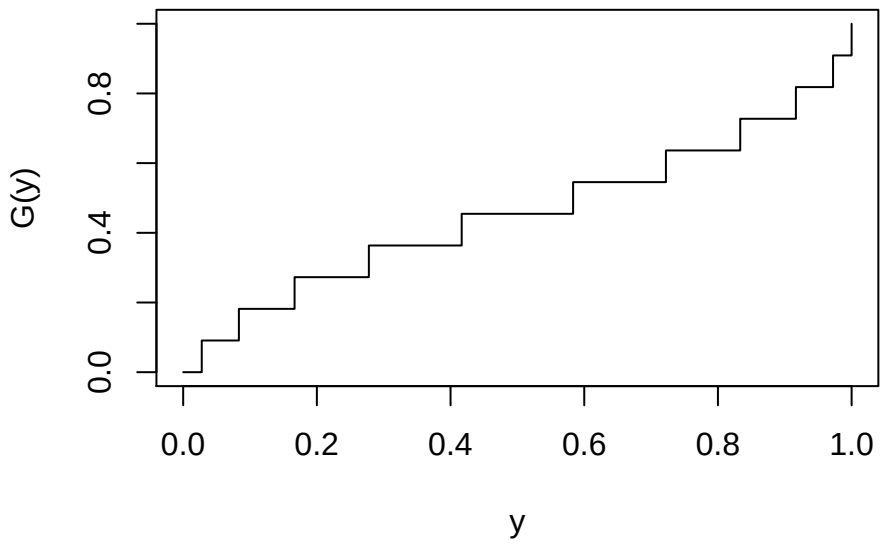
Once again, note that the jump heights of this discontinuous step function $G(y)$ (cdf of $F(x)$) will be given by the lengths of the corresponding intervals between x_i and x_{i+1}, while the step "widths" are determined based on the probabilities of individual x's. We also created an R function that shows that this procedure works by plotting the corresponding cdf of $F(x)$, given by $G(y)$, and also returning the corresponding cumulative probabilities.

```
ex_59<-function(sequen_x,sequen_p){
  n<-length(sequen_x)
  cdf<-c(0,cumsum(sequen_p))
  vec<-numeric(n)
  for(i in 1:(n-1)){
    vec[i]<-(sequen_x[i+1]-sequen_x[i])/n
```

```
  }
  vec[n]<-1/n
  vec<-c(0,vec)
  plot(cdf,cumsum(vec),type="s",ylim=c(0,1),xlim=c(0,1),xlab="y",ylab="G(y)")
  return(cumsum(vec))
}

#Random example
seq_x<-c(-1,-0.5,1)
seq_p<-c(0.2,0.5,0.3)
ex_59(seq_x,seq_p)
```




```
## [1] 0.0000000 0.1666667 0.6666667 1.0000000

#Two Laplace dice example
probs<-c(1,2,3,4,5,6,5,4,3,2,1)/36
ex_59(2:12,probs)
```




```
##  [1] 0.00000000 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455 0.63636364
##  [9] 0.72727273 0.81818182 0.90909091 1.00000000
```

One final informal remark on this: one can also see how $G(y)$ will converge to a continuous uniform distribution over $[0, 1]$, if the underlying $F(x)$ is also continuous. Decreasing the probabilities of the individual x_i towards zero, letting $n \to \infty$ (uncountable) and thereby finally also decreasing the length of the intervals between two different x's towards zero, will gradually shrink the jump heights of the cdf of $F(x)$ (i.e. $G(y)$) as well as the the step widths towards zero, so that the step function, for which we displayed various examples above, will gradually converge to the linear identity in y, i.e. $G(y) = y$ as was shown in exercise 58.

55 Exercise 60

Let $X \sim \text{Pareto}(a, b)$, then its density $f(x)$ and its cdf $F(x)$ are given by:

$$f(x) = \begin{cases} \frac{ab^a}{x^{a+1}} & x \geq b \\ 0 & \text{else} \end{cases} \qquad F(x) = 1 - (b/x)^a, \text{ with } x \geq b > 0, a > 0 \text{ for both.}$$

To apply the inverse transform method, we set $F(x) = u$ and solve for x:

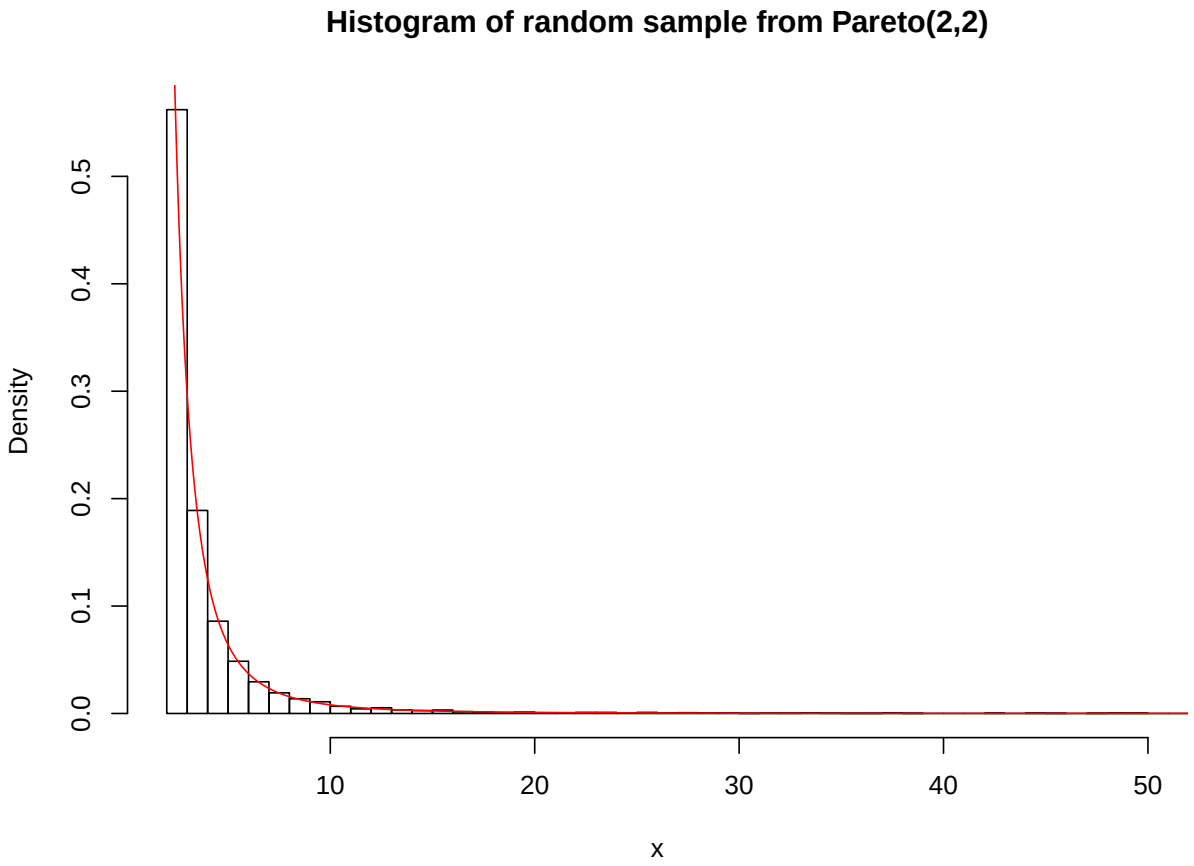
$$1 - \left(\frac{b}{x}\right)^a = u \iff \left(\frac{b}{x}\right)^a = 1 - u \iff \log\left(\frac{b}{x}\right) = \frac{1}{a}\log(1-u)$$
$$\iff \log(x) = \log(b) - \frac{1}{a}\log(1-u) \Rightarrow x = F^{-1}(u) = \exp\left(\log(b) - \frac{1}{a}\log(1-u)\right)$$

Since if $(1 - U) \sim \text{Uniform}(0, 1)$, then we also have $U \sim \text{Uniform}(0, 1)$. So, as in class, we can derive the following theorem: Let $U \sim \text{Uniform}(0, 1)$, then:

$$\exp\left(\log(b) - \frac{1}{a}\log(U)\right) \sim \text{Pareto}(a, b).$$

We use this result to generate a random sample from a Pareto(2,2) distribution. In the following R code, we generate this sample using the inverse transform method, graph the histogram of this sample with the corresponding density superimposed. Since the Pareto distribution is a strongly sweked distribution, we limit the histogram to display only the range from $x = 2$ to $x = 50$, although the random sample usually contains values greater than 50.

```
pareto<-function(u,a,b){
  exp(log(b)-(log(u)/a))
}
#Applying the inverse transform method:
uni<-runif(10000)
par_1<-pareto(uni,2,2)
#Note that the population mean is 2*(2/1)=4
mean(par_1) #sufficiently close

## [1] 3.996046

#Next, we set up the density function:
dpareto<-function(a,b,x){
  ifelse(x>=b,(a*b^a)/(x^(a+1)),0)
}
#We generate points, over which to plot this density:
x_values<-c(seq(2,20,by=0.05),20:floor(max(par_1)))
y_values<-dpareto(2,2,x_values)

#Finally, we plot the histogram with the density:
hist(par_1,freq=FALSE,breaks=200,xlim=c(2,50),
     main="Histogram of random sample from Pareto(2,2)",xlab="x")
lines(x_values,y_values,col="red")
```



56 Exercise 61

The generalised Pareto distribution has cdf

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

for in the support of this distribution, where $\mu \in \mathbb{R}$ is the location parameter, $\sigma > 0$ is the scale parameter, and $\xi \in \mathbb{R}$ is the shape parameter.

Remember that one of the definitions of Euler's number e is:

$$e = \lim_{n\to 0}(1+n)^{\frac{1}{n}} \quad \Longrightarrow \quad \lim_{n\to 0}(1+x\cdot n)^{-\frac{1}{n}} = e^{-x}$$

Hence, we write the above cdf of the generalised Pareto distribution as

$$F(x) = 1-\left(1+\frac{x-\mu}{\sigma}\cdot\xi\right)^{-\frac{1}{\xi}} \Rightarrow \quad \lim_{\xi\to 0} F(x) = \lim_{\xi\to 0}\left[1-\left(1+\frac{x-\mu}{\sigma}\cdot\xi\right)^{-\frac{1}{\xi}}\right] = 1-e^{-\frac{x-\mu}{\sigma}} = 1-\exp(-(x-\mu)/\sigma)$$

To establish the support of this distribution for the cases where $\xi \geq 0$ and $\xi < 0$, respectively, first note that $F(x) = 0$, if and only if $x = \mu$. Note that for $x < \mu$, we have $F(x) < 0$, regardless of whether $\xi \geq 0$ or $\xi < 0$. In order to verify this, let $x < \mu$ and examine the expression:

$$1-\left(1+\frac{x-\mu}{\sigma}\cdot\xi\right)^{-\frac{1}{\xi}}$$

For $\xi \geq 0$, the expression in the brackets will be smaller than 1, which - taken to the power of something negative, namely $-1/\xi$ - will deliver something greater than 1. Deducting the result from 1 will therefore deliver something negative.
For $\xi < 0$, the expression in the brackets will be greater than 1 already, which will not change when raising it to the power of something positive, namely $-1/\xi$. Deducting this result from 1 will thus also deliver something negative.
Since we can use the same arguments to show that $F(x) > 0$ for $x > \mu$, we now know that μ will be the lower bound (included) of the support of this distribution, regardless of whether $\xi \geq 0$ or $\xi < 0$. What is left to show is the upper bound of the support.
First, let $\xi \geq 0$. Note that in this case:

$$\lim_{x \to \infty} \left[1 - \left(1 + \frac{x-\mu}{\sigma} \cdot \xi \right)^{-\frac{1}{\xi}} \right] = 1 - 0^{+} = 1$$

That is, $F(x)$ continuously approaches 1 asymptotically from below in this case. Consequently, we find that the support of the distribution for $\xi \geq 0$ is indeed given by the interval $[\mu, \infty)$.
Finally, let $\xi < 0$ and reconsider the expression

$$1 - \left(1 + \frac{x-\mu}{\sigma} \cdot \xi \right)^{-\frac{1}{\xi}}$$

Note that in this case, the exponent $-1/\xi$ is positive, so the expression $(1 + \xi(x-\mu)/\sigma)^{-1/\xi}$ is just a usual transformed power function decreasing in x due to the negative multiplied factor ξ. Since - as was already determined - $F(x) = 0$, i.e. $(1 + \xi(x-\mu)/\sigma) = 1$, iff $x = \mu$, we thus just have to find the point at which $F(x) = 1$, i.e. $(1 + \xi(x-\mu)/\sigma) = 0$, to find the entire support of the distribution.

$$1 + \frac{x-\mu}{\sigma} \cdot \xi = 0 \iff \frac{x-\mu}{\sigma} \cdot \xi = -1 \iff x = \mu - \frac{\sigma}{\xi}$$

Since $F(x)$ is continuous in its domain and increasing around $x = \mu$ (remember that $F(x) < 0$ left of μ and $F(x) > 0$ right of μ), we have determined that the upper bound of the support of the distribution will therefore be $\mu - \sigma/\xi$. Note that for specific choices of ξ, the function $F(x)$ is not even defined in the real numbers beyond this point, since it will involve computing the square root of a negative number, as $(1 + \xi(x-\mu)/\sigma) < 0$ for $x > \mu - \sigma/\xi$. For this, consider the example of ξ being an even (negative) integer, for instance. In any case, we have thus finally found that the support of the distribution for $\xi < 0$ is indeed given by the interval $[\mu, \mu - \sigma/\xi]$.

57 Exercise 62

We derive the inverse function of the distribution function(or the quantile function):

$$F(x) = u = 1 - (1 + \xi(x-\mu)/\sigma)^{-1/\xi}$$

$$x = F^{-1}(u) = \frac{\frac{1}{1-u}^{\xi} - 1}{\xi} * \sigma + \mu$$

```
rgenpareto<-function(n,xi,sigma,mu) {
  if(sigma<=0) return("Incorrect sigma parameter!")
  singlenumber <- function(xi, sigma, mu) {
   u<-runif(1)
   x<-(1/(1-u)^xi-1)/xi*sigma+mu
   if((xi>=0 & x>=mu)|(xi<0 & x<=mu-sigma/xi)) return(x)
   else return("Computational error!")
  }
  replicate(n,singlenumber(xi, sigma, mu))
  }
rgenpareto(100,2,1,0)
```

```
##   [1] 1.548845e+02 5.585065e-01 8.720234e+00 9.833218e-01 2.348990e-01 9.375721e-01 1.013538e+01
##   [8] 1.734192e+00 3.284421e-01 2.821227e+00 3.069195e-01 3.893634e+02 7.309808e-02 2.773133e-01
##  [15] 2.797653e+00 1.471092e+03 2.183863e+00 2.284252e+00 4.743340e-01 1.981263e+00 6.061460e+00
##  [22] 4.006873e-01 9.956637e-02 4.873981e+00 1.148125e+00 2.969762e-01 1.974387e-01 4.902597e+01
##  [29] 3.039318e+00 1.248732e+00 5.500636e+01 5.045787e-01 2.028016e-01 1.239312e+00 1.299837e+00
##  [36] 2.475736e-01 1.078237e+00 7.081970e-01 2.401031e+00 1.713218e+02 2.917983e-01 2.383219e-02
##  [43] 1.061147e+01 5.949017e-01 2.028638e+01 6.744118e+01 1.571367e-01 1.758279e-01 1.960307e+00
##  [50] 3.994532e+00 4.866445e-01 5.038651e-02 1.526290e-01 2.889062e+01 3.854809e+01 2.623104e+01
##  [57] 2.208122e-01 2.012752e+00 1.391105e+00 5.595496e-01 1.652644e+03 7.133022e-01 3.190874e-01
##  [64] 1.009680e+01 7.598147e-01 1.844641e+00 3.398962e-01 7.416139e+01 2.326073e-01 7.456753e-03
##  [71] 1.311700e-01 1.779096e+01 2.107255e+01 3.854380e-01 1.466697e+00 3.616559e+00 2.648841e-01
##  [78] 1.916549e+02 7.455646e-03 5.271500e+02 7.445851e-01 1.588221e-01 5.621252e+00 2.938364e+00
##  [85] 7.835397e-02 6.122018e-02 1.811447e-01 1.555576e-01 1.763603e-01 7.019843e+00 1.078099e+00
##  [92] 3.672664e+03 2.036066e+00 2.982172e+00 7.472663e-01 1.216707e-01 6.464006e+00 9.519079e-01
##  [99] 5.395151e-02 1.212040e+00

rgenpareto(100,-2,1,0)

##   [1] 0.1009989015 0.4831761393 0.4905976956 0.4702808780 0.2626927323 0.4965180071 0.1472607993
##   [8] 0.2435221101 0.2011125788 0.0207815494 0.4925484107 0.4026312620 0.3737795551 0.4793067476
##  [15] 0.2528328202 0.4613060437 0.3286882491 0.3043370379 0.4983327524 0.3498980295 0.4428167138
##  [22] 0.1750166453 0.1788498187 0.1394698270 0.4965060720 0.2059654344 0.3855185749 0.4423365487
##  [29] 0.1642707607 0.4784241436 0.4993256799 0.0009090195 0.3533340013 0.4536911398 0.2486673881
##  [36] 0.4970529695 0.4177918295 0.1035393150 0.4679706150 0.3038738986 0.3407962023 0.4900986764
##  [43] 0.3081597943 0.2978840889 0.4046876705 0.4924390045 0.4891579840 0.3316281128 0.0726567321
##  [50] 0.4982198453 0.0970943156 0.3158283645 0.3184719853 0.4555408220 0.2943072277 0.4969669071
##  [57] 0.3717776861 0.3239364505 0.4946575639 0.4782875984 0.1676837639 0.3395406136 0.4380977040
##  [64] 0.1680378935 0.0618588261 0.1571069444 0.4731097953 0.4717827096 0.3739908004 0.2724183906
##  [71] 0.2591148268 0.3772128399 0.4795930051 0.4491571218 0.4103302147 0.4901319483 0.3843086992
##  [78] 0.4783148075 0.1860876023 0.4526225284 0.4978955113 0.2151999453 0.4716343561 0.3817037258
##  [85] 0.1676134489 0.0474433516 0.4141846662 0.4558830096 0.3351355032 0.3281369407 0.0096822080
##  [92] 0.4975624701 0.0844376255 0.4626643293 0.3755024254 0.4365711935 0.2138383086 0.4596279230
##  [99] 0.2910396630 0.4469531881

rgenpareto(100,2,0.1,200)

##   [1]    215.2947    200.2394    200.0209    207.7127    200.5413    200.4792    204.4964
##   [8]    200.0300    200.2374    200.0148    200.0111    200.1334    201.0687    200.2986
##  [15]    200.0192    200.0336    200.1127    200.0524    200.0084    200.0704    200.1241
##  [22]    200.1359    200.0088    200.2177    207.9700    200.3383    200.4893    222.0803
##  [29]    200.0014    200.0444    200.0260    200.1087    200.0320    200.0628    200.8724
##  [36]    200.2699    200.0310    200.0396    200.1284    200.5281    200.2063    200.2640
##  [43]    200.0418    200.1365    200.0141    200.0003    204.3880    200.8916    200.0026
##  [50]    200.0184    200.7346    200.3532    200.0610    200.1824    200.0066    200.0321
##  [57]    675.0445    201.8108    201.1784    200.6661    200.2852    200.9431    203.7350
##  [64]    200.0285    200.2836    200.2137    200.2617    200.0721    201.2520    200.0220
##  [71]    201.8593    228.2466    210.6513    200.1026   8353.7647    200.0436    200.0029
##  [78]    200.3394    200.2953    200.3971    200.2386    200.5061    200.0148    200.6949
##  [85]    200.3231    200.0415    200.0002    201.2589    200.4912 931968.7547    200.3233
##  [92]    204.7209    200.0785    200.1850    200.1032    200.3072    200.0487    200.1265
##  [99]    200.6044    200.0085

rgenpareto(100,2,0.5,1.5)

##   [1]    14.937850     9.364833     3.556138     1.916169     1.563095     1.710489     5.120265
##   [8]     1.880023     1.870423     4.152180     1.640770     1.685615     2.019831    10.992029
##  [15]     1.950830     1.726704     1.567448     1.707151     3.450970     3.280805     1.961086
##  [22]     3.775663     2.028477     2.731329    18.343443     1.684184     1.616800     1.735363
##  [29]     1.960021     1.757794     6.410600     1.574729     6.917233     2.392916     2.160892
##  [36]     1.650184     4.542225     4.773082   602.422179     2.286248     2.199378   200.167991
##  [43]     1.548021     2.286430   349.670499     1.582594     1.535402   309.569299     1.776878
##  [50]     2.395793    50.190732     1.547451     2.791749    77.257630     1.698078     6.690292
##  [57]     1.891272     6.795825     1.776325    32.039774     2.328178     1.548916     1.907250
##  [64]     2.901482  5632.472162     1.657585    72.191918     1.543762  1801.083229     1.991964
##  [71]     3.883875     1.904734     4.686220     4.115579     1.520756     2.085359     1.567222
##  [78]     2.770860     1.582024     2.388122     1.874044     2.086442     1.636873     1.590640
##  [85] 33638.437347     1.510392     1.637391     1.557854     3.024080     2.629862     1.745460
##  [92]     1.722482     1.674381   411.207238     2.723603     2.499602     2.196263  9666.125640
##  [99]     4.830050     1.516024
```

58 Exercise 63

```
rdiscretebins<-function(n) {
  singlenumber<-function() {
    u<-runif(1)
    .bincode(u,c(0,0.1,0.3,0.5,0.7,1),right = FALSE, include.lowest = TRUE)-1
  }
  replicate(n,singlenumber())
}
x<-rdiscretebins(1000)
freqs<-as.data.frame(table(x)/1000)
freqs['Theo']<-c(0.1,0.2,0.2,0.2,0.3)
freqs

##   x  Freq Theo
## 1 0 0.100  0.1
## 2 1 0.191  0.2
## 3 2 0.197  0.2
## 4 3 0.211  0.2
## 5 4 0.301  0.3

y<-sample(c(0,1,2,3,4),1000,replace = TRUE,prob = c(0.1,0.2,0.2,0.2,0.3))
freqs2<-as.data.frame(table(y)/1000)
freqs2['Theo']<-c(0.1,0.2,0.2,0.2,0.3)
freqs2

##   y  Freq Theo
## 1 0 0.106  0.1
## 2 1 0.189  0.2
## 3 2 0.215  0.2
## 4 3 0.195  0.2
## 5 4 0.295  0.3
```

59 Exercise 66

By definition, the chi-squared distribution (also χ^2-distribution) with n degrees of freedom is the distribution of a sum of the squares of n independent standard normal random variables. We simulate it in two different ways: first, we will use this sum of the squares $n = 8$ independent standard normal random variables, and then we will use the R function rchisq(), to see if we obtain comparable results:

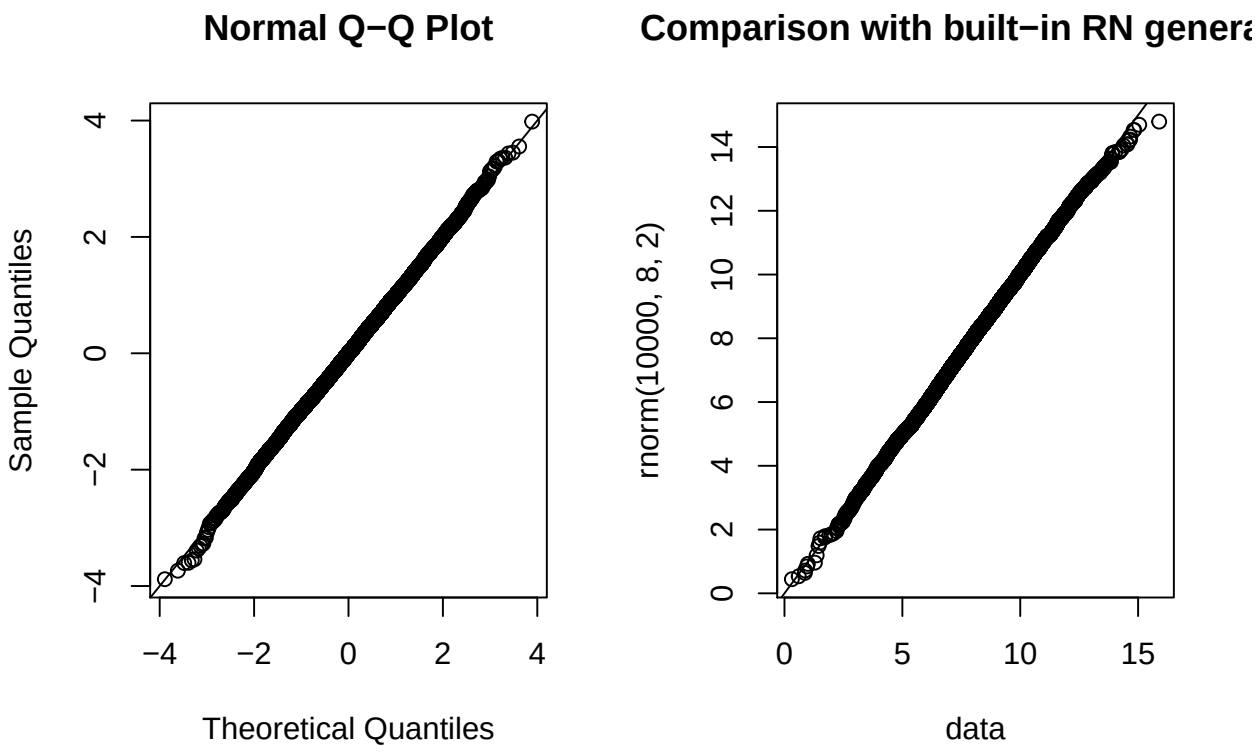
```
N<-100000
n<-8
normal_rvs<-replicate(n,rnorm(N)^2)
chi_normal<-rowSums(normal_rvs)
mean(chi_normal)

## [1] 7.997726

var(chi_normal)

## [1] 15.93186

chi_sim<-rchisq(N,n)
mean(chi_sim)
```

```
## [1] 8.001025

var(chi_sim)

## [1] 15.99099
```

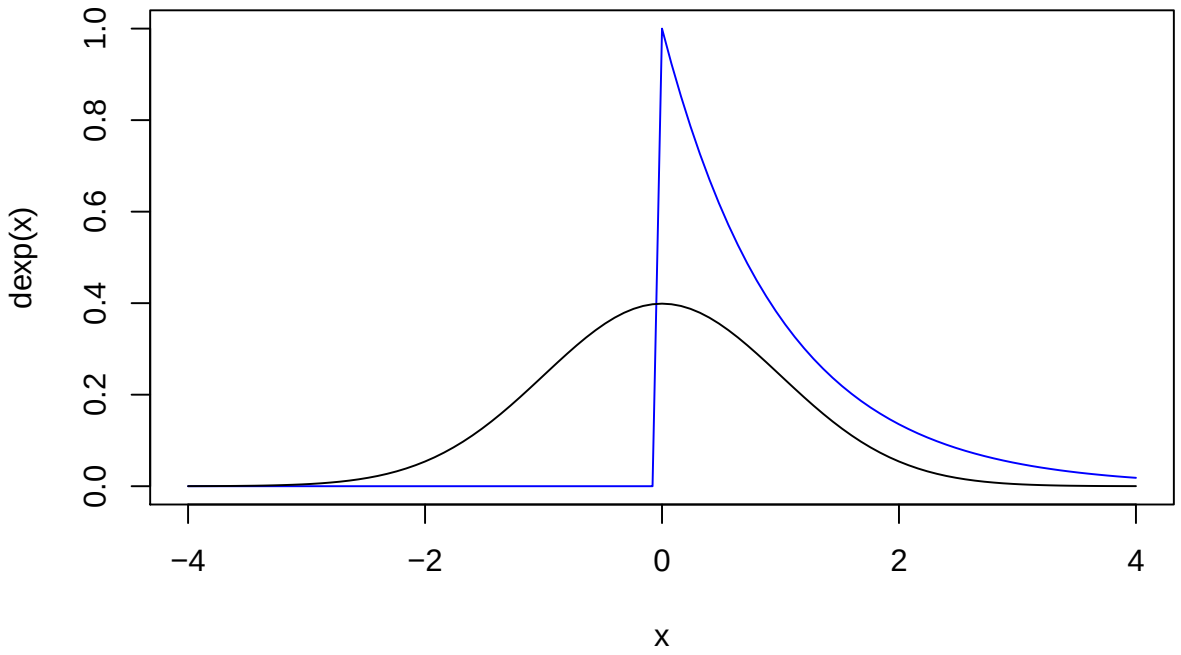
We see that in both cases, the estimated mean and variances are reasonably close to their true values of 8 and 16, although - based on the law of large numbers - we would expect this differences to true values to become smaller and smaller with an increasing simulation sample size, of course.

60 Exercise 67

```
rannorm <- function(n, mean = 0, sd = 1){
singlenumber <- function() {
repeat {
U <- runif(1)
U2 <- sign(runif(1, min = -1)) # value is +/- 1.
Y <- rexp(1) * U2 # Y is a double exponental r.v.
if (U < dnorm(Y) / exp(-abs(Y))) break
}
Y
}
replicate(n, singlenumber()) * sd + mean
}

##--a
data<-rannorm(10000,8,2)
mean(data)

## [1] 8.005267

sd(data)

## [1] 1.980889

##--b
data_normal<-(data-mean(data))/sd(data)
par(mfrow=c(1,2))
#QQ plot after normalizing to standard
qqnorm(data_normal)
abline(0,1)
##QQ plot compared with built-in random numbers from normal distribution
qqplot(data,rnorm(10000,8,2),main="Comparison with built-in RN generator")
abline(0,1)
```




```
##--c
par(mfrow=c(1,1))
curve(dexp,from=-4,to=4,col="blue")
curve(dnorm,from=-4,to=4,add=TRUE)
```



It is obvious that dnorm is less than dexp for all $x > 0$. In addition, note that the variable $U2$ creates those points left of zero (which are not visible in the plot) and thus basically forces the function dexp ($f(x)$) to be symmetric. From this, it follows that clearly $Uf(x) < kg(x)$ which is in this case equal to condition "if (U < dnorm(Y) / exp(-abs(Y)))". Hence, the rejection method has been implemented successfully.

61 Exercise 68

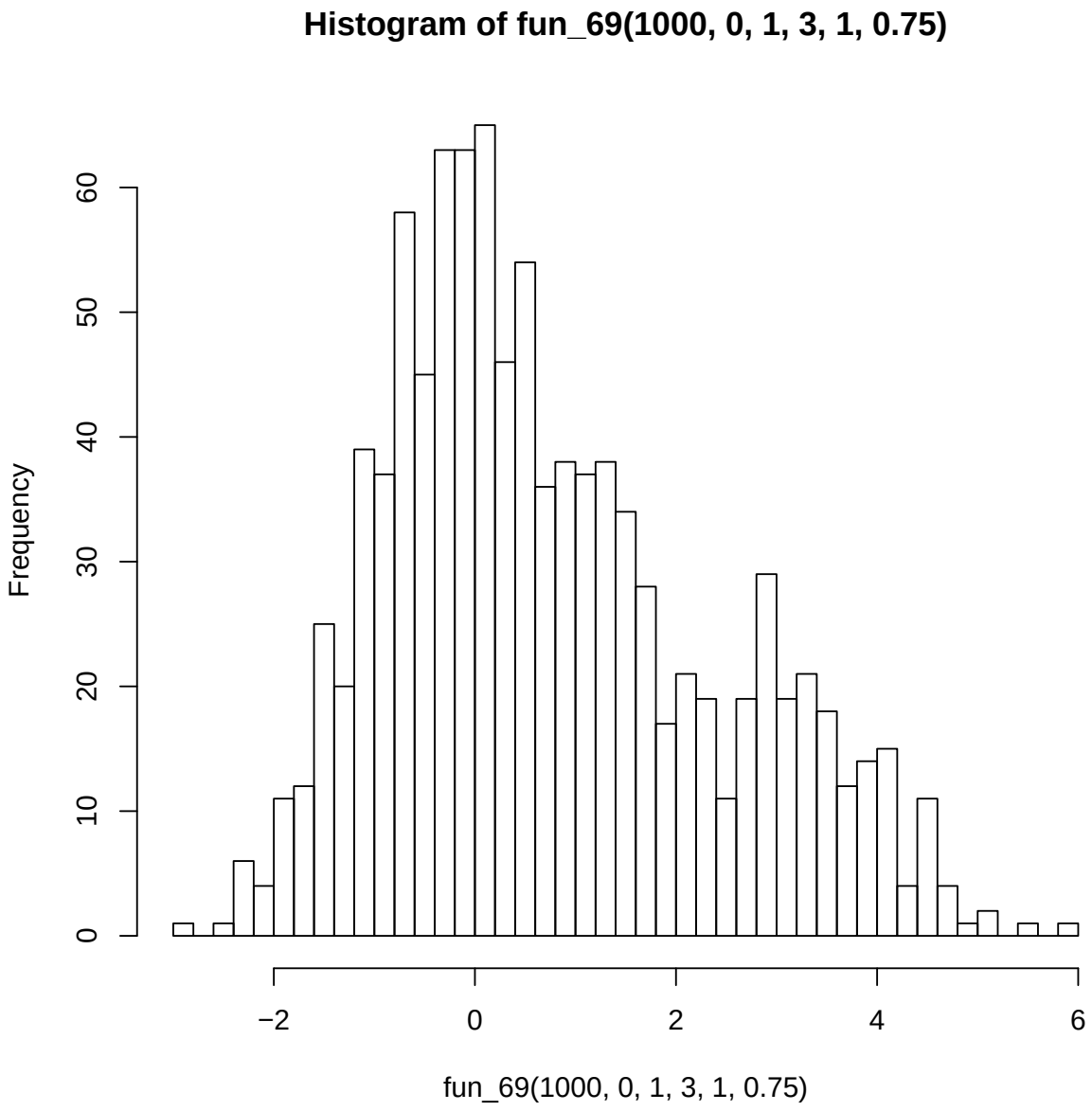
```
probs<-c(0.2,0.3,0.1,0.15,0.05,0.2)
randiscrete1<-function(n, probs) {
    cumprobs<-cumsum(probs)
    singlenumber<-function() {
           x<-runif(1)
           sum(x>cumprobs)
       }
       replicate(n,singlenumber())
   }

system.time(randiscrete1(100,probs))

##    user  system elapsed
##    0.02    0.00    0.01

system.time(randiscrete1(1000,probs))

##    user  system elapsed
##    0.02    0.00    0.02
```

```
system.time(randiscrete1(10000,probs))

##    user  system elapsed
##    0.01    0.00    0.01

randiscrete2<-function(n, probs) {
  singlenumber<-function(){
      repeat{
        U<-runif(2,
                 min=c(-0.5,0),
                 max=c(length(probs)-0.5,max(probs)))
        if(U[2]<probs[round(U[1])+1]) break
        }
        return(round(U[1]))
      }
    replicate(n,singlenumber())
}
system.time(randiscrete2(100,probs))

##    user  system elapsed
##       0       0       0

system.time(randiscrete2(1000,probs))

##    user  system elapsed
##    0.01    0.00    0.01

system.time(randiscrete2(10000,probs))

##    user  system elapsed
##    0.07    0.00    0.08
```

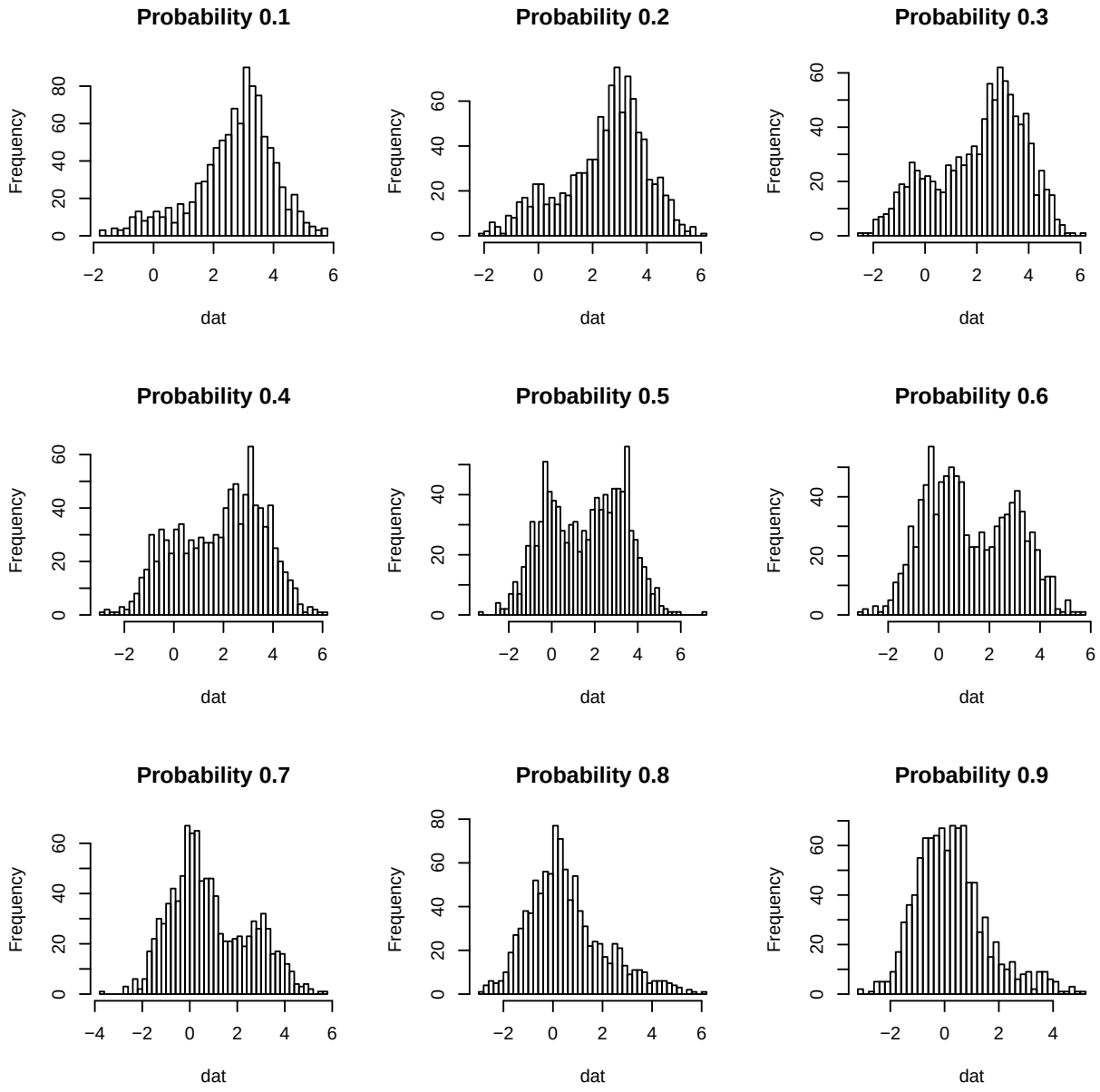
Although both methods work reasonably fast, the first one seems to be slightly faster.

62 Exercise 69

```
x<-rnorm(1000)
Y<-rnorm(1000,3,1)
W<-rbinom(1000,1,0.2)

fun_69<-function(n,mu_1,sd_1,mu_2,sd_2,p)
{
  x_1<-rnorm(n,mu_1,sd_1)
  x_2<-rnorm(n,mu_2,sd_2)
  w<-rbinom(n,1,p)
  ifelse(w==1,x_1,x_2)
}

hist(fun_69(1000,0,1,3,1,0.75),breaks=50)
```




```
par(mfrow=c(3,3))
for(i in seq(0.1,0.9,0.1)){
dat<-fun_69(1000,0,1,3,1,i)
hist(dat,breaks=50,main = paste("Probability",i))}
```




```
dev.off()

## null device
##           1
```

The location mixture distribution appears to be bimodal for values of p_1 being close to 0.5. On the contrary, for values of p_1 close to 0 and 1, the distribution is becoming gradually less bimodal.

63 Exercise 70

In order to simulate a mixture, we have to sample the state variable first, which is here the Λ varible of the rate parameter of the exponentially distributed Y. Λ follows a $Gamma(r, \beta)$ distribution, where r is the shape parameter and β is the rate parameter (or alternatively, $\frac{1}{\beta}$ is the scale parameter). Then, we obtain the

sample sequence $(\Lambda = \lambda_1, \Lambda = \lambda_2, ..., \Lambda = \lambda_n)$ and the conditional distributions from which we can sample the realisations of Y: $(Y|\Lambda = \lambda_1 \sim Exp(\lambda_1), Y|\Lambda = \lambda_2 \sim Exp(\lambda_2), ..., Y|\Lambda = \lambda_n) \sim Exp(\lambda_n)$.

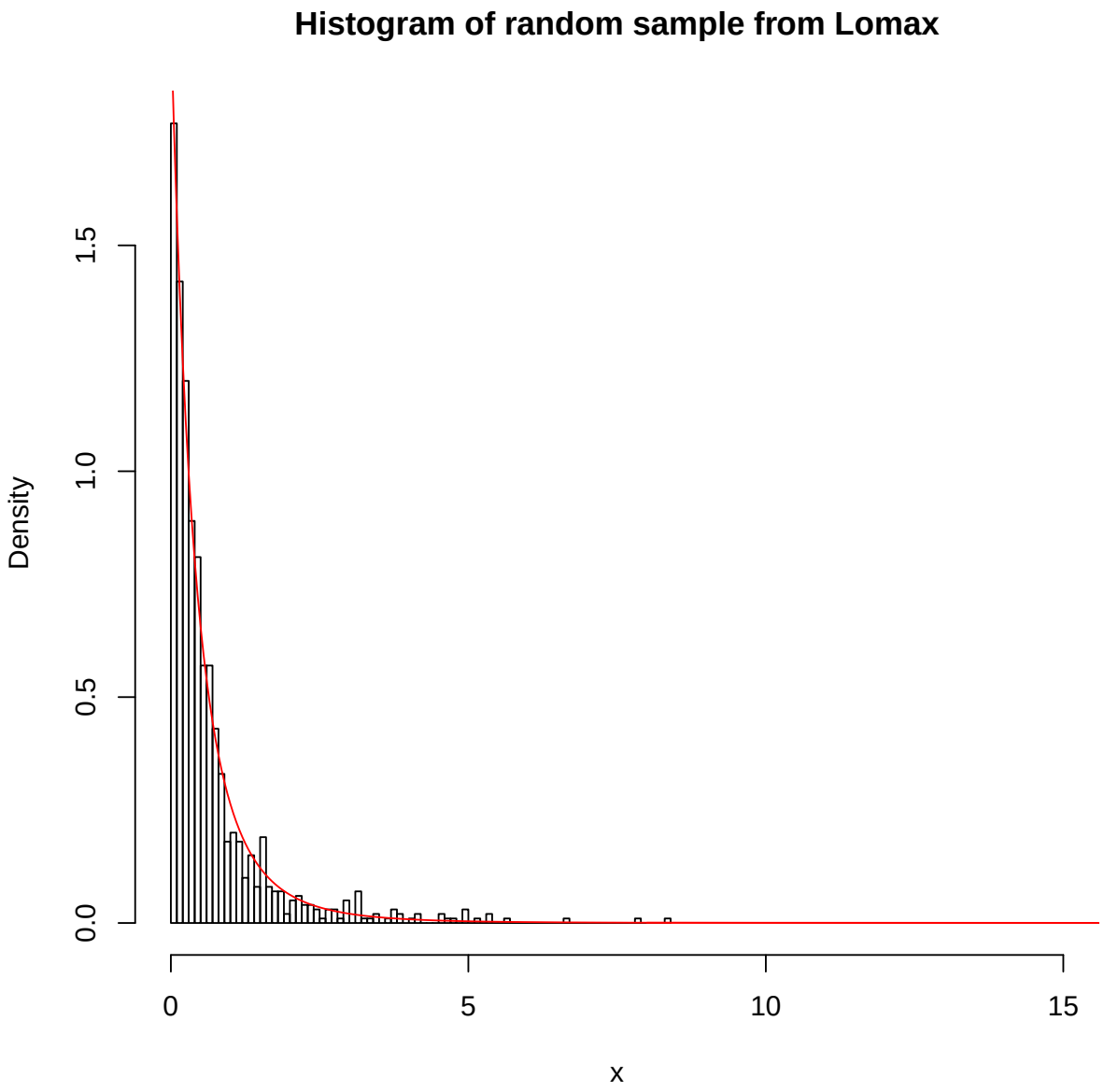
```
#First we determine the parameters
r<-4
beta<-2
n<-1000
#Then we simulate the lambda parameters of the conditional distributions
exp_rates<-rgamma(n,r,rate = beta)
#And we simulate the conditional distributions themselves
mix_obs<-rexp(n,rate=exp_rates)
```

We also need the calculate the density function of the theoretical Lomax-distribution. We obtain it by differentiating the cumulative distribution function:

$$F_{Lomax}(y) = 1 - (\frac{\beta}{\beta + y})^r$$

$$f_{Lomax}(y) = \frac{dF}{dy} = (-1) \cdot \beta^r \cdot (-r) \cdot (y + \beta)^{-r-1} = \frac{r\beta^r}{(\beta + y)^{r+1}}$$

```
#Next, we set up the density function:
dlomax<-function(shape,rate,y){
  ifelse(y>=0,(shape*rate^shape)/((rate+y)^(shape+1)),0)
}
#We generate points, over which to plot this density:
x_values<-c(seq(0,8,by=0.05),20:floor(max(mix_obs)))
y_values<-dlomax(r,beta,x_values)

#We then plot the histogram of the sample
hist(mix_obs,freq=FALSE,breaks=100,xlim=c(0,15),
     main="Histogram of random sample from Lomax",xlab="x")
#Finally, we superimpose the density over the histogram:
lines(x_values,y_values,col="red")
```



We see that the generated sample's histogram approximately fits the theoretical curve.

64 Exercise 71

The general multivariate Student t-distribution is characerized by 3 parameters. There are multiple conventions, here we use the $t(\nu, m, \Sigma)$ notation, where ν stands for the common degrees of freedom (the constituting marginal distributions all follow t-distribution with the same degrees of freedom) and m stands for the vector of the means of the marginal distributions corresponding the constituting random variables. In our solution, Σ will denote the covariance matrix of the constituting random variables, here $Cov(X_1, X_2)$.
One can show that (the source we here refer to is the website: `https://www.statlect.com/probability-distributions/multivariate-student-t-distribution`) the general bivariate Student's t (ν, m, Σ) random vector $(X_1, X_2)'$

is a linear function of a standard bivariate Student's t random vector $(Z_1, Z_2)'$ in the following way:

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \end{bmatrix} + W \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$$

where W is a (2×2) invertible matrix such that $V = WW' = W'W$, where V is the scale matrix of the bivariate t-distribution. Another identity to acquire the matrix V is the following:

$$\Sigma = \frac{\nu}{\nu - 2} V$$

. Consecutively, we can derive an algorithm how to generate such bivariate t-distributed random vectors. First, one has to create the scale matrix by transforming the (given) covariance matrix, then has to decompose it by using the Cholesky-decomposition, obtaining the W matrices. Then one has to generate two independent standard Student t-distributed random variables and apply the above formula to transform their vector to get a random vector following the desired general Student distribution.

Concerning the estimate of the ρ by inverting the sample Kendall's τ correlations, we have to apply the following transformation:

$$\rho_{pseudo} = sin(\frac{pi}{2}\rho_\tau(X_1, X_2))$$

```
#Give in desired standard deviations and expected values
stdev1<-0.8
stdev2<-1.2
m<-c(2,3.5)
nu<-3 #degrees of freedom required by the task
corr<-0.5 #correlation required by the task
amount<-180 #90 pairs of random variable realizations in one sample
#Then, we construct the covariance matrix
Sigma<-matrix(c( stdev1^2 , stdev1*stdev2*corr, stdev1*stdev2*corr , stdev2^2), 2)

#The following function creates one sample of bivariate Student distribution of size n/2.
#First it transforms the covariance matrix into the scale matrix (Cov_mod), then generates
#a sample of standard Student distributed numbers of size n, splits the sample into two
#rows (list_of_random_variables) and applies the proper transformation on each 2x1 vector
#to turn them into general Student t random vectors (matrix_of_bivariate_variables).
#It then measures the sample (Pearson) correlation and the sample Kendall's tau correlation.
#It transforms the Kendall's tau in the determined way above to get an alternative
#estimation of the underlying correlation. The function returns the two estimates in a vector.
single_t_sample<-function(df,n,mus,Cov) {
  Cov_mod<-Cov/(df/(df-2))
  list_of_random_variables<-split(rt(n,df),1:2)
  matrix_of_bivariate_variables<-mus+chol(Cov_mod)%*%rbind(list_of_random_variables$'1',
                                                           list_of_random_variables$'2')
  sample_corr<-cor(matrix_of_bivariate_variables[1,],matrix_of_bivariate_variables[2,])
  inverted_kendall_corr<-sin(cor(matrix_of_bivariate_variables[1,],matrix_of_bivariate_variables[2,],
                                 method = "kendall")*pi/2)
  return(c(sample_corr,inverted_kendall_corr))
}
#We replicate the above function 3000 times to obtain a matrix of sample correlation estimates
#(first row) and estimates by inverted Kendall's tau (second row).
big_sample<-replicate(3000,single_t_sample(nu,amount,m,Sigma))

mean(big_sample[1,])

## [1] 0.6000668
```

```
sd(big_sample[1,])

## [1] 0.1187526

mean(big_sample[2,])

## [1] 0.6099983

sd(big_sample[2,])

## [1] 0.07977909

#The results suggest that the inverted Kendall's tau estimate is more exact for the underlying
#correlation: its mean is closer to the real value and its standard deviation is smaller.
```

If we expand the dimension to $d > 2$, the Kendall's τ inversion estimation should still work, since the τ-s can be estimated pairwise for all constituting variables. From a Kendall's τ matrix, one can still pick any element and estimate a pseudo-correlation variable between two marginal distributions: $\tau_{i,j} = \frac{2}{pi} arcsin(\frac{\sigma_{ij}}{\sqrt{\sigma_{ii}}\sqrt{\sigma_{jj}}})$.
What could cause any complications in the estimation procedure? If we want to uphold the relatively large sample size (say, 3000*90), then the computational time of estimating the Kendall's τ correlations becomes inconveniently long as the number of dimensions is increased. We demonstrate our view with the following simple example.

We generate our known sample of size 90 of an i-dimensional standard Student random vector (here with independent marginal distributions), calculate the Kendall's τ matrix, then repeat the function of the single test 3000 times (that would enable us to estimate an expected value for the estimate). We measure the elapsed machine time and print these times under each other depending on the number of dimensions. We also print the number of dimensions and the $nlog(n)$ values where $n = d - 1$.

```
for(i in 3:10) {
  singletest<-function() {
  list_of_random_variables_test<-split(rt(90*i,3),1:i)
  testmatrix<-t(do.call("rbind", list_of_random_variables_test))
  cor(testmatrix, method = "kendall")
  }
  cat(i, system.time(replicate(3000,singletest()))["elapsed"], (i-1)*log((i-1)), "\n")
}

## 3 2.28 1.386294
## 4 3.48 3.295837
## 5 4.93 5.545177
## 6 6.69 8.04719
## 7 8.94 10.75056
## 8 11.16 13.62137
## 9 13.58 16.63553
## 10 16.43 19.77502
```

The time complexity approximately implies an $((d-1)log(d-1))$ or quasilinear running time which is definitely time consuming in case of large dimensions (was consuming more than 10 CPU time units in case of dimension 7).

65 Exercise 72

In our compound Poisson(λ)-Gamma process the lengths of time periods between 'jumps' come from an exponential distribution with parameter λ, $N(t)$ follows a Poisson(λt) distribution and the 'jump magnitudes'

are i.i.d. with a chosen Gamma distribution.
Next we show that the hints in the task hold. The expectation of the compound process is calculated using Wald's equation for separating the expected number of variables and the expectation of these variables. We know that since the Y_i are i.i.d., their expectations all equal the expectation of Y_1. We also know that the expected value of a variable $Z =^d Poisson(\lambda)$ is λ.

$$\mathbf{E}(X(t)) = \mathbf{E}(\sum_{i=1}^{N(t)} Y_i) = \mathbf{E}(N(t))\mathbf{E}(Y_1) = \lambda t \mathbf{E}(Y_1)$$

Next, we also show that the hint for the variance holds. Respectively, we use the law of total variance, then the variance of sum of independent variables and the conditional expectation, then the scalings of expectation and variance, then the expected value and the variance of a random varible following Poisson distribution (both are the parameter) and finally the definition of variance.

$$\begin{aligned}
&Var(X(t)) = \mathbf{E}(Var(X(t)|N(t))) + Var(\mathbf{E}(X(t)|N(t))) = \\
&\mathbf{E}(N(t))Var(Y_1) + Var(N(t)\mathbf{E}(Y_1)) = \\
&Var(Y_1)\mathbf{E}(N(t)) + \mathbf{E}(Y_1)^2 Var((N(t))) = \\
&Var(Y_1)\cdot \lambda t + \mathbf{E}(Y_1)^2 \cdot \lambda t = \lambda t(Var(Y_1) + \mathbf{E}(Y_1)^2) = \lambda t \mathbf{E}(Y_1^2)
\end{aligned}$$

If $Y_i =^d Gamma(\alpha, \beta)$, then $\mathbf{E}(Y_i) = \frac{\alpha}{\beta}$ and $\mathbf{E}(Y_1^2) = Var(Y_1) + \mathbf{E}(Y_1)^2 = \frac{\alpha}{\beta^2} + \frac{\alpha^2}{\beta^2}$. In our case, the simulations always run until $t = 10$ time.

```
scenarios<-10000 #the number of sample processes we want to use for parameter estimation
ttime<-10 #given t by the task

#First, we create a function for generating a single scenario of the compound Poisson
#process. It gets lambda as the exponential parameter, t (which here is always ttime),
#the Gamma parameters alpha and beta and a boolean deciding whether we want to plot the
#result of the process.
#The function simulates the Poisson-distributed number of jumps until time t, then draws
#a sample of this size from the uniform distribution supported by [0,t] and orders them.
#These random numbers will be the times of the jumps. We assign a jump magnitude to each
#jump time by generating a Gamma distributed random number (with desirable parameters).
#The function then returns the sum of the jump magnitudes as an output. It can also
#generate a plot where we represent the jump-process with a stair-type graph.
single_path<-function(lambda,t,alpha,beta,plotting=FALSE) {
  P<-rpois(1,lambda*t)
  event_times<-sort(runif(P, min = 0, max = t))
  magnitudes<-rgamma(P,alpha,rate=beta)
  x_values<-c(0,rep(event_times,each=2),ttime)
  y_values<-0*x_values
  for(i in 3:length(y_values)){
    if(i%%2==1){
      y_values[i]<-cumsum(magnitudes)[(i-1)/2]
    }else{y_values[i]<-y_values[i-1]}
}
      if(plotting==TRUE)
      {plot(x_values,y_values,type="l",
            main=paste0(c("Compound Poisson(",lambda,")-Gamma(",alpha,",",beta,"process"),collapse = ""),
            ylab="Amount",xlab="Time")}
      return(sum(magnitudes))
}

#We create another function that repeats the single sample generation scenario times by
#using the replicate method and also computes the sample mean and variance of the final
#cumulated jump amounts.
```

```
estimation<-function(size,l,tau,a,b) {
  m<-replicate(size,single_path(l,tau,a,b,plotting=FALSE))
  return(c(mean(m),var(m)))
}
#Here we compare some estimates with their theoretical values (using the formulas worked
#out previously).
estimation(scenarios,0.5,ttime,2,2)

## [1] 5.000008 7.437896

c(0.5*ttime*2/2,0.5*ttime*((2/2^2)+(2/2)^2))

## [1] 5.0 7.5

estimation(scenarios,2,ttime,2,2)

## [1] 20.00440 31.30184

c(2*ttime*2/2,2*ttime*((2/2^2)+(2/2)^2))

## [1] 20 30

estimation(scenarios,0.5,ttime,1,0.01)

## [1]    504.38 101900.29

c(0.5*ttime*1/0.01,0.5*ttime*((1/0.01^2)+(1/0.01)^2))

## [1] 5e+02 1e+05

estimation(scenarios,0.5,ttime,10,1)

## [1]  49.95371 557.80131

c(0.5*ttime*10/1,0.5*ttime*((10/1^2)+(10/1)^2))

## [1]  50 550

estimation(scenarios,8,ttime,5,10)

## [1] 40.03451 23.97550

c(8*ttime*5/10,8*ttime*((5/10^2)+(5/10)^2))

## [1] 40 24

#We understand that the obtained estimates are very close to their theoretical counterparts.
```

66 Exercise 73

We denote these five integrals by small Roman numerals and start by calculating the exact answers, which is known for the first three of these integrals:

$$\text{i)} \int_1^3 x^2 dx = \left[\frac{1}{3}x^3\right]_1^3 = 9 - \frac{1}{3} = 8\frac{2}{3}$$

$$\text{ii)} \int_0^\pi \sin(x)dx = [-\cos(x)]_0^\pi = -\cos(\pi) + \cos(0) = 1 + 1 = 2$$

$$\text{iii)} \int_0^\infty e^{-x} dx = \left[-e^{-x}\right]_0^\infty = 0 + 1 = 1$$

Up to six decimal digits of accuracy, the integrals iv) and v) are given by:

$$\text{iv)} \int_0^3 \sin(e^x)dx \approx 0.606124$$

$$\text{v)} \int_0^2 \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \approx 0.477250$$

Note that iii) is just the integral of the standard exponential distribution with parameter $\lambda = 1$ over its entire support from 0 to ∞, which will of course be 1. Note that v) is just the density of the standard normal distribution, so we are computing the probability of a standard normal random variable being an element of the interval $[0, 2]$.

For the computational implementation, we use the density of the uniform distribution as the density $f(x)$ for MC estimation of integrals i), ii), iv) and v). For iii), we use the density of the standard exponential distribution as $f(x)$ and then $g(x) = \lambda \cdot x = x$, since $\lambda = 1$ here. For v), we present a second alternative besides using the density of the uniform distribution over $[0, 2]$ as $f(x)$. That is, we use the density of the standard normal as $f(x)$, but since its support is the entire real line, we need to define $g(x)$ as an indicator function for the interval $[0, 2]$. Note that we lose more than 50% of the sample in this way, so this option is computationally less appealing.

```
#We write a function for the standard case of using the density of
#the uniform distribution as f(x):
mc_integral<-function(n,a,b,g){
  uni<-runif(n,a,b)
  interval<-(b-a)
  mean(interval*g(uni))
}
n<-1000000

int1<-function(x) x^2
int4<-function(x) sin(exp(x))
int5<-function(x) (1/sqrt(2*pi))*exp(-x^2/2)

#i), result should be around 8.667 based on the above result
mc_integral(n,1,3,int1)

## [1] 8.668119

#ii), result should be around 2 based on the above result
mc_integral(n,0,pi,sin)

## [1] 2.000351

#iii), result should be around 1 based on the above result
exp3<-rexp(n)
mean(exp3)
```

```
## [1] 1.000814

#iv), result should be around 0.606 based on the above result
mc_integral(n,0,3,int4)

## [1] 0.6083767

#v), result should be around 0.477 based on the above result
mc_integral(n,0,2,int5)

## [1] 0.4772337

#Alternative, using the standard normal density and an indicator
#function instead:
norm5<-rnorm(n)
ind_73<-norm5>=0&norm5<=2
mean(ind_73) #result is quite similar to the first alternative.

## [1] 0.478211
```

67 Exercise 74

Note that since for this first variant of this MC estimation, for which we sample from the uniform distribution, we have $g(x) = e^{-x}$, we estimate the variance of $\hat{\theta}$ by

$$\mathbb{V}(\hat{\theta}) = \frac{\mathbb{V}(g(x))}{n} = \frac{\mathbb{V}(e^{-x})}{n}$$

For the second variant, for which we sample from the standard exponential distribution, we have $g(x) = \mathbb{1}_{[0,0.5]}(x)$, which is an indicator function for x being in the interval $[0, 0.5]$. Similarly, in this case, we estimate the variance of the estimate by

$$\mathbb{V}(\hat{\theta}) = \frac{\mathbb{V}(g(x))}{n} = \frac{\mathbb{V}(\mathbb{1}_{[0,0.5]}(x))}{n}$$

```
n<-1000000
#First approach using uniform density
uni_74<-runif(n,0,0.5)
mc_74_1<-mean(0.5*exp(-uni_74))
mc_74_1

## [1] 0.3934161

#Estimate of the variance of theta_hat
var(exp(-uni_74))/n

## [1] 1.28644e-08

#Second approach using exponential density
exp_74<-rexp(n)
ind_74<-exp_74<=0.5
mc_74_2<-mean(ind_74)
mc_74_2
```

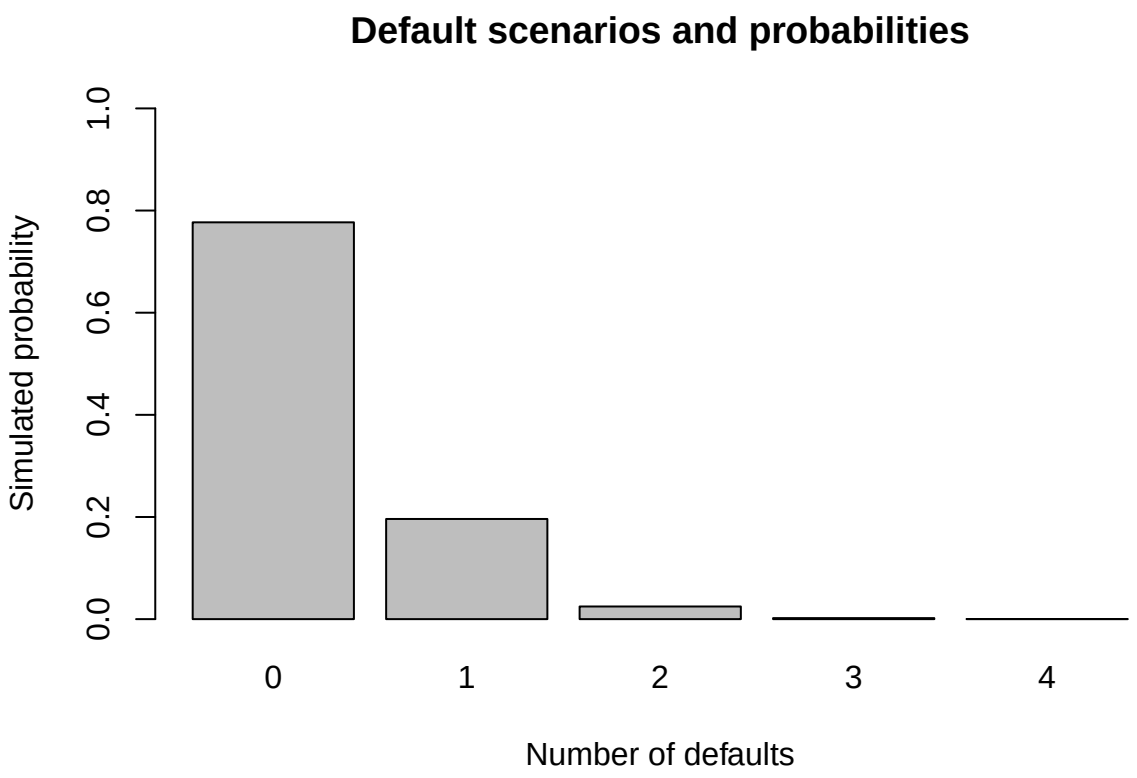
```
## [1] 0.393261

#Estimate of the variance of theta_hat_star
var(ind_74)/n

## [1] 2.38607e-07
```

We find that the variance of the first approach is considerably lower, which is a result of the fact that in the second variant, we lose around 60% of the generated sample, since they are greater than the upper bound of the interval and are thus put to zero by the indicator function. Of course, for (effectively) larger samples, the variance of the estimate will be smaller. Also the original variance of the distribution we draw from (uniform vs. exponential), is already smaller, which will not change by the respective transformations $g(x)$ we apply to them to estimate θ.

68 Exercise 75

We create default scenarios by drawing from a Bernoulli distribution, where the probability of success equals the probability of default for each of the three rating categories. For this we create matrices of random draws for each rating category, in which rows correspond to individual obligors, while columns represent draws of this entire category. The column sums of these matrices then correspond to the number of defaults in each draw, which we then display in a default matrix, where the columns are the rating categories and the rows are the draws.

```
ex_75<-function(p1,p2,p3,n){
  AA<-replicate(n,rbinom(10,1,p1))
  A<-replicate(n,rbinom(25,1,p2))
  BBB<-replicate(n,rbinom(96,1,p3))
  defaults_AA<-colSums(AA)
  defaults_A<-colSums(A)
  defaults_BBB<-colSums(BBB)
  default_matrix<-cbind(defaults_AA,defaults_A,defaults_BBB)
  colnames(default_matrix)<-c("AA","A","BBB")
  default_matrix
}

n<-100000
def_mat<-ex_75(0.0001,0.0005,0.0025,n)
total_defaults<-rowSums(def_mat)
barplot(table(total_defaults)/n,main="Default scenarios and probabilities",
        xlab="Number of defaults",ylab="Simulated probability",ylim=c(0,1))
```




```
table(total_defaults)/n

## total_defaults
##       0       1       2       3       4
## 0.77696 0.19611 0.02474 0.00200 0.00019

#We only show the head of the matrix here, since it has 100,000 rows
head(def_mat)

##      AA A BBB
## [1,]  0 0   0
## [2,]  0 0   1
## [3,]  0 0   0
## [4,]  0 0   0
## [5,]  0 0   0
## [6,]  0 0   0

#Finally, we create a matrix that shows the relative number of defaults
#per rating category
ex_75_summary<-function(matr){
  n<-nrow(matr)
  maxdef<-length(table(rowSums(matr)))-1
  dfdata<-matrix(0,ncol(matr),maxdef+1)
  colnames(dfdata)<-0:maxdef
  rownames(dfdata)<-c("AA","A","BBB")
  for(i in 1:ncol(matr)){
    for(j in 0:maxdef){
```

```
      dfdata[i,j+1]<-sum(matr[,i]==j)/n
    }
  }
  dfdata
}
ex_75_summary(def_mat)
```

```
##           0       1       2       3       4
## AA  0.99887 0.00113 0.00000 0.00000 0.00000
## A   0.98737 0.01251 0.00012 0.00000 0.00000
## BBB 0.78765 0.18827 0.02220 0.00172 0.00016
```

69 Exercise 76

We first start by using simulation. To determine the default correlation as a function of the asset correlation ρ, we create a simulating function that takes the number of draws n, the number of obligors d and exactly this asset correlation ρ as input. We can determine the default correlation therefore as a function of ρ by passing it as an argument to this function. In the following, for illustrative purposes, we take $\rho = 0.3$ for example:

```
#To obtain the data from multivariate normal distribuion with single
#correlation rho we use the function developed in exercise 42

fun_42<-function(n,d,rho)
{
  output<-matrix(0,n,d)
  F<-rnorm(n)
  for(i in 1:n)
  {
    output[i,]<-sqrt(rho)*F[i]+sqrt(1-rho)*rnorm(d)
  }
    output
}

rho<-0.3
#Get the data
n<-100000 #number of simulations
d<-qnorm(0.025)

dat<-fun_42(n,100,rho)

dcor_data<-dat<d
dcor<-cor(dcor_data)

dcor[1:6,1:6]
```

```
##             [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## [1,] 1.00000000 0.07600375 0.07334712 0.06973764 0.07387851 0.07277272
## [2,] 0.07600375 1.00000000 0.06565797 0.07027852 0.07108273 0.05945269
## [3,] 0.07334712 0.06565797 1.00000000 0.07056355 0.07908207 0.06740785
## [4,] 0.06973764 0.07027852 0.07056355 1.00000000 0.07889715 0.06914729
## [5,] 0.07387851 0.07108273 0.07908207 0.07889715 1.00000000 0.07705238
## [6,] 0.07277272 0.05945269 0.06740785 0.06914729 0.07705238 1.00000000
```

Based on this simulation we see that the default correlation is somewhere around 0.07. Next we follow by partially analytical result. Let Y_i be the r.v. which has value 1 if the value of underlying normal distribution is less than certain quantile (in our case quantile correspondin to probability 0.025) and 0 else. Then the $\mathbb{E}(Y_i) = 0.025$(denoted as p_i) for every $0 \geq i \geq 100$. Then $cor(Y_i, Y_j)$ where $i \neq j$.

$$cor(Y_i, Y_j) = \frac{\mathbb{E}((Y_i - \mathbb{E}(Y_i))(Y_j - \mathbb{E}(Y_j)))}{\sqrt{(\mathbb{E}(Y_i^2) - \mathbb{E}(Y_i)^2)(\mathbb{E}(Y_j^2) - \mathbb{E}(Y_j))^2}} = \frac{\mathbb{E}(Y_iY_j - p_ip_j - p_ip_j + p_ip_j)}{\sqrt{(\mathbb{E}(Y_i) - p_i^2)(\mathbb{E}(Y_j) - p_j^2)}} = \\ = \frac{\mathbb{E}(Y_iY_j) - p_ip_j}{\sqrt{(p_i - p_i^2)(p_j - p_j^2)}} \tag{27}$$

Since p_i and p_j are known, the only thing which remains to compute is the $\mathbb{E}(Y_iY_j)$. But this is the same as computing $P(X \leq d, Y \leq d)$, where $X \sim N(0,1)$, $Y \sim N(0,1)$ with correlation ρ and d is the desired quantile. This can also be approximated using the data we have and shoudl improve the approximation of the correlation.

```
E_YiYj<-sum(dat[,1]<=d & dat[,2]<=d)/nrow(dat)
dcor2<-(E_YiYj-0.025^2)/(0.025-0.025^2)
dcor2

## [1] 0.07323077
```

Absolutely precise analytical solution would mean solving following integral:

$$\int_{-\infty}^{d}\int_{-\infty}^{d} \frac{1}{\sqrt{2\pi|\Sigma|}} \exp -\frac{x^\intercal \Sigma^{-1} x}{2} dXdY = \phi(d,d) \tag{28}$$

where Σ is covariance matrix and $x = (X, Y)$. This integral, however cannot be analytically solved and we have to rely on numerical approximation. There are many ways of computing this probability, but we follow the approach developed by Alan Genz (1992), which approximates the probability by solving this:

$$\phi(a, \infty, \rho) = \phi(-a_1)\phi(-a_2) + \frac{1}{2\pi}\int_0^\rho \frac{1}{\sqrt{1-x^2}} \exp -\frac{a_1^2 - 2a_1a_2x + a_2^2}{2(1-x^2)} dx \tag{29}$$

We evaluate the integral using Monte-Carlo as follows:

```
#quantile
d<-qnorm(0.025)
#function
mc<-function(x){1/sqrt(1-x^2)*exp(-1/2*(2*(d)^2-2*(d^2)*x)/(1-x^2))}

n<-1000000
#probability function
r<-runif(n)*rho
x<-mc(r)

MC_result<-rho*mean(x)

E_YiYj_mc<-pnorm(d)*pnorm(d)+1/(2*pi)*MC_result
```

For this we provide a comparison of our simulated value of $\mathbb{E}(Y_iY_j)$ with the value provided by external package:

```
library(pbivnorm)
pbivnorm(d,d,rho)

## [1] 0.002370462

E_YiYj

## [1] 0.00241

E_YiYj_mc

## [1] 0.002369901

#thus all the E_YiYj we have give roughly the same result, but
#apparently those of using MC and external package are closer
#to each other
```

Then using the external package we can give a more precise result of default correlation as

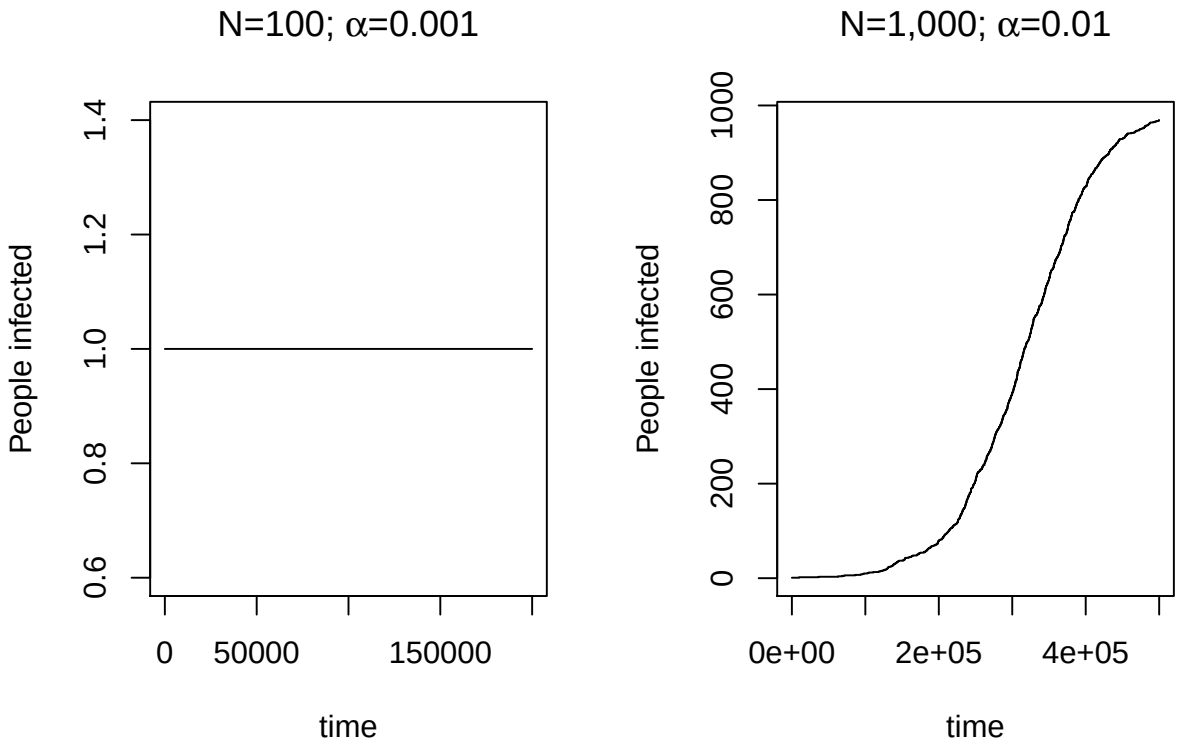
```
dcor3<-(pbivnorm(d,d,rho)-0.025^2)/(0.025-0.025^2)
dcor4<-(E_YiYj_mc-0.025^2)/(0.025-0.025^2)
dcor3 #using external package to get E_YiYj

## [1] 0.07160869

dcor2 #using existing data to get E_YiYj

## [1] 0.07323077

dcor4 #using MC estimate of E_YiYj

## [1] 0.07158566
```

70 Exercise 78

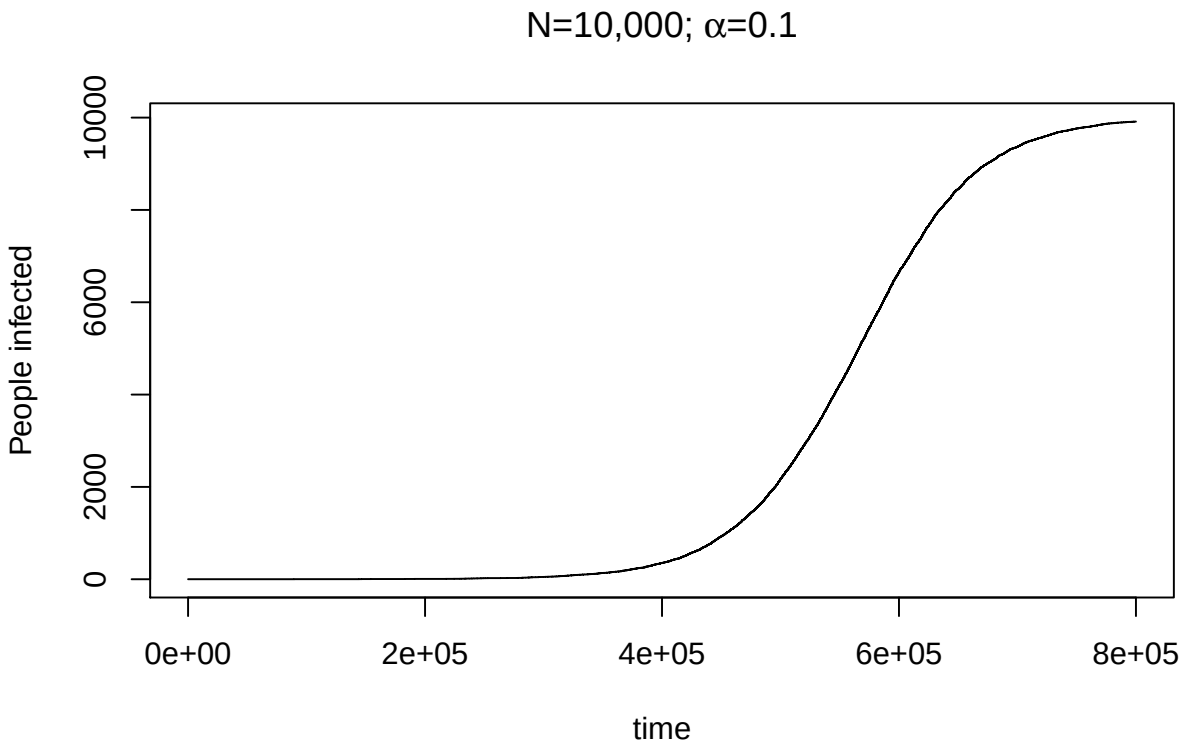
Part (a):
We create a function, in which 1 means infected and 0 means healthy and then simulate this model for various choices of N and α, summarising the respective history in plots, since they track the result in a more visually accessible manner.

```
ex_78<-function(N,alpha,initial,times){
  group<-c(rep(1,initial),rep(0,N-initial))
  history<-numeric(times)
  for(i in 1:times){
    #Sampling without replace gives equally likely encounters
    #between two people in the group:
    encounter<-sample(group,2)
    #if exactly one person in an encounter is infected, then
    #the sum of the sample from the group should be 1:
    if(sum(encounter)==1){
      rem_healthy<-which(group==0)[1]
      #Draw from Bernoulli distribution with probability alpha
      #to determine whether or not person will be infected
      infection<-rbinom(1,1,alpha)
```

```
      group[rem_healthy]<-group[rem_healthy]+infection
    }
    history[i]<-sum(group)
  }
  history
}

op <- par(mfcol = c(1, 2))
plot(ex_78(100,0.001,1,200000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=100; ",alpha,"=0.001")))
plot(ex_78(1000,0.01,1,500000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=1,000; ",alpha,"=0.01")))
```




```
par(op)
plot(ex_78(10000,0.1,1,800000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=10,000; ",alpha,"=0.1")))
```

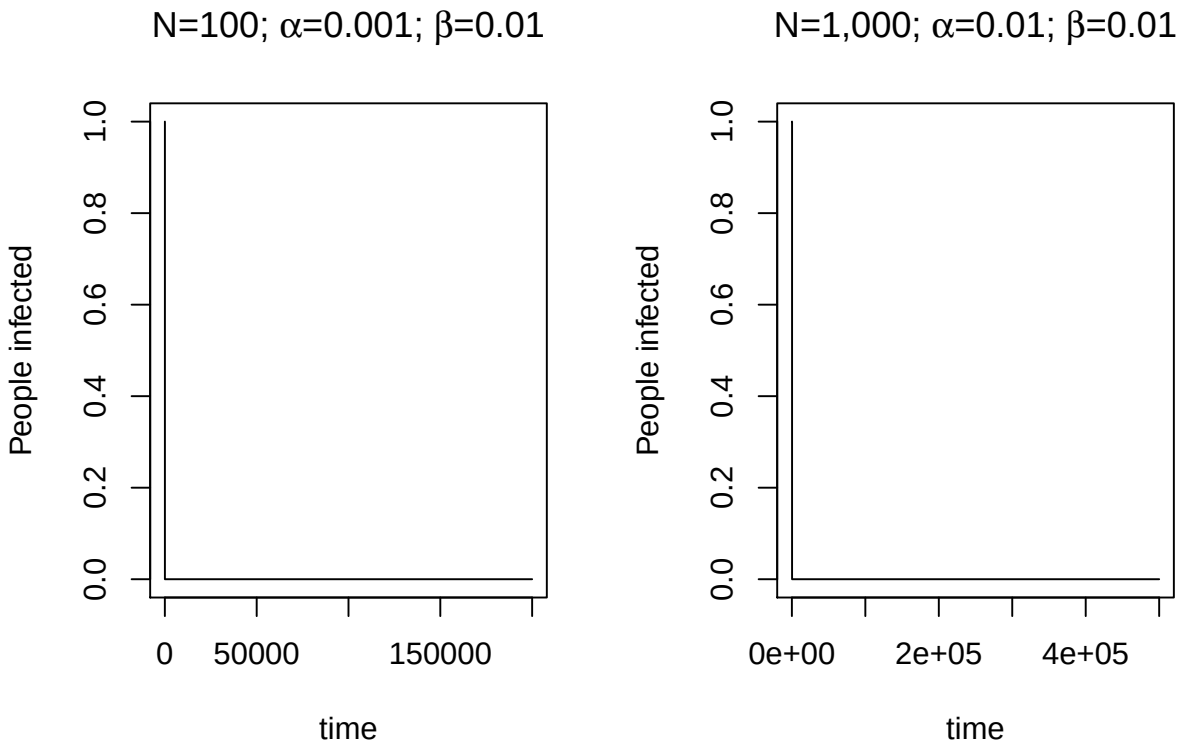


Part (b):
For this part, we create a slightly adapted version of the function in part (a) that uses a while- instead of a for-loop to determine the time necessary to reach 1,000 infections. We use this then to simulate 200 paths to determine the expected value.

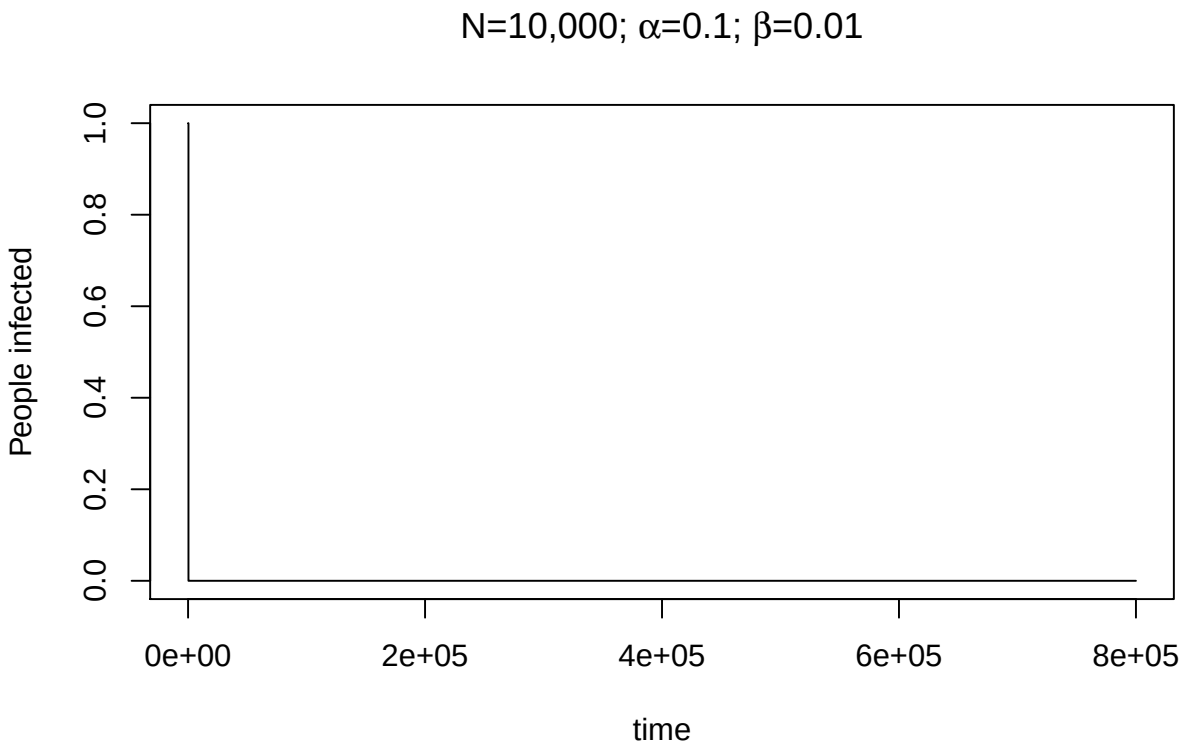
```
ex_78_b<-function(N,alpha,initial,goal){
  group<-c(rep(1,initial),rep(0,N-initial))
  result<-0
  while(sum(group)<goal){
    encounter<-sample(group,2)
    if(sum(encounter)==1){
      rem_healthy<-which(group==0)[1]
      infection<-rbinom(1,1,alpha)
      group[rem_healthy]<-group[rem_healthy]+infection
    }
    result<-result+1
  }
  result
}
mean(replicate(200,ex_78_b(10000,0.1,1,1000)))

## [1] 384247.9
```

Part (c):
For this, we adapt the function from part (a), so as to include the chance for recovery β. In this, we make the assumption that an individual that is infected at time t cannot recover in that same instant t, but his earliest point of recovery is the next unit of time:

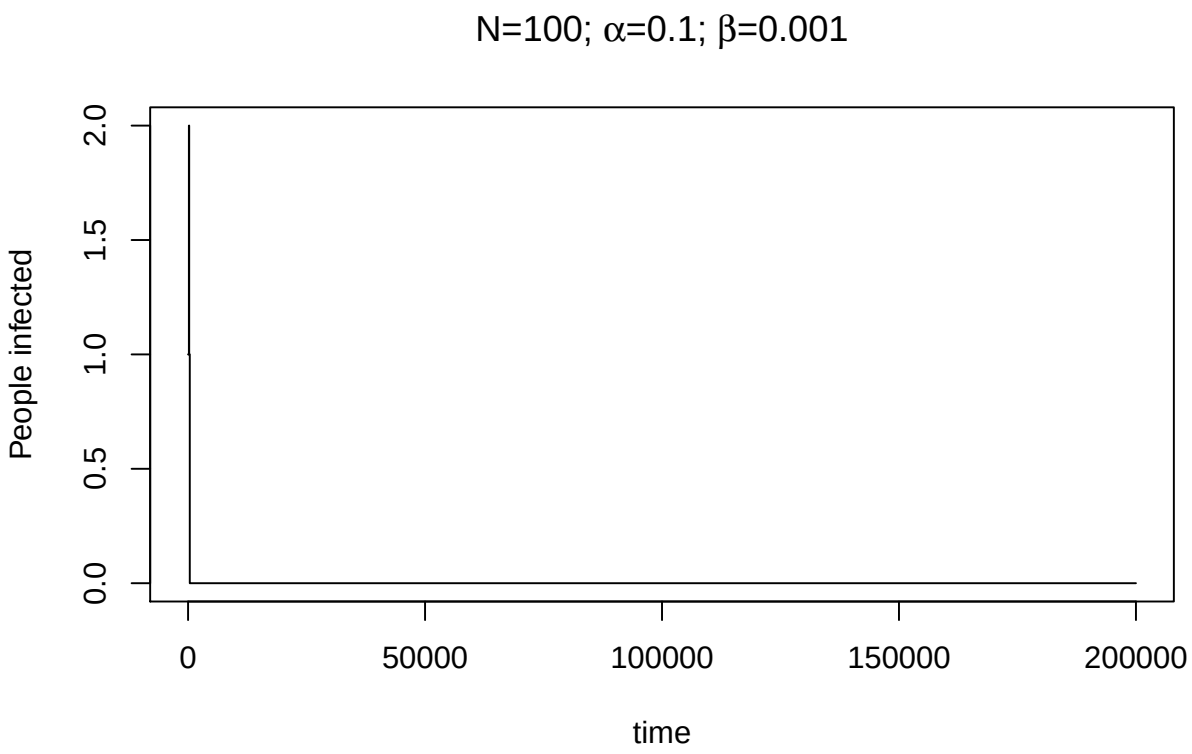
```r
ex_78_c<-function(N,alpha,beta,initial,times){
  group<-c(rep(1,initial),rep(0,N-initial))
  history<-numeric(times)
  for(i in 1:times){
    #There is of course no point in continuing the simulation, if
    #there are no infected people in the group left:
    if(sum(group)==0){break}
    #In the following vector, 1 means recovery, 0 means no recovery
    recovery<-rbinom(sum(group),1,beta)
    group[which(group==1)]<-group[which(group==1)]-recovery
    #Modelling encounters as before:
    encounter<-sample(group,2)
    if(sum(encounter)==1){
      rem_healthy<-which(group==0)[1]
      infection<-rbinom(1,1,alpha)
      group[rem_healthy]<-group[rem_healthy]+infection
    }
    history[i]<-sum(group)
  }
  history
}

op <- par(mfcol = c(1, 2))
plot(ex_78_c(100,0.001,0.01,1,200000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=100; ",alpha,"=0.001; ",beta,"=0.01")))
plot(ex_78_c(1000,0.01,0.01,1,500000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=1,000; ",alpha,"=0.01; ",beta,"=0.01")))
```



```r
par(op)
plot(ex_78_c(10000,0.1,0.01,1,800000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=10,000; ",alpha,"=0.1; ",beta,"=0.01")))
```




```r
#We see that for all the histories we simulated in part (a), the speed of recovery
#is now considerably faster than the propagation of the illness.
#However, we can modify alpha and beta for it to constitute and interesting,
#white-noise like pattern, where infections and recoveries lead to a fluctutation
#in total illnesses in the group. We do this for N = 100, to minimise the
#computational burden:
plot(ex_78_c(100,0.1,0.001,1,200000),type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=100; ",alpha,"=0.1; ",beta,"=0.001")))
```

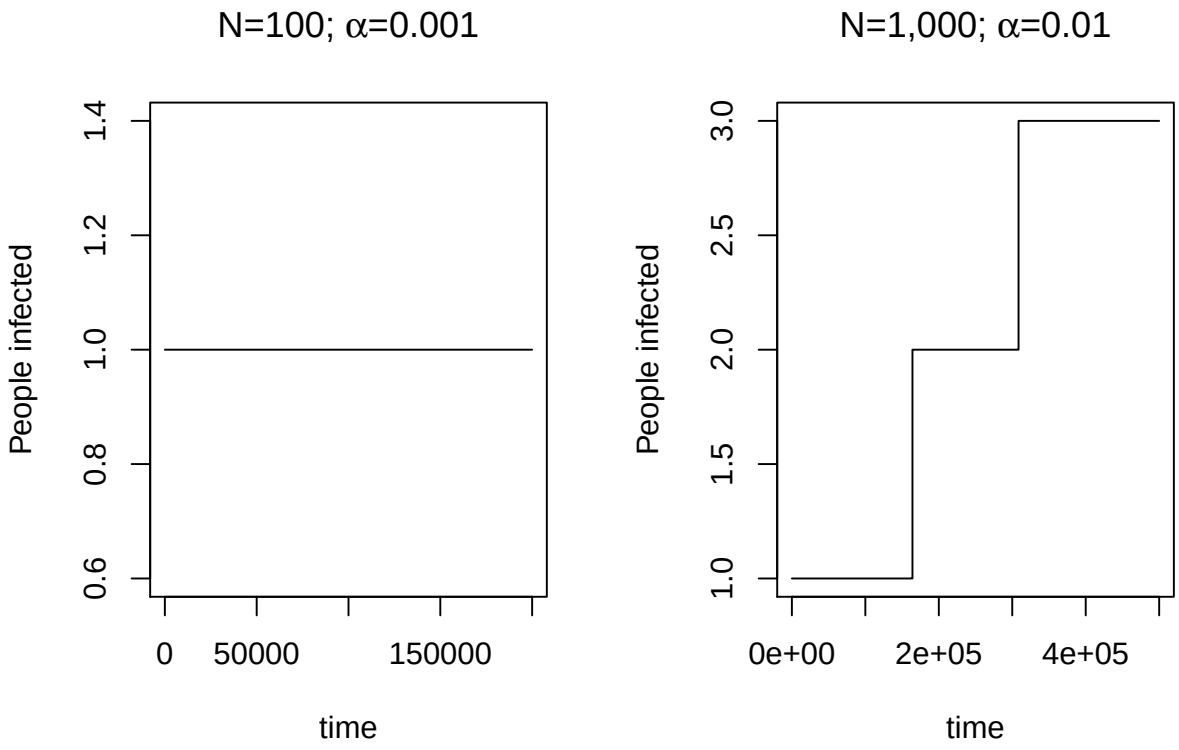


Part (d):

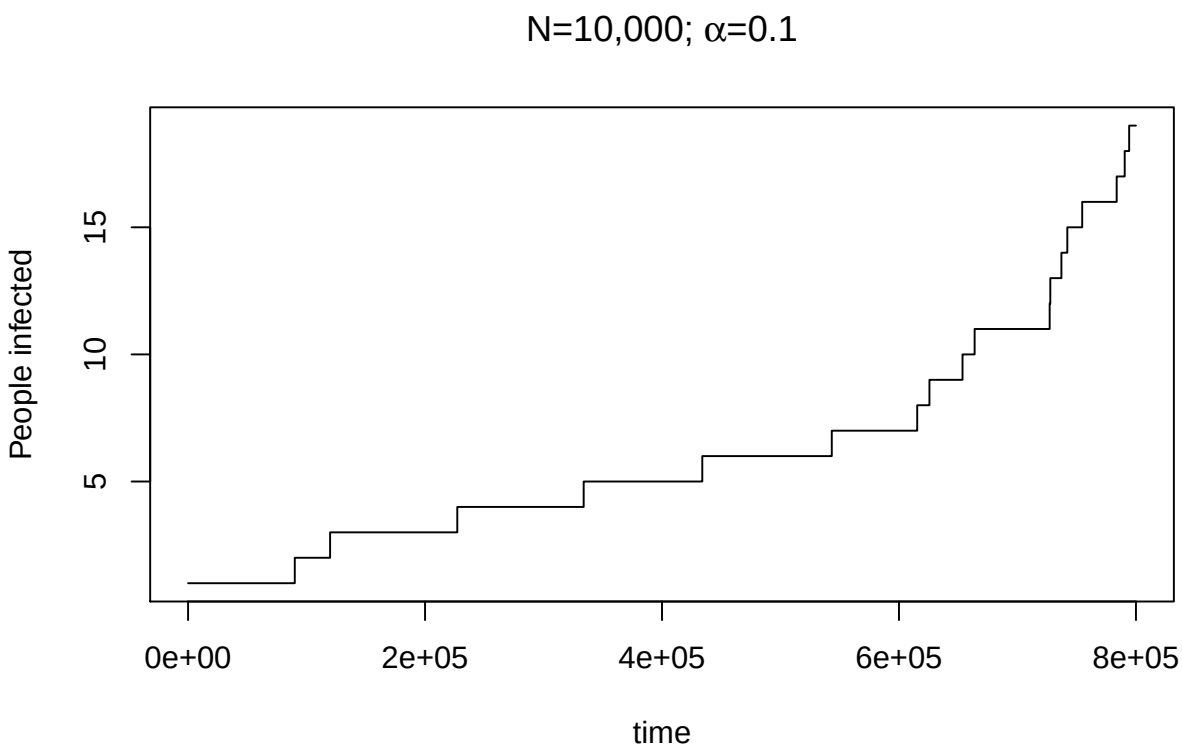
Here, we introduce an additional source of randomness by not having an encounter at each unit of time, but instead, having the times between encounters being distributed as an exponential random variable with mean of 5 minutes, by which we set the parameter of the exponential distribution $\lambda = 1/5$.

```
ex_78_d<-function(N,alpha,initial,times){
  group<-c(rep(1,initial),rep(0,N-initial))
  exp_times<-numeric(0)
  i<-1
  #We simulate points in time from the exponential distribution so long
  #as their cumulative sum is smaller than the time we give in as an input
  while(sum(exp_times)<times){
    exp_times[i]<-rexp(1,0.2)
    i<-i+1
  }
  #The last one will be greater than the input "times", so we delete it
  exp_times<-exp_times[-length(exp_times)]
  history<-numeric(length(exp_times))
  #Now, there is an encounter only for each simulated point in time, not
  #at every instant of time. The modelling of encounters and infection
  #remains the same in principle, however.
  for(i in seq_along(exp_times)){
    encounter<-sample(group,2)
    if(sum(encounter)==1){
      rem_healthy<-which(group==0)[1]
      infection<-rbinom(1,1,alpha)
      group[rem_healthy]<-group[rem_healthy]+infection
    }
```

```
    history[i]<-sum(group)
  }
  list(cumsum(exp_times),history)
}

plot1<-ex_78_d(100,0.001,1,200000)
plot2<-ex_78_d(1000,0.01,1,500000)
plot3<-ex_78_d(10000,0.1,1,800000)

op <- par(mfcol = c(1, 2))
plot(plot1[[1]],plot1[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=100; ",alpha,"=0.001")))
plot(plot2[[1]],plot2[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=1,000; ",alpha,"=0.01")))
```



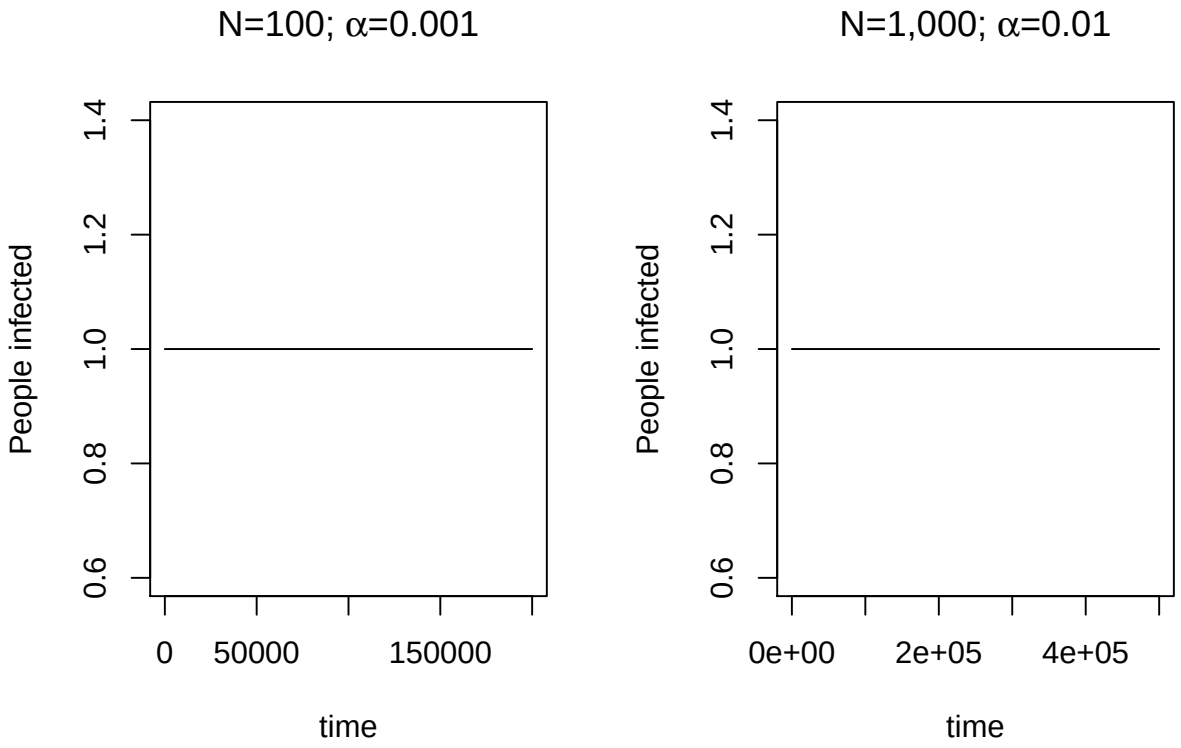

```
par(op)
plot(plot3[[1]],plot3[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=10,000; ",alpha,"=0.1")))
```



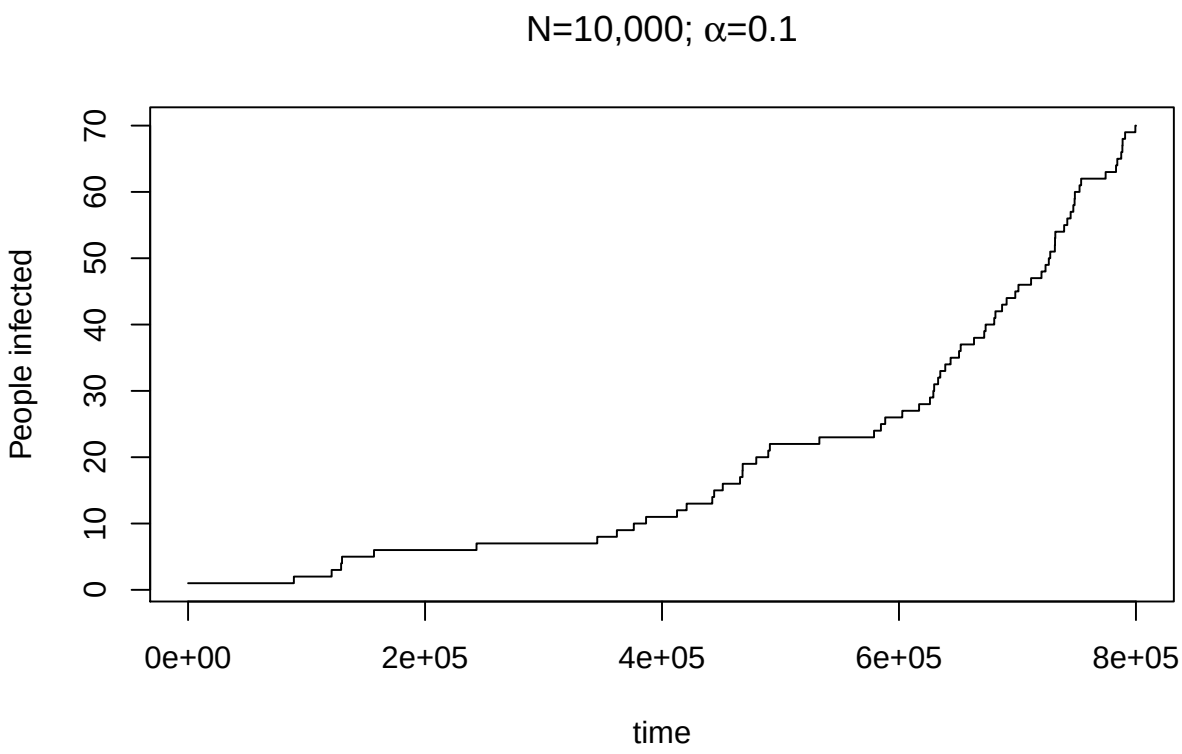
Part (e):
For this we have the times between encounters being distributed as the absolute value of a normal random variable a mean of 5 minutes and a standard deviation of one minute, by which we set the parameter of the normal distribution $\mu = 5$ and $\sigma^2 = 1$.

```
ex_78_e<-function(N,alpha,initial,times){
  group<-c(rep(1,initial),rep(0,N-initial))
  norm_times<-numeric(0)
  i<-1
  #In the following, the exact same logic applies as in d)
  #only that we know sample from the normal distribution, not from
  #the exponential distribution anymore.
  while(sum(norm_times)<times){
    norm_times[i]<-rnorm(1,5,1)
    i<-i+1
  }
  norm_times<-norm_times[-length(norm_times)]
  history<-numeric(length(norm_times))
  for(i in seq_along(norm_times)){
    encounter<-sample(group,2)
    if(sum(encounter)==1){
      rem_healthy<-which(group==0)[1]
      infection<-rbinom(1,1,alpha)
      group[rem_healthy]<-group[rem_healthy]+infection
    }
    history[i]<-sum(group)
  }
  list(cumsum(norm_times),history)
```

```
}

plot4<-ex_78_e(100,0.001,1,200000)
plot5<-ex_78_e(1000,0.01,1,500000)
plot6<-ex_78_e(10000,0.1,1,800000)

op <- par(mfcol = c(1, 2))
plot(plot4[[1]],plot4[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=100; ",alpha,"=0.001")))
plot(plot5[[1]],plot5[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=1,000; ",alpha,"=0.01")))
```




```
par(op)
plot(plot6[[1]],plot6[[2]],type="s",xlab="time",ylab="People infected",
     main=expression(paste("N=10,000; ",alpha,"=0.1")))
```



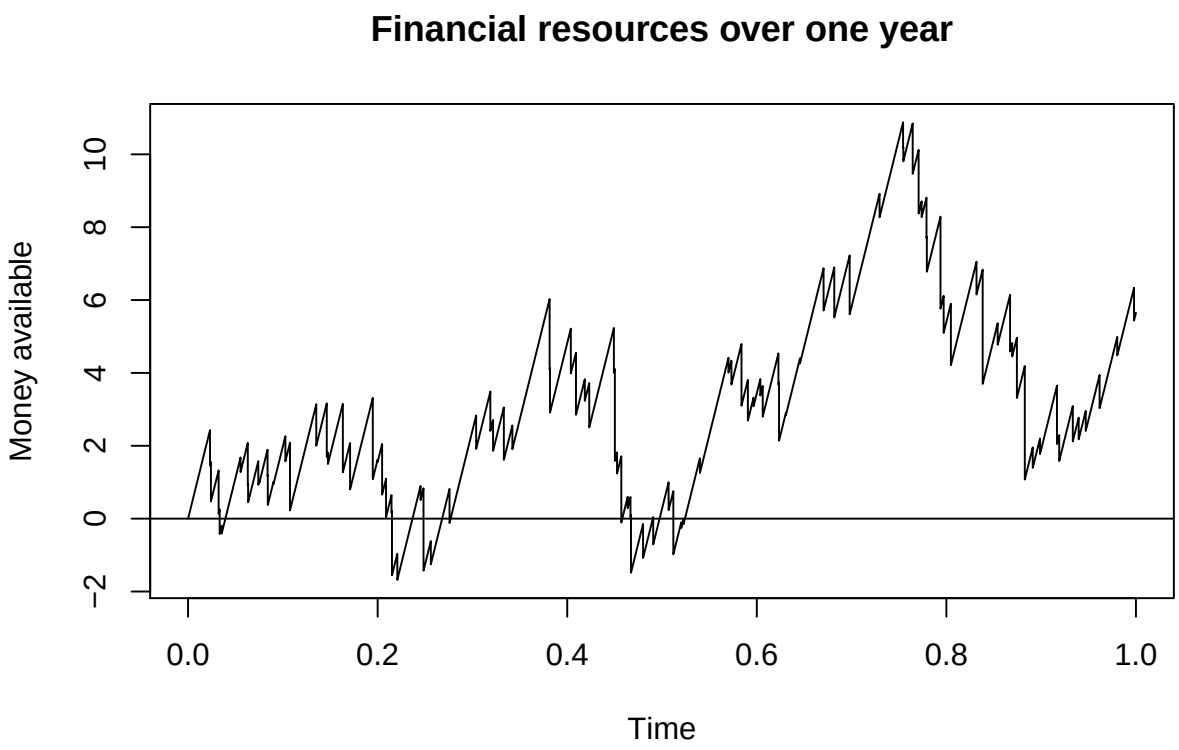
71 Exercise 79

Claims arise according to a Poisson process at a rate of 100 per year. Since we know that the number of claims in a Poisson process, given by $N(t)$, will be Poisson distributed with parameter λt, and the mean of a Poisson distributed random variable is just equal to its parameter, we can conclude that $\lambda t = 100$ for $t = 1 \Rightarrow \lambda = 100$. Claim size follows a $\Gamma(2, 2)$ distribution and the insurance company earns premiums at a rate of 105 per year, i.e. according to the function $105t$, where t is the time measured in years.

Part (a):

```
#Function to generate points in time of claims based on
#Poisson process:
rpp1 <- function(ttime, lambda){
  sort(runif(rpois(1, ttime * lambda), max = ttime))
}

ex_79<-function(ttime,lambda,premiumrate=105,plotoption=TRUE){
  claimtimes<-rpp1(ttime,lambda)
  claimnumber<-length(claimtimes)
  #Claims are Gamma distributed:
  claims<-rgamma(claimnumber,shape=2,rate=2)
  #Create sequence of how the available financial resources develop
  #through time. At each claim time, we have two points: one with the available
  #resources before and one with them after the claim:
  xvalues<-c(0,rep(claimtimes,each=2),ttime)
  yvalues<-premiumrate*xvalues
  claimindex<-1
  for(i in 3:length(yvalues)){
    if(i%%2==1){
```

```
      #Fill yvalues of points after outpayment of the claim
      yvalues[i]<-yvalues[i-1]-claims[claimindex]
      claimindex<-claimindex+1
    }else{
      #Fill yvalues of points before outpayment of the claim
      yvalues[i]<-premiumrate*xvalues[i]+(yvalues[i-1]-premiumrate*xvalues[i-1])
    }
  }
  if(plotoption){
  #Create plot
  plot(xvalues,yvalues,type="l",main="Financial resources over one year",
       ylab="Money available",xlab="Time")
  abline(0,0)
  }
  #Return list of important simulation outputs
  money<-cbind(xvalues,yvalues)
  colnames(money)<-c("t","Money")
  list(claimtimepoints=claimtimes,claim_amounts=claims,moneyovertime=money)
}
ex_79(1,100)
```




```
## $claimtimepoints
##  [1] 0.02314423 0.02408118 0.03221789 0.03331135 0.03546273 0.05534483 0.06299034 0.06333675
##  [9] 0.07409542 0.07518948 0.08384343 0.08407274 0.09015646 0.10265557 0.10751529 0.13524151
## [17] 0.14631308 0.14751968 0.16326724 0.17091586 0.19484363 0.19995703 0.20469957 0.20884635
## [25] 0.21461629 0.21511501 0.22073457 0.24527803 0.24836991 0.25610298 0.27581232 0.30390578
## [33] 0.31888116 0.32176029 0.33312580 0.34213704 0.38141691 0.38181968 0.40379308 0.40922567
## [41] 0.41853802 0.42322188 0.44923223 0.45027265 0.45255047 0.45706277 0.46380026 0.46672069
## [49] 0.46726866 0.48003160 0.49071495 0.50695091 0.51194356 0.52026285 0.52273813 0.53997410
```

```
## [57] 0.57012752 0.57324044 0.58384240 0.59059081 0.59658803 0.60372864 0.60625523 0.62279535
## [65] 0.62344993 0.63067436 0.64556567 0.67045473 0.68175338 0.69797899 0.72951092 0.75434035
## [73] 0.75456144 0.76446400 0.77072679 0.77393132 0.77904108 0.77938531 0.79378440 0.79713785
## [81] 0.80475473 0.83180683 0.83830718 0.85415870 0.86721225 0.86946819 0.87439434 0.88274094
## [89] 0.89113681 0.89880311 0.91670553 0.91903623 0.93340368 0.93960524 0.94703419 0.96166235
## [97] 0.98030623 0.99799593
##
## $claim_amounts
##  [1] 0.97467210 1.08357525 1.18769388 0.67334121 0.21228441 0.40232230 1.17252599 0.49435559
##  [9] 0.64954896 0.06029654 0.72272005 0.81956043 0.06421797 0.68949019 1.85453461 1.13873701
## [17] 1.46988438 0.32067448 1.88109306 1.27238184 2.23390797 0.06581243 1.38999034 1.05416153
## [25] 0.48540952 1.76828309 0.71590960 0.38730090 2.26306810 0.63687407 0.93539067 0.91539043
## [33] 1.08000669 0.84945811 1.44024070 0.65368135 1.93412139 1.23024405 1.22973512 1.70619715
## [41] 0.59513624 1.21994216 1.23512222 2.52816448 0.58179720 1.81896752 0.31346253 0.54602883
## [49] 1.59090270 0.93394246 0.74942006 0.76730862 1.73411880 0.15268325 0.15365702 0.40727041
## [57] 0.40516244 0.65638062 1.69430609 1.11664889 0.23936383 0.45377496 0.84400837 0.86326473
## [65] 1.60283061 0.06067159 0.14004003 1.16152870 1.37709879 1.61802314 0.64564253 0.70790491
## [73] 0.38724741 1.38373818 1.74643752 0.43723401 1.10985276 0.96032380 2.52371449 1.01779166
## [81] 1.68713193 0.89936881 3.13715952 0.59021691 1.55675107 0.37395440 1.65609352 3.11319219
## [89] 0.56025720 0.42929360 1.60202140 0.71222300 0.96734041 0.59492236 0.55109969 0.91249714
## [97] 0.50884323 0.89890073
##
## $moneyovertime
##                 t       Money
##   [1,] 0.00000000  0.00000000
##   [2,] 0.02314423  2.43014410
##   [3,] 0.02314423  1.45547201
##   [4,] 0.02408118  1.55385211
##   [5,] 0.02408118  0.47027686
##   [6,] 0.03221789  1.32463145
##   [7,] 0.03221789  0.13693757
##   [8,] 0.03331135  0.25175058
##   [9,] 0.03331135 -0.42159063
##  [10,] 0.03546273 -0.19569624
##  [11,] 0.03546273 -0.40798065
##  [12,] 0.05534483  1.67963987
##  [13,] 0.05534483  1.27731757
##  [14,] 0.06299034  2.08009701
##  [15,] 0.06299034  0.90757102
##  [16,] 0.06333675  0.94394362
##  [17,] 0.06333675  0.44958802
##  [18,] 0.07409542  1.57924835
##  [19,] 0.07409542  0.92969939
##  [20,] 0.07518948  1.04457613
##  [21,] 0.07518948  0.98427959
##  [22,] 0.08384343  1.89294360
##  [23,] 0.08384343  1.17022355
##  [24,] 0.08407274  1.19430166
##  [25,] 0.08407274  0.37474124
##  [26,] 0.09015646  1.01353109
##  [27,] 0.09015646  0.94931312
##  [28,] 0.10265557  2.26171985
##  [29,] 0.10265557  1.57222967
##  [30,] 0.10751529  2.08250023
##  [31,] 0.10751529  0.22796562
##  [32,] 0.13524151  3.13921867
##  [33,] 0.13524151  2.00048166
##  [34,] 0.14631308  3.16299729
```

```
##  [35,] 0.14631308  1.69311290
##  [36,] 0.14751968  1.81980604
##  [37,] 0.14751968  1.49913155
##  [38,] 0.16326724  3.15262535
##  [39,] 0.16326724  1.27153228
##  [40,] 0.17091586  2.07463668
##  [41,] 0.17091586  0.80225484
##  [42,] 0.19484363  3.31467043
##  [43,] 0.19484363  1.08076245
##  [44,] 0.19995703  1.61766973
##  [45,] 0.19995703  1.55185730
##  [46,] 0.20469957  2.04982453
##  [47,] 0.20469957  0.65983419
##  [48,] 0.20884635  1.09524596
##  [49,] 0.20884635  0.04108443
##  [50,] 0.21461629  0.64692843
##  [51,] 0.21461629  0.16151891
##  [52,] 0.21511501  0.21388431
##  [53,] 0.21511501 -1.55439878
##  [54,] 0.22073457 -0.96434550
##  [55,] 0.22073457 -1.68025510
##  [56,] 0.24527803  0.89680834
##  [57,] 0.24527803  0.50950744
##  [58,] 0.24836991  0.83415524
##  [59,] 0.24836991 -1.42891286
##  [60,] 0.25610298 -0.61694087
##  [61,] 0.25610298 -1.25381494
##  [62,] 0.27581232  0.81566591
##  [63,] 0.27581232 -0.11972475
##  [64,] 0.30390578  2.83008813
##  [65,] 0.30390578  1.91469770
##  [66,] 0.31888116  3.48711296
##  [67,] 0.31888116  2.40710627
##  [68,] 0.32176029  2.70941481
##  [69,] 0.32176029  1.85995671
##  [70,] 0.33312580  3.05333495
##  [71,] 0.33312580  1.61309424
##  [72,] 0.34213704  2.55927434
##  [73,] 0.34213704  1.90559299
##  [74,] 0.38141691  6.02997941
##  [75,] 0.38141691  4.09585802
##  [76,] 0.38181968  4.13814936
##  [77,] 0.38181968  2.90790530
##  [78,] 0.40379308  5.21511181
##  [79,] 0.40379308  3.98537669
##  [80,] 0.40922567  4.55579943
##  [81,] 0.40922567  2.84960229
##  [82,] 0.41853802  3.82739885
##  [83,] 0.41853802  3.23226261
##  [84,] 0.42322188  3.72406814
##  [85,] 0.42322188  2.50412598
##  [86,] 0.44923223  5.23521235
##  [87,] 0.44923223  4.00009012
##  [88,] 0.45027265  4.10933369
##  [89,] 0.45027265  1.58116921
##  [90,] 0.45255047  1.82034093
##  [91,] 0.45255047  1.23854373
##  [92,] 0.45706277  1.71233528
```

```
##  [93,] 0.45706277 -0.10663224
##  [94,] 0.46380026  0.60080355
##  [95,] 0.46380026  0.28734101
##  [96,] 0.46672069  0.59398664
##  [97,] 0.46672069  0.04795781
##  [98,] 0.46726866  0.10549470
##  [99,] 0.46726866 -1.48540799
## [100,] 0.48003160 -0.14529915
## [101,] 0.48003160 -1.07924160
## [102,] 0.49071495  0.04250949
## [103,] 0.49071495 -0.70691056
## [104,] 0.50695091  0.99786533
## [105,] 0.50695091  0.23055671
## [106,] 0.51194356  0.75478504
## [107,] 0.51194356 -0.97933375
## [108,] 0.52026285 -0.10580775
## [109,] 0.52026285 -0.25849101
## [110,] 0.52273813  0.00141301
## [111,] 0.52273813 -0.15224401
## [112,] 0.53997410  1.65753317
## [113,] 0.53997410  1.25026276
## [114,] 0.57012752  4.41637167
## [115,] 0.57012752  4.01120922
## [116,] 0.57324044  4.33806537
## [117,] 0.57324044  3.68168474
## [118,] 0.58384240  4.79489048
## [119,] 0.58384240  3.10058440
## [120,] 0.59059081  3.80916820
## [121,] 0.59059081  2.69251931
## [122,] 0.59658803  3.32222680
## [123,] 0.59658803  3.08286297
## [124,] 0.60372864  3.83262759
## [125,] 0.60372864  3.37885263
## [126,] 0.60625523  3.64414401
## [127,] 0.60625523  2.80013564
## [128,] 0.62279535  4.53684865
## [129,] 0.62279535  3.67358392
## [130,] 0.62344993  3.74231461
## [131,] 0.62344993  2.13948400
## [132,] 0.63067436  2.89804874
## [133,] 0.63067436  2.83737715
## [134,] 0.64556567  4.40096562
## [135,] 0.64556567  4.26092559
## [136,] 0.67045473  6.87427643
## [137,] 0.67045473  5.71274773
## [138,] 0.68175338  6.89910617
## [139,] 0.68175338  5.52200737
## [140,] 0.69797899  7.22569669
## [141,] 0.69797899  5.60767355
## [142,] 0.72951092  8.91852605
## [143,] 0.72951092  8.27288351
## [144,] 0.75434035 10.87997386
## [145,] 0.75434035 10.17206896
## [146,] 0.75456144 10.19528274
## [147,] 0.75456144  9.80803533
## [148,] 0.76446400 10.84780385
## [149,] 0.76446400  9.46406567
## [150,] 0.77072679 10.12165869
```

```
## [151,] 0.77072679  8.37522117
## [152,] 0.77393132  8.71169766
## [153,] 0.77393132  8.27446365
## [154,] 0.77904108  8.81098806
## [155,] 0.77904108  7.70113529
## [156,] 0.77938531  7.73727943
## [157,] 0.77938531  6.77695563
## [158,] 0.79378440  8.28886057
## [159,] 0.79378440  5.76514609
## [160,] 0.79713785  6.11725834
## [161,] 0.79713785  5.09946668
## [162,] 0.80475473  5.89923853
## [163,] 0.80475473  4.21210659
## [164,] 0.83180683  7.05257722
## [165,] 0.83180683  6.15320841
## [166,] 0.83830718  6.83574536
## [167,] 0.83830718  3.69858584
## [168,] 0.85415870  5.36299491
## [169,] 0.85415870  4.77277799
## [170,] 0.86721225  6.14340047
## [171,] 0.86721225  4.58664939
## [172,] 0.86946819  4.82352320
## [173,] 0.86946819  4.44956881
## [174,] 0.87439434  4.96681541
## [175,] 0.87439434  3.31072189
## [176,] 0.88274094  4.18711455
## [177,] 0.88274094  1.07392235
## [178,] 0.89113681  1.95548809
## [179,] 0.89113681  1.39523089
## [180,] 0.89880311  2.20019247
## [181,] 0.89880311  1.77089887
## [182,] 0.91670553  3.65065340
## [183,] 0.91670553  2.04863200
## [184,] 0.91903623  2.29335539
## [185,] 0.91903623  1.58113240
## [186,] 0.93340368  3.08971450
## [187,] 0.93340368  2.12237409
## [188,] 0.93960524  2.77353830
## [189,] 0.93960524  2.17861593
## [190,] 0.94703419  2.95865599
## [191,] 0.94703419  2.40755631
## [192,] 0.96166235  3.94351287
## [193,] 0.96166235  3.03101572
## [194,] 0.98030623  4.98862281
## [195,] 0.98030623  4.47977958
## [196,] 0.99799593  6.33719832
## [197,] 0.99799593  5.43829760
## [198,] 1.00000000  5.64872480
```

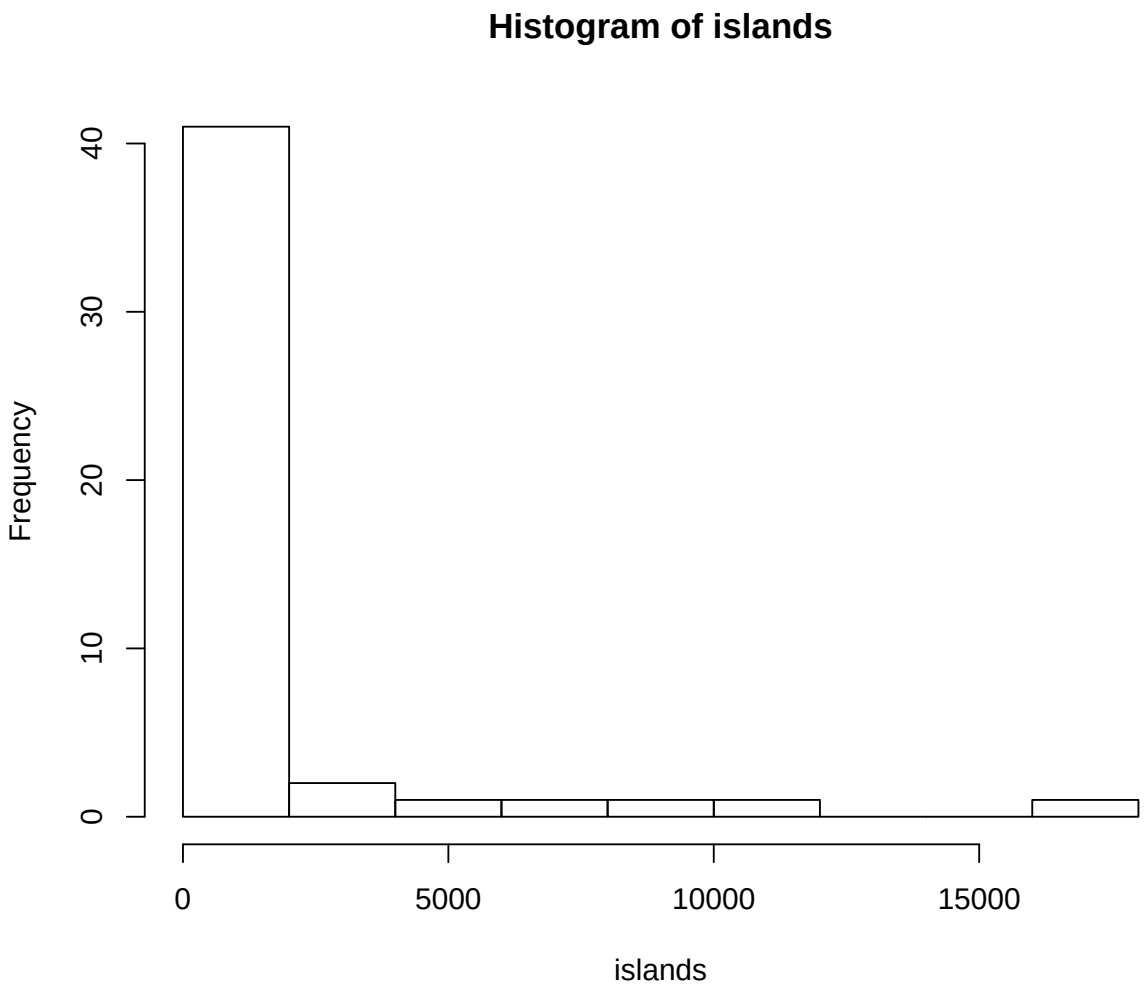
Part (b):

```
minima<-numeric(1000)
finals<-numeric(1000)
for(i in 1:1000){
  simresult<-ex_79(1,100,plotoption=FALSE)
  minima[i]<-min(simresult$moneyovertime[,2])
  finals[i]<-simresult$moneyovertime[,2][nrow(simresult$moneyovertime)]
}
```

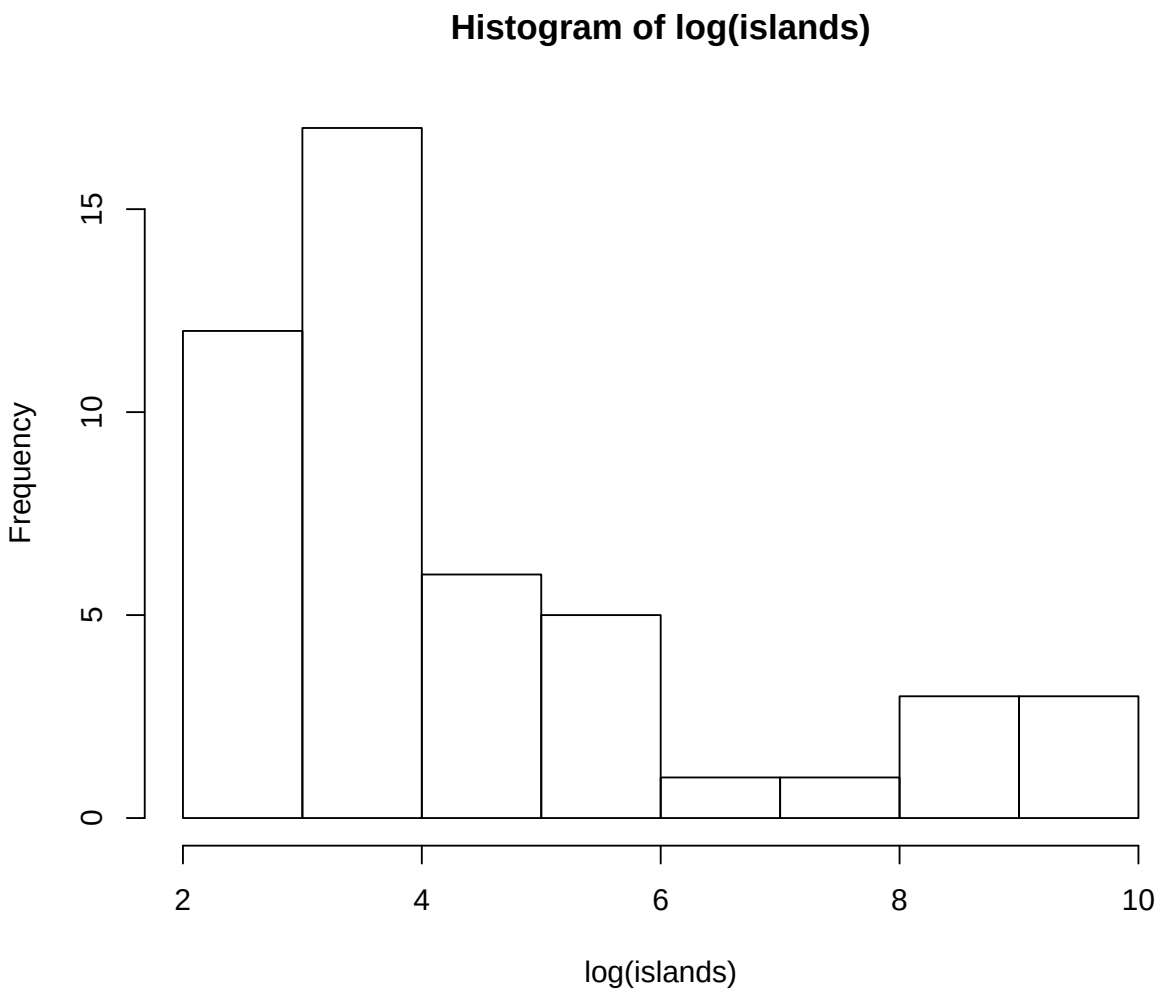
```r
#i) Expected minimum amount of money the insurance firm would have:
mean(minima)

## [1] -7.449423

#ii) Expected final amount of money the insurance firm would have:
mean(finals)

## [1] 4.942021
```

72 Exercise 83

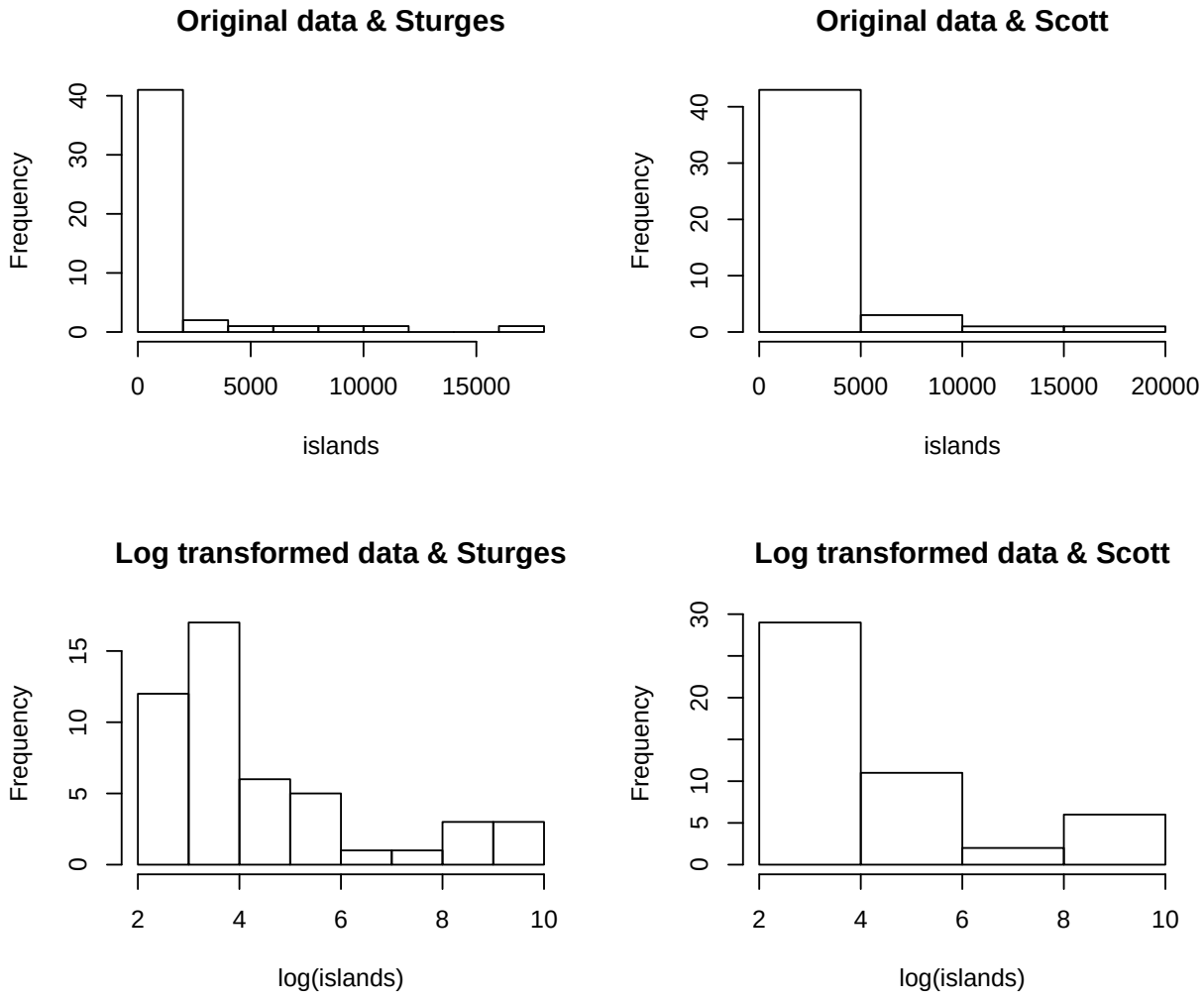
```r
#loading data into R
data(islands)

##--a
hist(islands)
```



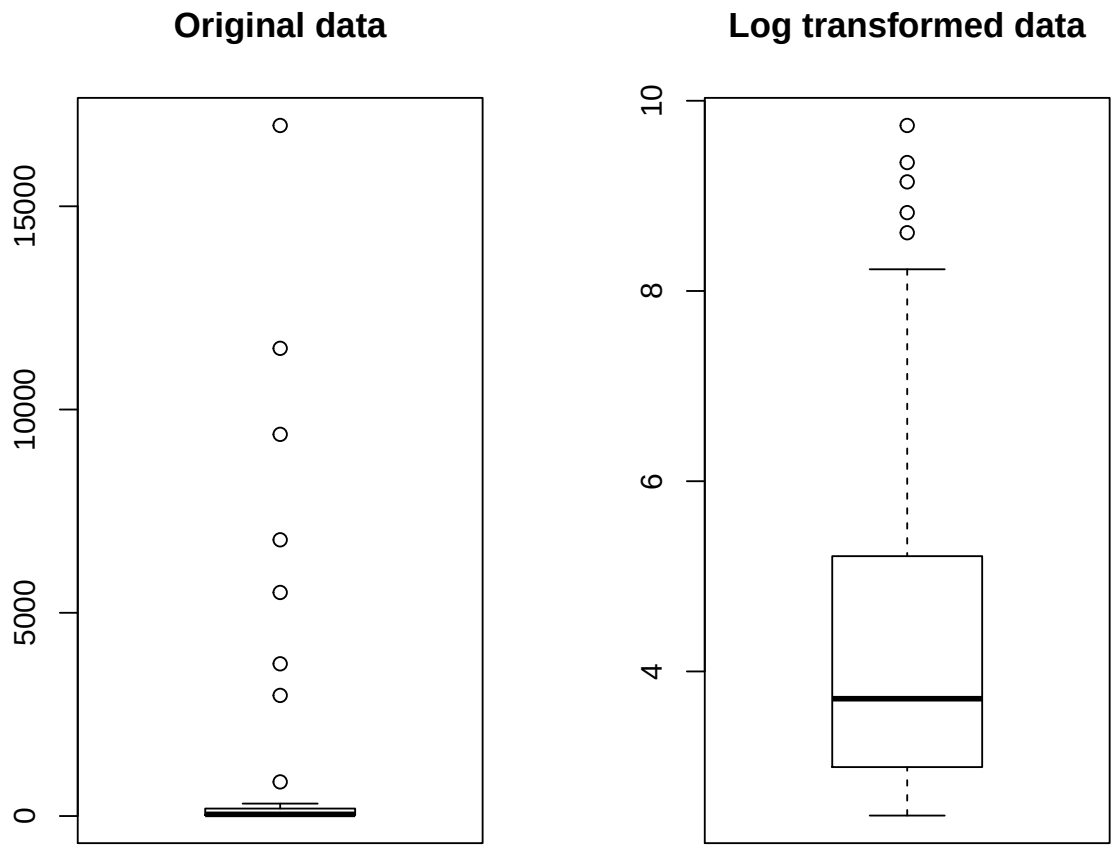
```r
##--b
hist(log(islands))
```



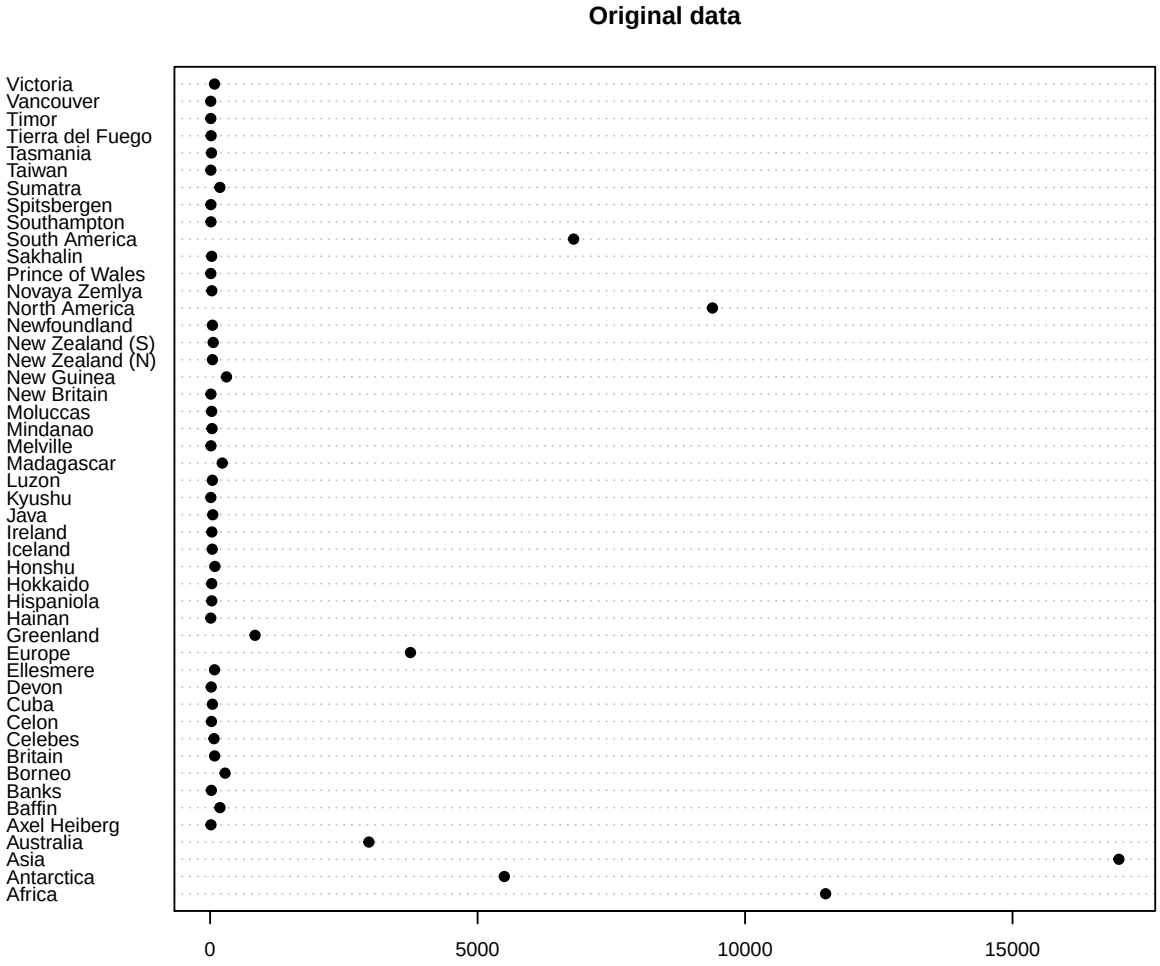

```r
#Yes, there is an advantage. Logarithms are used as a strictly increasing, concave
#transformation, which turns highly skewed distributions into less skewed distributions.
#From the histogram, it is obvious that the log transformation produces a less skewed
#distribution. It pushed observations closer to each other, which is particularly
#useful in this case, since we have a very skewed distribution without using logs.
#Also, note that both scott and sturges algorithms implicitly assume normal data and
#log transformation makes the data look more normal at a first impression.

##--c
par(mfrow=c(2,2))
hist(islands,breaks = "Sturges",main="Original data & Sturges")
hist(islands,breaks = "Scott",main="Original data & Scott")
hist(log(islands),breaks = "Sturges",main="Log transformed data & Sturges")
hist(log(islands),breaks = "Scott",main="Log transformed data & Scott")
```



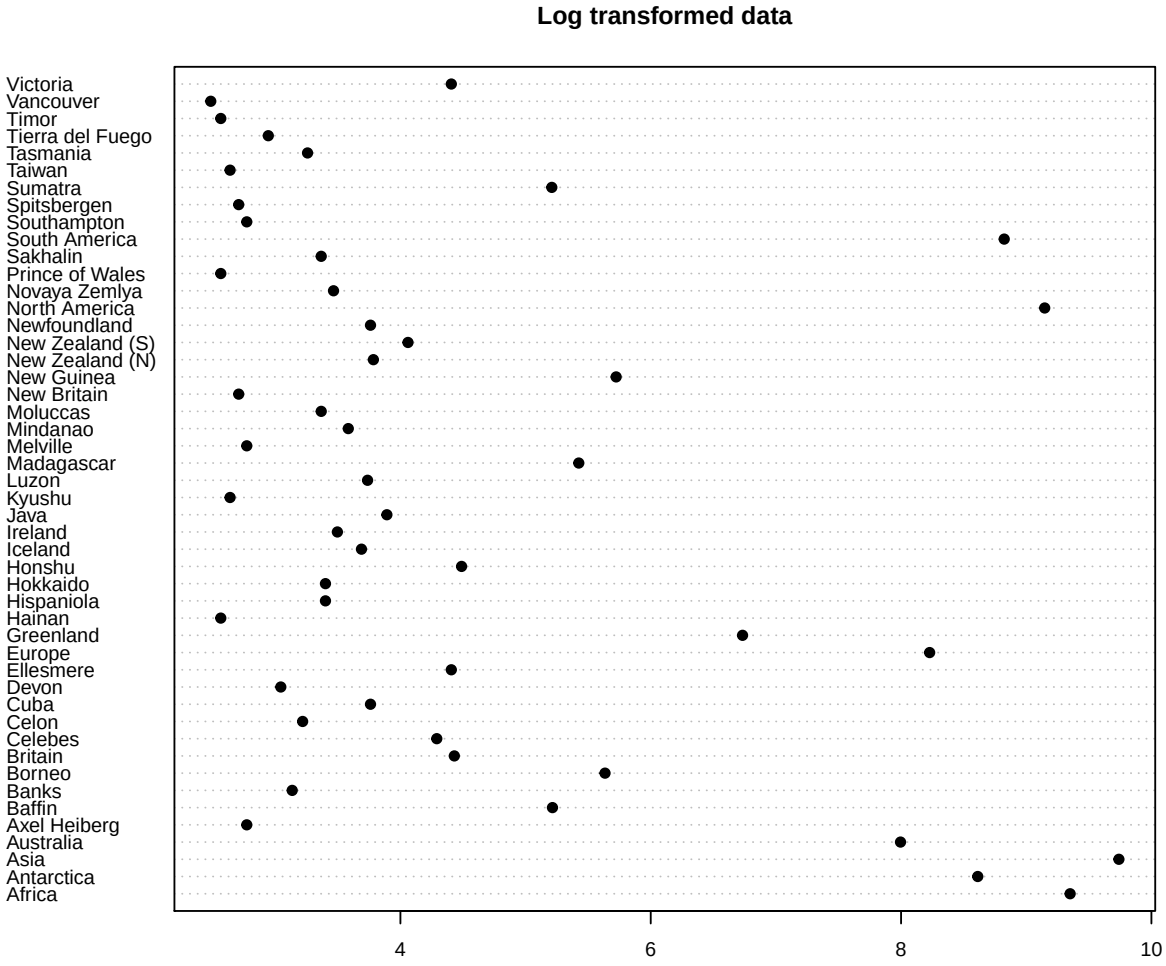

```
par(mfrow=c(1,1))
#Obviously, in each case, we get different histograms, however there does not exist
#any strict rule for choosing the number of bins. Sturges algorithm may sometimes
#be worse for datasets with large n, since it it gives too few bins in this case,
#which is not a worry here however. We may thus use the default option of Sturges
#algorithm for binning. In any case, as noted above log transformation should be
#used.

##--d
par(mfrow=c(1,2))
boxplot(islands, main="Original data")
boxplot(log(islands),main="Log transformed data")
```




```
#The advantage of log transformations is particularly obvious in this case.
#Since the quartiles of the original data are extremely close to one another, the
#boxplot on the original data does not deliver a very good impression of the data,
#while the log transformation remedies this concern.

##--e
par(mfrow=c(1,1))
dotchart(islands, main="Original data",cex=0.7,pch=19)
```



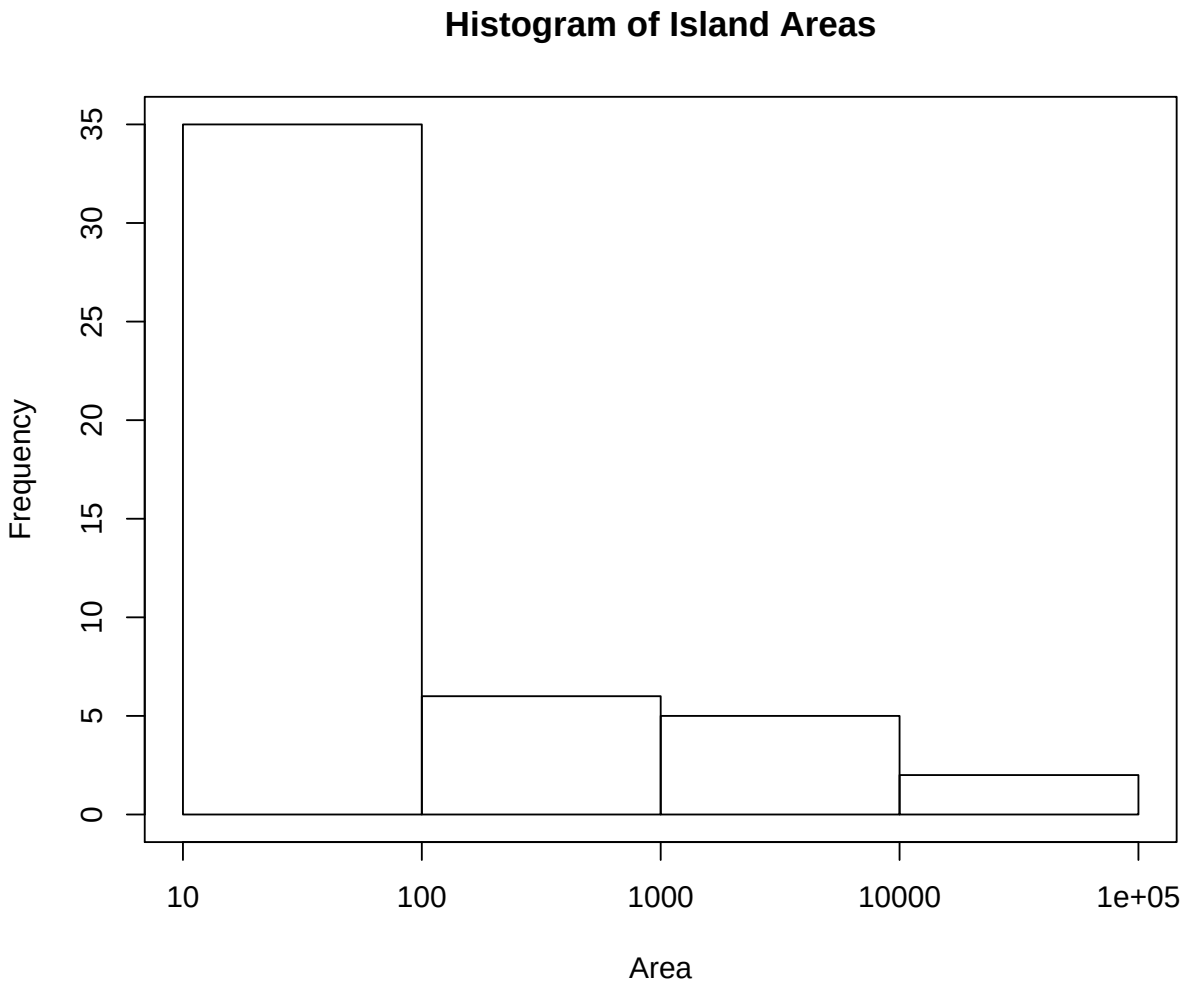

```
dotchart(log(islands),main="Log transformed data",cex=0.7,pch=19)
```




```
dev.off()

## null device
##           1

#This is again the situation where log-scale helps because it magnifies differences
#among observations. If we use the data without log-transformation, we see that most of
#the observations are very close to each other and thus the comparison using the plot
#is very difficult. For instance, it's very difficult to decide whether Spitsbergen
#has a greater area than Southampton. On the contrary, on log scale its immediately
#clear that Spitsbergen is smaller.

##--f
#In this case log-scale appears to be more appropriate than raw data, which has been
#justified above. However, the decision whether to use histogram, boxplot or dotplot
#will depend on purpose of the plot. For instance, to show the differences among
#islands it is appropriate to use dot plot, but on the contrary it would be less
#useful for studying the underlying distribution. However, bottom-line if one were to
#be forced to choose a single plot, the dot plot seems to be giving the most
#information.
```

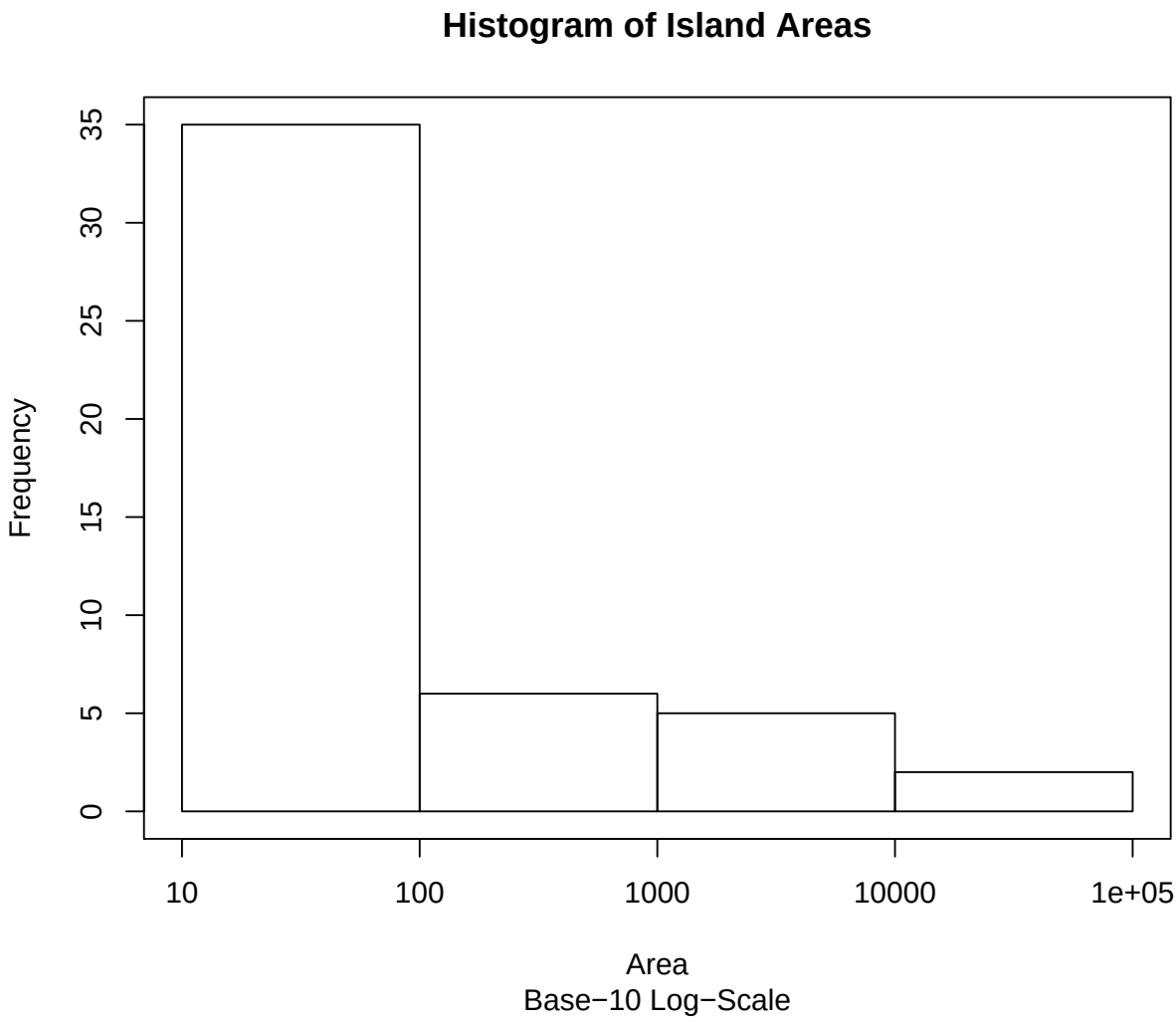
73 Exercise 84

```r
##--a
hist(log(islands, 10), # we plot histogram of logarithmically transformed data
    #(log with base 10)
     breaks = "Scott", #using Scott algorithm for binning
     axes = FALSE, # without axes,
     xlab = "Area", # with title for the x axis= "Area"
     main = "Histogram of Island Areas") # and plot title "Histogram of Island Areas"
axis(1, # we are setting labels to x axis
     at = 1 : 5, #positions where labels are displayed
     labels = 10 ^ (1 : 5)) # actual lables which will be visible
#Essentially, this step transforms the x-axis into log scale (base 10).
axis(2) #adding y-axis without any changes to labels from original data
box() #adding a black frame around the plot
```

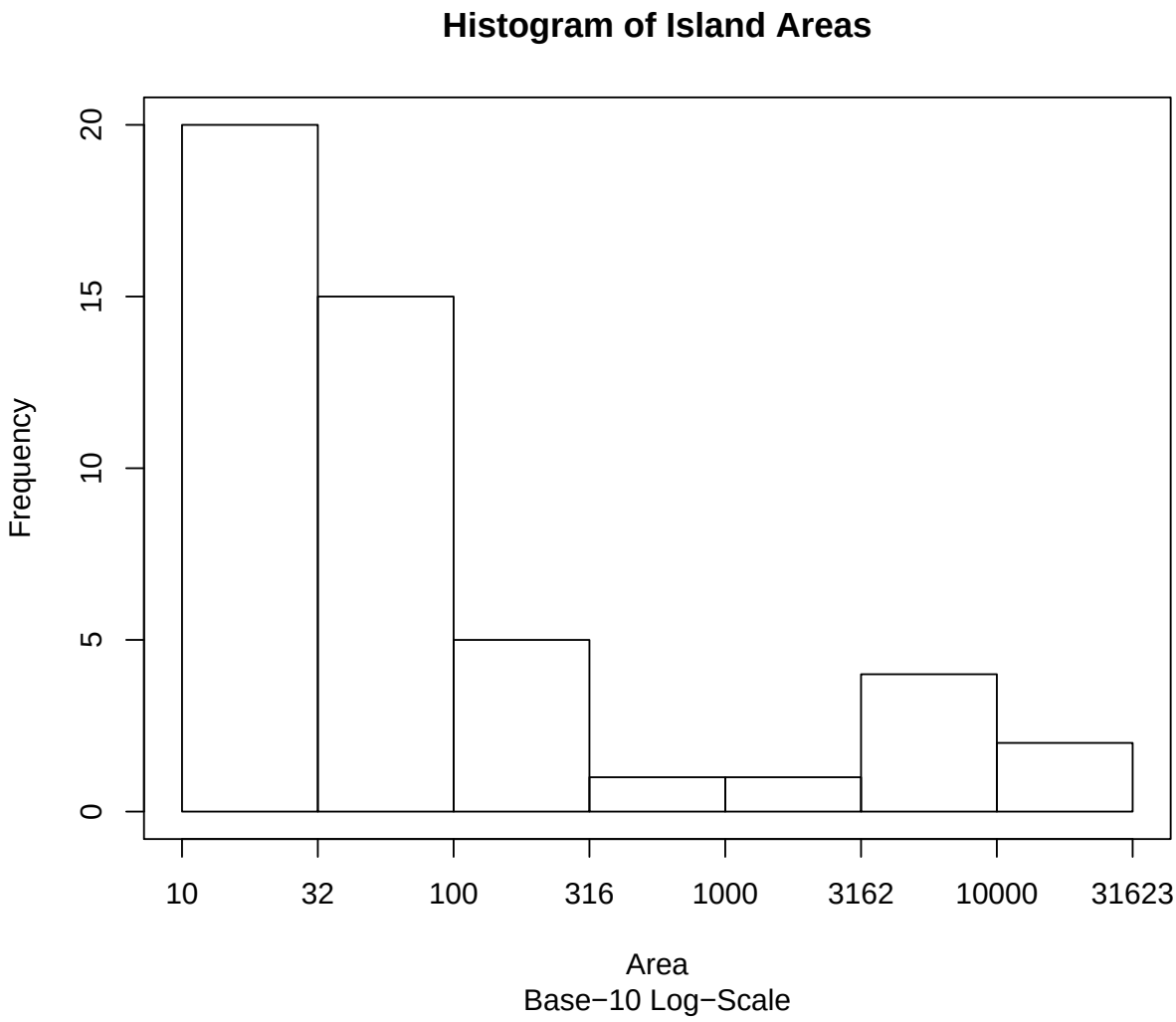



```r
##--b
hist(log(islands, 10), breaks = "Scott" ,axes = FALSE,
     sub="Base-10 Log-Scale",
```

```
     xlab = "Area", main = "Histogram of Island Areas")
axis(1, at = 1 : 5, labels = 10 ^ (1 : 5))
axis(2)
box()
```



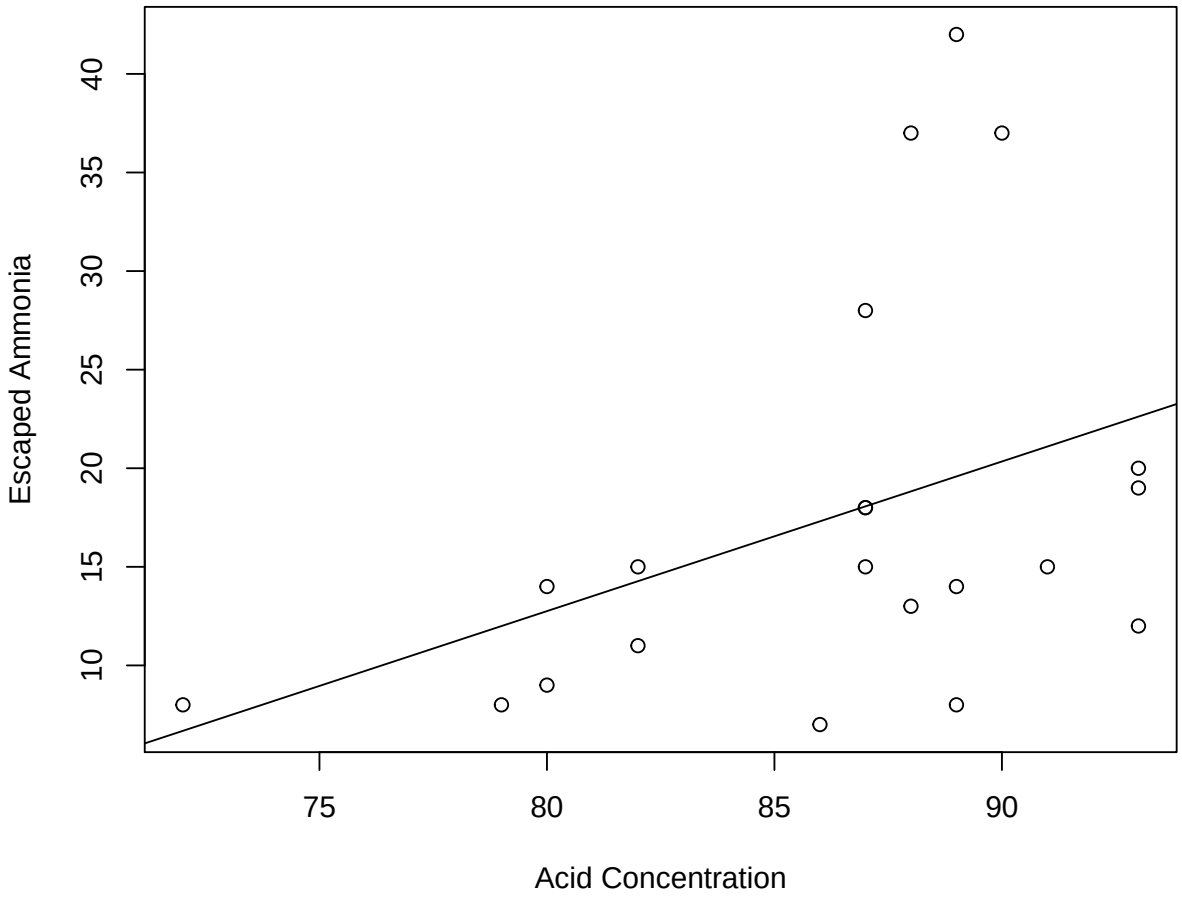

```
##--c
hist(log(islands, 10), breaks = "Sturges" ,axes = FALSE,
     sub="Base-10 Log-Scale",
     xlab = "Area", main = "Histogram of Island Areas")
#x-axis labels
axis(1, at = seq(1,4.5,0.5), labels = round( 10^seq(1,4.5,0.5)))
#y axis
axis(2)
box()
```



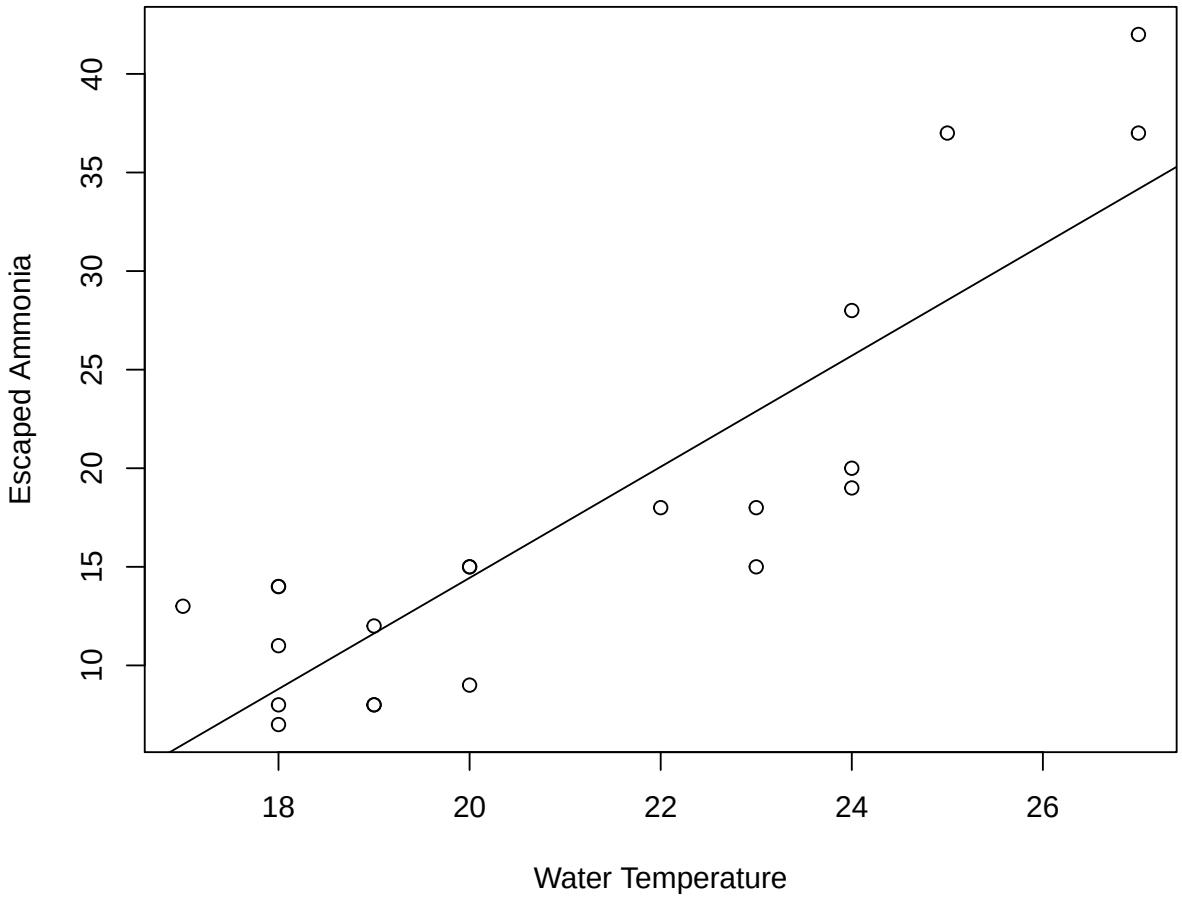
74 Exercise 85

Part (a):

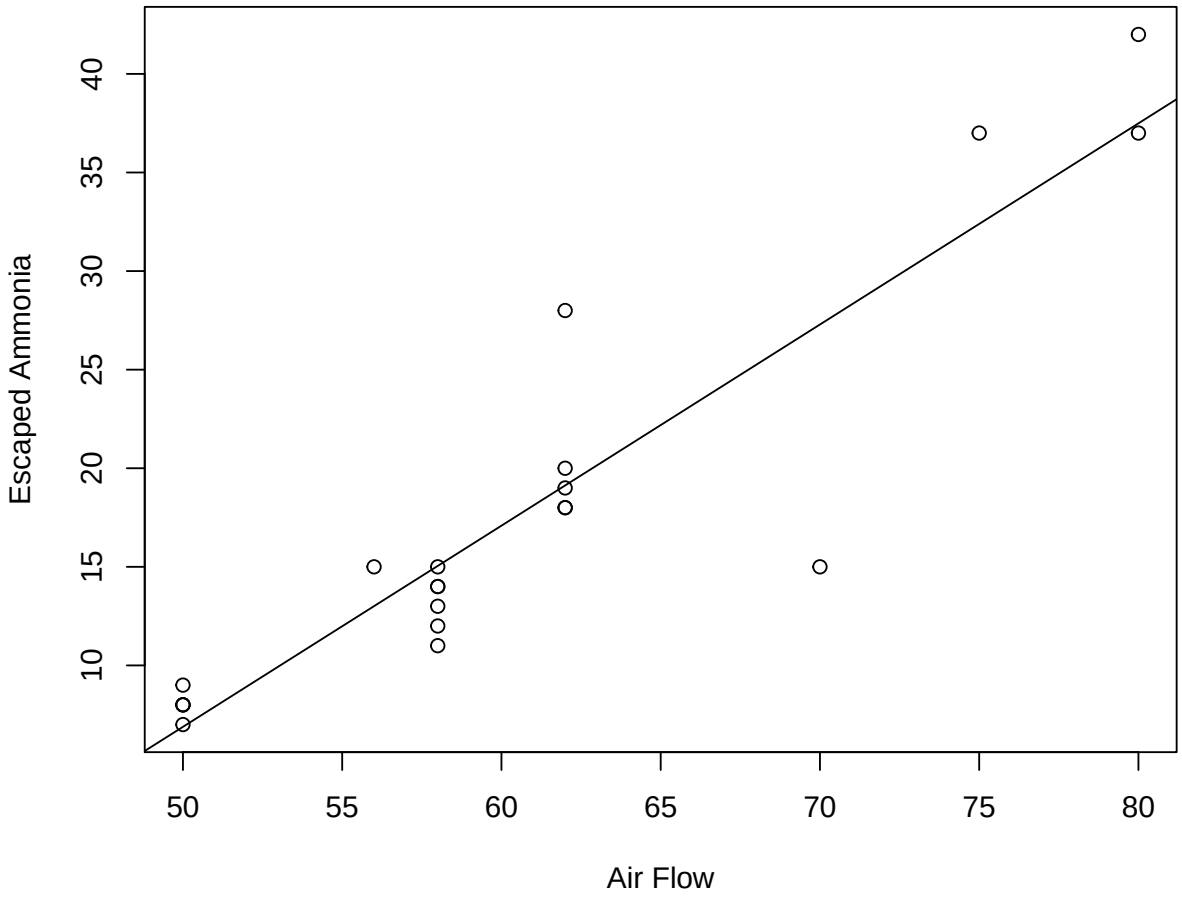
```
plot(stack.loss~Acid.Conc.,stackloss,ylab="Escaped Ammonia",xlab="Acid Concentration")
abline(lm(stack.loss~Acid.Conc.,data=stackloss))
```




```
#Relationship appears not to be linear, but like something convexly increasing
#judging on the basis of the plot alone.

plot(stack.loss~Water.Temp,stackloss,ylab="Escaped Ammonia",xlab="Water Temperature")
abline(lm(stack.loss~Water.Temp,data=stackloss))
```



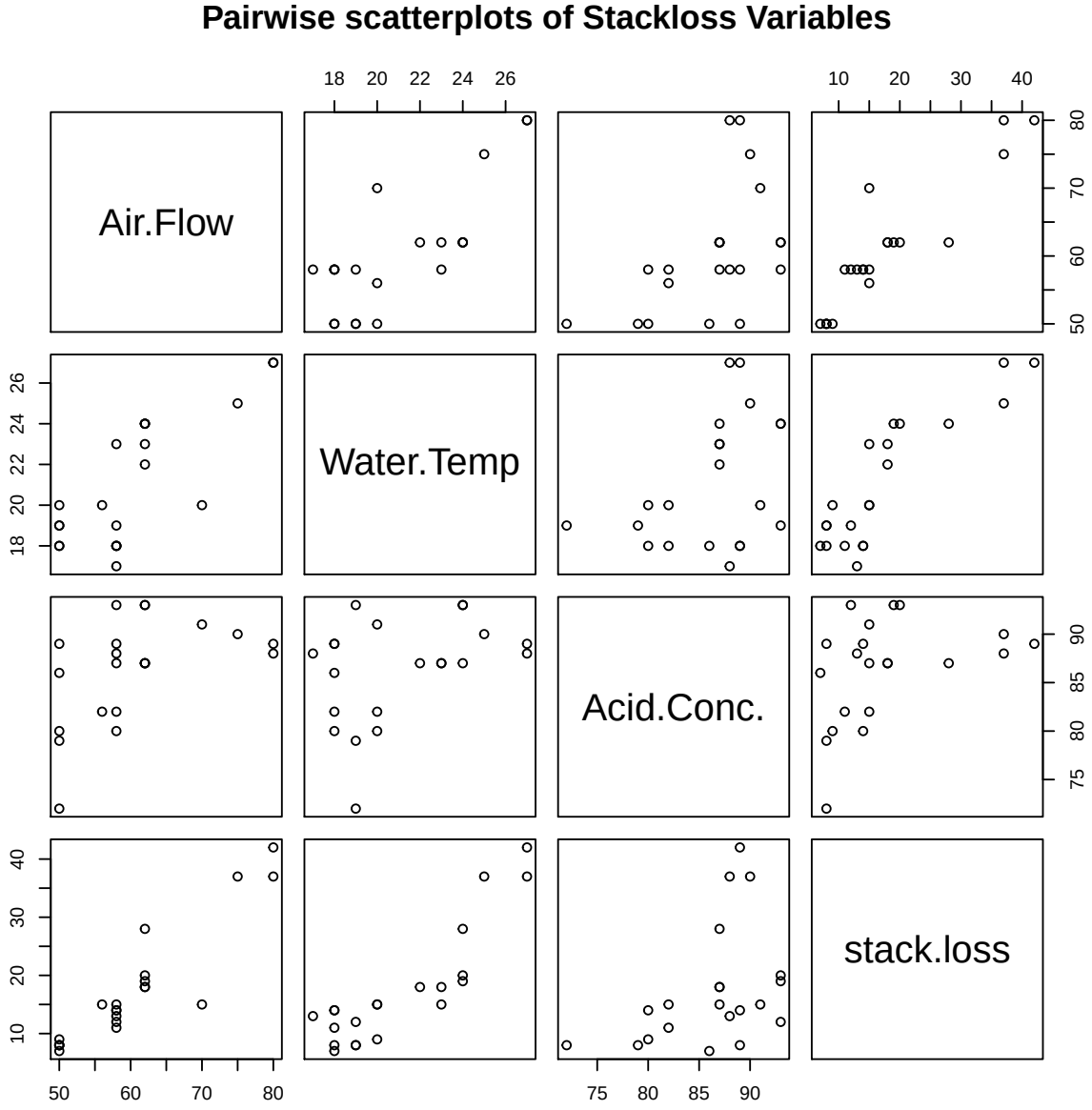

```
#Relationship appears to be close to linear judging on the basis of the plot alone.

plot(stack.loss~Air.Flow,stackloss,ylab="Escaped Ammonia",xlab="Air Flow")
abline(lm(stack.loss~Air.Flow,data=stackloss))
```




```
#Relationship appears rather linear judging on the basis of the plot alone.
```

Part (b):

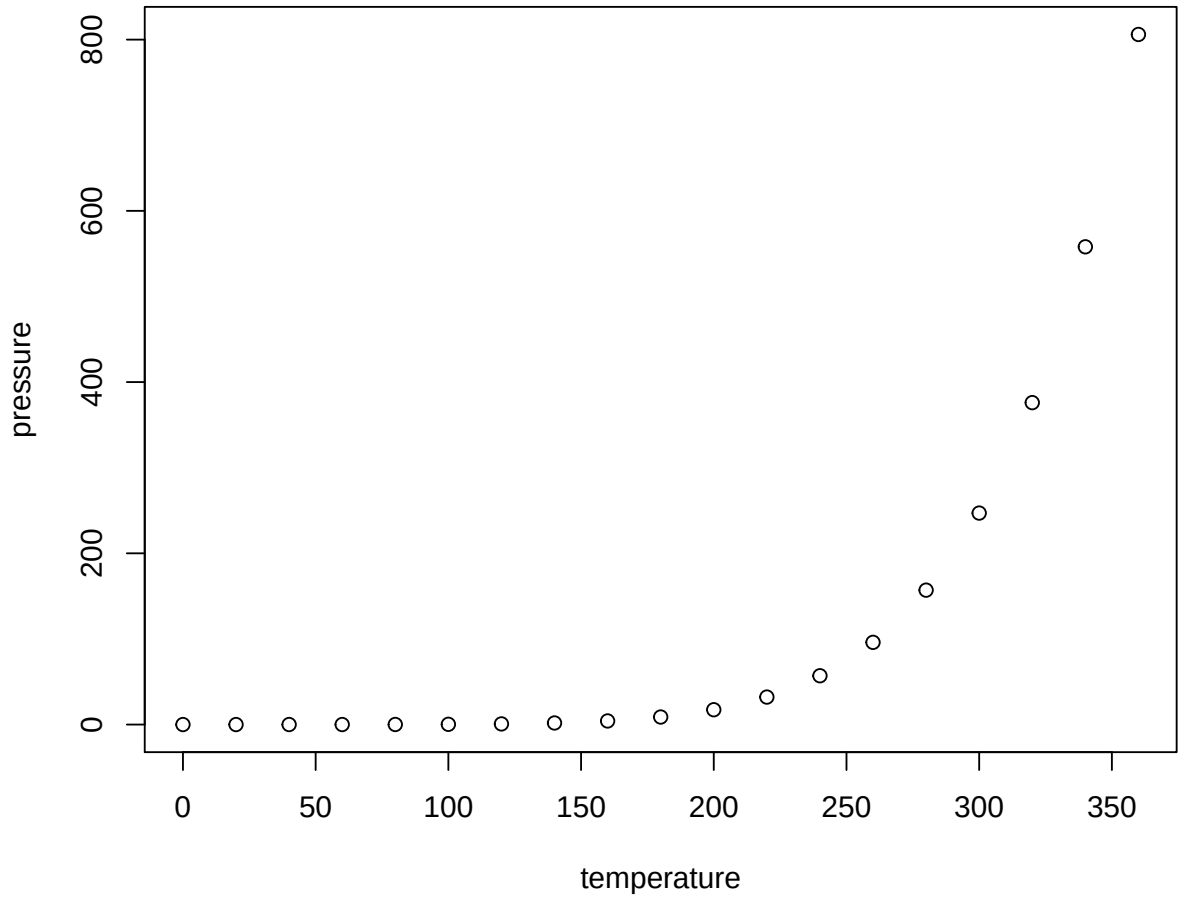
```
pairs(stackloss,main="Pairwise scatterplots of Stackloss Variables")
```



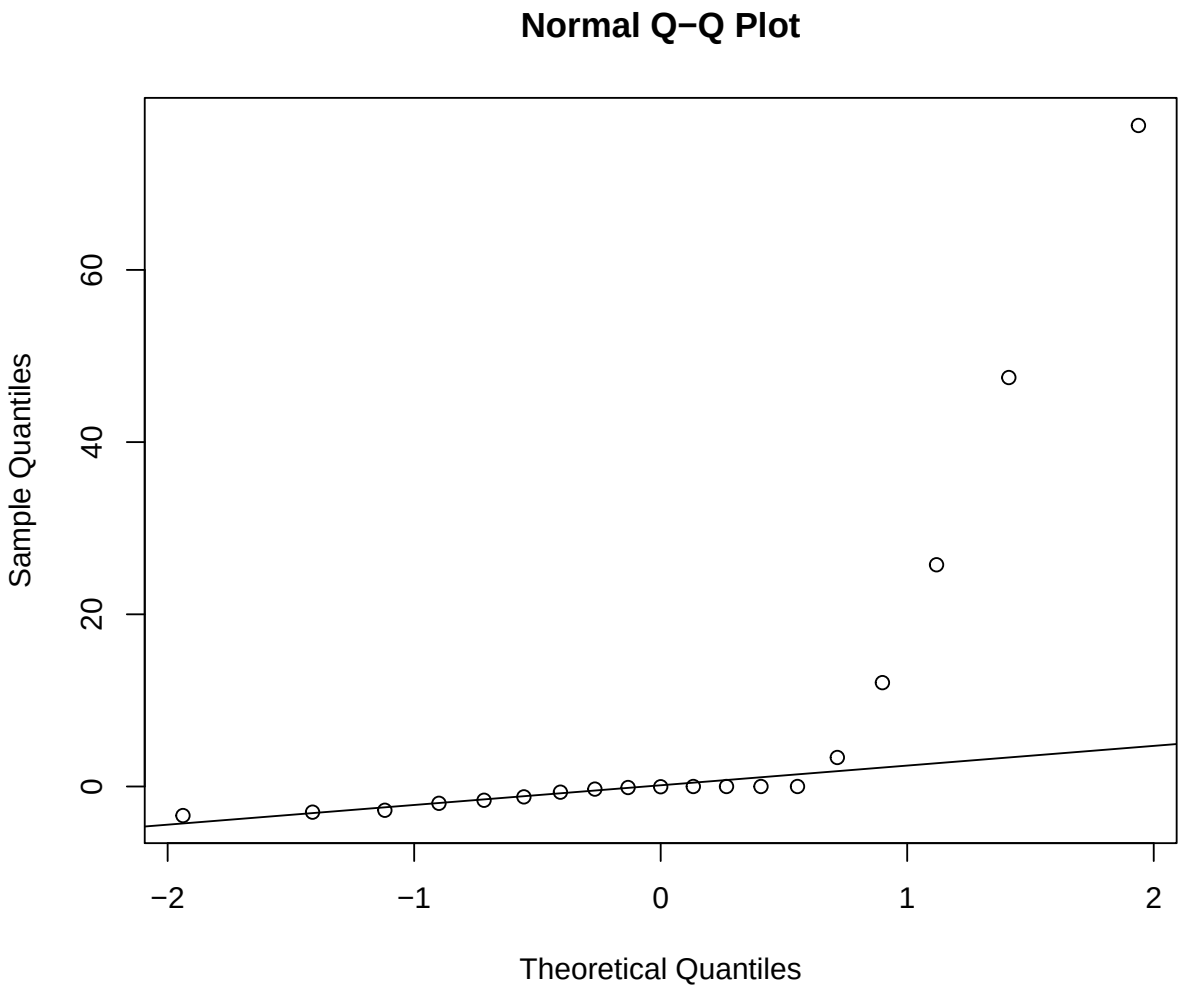

```
#Based on this graphical impression alone, the following relationships appear to be
#most linear:
#Air.Flow~Water.Temp, stack.loss~Water.Temp, stack.loss~Air.Flow

#While the following appear non-linear, or at least, not clearly linear:
#stack.loss~Acid.Conc., Water.Temp~Acid.Conc., Air.Flow~Acid.Conc.
```

75 Exercise 86

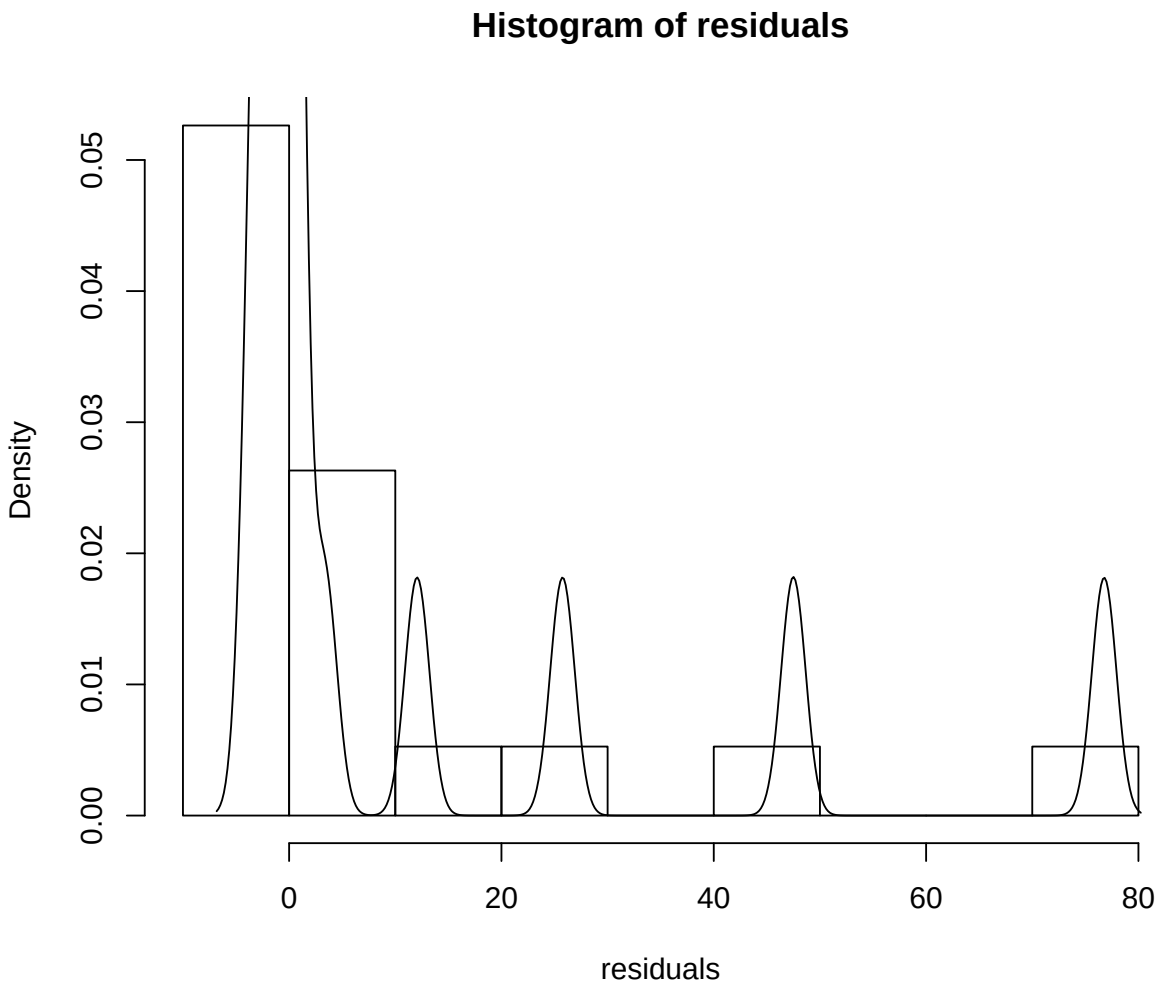
```
#Part a)
with(pressure,plot(temperature,pressure))
```



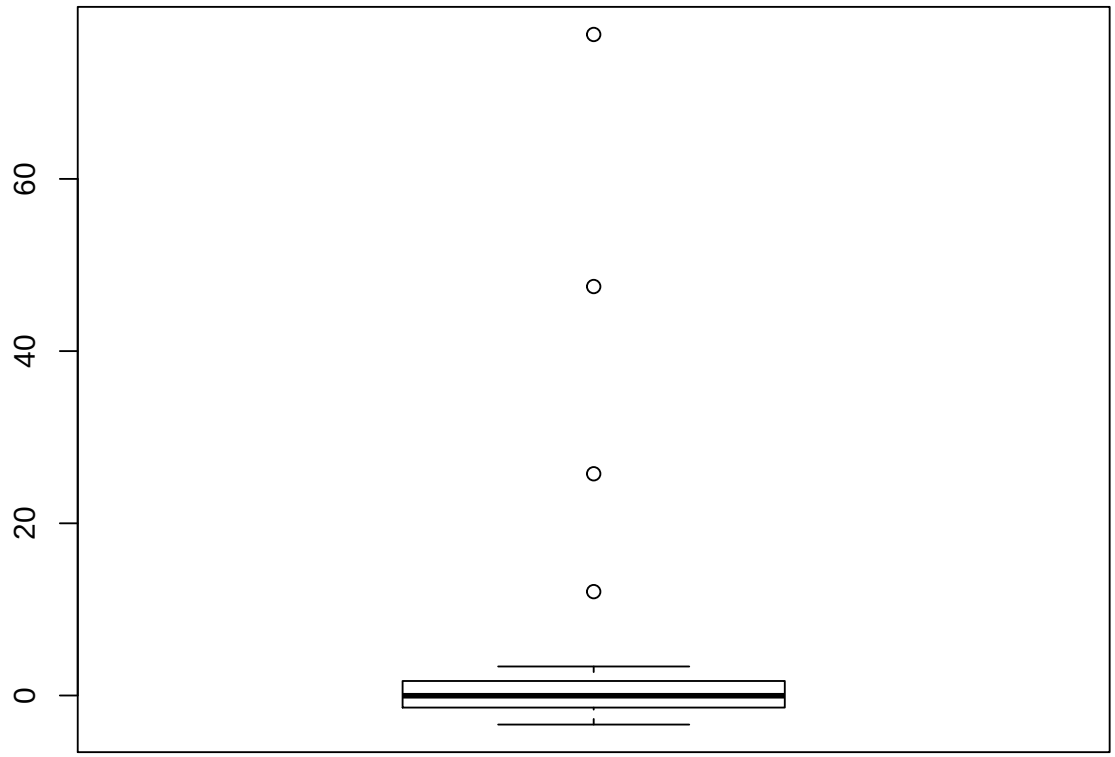

```
#The variables are definitely non-linearly related since the amount of pressure hikes up
#increasingly fast as temperature rises.

#Part b)
residuals <- with(pressure, pressure - (0.168 + 0.007 * temperature)^(20/3))
qqnorm(residuals)
qqline(residuals)
```



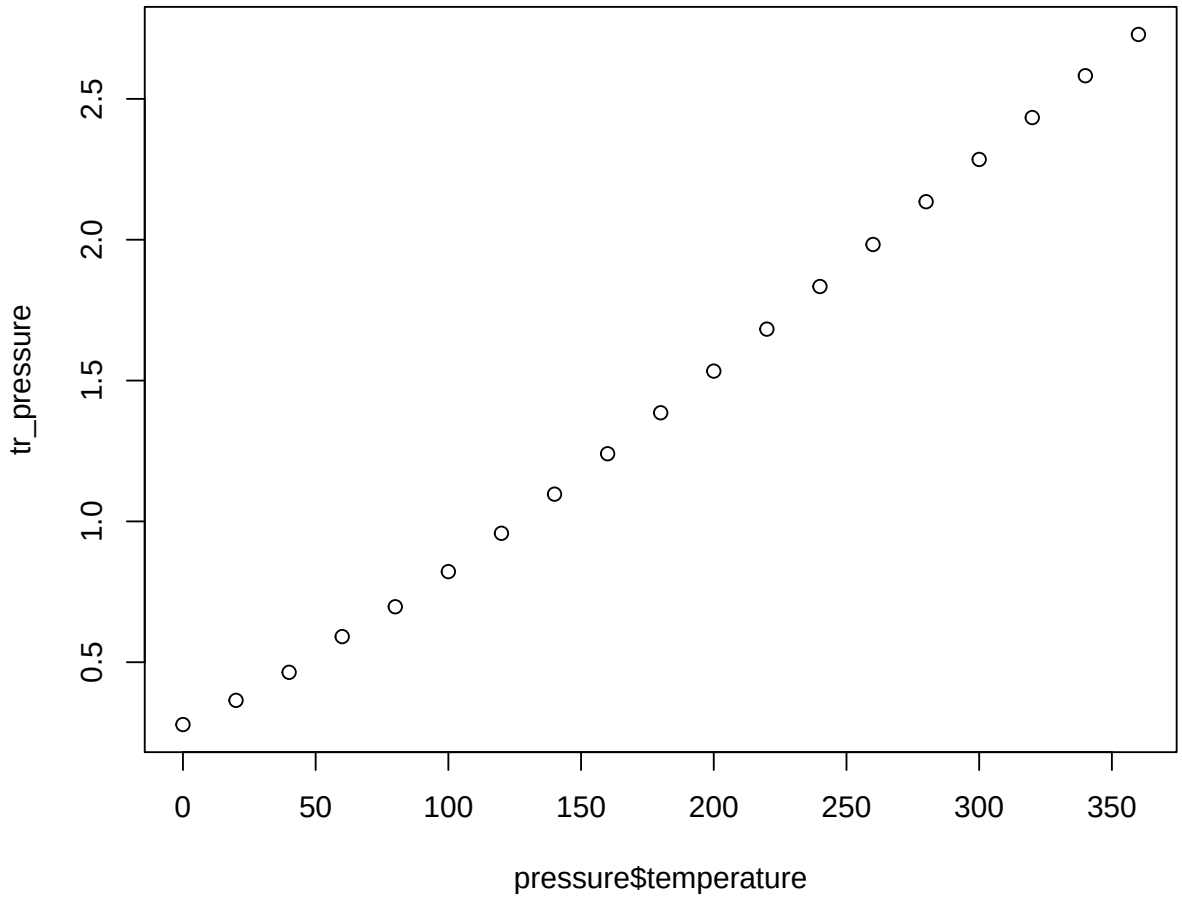

```
#We also draw the histogram of the residuals and the box/whiskers plot.
hist(residuals, probability = TRUE)
lines(density(residuals))
```



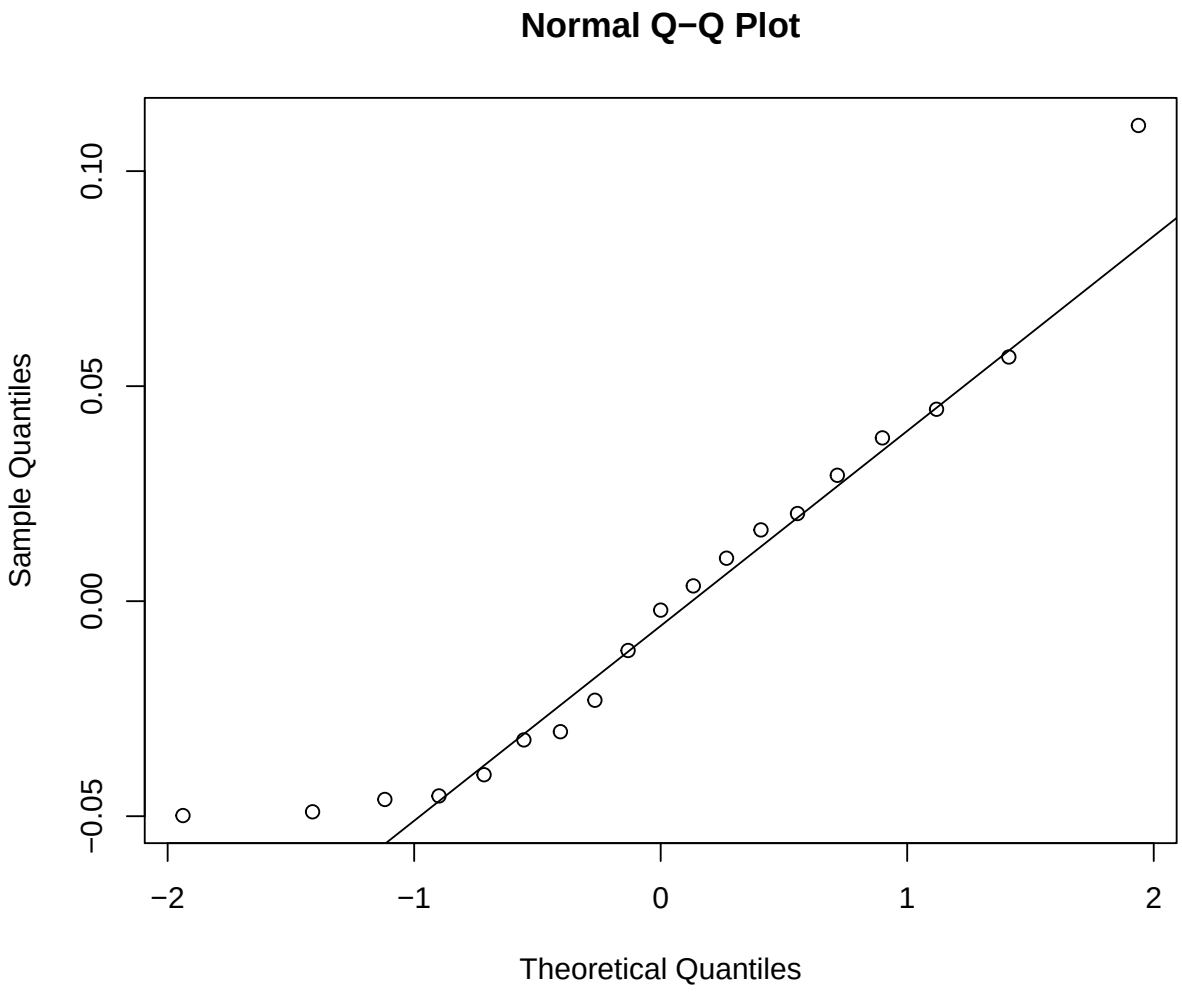

```
boxplot(residuals)
```



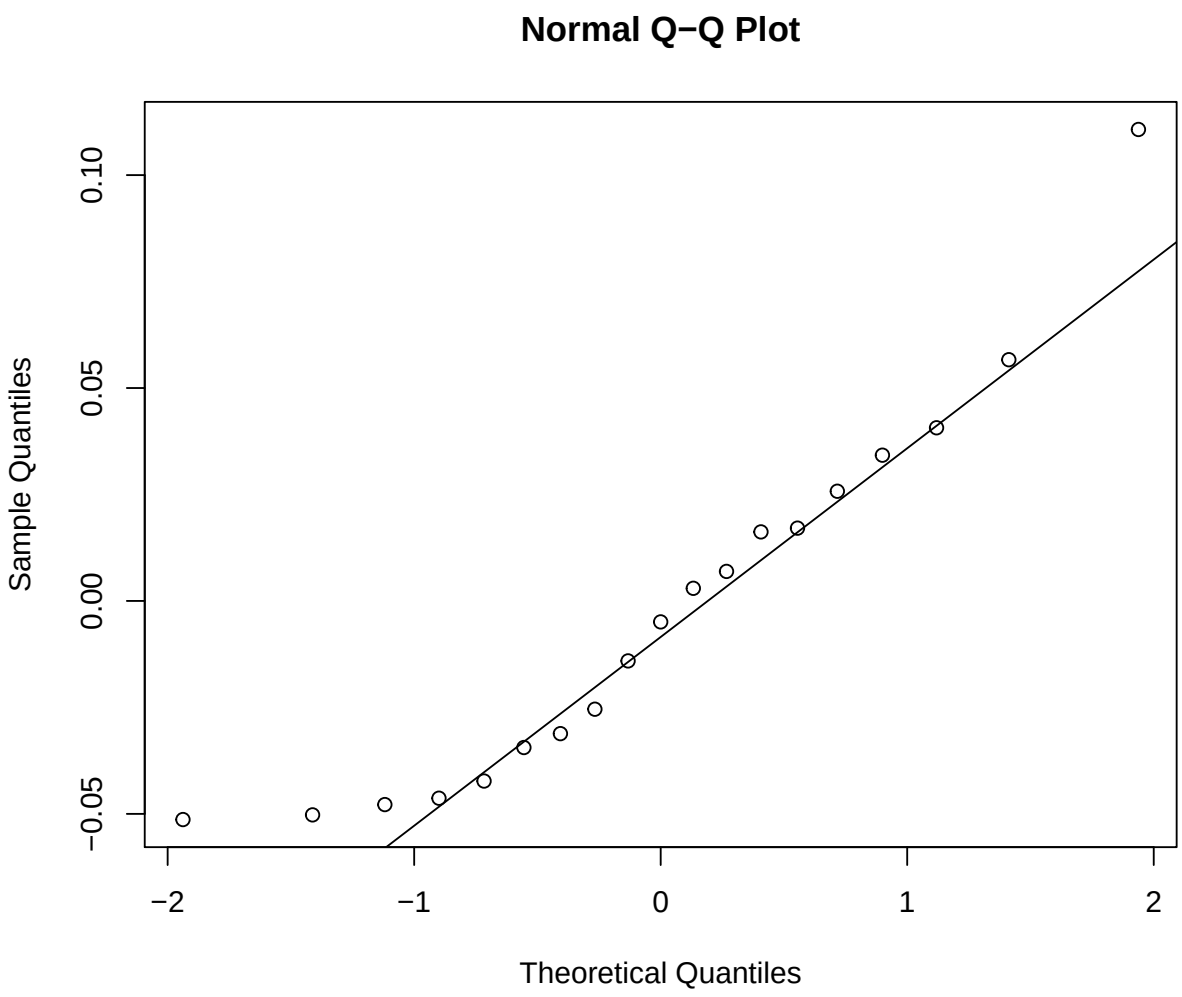

```
#The residuals are strongly skewed with a short left tail and a long right tail. This is
#easily discernible from either the QQ-plot or the box/whiskers plot.

#Part c)
tr_pressure<-with(pressure, pressure^(3/20)) #we define the transformed pressure
plot(pressure$temperature,tr_pressure)
```




```
#a close to linear relationship is evident

#Part d)
resids1<-lm(tr_pressure~pressure$temperature)$residuals #we can apply the linear
#prediction by directly calling the linear (regression) model
resids2<-with(pressure,tr_pressure-(0.168 + 0.007 * temperature))
#or by computing with the given coefficients
qqnorm(resids1)
qqline(resids1)
```



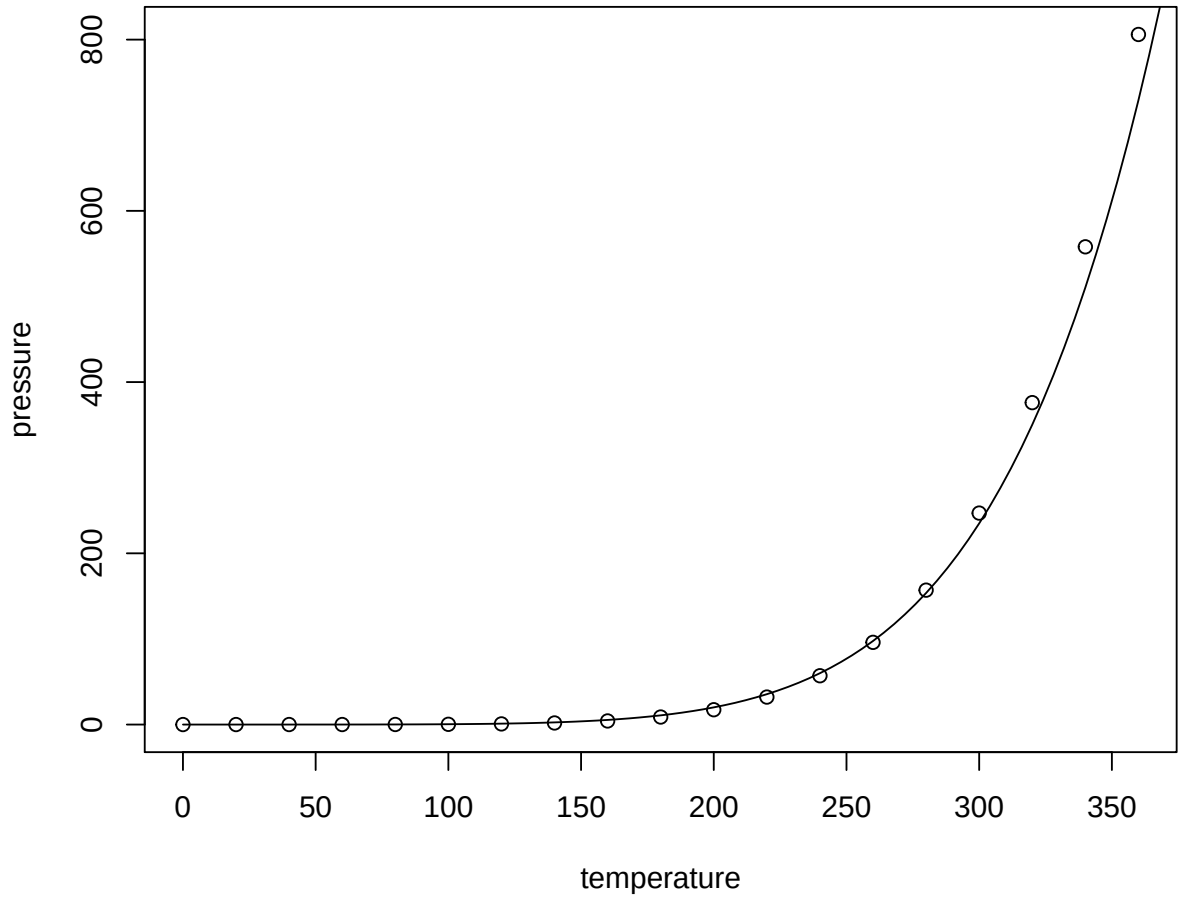

```
qqnorm(resids2)
qqline(resids2)
```



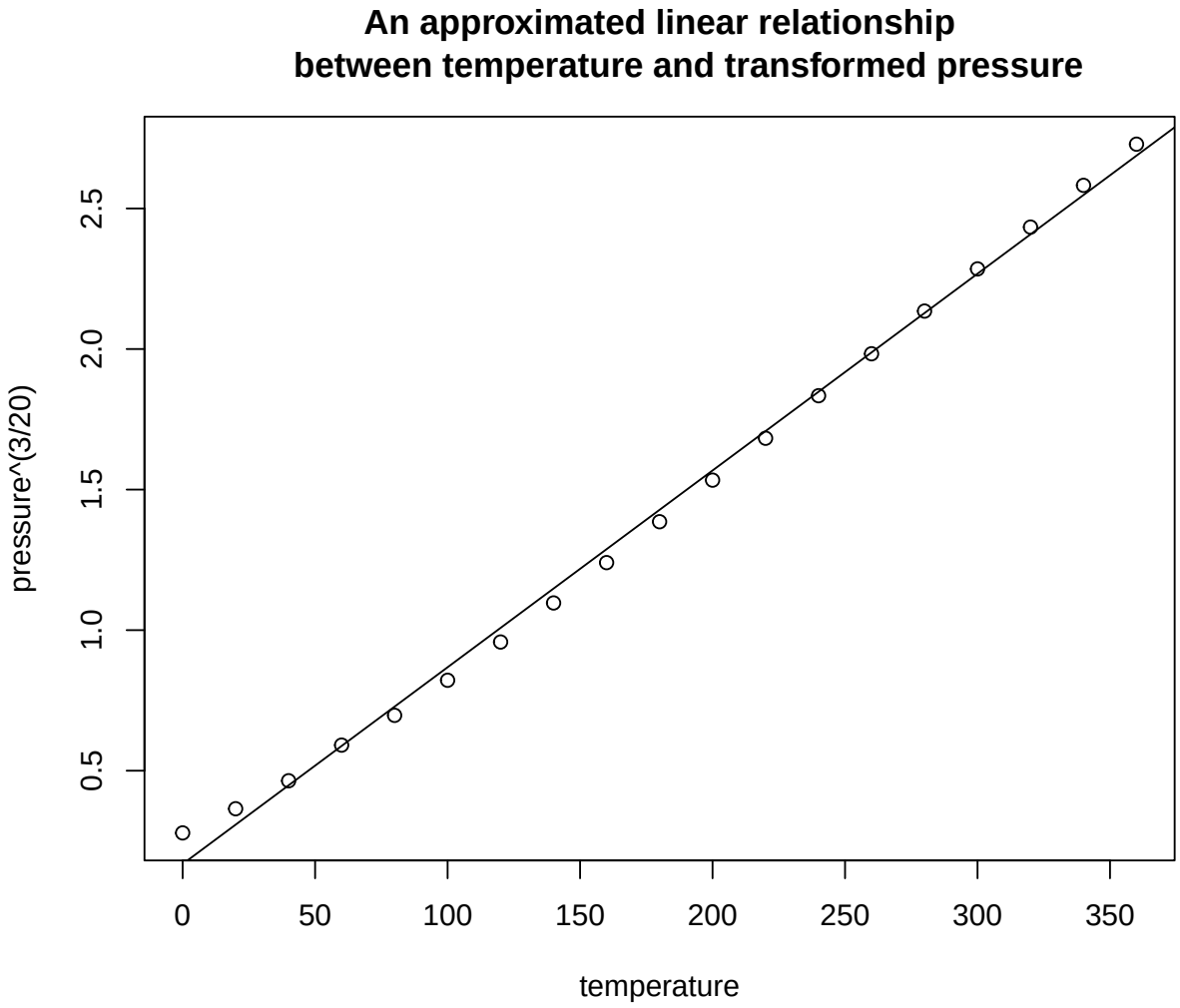

```
#there is little difference between the two set of residuals
#the residuals still don't follow a normal distribution, they show fat tails to
#(especially) the left and to the right
```

76 Exercise 87

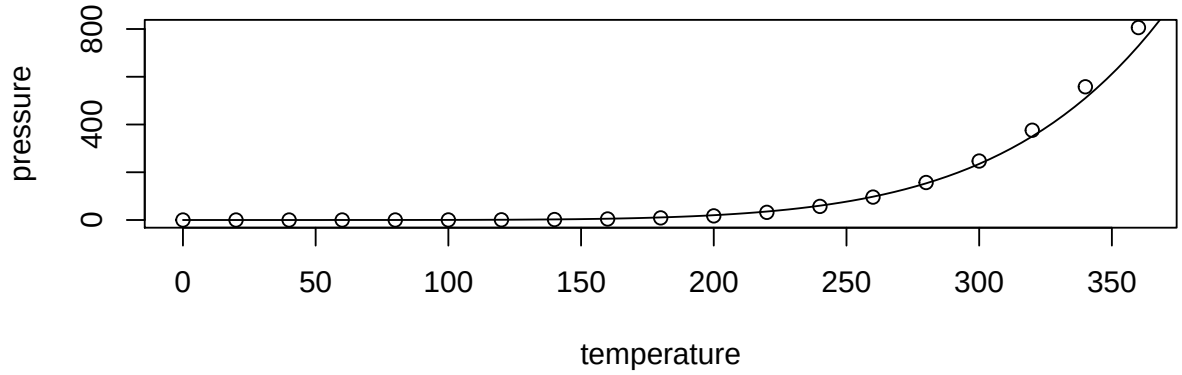
```
#Part a)
with(pressure,plot(temperature,pressure))
curve((0.168 + 0.007 * x)^(20/3), from = 0, to = 400, add = TRUE)
```

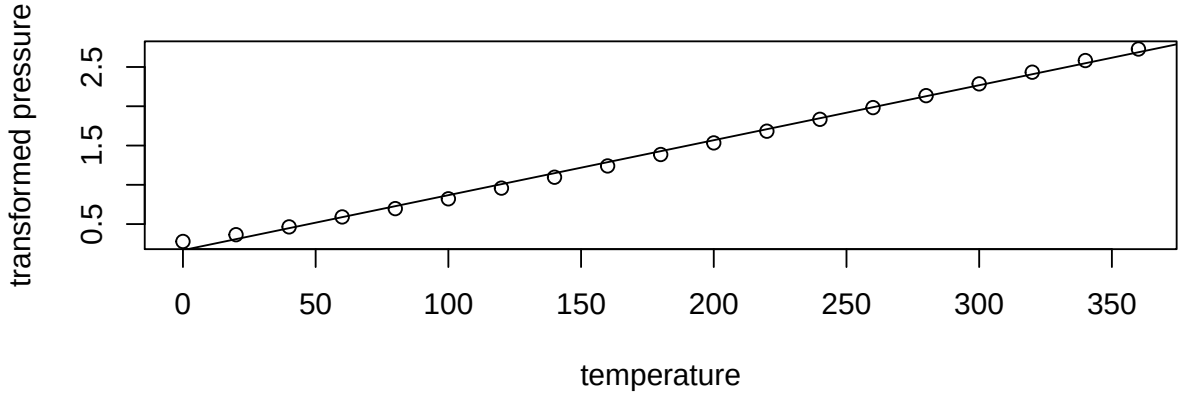



```
#Part b)
with(pressure,plot(temperature,pressure^(3/20)))
#relationship is clearly linear
abline(a=.168,b=0.007)

#Part c)
title('An approximated linear relationship\
      between temperature and transformed pressure')
```

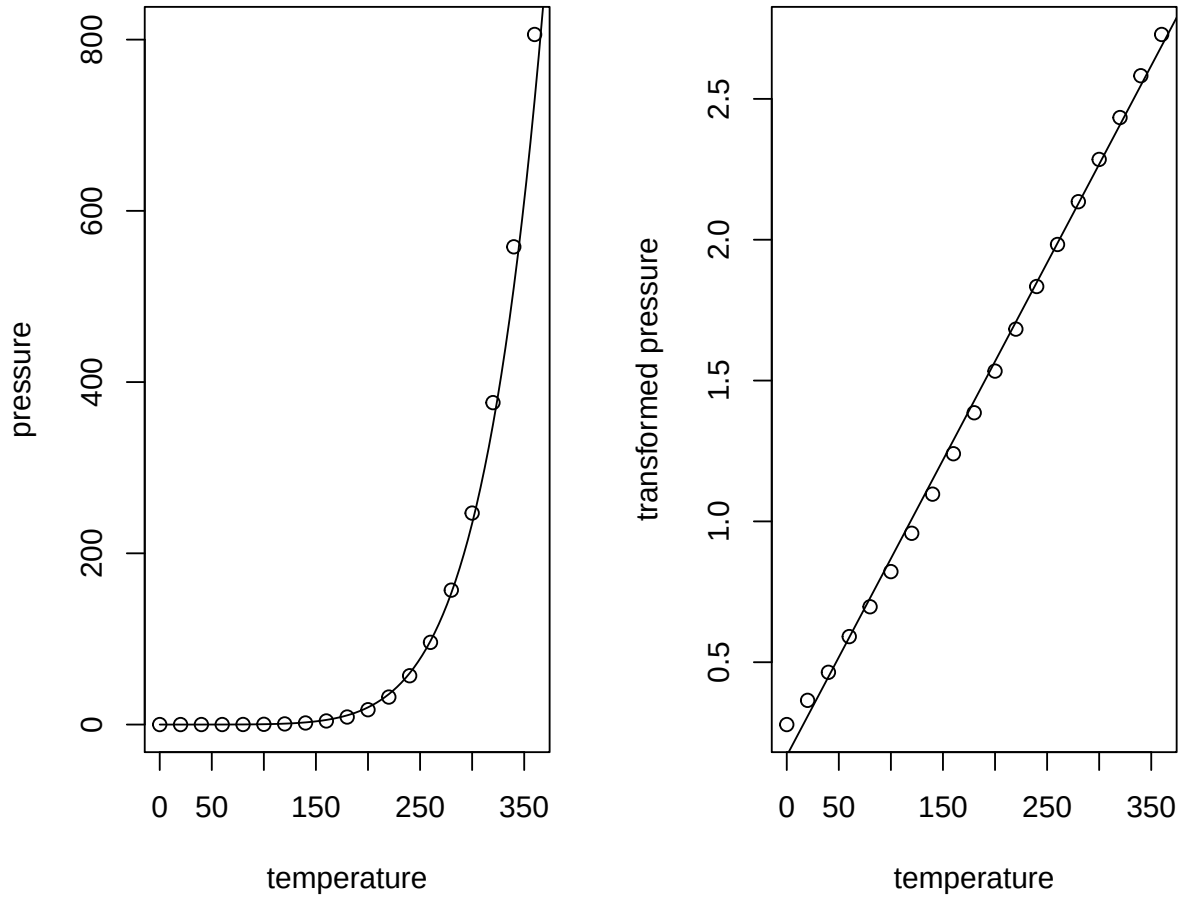



```
# Part d)
op<-par(mfrow = c(2,1))
with(pressure,plot(temperature,pressure))
curve((0.168 + 0.007 * x)^(20/3), from = 0, to = 400, add = TRUE)
with(pressure,plot(temperature,pressure^(3/20),ylab='transformed pressure'))
abline(a=.168,b=0.007)
```






```
op<-par(mfrow = c(1,2))
with(pressure,plot(temperature,pressure))
curve((0.168 + 0.007 * x)^(20/3), from = 0, to = 400, add = TRUE)
with(pressure,plot(temperature,pressure^(3/20),ylab='transformed pressure'))
abline(a=.168,b=0.007)
```



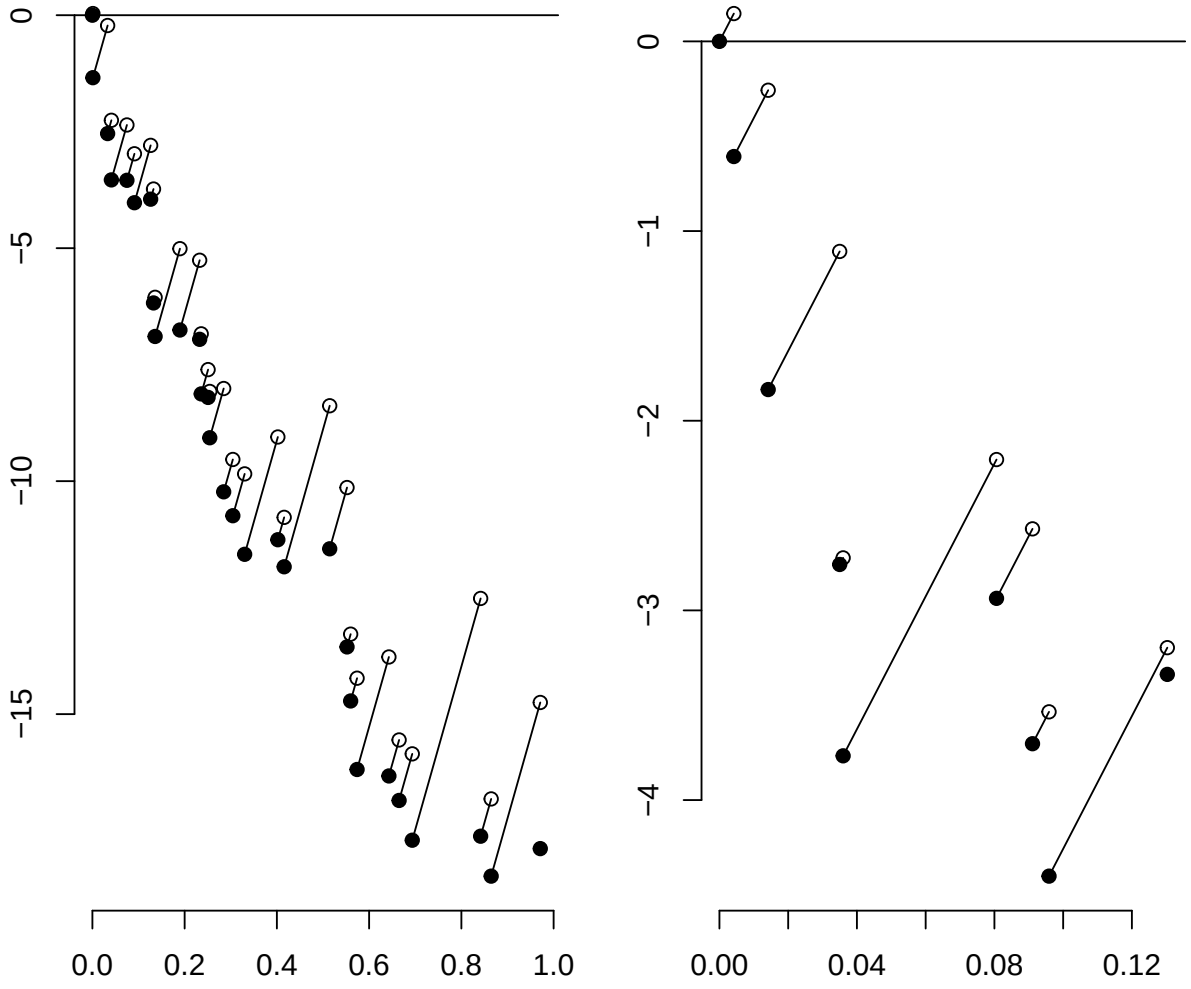
77 Exercise 88

We assume that the inputs for this function are two sequences and a single number representing the premium rate per year as before (exercise 79): the first sequence are points in time, i.e. (usually) zero and the times, at which the claims occur, subsequently. The second sequence is a sequence of money amounts, where the first one is the amount of money the company starts with and all subsequent amounts are money amounts at the point claim times, instantly **after** the money has been paid out (i.e. the filled points in the example graphs in the task). We therefore fill the sequence of x- and y-values by repeating each claim time twice for x and adding the amounts instantly before the claim outpayments (hollow points in the example graphs in the task) for y. The rest is then just about passing some appropriate plot options to the plot of these sequences.

```
plotinsurance<-function(timepoints,money,premiumrate){
  xvalues<-c(0,rep(timepoints[-1],each=2))
  yvalues<-numeric(length(xvalues))
  yvalues[1]<-money[1]
  claimindex<-2
  for(i in 2:length(yvalues)){
    if(i%%2==1){
      #Fill yvalues of points after outpayment of the claim
```

```
      yvalues[i]<-money[claimindex]
      claimindex<-claimindex+1
    }else{
      #Fill yvalues of points before outpayment of the claim
      yvalues[i]<-premiumrate*xvalues[i]+(yvalues[i-1]-premiumrate*xvalues[i-1])
    }
  }
  par(mar=c(2,2,2,2))
  plot(c(0,xvalues[length(xvalues)]),c(min(yvalues),max(yvalues)),
       type="n",xlab=NULL,ylab=NULL,bty="n",xlim=c(0,max(timepoints)))
  for(i in seq_along(timepoints)){
    segments(timepoints[i],yvalues[2*i-1],timepoints[i+1],yvalues[2*i])
  }
  points(xvalues[seq(2,length(xvalues)-1,by=2)],
         yvalues[seq(2,length(xvalues)-1,by=2)])
  points(xvalues[seq(1,length(xvalues),by=2)],
         yvalues[seq(1,length(xvalues),by=2)],pch=19)
  abline(0,0)
}

#We plot an example using the function created in the last unit for the
#auto insurance task in exercise 79, for which we have to adapt the output
#ever so slightly in order to fit the specified inputs of the plot function.

#Reinstate the necessary functions from exercise 79:
rpp1 <- function(ttime, lambda){
  sort(runif(rpois(1, ttime * lambda), max = ttime))
}

ex_79<-function(ttime,lambda,premiumrate=105,plotoption=TRUE){
  claimtimes<-rpp1(ttime,lambda)
  claimnumber<-length(claimtimes)
  #Claims are Gamma distributed:
  claims<-rgamma(claimnumber,shape=2,rate=2)
  #Create sequence of how the available financial resources develop
  #through time. At each claim time, we have two points: one with the available
  #resources before and one with them after the claim:
  xvalues<-c(0,rep(claimtimes,each=2),ttime)
  yvalues<-premiumrate*xvalues
  claimindex<-1
  for(i in 3:length(yvalues)){
    if(i%%2==1){
      #Fill yvalues of points after outpayment of the claim
      yvalues[i]<-yvalues[i-1]-claims[claimindex]
      claimindex<-claimindex+1
    }else{
      #Fill yvalues of points before outpayment of the claim
      yvalues[i]<-premiumrate*xvalues[i]+(yvalues[i-1]-premiumrate*xvalues[i-1])
    }
  }
  if(plotoption){
  #Create plot
  plot(xvalues,yvalues,type="l",main="Financial resources over one year",
       ylab="Money available",xlab="Time")
  abline(0,0)
  }
  #Return list of important simulation outputs
  money<-cbind(xvalues,yvalues)
```

```
  colnames(money)<-c("t","Money")
  list(claimtimepoints=claimtimes,claim_amounts=claims,moneyovertime=money)
}

#Create an example and plot for t=1 and t=0.2 as in the example given in the task:
premiumrate<-10
example_data<-ex_79(1,25,premiumrate,plotoption = FALSE)$moneyovertime
example_data<-example_data[-seq(2,length(example_data),by=2)]
example_data<-matrix(example_data,length(example_data)/2,2)
colnames(example_data)<-c("t","Money")

premiumrate<-35
example_data2<-ex_79(0.2,55,premiumrate,plotoption = FALSE)$moneyovertime
example_data2<-example_data2[-seq(2,length(example_data2),by=2)]
example_data2<-matrix(example_data2,length(example_data2)/2,2)
colnames(example_data2)<-c("t","Money")
op<-par(mfrow = c(1,2))
plotinsurance(example_data[,1],example_data[,2],premiumrate)
plotinsurance(example_data2[,1],example_data2[,2],premiumrate)
```



```
dev.off()

## null device
##           1
```

78 Exercise 89

```
library(quantmod)
djmembers<-c("CSCO","UNH","MSFT","PFE","DIS","CVX","MRK","WMT","JNJ","XOM","TRV","AAPL","KO",
             "VZ","BA","INTC","MCD","MMM","UTX","PG","HD","AXP","GS","GE","IBM","NKE")
N<-length(djmembers)
DJIA<-new.env()
for(i in 1:N){
  getSymbols(djmembers[i],env=DJIA,src="yahoo",from=as.Date("2008-01-01"),to=as.Date("2018-01-01"))
}

## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0.  You will still be able to use
## 'loadSymbols' to automatically load data.  getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).

pricedata<-merge(DJIA$CSCO[,6],DJIA$UNH[,6])
for(i in 3:N){
  pricedata<-merge(pricedata,get(djmembers[i],envir=DJIA)[,6])

}

#Daily frequency:
dailylogreturns<-periodReturn(pricedata[,1],period="daily",type="log")
for(i in 2:N){
  dailylogreturns<-merge(dailylogreturns,periodReturn(pricedata[,i],period="daily",type="log"))
}
dailylogreturns<-dailylogreturns[-1,]
colnames(dailylogreturns)<-djmembers

library(moments)
library(tseries)
momentsmatrix_d<-matrix(0,N,4)
rownames(momentsmatrix_d)<-djmembers
colnames(momentsmatrix_d)<-c("Skewness","Kurtosis","Test stat JB","p-value JB")
for(i in 1:N){
  momentsmatrix_d[i,1]<-skewness(dailylogreturns[,i])
  momentsmatrix_d[i,2]<-kurtosis(dailylogreturns[,i])
  momentsmatrix_d[i,3]<-jarque.bera.test(dailylogreturns[,i])$statistic
  momentsmatrix_d[i,4]<-jarque.bera.test(dailylogreturns[,i])$p.value
}
momentsmatrix_d

##          Skewness  Kurtosis Test stat JB p-value JB
## CSCO -0.442994932 15.835146    17359.534          0
## UNH   0.470104045 29.572236    74143.240          0
## MSFT  0.144281744 13.025262    10549.286          0
## PFE  -0.001700682  9.473716     4395.207          0
## DIS   0.165564322 11.788502     8111.810          0
```

```
## CVX   0.160066675 17.571430    22278.496          0
## MRK  -0.438897495 14.844468    14793.871          0
## WMT   0.151231686 14.653899    14253.020          0
## JNJ   0.469021629 15.227202    15771.564          0
## XOM   0.143057851 19.924529    30048.947          0
## TRV   0.330285895 29.431554    73314.275          0
## AAPL -0.527744306 11.351377     7431.395          0
## KO    0.435469356 16.028821    17882.102          0
## VZ    0.368758230 12.276811     9082.506          0
## BA    0.029253967  8.338213     2988.931          0
## INTC -0.125605136  8.685995     3397.284          0
## MCD   0.095073303  9.696644     4706.915          0
## MMM  -0.118953441  8.810655     3546.905          0
## UTX   0.182666076 10.701927     6235.149          0
## PG   -0.195253501 10.918159     6591.366          0
## HD    0.433029440  8.989181     3840.559          0
## AXP   0.022945098 14.957581    14995.642          0
## GS    0.313441353 19.280651    27839.339          0
## GE    0.018500283 13.767771    12159.865          0
## IBM  -0.085774996  9.269005     4124.718          0
## NKE   0.277812270 10.990489     6728.426          0

#Clearly, the null hypotheses of normality is strongly rejected at all reasonable
#significance levels. Specifically, the excess kurtosis is very noticeable among
#all assets in the DJIA, indicating the existence of heavy tails.

#Weekly frequency:
weeklylogreturns<-periodReturn(pricedata[,1],period="weekly",type="log")
for(i in 2:N){
  weeklylogreturns<-merge(weeklylogreturns,periodReturn(pricedata[,i],period="weekly",type="log"))
}
weeklylogreturns<-weeklylogreturns[-1,]
colnames(weeklylogreturns)<-djmembers

momentsmatrix_w<-matrix(0,N,4)
rownames(momentsmatrix_w)<-djmembers
colnames(momentsmatrix_w)<-c("Skewness","Kurtosis","Test stat JB","p-value JB")
for(i in 1:N){
  momentsmatrix_w[i,1]<-skewness(weeklylogreturns[,i])
  momentsmatrix_w[i,2]<-kurtosis(weeklylogreturns[,i])
  momentsmatrix_w[i,3]<-jarque.bera.test(weeklylogreturns[,i])$statistic
  momentsmatrix_w[i,4]<-jarque.bera.test(weeklylogreturns[,i])$p.value
}
momentsmatrix_w

##        Skewness  Kurtosis Test stat JB p-value JB
## CSCO -0.7065037  6.776195     352.8959          0
## UNH  -0.4931898 18.107351    4975.6588          0
## MSFT -0.5265971  7.136825     395.5809          0
## PFE  -0.9053013 11.850685    1771.6803          0
## DIS  -0.5622635  8.633314     716.3488          0
## CVX  -1.4531644 16.900220    4377.7649          0
## MRK  -0.4671728  8.205169     607.1124          0
## WMT  -0.8008928  7.683664     531.9069          0
## JNJ  -0.8619084 12.586552    2059.5460          0
## XOM  -1.0437850 10.449396    1299.2753          0
## TRV  -0.3675518 13.159614    2252.4164          0
## AAPL -0.7883814  7.259014     447.7427          0
## KO   -1.8223308 20.003316    6564.5199          0
## VZ    0.3677631  7.576061     466.3240          0
## BA   -0.5355364  7.328524     431.6336          0
## INTC  0.1439039  5.354053     122.0964          0
## MCD  -0.2061593  5.712863     163.4558          0
## MMM  -0.4022857  7.339915     422.9260          0
## UTX  -0.3270974  5.531213     148.3767          0
## PG   -0.8645412 10.513820    1290.5000          0
## HD    0.1981453  9.384437     888.2635          0
## AXP  -0.3555555 10.320143    1174.2074          0
```

```
## GS    0.2741915 14.902520    3081.9474          0
## GE    0.2651188 10.706105    1295.2323          0
## IBM  -0.3493471  7.287535     409.6607          0
## NKE  -0.2011296  6.730802     305.6685          0

#Still, the null hypothesis of normality is strongly rejected at all reasonable
#significance levels, although the test statistics generally seem to be decreasing
#slightly.

#Monthly frequency:
monthlylogreturns<-periodReturn(pricedata[,1],period="monthly",type="log")
for(i in 2:N){
  monthlylogreturns<-merge(monthlylogreturns,periodReturn(pricedata[,i],period="monthly",type="log"))
}
monthlylogreturns<-monthlylogreturns[-1,]
colnames(monthlylogreturns)<-djmembers

momentsmatrix_m<-matrix(0,N,4)
rownames(momentsmatrix_m)<-djmembers
colnames(momentsmatrix_m)<-c("Skewness","Kurtosis","Test stat JB","p-value JB")
for(i in 1:N){
  momentsmatrix_m[i,1]<-skewness(monthlylogreturns[,i])
  momentsmatrix_m[i,2]<-kurtosis(monthlylogreturns[,i])
  momentsmatrix_m[i,3]<-jarque.bera.test(monthlylogreturns[,i])$statistic
  momentsmatrix_m[i,4]<-jarque.bera.test(monthlylogreturns[,i])$p.value
}
momentsmatrix_m

##         Skewness  Kurtosis Test stat JB   p-value JB
## CSCO -0.38897250  3.737406    5.6969574 5.793239e-02
## UNH  -1.73022590  9.056658  241.2617369 0.000000e+00
## MSFT -0.33033132  3.523845    3.5248215 1.716306e-01
## PFE  -0.49506923  3.997114    9.7907734 7.481016e-03
## DIS  -0.60101771  3.957786   11.7127900 2.861541e-03
## CVX  -0.14779248  3.094207    0.4772165 7.877234e-01
## MRK  -0.35751192  3.690609    4.8998276 8.630102e-02
## WMT  -0.24680449  4.401104   10.9417634 4.207521e-03
## JNJ  -0.47608205  4.213606   11.7981369 2.741998e-03
## XOM  -0.16951671  2.830332    0.7126659 7.002394e-01
## TRV  -0.01235353  4.196973    7.1070528 2.862352e-02
## AAPL -1.05288590  6.817288   94.2378876 0.000000e+00
## KO   -0.49491963  4.968921   24.0797990 5.903888e-06
## VZ   -0.01752122  2.571324    0.9172472 6.321531e-01
## BA   -0.85838304  4.642361   27.9879860 8.365387e-07
## INTC -0.34516644  3.331140    2.9066389 2.337929e-01
## MCD  -0.06328978  3.220581    0.3206979 8.518465e-01
## MMM  -0.55061066  4.226123   13.4671610 1.190264e-03
## UTX  -0.42595908  3.034627    3.6045278 1.649251e-01
## PG   -0.25787625  3.350063    1.9265343 3.816440e-01
## HD   -0.41698477  3.662733    5.6263190 6.001507e-02
## AXP   1.21923677 14.840355  724.6116280 0.000000e+00
## GS   -0.50397702  3.899198    9.0466225 1.085303e-02
## GE   -0.64329735  4.903823   26.1793463 2.066461e-06
## IBM  -0.81055720  5.312520   39.5464696 2.585790e-09
## NKE  -0.41377012  3.474446    4.5116952 1.047847e-01

sum(momentsmatrix_m[,4]>0.05)/N

## [1] 0.4615385

#Now already around 40% of the assets exhibit JB test statistics, which are
#insignificantly different from zero at a significance level of 5%. This is to
#say that the data appears to become more and more normal for longer return periods.

#Quarterly frequency:
quarterlylogreturns<-periodReturn(pricedata[,1],period="quarterly",type="log")
for(i in 2:N){
  quarterlylogreturns<-merge(quarterlylogreturns,periodReturn(pricedata[,i],period="quarterly",type="log"))
```

```
}
quarterlylogreturns<-quarterlylogreturns[-1,]
colnames(quarterlylogreturns)<-djmembers

momentsmatrix_q<-matrix(0,N,4)
rownames(momentsmatrix_q)<-djmembers
colnames(momentsmatrix_q)<-c("Skewness","Kurtosis","Test stat JB","p-value JB")
for(i in 1:N){
  momentsmatrix_q[i,1]<-skewness(quarterlylogreturns[,i])
  momentsmatrix_q[i,2]<-kurtosis(quarterlylogreturns[,i])
  momentsmatrix_q[i,3]<-jarque.bera.test(quarterlylogreturns[,i])$statistic
  momentsmatrix_q[i,4]<-jarque.bera.test(quarterlylogreturns[,i])$p.value
}
momentsmatrix_q
```

```
##        Skewness Kurtosis Test stat JB   p-value JB
## CSCO -0.8306182 4.033521   6.22029116 4.459446e-02
## UNH  -0.9625922 4.105587   8.00906904 1.823277e-02
## MSFT -0.5462160 3.779243   2.92601904 2.315384e-01
## PFE  -0.4705272 3.230901   1.52571064 4.663330e-01
## DIS  -0.7634936 3.369400   4.01073791 1.346106e-01
## CVX  -0.1207793 2.031244   1.61986202 4.448888e-01
## MRK  -0.4330494 2.573051   1.51517085 4.687970e-01
## WMT   0.1965763 3.160629   0.29310258 8.636814e-01
## JNJ  -0.5149670 3.257683   1.83164185 4.001880e-01
## XOM   0.1296146 2.664183   0.29245554 8.639609e-01
## TRV  -0.2520837 3.058156   0.41854599 8.111738e-01
## AAPL -0.6778419 3.659383   3.69307980 1.577822e-01
## KO   -0.3000903 3.052487   0.58982880 7.445953e-01
## VZ    0.4517848 2.565430   1.63359517 4.418444e-01
## BA   -0.2851492 2.507182   0.92317814 6.302813e-01
## INTC -0.3861497 2.468283   1.42864952 4.895225e-01
## MCD   0.1176322 2.969073   0.09149691 9.552822e-01
## MMM  -0.7873125 3.987417   5.61345954 6.040220e-02
## UTX  -0.7743550 3.051431   3.90186556 1.421414e-01
## PG   -0.9399508 4.414545   8.99432232 1.114058e-02
## HD   -0.5146367 3.252387   1.82504179 4.015108e-01
## AXP  -0.6096128 7.554504  36.12377767 1.431599e-08
## GS   -0.2868228 3.017253   0.53522126 7.652057e-01
## GE   -0.7689168 3.669641   4.57169577 1.016878e-01
## IBM  -1.3014944 5.565121  21.70251909 1.938018e-05
## NKE  -0.7287214 2.926246   3.46056644 1.772342e-01
```

```
sum(momentsmatrix_q[,4]<=0.05)
```

```
## [1] 5
```

```
#Only 5 assets still exhibit JB test statistics, which are significant at the 5% level. For all other
#assets, the null hypothesis of normality of quarterly returns cannot be rejected. The reason for
#"increasing" normality of returns for longer periods is of course founded in the Central Limit Theorem,
#since the sums of iid random variables, converge to a normal distribution. As longer period log
#returns are just the sums of short period log returns, the result does not surprise.
```

79 Exercise 93

```
es_var_ratio_normal<-function(alpha,samplesize){
  normvec<-rnorm(samplesize)
  VAR<-quantile(normvec,alpha)
  #ES is the expected value of a loss, given that a loss greater than VaR
  #is incurred.
  es<-mean(normvec[normvec>=VAR])
  es/VAR
```

```
}
es_var_ratio_normal(0.9,1000000)

##      90%
## 1.369321

es_var_ratio_normal(0.95,1000000)

##      95%
## 1.255715

es_var_ratio_normal(0.99,1000000)

##      99%
## 1.146356

es_var_ratio_normal(0.999,1000000)

##    99.9%
## 1.087733

es_var_ratio_normal(0.9999,1000000)

##   99.99%
## 1.058004

#The shortfall-quantile ratio indeed seems to be converging to 1
#for the (standard) normal distribution.

es_var_ratio_t<-function(alpha,df,samplesize){
  tvec<-rt(samplesize,df)
  VAR<-quantile(tvec,alpha)
  es<-mean(tvec[tvec>=VAR])
  es/VAR
}
es_var_ratio_t(0.9,2,1000000)

##      90%
## 2.252805

es_var_ratio_t(0.95,2,1000000)

##      95%
## 2.091425

es_var_ratio_t(0.99,2,1000000)

##      99%
## 2.048494

es_var_ratio_t(0.999,2,1000000)

##   99.9%
## 2.00724

es_var_ratio_t(0.9999,2,10000000)
```

```
##   99.99%
## 2.028709

#Theoretical value:
2/(2-1)

## [1] 2

#The shortfall-quantile ratio indeed seems to be converging to 2
#for the Student t-distribution with 2 degrees of freedom.
```

After we have seen that our numerical estimations support the propositions about the two limits, we suggest the following argumentations for them, from an analytical point of view.
First, we look at the case when the loss distribution is standard normal: $X_{loss} \sim \mathcal{N}(0,1)$. Using a common definition of value-at-risk, we have

$$VaR_\alpha(X_{loss}) = \inf\{x \in \mathbb{R} : \Pr(X_{loss} + x < 0) \leq 1 - \alpha\} = -\sup\{y \in \mathbb{R} : F_{X_{loss}}(y) \leq 1 - \alpha\}.$$

Here we have indicated $-x$ with y and the cumulative distribution function of the loss variable by $F_{X_{loss}}$. Since the cds of the normal distribution is strictly increasing over the real line, we can get rid of the infimum/supremum part of the definition:

$$VaR_\alpha(X_{loss}) = \{x \in \mathbb{R} : F_{X_{loss}}(-x) = 1 - \alpha\} = -F^{-1}_{X_{loss}}(1 - \alpha)$$

$$\Rightarrow VaR_\alpha(X_{stnorm}) = -\Phi^{-1}(1 - \alpha)$$

where the last row indicates the cdf of the standard normal distribution.
We use the definition of the expected shortfall of a normal distribution for the standardized case:

$$ES_\alpha = \frac{\phi(\Phi^{-1}(\alpha))}{1 - \alpha}$$

Then we can describe the shortfall-to-quantile ratio in case of a standard normal distribution as

$$\frac{ES_\alpha}{VaR_\alpha} = \frac{\phi(\Phi(\alpha))}{(1-\alpha)(-\Phi^{-1}(1-\alpha))} = \frac{\phi(-\Phi(1-\alpha))}{-\Phi^{-1}(1-\alpha)} \frac{1}{1-\alpha}$$

In the last equation we used the property of the (centralized) normal distribution whose quantiles at complement probabilities are opposites.
Next, we examine the convergence where $\alpha \to 1^-$. This implies

$$\begin{aligned}
(1-\alpha) \to 0^+ \quad &\Rightarrow \\
(-\Phi^{-1}(1-\alpha)) \to +\infty \quad &\Rightarrow \\
1 - \frac{\phi(-\Phi(1-\alpha))}{-\Phi^{-1}(1-\alpha)} \approx \Phi(-\Phi^{-1}(1-\alpha)) \quad &\Rightarrow \\
1 - \Phi(-\Phi^{-1}(1-\alpha)) = 1 - (1 - (-\Phi(\Phi^{-1}(1-\alpha)))) = 1 - \alpha& \\
\approx \frac{\phi(-\Phi(1-\alpha))}{-\Phi^{-1}(1-\alpha)} = \frac{ES_\alpha}{VaR_\alpha}(1-\alpha) \quad &\Rightarrow \\
\frac{ES_\alpha}{VaR_\alpha} \approx 1&
\end{aligned}$$

In case of a generalized normal loss distribution $X_{loss} \sim \mathcal{N}(\mu, \sigma^2)$, the quantile function can be described as a transformation of the quantile function of the standard normal distribution:

$$VaR_\alpha = F^{-1}_{X_{loss}}(1-\alpha) = \mu + \sigma \cdot \Phi^{-1}(\alpha)$$

We also apply the definition of the expected shortfall of a generalized normal distribution:

$$ES_\alpha = \mu + \sigma * \frac{\phi(\Phi^{-1}(\alpha))}{1-\alpha}.$$

Then we can calculate the limit of the shortfall-to-quantile ratio as follows:

$$\begin{aligned}
\lim_{\alpha\to 1^-} \frac{ES_\alpha}{VaR_\alpha} &= \lim_{\alpha\to 1^-} \frac{\mu(1-\alpha)+\sigma\phi(\Phi^{-1}(\alpha))}{(1-\alpha)(\mu+\sigma\Phi^{-1}(\alpha))} && \text{Let } x = \Phi^{-1}(\alpha)\\
\lim_{\alpha\to 1^-} \frac{ES_\alpha}{VaR_\alpha} &= \lim_{x\to 1^-} \frac{\mu(1-\Phi(x))+\sigma\phi(x)}{\mu-\mu\Phi(x)+\sigma\cdot x-\sigma\Phi(x)\cdot x} && \text{We can use the L'Hopital rule}\\
\lim_{\alpha\to 1^-} \frac{ES_\alpha}{VaR_\alpha} &= \lim_{x\to 1^-} \frac{-\mu\phi(x)-\sigma\phi(x)x}{-\mu\phi(x)+\sigma(1-\Phi(x))-\sigma\phi(x)\cdot x} && = 1
\end{aligned}$$

At the end of the proof we used such property of the density of the standard normal distribution that $\phi'(x) = -x\cdot\phi(x)$ and the limit of the cdf of the standard normal distribution $\lim_{x\to\infty}\Phi(x) = 1$.
Finally, we show how the result intuitively follows for the standard Student-t distributed loss variable. We use the property given in the hint, i.e.

$$x \to \infty \Rightarrow f_{t_\nu} \approx \frac{c(\nu)}{x^{\nu+1}}$$

where f_{t_ν} is the density of the standard Student t distribution with ν degrees of freedom and $c(\nu)$ is a function of ν. Here, one has to recognize that the form of this function resembles the generalized Pareto distribution (GPD). The density of the GPD with μ location, σ scale and ξ shape parameters is

$$f_{GPD(\mu,\sigma,\xi)} = \frac{1}{\sigma}(1+\xi\cdot\frac{x-\mu}{\sigma})^{-(1/\xi+1)}.$$

If one sets $\mu = 0,\ \ \sigma = 1$ and $\xi = 1/\nu$, then the specific form of the density is $(1+x/\nu)^{-(\nu+1)}$ which can be written in the form of $\frac{c(\nu)}{x^{\nu+1}}$. In the solution of exercise 94, we provide a thorough derivation of the result we're only using now (adapted to our parameters):

$$VaR_\alpha = \nu((1-\alpha)^{-1/\nu}-1) = \frac{1}{\xi}((1-\alpha)^{-\xi}-1).$$

We also apply another definition of the expected shortfall:

$$\begin{aligned}
ES_\alpha &= \\
&\frac{1}{1-\alpha}\int_\alpha^1 VaR_x dx = \\
&\frac{1}{1-\alpha}\cdot\frac{1}{\xi}\cdot\frac{1}{-\xi+1}\cdot(-1)\cdot(1-x)^{-\xi+1}\Big|_\alpha^1 - \frac{1}{\xi}\cdot\frac{x}{1-\alpha}\Big|_\alpha^1 = \\
&0+\frac{1}{\xi}\cdot\frac{1}{1-\xi}(1-\alpha)^{-\xi+1-1} - \frac{1}{\xi} = \\
&\frac{1}{\xi}(\frac{1}{1-\xi}(1-\alpha)^{-\xi}-1)
\end{aligned}$$

The limit of the shortfall-to-quantile ratio is then the following:

$$\begin{aligned}
&\lim_{\alpha\to 1^-}\frac{ES_\alpha}{VaR_\alpha} = \\
&\lim_{\alpha\to 1^-}\frac{\frac{1}{\xi}(\frac{1}{1-\xi}(1-\alpha)^{-\xi}-1)}{\frac{1}{\xi}((1-\alpha)^{-\xi}-1)} = \\
&\lim_{\alpha\to 1^-}\frac{1}{1-\xi}\frac{(1-\alpha)^{-\xi}-(1-\xi)}{(1-\alpha)^{-\xi}-1} = \\
&\lim_{\alpha\to 1^-}\frac{1}{1-\xi}(1+\frac{\xi}{(1-\alpha)^{-\xi}-1})
\end{aligned}$$

Since $1-\alpha$ becomes a very small number as $\alpha \to 1^-$ and $\nu > 1$ means $0 < \xi < 1$, thus $(1-\alpha)^{-\xi} \to \infty$ and

$$\lim_{\alpha \to 1^-} \frac{ES_\alpha}{VaR_\alpha} = \frac{1}{1-\xi} = \frac{1}{1-\frac{1}{\nu}} = \frac{\nu}{\nu-1}.$$

80 Exercise 94

We start with providing an anlytical solution. In order to obtain VaR, we have to invert the distribution function of the GPD with parameters μ, σ and $\xi < 1$. Let $X \sim GPD_{\mu,\sigma,\xi}$, then its cdf is given by:

$$F(X) = P(X \leq x) = \begin{cases} 1 - (1 + \xi \frac{x-\mu}{\sigma})^{-1/\xi} & \text{for } \xi \neq 0 \\ 1 - \exp(-\frac{x-\mu}{\sigma}) & \text{for } \xi = 0 \end{cases}$$

Since for $\xi = 0$ we just get an exponential distribution with consequentially exponential decay, the interesting case, which is something specifically related to the GPD, we consider only the case of $\xi \neq 0$. We invert the cdf to obtain the quantile function, which allows us to calculate VaR, when using this distribution as a loss distribution:

$$\begin{aligned} & 1 - \left(1 + \xi \frac{x-\mu}{\sigma}\right)^{-1/\xi} = \alpha \Rightarrow 1 + \xi \frac{x-\mu}{\sigma} = (1-\alpha)^{-\xi} \\ \iff & x - \mu = \frac{\sigma}{\xi}((1-\alpha)^{-\xi} - 1) \\ \iff & \text{VaR}_\alpha = x = \frac{\sigma}{\xi}((1-\alpha)^{-\xi} - 1) + \mu \end{aligned}$$

Next, we calculate expected shortfall, which is defined - at level α - as:

$$\text{ES}_\alpha = \frac{1}{1-\alpha} \int_\alpha^1 \text{VaR}_u(L) du$$

where L is the underlying loss distribution. In our given GPD example, we therefore have:

$$\begin{aligned} & \text{ES}_\alpha = \frac{1}{1-\alpha} \int_\alpha^1 \frac{\sigma}{\xi}((1-u)^{-\xi} - 1) + \mu \, du = \frac{1}{1-\alpha} \left(\int_\alpha^1 \frac{\sigma}{\xi}(1-u)^{-\xi} du - \int_\alpha^1 \frac{\sigma}{\xi} du + \int_\alpha^1 \mu \, du \right) = \\ & = \frac{1}{1-\alpha} \left(\frac{\sigma}{\xi} \frac{1}{1-\xi}(1-\alpha)^{1-\xi} - \frac{\sigma}{\xi}(1-\alpha) + \mu(1-\alpha) \right) = \\ & = \frac{\sigma}{\xi} \left(\frac{1}{1-\xi}(1-\alpha)^{-\xi} - 1 \right) + \mu = \frac{\frac{\sigma}{\xi}((1-\alpha)^{-\xi} - 1) + \mu}{1-\xi} + \frac{\sigma - \xi\mu}{1-\xi} \\ & \implies \text{ES}_\alpha = \frac{\text{VaR}_\alpha}{1-\xi} + \frac{\sigma - \xi\mu}{1-\xi} \end{aligned}$$

Finally, we analytically derive the limit of the shortfall-to-quantile ratio for $\alpha \to 1-$:

$$\lim_{\alpha \to 1-} \frac{\text{ES}_\alpha}{\text{VaR}_\alpha} = \lim_{\alpha \to 1-} \frac{\frac{\text{VaR}_\alpha}{1-\xi} + \frac{\sigma - \xi\mu}{1-\xi}}{\text{VaR}_\alpha} = \lim_{\alpha \to 1-} \frac{1}{1-\xi} + \frac{\sigma - \xi\mu}{(1-\xi)\text{VaR}_\alpha}$$

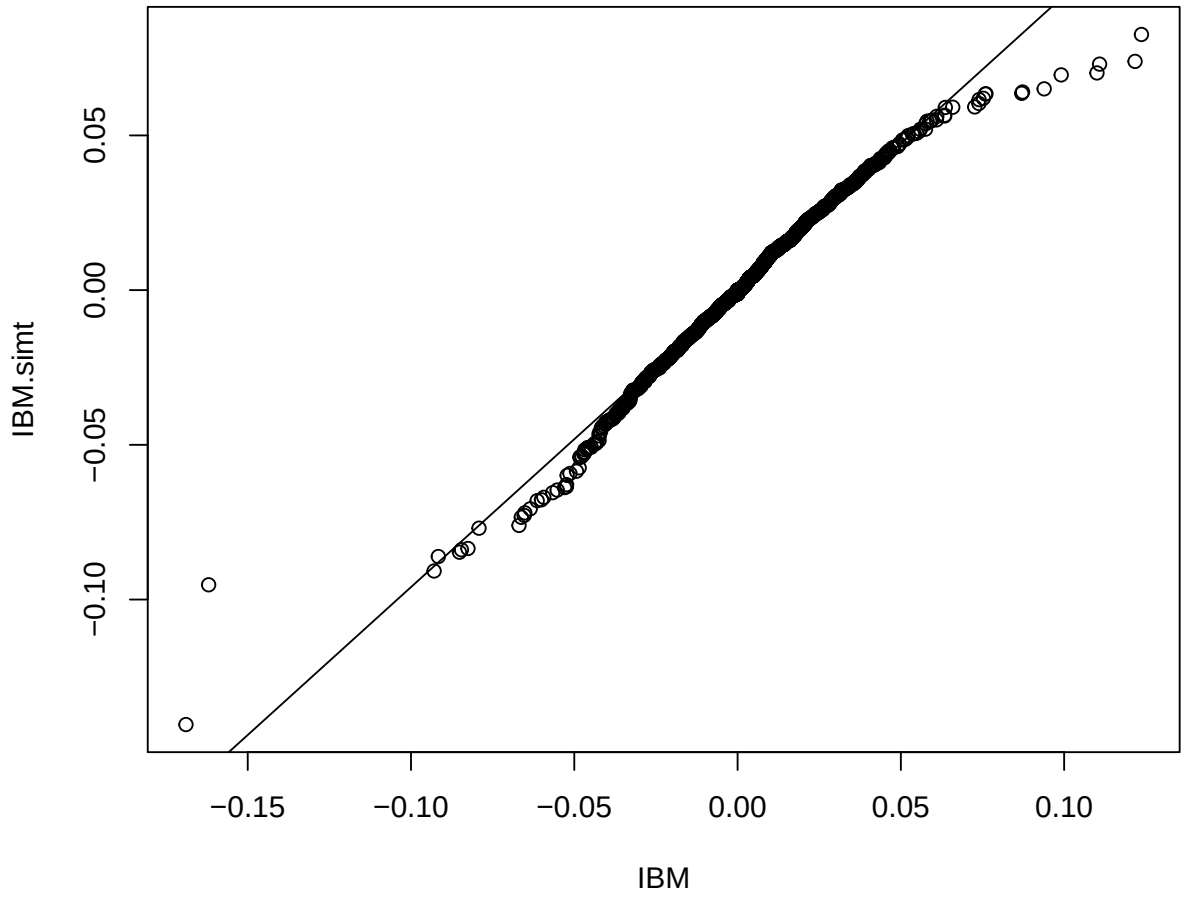
Since $\text{VaR}_\alpha \to \infty$ for $\alpha \to 1$ and $|\xi| < 1$, we get that:

$$\lim_{\alpha \to 1-} = \begin{cases} (1-\xi)^{-1}, & |\xi| < 1 \\ 1, & \xi < 0. \end{cases}$$

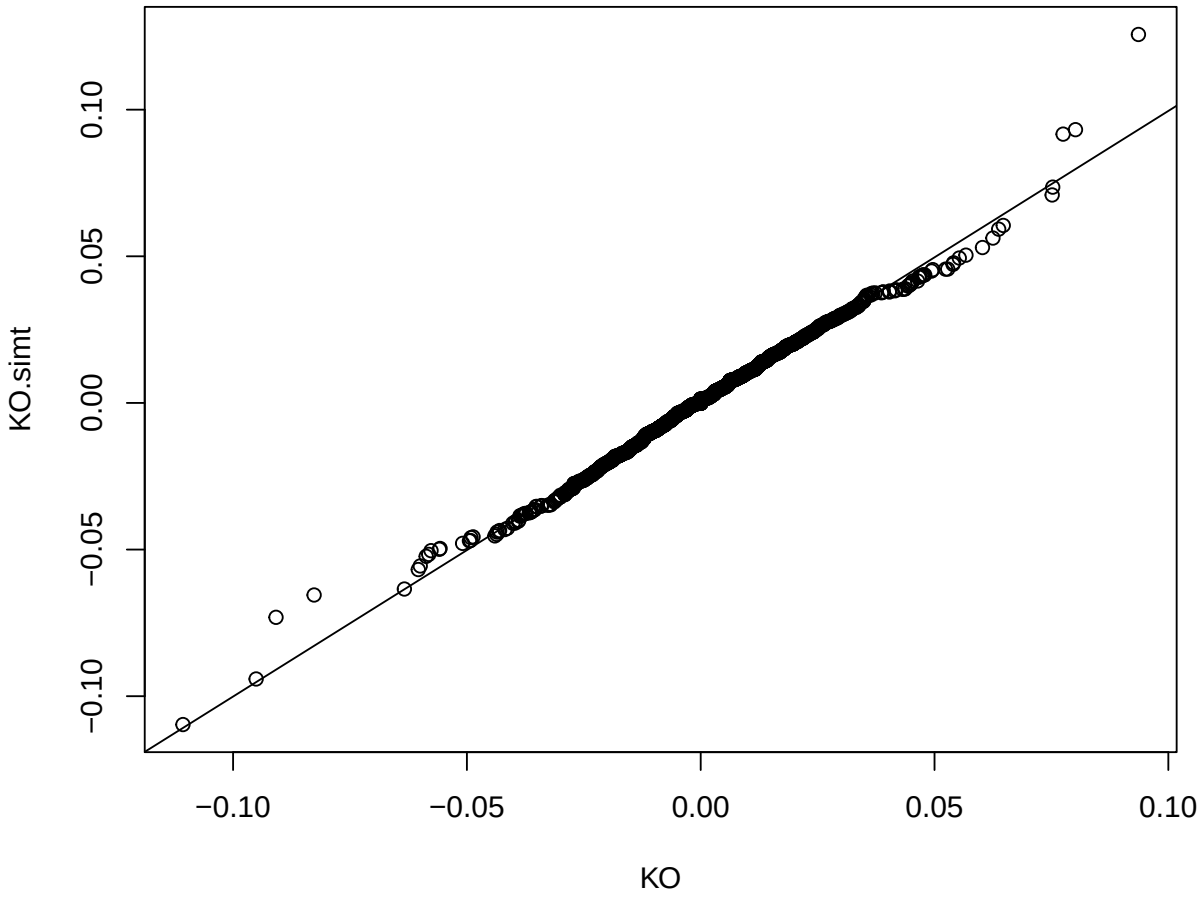
```
##Check analytical solution:
VaR_analy<-function(alpha,mu,sigma,xi) (sigma/xi)*((1-alpha)^(-xi)-1)+mu
ES_analy<-function(alpha,mu,sigma,xi){
  (VaR_analy(alpha,mu,sigma,xi)/(1-xi))+((sigma-(xi*mu))/(1-xi))
}
VaR_a<-VaR_analy(0.99,0,2,0.9)
ES_a<-ES_analy(0.99,0,2,0.9)
ES_a/VaR_a #should be around the limit for alpha -> 1, which is (1-0.9)^{-1}=10.

## [1] 10.14494
```

81 Exercise 95

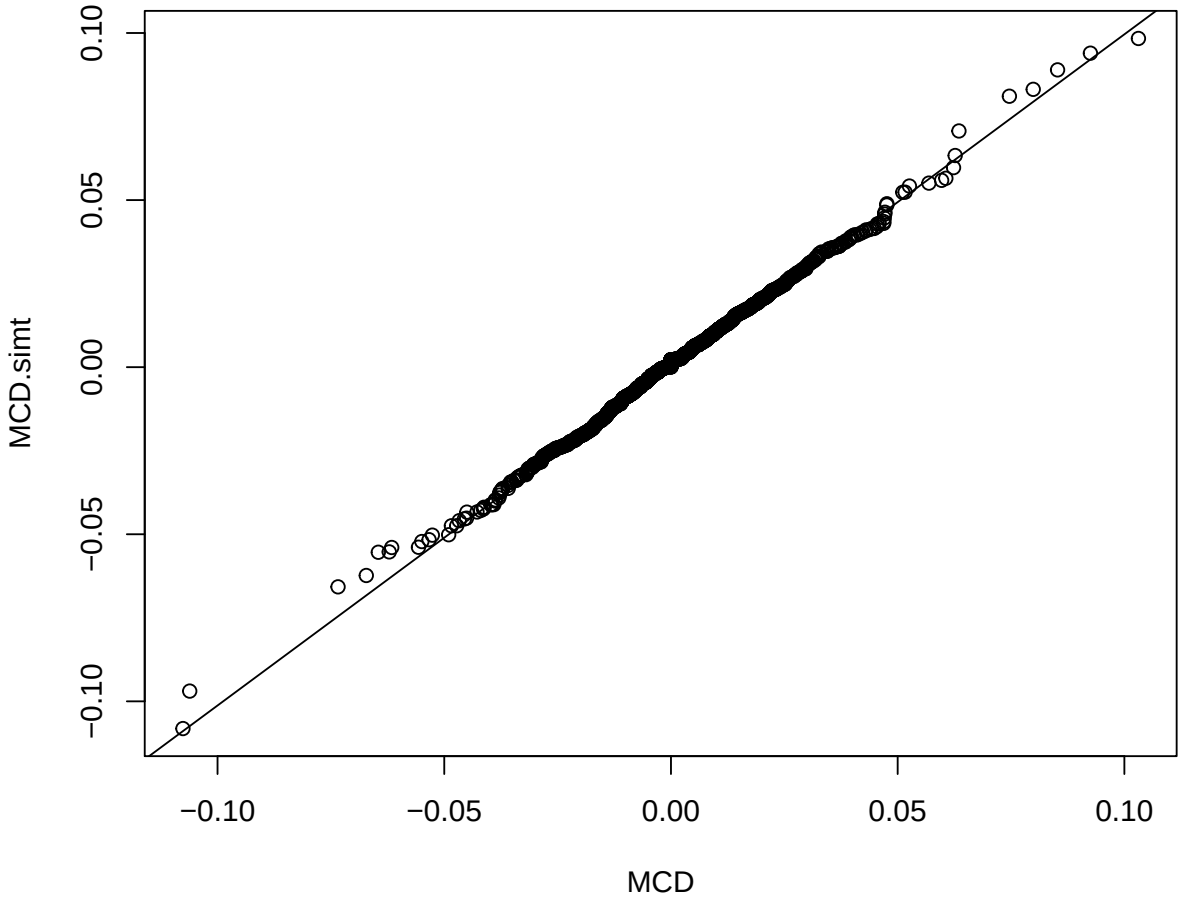
```
library(quantmod)
companies<-c("IBM","KO","MCD","NKE")
N<-length(companies)
COMPS<-new.env()
for(i in 1:N){
  getSymbols(companies[i],env=COMPS,src="yahoo",from=as.Date("1993-01-01"),to=as.Date("2000-12-31"))
}
pricedata<-merge(COMPS$IBM[,6],COMPS$KO[,6],COMPS$MCD[,6],COMPS$NKE[,6])
dailylogreturns<-periodReturn(pricedata[,1],period="daily",type="log")
for(i in 2:N){
  dailylogreturns<-merge(dailylogreturns,periodReturn(pricedata[,i],period="daily",type="log"))
}
dailylogreturns<-dailylogreturns[-1,]
colnames(dailylogreturns)<-companies

IBM<-as.numeric(dailylogreturns[,1])
KO<-as.numeric(dailylogreturns[,2])
MCD<-as.numeric(dailylogreturns[,3])
NKE<-as.numeric(dailylogreturns[,4])

#Fitting distributions:
#t fits:
library(MASS)
IBM.t<-fitdistr(IBM,"t")
#We simulate from the estimated t distribution and check graphically how
#the simulated data fits the original data in a QQ plot.
IBM.simt<-IBM.t$estimate[1]+IBM.t$estimate[2]*rt(length(IBM),IBM.t$estimate[3])
qqplot(IBM,IBM.simt)
qqline(IBM,distribution = function(x) quantile(IBM.simt,x))
```



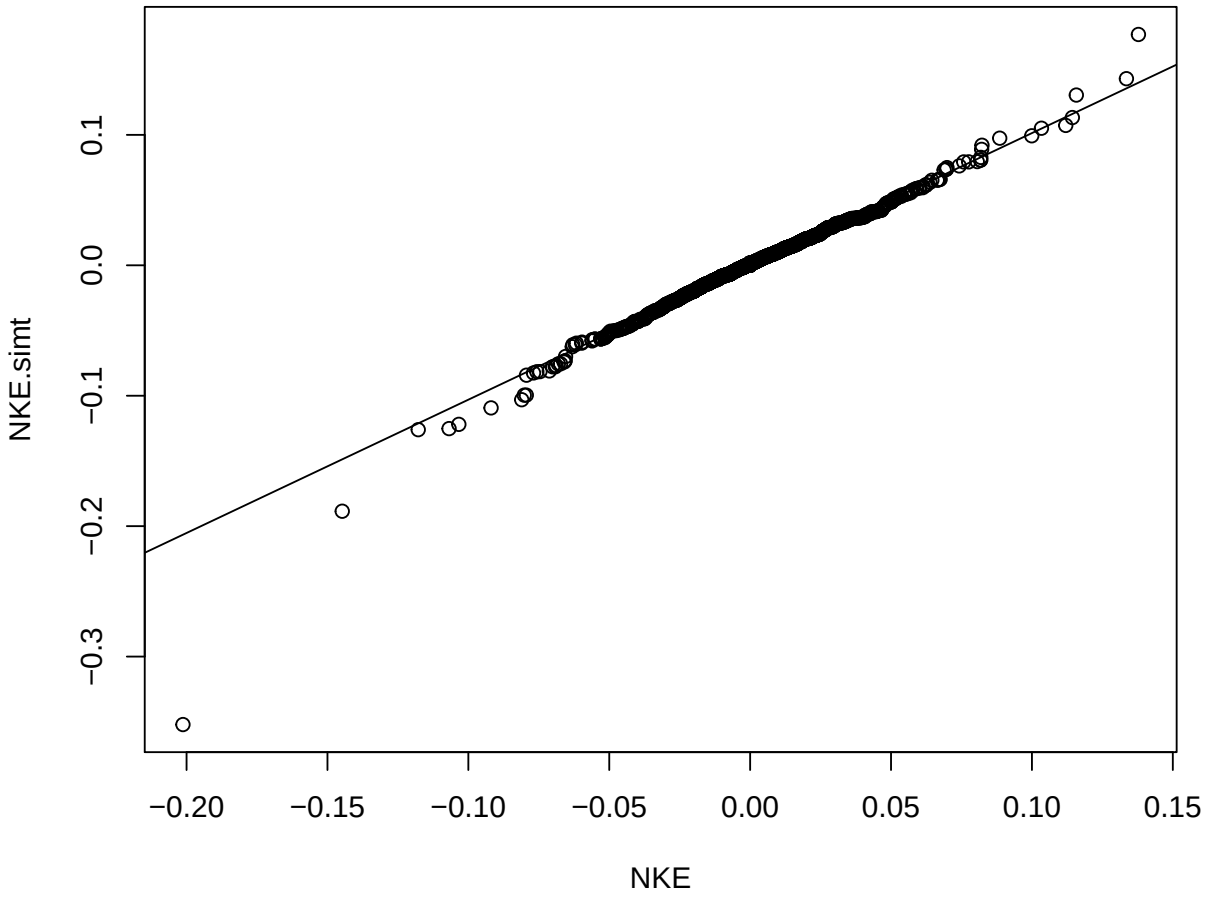

```
KO.t<-fitdistr(KO,"t")
KO.simt<-KO.t$estimate[1]+KO.t$estimate[2]*rt(length(KO),KO.t$estimate[3])
qqplot(KO,KO.simt)
qqline(KO,distribution = function(x) quantile(KO.simt,x))
```




```
MCD.t<-fitdistr(MCD,"t")
MCD.simt<-MCD.t$estimate[1]+MCD.t$estimate[2]*rt(length(MCD),MCD.t$estimate[3])
qqplot(MCD,MCD.simt)
qqline(MCD,distribution = function(x) quantile(MCD.simt,x))
```

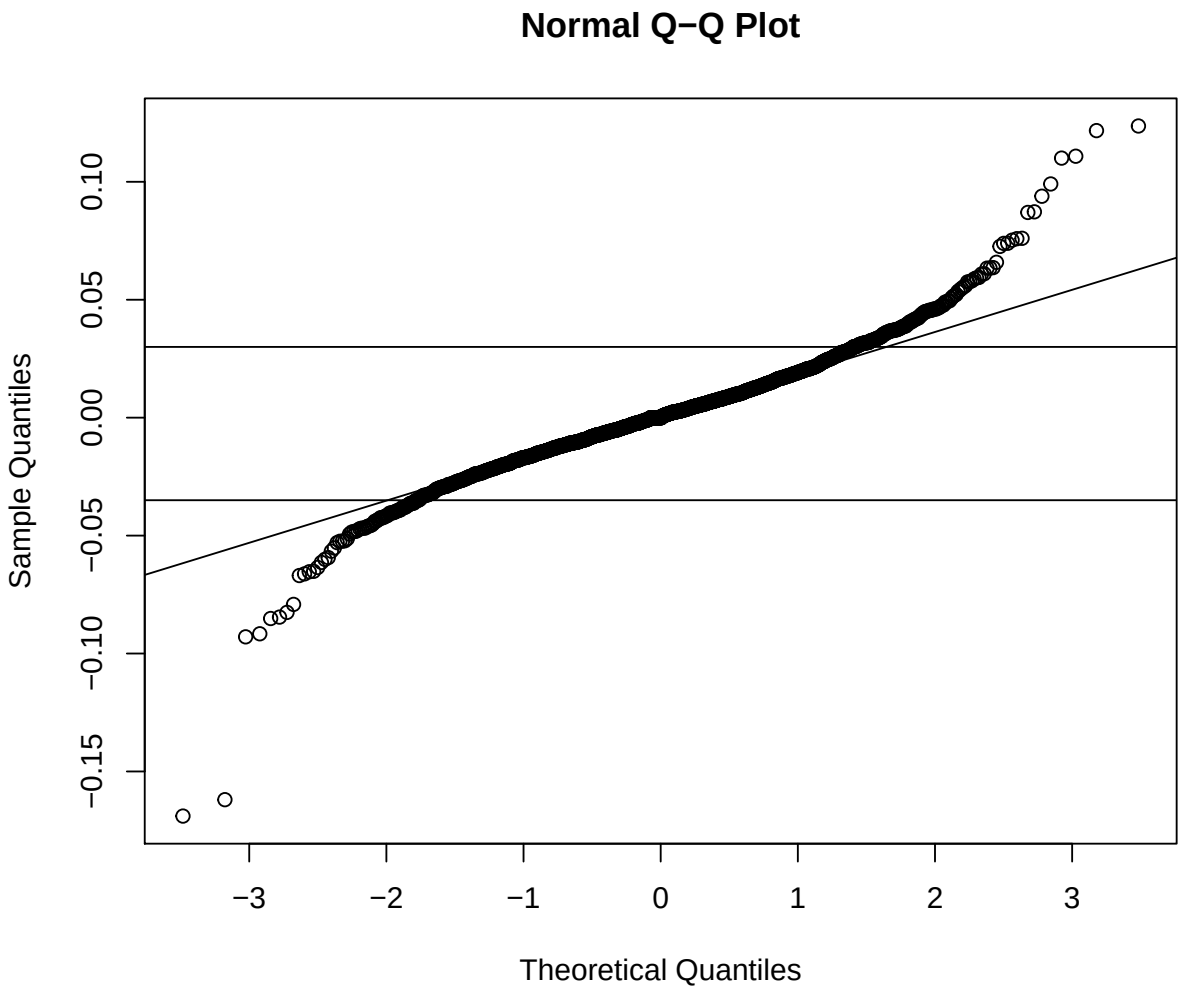



```
NKE.t<-fitdistr(NKE,"t")
NKE.simt<-NKE.t$estimate[1]+NKE.t$estimate[2]*rt(length(NKE),NKE.t$estimate[3])
qqplot(NKE,NKE.simt)
qqline(NKE,distribution = function(x) quantile(NKE.simt,x))
```

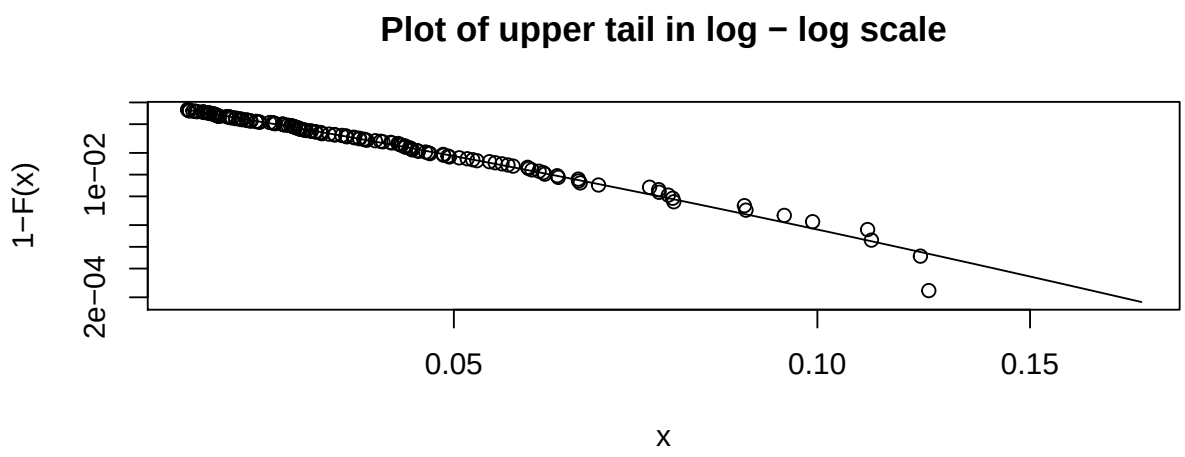


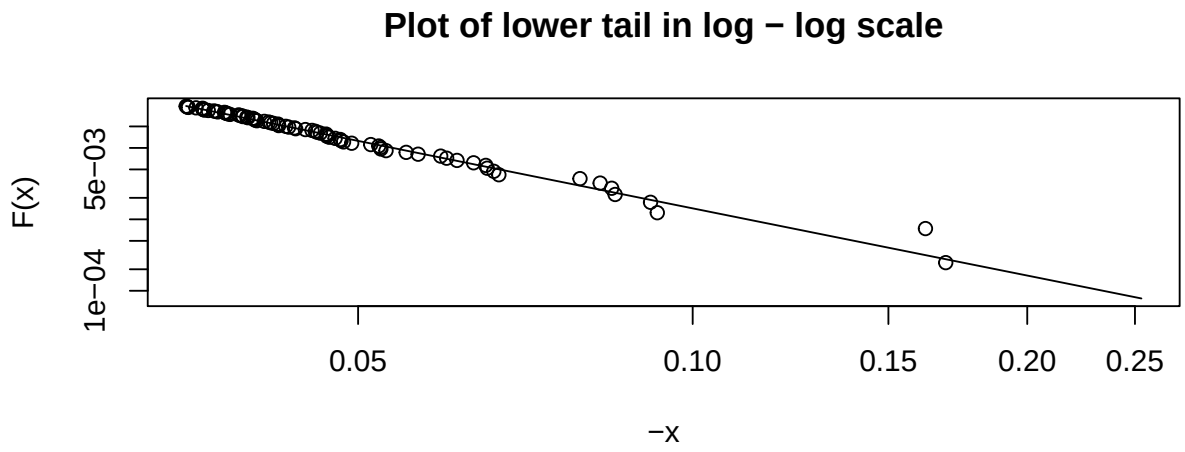

```
#GPD fits:
library(Rsafd)

##
## Attaching package:  'Rsafd'
## The following objects are masked _by_ '.GlobalEnv':
##
##    dmvnorm, dpareto, IBM
## The following object is masked from 'package:tseries':
##
##    garch
## The following objects are masked from 'package:mvtnorm':
##
##    dmvnorm, rmvnorm
## The following object is masked from 'package:stats':
##
##    qqnorm
## The following object is masked from 'package:datasets':
##
##    CO2

#IBM:
```

```
qqnorm(dailylogreturns[,1])
abline(h=-0.035)
abline(h=0.03)
```

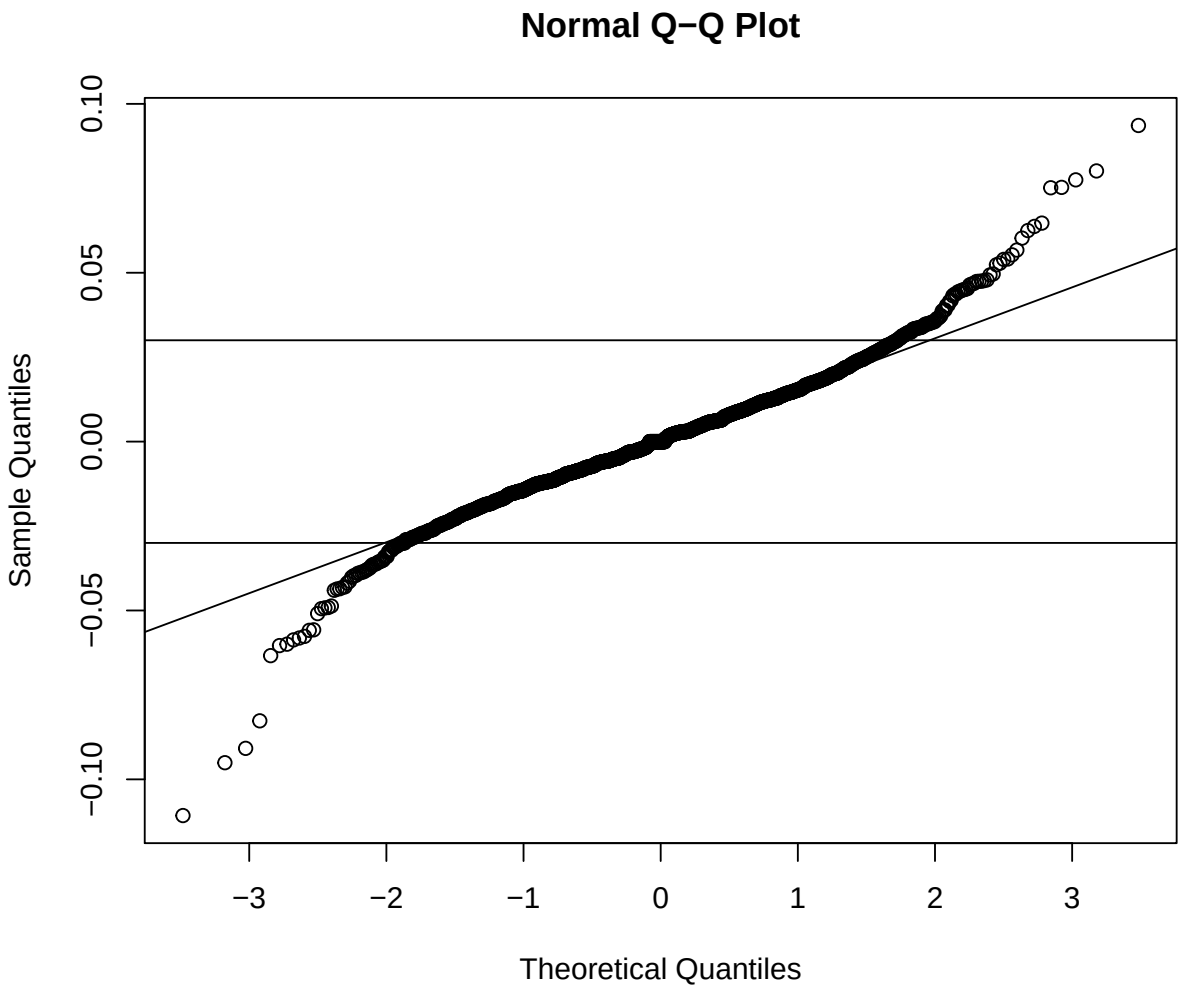



```
IBM.est<-fit.gpd(IBM,upper=0.03,lower=-0.035,plot=FALSE)
tailplot(IBM.est) #Clearly fits tails well
```

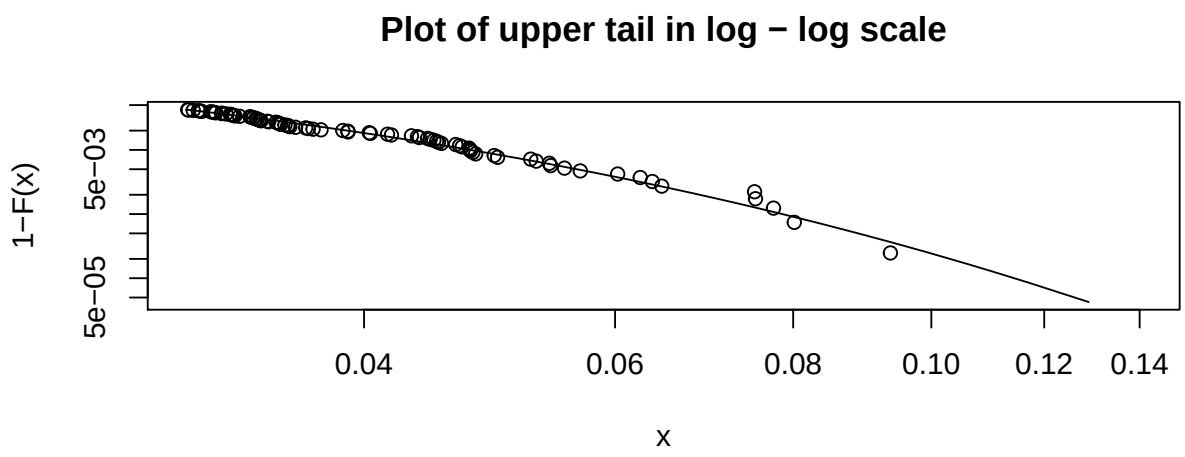


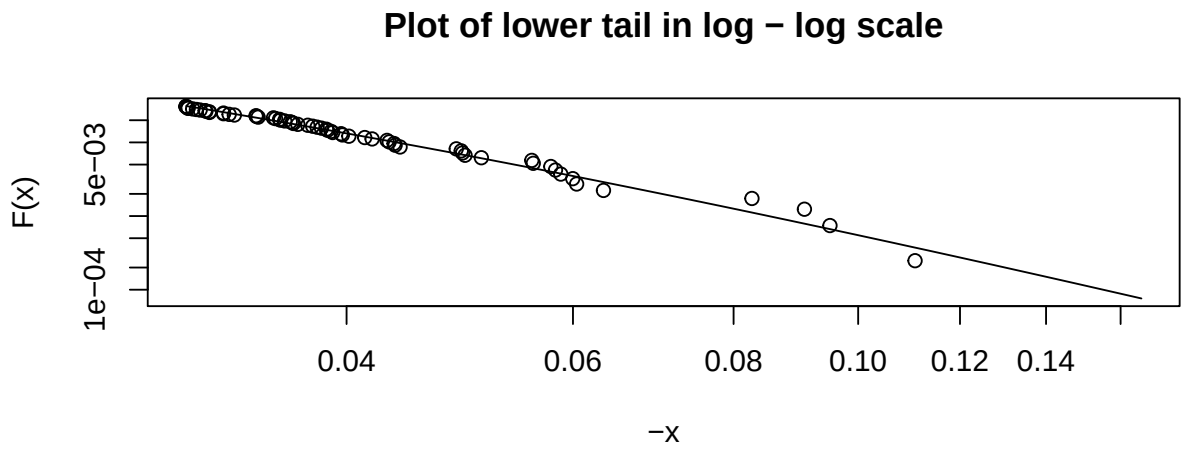



```
#KO:
qqnorm(dailylogreturns[,2])
abline(h=-0.03)
abline(h=0.03)
```

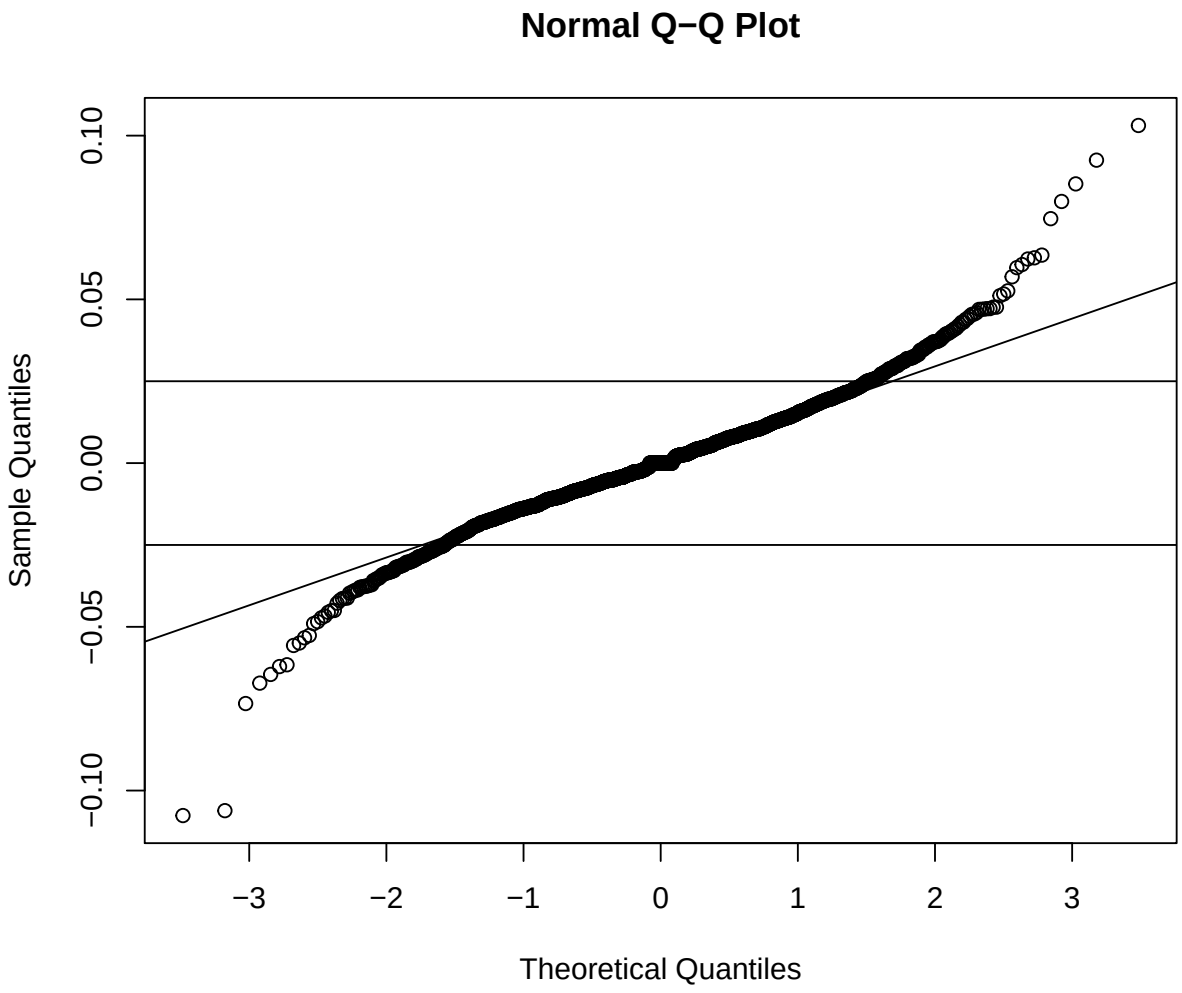



```
KO.est<-fit.gpd(KO,upper=0.03,lower=-0.03,plot=FALSE)
tailplot(KO.est) #Clearly fits tails well
```

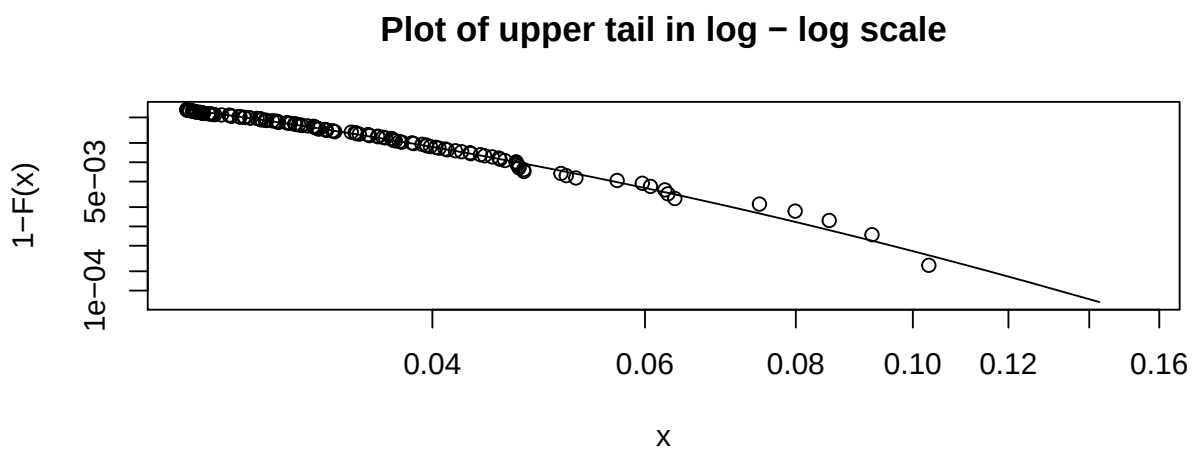


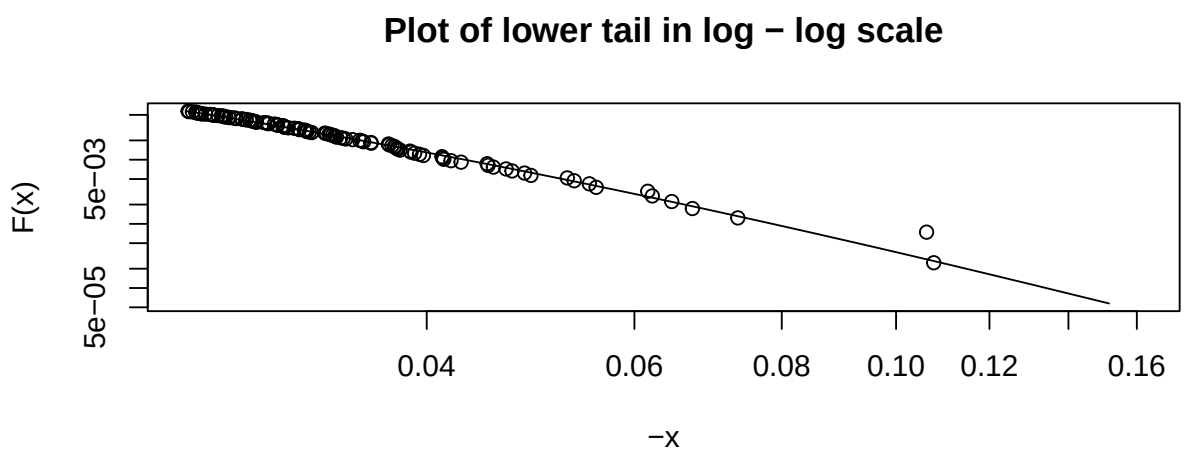



```
#MCD:
qqnorm(dailylogreturns[,3])
abline(h=-0.025)
abline(h=0.025)
```

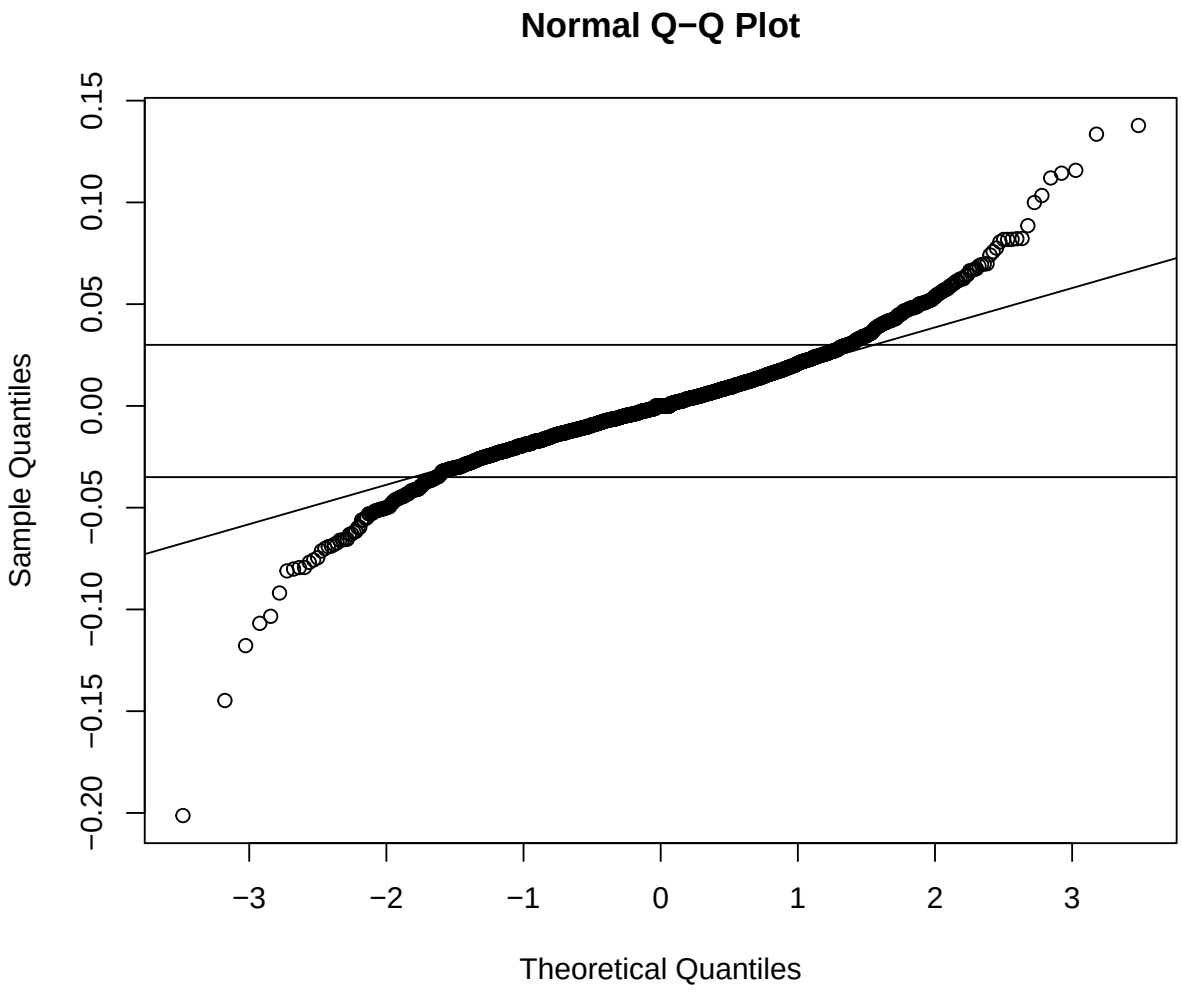



```
MCD.est<-fit.gpd(MCD,upper=0.025,lower=-0.025,plot=FALSE)
tailplot(MCD.est) #Clearly fits tails well
```

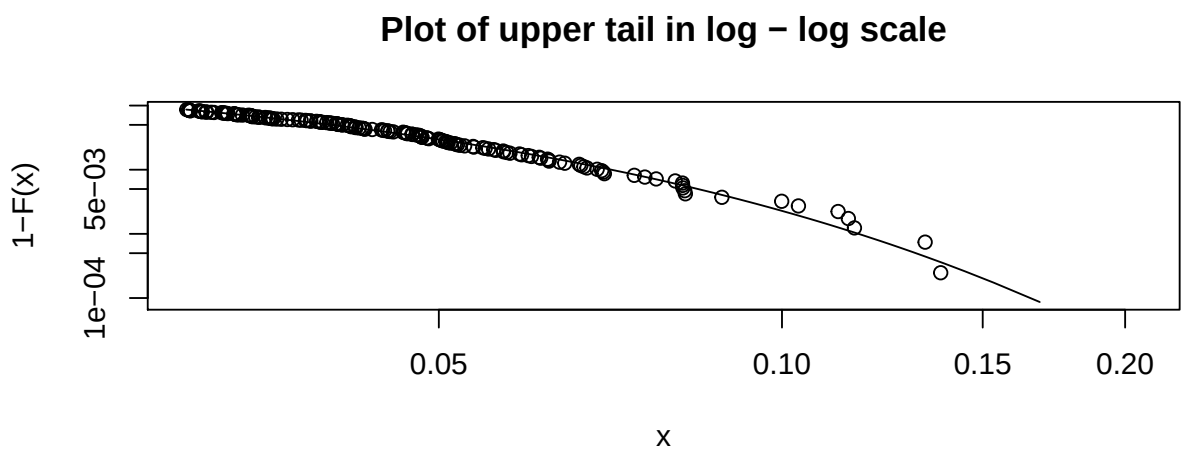


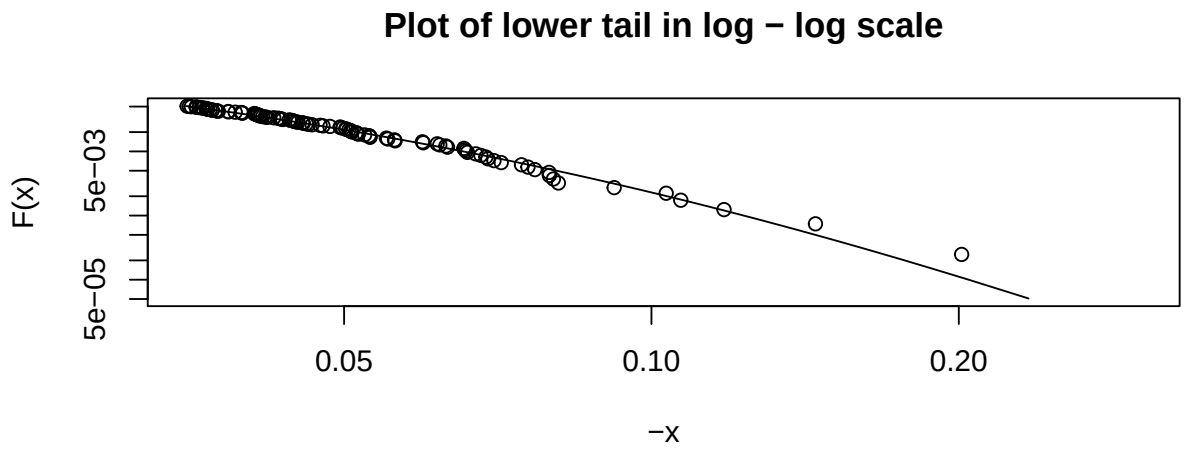



```
#NKE:
qqnorm(dailylogreturns[,4])
abline(h=-0.035)
abline(h=0.03)
```




```
NKE.est<-fit.gpd(NKE,upper=0.03,lower=-0.035,plot=FALSE)
tailplot(NKE.est) #Clearly fits tails well
```






```
#Discussion of implied tail behaviour:
parametermatrix<-matrix(0,4,3)
colnames(parametermatrix)<-c("degrees of freedom","xi_lowertail","xi_uppertail")
rownames(parametermatrix)<-companies

#Extracting dfs:
parametermatrix[1,1]<-IBM.t$estimate[3]
parametermatrix[2,1]<-KO.t$estimate[3]
parametermatrix[3,1]<-MCD.t$estimate[3]
parametermatrix[4,1]<-NKE.t$estimate[3]

#Extracting Xi's
parametermatrix[1,2]<-IBM.est@lower.par.ests[2]
parametermatrix[1,3]<-IBM.est@upper.par.ests[2]
parametermatrix[2,2]<-KO.est@lower.par.ests[2]
parametermatrix[2,3]<-KO.est@upper.par.ests[2]
parametermatrix[3,2]<-MCD.est@lower.par.ests[2]
parametermatrix[3,3]<-MCD.est@upper.par.ests[2]
parametermatrix[4,2]<-NKE.est@lower.par.ests[2]
parametermatrix[4,3]<-NKE.est@upper.par.ests[2]
```

```
parametermatrix

##     degrees of freedom xi_lowertail xi_uppertail
## IBM           5.903488    0.3212134   0.23646019
## KO            5.073225    0.2270490   0.06024308
## MCD           4.875847    0.1945999   0.13866727
## NKE           3.709355    0.1492712   0.02373270
```

Note that the lower the ν and the higher the ξ, the slower the decay of probabilities and hence the heavier the tails. From the impression of the above plots and also from the parameter matrix, we see that it definitely makes sense to model the lower tail and the upper tail separately, since there are rather marked differences between the values for ξ, depending on whether we fit the GPD for the upper or the lower tail of a given return distribution. This is thus an immediate advantage of using a 2-sided GPD approach for modelling of tail behaviour, rather than the inherently symmetric t-distribution, for which we are not able to account for differences in tail heaviness between left and right tail. For KO, for example, the lower tail seems to be considerably heavier than the upper tail, for which decay happens much faster. Notably, for all assets, the lower (loss) tail appears to be heavier than the upper (gain) tail, which corresponds to the stylised fact of asset returns usually exhibiting heavier loss tails than gain tails. It is mainly this stylised fact that justifies the separate modelling of both tails with the 2-sided GPD approach. Judging on the basis of the estimated ξ values, the IBM return distribution generally seems to exhibit the heaviest tails. The upper tail of KO and NKW, on the other hand, do not seem overly heavy at all on the other hand.

The problems involved with fitting a symmetric distribution, thereby not accounting for possible differences in tail heaviness, could also be seen from the QQ plots of the simulated data and the original data. Although the t-distribution fits generate simulated data, which are reasonably able to reproduce the empirical features of the dataset, there are definitely notable divergences in the tails. Note however that the t distribution is fitted to the entire distribution, while the 2-sided GPD approach explicitly models the tails only, so it is - in a sense - not an entirely fair comparison. This actually also explains why the degrees of freedom parameter ν, determining the heavy-tailedness, does not comply with the results from the estimated ξ's. More specifically, ν is estimated to be highest (i.e. lightest tails) for IBM, i.e. the asset we determined to have the heaviest tails based on the GPD approach. Again, this is likely to be a consequence of the fact that this choice of ν fits the observations in the centre of the distribution (which obviously constitute the majority of observations) better, although it provides slightly worse fits in the tails.

82 Exercise 111

First, we need to simulate data exponential margins with structure given by Gaussian copula. Here we use the parameters provided in the task, i.e. 125 obligors, $\lambda = 0.3$ and equicorrelation matrix with $\rho = 0.7$. The time scale is 1 year and time origin is now.

```
#library
library(mvtnorm)
n<-1:5 # variable to show only parts of correlation matrix

#equicorrelation matrix
correlation_matrix<-matrix(0.7,125,125)
diag(correlation_matrix)<-rep(1,125)

s<-100000 #number of simulations

##--1. Multivariate simulation
Ys<-rmvnorm(s,rep(0,125),correlation_matrix) # Simulation of multivariate normals
#with given dependence structure

#dependence structure check
```

```
cor(Ys,method="sp")[n,n]
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.6838673 0.6794988 0.6816383 0.6857818
## [2,] 0.6838673 1.0000000 0.6809213 0.6799206 0.6833268
## [3,] 0.6794988 0.6809213 1.0000000 0.6792409 0.6806698
## [4,] 0.6816383 0.6799206 0.6792409 1.0000000 0.6818166
## [5,] 0.6857818 0.6833268 0.6806698 0.6818166 1.0000000
```

```
#Note we use Spearman coefficient for measuring dependence since we
#are working with copulas, where in general its better to avoid using Pearson
#coefficient of correlation since it measures only linear dependence


##--2. Transformation from normals to exponentials
r<-0.3 #rate
Ti<-qexp(pnorm(Ys),r)

cor(Ti,method = "sp")[n,n]
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.6838673 0.6794988 0.6816383 0.6857818
## [2,] 0.6838673 1.0000000 0.6809213 0.6799206 0.6833268
## [3,] 0.6794988 0.6809213 1.0000000 0.6792409 0.6806698
## [4,] 0.6816383 0.6799206 0.6792409 1.0000000 0.6818166
## [5,] 0.6857818 0.6833268 0.6806698 0.6818166 1.0000000
```

```
##--3. Determining defaults on 1 year window
#Now we have simulated time in years since now when each particular obligor defaults and
#since we are interested in 1-year PDs we need to indicate defaults within this period
Defaults_table<-Ti<=1
Defaults_in_sim<-rowSums(Defaults_table)
#Probability of no default within 1- year from now
P_zero_default_1Y<-sum(Defaults_in_sim==0)/s
P_zero_default_1Y
```

```
## [1] 0.18729
```

Note that in the simulation we have used a Gaussian copula to generate margins from exponential distribution by inverting the copula in 30 . Next we provide a more precise solution, which is based on following logic. The event of having no default within 1 year is equivalent to having all defaults after 1 year. The probability u of each single exponential margin not to default within 1 year period is $u = 1 - F(x)$, where $F(x)$ is distribution function of exponential distribution (in our case $x = 1$). Then we can use a Gaussian copula to evaluate the probability using following formula,

$$\Phi_{\Sigma}(\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_{125})) \tag{30}$$

where Φ_{Σ} is multivariate normal distribution function with dependence structure given by Σ and zero means, $u_1 = \ldots = u_{125} = u$ (u defined above).Thus using the formula more precise solution is:

```
F_x<-1-pexp(1,r)
X<-qnorm(F_x)
mu<-rep(0,125)

##Probability of no default within 1 year
pmvnorm(upper = rep(X,125),mean = rep(0,125),sigma = correlation_matrix)[1]
```

```
## [1] 0.1866107

#The result is obviously very close to the result of simulation.
```