Statistics 1 Unit 4

Group 8

January 9, 2018

Contents

1 Task 66

Let's first use implemented rchisq funciton:

```
b <- seq(0,100000,0.01)

mean(rchisq(b,8))

## [1] 8.003719

var(rchisq(b,8))

## [1] 16.02174
```

So, simulated values of mean and variance are close to theoretical ones, but not exactly equal to them. The difference between calculated and theoretical values of mean and variance can be explained by the fact that our vector of χ^2 distributed values, while containing very large number of values with quite small difference 0.01, still is not a continious line (and could not be converted to it using R). Let's now try to simulate χ^2 variable as the sum of the squares of n independent standard normal random variables using implemented rnorm function:

```
n <- 1000000

x <- c(0, rep=n)

for (i in (1:n)){
x[i] <- sum(rnorm(8,0,1)^2)
}

mean(x)

## [1] 8.006394

var(x)

## [1] 16.03997
```

Again, simulated values of mean and variance are close to theoretical ones, but not exactly equal to them. Results of rchisq are close to that of rnorm. On some simulations rchisq performs better, on other - rnorm. However, in general it seems to be that rchisq performs better than rnorm. Remark: Sizes of sets were found empirically until computer hangs.

2 Task 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
}
```

2.1 a)

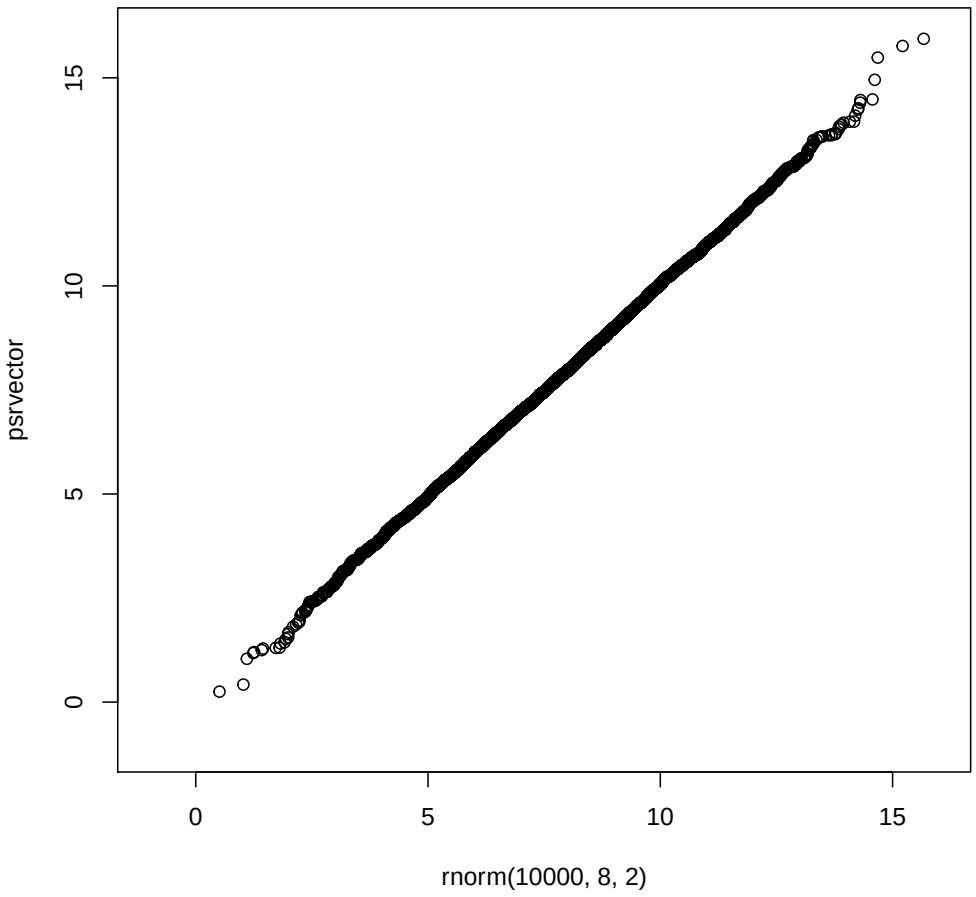
Let's generate a vector of 10000 normal pseudorandom numbers with a mean of 8 and a standard deviation of 2:

```
psrvector <- rannorm(10000, mean = 8, sd =2)

length(psrvector)

## [1] 10000

head(psrvector)

## [1]  6.767608  6.112975  5.834931  9.967979
   10.723156
## [6]  8.355115

tail(psrvector)

## [1] 9.168238 7.383033 7.749550 7.681412 3.766710
   9.192922
```

2.2 b)

Let's compare generated numbers with a QQ-plot of the implemented rnorm function.

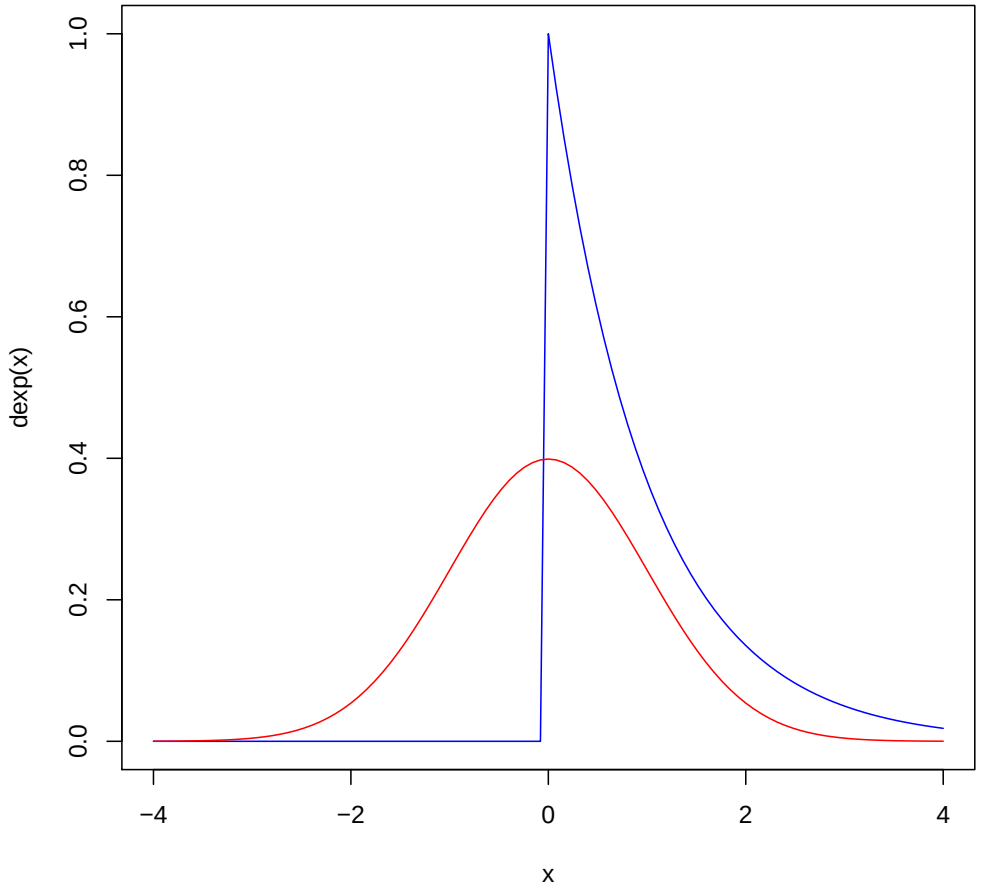
```
qqplot(rnorm(10000,8, 2), psrvector, xlim=c(-1,16),
   ylim=c(-1,16))
```



It seems to fit quite well. For numbers rather far away from the mean our QQ plot is not that straight anymore. This shape suggests that we have a bit heavier tails in the distribution of the numbers generated by our function rannorm or that there are not enough datapoints to account for the outliers.

2.3 c)

```
curve(dexp, -4,4, col="blue")

curve(dnorm, add=T, col="red")
```



It is implemented correctly. We basically draw first from the exponential distribution and then accept, if the variable is under the red normal curve.

3 Task 68

```
probs <- c(0.2, 0.3, 0.1, 0.15, 0.05, 0.2)
```

The first method is an inversion method:

```
randiscrete1 <- function(n, probs) {
 cumprobs <- cumsum(probs)
 singlenumber <- function() {
 x <- runif(1)
 sum(x > cumprobs)
 }
 replicate(n, singlenumber())
 }
```

The second method is a rejection method:

```
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())
 }
```

Let's try them first:

```
randiscrete1(100,probs)
```

```
##   [1] 1 2 0 0 2 4 1 4 0 2 1 5 2 3 1 4 5 2 5 3 5 5 5
    4 5 5 2
##  [28] 1 1 3 4 1 5 5 0 4 2 1 1 5 1 5 5 1 5 0 2 3 1 1
    2 0 2 1
```

```
##  [55] 1 5 0 0 1 2 1 1 3 0 4 4 5 1 3 2 0 2 2 2 1 5 1
    1 1 0 3
##  [82] 1 1 0 5 0 2 1 3 1 1 4 0 3 1 0 1 3 3 5

randiscrete2(100,probs)

##   [1] 4 0 5 0 5 1 5 1 1 4 1 3 1 5 0 4 5 1 5 3 1 0 5
    2 1 0 0
##  [28] 5 5 4 2 1 1 5 2 3 1 1 3 1 1 1 3 2 1 3 5 1 1 0
    1 2 3 1
##  [55] 5 1 0 5 0 5 3 1 1 1 4 1 0 1 3 0 1 4 1 1 5 3 1
    1 0 0 1
##  [82] 1 3 5 1 0 0 2 3 0 1 2 1 3 3 0 4 4 2 0
```

Let's check whether the results are reasonable:

```
checksample1 <- (randiscrete1(10000,probs))

length(checksample1[checksample1 == 0])/10000

## [1] 0.1999

length(checksample1[checksample1 == 1])/10000

## [1] 0.292

length(checksample1[checksample1 == 2])/10000

## [1] 0.0971

length(checksample1[checksample1 == 3])/10000

## [1] 0.1514

length(checksample1[checksample1 == 4])/10000

## [1] 0.0528

length(checksample1[checksample1 == 5])/10000

## [1] 0.2068
```

It can be seen that the probabilities fit quite good to the values as stated in the exercise.
Let's check the second function:

```
checksample2 <- (randiscrete2(10000,probs))

length(checksample2[checksample2 == 0])/10000

## [1] 0.2057

length(checksample2[checksample2 == 1])/10000

## [1] 0.297

length(checksample2[checksample2 == 2])/10000

## [1] 0.102

length(checksample2[checksample2 == 3])/10000

## [1] 0.1484

length(checksample2[checksample2 == 4])/10000

## [1] 0.044

length(checksample2[checksample2 == 5])/10000

## [1] 0.2029
```

Again, the probabilities fit quite good to the values as stated in the exercise.
Finally, let's check the performance time of the two functions for different sample sizes.

```
system.time(randiscrete1(100,probs))

##    user  system elapsed
##       0       0       0

system.time(randiscrete2(100,probs))
```

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

system.time(randiscrete1(1000,probs))

##    user  system elapsed
##       0       0       0

system.time(randiscrete2(1000,probs))

##    user  system elapsed
##    0.01    0.00    0.01

system.time(randiscrete1(10000,probs))

##    user  system elapsed
##    0.02    0.00    0.02

system.time(randiscrete2(10000,probs))

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

So, the inversion method seems to be quite efficient and faster than the rejection method in all three cases.

4 Task 69

A good way to sample from a normal location mixture is to draw first pseudo-random variables from the uniform distribution which will serve as a reference for comparing our probability, as we revealed after some research. So let's start with that:

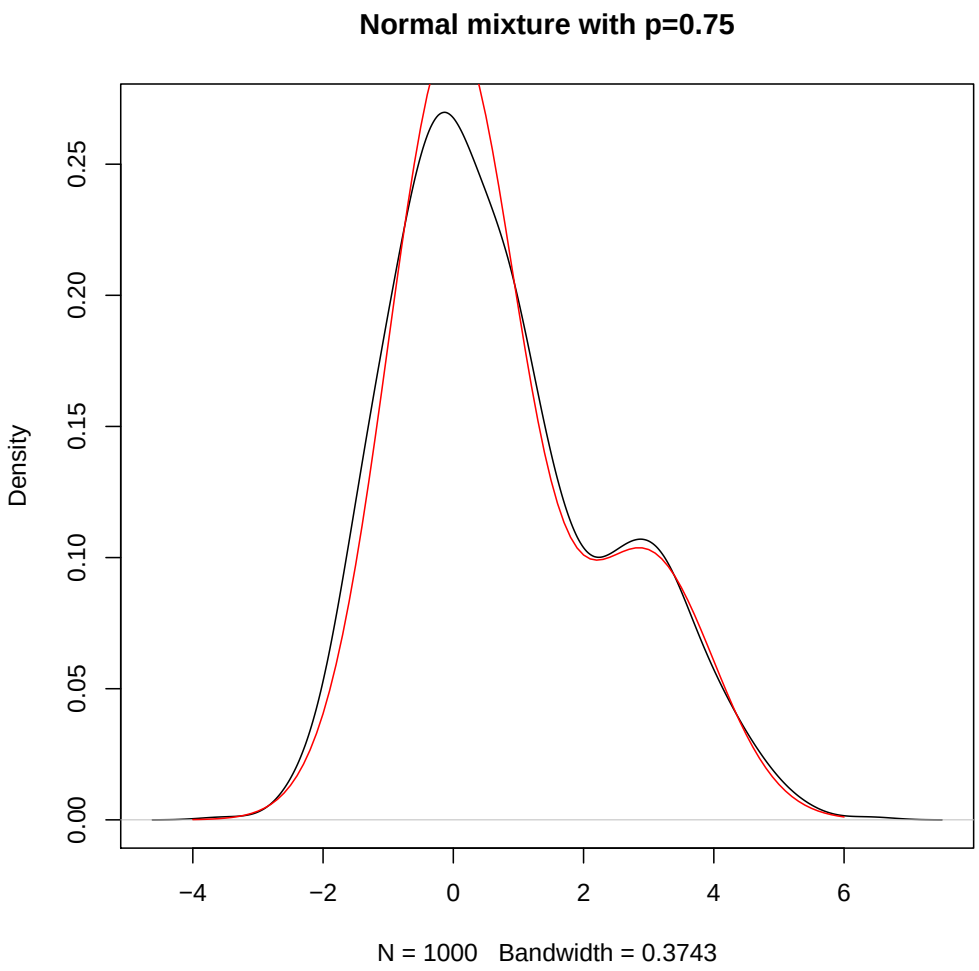
```
num <- 1000

prob_ref <- runif(num,0,1)
```

Now we write a function where we draw from our two given normal distributions based on the sampled probabilities.

```
normal_mixture <- function(p){
  mix <- c()
  for(i in (1:num)){
    if(prob_ref[i]<=p) {
    mix[i] <- rnorm(1,0,1)
    }
    else{
      mix[i] <- rnorm(1,3,1)
    }
  }
  mix
}
```

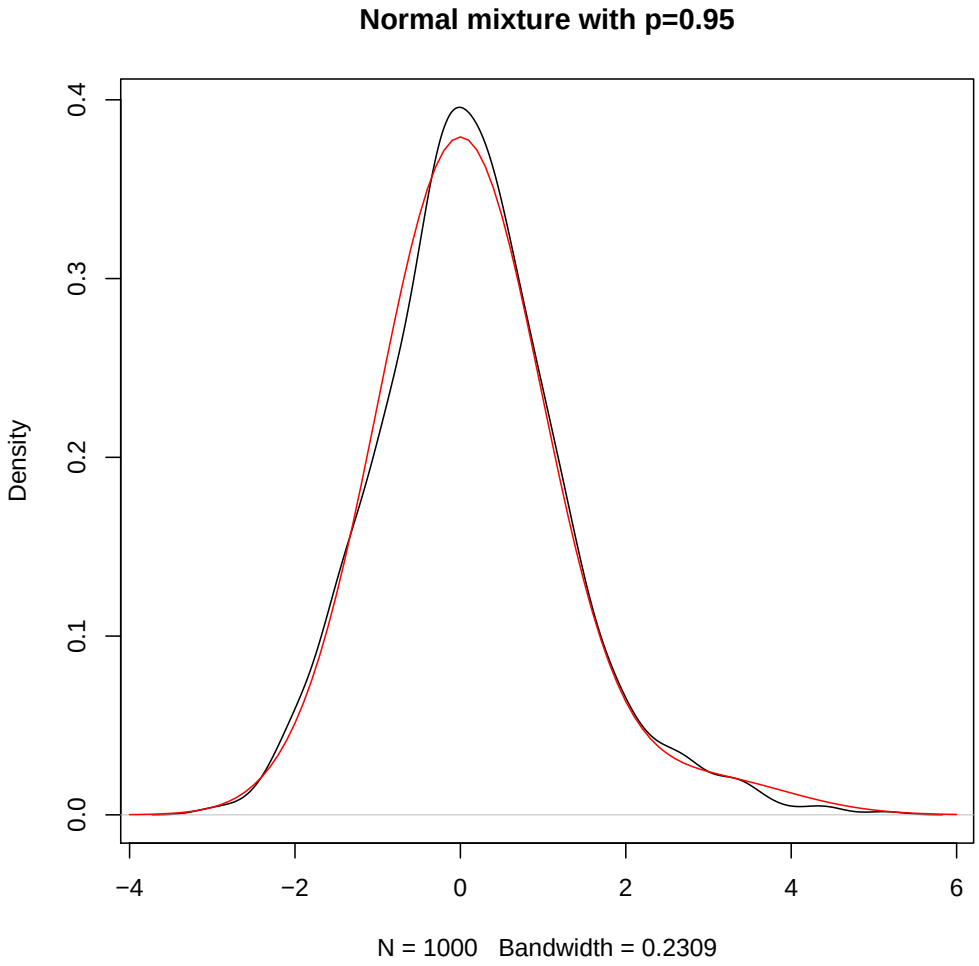
So let's try it and we also add a "reality check" which you can see in the red theoretical graph:

```
p1 <- 0.75

rcheck1<- normal_mixture(p1)

plot(density(rcheck1), main="Normal mixture with p
   =0.75")

lines(seq(-4,6, 0.1), p1*dnorm(seq(-4,6,0.1),0,1)
      + (1-p1)*dnorm(seq(-4,6,0.1),3,1), col="red")
```



It looks a bit bimodal (there are 2 modes), although we expected a bit more spectacular bimodal graph. But it definitely fits our theoretical expectation quite well.
Let's try a larger p:

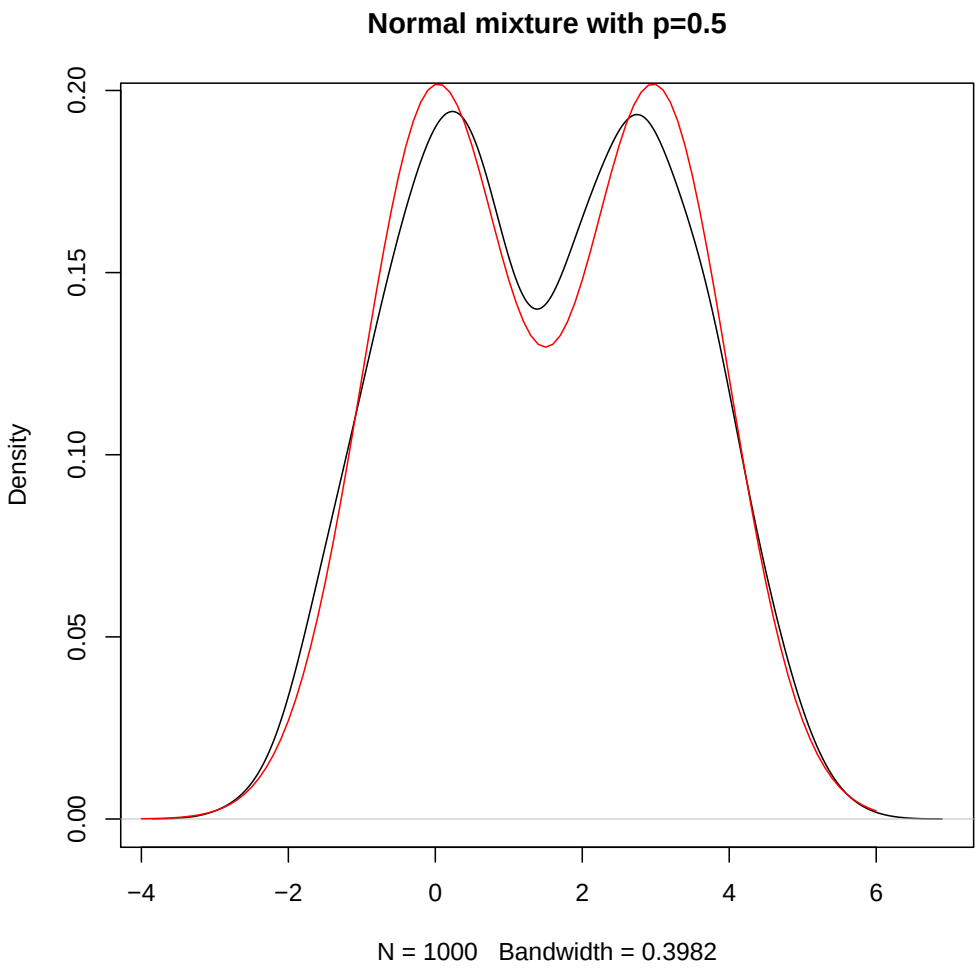
```
p1 <- 0.95

rcheck2<- normal_mixture(p1)

plot(density(rcheck2), main="Normal mixture with p
   =0.95")

lines(seq(-4,6, 0.1), p1*dnorm(seq(-4,6,0.1),0,1)
      + (1-p1)*dnorm(seq(-4,6,0.1),3,1), col="red")
```



It can be seen now that there is so much weight on the standard normal distribution that we almost can't see anymore the second normal distribution. So this one does not really look bimodal.
Maybe we should reduce p close to one half in order to get a really impressive bimodal graph:

```
p1 <- 0.5

rcheck3<- normal_mixture(p1)

plot(density(rcheck3), main="Normal mixture with p
   =0.5")
```

```
lines(seq(-4,6, 0.1), p1*dnorm(seq(-4,6,0.1),0,1)
      + (1-p1)*dnorm(seq(-4,6,0.1),3,1), col="red")
```



At last, we get a nice symmetric bimodal distribution.

To sum up, we can clearly see how the shape of the bimodal mixture depends on the probability. The higher the probability, the more "weight" will be on the respective distribution.

5 Task 70

First, let's simulate the mixture of the gamma and exponential distributions (emperical):

```
x <- rgamma(1000, 4, 2)

Y <- rexp(x)
```

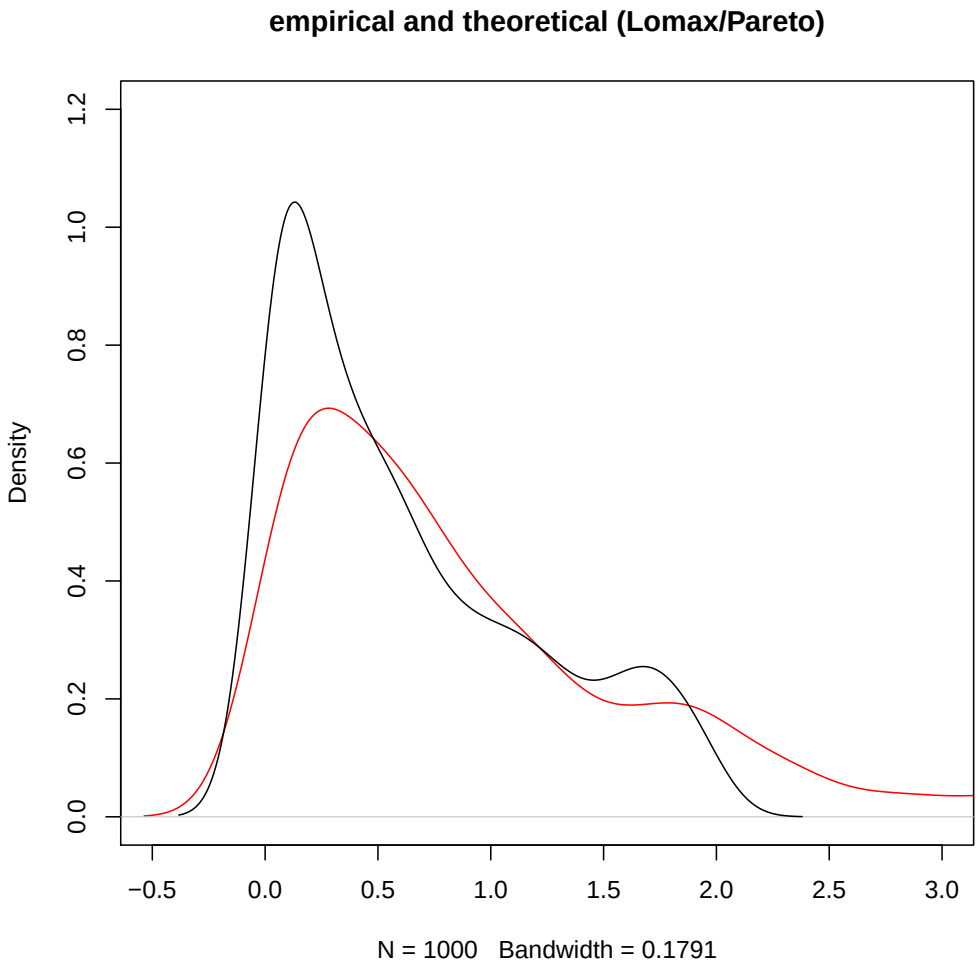
For the theoretical (Lomax/Pareto) we used the package actuar:

```
require(actuar)
```

```
## Loading required package: actuar
## Warning: package 'actuar' was built under R version 3.4.3
##
## Attaching package: 'actuar'
## The following object is masked from 'package:grDevices':
##
##     cm
```

The red one is emperical.

```
d1 <- density(Y)

d2 <- density(dpareto(Y, 4, 2))

plot(d1, main="empirical and theoretical (Lomax/Pareto
   )", col="red", ylim=c(0, 1.2), xlim=c(-0.5, 3))

lines(d2)
```



6 Task 71

```
require(mvtnorm)
## Loading required package: mvtnorm
B <- 3000

corel <- numeric(B)

kendall <- numeric(B)
```

Let's create the correlaton coefficient artifically by using an already-preadjusted scale matrix, and then calculate the correlation from the kendall coefficient.

```
for (n in 1:B){
  X <- rmvt(90, sig = matrix((3/(3-2))*c(1,0.5,0.5,1)
     ,2), df = 3)
  corel[n] <- cor(X)[1,2]
  kendall[n] <- sin(cor(X[,1],X[,2], method="kendall")
     *pi/2)
  }
```

Now we compare the standard correlation coefficient with the coefficient we gained from the Kendall rank correlation.

```
mean(corel)

## [1] 0.4903786

mean(kendall)

## [1] 0.4963534
```

We see that the Kendall method works better than the standard method (it is closer to 0.5).

7 Task 72

Poisson process can be represented as

$$Y(t) = \sum_{i=1}^{N(t)} D_i,$$

where $N(t)$ is a Poisson process and $Y_1, Y_2...$ are i.i.d. and independent of $N(t)$. In our case Y_i has gamma distribution with shape parameter.

Mean

Using a result known as Wald's equation:

$$\mathbb{E}[Y(t)] = \mathbb{E}[D_1 + \cdots + D_N] = \mathbb{E}[N]\mathbb{E}[D_1].$$

Moreover:

$$\mathbb{E}[N] = \lambda * t \Rightarrow \mathbb{E}[D(t)] = \lambda * t\mathbb{E}[D_1]$$

Variance

Using the Law of total variance: $\mathbb{E}[\mathbb{V}[Y \mid X]] + \mathbb{V}[\mathbb{E}[Y \mid X]]$ variance of compound Poisson process can be calculated as:

$$\begin{aligned}
\mathbb{V}(Y(t)) &= E(\mathbb{V}(Y(t)|N(t))) + \mathbb{V}(E(Y(t)|N(t))) \\
&= E(N(t)\mathbb{V}(D)) + \mathbb{V}(N(t)E(D)) \\
&= \mathbb{V}(D)E(N(t)) + E(D)^2\mathbb{V}(N(t)) \\
&= \mathbb{V}(D)\lambda t + E(D)^2\lambda t \\
&= \lambda t(\mathbb{V}(D) + E(D)^2) \\
&= \lambda t E(D^2).
\end{aligned}$$

```
# Task 72

compound_poisson_process <- function(T_time, lambda,
   shape, scale)
{
  t_time <- 0
  N <- 0
  X <- 0
```

```
  Y1 <- 0

  while (TRUE)
  {
    U <- runif(n = 1)
    t_time <- t_time + ((-1 / lambda) * log(U))
    if (t_time > T_time)
    {
      break
    }

    Y <- rgamma(n = 1, shape = shape, scale = scale)
    X <- X + Y
    N <- N + 1
    if (N == 1)
    {
      Y1 <- Y

    }
  }
  return(c(X, Y1))
}


size <- 100000


# Set of parameters 1

T_time_1 <- 10

lambda_1 <- 1

shape_1 <- 1

scale_1 <- 2


X_vector <- rep(0, size)

sum_Y1 <- 0
```

```
sum_Y1_squared <- 0


for (i in 1:size)
{
  X_Y1_vector <- compound_poisson_process(T_time_1,
     lambda_1, shape_1, scale_1)
  X_vector[i] <- X_Y1_vector[1]
  sum_Y1 <- sum_Y1 +  X_Y1_vector[2]
  sum_Y1_squared <- sum_Y1_squared + X_Y1_vector[2]^2
}


mean_empirical_1 <- mean(X_vector)

var_empirical_1 <- var(X_vector)

mean_theoretical_1 <- lambda_1*T_time_1*sum_Y1/size

var_theoretical_1 <- lambda_1*T_time_1*sum_Y1_squared/
   size


# Set of parameters 2

T_time_2 <- 10

lambda_2 <- 1

shape_2 <- 5

scale_2 <- 1


X_vector <- rep(0, size)

sum_Y1 <- 0

sum_Y1_squared <- 0
```

```
for (i in 1:size)
{
  X_Y1_vector <- compound_poisson_process(T_time_2,
     lambda_2, shape_2, scale_2)
  X_vector[i] <- X_Y1_vector[1]
  sum_Y1 <- sum_Y1 +  X_Y1_vector[2]
  sum_Y1_squared <- sum_Y1_squared + X_Y1_vector[2]^2
}


mean_empirical_2 <- mean(X_vector)

var_empirical_2 <- var(X_vector)

mean_theoretical_2 <- lambda_2*T_time_2*sum_Y1/size

var_theoretical_2 <- lambda_2*T_time_2*sum_Y1_squared/
   size


cat("Set of parameters 1: T = ", T_time_1, ", lambda =
    ", lambda_1, ", shape = ",
    shape_1, ", scale = ", scale_1, '\n')

## Set of parameters 1: T =  10 , lambda =  1 , shape
   =  1 , scale =  2

cat("The empirical mean:", mean_empirical_1, '\n')

## The empirical mean: 19.97183

cat("The theoretical mean:", mean_theoretical_1, '\n')

## The theoretical mean: 19.92938

cat("The empirical variance:", var_empirical_1, '\n')

## The empirical variance: 79.67395

cat("The theoretical variance:", var_theoretical_1, '\
   n')

## The theoretical variance: 79.32
```

```
cat("\nSet of parameters 2: T = ", T_time_2, ", lambda
    = ", lambda_2, ", shape = ",
    shape_2, ", scale = ", scale_2, '\n')

##
## Set of parameters 2: T =  10 , lambda =  1 , shape
   =  5 , scale =  1

cat("The empirical mean:", mean_empirical_2, '\n')

## The empirical mean: 50.02746

cat("The theoretical mean:", mean_theoretical_2, '\n')

## The theoretical mean: 49.9642

cat("The empirical variance:", var_empirical_2, '\n')

## The empirical variance: 299.5416

cat("The theoretical variance:", var_theoretical_2, '\
   n')

## The theoretical variance: 299.7337
```

8 Task 73

To calculate the integrals we need to recall that expected value of a function of a random variable f(X) can be defined as:

$$E[f(X)] = \int f(X)P_X(X)dX$$

Where $P_X(X)$ is the probability distribution of the random variable X.

Using the idea of Monte Carlo estimator, we have next result:

$$\int_a^b f(X)dX = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(X_i).$$

So, all we need is to draw a sample of $x_1, \ldots, x_n$ from uniform distribution and use the equation above.

a)

Calculate the integral below:

$$\int_1^3 x^2 dx,$$

To calculate it let's use a random variable $X_{i \in N} \sim U(1,3)$ and 10k trials.

```
n <- 10000

x <- runif(n, 1,3)

integral_1 <- mean(x^2)*2

c(integral_1, 9-1/3)

## [1] 8.710404 8.666667
```

As we can see empirical and theoretical results are very close.

b)

Calculate the integral below:

$$\int_0^\pi sin(x) dx,$$

```
n <- 10000

x <- runif(n, 0, pi)

integral_2<- mean(sin(x))*pi

c(integral_2,  2)

## [1] 2.012659 2.000000
```

As we can see empirical and theoretical results are very close.

c)

Calculate the integral below:

$$\int_0^\infty e^{-x} dx$$

But we can still use Monte Carlo inegration. First we draw from the exponential and instead of dividing by $(b - a)$, we divide by the density of the exponential.

```
n <- 10000

x <- rexp(n)

integral_3 <- mean(exp(-(x) )/ dexp(x))

c(integral_3 , 1)

## [1] 1 1
```

Here the results are the same!

d)

Calculate the integral below:

$$\int_0^3 sin(e^x) dx,$$

```
n <- 1000000

x <- runif(n, 0, 3)

integral_4 <- mean(sin(exp(x))) * 3

c(integral_4 ,0.6061244734187699)

## [1] 0.6088091 0.6061245
```

To be close more trials are required, but finally the result is good.

e)

Calculate the integral below:

$$\int_0^2 \frac{1}{\sqrt{2\pi}} e^{x^2/2} dx$$

```
n <- 10000

x <- runif(n, 0, 2)

integral_5 <- mean(1 / (sqrt(2 * pi)) * exp(-x ^ 2 /
   2) * 2)

c(integral_5, pnorm(2)-pnorm(0))

## [1] 0.4785244 0.4772499
```

Here we have a close result too.

9 Task 74

We need to estimate:

$$\theta = \int_0^{0.5} e^{-x} dx$$

Uniform distribution

Uniform

```
n <- 10000

x <- runif(n,max = 0.5, min = 0)

theta_hat <- mean(exp(-x))*0.5

c(theta_hat, -1/sqrt(exp(1)) + 1)

## [1] 0.3934276 0.3934693

var(exp(-x))

## [1] 0.01294584
```

Exponential

```
y <- rexp(n,10)

y <- y[y < 0.5]

theta_hat2 <- mean(exp(-y) / dexp(y,rate = 10))

c(theta_hat2,-1 / sqrt(exp(1)) + 1)

## [1] 0.3960230 0.3934693

var(exp(-y) / dexp(y,rate = 10))

## [1] 0.5535403
```

As we can see variance of Exponential is much bigger. This is because exponential function is much more sensitive.

10 Task 75

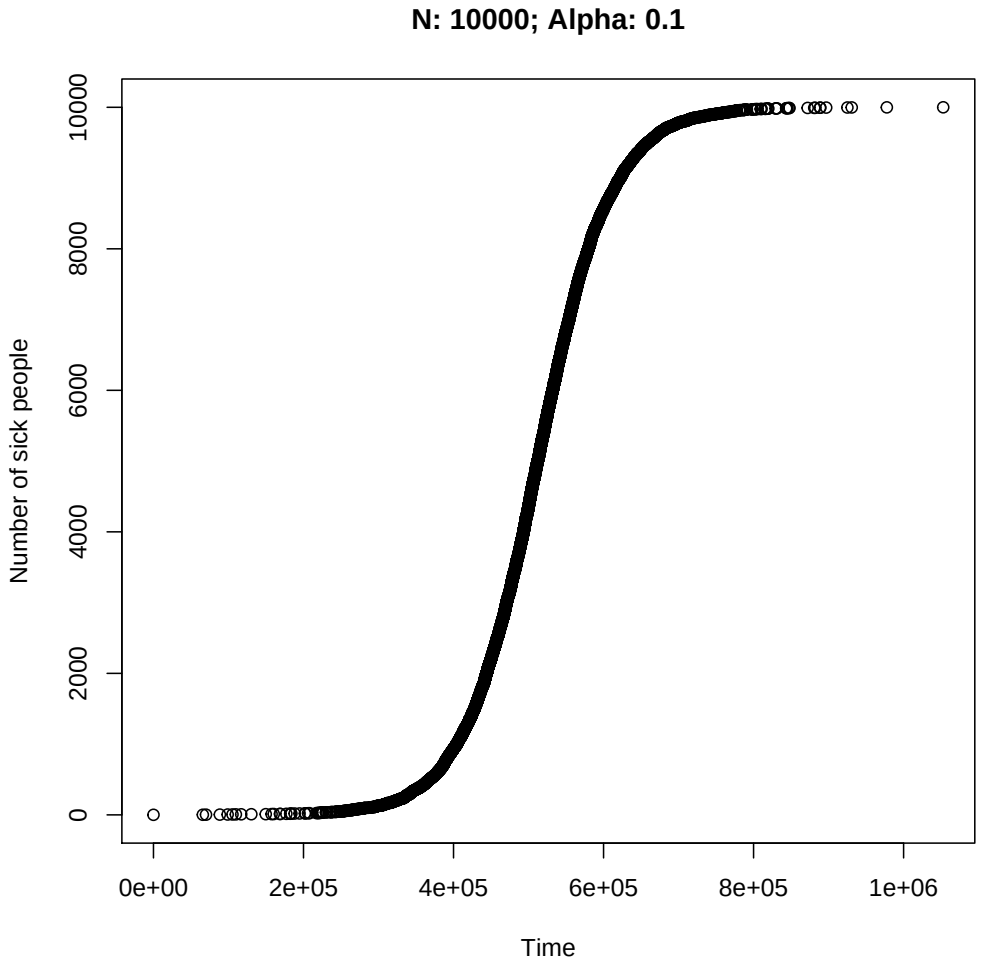
```
# Task 75

# Defaults follow binomial distribution


AA_size <- 10

A_size <- 25

BBB_size <- 96


AA_pd <- 0.0001

A_pd <- 0.0005

BBB_pd <- 0.0025


n <- 1000


AA_defaults <- rbinom(n, AA_size, AA_pd)

A_defaults <- rbinom(n, A_size, A_pd)

BBB_defaults <- rbinom(n, BBB_size, BBB_pd)


cat("Minimal number of defaults of AA obligors (1000
   simulations)", min(AA_defaults), "\n")

## Minimal number of defaults of AA obligors (1000
   simulations) 0

cat("Maximal number of defaults of AA obligors (1000
   simulations)", max(AA_defaults), "\n")

## Maximal number of defaults of AA obligors (1000
   simulations) 0
```

```
cat("Minimal number of defaults of A obligors (1000
   simulations)", min(A_defaults), "\n")

## Minimal number of defaults of A obligors (1000
   simulations) 0

cat("nMaximal number of defaults of A obligors (1000
   simulations)", max(A_defaults), "\n")

## nMaximal number of defaults of A obligors (1000
   simulations) 2

cat("Minimal number of defaults of BBB obligors (1000
   simulations)", min(BBB_defaults), "\n")

## Minimal number of defaults of BBB obligors (1000
   simulations) 0

cat("Maximal number of defaults of BBB obligors (1000
   simulations)", max(BBB_defaults), "\n")

## Maximal number of defaults of BBB obligors (1000
   simulations) 3
```

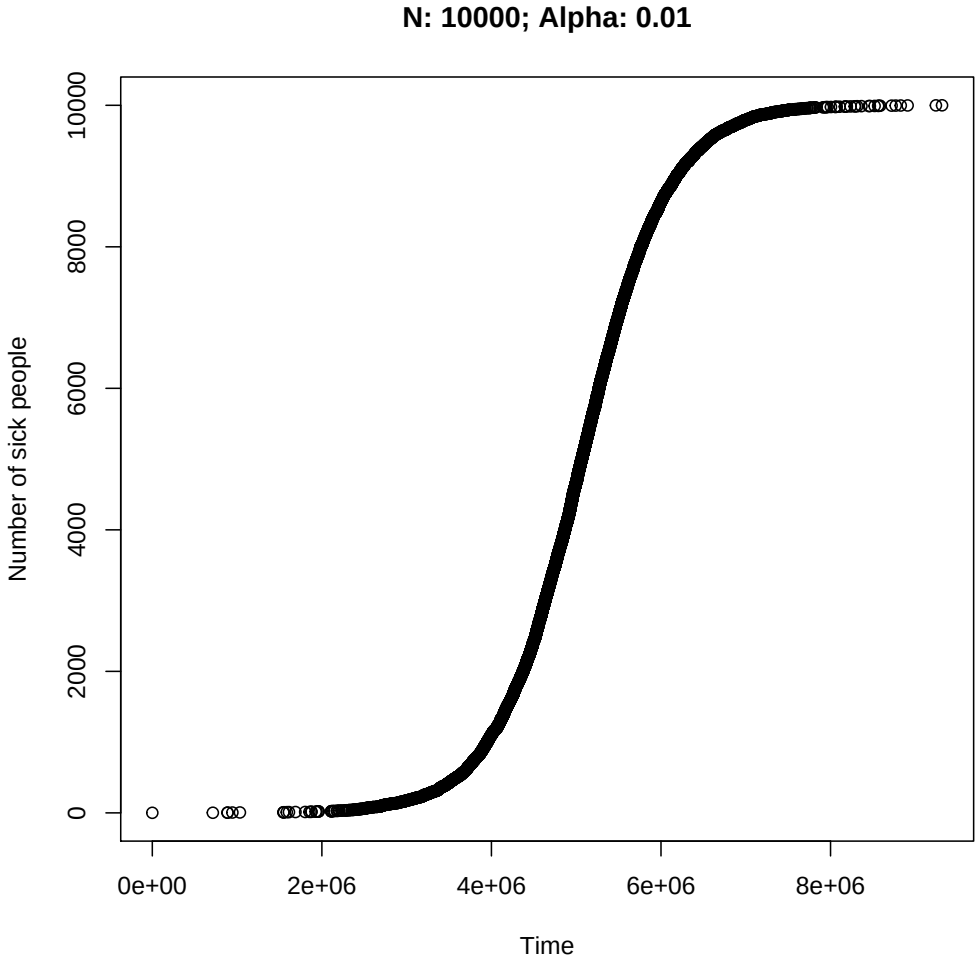
11 Task 78

```
# Task 78

# a

simulate_78a <- function(N, alpha)
{
  time.line <- 0
  time.current <- 0
  n.sick <- 1

  while (n.sick < N)

  {
    time.current <- time.current + 1
    #if encounters==0, then none of the two persons is
        sick
    #if encounters==1, then one of the two persons is
       sick
    #if encounters==2, then both of the two persons
       are sick
    encounters <- rhyper(
      nn = 1,
      m = n.sick,
      n = N - n.sick,
      k = 2
    )

    if (encounters == 1)
    {
      contage <- rbinom(n = 1, size = 1, alpha)
      if (contage == 1)
      {
        #record current time
        time.line <- c(time.line, time.current)
        n.sick <- n.sick + 1
      }
    }
  }
```

```
  time.line
}


# Test

N <- 10000

alpha_1 <- 0.1

result_1 <- simulate_78a(N, alpha_1)

title_1 <- "N: 10000; Alpha: 0.1"

plot(result_1,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title_1)
```




```
alpha_2 <- 0.01

result_2 <- simulate_78a(N, alpha_2)

title_2 <- "N: 10000; Alpha: 0.01"

plot(result_2,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title_2)
```

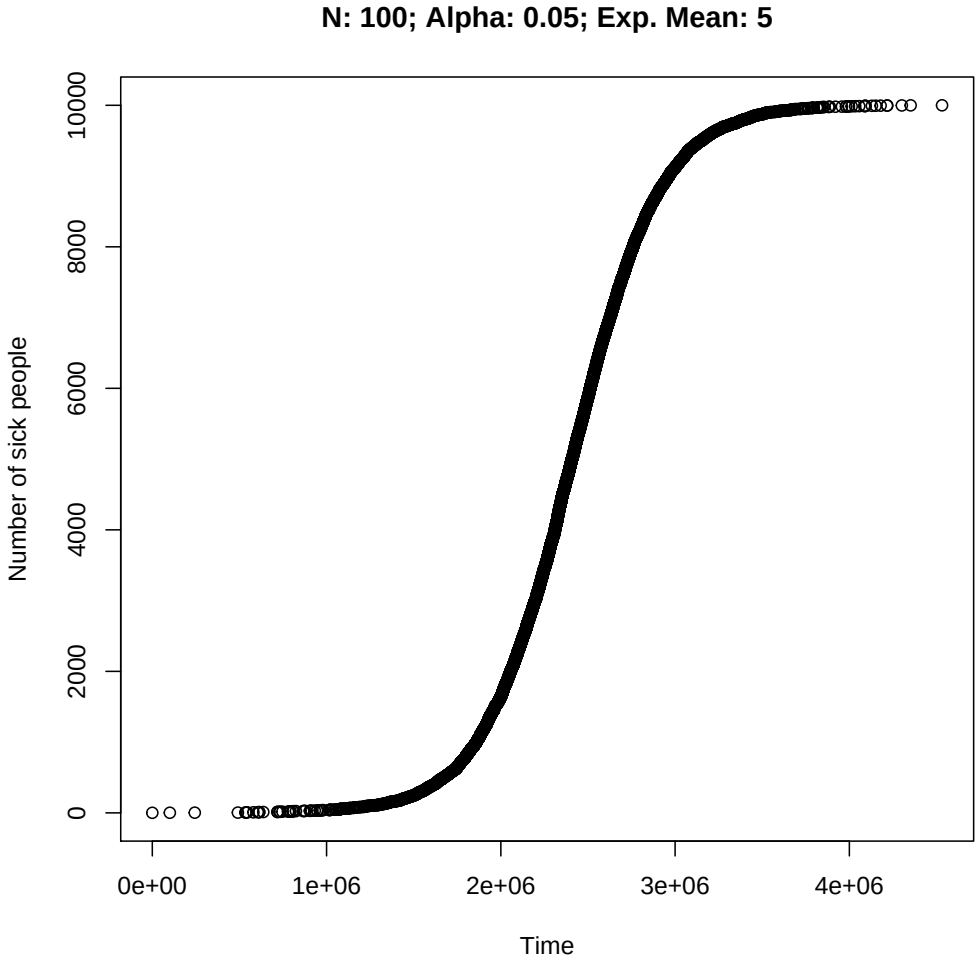



```
# b

find_time_people_infected <-
  function(N, number_of_people_infected, alpha)
  {
    if (number_of_people_infected > N)
    {
      return("Number of infected people cannot be
         greater than number of people")
    }
    time.line <- 0
    time.current <- 0
```

```
    n.sick <- 1

    while (n.sick < N)

    {
      time.current <- time.current + 1
      #if encounters==0, then none of the two persons
         is sick
      #if encounters==1, then one of the two persons
         is sick
      #if encounters==2, then both of the two persons
         are sick
      encounters <- rhyper(
        nn = 1,
        m = n.sick,
        n = N - n.sick,
        k = 2
      )

      if (encounters == 1)
      {
        contage <- rbinom(n = 1, size = 1, alpha)
        if (contage == 1)
        {
          #record current time
          time.line <- c(time.line, time.current)
          n.sick <- n.sick + 1
          if (n.sick == number_of_people_infected)
          {
            return (time.current)
          }
        }
      }
    }
  }


# Find time until 1000 people are infected

N <- 10000
```

```
alpha <- 0.1

number_of_people_infected <- 1000

find_time_people_infected(N, number_of_people_infected
   , alpha)

## [1] 398363

# d

simulate_78d <- function(N, alpha, exp_mean)
{
  set.seed(1)
  time.line <- 0
  time.current <- 0
  n.sick <- 1

  while (n.sick < N)

  {
    time.current <- time.current + rexp(n = 1, rate =
       1 / exp_mean)
    #if encounters==0, then none of the two persons is
        sick
    #if encounters==1, then one of the two persons is
       sick
    #if encounters==2, then both of the two persons
       are sick
    encounters <- rhyper(
      nn = 1,
      m = n.sick,
      n = N - n.sick,
      k = 2
    )

    if (encounters == 1)
    {
      contage <- rbinom(n = 1, size = 1, alpha)
      if (contage == 1)
      {
```

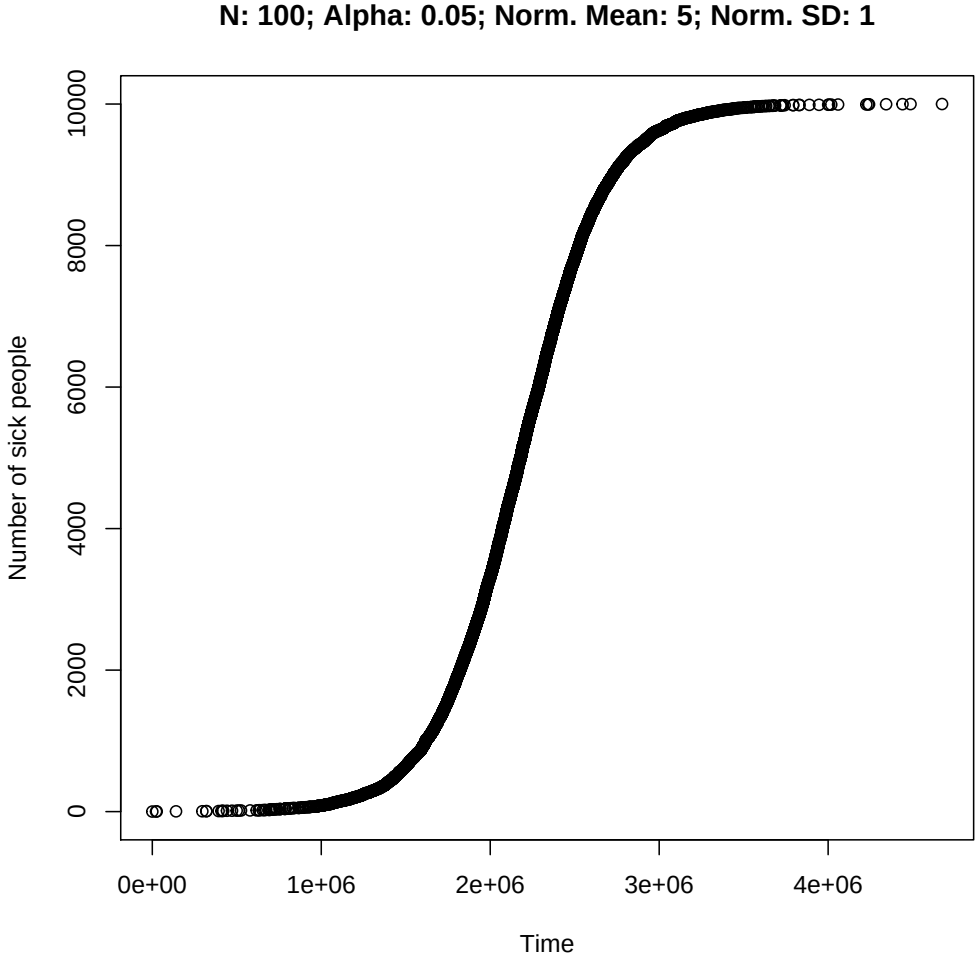
```
        #record current time
        time.line <- c(time.line, time.current)
        n.sick <- n.sick + 1
      }
    }
  }
  time.line
}


# Test

N <- 10000

alpha <- 0.1

exp_mean <- 5

result <- simulate_78d(N, alpha, exp_mean)

title <- "N: 100; Alpha: 0.05; Exp. Mean: 5"

plot(result,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title)
```




```
# e

simulate_78e <- function(N, alpha, norm_mean, norm_sd)
{
  time.line <- 0
  time.current <- 0
  n.sick <- 1

  while (n.sick < N)

  {
    time.current <-
```

```
      time.current + rnorm(n = 1, mean = norm_mean, sd
          = norm_sd)
    #if encounters==0, then none of the two persons is
        sick
    #if encounters==1, then one of the two persons is
       sick
    #if encounters==2, then both of the two persons
       are sick
    encounters <- rhyper(
      nn = 1,
      m = n.sick,
      n = N - n.sick,
      k = 2
    )

    if (encounters == 1)
    {
      contage <- rbinom(n = 1, size = 1, alpha)
      if (contage == 1)
      {
        #record current time
        time.line <- c(time.line, time.current)
        n.sick <- n.sick + 1
      }
    }
  }
  time.line
}


# Test

N <- 10000

alpha <- 0.1

norm_mean <- 5

norm_sd <- 1

result <- simulate_78e(N, alpha, norm_mean, norm_sd)
```

```
title <- "N: 100; Alpha: 0.05; Norm. Mean: 5; Norm. SD
   : 1"

plot(result,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title)
```

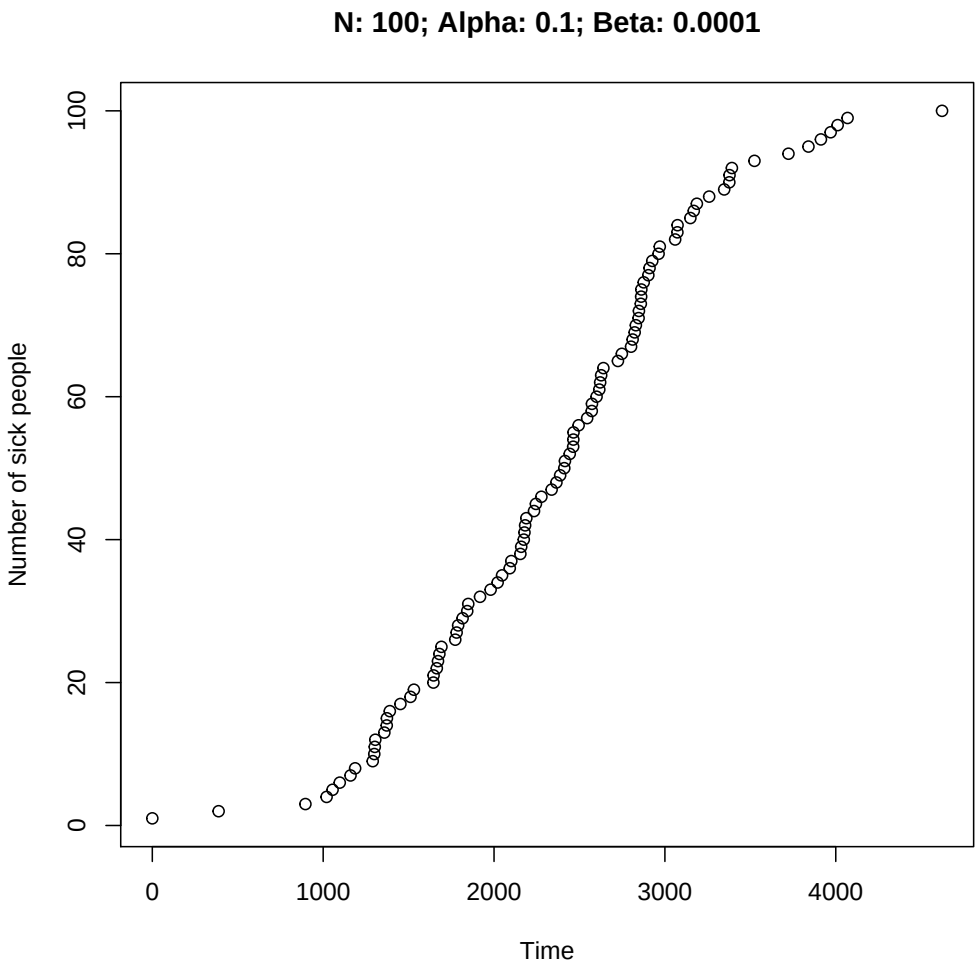



```
#c
```

```
simulate_78c <- function(N, alpha, beta)
{
  time.line <- 0
  time.current <- 0
  n.sick <- 1

  while (n.sick < N)

  {
    time.current <- time.current + 1
    #if encounters==0, then none of the two persons is
        sick
    #if encounters==1, then one of the two persons is
       sick
    #if encounters==2, then both of the two persons
       are sick
    encounters <- rhyper(
      nn = 1,
      m = n.sick,
      n = N - n.sick,
      k = 2
    )

    if (encounters == 1)
    {
      # check whether infected person is recovering
         now
      recov_1 <- rbinom(n = 1, size = 1, beta)
      if (recov_1 == 1)
        # infected person is recovering
      {
        n.sick <- n.sick - 1
      }
      else
        # infected person is not recovering
      {
        infect_2 <- rbinom(n = 1, size = 1, alpha)
        if (infect_2 == 1)
        {
          #record current time
```

```
          time.line <- c(time.line, time.current)
          n.sick <- n.sick + 1
        }
      }
    }
    if (encounters == 2)
    {
      # check whether infected person 1 is recovering
         now
      recov_1 <- rbinom(n = 1, size = 1, beta)
      if (recov_1 == 1)
        # infected person 1 is recovering
      {
        n.sick <- n.sick - 1
      }

      # check whether infected person 2 is recovering
         now
      recov_2 <- rbinom(n = 1, size = 1, beta)
      if (recov_2 == 1)
        # infected perso 2 is recovering
      {
        n.sick <- n.sick - 1
      }
    }


  }
  time.line
}


# Test

N <- 100

alpha_1 <- 0.1

beta_1 <- 0.0001

result_1 <- simulate_78c(N, alpha_1, beta_1)
```

```
title_1 <- "N: 100; Alpha: 0.1; Beta: 0.0001"

plot(result_1,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title_1)
```




```
alpha_2 <- 0.05

beta_2 <- 0.00001
```

```
result_2 <- simulate_78c(N, alpha_2, beta_2)

title_2 <- "N: 100; Alpha: 0.05; Beta: 0.00001"

plot(result_2,
     1:N,
     xlab = 'Time',
     ylab = 'Number of sick people',
     main = title_2)
```