

Assignment 2

Bjørn Christian Weinbach

16th October, 2020

Clear R environment

```
rm(list = ls())
```

Problem 1

Consider the integral

$$\int_{-1}^1 \int_{-1}^1 1_D(x, y) dx dy$$

Where $1_D(x, y)$ is the indicator function defined so that

$$1_D(x, y) = \begin{cases} 1 & \text{if } x^2 + y^2 \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

As a crude first attempt, consider the estimator

$$\theta_{CMC} = \frac{4}{N} \sum_{i=1}^n 1_D(X_i, Y_i)$$

A.) Argue for why θ_{CMC} is a monte carlo estimator for the integral above.

According to [this](#) wikipedia article which was accessed the 9th October, 2020, the monte carlo estimate for a multidimensional definite integral (Wikipedia contributors 2020)

$$I = \int_{\Omega} f(\bar{x}) d\bar{x}$$

where Ω is a subset of R^m , has volume

$$V = \int_{\Omega} d\bar{x}$$

The naive Monte Carlo approach is to sample points uniformly on Ω given N uniform samples,

$$\bar{x}_1, \dots, \bar{x}_n \in \Omega,$$

I can be approximated by

$$I \approx \Omega_N \equiv V \frac{1}{N} \sum_{i=1}^N f(\bar{x}_i)$$

Which is true due to the law of large numbers.

In our case, which is also very similar to the [example](#) on the wikipedia article for monte carlo integration, $\Omega = [-1, 1] \times [-1, 1]$ with $V = \int_{-1}^1 \int_{-1}^1 dx dy = 4$ which gives the following crude way to estimate I (Wikipedia contributors 2020)

$$I = \frac{4}{N} \sum_{i=1}^N 1_D(X_i, Y_i)$$

Which is the proposed estimator θ_{CMC} .

A.) Show that $1_D(X_i, Y_i)$ has a bernoulli distribution with $p = \frac{\pi}{4}$

The function returns 1 / success or 0 / failure, and is therefore has a potential bernoulli distribution. To calculate $P(X^2 + Y^2 \leq 1)$ we need to calculate the integral

$$P(X, Y \in X^2 + Y^2 \leq 1) = \int_A f_{XY}(x, y) dx dy$$

Where A is the region of the $X^2 + Y^2 \leq 1$ i.e disc with radius 1. The pdf of a bivariate uniform distribution is a single constant c . To find this constant, we use the fact that the probability of a point being inside the sample space Ω is 1 and

$$\int_{\Omega} f_{XY}(x, y) dx dy = 1 \implies c \int_{\Omega} dx dy = 1$$

Where the integral is the area of the sample space and we get

$$c \cdot \text{area of square} = 1 \implies c = \frac{1}{\text{area of square}} \implies f_{xy} = c = \frac{1}{4}$$

and then the integral of a point being inside the unit square is

$$\int_A f_{XY}(x, y) dx dy = c \cdot \text{area of circle}$$

and we get $\frac{\pi}{4}$. The indicator function therefore is a bernoulli distribution with probability $\frac{\pi}{4}$ of being inside the unit circle.

b.) Which distribution does $Z = \sum_{i=1}^n 1_d(X_i, Y_i)$ have

If we sum N independent bernoulli samples/trials with constant p we have a binomial distribution. P is constant as discussed above and the trials are independent due to X and Y being independent uniform variables.

The sum is therefore a valid binomial distributed random variable $Z \sim B(n, \frac{\pi}{4})$. The expected value of a binomial distribution is $E[Y] = n \cdot p = \frac{n\pi}{4}$

The expected value of the estimator θ_{CMC} is then

$$E[\theta_{CMC}] = \frac{4}{n} \cdot Y = \frac{4}{n} \cdot E[Y] = \frac{4}{n} \cdot \frac{n\pi}{4} = \pi$$

And we see that the the estimator has expected value π .

The variance of the estimator θ_{CMC} is

$$Var(\theta_{CMC}) = Var(\frac{4}{n}Y) = \frac{16}{n^2} Var(Y) = \frac{16}{n^2} n \frac{\pi}{4} (1 - \frac{\pi}{4}) = \frac{4(1 - \frac{\pi}{4})\pi}{n}$$

c) Implementation of monte carlo estimate of θ_{CMC} with $N = 1000$

```

indicator1 <- function (x, y) {           # Indicator function for unit circle
  return((x^2+y^2 <= 1))
}

mcEstimatePi <- function (Sims) {         # function for monte carlo estimate
  mcPi <- numeric(Sims)
  for (i in 1:Sims) {
    N <- 1000                             # Number of samples
    x <- runif(N, -1, 1)                   # x values from uniform
    y <- runif(N, -1, 1)                   # y values from uniform
    mcPi[i] <- (4/N)*sum(indicator1(x, y)) # return monte carlo estimate
  }
  return(mcPi)
}

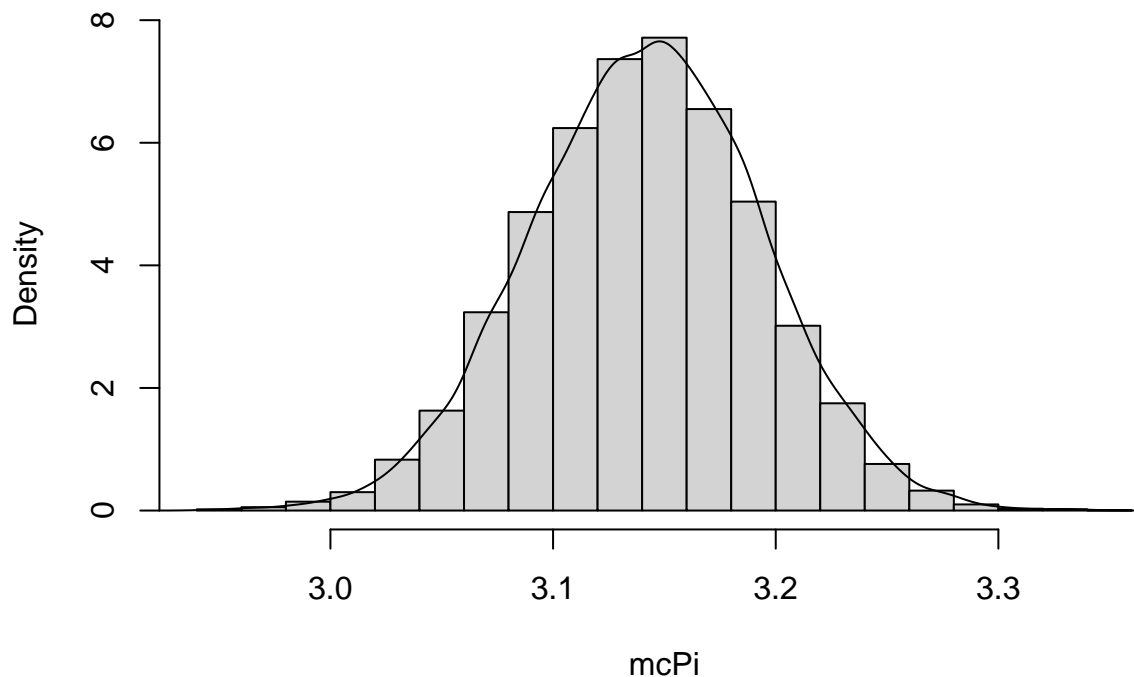
Sims <- 10000                             # Simulations of pi
mcPi <- mcEstimatePi(Sims)                 # Function call
summary(mcPi)                             # Summary stats for MC estimate

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.944   3.108   3.144   3.142   3.176   3.344

hist(mcPi, probability = TRUE)             # Histogram
lines(density(mcPi))                       # Plot empirical density

```

Histogram of mcPi



```

mean <- mean(mcPi)
var <- var(mcPi)

```

```

exact_mean <- pi
exact_var <- (4*(1 - (pi/4))*pi / Sims)
print(paste("Simulated mean: ", mean))

```

```
## [1] "Simulated mean: 3.1419084"
```

```
print(paste("Exact mean: ", exact_mean))
```

```
## [1] "Exact mean: 3.14159265358979"
```

```
print(paste("Simulated var: ", var))
```

```
## [1] "Simulated var: 0.00267279968940895"
```

```
print(paste("Exact var: ", exact_var))
```

```
## [1] "Exact var: 0.000269676621326981"
```

We see that the distribution is approximately normally distributed with sample mean at approx $Mean = 3.141$ and sample variance at approx $Variance = 0.00260$. This is due to the central limit theorem because our estimate of π is based on a sum of several samples.

d.) Calculate probability of correctly estimating to two decimal places

```

# count average of booleans (probability) of estimates from 3.14 to 3.14999...
mean(mcPi < 3.15) - mean(mcPi <= 3.14)

```

```
## [1] 0.0665
```

e.) Introducing antithetic variables

The assignment proposes two antithetic variables $V = a + b - X = -X$ and $W = -Y$

These will not reduce the monte carlo variance since X and Y are independent uniform random variables and due to their independence there is no negative correlation gained by just flipping the sign of both variables from X and Y to $-X$ and $-Y$.

```

x <- runif(1000, -1, 1)
y <- runif(1000, -1, 1)
cov(-x, -y)

```

```
## [1] -0.009770841
```

f.) Let's see if our variance is reduced by doing exercise c. with the new variables.

```

mcEstimatePi <- function (Sims) {          # function for monte carlo estimate
  mcPi <- numeric(Sims)
  for (i in 1:Sims) {
    N <- 1000                               # Number of samples
    x <- runif(N, -1, 1)                     # x values from uniform
    y <- runif(N, -1, 1)                     # y values from uniform
    mcPi[i] <- (4/N)*sum(indicator1(-x, -y)) # mcEstimate with new variables
  }
  return(mcPi)
}

Sims <- 10000                                # Simulations of pi
mcPi <- mcEstimatePi(Sims)                    # Function call
summary(mcPi)                                # Summary stats for MC estimate

```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.936   3.108   3.140   3.141   3.176   3.328
```

```
mean(mcPi)
```

```
## [1] 3.141311
```

```
var(mcPi)
```

```
## [1] 0.002656518
```

g.) Check if shift function reduces variance

```
shift <- function(u) {                                # Introducing shift function
  return(((u+2.0) %% 2.0) - 1.0)
}

mcEstimatePi <- function (Sims) {                     # function for monte carlo estimate
  mcPi <- numeric(Sims)
  for (i in 1:Sims) {
    N <- 1000                                         # Number of samples
    x <- runif(N, -1, 1)                             # x values from uniform
    y <- runif(N, -1, 1)                             # y values from uniform
    sx <- shift(x)
    sy <- shift(y)
    mcPi[i] <- (4/N)*sum(indicator1(sx, sy)) # mcEstimate with new variables
  }
  return(mcPi)
}

Sims <- 10000                                         # Simulations of pi
mcPi <- mcEstimatePi(Sims)                           # Function call
summary(mcPi)                                        # Summary stats for MC estimate
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.928   3.108   3.140   3.141   3.176   3.324
```

```
mean(mcPi)
```

```
## [1] 3.141308
```

```
var(mcPi)
```

```
## [1] 0.002691387
```

h.) Use important sampling

```
# zero mean independent Normal
f <- function (x, y, sigma) {
  const <- (1/(2*pi*sigma^2))
  exp1 <- exp(-x^2/(2*sigma^2))
  exp2 <- exp(-y^2/(2*sigma^2))
  return(const*exp1*exp2)
}

mcEstimatePi <- function (Sims, sigma) {             # function for monte carlo estimate
  mcPi <- numeric(Sims)
  for (i in 1:Sims) {
    N <- 1000                                         # Number of samples
    x <- rnorm(N, 0, sigma^2)                        # x values from uniform
```

```

    y <- rnorm(N, 0, sigma^2) # y values from uniform
    mcPi[i] <- mean(indicator1(x, y) / f(x, y, sigma))
  }
  return(mcPi)
}

```

```
sigmas <- c(0.3, 0.624127, 1.0)
```

```

for (s in sigmas) {
  mcPi <- mcEstimatePi(10000, s)
  print(summary(mcPi))
}

```

```

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.6138 0.6224 0.6244 0.6245 0.6266 0.6381
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 3.451 3.692 3.751 3.754 3.813 4.109
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 2.768 3.121 3.209 3.210 3.298 3.755

```

The estimators are not that good and i was not able to replicate the results described in the assignment, which makes my findings quite uncertain. My best answer to why this estimator did not work is mainly due to tail behaviour and $f(x)$ not being an appropriate function for this use.

But due to the fact that i was not able to replicate the results also suggests a faulty implementation above.

Problem 2

Define $\lambda(t)$ in R

```

# Specify the intensity function for storms
lambdastorm <- function(t) {
  (297 / 10) * (1 + cos(2*pi*(t + (1/10)))) * (1 - (exp(-t/10)/2)) + 3/5
}

lambdastorm(0.5)

```

```
## [1] 3.574416
```

a.) Calculating number of storms, expected value and variance

According to Rizzo on page 103. A poisson process with a intensity function $\lambda(t)$ has the property that the number of events $N(t)$ in interval $[0, t]$ has the poisson distribution with mean

$$E[N(t)] = \int_0^t \lambda(y) dy$$

Which in our case gives

```
integrate(lambdastorm, 0, 1)
```

```
## 16.29763 with absolute error < 1.8e-13
```

Let's find the expected number of storms in 2025 by calculating the integral

```
integrate(lambdastorm, 5, 6)
```

```
## 21.80713 with absolute error < 2.4e-13
```

And let's find the expected value and standard deviation of storms in 2020 and 2021 combined

Expected value is calculated using the integral below

```
integrate(lambdastorm, 0, 2)
```

```
## 33.92776 with absolute error < 2.3e-08
```

Which means the number of events in 2020 and 2021 combined is poisson distributed with $\lambda = 33.92776$. The variance of a poisson distribution is $Var(X) = \lambda$ (according to Rizzo on page 44).

$$SD(X) = \sqrt{Var(X)} = \sqrt{33.92776} = 5.824754$$

b.) Find smallest possible λ_{max} for all $\lambda(t)$, $t \geq 0$

The function $\lambda(t)$ does not have a global maximum because it is modeled with a increasing winter intensity due to climate change that does not stop increasing. Solving for $\frac{d}{dt}\lambda(t) = 0$ gives an infinite number of potential maximum or minimum points and there is no λ_{max} for all $\lambda(t)$ values.

c.) Validate previous points by simulation

simtNHPP borrowed from lectures on stochastic processes

```
# Function for simulating arrival times for a NHPP between a and b using thinning
simtNHPP <- function(a,b,lambdamax,lambdafunc){
  # Simple check that a not too small lambdamax is set
  if(max(lambdafunc(seq(a,b,length.out = 100)))>lambdamax)
    stop("lambdamax is smaller than max of the lambdafunction")

  # First simulate HPP with intensity lambdamax on a to b
  expectednumber <- (b-a)*lambdamax
  # Simulate more than the expected number to be certain to exceed stoptime
  Nsim <- 3*expectednumber
  # Simulate interarrival times
  timesbetween <- rexp(Nsim,lambdamax)
  # Calculate arrival times starting at a
  timesto <- a+cumsum(timesbetween)
  # Discard the times larger than b
  timesto <- timesto[timesto<b]
  # Count the number of events
  Nevents <- length(timesto)
  # Next do the thinning. Only keep the times where u<lambda(s)/lambdamax
  U <- runif(Nevents)
  timesto <- timesto[U<lambdafunc(timesto)/lambdamax]
  timesto # Return the remaining times
}

Nsim <- 1000
a <- 0
b <- 1
NHPPnumbers <- vector(length=Nsim)
for(i in 1:Nsim) {
  NHPPnumbers[i] <- length(simtNHPP(a=a,b=b,
    lambdamax=max(lambdastorm(seq(a, b, 0.01))),
    lambdafunc=lambdastorm))
}
```

```

# Exepcted number of storms in 2020
mean(NHPPnumbers)

## [1] 16.059

Nsim <- 1000
a <- 5
b <- 6
NHPPnumbers <- vector(length=Nsim)
for(i in 1:Nsim) {
  NHPPnumbers[i] <- length(simtNHPP(a=a, b=b,
                                   lambdamax=max(lambdastorm(seq(a, b, 0.01))),
                                   lambdafunc=lambdastorm))
}

# Exepcted number of storms in 2025
mean(NHPPnumbers)

## [1] 21.636

Nsim <- 1000
a <- 0
b <- 2
NHPPnumbers <- vector(length=Nsim)
for(i in 1:Nsim) {
  NHPPnumbers[i] <- length(simtNHPP(a=a, b=b,
                                   lambdamax=max(lambdastorm(seq(a, b, 0.01))),
                                   lambdafunc=lambdastorm))
}

# Expected number of storms in 2020 and 2021
mean(NHPPnumbers)

## [1] 34.065

# Variance of number of storms in 2020 and 2021
var(NHPPnumbers)

## [1] 30.91769

# Standard deviation of number of storms in 2020 and 2021
sd(NHPPnumbers)

## [1] 5.560368

```

d.) Simulate claim size

To calculate the claim size for a given year, simulate a poisson process and calculate mean parameter for all storms, then draw claim size for exponential distribution with mean parameter $c(t_i)$ where t_i is time of a given storm simulated from the NHPP.

```

# mean function
claim <- function (t) {
  return(10*exp(5*t / 100))
}

# function for simulatin Nsim claims, from a to b
simulate_claims <- function(Nsim, a, b) {
  Claims <- vector(length=Nsim)

```

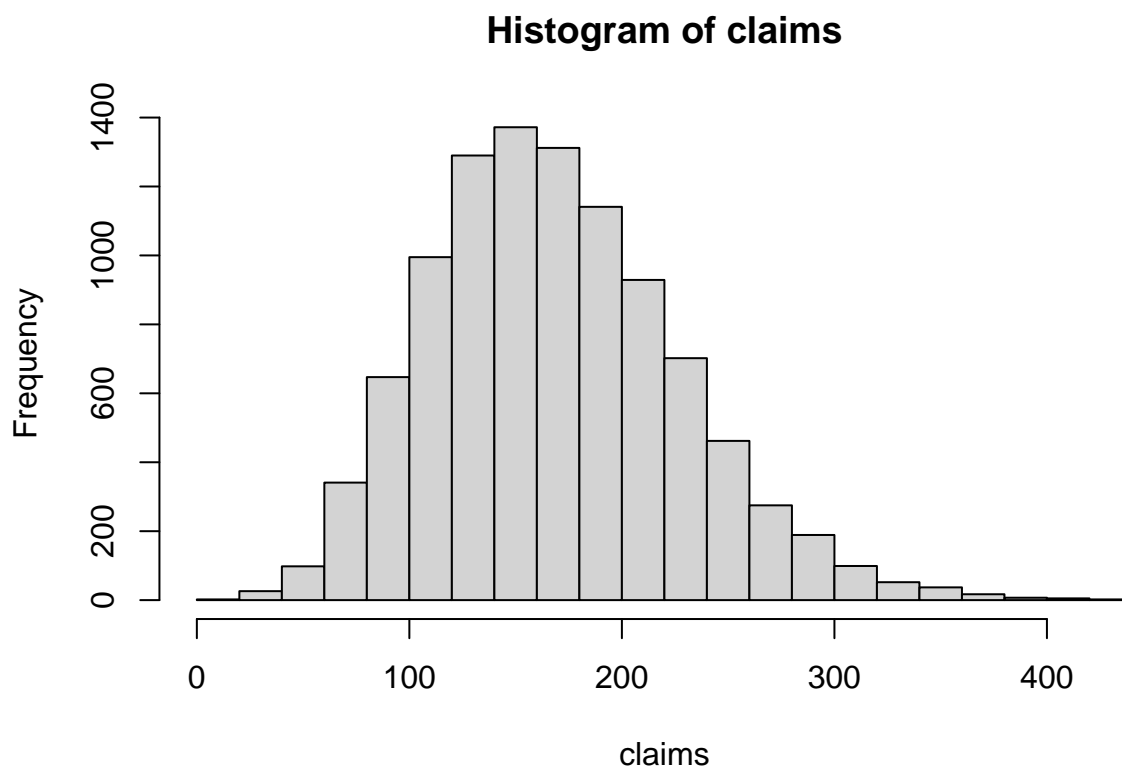


```

for(i in 1:Nsim) {
  # Calculate mean parameter of storm based on time of storm
  expmean <- claim(simtNHPP(a=a, b=b,
    lambdamax=max(lambdastorm(seq(a, b, 0.01))),
    lambdafunc=lambdastorm))
  # Draw claim size for all storms from exponential with calculated mean param
  Claims[i] <- sum(rexp(length(expmean), (1/expmean)))
}
return(Claims)
}

claims <- simulate_claims(10000, 0, 1)
hist(claims, breaks=20)

```



```

# Calculate mean
mean(claims)

```

```
## [1] 168.405
```

```

# Calculate std
sd(claims)

```

```
## [1] 58.49466
```

To find a confidence interval one simple approach is to calculate the standard normal confidence interval. This can be done since the distribution of means approach a normal distribution due to the central limit theorem.

```
# Standard normal confidence interval
CI<- c(mean(claims) - qnorm(0.975, 0, 1)*(sd(claims)/sqrt(length(claims))),
      mean(claims) + qnorm(0.975, 0, 1)*(sd(claims)/sqrt(length(claims))))
CI
```

```
## [1] 167.2585 169.5515
```

To be 97.5% certain to be able to be sure that the company is able to cover all claims, 97.5% of the simulated costs must be possible to pay. I.e, the 97.5 percentile of the simulated claims must be calculated

```
quantile(claims, c(0.975))
```

```
##      97.5%
## 295.4533
```

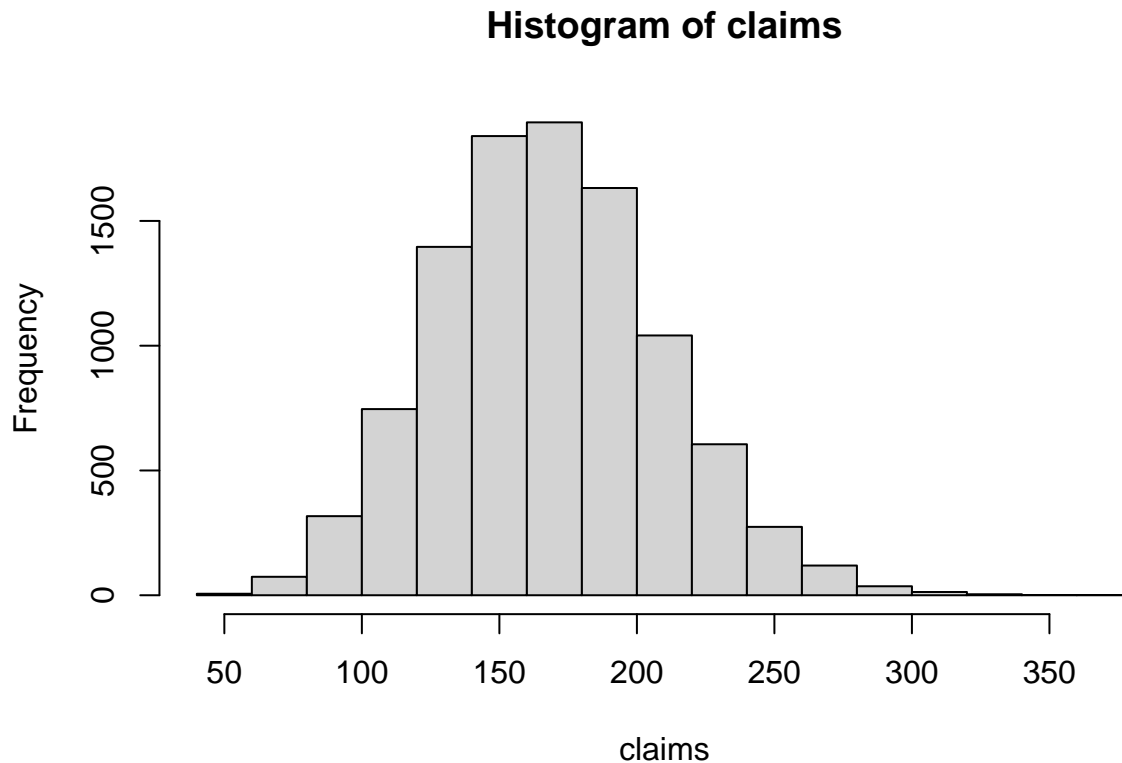
And we see that 97.5% of the simulated costs are less than approx 300 million kroners during 2020.

e.) Calculate claims using Rao-Blackwellization

R code for estimating $E[X] = E[Y]$ using equation 6 and 7 in assignment 2.

```
# function for simulatin Nsim claims, from a to b
# Rao-Blackwellization approach
simulate_claims <- function(Nsim, a, b) {
  Claims <- vector(length=Nsim)
  for(i in 1:Nsim) {
    # Calculate mean parameter of storm based on time of storm and sum them
    Claims[i] <- sum(claim(simtNHPP(a=a, b=b,
                                lambdamax=max(lambdastorm(seq(a, b, 0.01))),
                                lambdafunc=lambdastorm)))
  }
  return(Claims)
}

claims <- simulate_claims(10000, 0, 1)
hist(claims, breaks=20)
```



```
# Calculate mean
mean(claims)
```

```
## [1] 168.0586
```

```
# Calculate std
sd(claims)
```

```
## [1] 41.97776
```

We observe that the standard deviation is smaller and by inspection see that the histogram is not as wide as the previous estimate. Now, let's calculate the confidence interval for the mean of claims.

```
# Standard normal confidence interval
CI<- c(mean(claims) - qnorm(0.975, 0, 1)*(sd(claims)/sqrt(length(claims))),
       mean(claims) + qnorm(0.975, 0, 1)*(sd(claims)/sqrt(length(claims))))
CI
```

```
## [1] 167.2359 168.8814
```

f.) Propose and implement improved estimator of $Var(X)$

Problem 3

Load data and run simple regression

```
load("prob23.dat")
lm.obj <- lm(y ~ x1+x2+x3+x4+x5,data=df)
Rsquared <- summary(lm.obj)$r.squared
```

```
Rsquared
```

```
## [1] 0.8943925
```

a.) Calculate B bootstrap samples for R^2 and plot histogram

Code below is borrowed both from `bootstra_examples.R` file from the lectures as well as [this](#) article by statmethods.net (Robert I. Kabacoff, Ph.D. 2017)

```
# Bootstrap 95% CI for R-Squared
library(boot)

# function to obtain R-Squared from the data
rsq <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}

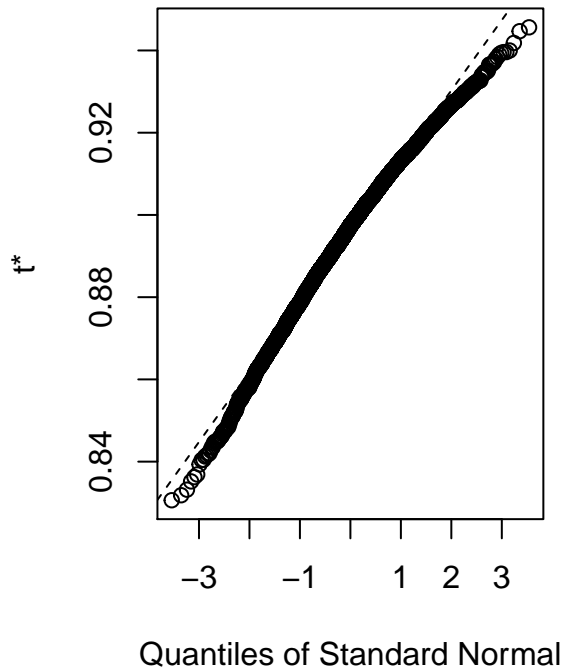
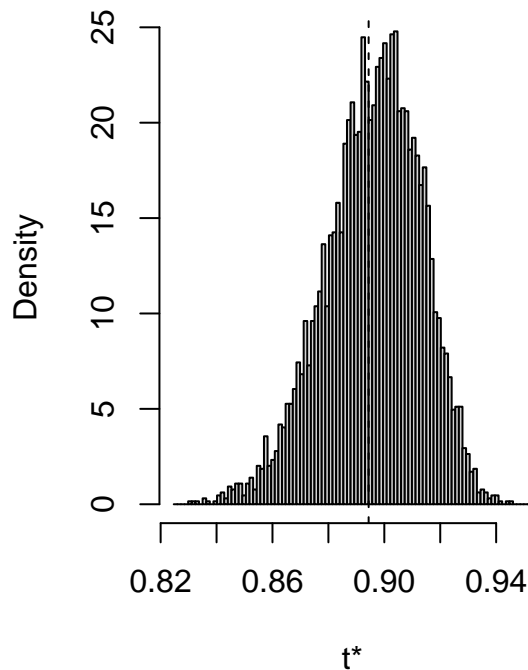
# bootstrapping with 5000 replications
results <- boot(data=df, statistic=rsq, R=5000, formula=y~x1+x2+x3+x4+x5)

# view results
results

##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = df, statistic = rsq, R = 5000, formula = y ~ x1 +
##       x2 + x3 + x4 + x5)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.8943925 0.001691808 0.01710067

plot(results)
```

Histogram of t



```
# get 95% confidence interval
boot.ci(results, type=c("bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = c("bca"))
##
## Intervals :
## Level      BCa
## 95%      ( 0.8492,  0.9211 )
## Calculations and Intervals on Original Scale
```

We see from the histogram, the quantile plot and the confidence intervals that the bootstrap samples of R^2 is fairly normal.

b.) Find bootstrap estimate for bias of R^2

According to the ordinary nonparametric bootstrap using the boot library, $bias(R^2) \approx 0.00148$ and $std.err(R)^2 \approx 0.0167$

b.) Calculate 99% confidence interval

To calculate the 99% confidence interval for R^2 using standard normal interval and percentile interval, we use the boot library.

```
boot.ci(results, conf=0.99, type=c("norm", "perc"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
```

```
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, conf = 0.99, type = c("norm", "perc"))
##
## Intervals :
## Level      Normal      Percentile
## 99%    ( 0.8487, 0.9367 )  ( 0.8459, 0.9327 )
## Calculations and Intervals on Original Scale
```

We see that the confidence interval is not quite equal, the normal confidence interval is a tiny bit wider than the percentile interval. This is due to the data not being quite normal. We can see this from the quantile plot above where we see that R^2 values far from the mean of the data is not as often as expected from a standard normal distribution.

Problem 4

According to (Burkardt 2014, 20–24), the CDF of a general truncated normal distribution bounded on the interval $[a, b]$ is

$$\psi(\bar{\mu}, \bar{\sigma}, a, b; x) = \begin{cases} 0 & x \leq a \\ \frac{\phi(\bar{\mu}, \bar{\sigma}^2; x) - \phi(\bar{\mu}, \bar{\sigma}^2; a)}{\phi(\bar{\mu}, \bar{\sigma}^2; b) - \phi(\bar{\mu}, \bar{\sigma}^2; a)} & a < x < b \\ 0 & b \leq x \end{cases}$$

Where ϕ being the CDF of a normal distribution.

By replacing some of the notation to reflect the notation used in the assignment, namely:

- replacing x with r
- using F for notating the CDF of a normal ϕ
- the fact that $b = +\infty$ and thus the $\phi(b) = 1$
- replacing the left bound a with l

We get

$$G_R(l, \mu, \sigma; r) = \begin{cases} 0 & r < l \\ \frac{F(r) - F(l)}{1 - F(l)} & l \leq r \end{cases}$$

To find the inverse $G_R^{-1}(u)$ we replace G with U where U represent the *Uniform*(0, 1) distribution and solve for r

$$U = \frac{F(r) - F(l)}{1 - F(l)} \implies F(r) = F(l) + U(1 - F(l)) \implies r = F^{-1}(F(l) + U(1 - F(l)))$$

Which is the inverse cumulative distribution shown in the assignment.

b.) Inverse transform sampling of left-truncated normal distribution

The code below evaluates $G_R^{-1}(u)$ by using the fact that the inverse of a CDF of a normal distribution is the quantile function of a normal distribution and utilizes the definition of the pdf of a left-truncated normal distribution defined by (Burkardt 2014, 20)

```
inverse_truncated <- function(samples, l, mu, sigma) {
  # Sample uniform numbers
  u <- runif(samples, min = 0, max = 1)
  # Cumulative normal distribution
```

```

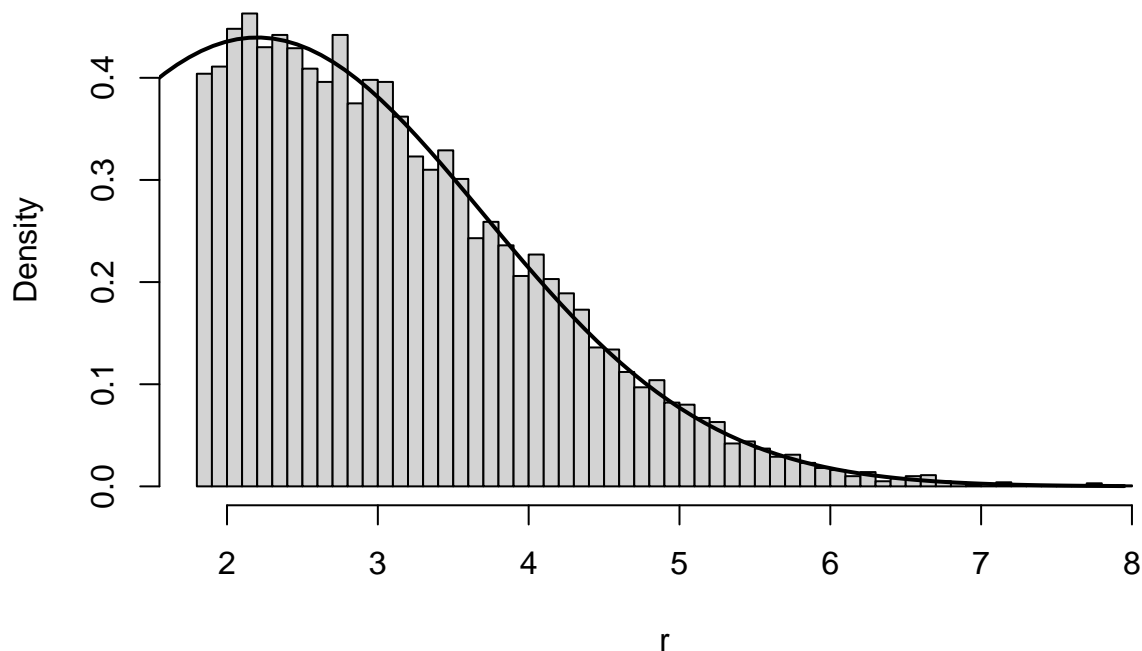
f <- function(x) pnorm(x, mu, sigma)
# Inverse cumulative normal distribution (quantile)
f.inv <- function(x) qnorm(x, mu, sigma)
# Inverse transform sampling
r <- f.inv(f(l) + u*(1 - f(l)))
return(r)
}

# Density of truncated normal distribution
# Based on pdf by (Burkardt 2014, 20)
pdf_truncated <- function(x, l, mu, sigma) {
  return(dnorm(x, mu, sigma) / (1 - pnorm(l, mu, sigma)))
}

l <- 1.8
mu <- 2.2
sigma <- 1.5
x <- inverse_truncated(10000, l, mu, sigma)
hist(x, probability = TRUE, xlab="r", breaks=50)
curve(pdf_truncated(x, l, mu, sigma), 0, max(x), lwd=2, xlab = "", ylab = "", add = T)

```

Histogram of x



b.) Calculate the theoretical mean and standard deviation

The pdf as defined in the pdf_truncated function was not possible to integrate. The function is defined again below and integrated.

```

# r*Fr(r) R integral
expected <- integrate( function(r, l=1.8, mu=2.2, sigma=1.5)
{
  r*(exp(-(r-mu)^2 / (2*sigma^2))/(sqrt(2*pi*sigma^2)*
    (1-pnorm(l, mu, sigma))))*(r>= l)
}, -Inf, Inf)$value

# r^2*Fr(r) R integral
expected_r2 <- integrate( function(r, l=1.8, mu=2.2, sigma=1.5)
{
  (r^2)*(exp(-(r-mu)^2 / (2*sigma^2))/(sqrt(2*pi*sigma^2)*
    (1-pnorm(l, mu, sigma))))*(r>= l)
}, -Inf, Inf)$value

print(paste("Mean: ", expected))

## [1] "Mean: 3.1543441055124"

print(paste("SD: ", sqrt(expected_r2 - expected^2)))

## [1] "SD: 0.978516698425868"

print(paste("Simulated mean: ", mean(x)))

## [1] "Simulated mean: 3.16787757958448"

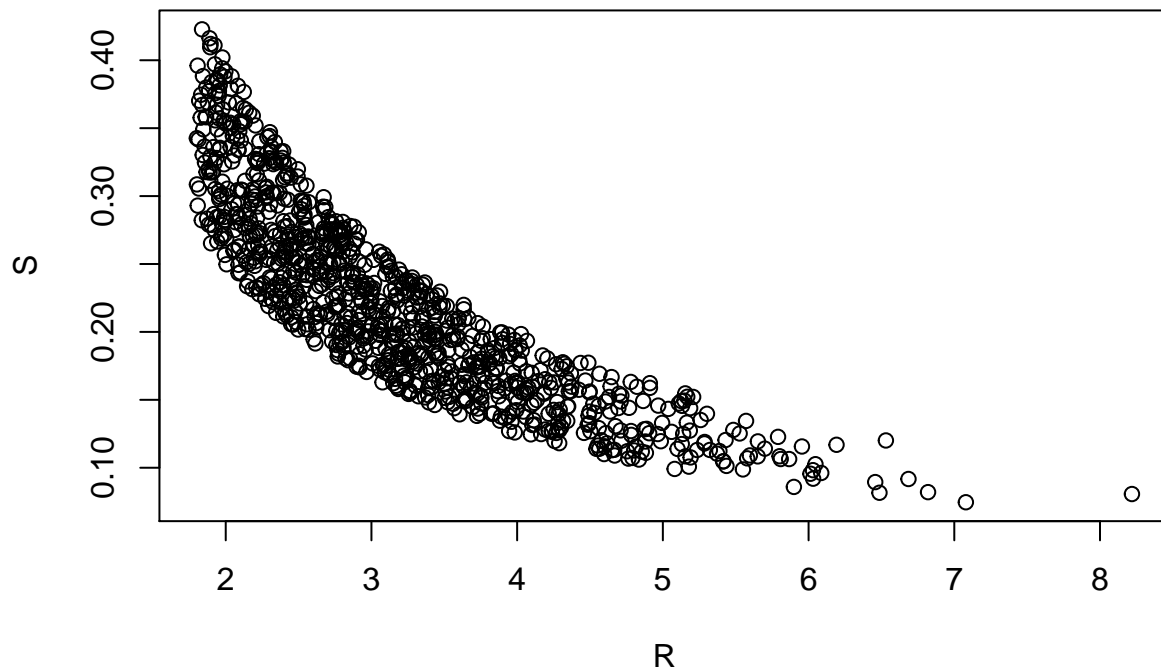
print(paste("Simulated sd: ", sd(x)))

## [1] "Simulated sd: 0.985429905360498"

c.) Simulate R values and S values

# No of samples
Samp <- 1000
# Simulate R from truncated normal
R <- inverse_truncated(Samp, l=1.8, mu=2.2, sigma=1.5)
# Simulate S conditionally on R
S <- runif(Samp, 0.5/R, 0.8/R)
# N x 2 matrix
m <- cbind(R, S)
plot(m)

```

To obtain the mean vector or **expected value vector** $E[\mathbf{X}]$ where

$$\mathbf{X}_i = \begin{bmatrix} R_i \\ S_i \end{bmatrix} \quad \text{for } i = 1, 2, \dots, N$$

is a vector of R values and S values for N simulated vectors. We can estimate the mean vector by estimating the individual means for each random variable in the vectors and we get

$$E[\mathbf{X}] = \begin{bmatrix} E[R] \\ E[S] \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum_{i=1}^N R_i \\ \frac{1}{N} \sum_{i=1}^N S_i \end{bmatrix}$$

In r:

```
meanvec <- colMeans(m)
meanvec
```

```
##           R           S
## 3.1773019 0.2220725
```

To calculate the covariance matrix $K_{RS} = \text{cov}(R, S)$ and the correlation matrix (which is the covariance matrix of standardized random variables). We use the r functions `cov` and `cor`.

```
cov(m)
```

```
##           R           S
## R  1.00385266 -0.060474688
## S -0.06047469  0.004990053
```

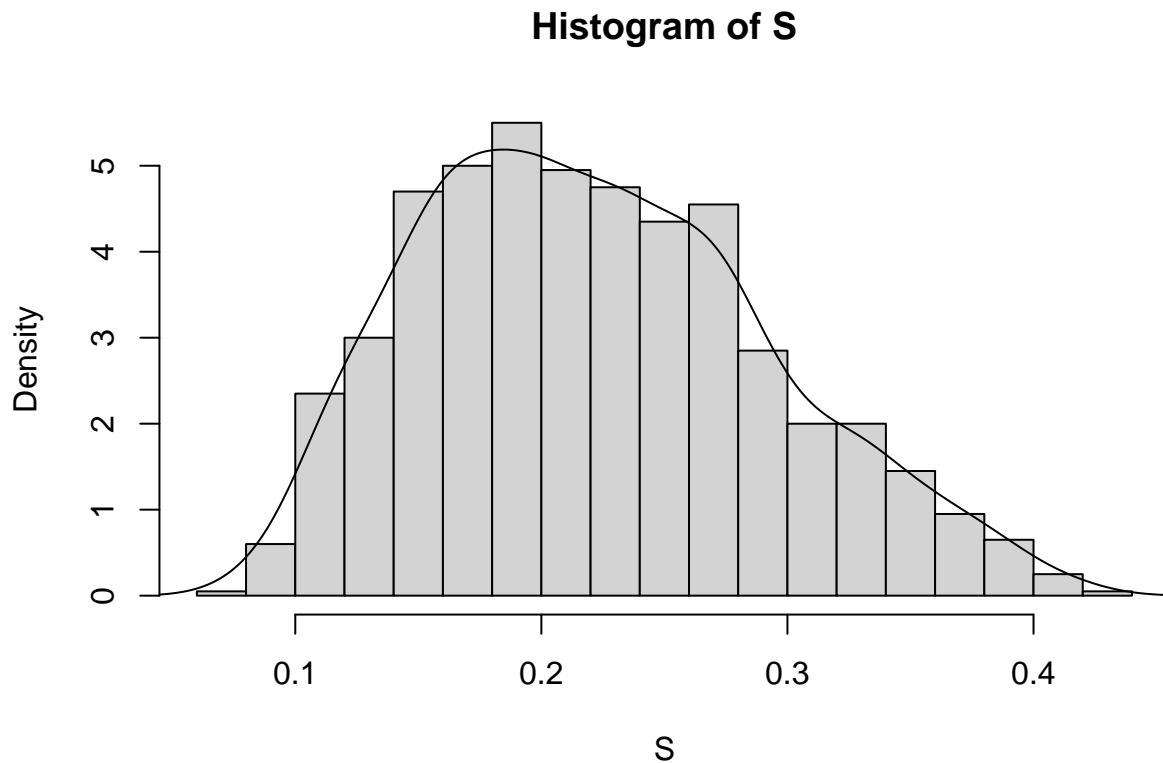
```
cor(m)
```

```
##           R           S
## R  1.0000000 -0.8544489
## S -0.8544489  1.0000000
```

d.) Histogram and distribution of S

We plot the values of S as an histogram and use the `R` function for estimating the density to plot the estimated density against the histogram.

```
hist(S, breaks = 20, probability = TRUE)
lines(density(S))
```



And we visually can see that the density of S is **not** uniform.

Let's calculate the marginal density $f_S(s)$. The limits of integration utilizes the support of the pdf $\frac{0.5}{R} \leq S$ to determine values of R when integrating.

R code below calculates the integral

$$f_S(s) = \int_{\frac{0.5}{s}}^{\frac{0.8}{s}} f_{RS}(r, s) dr$$

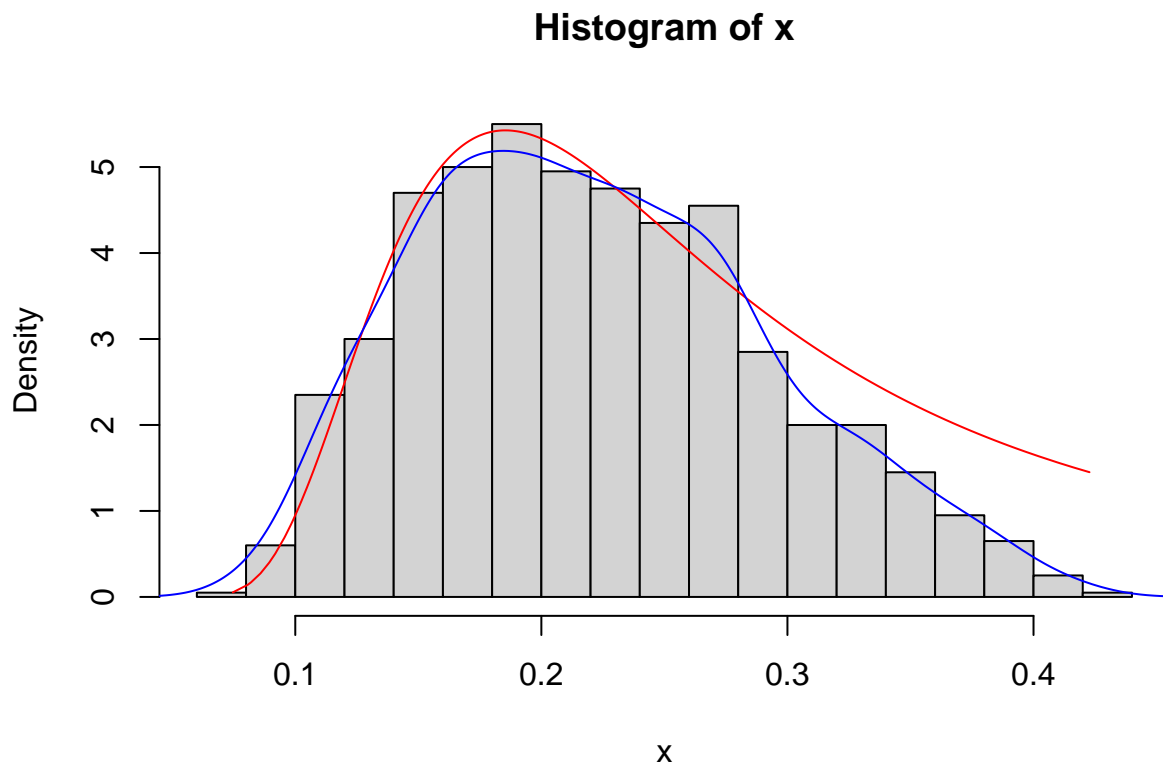
```
# Marginal distribution of S
marginal_s <- function(s, l, mu, sigma) {
  integrate (
    function(r) {
      ((exp(-(r-mu)^2 / (2*sigma^2))*10*r) /
       (sqrt(2*pi*sigma^2)*(1-pnorm(l, mu, sigma))*3))*(s >= 0.5/r)*(s <= 0.8/r)
    },
    lower = 0.5/s, upper = 0.8/s
  )
}
```

```

    }, 0.5/s, 0.8/s
  )$value
}

x <- S
x <- sort(x, decreasing = FALSE)
marginal <- numeric(length(x))
i <- 0
for (value in x) {
  i <- i+1
  marginal[i] <- marginal_s(value, 1.8, 2.2, 1.5)
}
hist(x, breaks = 20, probability = TRUE)
lines(x, marginal, col="red")
lines(density(S), col="blue")

```



And we see that the marginal density plotted in red fits the simulated values of S quite well.

e.) Generate R and S using antithetic variables.

We introduce an antithetic uniform variable when sampling R via the inverse transform method. u_1 and u_2 are negatively correlated and variance is reduced.

```

inverse_truncated_anti <- function(samples, l, mu, sigma) {
  # Sample uniform numbers
  u1 <- runif(samples/2, min = 0, max = 1)
  u2 <- 1 - u1
  u <- c(u1, u2)
}

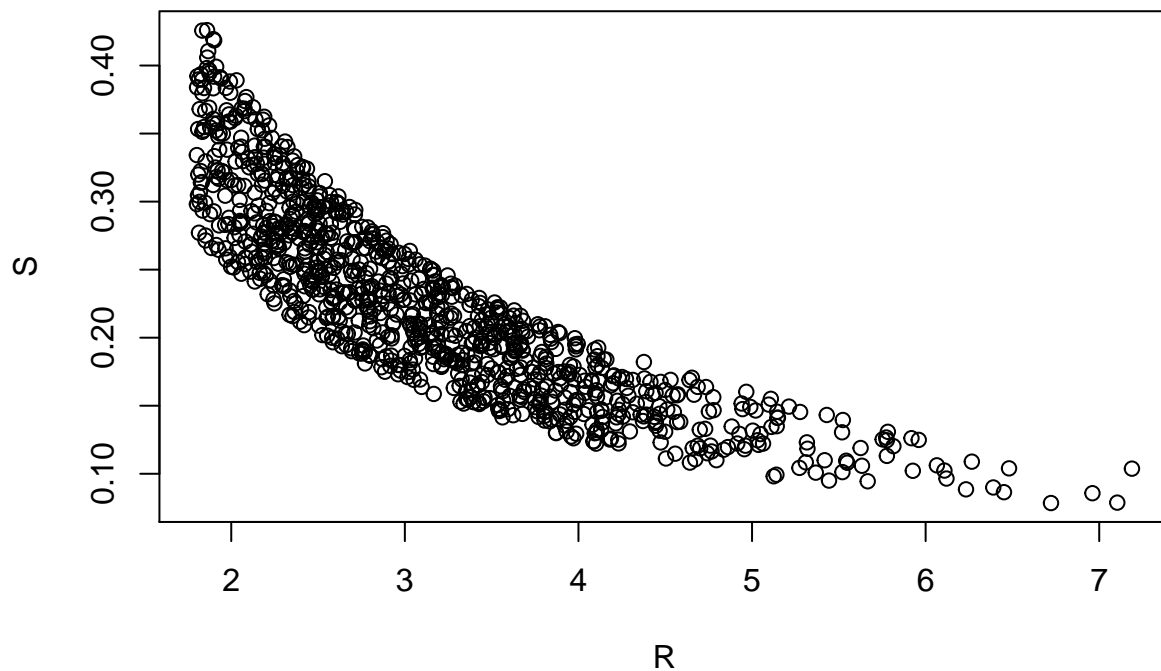
```

```

# Cumulative normal distribution
f <- function(x) pnorm(x, mu, sigma)
# Inverse cumulative normal distribution (quantile)
f.inv <- function(x) qnorm(x, mu, sigma)
# Inverse transform sampling
r <- f.inv(f(l) + u*(1 - f(l)))
return(r)
}

# No of samples
Samp <- 1000
# Simulate R from truncated normal
R <- inverse_truncated_anti(Samp, l=1.8, mu=2.2, sigma=1.5)
# Simulate S conditionally on R
S <- runif(Samp, 0.5/R, 0.8/R)
# N x 2 matrix
m <- cbind(R, S)
plot(m)

```



Still, we don't observe an significant reduction in the variance of the (R, S) values.

e.) Calculate hospital beds needed

Import SEIR-model

```

source("seir_model.R")
f <- seir_sim()

```

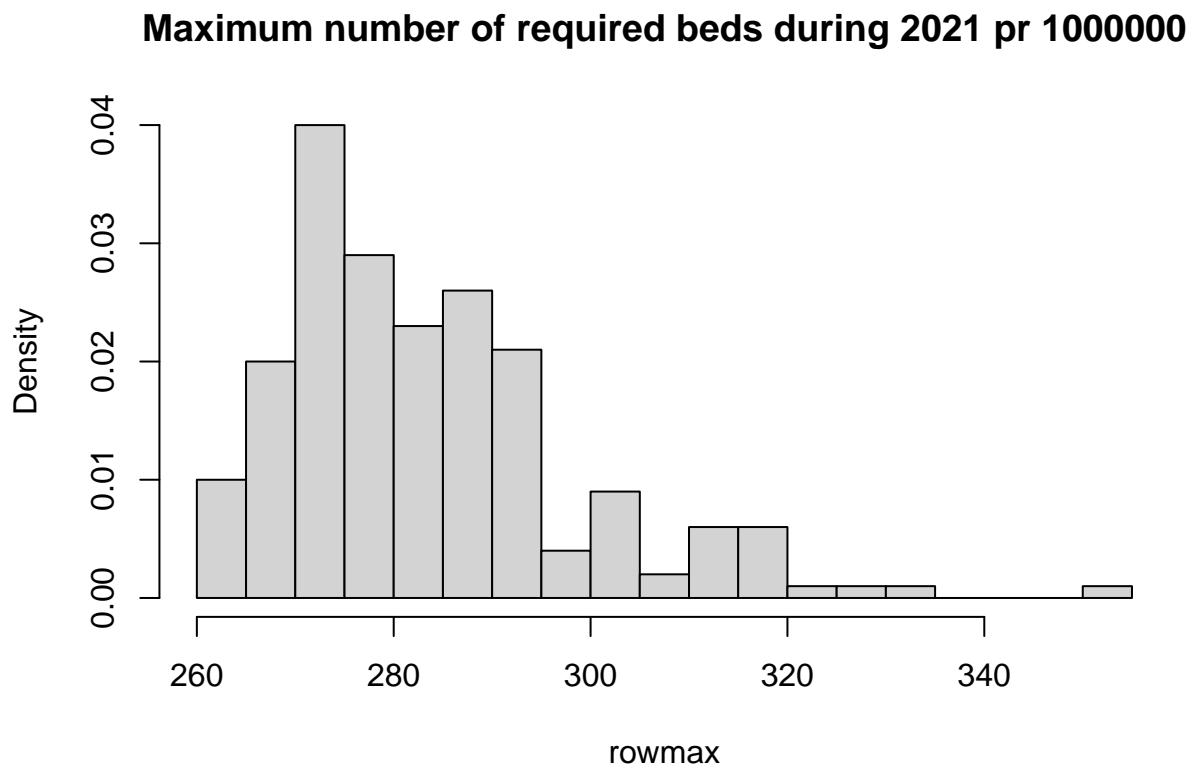
```

source("seir_model.R") # assumed already run
nsim <- 200 # number of simulation replica
days <- 365 # 366 days in 2021.
population <- 10^6

t <- which(f$dates<="2021-12-31" & f$dates>="2021-01-01")
m <- matrix(nsim, days)
for(i in 1:nsim){
  ff <- seir_sim(R0max = R[i],
                 soc_dist_Rfac = S[i])
  m[i,] <- population*(ff$ode[t,"HH"] + ff$ode[t,"HC"] + ff$ode[t,"CC"])
}

rowmax <- apply(m, 1, function(x) max(x))
hist(rowmax, probability = TRUE, breaks = 30,
     main="Maximum number of required beds during 2021 pr 1000000")

```



f.) Uncertainty in daily required hospital beds

Plot cumulative sums of daily needed beds over 2021

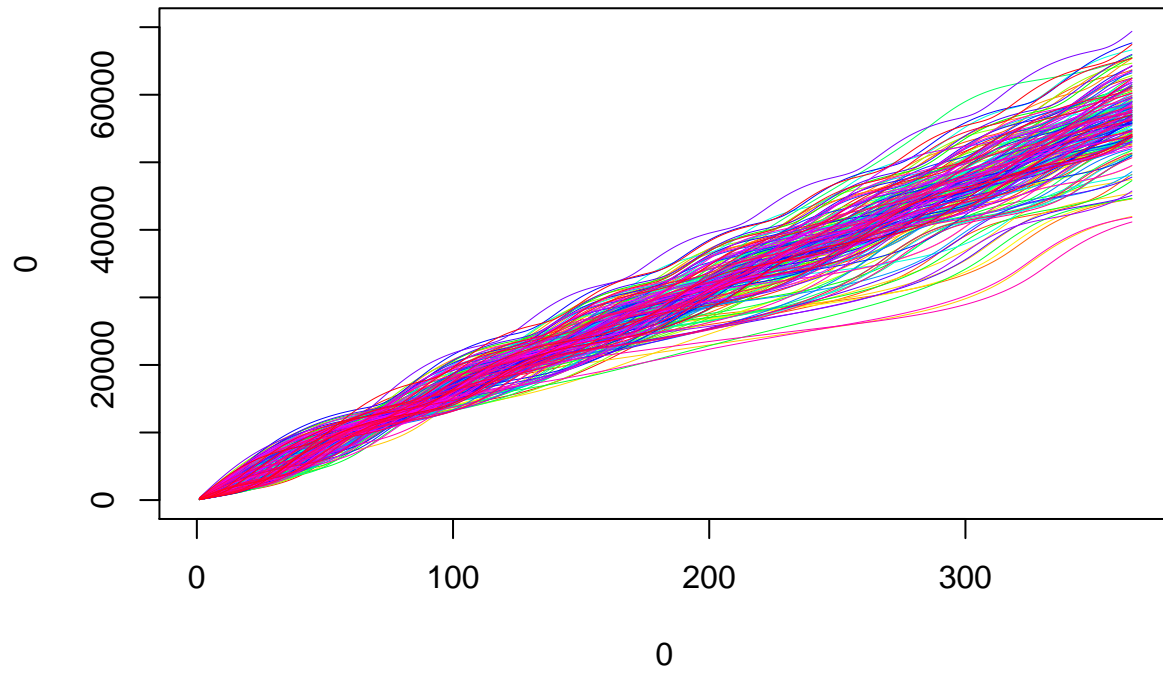
```

cummat <- matrix(nsim, 365)
cl <- rainbow(200)

plot(0,0,xlim = c(0,365),ylim = c(0,70000), type = "n")
for (i in 1:nsim) {
  lines(cumsum(m[i,]), col=cl[i], type="l", lw=0.3)
}

```

```
}
```



We see that for our 200 simulations, the cumulative sum of daily beds needed at the end of the year is in the range of 30000 - 70000.

Bibliography

Burkardt, John. 2014. “The Truncated Normal Distribution.” *Department of Scientific Computing Website, Florida State University*, 1–35.

Robert I. Kabacoff, Ph.D. 2017. “Bootstrapping.” <https://www.statmethods.net/advstats/bootstrapping.html>.

Wikipedia contributors. 2020. “Monte Carlo Integration — Wikipedia, the Free Encyclopedia.” https://en.wikipedia.org/w/index.php?title=Monte_Carlo_integration&oldid=976847254.