Assignment 2

Bjørn Christian Weinbach

13th October, 2020

Clear R environment

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

Problem 1

Consider the integral

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

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

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

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

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

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

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

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

$$\bar{\boldsymbol{x}}_{\boldsymbol{1}}, \ldots, \bar{\boldsymbol{x}}_{\boldsymbol{n}} \in \Omega,$$

I can be approximated by

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

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 $V = \int_{-1}^{1} \int_{-1}^{1} dxdy = 4$ which gives the following crude way to estimate I

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

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 success or failure, and is therefore has aa potential ' bernoulli distribtion. To calculate $P(X^2 + Y^2 \leq 1)$ we need to calculate the

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

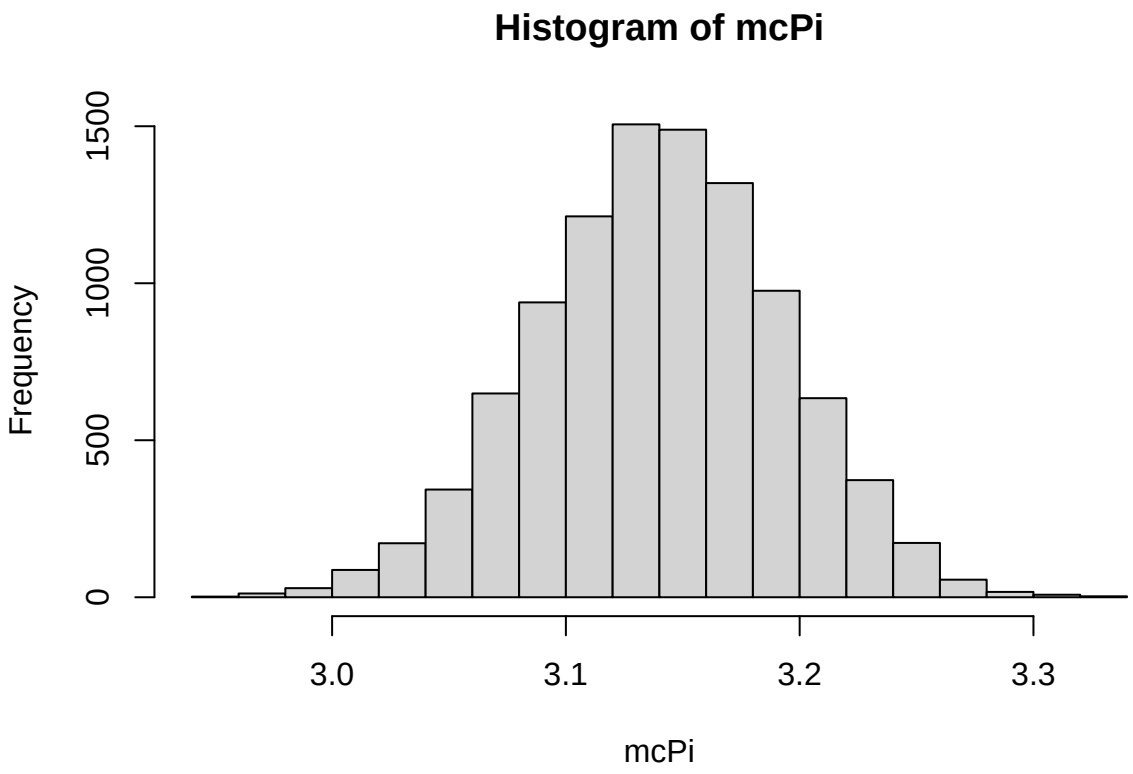
```
indicator1 <- function (x, y) {          # Indicator function for unit circle
  return((x^2+y^2 <= 1))
}

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.
##   2.948   3.108   3.144   3.142   3.176   3.340
```

```
hist(mcPi)                              # Histogram
```




```
mean(mcPi)
```

```
## [1] 3.142004
```

```
var(mcPi)
```

```
## [1] 0.002755476
```

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

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

```
mean(mcPi < 3.15) - mean(mcPi <= 3.14)
```

```
## [1] 0.0596
```

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] -0.008782487
```

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.
##   2.944   3.108   3.144   3.142   3.176   3.312
```

```
mean(mcPi)
```

```
## [1] 3.141921
```

```
var(mcPi)
```

```
## [1] 0.002705807
```

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.
##   2.936   3.108   3.144   3.142   3.176   3.376
```

```
mean(mcPi)
```

```
## [1] 3.141644
```

```r
var(mcPi)
```

```
## [1] 0.00271715
```

h.) Use important sampling

```r
f <- function (x, y, sigma) {
  return((1/(2*pi*sigma^2))*exp(-(x^2)/(2*sigma^2))*exp(-(y^2)/(2*sigma^2)))
}

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)
}
```

```r
mcPi <- mcEstimatePi(10000, 0.3)
summary(mcPi)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.6143  0.6201  0.6214  0.6214  0.6227  0.6288
```

Problem 2

Define $\lambda(t)$ in R

```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 proccess 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

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

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

Which we have confirmed by the simulation above which has a simulated mean of approximately 16.3

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

```r
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

```r
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 distrubuted with $\lambda = 39.72776$. 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 $\lambda(t), \quad t \geq 0$

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

```r
# 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
  Nsim <- 3*expectednumber  # Simulate more than the expected number to be certain to exceed stoptime
  timesbetween <- rexp(Nsim,lambdamax) # Simulate interarrival times
  timesto <- a+cumsum(timesbetween)   # Calculate arrival times starting at a
  timesto <- timesto[timesto<b] # Dischard the times larger than b
  Nevents <- length(timesto) # Count the number of events
  # 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] 16.386
```

```r
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] 21.623
```

```r
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] 33.919
```

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

```
## [1] 35.55199
```

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

```
## [1] 5.962549
```

d.) Simulate claim size

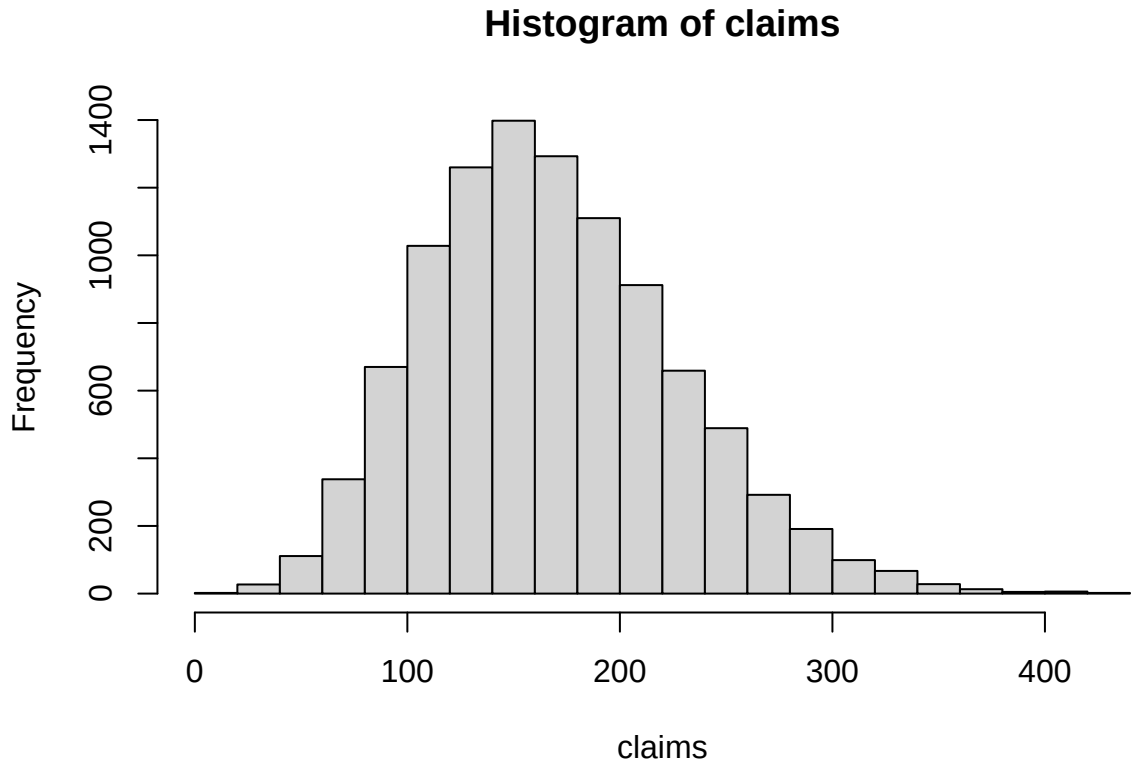
To calculate the claim size for a given year, simulate a poisson process and calculate mean parameter for all storms, then draw claim size for exponential distribution with mean parameter $c(t_i)$ where t_i is time of a given storm simulated from the NHPP.

```r
# 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] 168.0899
```

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

```
## [1] 58.90842
```

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] 166.9353 169.2445
```

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 precentile of the simulated claims must be calculated

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

```
##    97.5%
## 294.9318
```

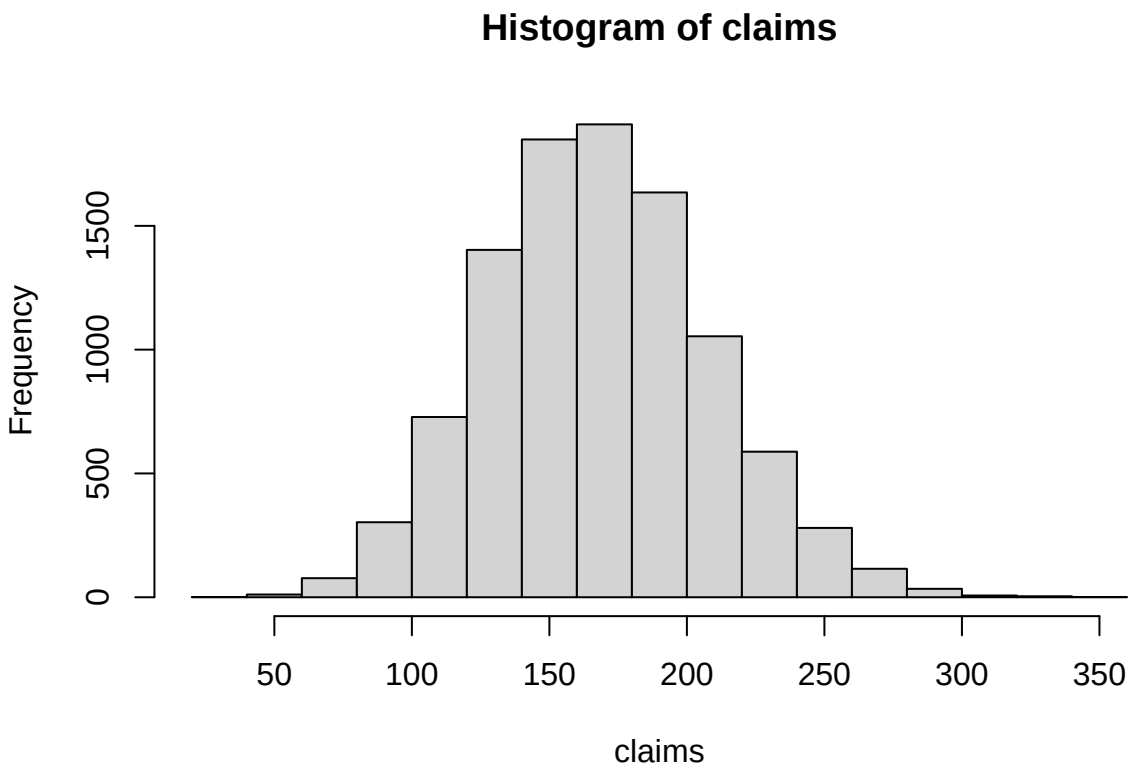
And we see that 97.5% of the simulated costs are less than approx 300 million kroners during 2020.

e.) Calculate claims using Rao-Blackwelliztion

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

```
# function for simulatin Nsim claims, from a to b
# Rao-Blackwelliztion 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] 167.9188
```

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

```
## [1] 41.62502
```

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] 167.1029 168.7346
```

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

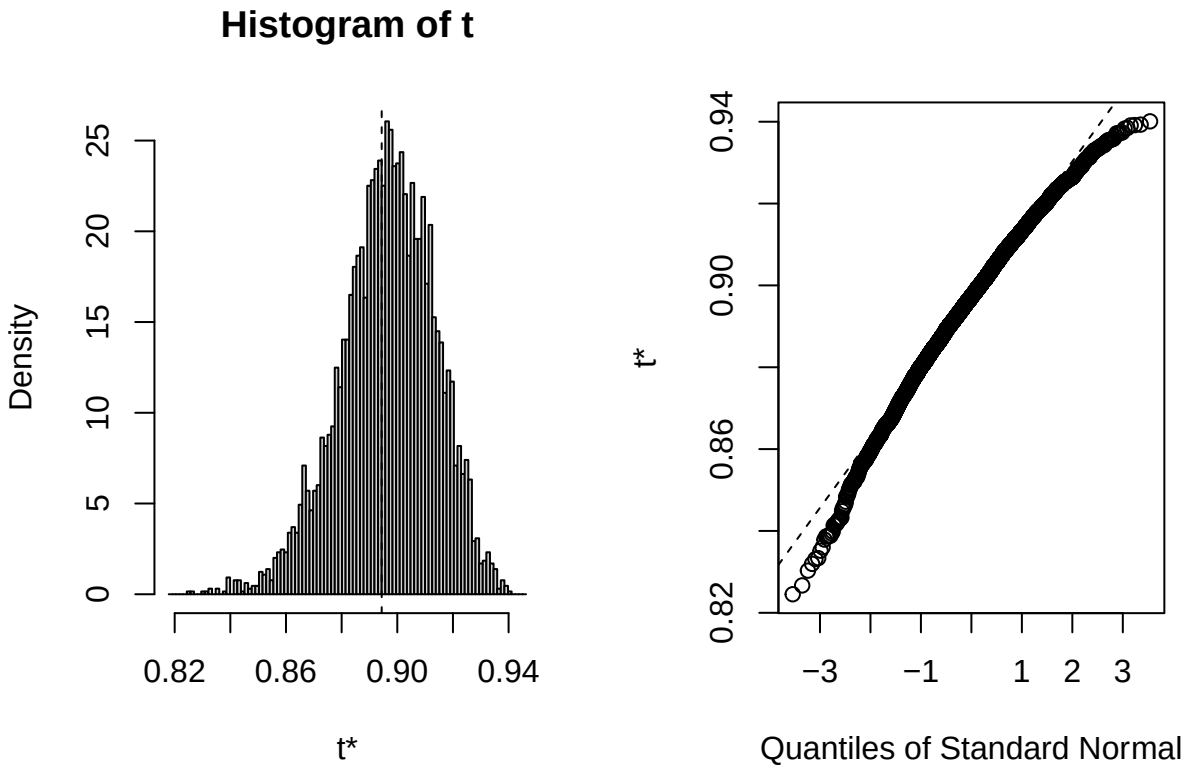
```
# Bootstrap 95% CI for R-Squared
library(boot)

# 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 0.002052176  0.01692721
```

```
plot(results)
```




```
# 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%   ( 0.8491,  0.9209 )
## 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 $\text{std.err}(R)^2 \approx 0.0167$

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,  0.9359 )   ( 0.8433,  0.9341 )
## 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 thatn the percentile interval. This is due to the data not beeing quite normal. We can see this form the quantile plot above where we see that R^2 values fart from the mean of the data is not as often as expected from a standard normal distributition.

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

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

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

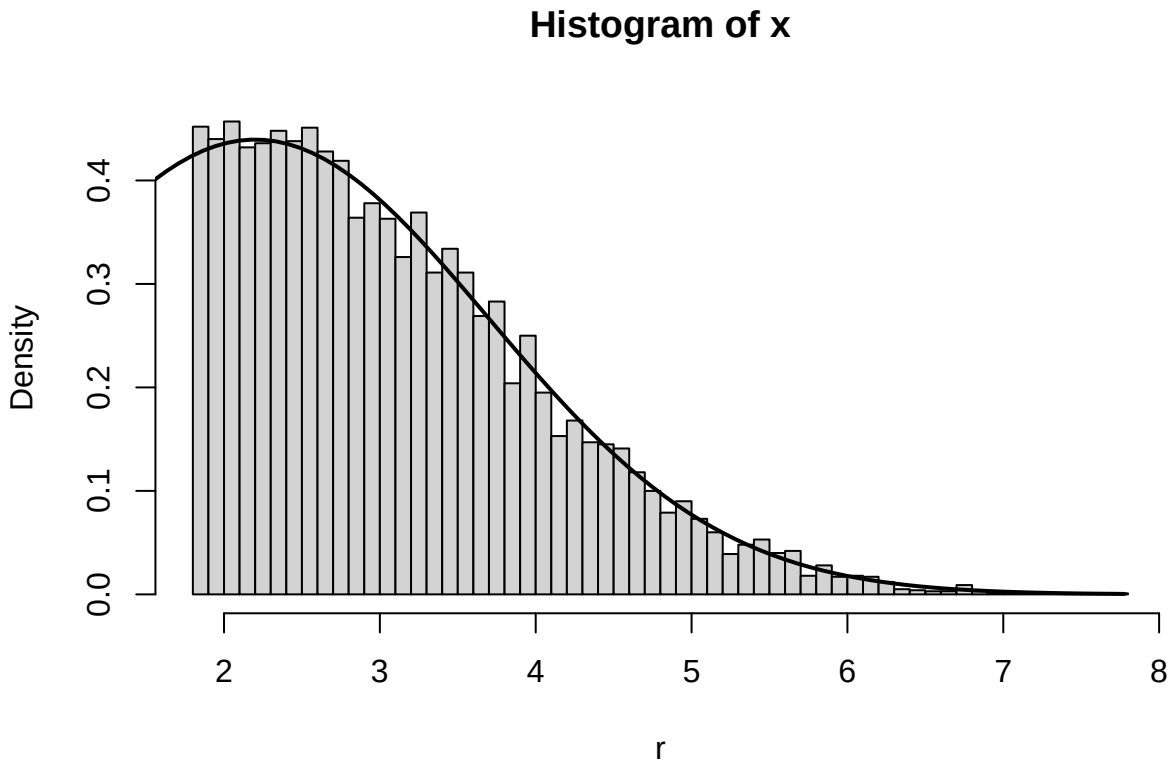
The code below evaluates $G_R^{-1}(u)$ by using the fact that the inverse of a CDF of a normal distribution is the quantile function of a normal distribution and utilizes the definition of the pdf of a left-truncated normal distribution defined by (Burkardt 2014, 20)

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

```r
  f <- function(x) pnorm(x, mu, sigma)
  # Inverse cumulative normal distribtution (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)
```



Bibliography

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