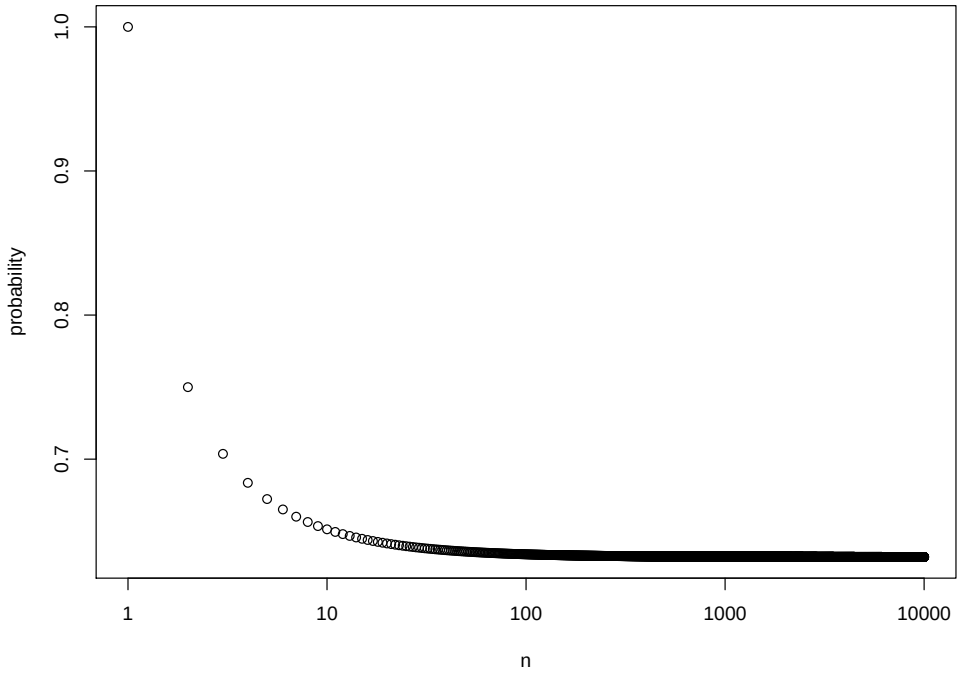
# Solution to Series 5

**1.** **a)** We sample with replacement. The probability that given observation is not picked in every sampling iteration is $1 - 1/n$. Each draw is independent. Hence, we apply the product rule. The solution is $(1 - 1/n)^n$ since we generate the samples of size $n$.

**b)** The solution is $1 - (1 - 1/n)^n = 1 - (1 - 1/100)^{100} = 1 - (99/100)^{100} = 63.4\%$.

**c)**
```
> foo <- function(n) {1 - (1 - 1/n)^n}
> n <- 1:10000
> plot(n, foo(n), log = "x", ylab = "probability")
```




**d)** We expect that about $2/3$ of the original observations occur in the bootstrap sample, i.e. $\lim_{n\to\infty} 1 - (1 - 1/n)^n = 1 - 1/e \approx 63.21\%$ and this limit is visible in the plot.

**2.** For any random variable $Z$, let $q_Z(\alpha)$ denote the $\alpha$-quantile of $Z$. We have:

$$
\begin{aligned}
1 - \alpha &= P\left( q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) < \frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)} < q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \right) \\
&\approx P\left( q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) < \frac{\hat{\theta} - \theta}{\widehat{sd}(\hat{\theta})} < q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \right)
\end{aligned}
$$

The second equality follows from the bootstrap consistency: if the distribution of $\hat{\theta}^* - \hat{\theta}$ can be approximated by the distribution of $\hat{\theta} - \theta$ and both random variables are divided by their standard deviations, the

distribution of $\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}$ can be approximated by the distribution of $\frac{\hat{\theta} - \theta}{\widehat{sd}(\hat{\theta})}$. Hence,

$$\begin{aligned}
1 - \alpha &\approx P\left( q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) < \frac{\hat{\theta} - \theta}{\widehat{sd}(\hat{\theta})} < q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \right) \\
&= P\left( q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) \cdot \widehat{sd}(\hat{\theta}) < \hat{\theta} - \theta < q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \cdot \widehat{sd}(\hat{\theta}) \right) \\
&= P\left( -q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) \cdot \widehat{sd}(\hat{\theta}) > \theta - \hat{\theta} > -q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \cdot \widehat{sd}(\hat{\theta}) \right) \\
&= P\left( -q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \cdot \widehat{sd}(\hat{\theta}) < \theta - \hat{\theta} < -q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) \cdot \widehat{sd}(\hat{\theta}) \right) \\
&= P\left( \hat{\theta} - q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(1 - \alpha/2) \cdot \widehat{sd}(\hat{\theta}) < \theta < \hat{\theta} - q_{\frac{\hat{\theta}^* - \hat{\theta}}{\widehat{sd}(\hat{\theta}^*)}}(\alpha/2) \cdot \widehat{sd}(\hat{\theta}) \right).
\end{aligned}$$

**3.** **a)** Approximate the true value

```
> set.seed(3)
> (true.par <- mean(rgamma(100000000, shape = 2, rate = 2), trim = 0.1))
[1] 0.9103737
```

**b)**
```
> set.seed(1)
> sample40 <- rgamma(n = 40, shape = 2, rate = 2)
> mean(sample40, trim = 0.1)
[1] 0.8824166
```

**c)**
```
> require("boot")
> tm <- function(x, ind) {mean(x[ind], trim = 0.1)}
> tm_var <- function(x, ind) {
   # trimmed mean
   t1 <- tm(x, ind)
   # bootstrap variance of the trimmed mean (required for the bootstrap T CI)
   t2 <- var(boot(data = x[ind], statistic = tm, R = 50)$t)
   return(c(t1, t2))
 }
> res.boot <- boot(data = sample40, statistic = tm_var, R = 10000,
                   sim = "ordinary")
> boot.ci(res.boot, conf = 0.95, type = c("basic", "norm", "perc", "stud"),
         var.t0 = var(res.boot$t[, 1]))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = res.boot, conf = 0.95, type = c("basic", "norm",
    "perc", "stud"), var.t0 = var(res.boot$t[, 1]))

Intervals :
Level      Normal              Basic
95%   ( 0.7022,  1.0569 )   ( 0.6945,  1.0492 )

Level    Studentized          Percentile
95%   ( 0.6995,  1.0792 )   ( 0.7156,  1.0704 )
Calculations and Intervals on Original Scale
```

**d)** We first define two functions before running the simulation.

```
> ##' Checks if a confidence interval contains the true parameter (separately
> ##' for the lower and the upper end)
> ##'
> ##' @param ci: Output of the function boot.ci which contains CIs
> ##' @param ty: Type of confidence interval
```

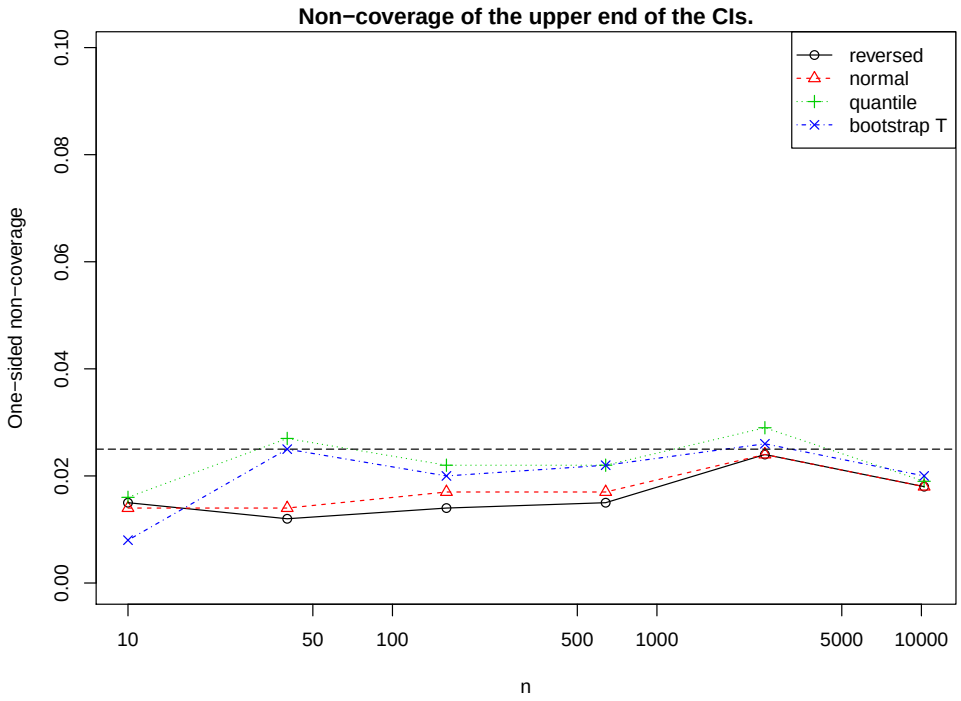
```
> ##' @param true.par: True parameter
> ##'
> ##' @return Vector with two elements where first one corresponds to the lower
> ##'         end and the second to the upper end of the confidence interval.
> ##'         If the CI is [CI_l, CI_u], the first element is 1 if theta < CI_l
> ##'         and 0 otherwise. The second element is 1 if theta > CI_u and 0
> ##'         otherwise.
> check_ci <- function(ci, ty, true.par) {
   # Get confidence interval of type ty from object ci
   lower.upper <- switch (ty,
     "norm" = ci[["normal"]][2:3],
     "perc" = ci[["percent"]][4:5],
     "basic" = ci[["basic"]][4:5],
     "stud" = ci[["student"]][4:5]
   )

   res <- if (true.par < lower.upper[1]) {
       c(1, 0)
     } else if (true.par > lower.upper[2]) {
       c(0, 1)
     } else {
       c(0, 0)
     }
   names(res) <- c("lower", "upper")

   return(res)
 }
> ##' Runs one simulation run, i.e. creates new data set, calculates bootstrap
> ##' CIs, and checks if true parameter is contained.
> ##'
> ##' @param n: Size of sample
> ##' @param true.par: True parameter
> ##' @param R: Number of bootstrap replicates
> ##' @param type: Type of bootstrap CIs, see function boot.ci
> ##'
> ##' @return A vector containing the result of the function check_ci for each
> ##'         of the confidence intervals
> do_sim <- function(n, true.par, R = 1000,
                     type = c("basic", "norm", "perc", "stud")) {
   # Generate the data
   x <- rgamma(n = n, shape = 2, rate = 2)
   # Constract the CIs for the trimmed mean
   res.boot <- boot(data = x, statistic = tm_var, R = R, sim = "ordinary",
                    parallel = "multicore", ncpus = 20)
   res.ci <- boot.ci(res.boot, conf = 0.95, type = type,
                     var.t0 = var(res.boot$t[, 1]))

   # Check if CIs contain true.par
   res <- vector(mode = "integer", length = 0)
   for (ty in type) {
     res <- c(res, check_ci(ci = res.ci, ty = ty, true.par = true.par))
     names(res)[(length(res) - 1):length(res)] <-
       paste(c(ty, ty), c("lower", "upper"), sep = "_")
   }
   # Alternatively, one could use a function of the apply family, e.g. sapply.

   return(res)
 }
```

```
> ##########################
> ### Run simulation     ###
> ##########################
> set.seed(22)
> require("boot")
> sample.size <- c(10, 40, 160, 640, 2560, 10240)
> n.sim <- 1000
> type <- c("basic", "norm", "perc", "stud")
> # The object RES is used to store the results, i.e. each row corresponds
> # to non-coverage rate for the lower and upper end of the confidence intervals,
> # i.e. the percentage of times that theta < CI_l and the percentage of times
> # that theta > CI_u if the CI is denoted by (CI_l, CI_u). The last column
> # corresponds to the number of observations.
> RES <- matrix(NA, nrow = length(sample.size), ncol = length(type) * 2 + 1)
> colnames(RES) <- c(paste(rep(type, each = 2),
                           rep(c("lower", "upper"), times = length(type)),
                           sep = "_"), "n")
> for (j in 1:length(sample.size)) {
   n <- sample.size[j]
   # The object res.sim is used to store the results, i.e. each row corresponds
   # to the output of the function do_sim. This means that each row contains 0
   # and 1 encoding whether the true parameter lied in the CI or outside. See
   # function check_ci.
   res.sim <- matrix(NA, nrow = n.sim, ncol = length(type) * 2)
   for (i in 1:n.sim) {
     # Calculate CIs and check if true.par is contained
     res.sim[i, ] <- do_sim(n = n, true.par = true.par, type = type, R = 2000)
   }
   # Calculate the upper and lower non-coverage rate
   RES[j, ] <- c(apply(res.sim, 2, mean), n)
 }
```

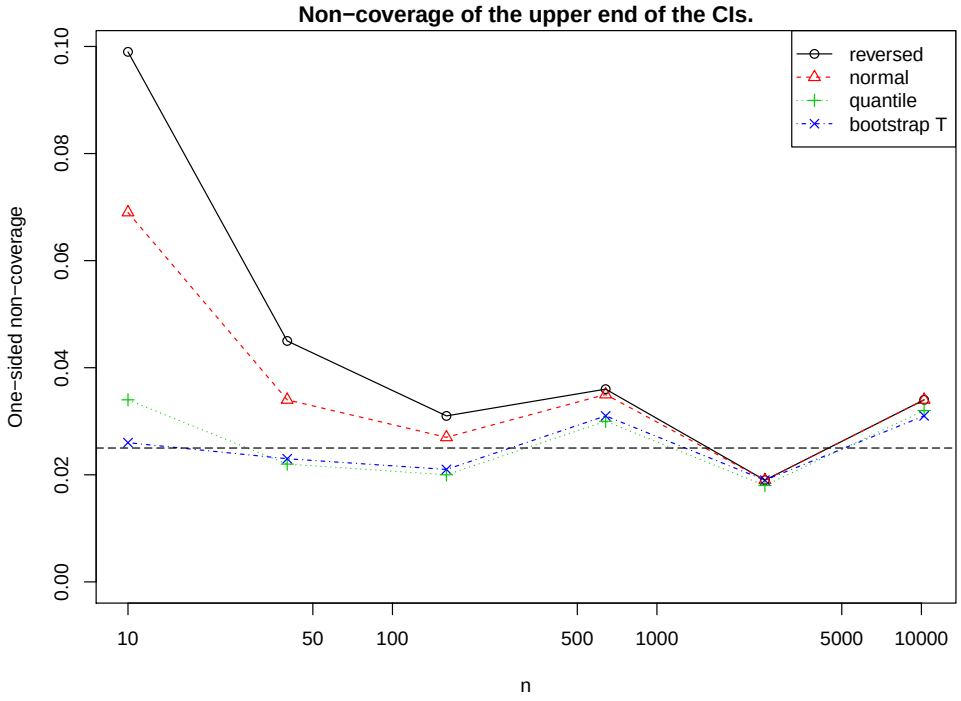
Note that the above code runs in parallel on 20 cores. We chose larger values for some parameters then you were ask to do on the exercise sheet, i.e. `R = 2000`, `n.sim = 1000`, and `sample.size = c(10, 40, 160, 640, 2560, 10240)`.

The plots have the same limits on the y-axis and we use log-scale for the x-axis.

```
> y.lim <- max(RES[, -ncol(RES)])
> # Plot of lower non-coverage
> plot(basic_lower ~ n, data = RES, col = 1, pch = 1, ylim = c(0, y.lim),
      log = "x", ylab = "One-sided non-coverage",
      main = "Non-coverage of the upper end of the CIs.")
> points(norm_lower ~ n, data = RES, col = 2, pch = 2, xlog = TRUE)
> points(perc_lower ~ n, data = RES, col = 3, pch = 3, xlog = TRUE)
> points(stud_lower ~ n, data = RES, col = 4, pch = 4, xlog = TRUE)
> lines(basic_lower ~ n, data = RES, col = 1, lty = 1, xlog = TRUE)
> lines(norm_lower ~ n, data = RES, col = 2, lty = 2, xlog = TRUE)
> lines(perc_lower ~ n, data = RES, col = 3, lty = 3, xlog = TRUE)
> lines(stud_lower ~ n, data = RES, col = 4, lty = 4, xlog = TRUE)
> abline(h = 0.025, lty = 5)
> legend("topright", legend = c("reversed", "normal", "quantile", "bootstrap T"),
        pch = 1:4, lty = 1:4, col = 1:4)
```




```
> # Plot of upper non-coverage
> plot(basic_upper ~ n, data = RES, col = 1, pch = 1, ylim = c(0, y.lim),
      log = "x", ylab = "One-sided non-coverage",
      main = "Non-coverage of the upper end of the CIs.")
> points(norm_upper ~ n, data = RES, col = 2, pch = 2, xlog = TRUE)
> points(perc_upper ~ n, data = RES, col = 3, pch = 3, xlog = TRUE)
> points(stud_upper ~ n, data = RES, col = 4, pch = 4, xlog = TRUE)
> lines(basic_upper ~ n, data = RES, col = 1, lty = 1, xlog = TRUE)
> lines(norm_upper ~ n, data = RES, col = 2, lty = 2, xlog = TRUE)
> lines(perc_upper ~ n, data = RES, col = 3, lty = 3, xlog = TRUE)
> lines(stud_upper ~ n, data = RES, col = 4, lty = 4, xlog = TRUE)
> abline(h = 0.025, lty = 5)
> legend("topright", legend = c("reversed", "normal", "quantile", "bootstrap T"),
       pch = 1:4, lty = 1:4, col = 1:4)
```



In this setting, the reversed bootstrap CI and the normal approximation CI are biased in the sense that they estimate a too small upper end of the CI for small sample sizes. There are only small differences between the CIs for large sample sizes.