STAT432 Assignment 1

Rory

5 August 2020

Question 1

```
my_rbinom <- function(n, size, prob) {
  sapply(seq_len(n), function(x) sum(ifelse(runif(size) <= prob, 1, 0)))
}
my_rbinom(5, 10, 0.4)
```

```
## [1] 5 4 3 4 5
```

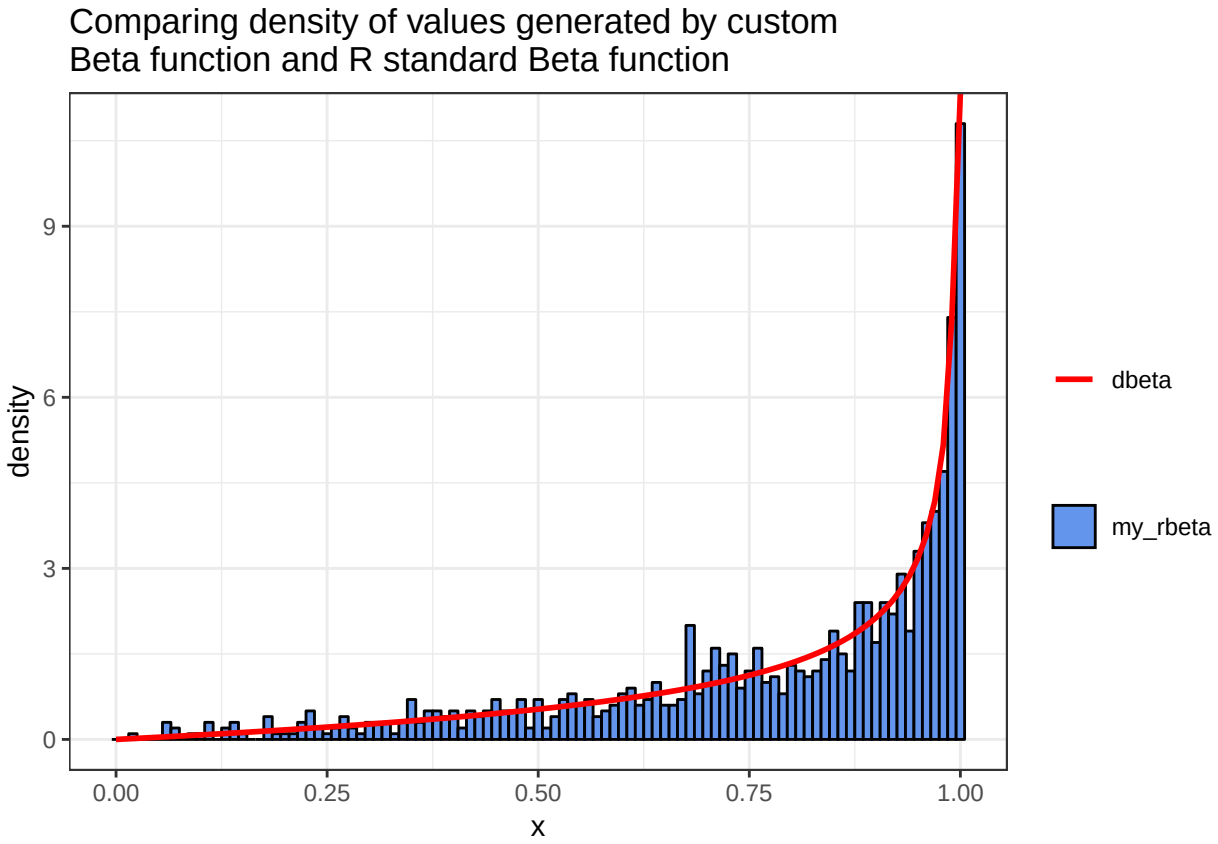
Question 2

```
my_beta <- function(n, alpha, beta) {
  params <- c(alpha = alpha, beta = beta)
  x <- rep_len(NA_real_, length.out = n)
  count <- 1

  while (count <= n) {
    u <- runif(2)
    v <- u^(1/params)
    w <- sum(v)
    if (w <= 1) {
      x[count] <- v[1]/w
      count <- count + 1
    }
  }
  return(x)
}

n <- 1000
alpha <- 2
beta <- 0.5

ggplot() +
  geom_histogram(data = data.frame(x = my_beta(n, alpha, beta)),
                 mapping = aes(x = x, y =..density.., fill = "cornflowerblue"),
                 colour = "black", binwidth = 0.01) +
  ggtitle("Comparing density of values generated by custom\nBeta function and R standard Beta function")
  theme_bw() +
  stat_function(data = (data.frame(x = seq(0, 1, length.out = 100))),
                aes(x = x, colour = "red"),
                fun = dbeta, n = 100, args = list(shape1 = alpha, shape2 = beta), size = 1) +
  scale_colour_manual("", values = c("red" = "red"), labels = "dbeta") +
  scale_fill_manual("", values = c("cornflowerblue" = "cornflowerblue"), labels = "my_rbeta")
```



The generated values appear to match the intended Beta distribution reasonably well.

Question 3

$$F(X) = u = 1 - \exp\left(-\eta(e^{(bx)} - 1)\right)$$

$$1 - u = \exp\left(-\eta(e^{(bx)} - 1)\right)$$
$$\log(1 - u) = -\eta(e^{(bx)} - 1)$$
$$1 - \frac{\log(1 - u)}{\eta} = e^{bx}$$

$$\log\left(1 - \frac{\log(1 - u)}{\eta} + 1\right)/b = x$$

```
my_rgompertz <- function(n, shape, scale) log(1 - (log(1 - runif(n))/shape))/scale
my_rgompertz(10, 0.1, 1)
```

```
##  [1] 1.3053393 1.4277809 2.1909785 0.4973284 0.2085607 1.8867221 0.4775321
##  [8] 2.7349639 0.1345173 2.1049846
```

Comparing efficiency using `system.time`:

```
shape = 0.1
scale = 1
n = 1000

system.time(my_rgompertz(n, shape, scale))
```

```
##    user  system elapsed
##   0.004   0.000   0.003
```

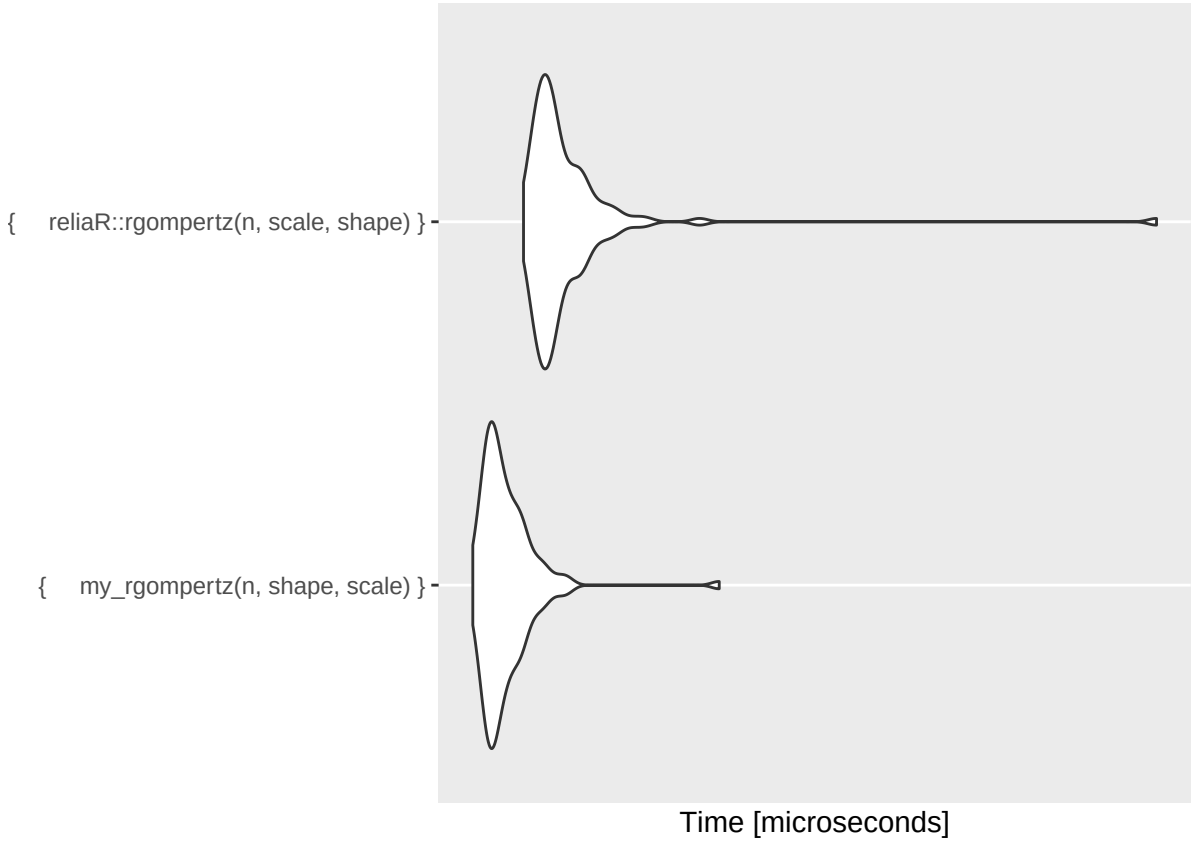
```
system.time(reliaR::rgompertz(n, scale, shape))
```

```
##    user  system elapsed
##   0.003   0.000   0.004
```

We see that the function provided in the `reliaR` package is significantly more efficient than the one I built.

Although it should be noted that the actual execution time of the function (exected multiple times) without OS or garbage collection overhead we see that it is actually slightly superior in this context.

```
microbenchmark::microbenchmark(
  { my_rgompertz(n, shape, scale)},
  { reliaR::rgompertz(n, scale, shape)}) %>% ggplot2::autoplot()
```



Question 4

a)

$$f(x) = \frac{e^{-x}}{1 - e^{-a}}$$

Therefore

$$F(x) = \int_0^x \frac{e^{-s}}{1 - e^{-a}} ds = \frac{1 - e^{-x}}{1 - e^{-a}} = u$$

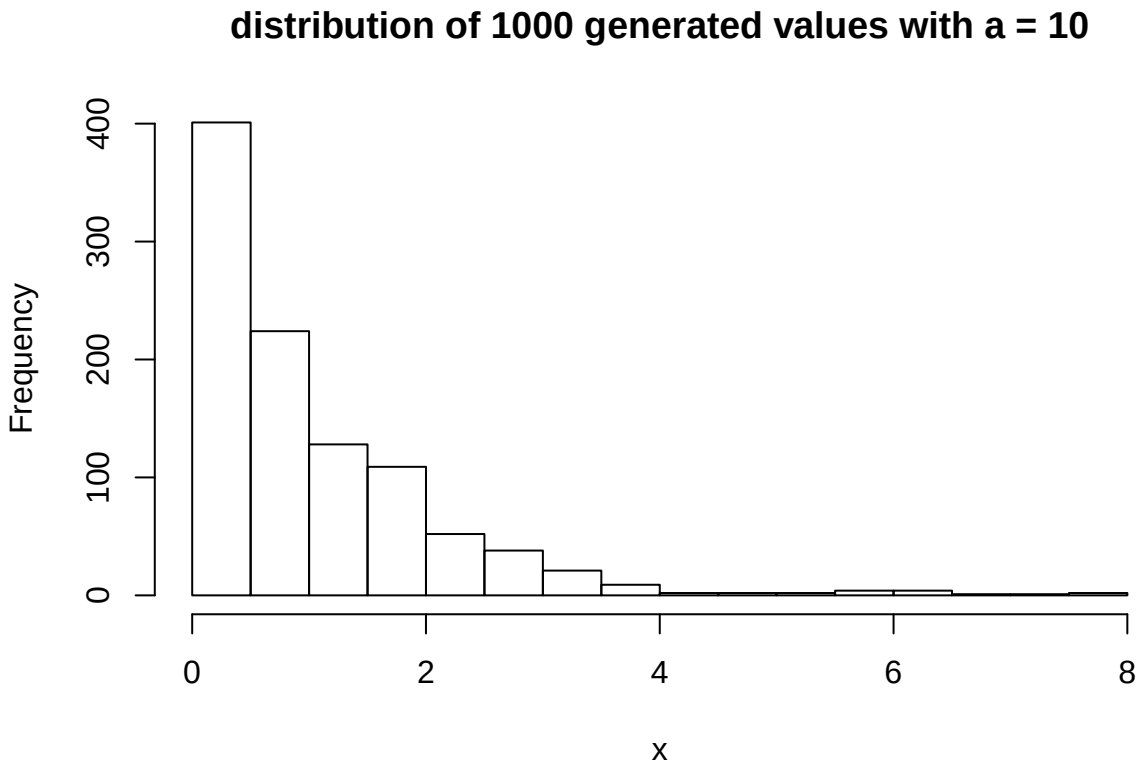
$$u = \frac{1 - e^{-x}}{1 - e^{-a}}$$
$$u(1 - e^{-a}) = 1 - e^{-x}$$
$$\frac{1}{1 - u(1 - e^{-a})} = e^{x}$$
$$x = \log\left(\frac{1}{1 - u(1 - e^{-a})}\right)$$

Algorithm

1. Generate a random variables u from $U \sim Uniform(0, 1)$
2. Return x where $x = F_x^{-1}(u) = \log\left(\frac{1}{1 - u(1 - e^{-a})}\right)$

b)

```
my_trunc_exp <- function(n, a) log(1/(1 - runif(n)*(1-exp(-a))))
hist(my_trunc_exp(1000, 10), breaks = 20,
     main = "distribution of 1000 generated values with a = 10",
     xlab = "x")
```



c)

If we choose $g(x) = Exp(1) = e^{-x}$

and we set $h(x) = \frac{\left(\frac{e^{-x}}{1 - e^{-a}}\right)}{e^{-x}} = \frac{1}{1 - e^{a}}$

as $h(x)$ is a constant, we will assume the max of to be $\frac{1}{1 - e^{-a}} = 1.000045 = c$

Algorithm

1. Generate y from $Y \sim Exp(1)$. As we are drawing from a truncated distribution, all y must be greater than `dexp(10, 1)` so that $0 \leq x \leq 10$
2. Calculate $r = \frac{f(y)}{c.g(y)} = \frac{(\frac{e^{-y}}{1-e^{-a}})}{\frac{1}{1-e^{-a}} \times e^{-y}} = 1$

At this point I've found myself a little lost, so I will wait patiently for the solution to this problem.

Question 5

a) Using classical Monte-Carlo integration

```
mci <- function(n, niter) {
  values <- sapply(seq_len(niter), function(n) {
    v <- rnorm(n)^2 + rnorm(n)^2
    sum(v >= 5)/n
  })
  results <- list()
  results[["mean"]] <- mean(values)
  results[["var"]] <- var(values)
  results
}
mci(1000, 1000)
```

```
## $mean
## [1] 0.08220782
##
## $var
## [1] 0.0006368218
```

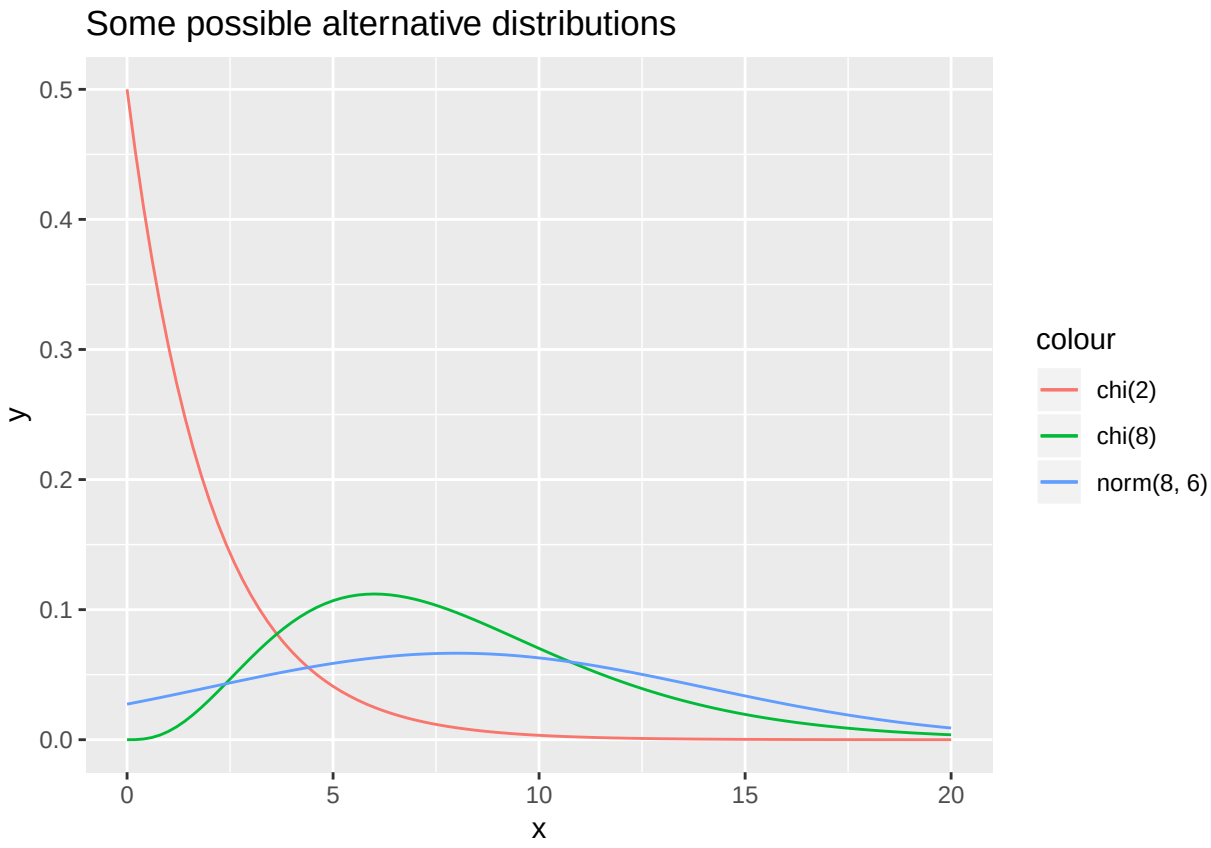
Given we can define this as a Chi Square distribution we can verify our result is close to the expected result

```
## this approximates to the result of
1 - pchisq(5, df = 2)
```

```
## [1] 0.082085
```

b) Using importance sampling method

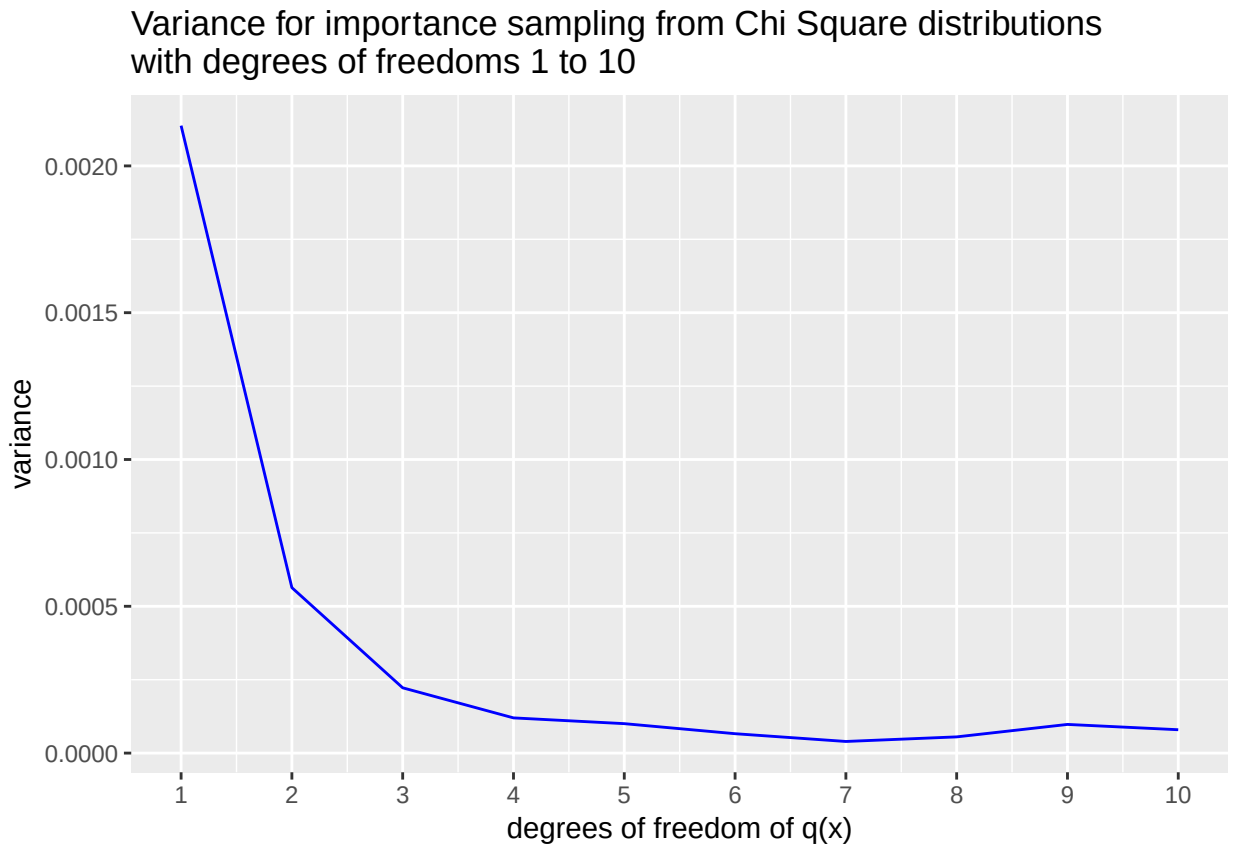
There are any number of potential alternative distributions to sample from. I looked at several distributions before deciding that I would investigate various Chi Square distributions as these seem to cover the tail well with larger degrees of freedom, e.g.:



To find the most appropriate df I ran Importance Sampling with a range of df from 1 to 10, and settled on 7 as the point where the variance settles around a low point.

```
set.seed(100)
imps <- function(df_compare, n, niter) {
  values <- sapply(seq_len(niter), function(n) {
    q <- rchisq(n, df_compare)
    h <- q >= 5
    w <- dchisq(q, 2)/dchisq(q, df_compare)
    sum(w * h)/n
  })
  results <- list()
  results[["mean"]] <- mean(values)
  results[["var"]] <- var(values)
  results
}

vars <- lapply(1:10, imps, 1000, 1000) %>% sapply(function(x) x[["var"]])
ggplot() +
  geom_line(data = data.frame(x = 1L:10L, y = vars), aes(x = x, y = y), colour = "blue") +
  scale_x_continuous(breaks = as.integer(1:10)) +
  ggtitle("Variance for importance sampling from Chi Square distributions\nwith degrees of freedoms 1 t
  xlab("degrees of freedom of q(x)") +
  ylab("variance")
```



Based on this rough result we will choose to use a Chi Square distribution with 7 degrees of freedom as our alternate sampling distribution.

```
#imps(7, 1e4, 1e4)
```

c)

I will compare the variance over different numbers of iteration settings for classicla Monte Carlo and Importance Sample methods.

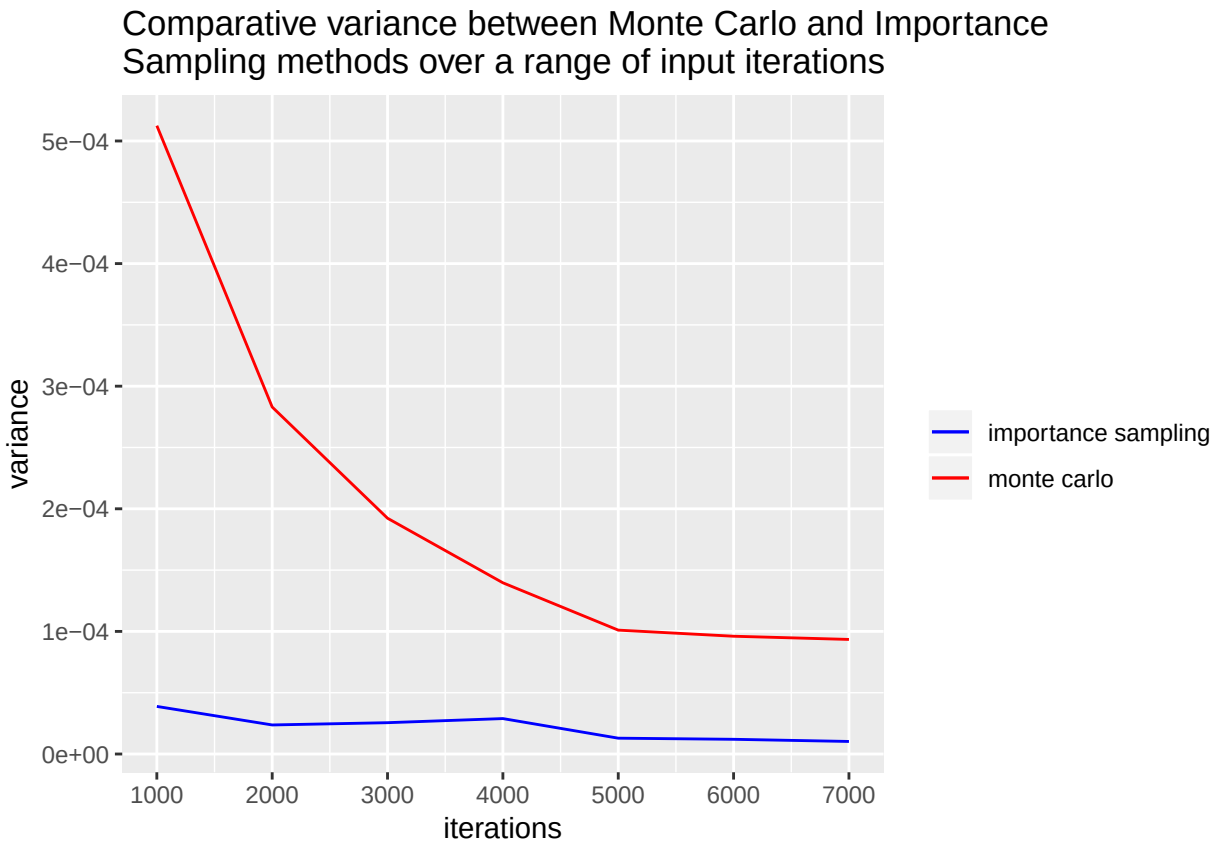
```
actual_p = pchisq(5, 2, lower.tail = FALSE)
n = 1e4

s <- seq(1e3, 0.7e4, 1e3)
mc <- rep_len(0, length(s))
is <- rep_len(0, length(s))
for (i in seq_along(s)) {
  mc[i] <- mci(n, s[i])[["var"]]
  is[i] <- imps(7, n, s[i])[["var"]]
}

results <- data.frame(
  mc = mc,
  is = is,
  iterations = s
)

ggplot() +
  geom_line(data = results, aes(x = iterations, y = mc, colour = "monte carlo")) +
```

```
  geom_line(data = results, aes(x = iterations, y = is, colour = "importance sampling")) +
  ylab("variance") +
  scale_x_continuous(breaks = as.integer(s)) +
  scale_colour_manual(name = "", values = c("monte carlo" = "red", "importance sampling" = "blue")) +
  ggtitle("Comparative variance between Monte Carlo and Importance\nSampling methods over a range of inp
```



We see that using the Importance Sampling Method our variance is much lower than traditional Monte Carlo and stabilises very quickly even at lower iterations.