

Assignment 1

Rory

23 July 2020

Question 1

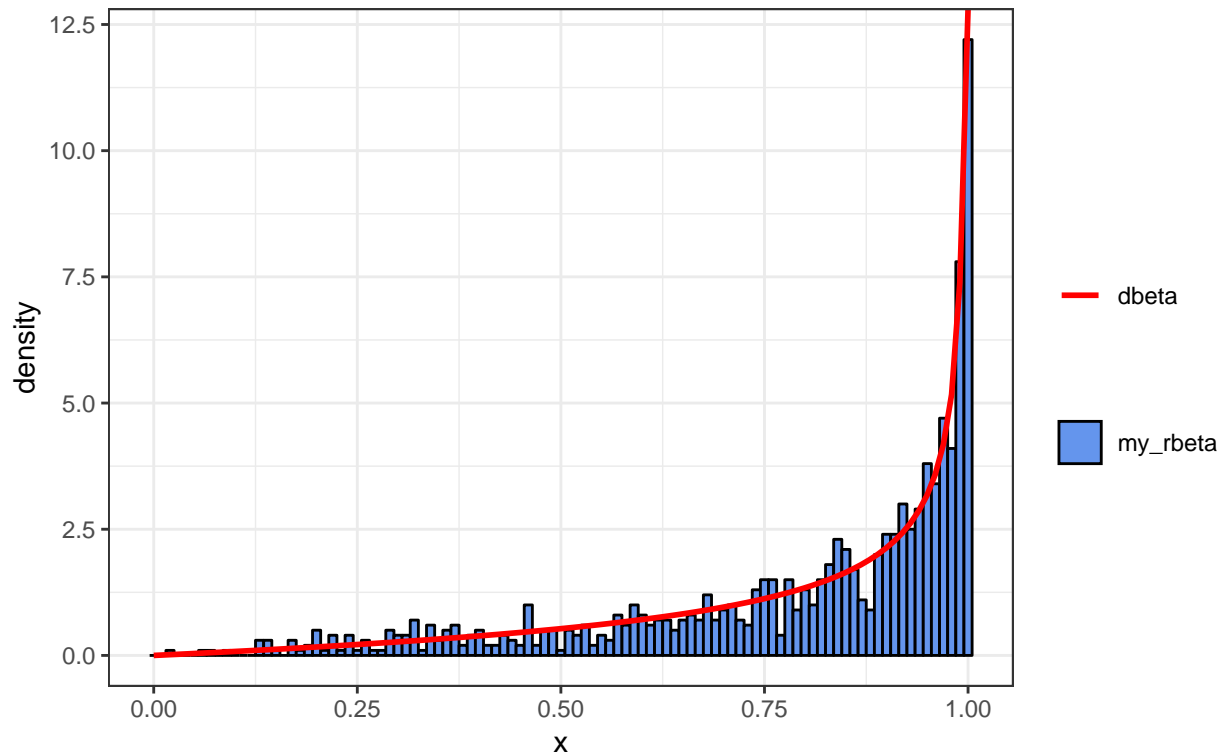
```
my_rbinom <- function(n, size, prob) {  
  apply(seq_len(n), function(x) sum(ifelse(runif(size) <= prob, 1, 0)))  
}  
my_rbinom(5, 10, 0.4)
```

```
## [1] 4 4 5 9 4
```

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")
```

Comparing density of values generated by custom Beta function and R standard Beta function



Question 3

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

$$1 - u = \exp(-\eta(e^{bx} - 1))$$

$$\log(1 - u) = -\eta(e^{bx} - 1)$$

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

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

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

```
## [1] 2.570429 1.152257 1.249283 1.030642 2.506641 1.537333 2.201158
```

```
## [8] 2.147982 3.438383 3.469292
```

what's up with the order of arguments????

```
shape = 0.1
scale = 1
n = 2000
compare <- data.frame(
  value = c(my_rgompertz(n, shape, scale), reliaR::rgompertz(n, scale, shape)),
```

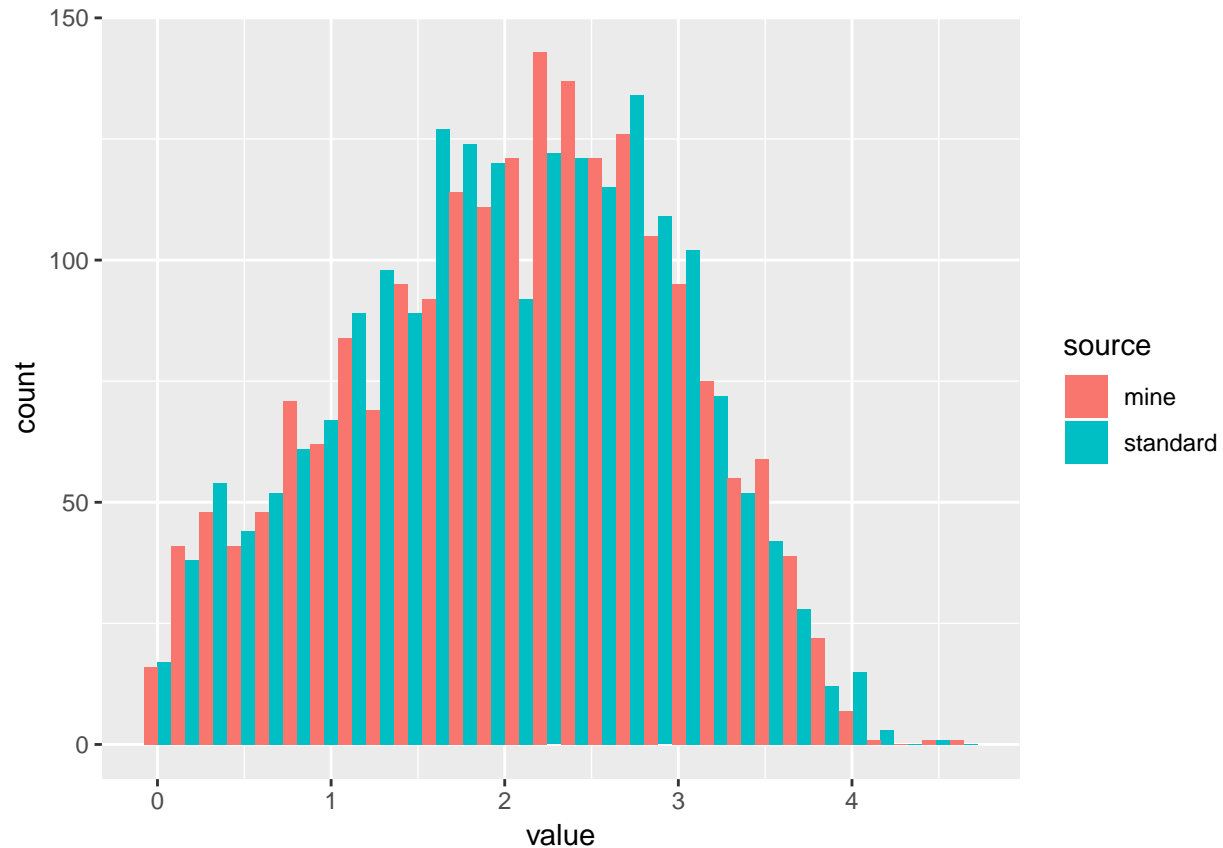
```

source = c(rep_len("mine", n), rep_len("standard", n))
)

ggplot(data = compare, aes(x = value, fill = source)) + geom_histogram(position = "dodge")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

```



```

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

##    user  system elapsed
## 0.001  0.000  0.000

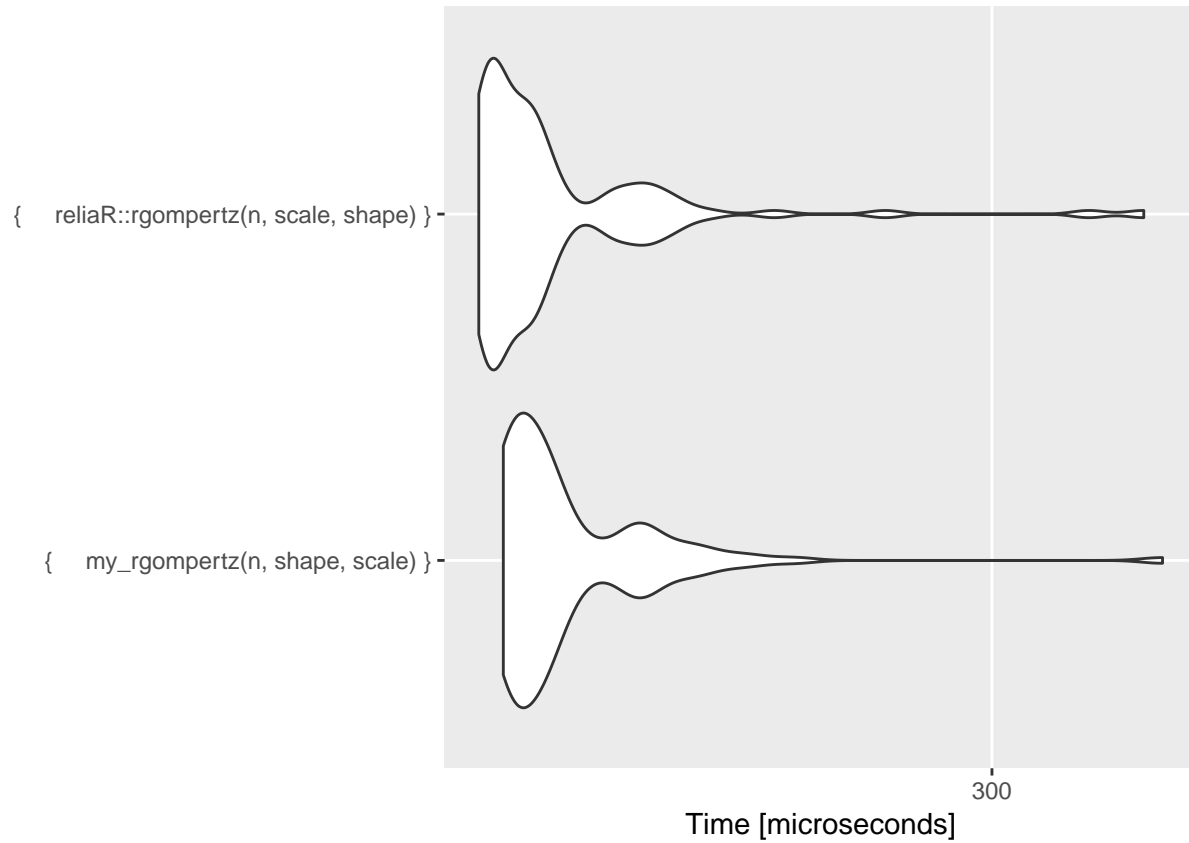
system.time(reliaR::rgompertz(n, scale, shape))

##    user  system elapsed
## 0.000  0.000  0.001

mbm <- microbenchmark::microbenchmark(
  {
    my_rgompertz(n, shape, scale)
  },
  {
    reliaR::rgompertz(n, scale, shape)
  }
)
ggplot2::autoplot(mbm)

```

Coordinate system already present. Adding new coordinate system, which will replace the existing one



Question 4

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

Therefore

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

$$u = \frac{1 - e^{-x}}{1 - e^{-a}}$$

$$u(1 - e^{-a}) = 1 - e^{-x}$$

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

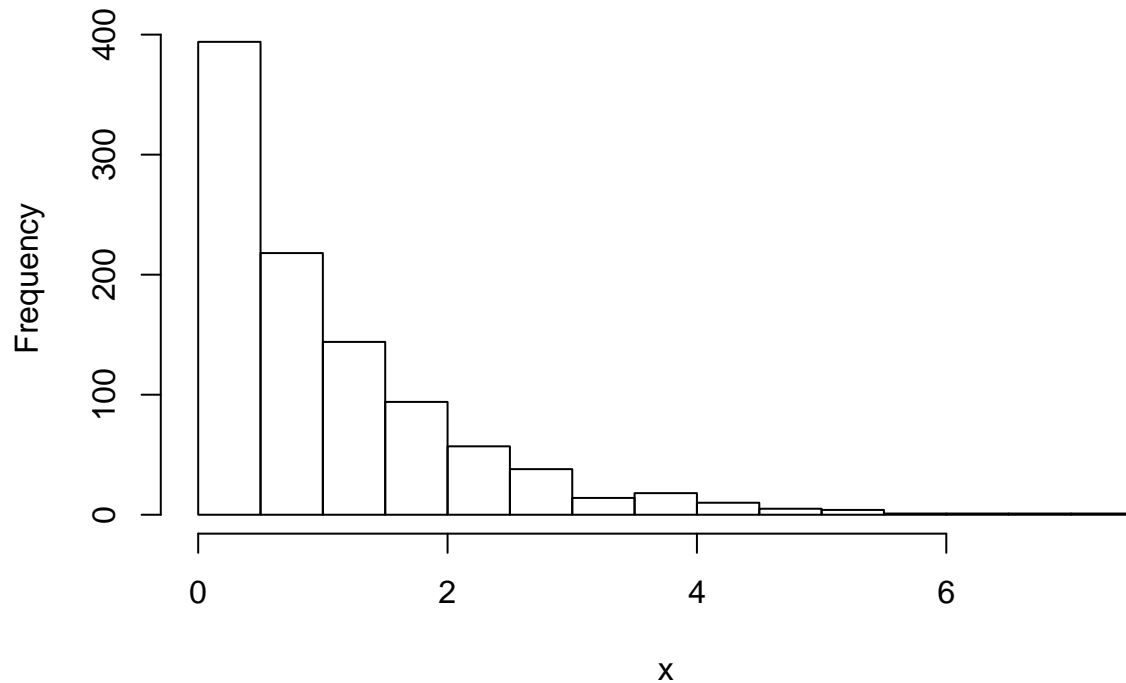
$$x = \log\left(\frac{1}{1 - u(1 - e^{-a})}\right)$$

Write algorithm

1. Generate n random variables u from $U \sim Uniform(0, 1)$
2. Return $x = F_x^{-1}(u)$ for each u

```
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")
```

distribution of 1000 generated values with a = 10



1. Generate y from $Y \sim \text{Exp}(1)$ where $0 \leq x \leq 10$
2. Calculate $r = \frac{f(y)}{c \cdot q(y)}$
3. If $u < r$, set $x = y$, otherwise return to Step 1

Checking for $c = \max \frac{f(x)}{q(x)}$

```
pdf <- function(x, a) exp(-x)/(1 - exp(-a))
s <- seq(0, 10, 0.01)
```

```
c <- max(sapply(s, pdf, 10)/dexp(s, 1))
c
```

```
## [1] 1.000045
```

```
yvec <- dexp(runif(1, min = 0, max = 10), 1)
```

Question 5

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] 0.08213605
##
## $var
## [1] 0.0004925439

```

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)

```

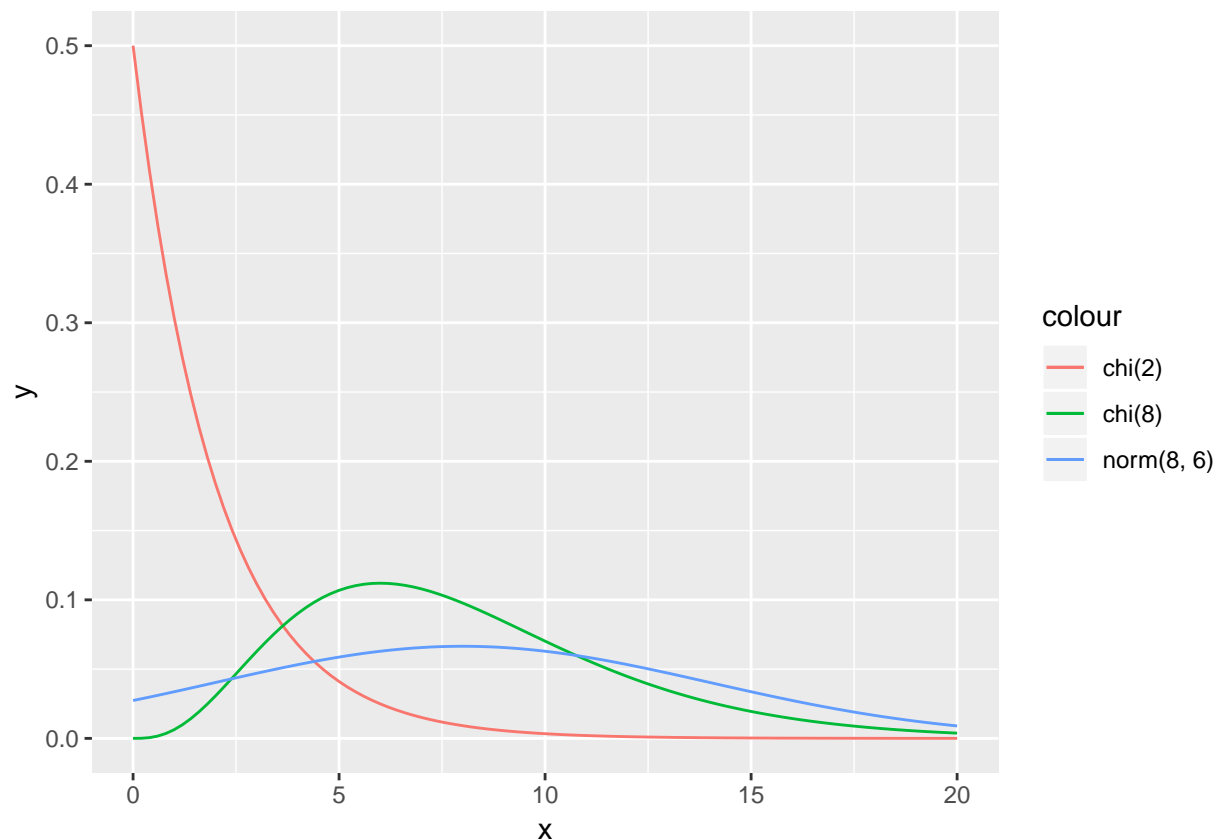
```

## [1] 0.082085

```

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.



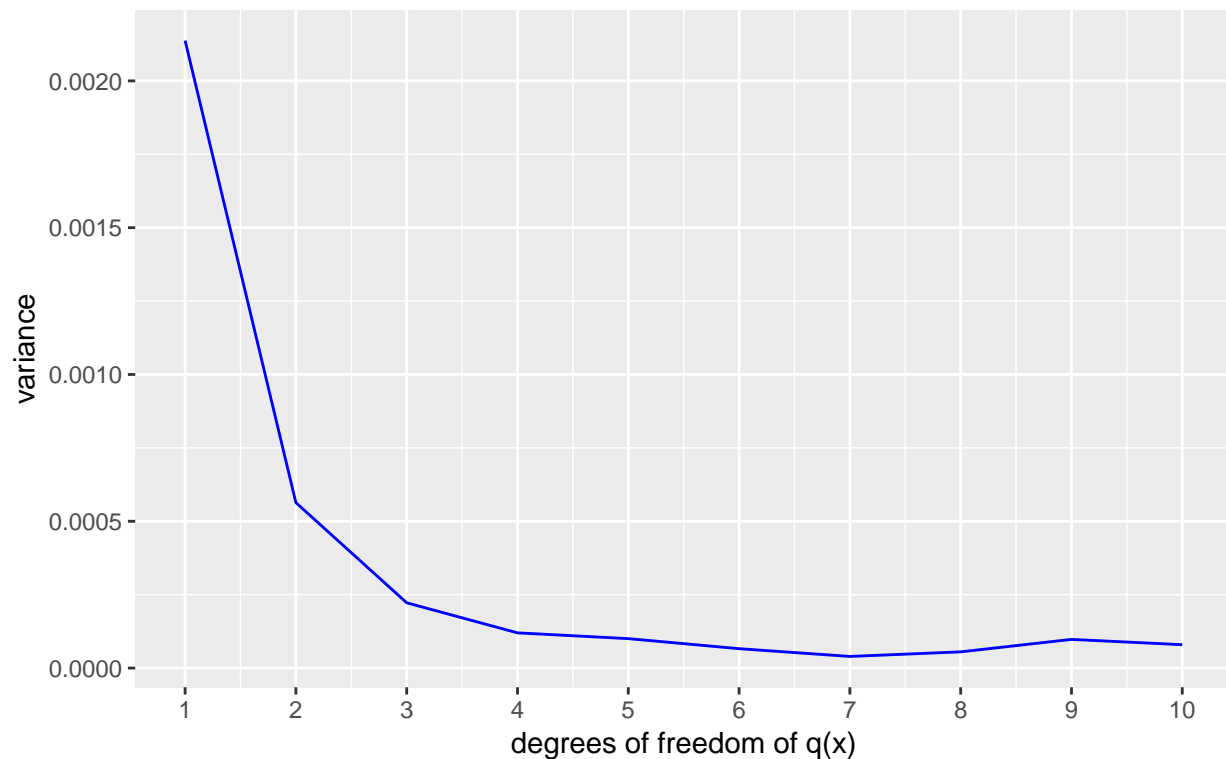
```

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 to 10")
  xlab("degrees of freedom of q(x)") +
  ylab("variance")

```

Variance for importance sampling from Chi Square distributions
with degrees of freedoms 1 to 10



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)
```

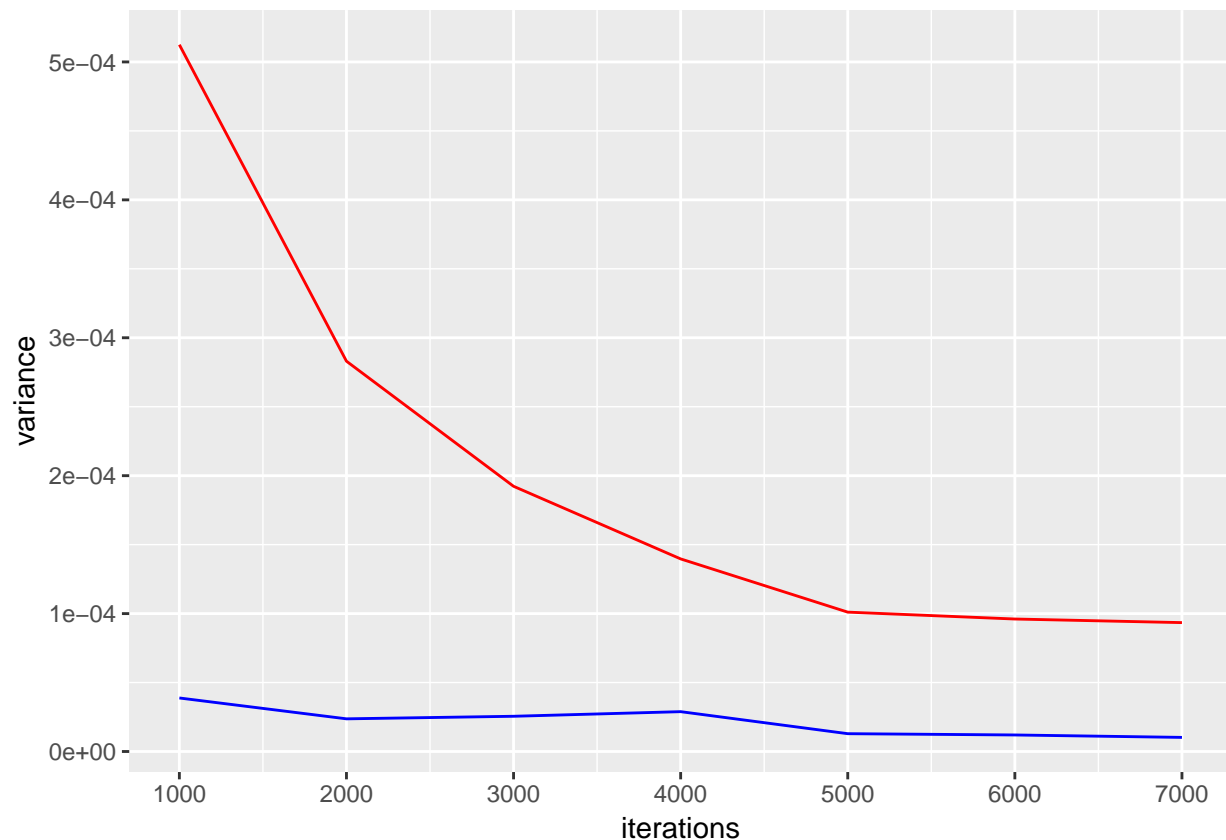
In order to compare the performance of the methods I will write slightly customised functions to demonstrate ...

```
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 = "red") +
  geom_line(data = results, aes(x = iterations, y = is), colour = "blue") +
  ylab("variance") +
  scale_x_continuous(breaks = as.integer(s))
```



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.