STAT 243: Final Project

Jonathan Larson, Chenzhi Li, Courtney Schiffman, & Shamindra Shrotriya

December 17th, 2015

1 Package source

The package source file and installation instruction detailscan be found in the following location:

- The source file (`ars.tar.gz`) is located here: https://github.com/shamindras/ars/blob/master/ars.tar.gz
- The installation instructions are located here: here: https://github.com/shamindras/ars/blob/master/README.md

2 Function Design

We chose to use a functional coding style as opposed to an object-oriented coding style, and composed our main function `ars` from the auxiliary functions that are summarized in Table 1. Note that we employed a naming convention wherein all auxiliary functions begin with the prefix `faux`. In order to encourage modularity, we distributed these auxiliary functions among our group. This necessitated an explicit mapping of the output of early functions to the input of later functions. Note also that objects that translate to the same ideas in Gilks and Wild [1992] are named in a similar way to each other, and also to how those objects are named in the paper. For example, `inp_Dvec` is used consistently to represent D, and `inp_xvec` is used consistently to represent T_k.

A pseudocode summary of the overall function `ars` is included here:

```
ars <- function(n, g, D, k = 100) {
  Check for log-concavity
  Get initial points in Tk
  Get h, uk, lk, and sk
  while (length of sample < n) {
    Sample x* from s
    Generate w ~ uniform(0,1)
    Determine which interval x* falls into
    if (w passes squeezing test) {
      Include x* in sample
    } else if (w passes rejection test) {
      Include x* in sample
      if (there are no numerical issues with the derivative) {
        Include x* in Tk
        Recalculate z, u, l, and s
      }
    }
  }
  return(sample)
}
```

3 Testing

We used `testthat` throughout our work on this project. We tested our functions as we created them, and in some cases we modeled test-driven development by writing the tests before the function. Writing tests for each others' functions was our main method of code review, although we did engage in further code review as well as utilising pair programming to systematically discuss and handle tough corner cases.

The tests for our auxiliary functions mainly took these three forms:

1. Validity of input. For example, testing that

Table 1: Summary of the inputs/ outputs of the auxiliary functions.

Function Name	Summary
`faux_CheckLogConcavity`	Takes g and D and returns `TRUE` if g is log-concave. We chose to do one global check at the beginning as opposed to multiple checks throughout
`faux_hx`	Takes g and returns h.
`faux_findmode`	Takes g and D and returns the mode of g. Obtaining initial points in T_k on either side of the mode was how we ensured $h'(x) > 0$ and $h'(x_k) < 0$.
`faux_InitChoose`	Takes g, D, and an initial k and returns T_k.
`faux_hPrimex`	Takes g and T_k and returns $h'(T_k)$.
`faux_Lkx`	Takes g and T_k and returns l_k.
`faux_Zj`	Takes g, T_k, and D and returns the z_j.
`faux_uInterval`	Takes the z_j and returns the intervals between the z_j.
`faux_Ukx`	Takes g and T_k and returns u_k.
`faux_Skx`	Takes u_k and the intervals between the z_j and returns s_k.
`faux_SampleSkx`	Takes u_k and the intervals between the z_j and returns a sample from s_k.

   (a) g is an R function;

   (b) D is a numeric vector of length 2; and

   (c) T_k is a numeric vector.

2. Validity of output of `R` object types. For example, testing that

   (a) `faux_CheckLogConcavity` outputs a logical vector of length 1;

   (b) `faux_hx` outputs an R function; and

   (c) `faux_Ukx` outputs a list of R functions.

3. Validity of output of values. For example, testing that

   (a) `faux_CheckLogConcavity` detects that $g(x) = \phi(x)$ is log-concave;

   (b) `faux_hPrimex` returns $h'(x) = -2$ for $g(x) = 2e^{-2x}$; and

   (c) `faux_findmode` returns 0 for $g(x) = \phi(x)$.

In addition to the tests for each auxiliary function, we also wrote tests for the overall `ars` function, checking its output against theory. This modular testing structure allowed us to easily create an overall testing function `test-main.R` which combined all tests into one. After all was said and done, we tested the package on our dev environments using Windows and Mac.

CAVEAT: This function relies on the package `numDeriv` for numerical differentiation. Depending on the specs of your computer (CPU/ RAM availibility) some of tests may have trouble converging as a result. We have tested on Mac/ PC and all our tests have passed successfully numerous times. As a reference guide to specs, our Mac used the following 2.5GHz Macbook Pro 15" Retina Display specs:http://www.apple.com/macbook-pro/specs-retina. We recommend using similar specs when testing.

Please see the Appendix for how to use the function with end-to-end testing performed on the validity of the output (e.g. histograms). The Appendix also contains a printout of our github `README.md` file which has an installation guide for the package.

4 Documentation

We used `roxygen2` to facilitate the creation of documentation. Essentially it allows us to create the `CRAN` style reference manual with a single command. We maintained the `roxygen2` style throughout the coding process, ensuring that function descriptions, inputs, outputs, and examples were up-to-date. We followed the R Style Guide for coding and linting practices, and heavily commented our code to ensure clarity and to facilitate code review. Please see appendix for the auto-generated reference manual.

5 Collaboration

We used Git in order to collaborate on the code itself. We also created a private Slack group for collaboration and real-time communication. Slack employs a markdown-based text formatting interface which allows easy formatting of code and text. Lastly, we used a Google doc for group documentation, including meeting minutes, technical notes on the `ars` function, and to-do lists.

We all contributed to the writing of functions, tests, and documentation, with major contributions breaking down as follows:

1. Jonathan wrote `faux_CheckLogConcavity` and assisted with `faux_hx`, `faux_Lkx`, `faux_Ukx`, and `faux_SampleSkx`.
2. Chenzhi wrote `faux_hx`, `faux_hPrimex`, `faux_Lkx` and `faux_Zj`, as well as their tests.
3. Courtney wrote `faux_SampleSkx`, `faux_Skx`, and `faux_uInterval`, and helped to create the final `ars` function.
4. Shamindra helped design the testing-documentation-coding-git workflow which enabled the team to develop with greater efficiency and also assisted in the coding of `faux_findmode`, `faux_InitChoose`, `faux_hPrimex`, `faux_Lkx`, `faux_Ukx`, and pair programmed with Courtney on some aspects of `ars.R`.

6 Appendix

6.1 Using the ars Function and some Validity Testing

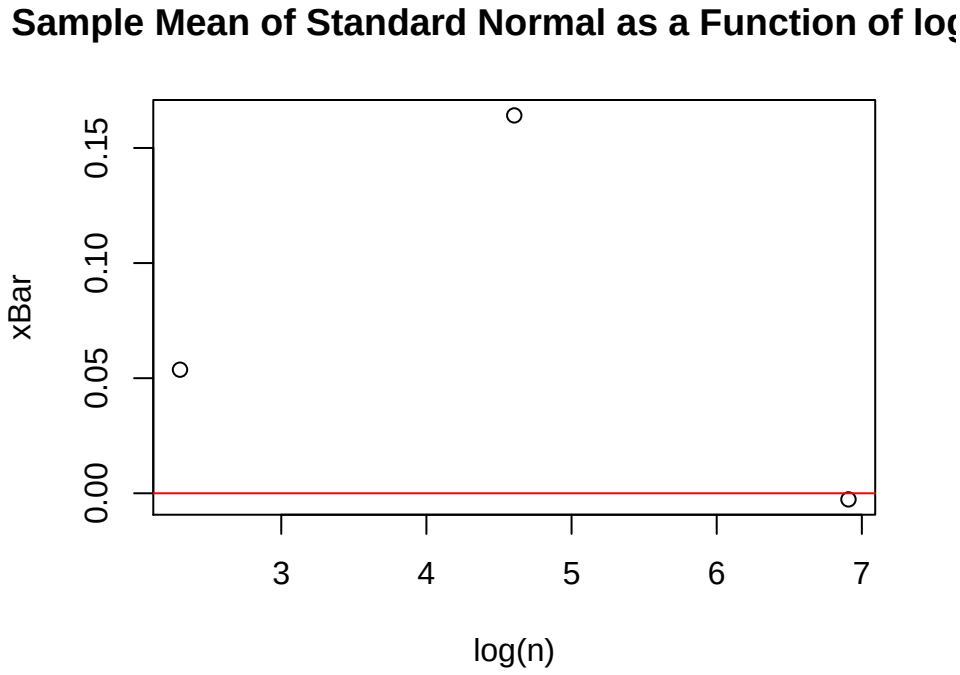
Below are some test use cases for the **ars** function with detailed comments. Also shown are some standard statistical tests and histograms which can help determine the validity of the distribution of the sample points returned from the **ars** function. This is effectively a demonstration of our end-to-end testing methodology.

```
install.packages("../ars.tar.gz", repos=NULL, type="source")
library(ars)

# We can reference a standard normal density using R's built in 'dnorm' function
set.seed(0)
dnorm_N0_1_1000_Rinbult <- ars(n = 1000, g = dnorm, D = c(-Inf,Inf))

# We can reference a non-standard normal density using R's built in 'dnorm'
# function as well e.g a N(5, 3) density
# This is done using an anonymous function call
set.seed(0)
dnorm_N5_3_1000_Rinbult <- ars(n = 1000, g = function(x) dnorm(x = x, mean = 5
                                                               , sd = 3)
                               , D = c(-Inf,Inf))

# We can reference a density using an explicit form for g(x) directly by writing
# it as an explicit function of x e.g. a N(0, 1) density can be done as follows
set.seed(0)
dnorm_N0_1_1000_exf <- ars(n = 1000
                           , g = function(x) (1/sqrt(2*pi))*exp(-(x^2)/2)
                           , D = c(-Inf,Inf))

# Other densities can also be tested as follows
# Chi-Squared with 5 degrees of freedom
set.seed(0)
dchisq_5_1000_Rinbult <- ars(1000
                             , g = function(x) dchisq(x, df = 5)
                             , D = c(0, Inf))

# Other densities can also be tested as follows
# Exponential(10)
set.seed(0)
dexp_10_1000_exf <- ars(1000
                        , g = function(x) 10*exp(-10*x)
                        , D=c(0, Inf))
```

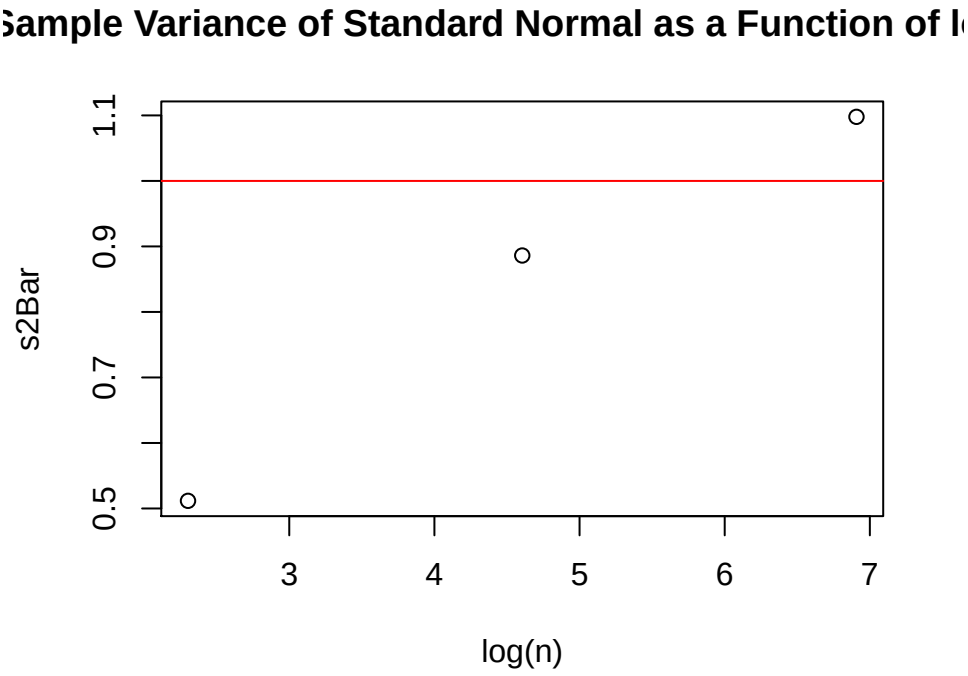
For the above densities we can conduct some some standard statistical tests and histograms which can help determine the validity of the distribution of the sample points returned from the **ars** function. This is effectively a demonstration of our end-to-end testing methodology.

```
 ## Define various sample sizes
 n <- c(1e1,1e2,1e3)

 ## Standard normal
 # Draw samples
 x <- sapply(X = n,
             FUN = ars,
             g = dnorm,
             D = c(-Inf,Inf))
 # Plot of sample mean as a function of sample size
 xBar <- unlist(lapply(X = x,
                       FUN = mean))
 plot(x = log(n),
```

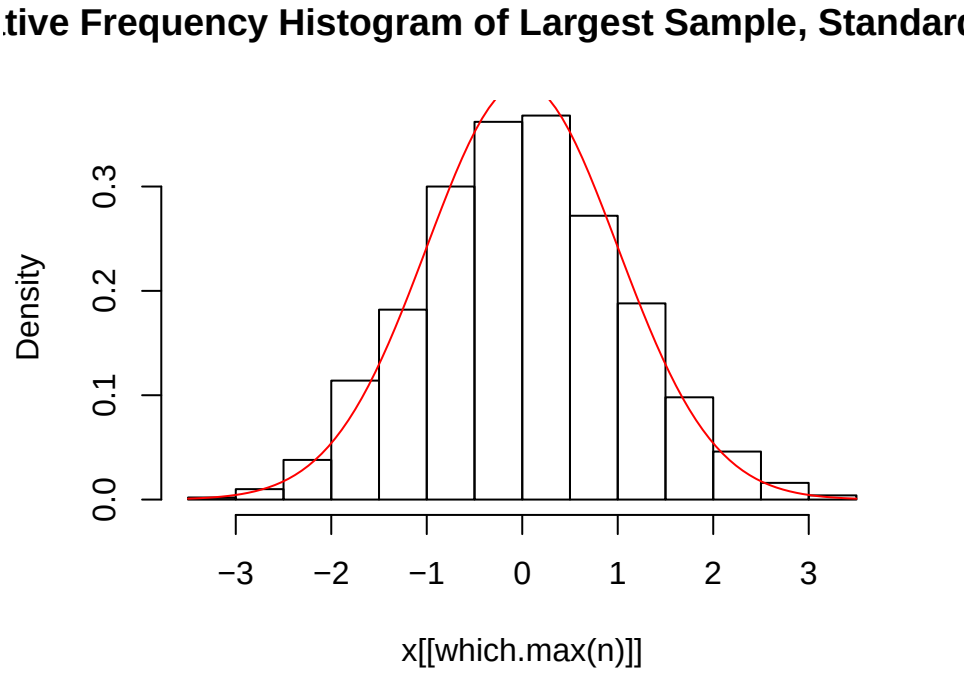
```
     y = xBar,
     main = 'Sample Mean of Standard Normal as a Function of log(n)')
abline(h = 0,
       col = 'red')
```



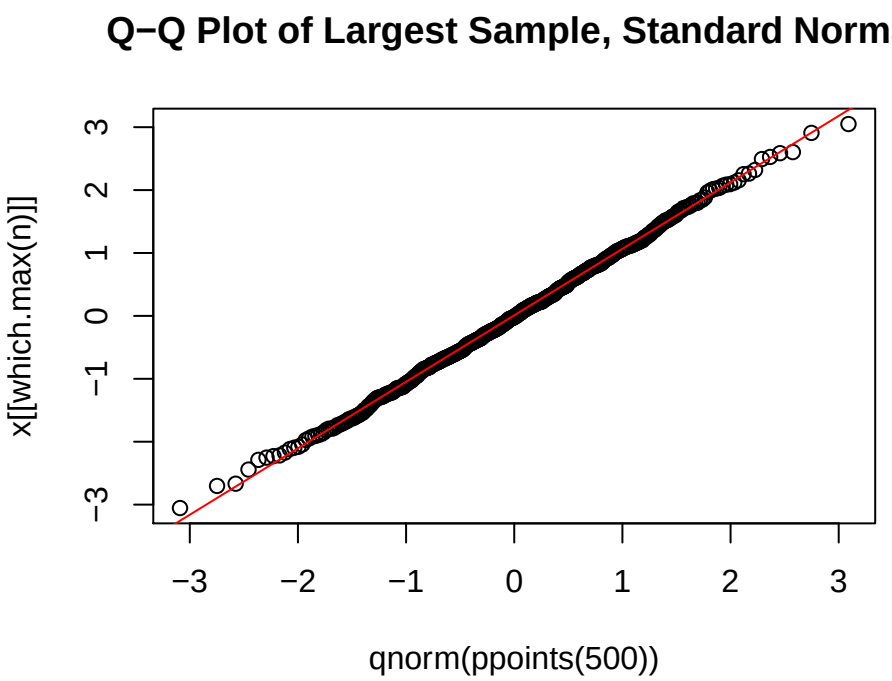

```
# Plot of sample variance as a function of sample size
s2Bar <- unlist(lapply(X = x,
                       FUN = var))
plot(x = log(n),
     y = s2Bar,
     main = 'Sample Variance of Standard Normal as a Function of log(n)')
abline(h = 1,
       col = 'red')
```



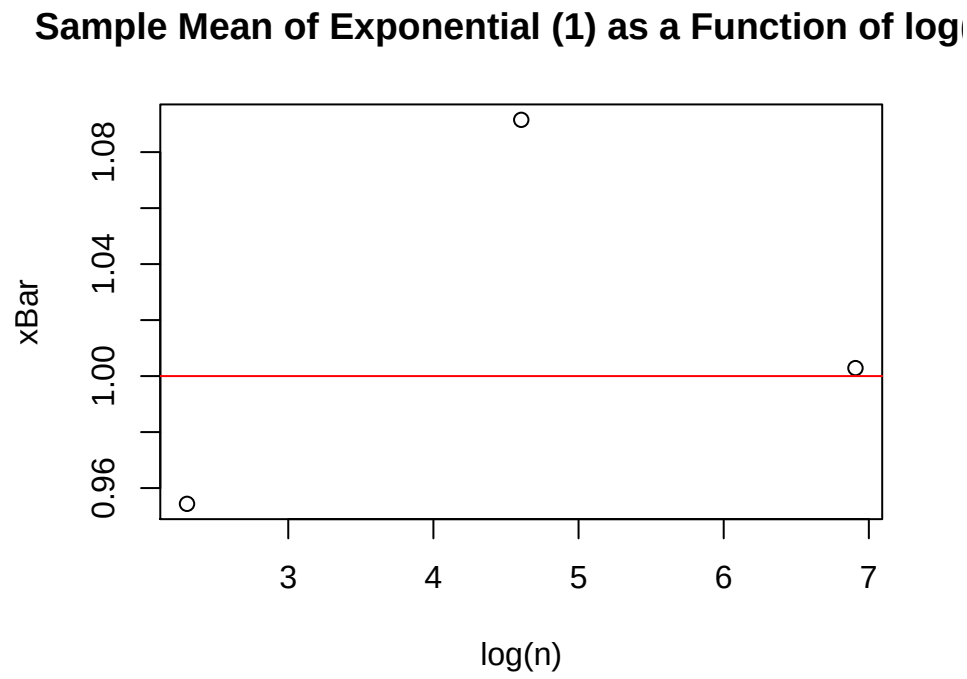

```
# Relative frequency histogram of largest sample with superimposed density
hist(x[[which.max(n)]],
     freq = FALSE,
     main = 'Relative Frequency Histogram of Largest Sample, Standard Normal')
curve(dnorm,
      add = TRUE,
      col = 'red')
```



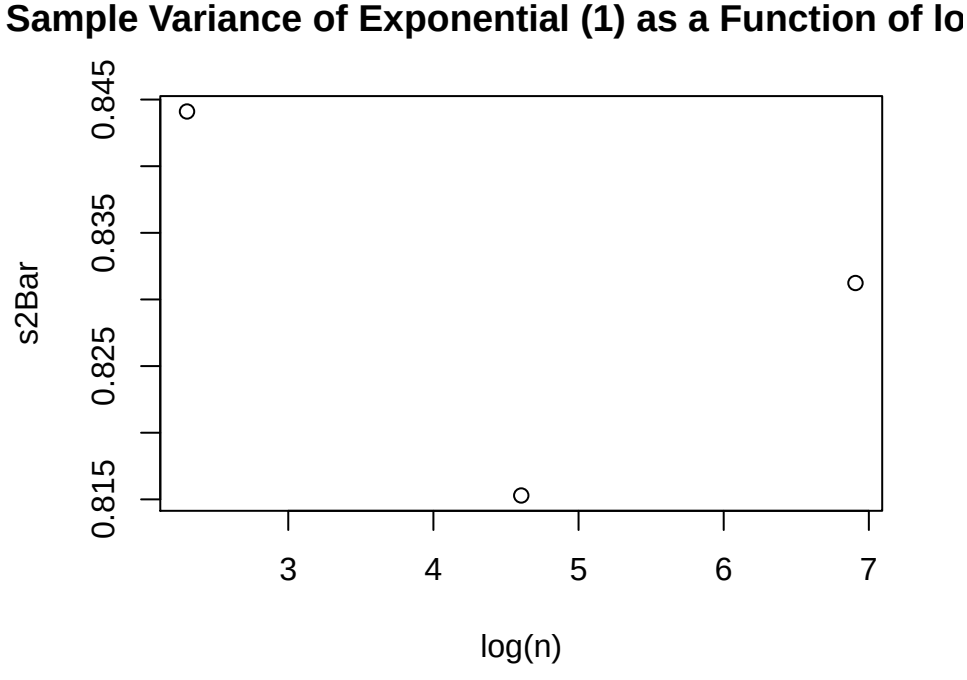
```r
# Q-Q plot of largest sample
qqplot(x = qnorm(ppoints(500)),
       y = x[[which.max(n)]],
       main = 'Q-Q Plot of Largest Sample, Standard Normal')
qqline(y = x[[which.max(n)]],
       distribution = qnorm,
       col = 'red')
```



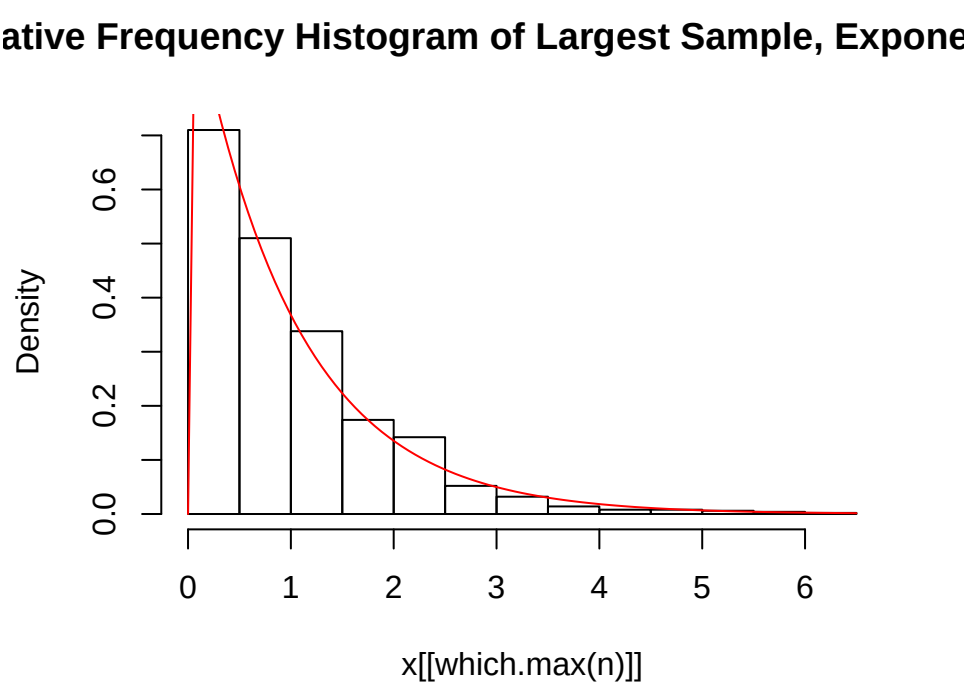

```r
## Exponential, rate 1
# Draw samples
x <- sapply(X = n,
            FUN = ars,
            g = dexp,
            D = c(0,Inf))
# Plot of sample mean as a function of sample size
xBar <- unlist(lapply(X = x,
                      FUN = mean))
plot(x = log(n),
     y = xBar,
     main = 'Sample Mean of Exponential (1) as a Function of log(n)')
abline(h = 1,
       col = 'red')
```



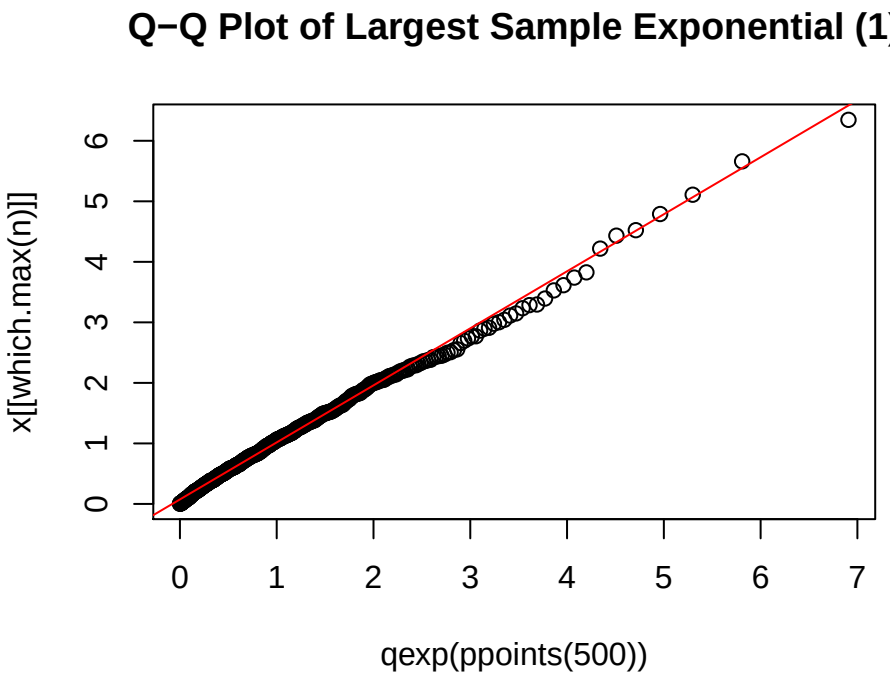

```
# Plot of sample variance as a function of sample size
s2Bar <- unlist(lapply(X = x,
                       FUN = var))
plot(x = log(n),
     y = s2Bar,
     main = 'Sample Variance of Exponential (1) as a Function of log(n)')
abline(h = 1,
       col = 'red')
```



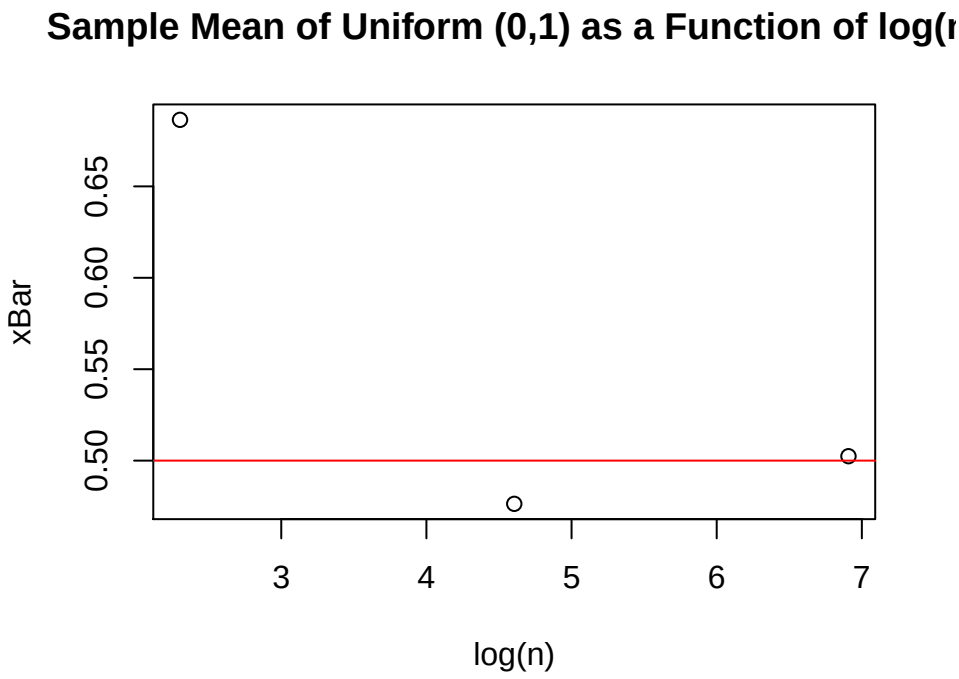
```r
# Relative frequency histogram of largest sample with superimposed density
hist(x[[which.max(n)]],
     freq = FALSE,
     main = 'Relative Frequency Histogram of Largest Sample, Exponential (1)')
curve(dexp,
      add = TRUE,
      col = 'red')
```



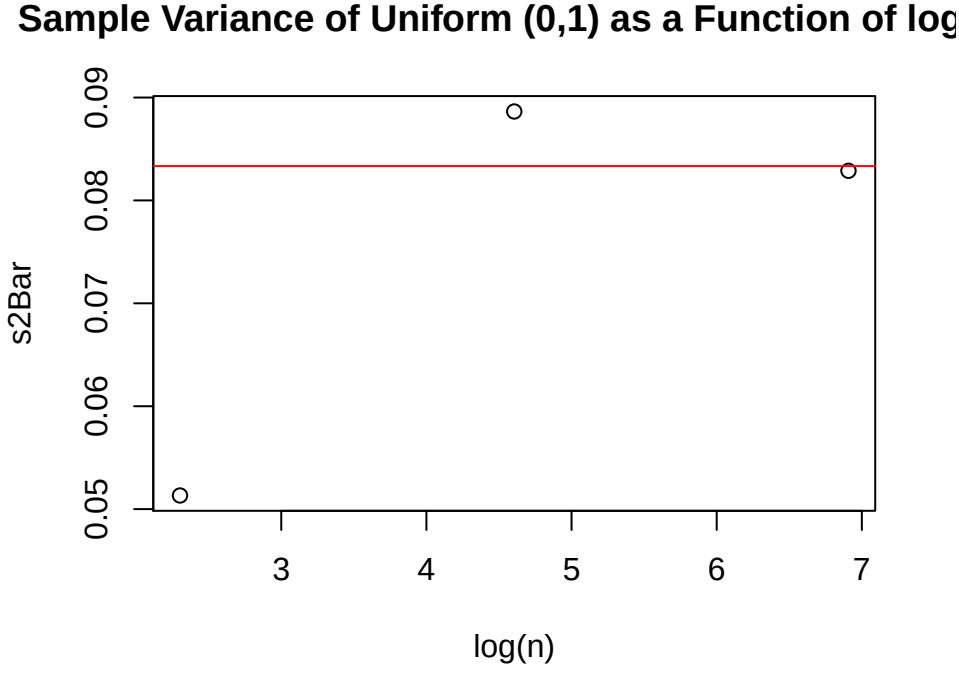

```r
# Q-Q plot of largest sample
qqplot(x = qexp(ppoints(500)),
       y = x[[which.max(n)]],
       main = 'Q-Q Plot of Largest Sample Exponential (1)')
qqline(y = x[[which.max(n)]],
       distribution = qexp,
       col = 'red')
```



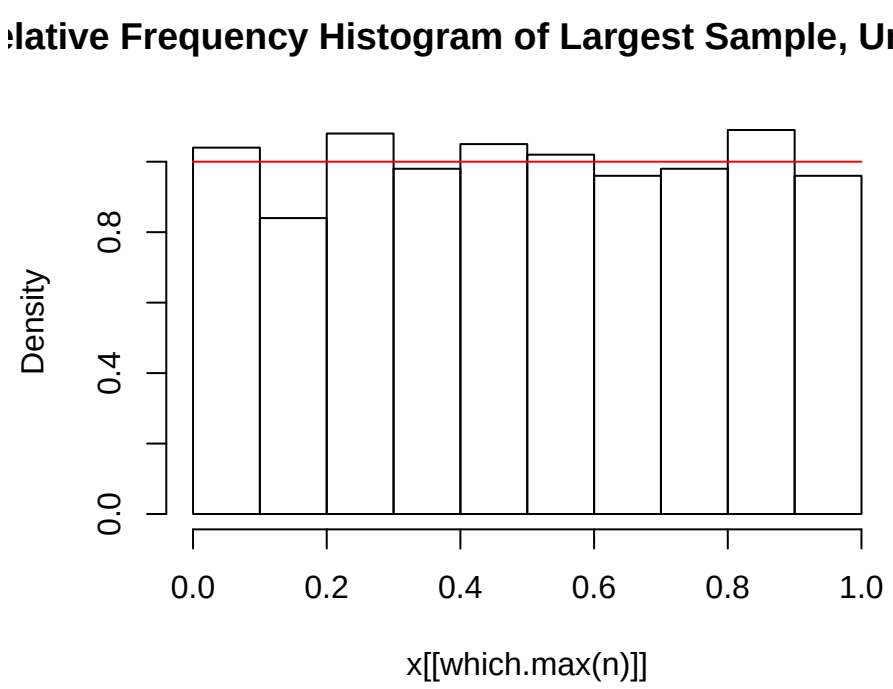

```
## Uniform (0,1)
# Draw samples
x <- sapply(X = n,
            FUN = ars,
            g = dunif,
            D = c(0,1))
# Plot of sample mean as a function of sample size
xBar <- unlist(lapply(X = x,
                      FUN = mean))
plot(x = log(n),
     y = xBar,
     main = 'Sample Mean of Uniform (0,1) as a Function of log(n)')
abline(h = 0.5,
       col = 'red')
```



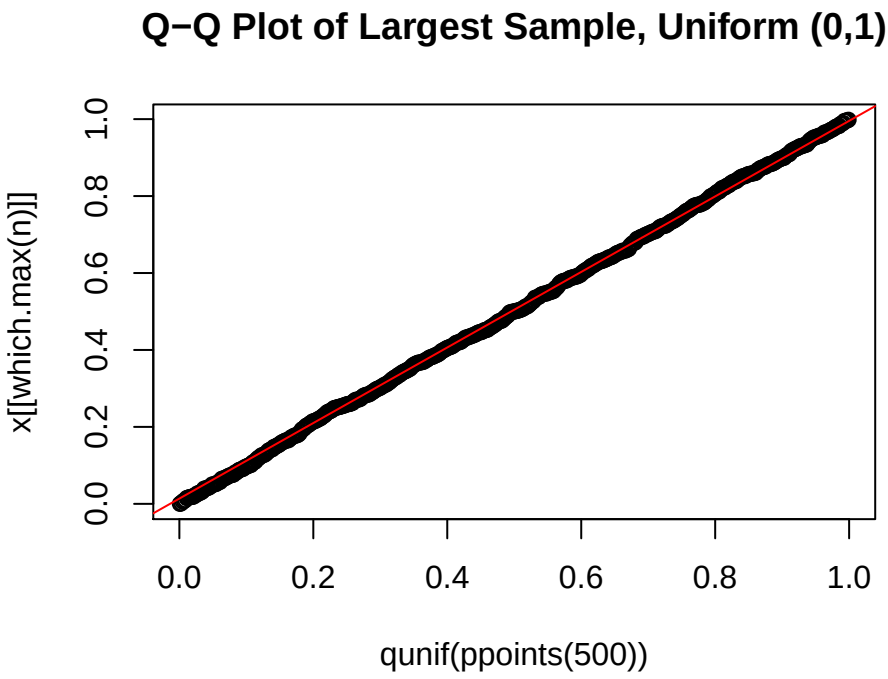

```
# Plot of sample variance as a function of sample size
s2Bar <- unlist(lapply(X = x,
                       FUN = var))
plot(x = log(n),
     y = s2Bar,
     main = 'Sample Variance of Uniform (0,1) as a Function of log(n)')
abline(h = 1/12,
       col = 'red')
```



```r
# Relative frequency histogram of largest sample with superimposed density
hist(x[[which.max(n)]],
     freq = FALSE,
     main = 'Relative Frequency Histogram of Largest Sample, Uniform (0,1)')
curve(dunif,
      add = TRUE,
      col = 'red')
```




```r
# Q-Q plot of largest sample
qqplot(x = qunif(ppoints(500)),
       y = x[[which.max(n)]],
       main = 'Q-Q Plot of Largest Sample, Uniform (0,1)')
qqline(y = x[[which.max(n)]],
       distribution = qunif,
       col = 'red')
```



References

W. R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 41(2):337–348, 1992. ISSN 00359254, 14679876. URL `http://www.jstor.org/stable/2347565`.