STAT 243 Final Project

Ugur Yildirim, Shubei Wang, Ramon Crespo

December 2018

1 Introduction

The objective of this project is to write a package using the **R** programming language that performs adaptive rejection sampling based on the tangent approach described by Gilks and Wild (1992). The code was developed by taking three aspects into consideration: modularity, ease of use and useful feedback for the user. The package is summarized in two files, *ars.R* and *auxiliary.R*. While *ars.R* performs the calculations, *auxiliary.R* defines the modular functions that *ars.R* uses (including functions that check for validity of the inputs from the user and provides feedback if the program encounters any errors in the implementation of the algorithm). The function *ars* is composed of three steps: i) initialization, ii) sampling, and iii) update steps. For a full description, refer to the code flow section of this report. The full package is available to the public and can be installed by running *devtools* :: *install_github*("*uguryi/ars*") in **R** (on some computers the $INSTALL\_opts$ = "$--install-tests$" option needs to be included explicitly).

2 Code Flow

When the user calls the main function of our package, *ars*, the function initializes T_k as a numeric vector of length k and evaluates h and $h^{'}$ on these points. Next, z is initialized as a $k+1$ numeric vector. The final preliminary step before sampling is to initialize the denominator of s_k, which the function later uses to select the piece to sample from. Throughout the initialization step, the function conducts a series of tests to make sure that the input provided by the user is valid. For example, the function checks that h provided is concave. We also give the user some flexibility as to which parameters need to be present when calling the function. For example, even if the user does not provide h', our function simply calculates it numerically.

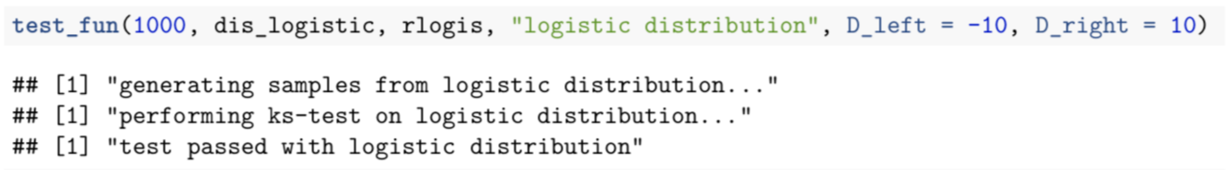
As far as the sampling step is concerned, the function uses a *while* loop to continue sampling until the target sample size is reached. In order to sample from s_k, the function first selects the piece to be sampled from with probabilities proportional to the area under each piece. Once the piece is selected, the function samples an x^* from the truncated exponential function that it corresponds to and evaluates $l_k(x^*)$ and $u_k(x^*)$. Next, the function also generates a random sample w from $Unif(0,1)$ and checks if $w \leq e^{l_k(x^*)-u_k(x^*)}$. If so, x^* is accepted and added to the vector of final samples.

If $w > e^{l_k(x^*)-u_k(x^*)}$, then the function also evaluates $h(x^*)$ and $h^{'}(x^*)$ and checks if $w \leq e^{h(x^*)-u_k(x^*)}$. If so, x^* is accepted and added to the vector of final samples. If not, x^* is rejected. If the function has to evaluate $h(x^*)$ and $h^{'}(x^*)$ during sampling, the *update_needed* flag is turned on and an updating procedure is performed before sampling another point in the next iteration. During updating, x^* is added to the vector T_k and the vector is sorted; h and $h^{'}$ are updated by adding $h(x^*)$ and $h^{'}(x^*)$ to these vectors in the correct places; z and denominator of s_k are updated by making changes only around the neighborhood of x^*; and finally, k is incremented by 1.

Our main function, which the user can use to sample from any log-concave distribution, relies on a number of secondary functions that live in *auxiliary.R*. These functions are defined outside the main function for sake of keeping the main **R** script as short and easy to read as possible. We also tried to make all of our functions modular so that every function is responsible for a clear and specialized task.

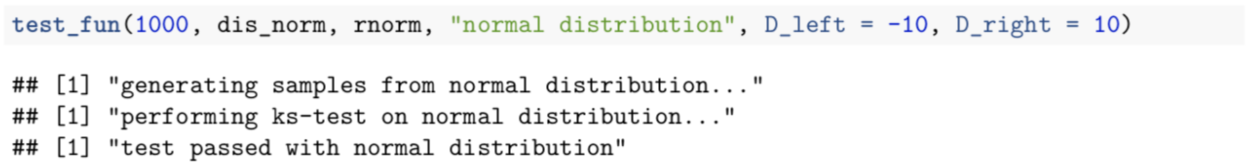
3 Sampling Results

A set of Kolmogorov–Smirnov tests are conducted to evaluate the overall performance of the function. The p-value is set to be 0.05. We carried out tests on logistic, normal, uniform, laplace and gamma distributions, and all the tests are passed consistently (see figures below). (However, due to the inherent randomness in the algorithm, it is possible that sometimes a test fails.)



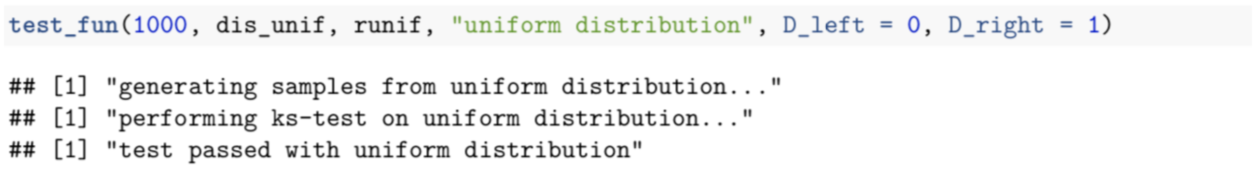
```
test_fun(1000, dis_logistic, rlogis, "logistic distribution", D_left = -10, D_right = 10)
```

```
## [1] "generating samples from logistic distribution..."
## [1] "performing ks-test on logistic distribution..."
## [1] "test passed with logistic distribution"
```



```
test_fun(1000, dis_norm, rnorm, "normal distribution", D_left = -10, D_right = 10)
```

```
## [1] "generating samples from normal distribution..."
## [1] "performing ks-test on normal distribution..."
## [1] "test passed with normal distribution"
```



```
test_fun(1000, dis_unif, runif, "uniform distribution", D_left = 0, D_right = 1)
```

```
## [1] "generating samples from uniform distribution..."
## [1] "performing ks-test on uniform distribution..."
## [1] "test passed with uniform distribution"
```

```r
test_fun(1000, dis_laplace, rlaplace, "laplace distribution", D_left = -10, D_right = 10)
```

```
## [1] "generating samples from laplace distribution..."
## [1] "performing ks-test on laplace distribution..."
## [1] "test passed with laplace distribution"
```

```r
test_fun(1000, dis_gamma, r_gamma, "gamma distribution", D_left = 0.01, D_right = Inf)
```

```
## [1] "generating samples from gamma distribution..."
## [1] "performing ks-test on gamma distribution..."
## [1] "test passed with gamma distribution"
```

```r
ars(1000, g = dis_nc, D_left = -10, D_right = 10)
```

```
## Error: Input is not a log-concave function
```

Appendix

Process to develop the R Package

To develop the **R** package we followed the steps described below in **Algorithm 1**.

Algorithm 1 Generate R Package

```
Result: R Package
  Initialize libraries
    → library(”devtools”)
    → library(roxygen2)
  Generate Package Folder
    → setwd(”∼/repos/”)
    → create(”ars”)
  Move the functions to the R folder in the ars package
  Move the tests to the tests/testthat folder in the ars package
  Add comments using roxygen
  Generate package
    → document()
```

To generate the description of the function that would be accessible to the user when typing ?*ars* we used the library *roxygen*2. This library allows the use of a simple written format, that needs to be included at the beginning of each function, and automatically generates the correct format to include the help functionality. To produce the *roxygen*2 comments we followed the following format:

- #’ Title
- #’

- #' Description
- #'
- #' @param description of first parameter
- #' ↓
- #' @param description of last parameter
- #'
- #' @return description of the expected return from the function
- #'
- #' @examples
- #' Example use of the function
- #'
- #' @export This line is followed by the function written in R

Contributions of group members

Ugur was responsible for coding the first draft of the algorithm.

The group members then revised the algorithm.

Shubei was responsible for adding tests and finalizing the code.

Ramon was responsible for creating a fully functioning package.