ARS Explanation

Malvika Rajeev, Yihuan Song, RJ Lee

12/2/2018

Github account for the final version of the project

Final version of the project resides in RJ's Github account. Github user name: rokjunlee

Explanation

Here are some assumptions that we took for granted. We assume that the the user will input valid density with correct minimum and maximum values of the domain. For example, if the input density is the standard normal density, then the minimum and maximum values will be `-Inf` and `Inf`, respectively.

We also started with assuming that the starting points (2) will be provided, just to make computations easier. `ars()` will make sure that the two given starting points are on the opposite sides of the maximum of the input (log) function by checking if the product of the first derivative evaluations at the two starting points are negative. Later, we relaxed those conditions to make the function work without starting points input. The mechanism for choosing two starting points will be described shortly.

The very first main task that our `ars()` needs to be able to accomplish is to transform the **input function** f **into** g, which is equal to $cf(x)$ for some constant c. We created function so that c equals $\frac{1}{\int_D f(x)}$ to make sure that function $g(x)$ becomes a sensible density function that integrates to one over its support.

When the user does not provide starting points, `ars()` function will sample **two starting points** using `starting_x1()` function. This function takes the input function, minimum and maximum values of the domain. If both minimum and minimum values are finite, then the function will locate the 'x-coordinate', at which the maximum of the density function occurs. Then we create a vector `c(minimum_x, 'x-coordinate', maximum_x)` and then designate the 49 percentile and 51 percentile as the first `x_1` and `x_2` points to start the adaptive rejection sampling. If however, either minimum or maximum (or both) is non-finite value, then minimum and maximum will be replaced with -100 and 100, respectively. Then the procedure involving 49 and 51 percentile as above are carried out to choose the first `x_1` and `x_2` points to start the adaptive rejection sampling.

[[EXPLANATION ON U, L, Z, AND SAMPLING FUNCTION SHOULD GO HERE]]

Next step is to determine if the input function is **logarithmically concave or not**. The function should not proceed to the next task if the input function is not even logarithmically concave. To verify log-concavity, we first created an auxiliary function that simply puts `log()` around the input function. Then the second derivative at numerious points in between the maximum and minimum (if they are given) were computed numerically. In order to numerically evaluate second derivative of the log-function, the following formula was used:

$$f''(x) = lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x+h)}{h^2}$$

, which is also known as the second symmetric derivative [https://en.wikipedia.org/wiki/Second_derivative].

After second derivatives were evaluated at several points in the domain, the sign of the largest evaluated value was checked. If the sign is negative, then we accept that the input function is logarithmically concave since all the other evaluated values must also be negative. Otherwise, we stop the function from moving on to the next task.

To calculate the linear lower and upperhull , the R function `findInterval()` was used, which helped determine which interval the input 'x' belonged to, and therefore to calculate the upper/lower hull corresponding only to that interval.

The basic design of the function was synchronous with the general pattern described in the paper. One challenge was sampling from the piecewise exponential density. But since the cdf of the upperhull (being linear) was calculable analytically, the inverse cdf method was useful in this case.

There were some challenges we ran into, like the calculation of exponents tending to infinity. For that, we tried to add sanity tests for input arguments.

For each point(named xstar) randomly generalized using the s_k_sample function, we use the squeezing and rejection test as methods of rejecting or accepting. Specifically, if the xstar passes the squeezing test formed by the upper and lower hull functions, then it is accepted at the squeezing step, and will thus will not go through the rejection test. Otherwise, if the xstar failed to go through the squeezing test, it will proceed to the rejection test formed by the h density function and the upper hull function. If xstar passes the rejection test, it will be accepted; otherwise, xstar will be rejected. Additionally, if xstar failed to pass the sqeezing test, and once h(x) and dh(x) are evaluated, we will update the starting point vector by adding xstar to the vector of starting points, and sort the starting points again to ensure that the starting points are in ascending order.

The while loop in the ars function will allow us to repeatedly sample xstar using the s_k_sample function, until we have an xstar that is accepted, and the accepted xstar will be added as one point in our final sample. We will continue this process as a for loop until we have n accepted points sampled.

Testings

To test the package, we created the test.R file in located in the tests directory of the package. To implement tests, we sourced the ars function and defined all the functions we used in the ars function. Using `test_that()`, we first tested all the Auxiliary functions (such as `numeric_first_d`, `numeric_sec_deri`, `starting_x1`) and functions defined inside the ars function (such as logconcav_check, u_func, l_func, g_func, h, and dh) to check if these function outputs are of reasonable types and values. Then we tested the primary `ars` function for different distributions (such as dnorm, dgamma, and dbeta) to see if the `ars` function samples correctly, including cases where user does not provide the starting points. Lastly, we tested to see that if the input function is not log-concave, we could catch such cases and quit the sampling process as the sampling proceeds.

Specific Contributions of each team member

Although we divided the task as below, we all contributed to all the tasks to some extent by checking for errors and improving the codes.

Malvika Rajeev

- vectorizing computation of Zjs,
- computing the upper linear hull,
- computing the cdf,
- sampling from the exponential density.

Yihuan Song

- The basic structure of the ars function (for loop and while loop)
- The functions of h(x), h'(x), uk(x), lk(x), sk(x), and finding zj's
- The squeezing and rejection tests

- Input validation: Check if the input is reasonable
- Check the log-concavity for starting points
- Testing the primary ars function and the functions defined in the ars function

RJ Lee

- g_func that transforms f into g as described in the paper
- Function that checks for log-concavity of the input function by evaluating second derivative numerically
- Function that can join tangent segments
  - Finding tangent lines
  - Simple linear models to find endpoints of each tangent lines
- Function that chooses the starting point: X_1 and X_k (k=2)

Sample Results

The following is a correctly working output where we have 1000 samples, the input function is the normal density with mean 2 and unit standard deviation, and starting points are $(-2, 3)$, ie, `ars(1000, f = function(x) dnorm(x, 2), -Inf, Inf, c(-2, 3))`.

The `ars()` automatically creates a density histogram using output of the `ars()` function, and then on top of the histogram the true density curve is drawn in yellow dotted line. Fortunately we observe similarity between the density histogram and the true density curve.

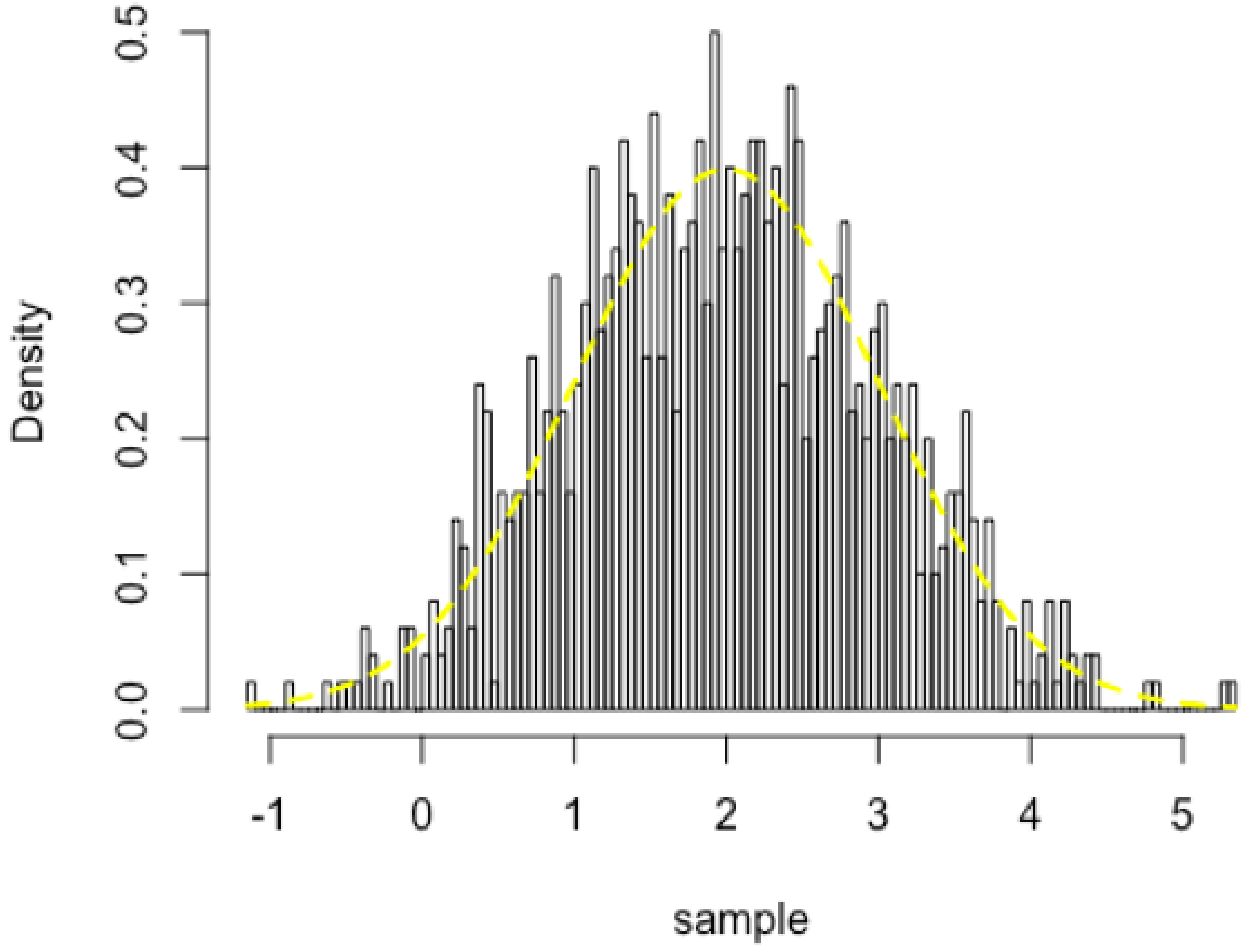

Figure 1: sample result 1