Final Project 2018 - STAT243

Bunker, JD; Bui, Anh; Lavrentiadis, Grigorios

December 10, 2018

I. Purpose

This project aims to apply the adaptive rejection sampling method described in Gilks et al. (1992). The main function `ars()` returns a sample based on any (scaled) univariate log-concave probability density function (pdf), $g(x)$. In nonadaptive rejection sampling, the envelope and squeezing functions, $g_l(x)$ and $g_u(x)$, are determined in advance. Under the adaptive rejection sampling framework, these functions are instead updated iteratively as points are sampled. The `ars()` function will (1) initialize the starting abscissae, and iteratively (2) sample points x^* from $g_u(x)$ and (3) accept / reject x^* and accordingly update the abscissae as discussed in the paper. Note that steps (2) and (3) are repeated until the user-entered number of points (`nsamp`) have been accepted.

II. Contributions of members

Each team member was primarily responsible for one of the three steps mentioned in Gilks et al. (1992): (1) the initialization step, (2) the sampling step, (3) the updating step. Summary of contributions:

- Bunker, JD: Primarily responsible for the initialization step, which sets the starting abscissae. In the main `ars()` function, the user can provide these points. Otherwise, the starting points are automatically selected. Other contributions include improving the computational efficiency of the `UpdateAccept()` function, editing the report, writing the documentation, testing the log-concavity of the given function, and miscellaneous code optimization.
- Lavrentiadis, Grigorios: Primarily responsible for the sampling step, which generates the value x^* from a piece-wise exponential distribution, $s_k(x)$. Other contributions include assembling the auxiliary functions into the comprehensive `ars()` function, testing the log-concavity of the given function, and miscellaneous code optimization.
- Bui, Anh: Primarily responsible for the updating step, which decides whether x^* is accepted or rejected in the final sample. In addition, the envelope and squeezing functions are updated accordingly. Other contributions include writing the report and implementing the formal testing.

III. Theoretical background for rejection sampling

Assume $f(x)$ is a univariate log-concave probability density function, and that $g(x)$ is a scaled version of $f(x)$. We define envelope function $g_u(x)$ and the squeezing function $g_l(x)$ about $g(x)$. We iteratively sample a point x^* from $g_u(x)$ and independently sample a value y from $Y \sim Unif(0, g_u(x))$.

We reject x^* if

$$y \leq g_l(x^*)$$

Instead of choosing a y from $Y \sim Unif(0, g_u(x))$, we could equivalently choose a w from a $Unif(0, 1)$ distribution, as detailed within the paper. In which case, the comparison becomes

$$w \leq \frac{g_l(x^*)}{g_u(x^*)}$$

In non-adaptive rejection sampling, the $g_l(x)$ and $g_u(x)$ functions are invariant throughout the sampling process. However, in adaptive rejection sampling and within the main `ars()` function, these functions are updated iteratively. The paper gives the algorithm of adaptive rejection sampling when working with the log of $g(x)$, $g_l(x)$, and $g_u(x)$. See the *Auxiliary functions* section below for our specific coding implementation.

IV. Auxiliary functions

1. Initialization functions: Determine starting points

If the user specifies the starting points, the function `CheckXStart()` tests that these points satisfy the requirements (e.g., if the sampling distribution is unbounded) and then creates the arrays that are used for the adaptive sampling These requirements of the starting points are:

- If the sampling distribution is unbounded to the left (e.g., $-\infty$ is the lower limit), the derivative of the smallest sampling point must be positive
- If the sampling distribution is unbounded to the right (e.g., $+\infty$ is the upper limit), the derivative of the largest sampling point must be negative

If the user doesn't specify starting points, the `SampleXStart()` function is used to sample points, searching for points that satisfy the previously mentioned requirements. However, finding such points is not guaranteed. At the first stage, `SampleXStart()` samples starting points out of a standard normal distribution. If the initial points don't satisfy the requirements, it truncates the lower and/or upper bound of the normal sampling distribution based on the sign of the first derivative at some point x^*. For example, if it is required to have a point with a positive first derivative (e.g., the sampling distribution is unbounded to left), yet the first derivative at the current point x^* is negative, the upper bound of the truncated normal distribution is updated to x^*. Thus, the next point will be sampled out of the $(-\infty, x^*]$ range. Since $f(x)$ is log-concave, any sampling points with a positive derivative will necessarily be to the left of x^*.

2. Sampling functions: Generate x* from piece-wise exponential distribution

The `SamplePieceExp()` function draws samples x^* out of a piece-wise exponential distribution using the inverse sampling approach. Before we determine the x^*, a P_{inv} sample is drawn from a 0 to 1 uniform distribution that corresponds to the cumulative probability of the random sample x^*. Let $X = \{x_i; i = 1, \cdots, k\}$ represent the k abscissae where $h(x_i)$ and $h'(x_i)$ have been evaluated. Also, let each $z_j (j \in \{0, 1, \cdots, k\})$ represent the intersection of the tangent lines between consecutive x_i, along with the lower and upper bound of the domain of $g(x)$. Then, we define k bins representing the intervals between each pair of adjacent z_j. To find the bin at which x^* belongs, P_{inv} is compared with the cumulative probability of each bin i, $P_{cum,i}$ $(i \in \{1, 2, \cdots, k\})$. Specifically, x^* belongs to the bin with the smallest cumulative probability ($P_{cum,i}$) that is larger than P_{inv}. Once we've located the correct bin, x^* is estimated by solving the following cumulative probability equation, where z_0 is the left bound of the distribution and P_j is the probability of bin j.

$$P_{inv} = \int_{z_0}^{x^*} s(x)dx = \sum_{j=1}^{i} P_j + \int_{z_i}^{x^*} s(x)dx$$

Let DP be equal to the probability $P(z_i > x > x^*)$, where z_i is the lower bound of the bin i in which x^* belongs. See the formula for DP below.

$$DP = \int_{z_i}^{x^*} e^{h(x_j)+(x-x_j)h'(x_j)}dx$$

To simplify this equation, we define: $a = h(x_j) - x_j h'(x_j)$ and $b = h'(x_j)$. From this equation, x^* is equal to:

$$x^* = \frac{1}{b} log(DP\ b\ e^{-a} + e^{b\ z_i})$$

3. Updating functions: Testing x* and updating

Algorithm

Inputs:

- $w \sim Unif(0, 1))$
- $l_k(x^*) = log(g_l(x^*))$
- $u_k(x^*) = log(g_u(x^*))$
- $h(x^*) = log(g(x^*))$
- $s_k(x) = \frac{\exp u_k(x)}{\int_D \exp u_k(x')dx'} = \frac{g_u(x)}{\int_D g_u(x')dx'}$

The lower bound of $h(x)$ is $l_k(x)$, which connects the values of function $h(x)$ between pairs of adjacent abscissae. The function $l_k(x)$ between two consecutive abscissae x_j and x_{j+1} is defined as

$$l_k(x) = \frac{(x_{j+1} - x)h(x_j) + (x - x_j)h(x_{j+1})}{x_{j+1} - x_j}$$

And $h'(x)$ is

$$h'(x) = \frac{d}{dx} log(g(x)) = \frac{g'(x)}{g(x)}$$

The intersection of the tangents at x_j and x_{j+1} is

$$z_j = \frac{h(x_{j+1}) - h(x_j) - x_{j+1}h'(x_{j+1}) + x_j h'(x_j)}{h'(x_j) - h'(x_{j+1})}$$

Then for x between z_{j-1} and z_j

$$u_k(x) = h(x_j) + (x - x_j)h'(x_j)$$

In the code, variable `X` is a vector containing all abscissae x_i. Once more, `H` is a vector of the realized function $h(x_i)$ pertaining to each abscissae x_i. Similarly, `H_prime` represents the vector of $h'(x_i)$. That is, `H_prime` is the realized first derivative of function $h(x_i)$ at abscissae x_i. Finally, `Z` contains the intersections of tangent lines for the abscissae, in addition to the lower and upper bounds of $g(x)$.

Step 1: If $w < exp(l_k(x^*) - u_k(x^*))$, either

- Accept x^* when the condition is satisfied. Draw another x^* from $s_k(x)$
- Reject x^* when the condition is not satisfied.

Step 2: These two procedures can be done in parallel.

- If $h(x^*)$ and $h'(x^*)$ were evaluated, update $l_k(x), u_k(x), s_k(x)$. X now includes x^* as an element.
- Accept x^* if $w < exp(h(x^*) - u_k(x^*))$. Otherwise, reject.

The $l_k(x)$ and $u_k(x)$ functions are generated based on the vector variables `H`, `H_prime`, and `Z`. As mentioned earlier, the first two variables represent the values of $h(x)$, $h'(x)$ for each point in `X`. When a new x^* is accepted, we append the corresponding $h(x^*)$ and $h'(x^*)$ to vectors `H` and `H_prime`, respectively. In the vector `Z` of z_j values, we append a new value corresponding to x^*. In cases where x^* is between existing values in `X`, we also modify an existing value of `Z`.

Depending on the value that x^* takes, the function `UpdateAccept()` behaves slightly differently. For example, if x^* is out of the domain of `X` (e.g., below x_1 or above x_k), then we set $l_k(x^*) = -\infty$ as mentioned in the paper. Once more, if x^* is at the minimum value $X[1]$ or maximum value $X[n]$ within `X`, then $l_k(x^*)$ will take values along the lines connecting $X[1]$ and $X[2]$ or connecting $X[n-1]$ and $X[n]$, respectively. Once more, vector `Z` is also generated for the case when $h(x)$ is a linear function of x (e.g., the exponential distribution). In this case, we take the average of corresponding points x_i and x_{i+1} within vector `X`.

V. Testing

1. Formal tests for ars function

The `ars()` function passes the test for the following distributions in the testing phase using the Kolmogorov-Smirnov Test:

- Normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$
- Normal distribution with mean $\mu = 7$ and standard deviation $\sigma = 2$
- $Beta(1, 3)$ distribution
- $Gamma(2, 3)$ distribution

2. Tests for auxiliary functions

a. Test CalcProbBin function

The `CalcProbBin()` function generates the cumulative probability based on elements in vector `Z` (i.e. the intersections of tangent lines between abscissae). Based on vector `Z` under the $N(0, 1)$ density, we test whether the calculated cumulative probabilities are (nearly) equal to the cumulative probability using `pnorm(Z)`.

b. Test UpdateAccept function

The `UpdateAccept()` function decides whether to accept x^* or not based on designed conditions. We test this function by checking if x^* is included in `X`. In this case, x^* is accepted and included in the `x_accept` vector. In addition, there is no update here, e.g., `H` and `H_prime` stay the same.

VI. Repository information

The repository for the *ars* package can be located on GitHub under 'glavrentiadis/stat243_project/ars'. Specifically, to install the package, examine the contents, and run the tests please use the following code:

```
library(testthat)
library(devtools)
devtools::install_github('glavrentiadis/stat243_project/ars')
library(ars)

help(ars) #View documentation
test_package('ars') #Run tests
```