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()* generates a sample from any univariate log-concave probability density function. 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.

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 *Update_accept()* function, testing the log-concavity of the given function, editing the report, and writing the documentation.
- Lavrentiadis, Grigorios: Primarily responsible for the sampling step, which generates the value x^* from a piecewise exponential distribution, $s_k(x)$. Other contributions include assembling the auxiliary functions into the comprehensive *ars()* 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 $W \sim Unif(0, 1)$ distribution, like within the paper. In which case, the comparison becomes

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

In non-adaptive sampling, the $g_l(x)$ and $g_u(x)$ functions are invariant throughout the sampling process. However, in adaptive sampling, 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. Initial function

2. Generate x* from piecewise exponential probabilities

The `SamplePieceExp` function draws samples x^* out of a piece-wise exponential distribution using the inverse sampling approach. Initially a P_{inv} sample is drawn from a 0 to 1 uniform distribution that corresponds to the cumulative probability of the random sample x^* To find the bin at which x^* belongs, P_{inv} is compared with the cumulative probability of each bin i, $P_{cum}i$. x^* belongs to the bin with the smallest cumulative probability ($P_{cum}i$) larger than P_{inv}. 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.

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

DP is 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^* equals to:

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

3. Update_accept() function

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 on abscissaes. The function of $l_k(x)$ between two consecutive abscissaes x_j and x_{j+1} is

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

Let X be the domain of abscissaes, H be the domain of the realized function H at abscissaes, H_prime be the domain of the realized first derivative of function H at abscissaes, Z be the domain of intersection of tangent lines at abscissaes.

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

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

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

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

- 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.

- Evaluate $h(x^*), h'(x^*)$. Update $l_k(x), u_k(x), s_k(x)$. X 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 from the vector variables H, H_prime, and Z. The first two variables represent the values of $h(x)$, $h'(x)$ for each value in X. When a new x^* is accepted, we add the corresponding $h(x^*)$ and $h'(x^*)$ to H and H_prime. For the vector of z values, variable Z, 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.

Multiple testing are generated based on values that x^* can take. For example, if x^* is out of the domain of X, $l_k(x^*) = -Inf$. If x^* is at the minimum value $X[1]$ and maximum value $X[n]$ in the domain of X, $l_k(x^*)$ will take the values on the lines connecting $X[1]$ and $X[2]$, and connecting $X[n-1]$ and $X[n]$ respectively. Vector Z is also generated for the case when $H(x)$ is a linear function of x (for example, the exponential distribution).

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 = 0 and standard deviation = 1
- Normal distribution with mean = 7 and standard deviation = 2
- Beta(1,3) distribution
- Gamma(2,3) distribution
- Exponential(5) 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 intersection of the tangent lines of abscissaes).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 that if $x*$ is included in X, $x*$ is accepted and included in the x_accept vector, as well as there is no update, i.e. H and H_prime stay the same.