Stat 243: Building an adaptive rejection sampler

James Bladen, Lisa Felberg, Siwei Tu, Hsin-Wei Tsao

December 13, 2013

1 Introduction

(i)

Adaptive Rejection sampling was first introduced in 1992 by Gilks and Wild. It was proposed as an alternative to vanilla rejection sampling for functions that are difficult to evaluate multiple times. The objective of this method is to sample from a "difficult" distribution by representing it with tangent lines and evaluating the actual function as few times as possible. The basic algorithm is as follows:

- Take the log of the function to sample from, $h(x) = log(f(x))$

- Select two starting points, x_1 and x_2 that are to the right and to the left of the function's maximum, respectively

- Sample two random numbers, one from the uniform random number distribution, URN, and one from the normalized exponential of the upper bounded line, x^*

- If $URN \leq exp(l(x*) - u(x*))$, accept and add x^* to vector of x's

- Else if $URN \leq exp(h(x*) - u(x*))$, accept and add x^* to vector of x's

- Else reject

- Continue to sample numbers and testing procedure until a desired number of samples is obtained

The image below is a graphical representation of the algorithm.

Figure 1: Algorithm of ARS method

1
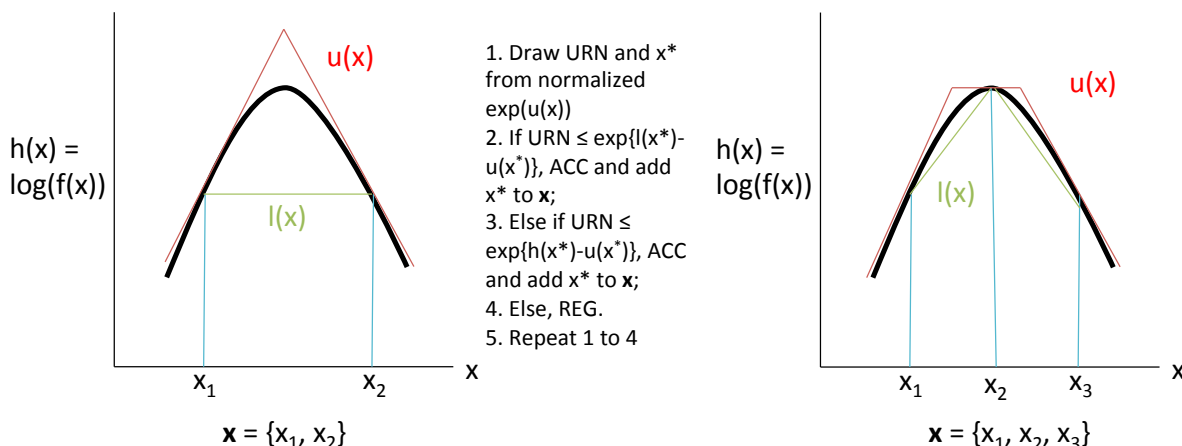
2   Code Structure

We decided to utilize the S4 class as a format for our code. It was chosen because of the formal structure and modularity. These properties also made it easy to test and create in a very piecewise manner. The general structure to the code is a follows: A wrapper function a_r_s runs the sampling method. The program is run as follows:

$$ars \leftarrow a\_r\_s(\mathbf{n} = \{\# \, of \, samps \, desired\}, \mathbf{f\_x} = \{f(x) \, to \, sample\}, \mathbf{bounds} = \{bds \, of \, f(x)\, \})$$

This will return a *Cadapt_reject_sample* class which contains sampled points in *ars@samples*. The class itself contains all the methods and variables required to perform adaptive rejection sampling.

First, two points are chosen as starting x_1 and x_2 values. Next, x^* values are drawn from the corresponding $u(x)$ distribution and the sampling criteria is tested. This is repeated until the desired number of samples is obtained. Brief descriptions of the most important methods are given below. We've included more detailed source documentation as part of our solution as well (see *a_r_sManual.pdf*).

gen_x method

In this method, we generate two initial points i.e. x_1, x_2 according to user's input of bounds.
There are 4 possible cases:

1. If the target function is bounded on both sides, then we take 2 bounds as x_1, x_2.

2. If the target function is bounded only on left side, we take x_1 as the left bound, and find an x on the right side of x_1 s.t. $h'(x_1)h'(x) < 0$, then take x as x_2.

3. If the target function is bounded only on right side, we take x_1 as the right bound, and find an x on the left side of x_1 s.t. $h'(x_1)h'(x) < 0$, then take x as x_2.

4. If the target function is unbounded on both sides, we first set $x_1 = 0$. If $h'(0) > 0$ then find an x on the right side of x_1 s.t. $h'(x) < 0$, then take x as x_2. Similary, if $h'(0) < 0$ then find an x on the left side of x_1 s.t. $h'(x) > 0$, then take x as x_2. In addition, if $h'(0) = 0$ which means $h(x)$ is symmetric to $x = 0$, then we set $(-\frac{1}{2}, \frac{1}{2})$ to be 2 starting value.

Note that $h(x) = log(f(x))$ which is concave. So for any x_1, x_2, if $h'(x_1)h'(x_2) < 0$, x_1 and x_2 would be on the different side of maximum of $h(x)$. And with the options of 2 to 4, we save x, $h(x)$ and $h'(x)$ for all x's we computed in the process for later use.

ev_h method

In this method, we calculate the values of $h(x)$ and $h'(x)$ for any x with $genD(\,)$ in package *numDeriv*.

s_x method

In this method, we calculate $s_k(x)$ by normalizing $e^{u_k(x)}$. To normalize the $e^{u_k(x)}$, we integrate each piece of $u(x)$ with following algorithm:
First note that $u(x)$ is a piecewise defined function. It is linear in each piece of interval. Actually for

$x \in [z_{j-1}, z_j]$, $u(x) = ax + b$, where $a = h'(x_j)$ and $b = h(x_j) - x_j h'(x_j)$.
So for $x \in [z_{j-1}, z_j]$,

$$\int_{z_{j-1}}^{z_j} e^{u(x)} dx = \frac{1}{h'(x_j)} exp(h'(x_j)x + h(x_j) - x_j h'(x_j)) \Big|_{z_{j-1}}^{z_j},$$

and

$$\int_D e^{u_k(x')} dx' = \sum_j \frac{1}{h'(x_j)} exp(h'(x_j)x + h(x_j) - x_j h'(x_j)) \Big|_{z_{j-1}}^{z_j}.$$

In the special case of $a = 0$ which means that for $x \in [z_{j-1}, z_j]$, $u(x) = b$ where $b = h(x_j)$.
Then

$$\int_{z_{j-1}}^{z_j} e^{u(x)} dx = e^b x_j \Big|_{z_{j-1}}^{z_j},$$

and,

$$\int_D e^{u_k(x')} dx' = \sum_j e^b x_j \Big|_{z_{j-1}}^{z_j}.$$

sampling method

In this method, we sample a new x^* from the $s_k(x)$ by using inverse CDF of $s_k(x)$ with a random value $u \sim unif(0,1)$. Moreover, the inverse CDF algorithm we use is as follow:
Once we decided which interval x^* falls in, we want to sample from $\frac{e^{u(x)}}{\int e^{u(x)} dx}$. Suppose $x^* \in [z_{j-1}, z_j]$, then $u(x) = ax + b$, where $a = h'(x_j)(\neq 0)$ and $b = h(x_j) - x_j h'(x_j)$.
Let

$$C = \int_{z_{j-1}}^{z_j} e^{u(x)} dx,$$

then the pdf is

$$\frac{e^{ax+b}}{C}.$$

So the CDF

$$F'(x) = \int_{z_{j-1}}^{x'} \frac{e^{ax+b}}{C} dx = \frac{1}{ca} e^{ax+b} \Big|_{z_{j-1}}^{x'} = \frac{ca}{e^b}(e^{ax'} - e^{az_{j-1}}).$$

Then the inverse of $F(x')$ will be

$$\frac{\log\left(\frac{ac}{e^b} + e^{az_{j-1}}\right)}{a}.$$

In the special case of $a = 0$ which means that for $x^* \in [z_{j-1}, z_j]$, $u(x) = b$ where $b = h(x_j)$.
Then the pdf will be be $\frac{e^b}{c}x$ which is an uniform distribution in $[z_{j-1}, z_j]$. In this situation, we just sample x^* from $unif \sim (z_{j-1}, z_j)$.

upper method

In this method, we calculate the $u_k(x)$ for x^* which we sample from the sampling method.

lower method

In this method, we calculate the $l_k(x)$ for x^* which we sample from the sampling method.

update method

In this method, we test the sample element x^* from sampling method. Once it is accepted, we add the x^* into the set of outputs.

3 Testing

For testing, we rigorously tested each method to ensure it was functioning properly. To test the overall function, we sampled over a few well known distributions with different bounds, including: normal, gamma, truncated normal. The results are shown below.

```
ars_class <- new( "Cadapt_reject_sample", n=100, f_x = function(x){(1/
    sqrt(2*pi)*exp((-(x-0)^2)/2))}, bounds=c(-Inf, Inf) )

ars_class<-gen_x(ars_class)
ars_class@x
ars_class@h_at_x
ars_class@hprime_at_x


ars_class<-s_x(ars_class)
ars_class@z
ars_class@mat_sorted
ars_class@piecewise_integration
ars_class@normalized_factor
ars_class@weights

ars_class<-sampling(ars_class)
ars_class@samples


ars_class<-update(ars_class)
ars_class@x
ars_class@h_at_x
ars_class@hprime_at_x
ars_class@output
while(length(ars_class@h_at_x)<15){
ars_class<-s_x(ars_class)
ars_class<-sampling(ars_class)
ars_class<-update(ars_class)
length(ars_class@h_at_x)
length(ars_class@output)
}
```

4 Contributions

Everyone in the group contributed relatively equally to each aspect of the project. Some efforts were more focused as follows:

Code writing

General structure: Lisa

Methods: James, Siwei, Hsin-Wei

Code testing

Methods testing: James, Siwei

Overall function tests: Lisa, Hsin-Wei

Documentation

Manual creation: Lisa

Project write-up: Hsin-Wei, Lisa