isid/ms/2005/09
September 5, 2005
http://www.isid.ac.in/~statmath/eprints

On Adaptive Rejection Sampling

Rajeeva L. Karandikar

Indian Statistical Institute, Delhi Centre
7, SJSS Marg, New Delhi–110 016, India

On Adaptive Rejection Sampling

Rajeeva L. Karandikar
Indian Statistical Institute
7, S. J. S. Sansawal Marg
New Delhi 110 016, INDIA
rlk@isid.ac.in

October 27, 2005

Abstract

Adaptive Rejection Algorithm (ARS) is an algorithm to generate samples from a log-concave density. The algorithm, due to Gilks [1], is very efficient. As in MCMC, this requires that the user specify the density upto a normalising constant. However, unlike in MCMC, this algorithm produces i.i.d. samples. The algorithm requires the user to also give a set of points $x_0 < x_1 < \ldots < x_M$ such that the interval (x_0, x_M) covers most of the probability. In this article, we propose an extension, which does not require the user to specify the points $x_0 < x_1 < \ldots < x_M$. This version only requires the user to specify the density (upto a constant). Thus, it can be used in Gibbs sampler if all the full conditional distributions have log-concave densities.

1 Adaptive Rejection Algorithm

Let f be a *log-concave* density on $\mathbb{R}$. Adaptive Rejection Algorithm (ARS) is an algorithm to generate i.i.d. samples from the density f. The algorithm, due to Gilks [1] (also see [2]) uses log-concavity to construct upper and lower envelopes for the target density (based on an initial set of points specified) and one can use the upper envelope as the proposal density to perform the accept-reject step of the Rejection sampling algorithm. Moreover, we can update the envelopes using the chosen sample, taking into account the "gap" between the lower envolope and the function.

For $u < v$, let $\mathbb{L}_{uv}$ (*i.e.* $y = \mathbb{L}_{uv}(x)$) be the line passing through $(u, \log f(u))$ and $(v, \log f(v))$. Then for $x \in (u, v)$, the curve $\log f(x)$ lies below the line $\mathbb{L}_{uv}$ and for $x \notin (a, b)$ lies above $\mathbb{L}_{uv}$. Let $\mathbb{S} = \{x \in \mathbb{R} : \ f(x) > 0\}$.

Let

$$\{a = x_1 < x_2 < \ldots < x_M = b\}$$

be an initial set of points. Suppose that $a, b \in \mathbb{S}$ and that a is less than the $1-$ percentile point of f and b is larger than the $99-$ percentile point of f. Let $y = \mathbb{L}_{i,i+1}(x)$ denote the line passing through the points

$$(x_i, \log f(x_i)), \ (x_{i+1}, \log f(x_{i+1})).$$

Then for $x_i \le x \le x_{i+1}, \ 2 \le i \le M-2$ one has

$$\mathbb{L}_{i,i+1}(x) \le \log f(x) \le \min\{\mathbb{L}_{i-1,i}(x), \mathbb{L}_{i+1,i+2}(x)\} \tag{1.1}$$

Let $\mathbb{L}_{0,1}(x) = \mathbb{L}_{2,3}(x)$ and $\mathbb{L}_{M,M+1}(x) = \mathbb{L}_{M-2,M-1}(x)$. Then (1.1) is also valid for $x_i \le x \le x_{i+1}$, $1 \le i \le M-1$. Also, for $x \le x_1$,

$$\log f(x) \le \mathbb{L}_{1,2}(x) \tag{1.2}$$

and for $x \ge x_M$,

$$\log f(x) \le \mathbb{L}_{M-1,M}(x) \tag{1.3}$$

Thus, defining

$$\begin{aligned} g(x) &= \exp\{\min(\mathbb{L}_{i-1,i}(x), \mathbb{L}_{i+1,i+2}(x))\} \\ h(x) &= \exp\{\mathbb{L}_{i,i+1}(x)\} \end{aligned}$$

for $x_i \le x \le x_{i+1}$, $1 \le i \le M-1$, and for $x \le x_1$,

$$\begin{aligned} g(x) &= \exp\{\mathbb{L}_{1,2}(x)\} \\ h(x) &= 0 \end{aligned}$$

and for $x \ge x_M$,

$$\begin{aligned} g(x) &= \exp\{\mathbb{L}_{M-1,M}(x)\} \\ h(x) &= 0 \end{aligned}$$

we have

$$h(x) \le \log f(x) \le g(x) \ \ \forall x.$$

We assume that $\{x_i :\ 1 \le i \le M\}$ (or rather x_1, x_2, x_{M-1}, x_M) have been chosen such that the slope of $\mathbb{L}_{1,2}$ is positive, while slope of $\mathbb{L}_{M-1,M}$ is negative. Then the function $g(x)$ is integrable.

Let $K = \int_{-\infty}^{\infty} g(x)\, dx$ and let

$$p(x) = \frac{1}{K} g(x).$$

Then p is a density and by our choice, p is an envelope for f and hence we can use the Rejection Sampling algorithm to generate a sample from f. Moreover, we can (adaptively) add chosen sample points to the initial partition to improve the envelope. It has been suggested that we add a sample point x (chosen according to density p) to the partition with probability

$$\frac{g(x) - h(x)}{g(x)}$$

so if the gap between the upper envolope and lower envolope is "large", we add the point with high probability while if the gap is "small" we add the point with small probability.

Thus as we generate more and more points, the gap between upper and lower reduces, and thus the acceptence rate increases.

Observe that it is easy to generate samples from the envolope $p(x)$. For this, first note that denoting by (w_i, z_i) the point of intersection of the lines $y = \mathbb{L}_{i-1,i}(x)$ and $y = \mathbb{L}_{i+1,i+2}(x)$ for $2 \le i \le M-2$ and writing $y_i = \log f(x_i)$, $1 \le i \le M$, we can see that $\log p(x)$ is a piecewise linear function, obtained by joining

$$(x_2, y_2), (w_2, z_2), (x_3, y_3), (w_4, z_4), \ldots (x_{M-2}, y_{M-2})$$

and for $x \le x_2$, it is the line $y = \mathbb{L}_{1,2}(x)$ while for $x \ge x_M$, it is the line $y = \mathbb{L}_{M-1,M}(x)$. Thus $p(x)$ is a mixture of functions that are piecewise linear on the log -scale. Thus we can generate random samples from $p(x)$ easily.

If f is a log-concave function that is integrable, then the algorithm described above yields samples from a distribution whose density is proportional to f, *i.e.* whose density is $f_0(x)$ given by

$$f_0(x) = f(x) \Big[\int f(x) dx \Big]^{-1}.$$

In such a situtuation, f may be called the unnormalised density.

2 AARS

In this section, we will give an algorithm that would choose the initial partition automatically. Thus, the only input required from the user will be the function $f(x)$. Indeed, we would not require that the support of the density is specified separately : only thing needed is a "black box" or an "oracle" that on receiving an input x returns the value $f(x)$, which will be zero if the point x is outside the support of f.

The discussion given below (and the software implementation that comes with this article) is for the language `C++`. But it is valid for other languages as well.

Let $\phi(x)$ denote the approximate value of $f(x)$ computed by the computational algorithm in the language `C++`. Note that if $f(x) = \exp\{-x * x/2\}$, then $f(x) > 0$ for all x while $\phi(x)$ will be zero for large $|x|$. Here the numbers are represented as long double. It can be seen that $\phi(40) = 0$. In general, even when the support of the distribution corresponding to a log-concave density function f is the whole real line, limit as x converges to ∞ or $-\infty$ of $f(x)$ is zero and thus for large values of $|x|$, $\phi(x)$ will be zero.

There is another kind of approximation in the background. Each number is stored in the internal memory upto a specified number of significant binary digits. So it is possible for $u > 0$, $v > 0$ and yet u and $u + v$ have the same representation. For example, if $u = 1$ and $v = 10^{-30}$ (*i.e.* 1e-30), then it can be seen that the expression $(u == u + v)$ evaluates to true. We will write $< x >$ to denote the number stored by the system, so that with u, v as above, $< u + v >$ is also equal to 1. Indeed, given any u, for sufficiently small v, $< u + v >$ will be equal to $< u >$.

Thus, whatever be the support of the distribution, we can get two points a, b with the property that for some $\gamma > 0, \theta > 0$

$$\begin{aligned} &\phi(b) > 0,\ \ \phi(b+\gamma) = 0 \\ &\qquad\qquad \text{and} \\ &\qquad \text{either}\ \ < b > = < b + \frac{\gamma}{2} > \ \text{ or }\ < b + \frac{\gamma}{2} > = < b + \gamma >; \\ &\phi(a) > 0,\ \ \phi(a-\theta) = 0 \\ &\qquad\qquad \text{and} \\ &\qquad \text{either}\ \ < a > = < a - \frac{\theta}{2} > \ \text{ or }\ < a - \frac{\theta}{2} > = < a - \theta > . \end{aligned}$$

Thus essentially, $[a, b]$ is the support of the distribution. So the first task is to identify such a, b. Once we have done that we can then choose the required partition.

The first step is to find a point z such that $\phi(z) > 0$. We will do a sequential search as follows. For $n \geq 0$, let

$$T_n = \{(-2^{4n} + i \times 2^{-n}) :\ i = 0, 1, \ldots, 2^{5n+1}\}.$$

We will search in the sets $T_0, T_1, \ldots, T_k \ldots$ one by one, till we find z such that $\phi(z) > 0$.

Next step is to find a, b. For finding b we proceed as follows. Let $\gamma = 1$. Let $c = z$.

Step A Define $u_0 = c, u_{m+1} = u_m + 2^m\gamma$ and let m be the smallest integer such that $\phi(u_{m+1}) = 0$. Let $c = u_m$

Step B If $m > 0$, then Goto Step A. (*So when we exit Step A with $m = 0$ and go onto Step C, we would have found c such that $\phi(c) > 0$ but $\phi(c + \gamma) = 0$.*)

Step C Let $u = c$ and $v = c + \gamma$.

Step D Let $w = (u + v)/2$. If $< u >=< w >$ or $< w >=< v >$, goto Step F.

Step E If $\phi(w) > 0$ then $u = w$ else $v = w$. goto Step D.

Step F $b = u$

This algorithm gives the upper bound b. Proceeding simillarly, we can get lower bound a.

Note that in determining a, b, we have not used any additional information about $\phi(x)$. Now to get the initial partition. One of the objectives is to choose very few points in the tail of the distribution and more points in the center. At the same time, we have to allow for the possibility that the mode could be one of the end points, a or b. We proceed as follows. Let $x_0 = a$ and $x_{100} = b$. First we define

$$z_i = a + \frac{(b-a)i}{400},\ i = 0, 1, \ldots 400$$

and let M be the maximum of $\phi(z_i)$, $i = 0, 1, \ldots 200$. Let $M_1 = M/100$ and let l be the first $k \geq 1$ such that $\phi(z_k)$ is greater than M_1 and j be the last $k \leq 399$ such that $\phi(z_k)$ is greater than M_1. We put $x_{10} = z_l$ and $x_{90} = z_j$. Then,

$$x[i] = x[i-1] + (x_{10} - x_0)/10 \quad i = 1, 2, \ldots, 9;$$

$$x[i] = x[i-1] + (x_{90} - x_{10})/80 \quad i = 11, 12, \ldots, 89;$$

$$x[i] = x[i-1] + (x_{100} - x_{90})/10 \quad i = 91, 2, \ldots, 99.$$

This defines the initial partition $x_0, x_1, \ldots, x_{100}$. This choice seems to work well. Once we have the initial partition, we can proceed with ARS as described in previous section.

3 Software implementation

The algorithm described above has been implemented in `C++`. There are many numerical stability issues associated with the AARS algorithm as with ARS algorithm which need to be addressed. The `C++` programming has been done with help from Prateek Karadikar. For this implementation, the executables created using Gnu compiler (gcc) are available in a zip file at the following URL:

```
http://www.isid.ac.in/~rlk/AARS.zip
```

This zip file contains two folders, Windows and RedHat, each containing the executables.

For Windows OS (98 or XP) use the files under the directory Windows (two files: `AARS.exe` and `cygwin1.dll`). Both files should be kept in one directory, the working directory. For RedHat Linux, use the file under directory RedHat (one file `AARS`) (you will need to change permissions of the file `AARS` via "`chmod 700 AARS`").

Usage under windows :
`AARS.exe target_function`
Usage under Linux:
`./AARS target_function`

It will produce a sample of size 10000 and it will be written to a file "sample.csv". The target function (the unnormalised density) is specified as a string using the variable x and the following functions (from C++ math library) sin, cos, tan, exp, log, sqrt, asin, acos, atn, abs, floor, ceil, sinh, cosh, tanh, pow, max, min. In addition, to facilitate specifying functions two additional functions `Igt(x,a)`, `Ilt(x,b)` are available: `Igt(x,a)` is the indicator of the interval (a, ∞) and `Ilt(x,b)` is the indicator of the interval $(-\infty, b)$.
Examples of Usage (given here for windows: open "Command prompt" and type the command given below; for linux replace `AARS.exe` with `./AARS` in terminal window):

For standard normal density $f(x) = \exp\{-x^2/2\}$,
target_function="`exp(-x*x/2)`".
Usage:
`AARS.exe "exp(-x*x/2)"`

For Gamma with parameter 13 (say),
$f(x) = x^{12}\exp\{-x\}$, for $x > 0$ and zero otherwise
target_function="`Igt(x,0)*pow(x,12)*exp(-x)`".
Usage:
`AARS.exe "Igt(x,0)*pow(x,12)*exp(-x)"`

For truncated normal density
$f(x) = \exp\{-(x + 100)^2/60\}$, for $10 \leq x \leq 150$
target_function="`Igt(x,10)*Ilt(x,150)*exp(-(x+100)*(x+100)/60)`".
Usage:
`AARS.exe "Igt(x,10)*Ilt(x,150)*exp(-(x+100)*(x+100)/60)".`

It should be noted that the algorithm works even when the support of the distribution is a small set far removed from the origin, just that the search takes longer. Thus,
`AARS.exe "Igt(x,10000)*Ilt(x,10000.0001)*exp(-(x-10000)*5000)"`
also works. So the algorithm is able to find a "needle in a haystack".

References

1 Gilks, W. R. (1992) Derivative-free adaptive rejection sampling for Gibbs sampling. Bayesian

Statistics 4, (eds. Bernardo, J., Berger, J., Dawid, A. P., and Smith, A. F. M.) Oxford University Press.

2 Ripley, B. (1987) Stochastic Simulation. New York, Wiley.