Algorithm AS 287: Adaptive Rejection Sampling from Log-Concave Density Functions

P. Wild, W. R. Gilks

Applied Statistics, Volume 42, Issue 4 (1993), 701-709.

Stable URL:
http://links.jstor.org/sici?sici=0035-9254%281993%2942%3A4%3C701%3AAA2ARS%3E2.0.CO%3B2-4

Applied Statistics

Plackett, R. L. (1960) *Principles of Regression Analysis.* Oxford: Clarendon.
Reilly, P. M. and Patino-Leal, H. (1981) A Bayesian study of the error-in-variables model. *Technometrics*, **23**, 221–231.
Seber, G. A. F. and Wild, C. J. (1989) *Nonlinear Regression.* New York: Wiley.

Algorithm AS 287

Adaptive Rejection Sampling from Log-concave Density Functions

By P. Wild†

National Institute for Research and Safety in Occupational Health, Nancy, France

and W. R. Gilks

Medical Research Council Biostatistics Unit, Cambridge, UK

[Received March 1990. Final revision October 1992]

Keywords: Adaptive rejection sampling; Concave log-density; Gibbs sampling; Integration; Squeezing

Language

Fortran 77

Description and Purpose

The purpose of the algorithm is to provide an omnibus but nevertheless efficient way to sample from a distribution from the large class of concave log-densities, by using the method of adaptive rejection sampling (Gilks and Wild, 1992).

Adaptive rejection sampling makes use of the fact that a concave function is bounded above by its tangents (which define an upper hull) and is bounded below by its chords (which define a lower hull). These upper and lower hulls are used to perform rejection sampling with squeezing (see Fig. 1 and Ripley (1987)).

Our procedure is adaptive in that the function evaluations used in rejection sampling are used to update both upper and lower hulls.

Notation

Set $h(x) = \ln g(x)$, where $g(x)$ is proportional to the density $f(x)$ from which we want to sample. Let D be its domain.

Let $\{x_i;\ i=1, \ldots, n\}$ denote points at which the function $h(x)$ has already been evaluated. Let $x_{(i)}$ denote the ith-lowest element of $\{x_i\}$; for example $x_{(1)}$ is the lowest $x_{(i)}$, $x_{(2)}$ the second lowest etc. The lower hull $h_l(x)$ is defined by the chords between $(x_{(i)}, h(x_{(i)}))$ and $(x_{(i+1)}, h(x_{(i+1)}))$ (see Fig. 1).

†*Address for correspondence*: Centre de Recherche et de Formation, Institut National de Recherche et de Sécurité, Avenue de Bourgogne, BP 27, 54501 Vandoeuvre Cedex, France.

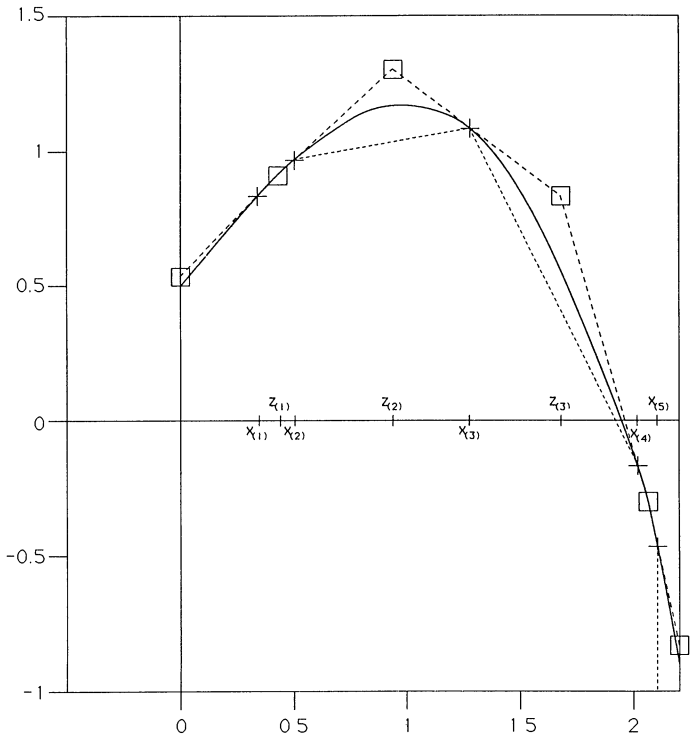

Fig. 1. Concave log-density $h(x)$ (———), bounded on the left-hand side at $x=0$ showing upper (--□--) and lower (·· + ··) hulls based on five abscissae ($x_1, x_2, \ldots, x_5$)

The upper hull $h_u(x)$ of $h(x)$ consists of the tangents at $\{x_i\}$. It is defined by the slope of the tangents $h'(x_i)$, the abscissae $\{z_{(i)}\}$ of the intersection between the tangents at $x_{(i)}$ and at $x_{(i+1)}$ and the values $h_u(z_{(i)})$ (see Fig. 1).

Algorithm

Let A be a set of points $[x_i, h(x_i), h'(x_i)]$; the tangents at these points define the upper hull h_u; the chords define the lower hull h_l (see Fig. 1).

Repeat
Sample x from $s(x)$, the normalized exponential of the upper hull
Sample u from a uniform distribution over (0, 1)
  if $u < \exp\{h_l(x) - h_u(x)\}$ *then* accept x without any function evaluation (squeezing)
  else perform the rejection test
    evaluate function h at x
    if $u < \exp\{h(x) - h_u(x)\}$ *then* accept x
    else reject x
    endif
    update lower and upper hull by adding $[x, h(x), h'(x)]$ to A, yielding closer upper and lower bounds for the subsequent rejection sampling
  endif
until the user-defined number of points have been sampled.

Starting Points

If there are m starting points, the algorithm will fail if one of the two following conditions is met:

(a) $h'(x_{(1)}) \leqslant 0$, and there is no lower bound to the domain D of $f(x)$;
(b) $h'(x_{(m)}) \geqslant 0$, and there is no upper bound to D.

In other words, if the domain D is not bounded there must be starting points on both sides of the mode of $g(x)$; if D is unbounded on the left-hand side, the lowest starting point must be left of the mode; if D is unbounded on the right-hand side, the highest starting point must be right of the mode.

Updating and Sampling Formulae

To implement this algorithm we need to be able to calculate the abscissae and ordinates of the intersection points between the tangents

$$z_{(i)} = x_{(i)} + \frac{h(x_{(i)}) - h(x_{(i+1)}) + h'(x_{(i+1)})(x_{(i+1)} - x_{(i)})}{h'(x_{(i+1)}) - h'(x_{(i)})},$$

$$h_u(z_i) = h'(x_i)(z_i - x_i) + h(x_i).$$

c_u is the normalizing factor of the exponentiated upper hull:

$$c_u = \int \exp h_u(x)\,dx = \sum_{j=0}^{n-1} \frac{1}{h'(x_{(j+1)})} \left\{ \exp h_u(z_{(j+1)}) - \exp h_u(z_{(j)}) \right\}$$

where $z_{(0)} = -\infty$ if there is no lower bound for x, $z_{(0)} =$ xlb if xlb is the lower bound of domain D, $z_{(n)} = +\infty$ if there is no upper bound for x and $z_{(n)} =$ xub if xub is the upper bound of domain D. Let us denote s_{cum} the cumulative distribution function of $s(x)$; then

$$s_{\text{cum}}(z_{(i)}) = \frac{1}{c_u} \sum_{j=0}^{i} \frac{1}{h'(x_{(j+1)})} \{ \exp h_u(z_{(j+1)}) - \exp h_u(z_{(j)}) \}.$$

To sample from $s(x)$, we sample first from a uniform distribution yielding u; we then find the largest $z_{(i)}$ such that $s_{\text{cum}}(z_{(i)})$ is lower than u. The x-value sampled from $s(x)$ is then given by

$$x = z_{(i)} + \frac{1}{h'(x_{(i+1)})} \log \left[1 + \frac{h'(x_{(i+1)})\, c_u \{ u - s_{\text{cum}}(z_{(i)}) \}}{\exp h_u(z_{(i)})} \right].$$

Discussion

The primary purpose of this algorithm is to sample efficiently from complicated log-concave densities: such densities occur often in applications of Gibbs sampling. Detailed discussion of this algorithm in connection with Gibbs sampling is given by Gilks and Wild (1992). We mention here another potential use of the algorithm.

Set $c_l = \int_D \exp h_l(x)\,dx$, i.e.

$$c_l = \sum_{i=1}^{n-1} \frac{x_{(i+1)} - x_{(i)}}{h(x_{(i+1)}) - h(x_{(i)})} \{ \exp h(x_{(i+1)}) - \exp h(x_{(i)}) \}.$$

Thus $c_l \leqslant \int_D g(x)\,dx \leqslant c_u$.

As $n \to \infty$, $s(x)$ and $t(x)$ the normalized exponential of the lower hull tend to $f(x)$. We can thus obtain an upper and a lower bound not only for the integral of $g(x)$ over D but for the whole (cumulative) distribution.

The choice of the points x_i is optimal in that they are sampled from a distribution which becomes increasingly closer to $f(x)$. Moreover, if a certain x is sampled, the probability that we need a function evaluation at this point is proportional to $\exp\{h_l(x) - h_u(x)\}$; thus the less good the approximation the more likely we are to update the lower and upper hulls at this point.

Structure

SUBROUTINE INITIAL(NS, M, EMAX, X, HX, HPX, LB, XLB, UB, XUB, IFAULT, IWV, RWV)

Formal parameters

NS	Integer	input:	the maximum number of points defining the hulls
M	Integer	input:	the number of starting points
EMAX	Real	input:	a large number for which exp(EMAX) and exp(−EMAX) can be calculated
X	Real array (*m*)	input:	the starting abscissae (in the first *m* elements)
HX	Real array (*m*)	input:	HX(*i*) = $h(x_{(i)})$, $i = 1, \ldots, m$
HPX	Real array (*m*)	input:	HPX(*i*) = $h'(x_{(i)})$, $i = 1, \ldots, m$
LB	Logical	input:	indicates whether there is a lower bound for domain *D*
XLB	Real	input:	the lower bound of *D*
UB	Logical	input:	indicates whether there is an upper bound for domain *D*
XUB	Real	input:	the upper bound of *D*
IFAULT	Integer	output:	= 0 indicates successful initialization; = 1 if there are not enough starting points; = 2 if NS is lower than *m*; = 3 if the lowest starting point is right of the mode; = 4 if the highest starting point is left of the mode; = 5 if non-concavity is detected
IWV	Integer array (NS + 7)	output:	an integer working vector
RWV	Real array (6NS + 15)	output:	a real working vector

Subroutine INITIAL takes as input the number of starting points m and the starting points $\{(X(i), HX(i), HPX(i)), i = 1, \ldots, m\}$. As output we have the index of the lowest $X(i)$, a pointer which gives the increasing sequence of $X(i)$, all the parameters defining the lower and upper hulls and a diagnostic of the starting points. All these parameters are consolidated in the working vectors as described in the comments of subroutines INITIAL and UPDATE. NS is the maximum number of function evaluations (see the section on constants later).

SUBROUTINE SAMPLE(IWV, RWV, EVAL, BETA, IFAULT)

Formal parameters

IWV	Integer array (NS+7)	output:	an integer working vector.
RWV	Real array (6NS+15)	output:	a real working vector
EVAL	Subroutine	input:	the name of the subroutine which calculates $h(x)$ and $h'(x)$ with the structure SUBROUTINE EVAL(a, b, c) where a, b and c are reals; $b=h(a)$ and $c=h'(a)$ (see below)
BETA	Real	output:	a sampled value
IFAULT	Integer	output:	=0 indicates successful sampling; =5 if non-concavity is detected; =6 if the random number generator generated 0; =7 indicates numerical instability

Subroutine SAMPLE samples one point (BETA) with the adaptive rejection sampling algorithm. As input it takes the lower and upper hull and the subroutine which performs the function evaluation. The output consists of the sampled point and the possibly updated hulls.

Failure Indications

The algorithm will fail either if it detects non-concavity or if the random number generator generates 0. Another instance of failure occurs with wrong starting points; this is detected in subroutine INITIAL, which gives the relevant parameter IFAULT. A last possibility of failure occurs when the intersection between two tangents does not fall between the points. This happens because of numerical instability in single-precision calculations.

Auxiliary Algorithms

Subroutine SAMPLE calls subroutine SPL1: the core of the program which performs the adaptive rejection sampling. To do this it calls the following subroutines:

(a) subroutine SPLHULL samples from the upper hull;
(b) function RANDOM—pseudorandom number generator AS 183 (Wichmann and Hill, 1982) modified according to AS R58 (McLeod, 1985) to avoid 0 being sampled;
(c) subroutine UPDATE performs all the operations needed to update both lower and upper hulls;
(d) subroutine INTERSECTION computes the intersection between tangents—non-concavity can be detected at this stage;
(e) function EXPON calculates an exponential without underflow error (this function can be suppressed if there is a compiler option which prevents underflow);
(f) subroutine EVAL(X, HX, HPX) computes the value of $h(x)$ and $h'(x)$ at given point x (this subroutine can be given any name as it is a parameter

of subroutine SAMPLE; it must be declared in the main program by EXTERNAL EVAL).

Constants

There are two user- and machine-dependent constants. EMAX must be lower than the largest machine-dependent constant for which both exp(EMAX) and exp(−EMAX) can be calculated. Yet EMAX determines the precision of the calculation and, if EMAX is too small, numerical problems may occur. EMAX should at least verify

$$1.0 + \exp(\text{EMAX}) = \exp(\text{EMAX}).$$

EPS = exp(−EMAX) is taken as a very small number used for various tests; EPS must be greater than 0. NS is the maximum number of points defining the hulls after which the updating is stopped; after this, the adaptive rejection sampling, changes to ordinary rejection sampling with squeezing (see Ripley (1987)).

Main Program

Before sampling, the main program must initialize NS, LB, XLB, UB, XUB, the number of starting points m and the starting points (X(i), HX(i), HPX(i), $i=1, \ldots, m$). It calls subroutine INITIAL, and the sampling proceeds by repeatedly calling subroutine SAMPLE.

Precision

In the single-precision version failure for numerical instability happens sometimes; we thus recommend the systematic use of double-precision arithmetic especially if the program is used in the context of Gibbs sampling where extreme situations are likely to happen.

To do so all the REAL declarations should be changed to DOUBLE PRECISION; likewise the function statements ZMAX, ZEXP and ZLOG should be given the corresponding definition.

Timing

The time needed to sample r values depends on the number of function evaluations needed especially if this evaluation is costly. Fig. 2 shows the mean number of function evaluations n, plus or minus one standard deviation, necessary to sample r points from a normal distribution. Each plotted point is based on 100 runs. The data from Fig. 2 give a surprisingly linear relationship ($R^2=0.999$) between $\log n$ and $\log r$ yielding approximately the following relationship:

$$n = 3r^{1/3}.$$

Very similar results were obtained for quite different distributions:

(a) $h_1(x) = -x^4/4$;
(b) $h_2(x) = \log(2x) - x^2$, $x > 0$ (Weibull);
(c) $h_3(x) = 0.3\log x + 1.7\log(1-x)$, $x \in (0, 1)$ (beta);
(d) $h_4(x) = -x - \exp(-x)$ (extreme value).

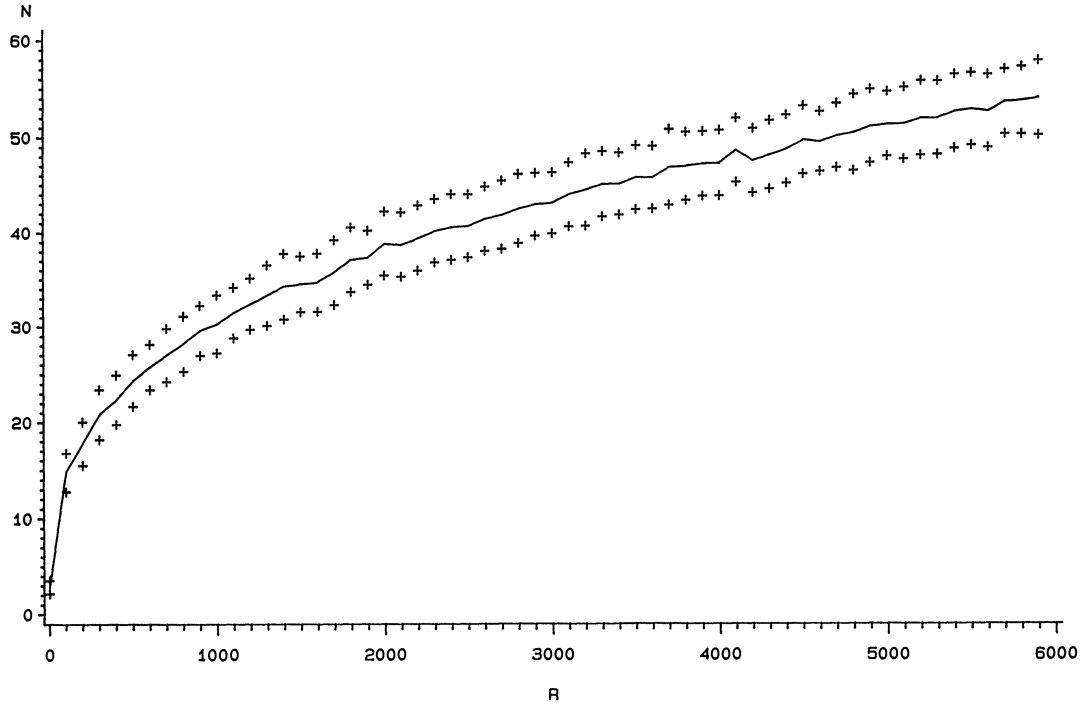

Fig. 2. Mean number n (plus or minus one standard deviation) of function evaluations necessary to obtain a sample of size r from the standard normal distribution: each point is based on 100 runs of the algorithm

A tentative proof of this relation is as follows.

The probability P_n that a sampling adds a function evaluation is

$$P_n = \int_D p_n(x)\, s(x)\, \mathrm{d}x$$

where $p_n(x)$ is the probability that the rejection test necessitates the calculation of $g(x)$ for a given x sampled from $s(x)$, the normalized exponential of the upper hull. According to the algorithm

$$p_n(x) = 1 - \frac{\exp h_l(x)}{\exp h_u(x)}.$$

Let us suppose that the x_i are in increasing order: if $x \in [x_i, z_i)$,

$$p_n(x) = 1 - \exp\left[(x - x_i)\left\{\frac{h(x_{i+1}) - h(x_i)}{x_{i+1} - x_i} - h'(x_i)\right\}\right].$$

Taking a first-order Taylor series approximation in i around x_i and a second-order Taylor series approximation for $h(x)$ around z_i gives, for $x \in [x_i, z_i)$,

$$p_n(x) \sim -\tfrac{1}{2}(x - x_i)(x_{i+1} - x_i)\, h''(z_i).$$

Similarly, for $x \in [z_i, x_{i+1})$:

TABLE 1
Number of function evaluations and mean time needed to sample 30000 values

Computer	*Mean times (s) for the following runs:*							
	$h_1(x)$		$h_2(x)$		$h_3(x)$		$h_4(x)$	
	NS = 100	*NS = 10*	*NS = 100*	*NS = 10*	*NS = 100*	*NS = 10*	*NS = 100*	*NS = 10*
PC 286 with coprocessor	204	147	209	142	202	142	212	145
HP 9000/825	22.6	15.8	23.0	15.4	22.1	15.0	22.9	15.6
SUN SPARCSTATION 1	9.7	7.7	9.9	7.6	9.6	7.6	10.3	7.7
Number of function evaluations	87.8	3556	82.8	2693	85.2	1706	91	2813

$$p_n(x) \sim -\tfrac{1}{2}(x_{i+1}-x)(x_{i+1}-x_i)\,h''(z_i).$$

Now, for large n, the probability that a further function evaluation will be required at the next sampling is approximately

$$P_n \sim \int_D p_n(x) f(x)\,dx \sim \sum_{i=0}^{n} f(z_i) \int_{x_i}^{x_{i+1}} p_n(x)\,dx = -\frac{1}{8}\sum_{i=0}^{n} (x_{i+1}-x_i)^3 h''(z_i) f(z_i).$$

Approximately, we might expect $-(x_{i+1}-x_i)^2 h''(z_i) \sim c/n^2$ for some positive constant c because $x_{i+1}-x_i$ should decrease in order $1/n$ and $(x_{i+1}-x_i)^2 h''(z_i)$ is scale free.

Therefore

$$P_n \sim \frac{c}{8n^2}\sum_{i=0}^{n} (x_{i+1}-x_i) f(z_i) \sim \frac{c}{8n^2}.$$

Then

$$\frac{\Delta r}{\Delta n} \sim \frac{1}{P_n} = \frac{8n^2}{c},$$

so $r \sim kn^3$.

When r values have been sampled, subroutine SAMPLE has been called r times, subroutine UPDATE $n-1$ times and subroutine EVAL n times. UPDATE involves the computation of two intersections and n exponentials. SAMPLE involves sampling the upper hull, two random values and one logarithm. The sampling of the upper hull necessitates one exponential, two algorithms and a linear search among n values.

This is not optimal asymptotically; if we replace the pointer structure by inverse ranks, we can do a binary search. This led to a 30% improvement for $r = 30000$, but led to a 20% loss in the context of Gibbs sampling where only one value is sampled at a time. If the function evaluation is cheap as for $h_1(x)$-$h_4(x)$, an alternative way to increase efficiency is to take small values for NS. Table 1 gives real timings based on 10 runs of the program on three configurations. These results show that, if the function evaluation is cheap, it is more efficient to use small values for NS; however, if the distributions are very complicated it is probably wiser to choose a large value for NS.

Acknowledgements

We thank the referee whose comments helped to improve greatly both the code and the text of the algorithm. We thank furthermore Professor Depaix for helpful discussions and Mrs Bertrand for her skilful typing.

References

Gilks, W. R. and Wild, P. (1992) Adaptive rejection sampling for Gibbs sampling. *Appl. Statist.*, **41**, 337–348.

McLeod, A. I. (1982) Remark AS R58—A remark on algorithm AS 183: An efficient and portable pseudo-random number generator. *Appl. Statist.*, **34**, 198–200.

Ripley, B. D. (1987) *Stochastic Simulation.* Chichester: Wiley.

Wichmann, B. A. and Hill, I. D. (1982) Algorithm AS 183: An efficient and portable pseudo-random number generator. *Appl. Statist.*, **31**, 188–190.

Correction

Correction to Algorithm AS 251: Multivariate Normal Probability Integrals with Product Correlation Structure

By Charles W. Dunnett†

McMaster University, Hamilton, Canada

[*Appl. Statist.*, **38** (1989), 564–579]

The algorithm contains an error which, for very small values of PROB, may make it impossible to exit from the loop between statements 40 and 100 in subroutine MVNPRD. To correct the error, the statement

```
GO TO 40
```

which immediately precedes statement 100 on p. 575 should be changed to

```
IF (Z.LT.ZU .AND. NTM.LT.NMAX) GO TO 40
```

A copy of the corrected algorithm, either by electronic mail or on a diskette, as well as an algorithm which extends algorithm AS 251 to include multivariate Student probability integrals, may be obtained by writing to the author.

†*Address for correspondence*: Department of Mathematics and Statistics, McMaster University, 1280 Main Street West, Hamilton, Ontario, L8S 4K1, Canada.