Adaptive Rejection Sampling And Squeeze Sampling

Peter Trubey

December 15, 2012

Abstract

This paper will briefly explain the Accept/Reject algorithm, then introduce differences between it and Squeeze Sampling. From this it will expand on that to introduce Adaptive Rejection Sampling (ARS), a method which builds on Accept/Reject and Squeeze Sampling. ARS creates progressively more exact sampling and evaluation densities which converge to the target density over a number of iterations.

Introduction

We use random samples in every form of simulation or modeling. The validity of these simulations depends entirely on the validity of samples they are based on. The need for generating independent random samples from a given distribution is apparent from their use in modeling.

Accept/Reject

The Accept/Reject algorithm is a widely recognized method to draw independent random samples from a distribution. The algorithm offered here(Algorithm 1) comes from Robert and Casella[4].

Essentially, we sample from a sampling distribution, which we have fit over our target distribution with the help of a multiplier, M. Then we take one sample from the standard uniform distribution, and compare that sample to the ratio of the densities of the target distribution over the sampling distribution multiplied by M.

This method carries with it a few drawbacks. For every target distribution we need to select both a sampling density that closely matches the shape of the target distribution,

Algorithm 1: Accept Reject Algorithm

For Target Distribution $f(y)$
begin
  Select Appropriate Proposal Density $g(y)$
  Calculate M such that $Mg(y) \geq f(y) \qquad \forall Y \in R$
  while *Length(Accepted Samples) < Desired Sample Size* **do**
    Generate $y \sim g(y)$
    Generate $u \sim U[0,1]$
    if $u \leq \frac{f(y)}{Mg(y)}$ **then**
      Accept y
    else
      Reject y
    end
  end
end

as well as a multiplier that places the sampling density above the target density for the whole support. This may lead to the need rejecting many samples to accept only a few. Choosing the sampling density and multiplier leads us to the second problem; we have to know a great deal of information about the target distribution. Further, evaluating the target density at every interval can be computationally expensive. Squeeze Sampling attempts to solve this particular problem.

Squeeze Sampling

Squeeze Sampling (Algorithm 2) is a simple expansion to the Accept/Reject algorithm. It adds an evaluation curve below the target density which should be quicker to evaluate than the target. The concept of Squeeze Sampling goes at least as far back as 1977[3][2]. Squeeze Sampling brings with it many of the drawbacks of accept/reject along with the fact we need to know even more information about the target density, in order to select an appropriate evaluation density.

Due to the need to know a great deal of information about the target density prior to implementing the algorithm, we cannot generalize these methods for families of functions. Every target density will require a new squeezing function, calculated prior to implementation. Finding that squeezing function can be difficult.

Squeeze Sampling Example

In this example (Figure 1) We will draw a sample from the density $f(x) = 1 - x^2$, $0 \leq x \leq 1$. To do this, we will draw from $g_u(x)$ where $x \sim Unif[0,1]$. we will use an evaluation density of $g_l(x) = 1 - x$. We see the samples first accepted by the evaluation

Algorithm 2: Squeeze Sampling

For Target Distribution $f(y)$
begin
  Select Appropriate Proposal Curve $g_u(y)$ such that $g_u(y) \geq f(y) \quad \forall y \in R$
  Select Appropriate Evaluation Curve $g_l(y)$ such that $g_l(y) \leq f(y) \quad \forall y \in R$
  while *Length(Accepted Samples) < Desired Sample Size* **do**
    Generate $y \sim g_u(y)$
    Generate $u \sim U[0, 1]$
    if $u \leq \frac{g_l(y)}{g_u(y)}$ **then**
      Accept y
    end
    else if $u \leq \frac{f(y)}{g_u(y)}$ **then**
      Accept y
    end
    else
      Reject y
    end
  end
end

curve $u_l(x)$ in green. those samples rejected initially but accepted after comparison to the target density are shown in blue. We can see that given a sufficiently complex target density, and a sufficiently close evaluation curve, we can make great gains in efficiency by evaluating first against this evaluation curve.

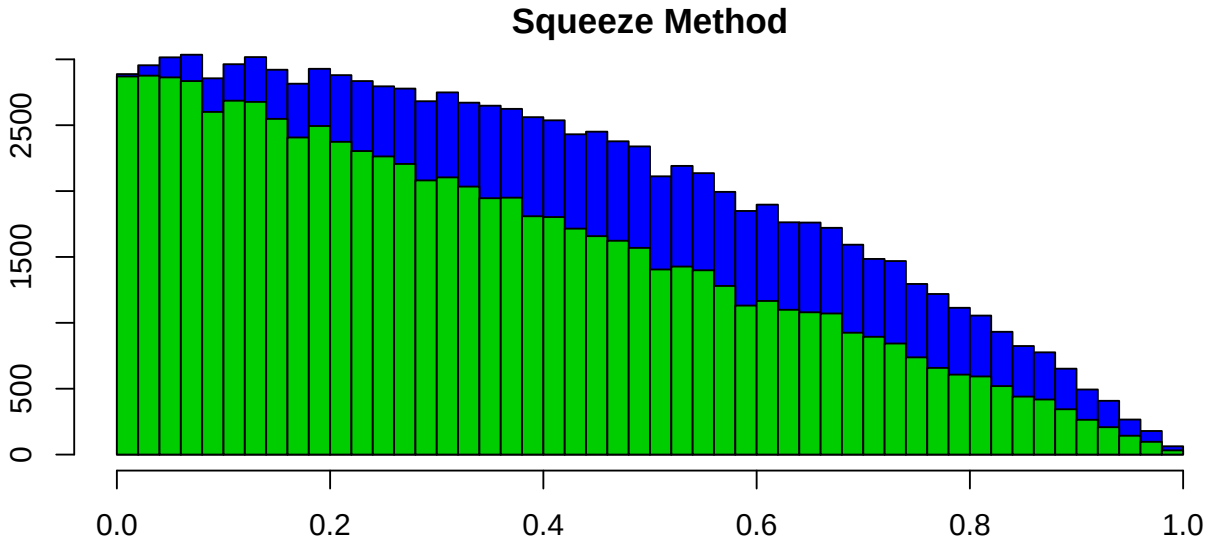
Adaptive Rejection Sampling

The ARS Method (Algorithm 3) was released in 1992 by Gilk's and Wild[1]. The basic idea involves fitting progressively more exact piecewise exponential functions as lower and upper hulls, which replace the sampling and evaluation densities used in Squeeze Sampling. This method makes a few requirements of the target distribution: log concavity, continuity, and differentiability over the support. Many distributions that are commonly used meet these requirements, whether through their independent variable, or transformations of them. These include, but are not limited to, normal, log-normal, exponential, beta, gamma, logistic, Weibull, and Pareto.[1]

The goal of ARS is to avoid the pitfalls associated with Accept/Reject and Squeeze Sampling. Given the above requirements of the distribution are satisfied, we can use ARS with a black box method where we know no information about the target density other than its function. Because the upper hull converges to the target density over time, we have fewer rejected samples compared to Accept/Reject and Squeeze Sampling. Because the lower hull converges to the target density over time, we will have to evaluate the target density fewer times compared to Squeeze Sampling.

Figure 1: Squeeze Sampling in Use

```
nsim = 10^5
f = function(x){1-x^2}
gl = function(x){1-x}
x = numeric(0)
while(length(x)<nsim){
  y = runif(1);
  w = runif(1);
  if(w<gl(y)){x = c(x,y)}
  else if(w<f(y)){x = c(x,y)}
}
```



Adaptive Rejection Sampling Method

The first step necessary for ARS is to isolate the kernel of the target distribution, and calculate the natural log of it. This act moves the function into log space, which is advantageous for us in that exponential functions are straight lines in log space. Thus tangent lines calculated in log space that form a piecewise linear upper hull will form a piecewise exponential upper hull in Euclidean Space. We also calculate the derivative of the log-transformed kernel, allowing us to make those tangent lines. Then we move on to initializing the chain.

Algorithm 3: Adaptive Rejection Sampling

Data: For Target Distribution $f(x)$
begin
  Calculate $h(x) = ln(g(x))$ where $g(x) \propto f(x)$
  Calculate $h'(x) = \frac{d}{dx}h(x)$
  Initialize the Chain
  while *length(Sample) < Desired Sample Size* **do**
    $x* =$ Sample Once from $s(x)$, Piecewise Exponential Upper Hull
    $w =$ Generate One Random Sample from $Uniform(0, 1)$
    Compare w to Ratio of Upper and Lower Hulls at $x*$
    if *Reject* **then**
      Evaluate $h(x*)$, $h'(x*)$
      Update Upper and Lower Hulls with new Abscissa
      Compare w to Ratio of Upper Hull and Target Density at $x*$
      if *Reject* **then**
        Point is Rejected
        Return to Sampling Step
      else
        Accept
      end
    else
      Accept
    end
  end
end

Initialize the Chain

To initialize the chain, (Algorithm 4) based on whether we have a bounded domain we choose starting values in order to generate a starting upper and lower hull. If we're using an unbounded domain, we place requirements on those values such that for the lowest starting value, the log-kernel has a positive derivative at that value. We place a similar but opposite requirement on the highest starting value, requiring it to have a negative derivative. We place these requirements in order to keep the integration space underneath the upper hull finite.

Having defined the starting abscissae, we calculate their tangent lines, and their intersection points. For k abscissae, there are k tangent lines, and $k-1$ intersection points. At intersection j in $j = \{1, \ldots, k-1\}$, the point of intersection and boundary of integration space is given as

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

So for $x \in [z_{j-1}, z_j]$, $j = \{1 \ldots k\}$

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

Algorithm 4: ARS: Initializing the Chain

begin
  if *Domain is Bounded* **then**
    Choose Starting Values within Domain Endpoints
  else
    Choose Starting Values such that $h'(x_1) > 0$, and $h'(x_k) < 0$
  end
  Define Functions $u_k(x)$, $s_k(x)$, and $l_k(x)$
  Begin Sampling Step
end

We then normalize $u_k(x)$ over the domain to create the piecewise exponential distribution we sample from, $s_k(x)$.

$$s_k(x) = \frac{exp(u_k(x))}{\int_D exp(u_k(x')dx')}$$

The lower hull $l_k(x)$ is a piecewise linear function whose intercepts are at the abscissae where $h(x)$ was evaluated. For $x \in [x_j, x_{j+1}]$, $j = \{1, \dots k-1\}$,

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

Because this is being calculated in log space, we can visualize these lower hull pieces as simple lines between abscissae. The slope of these lines can be calculated through $\frac{rise}{run}$. When transformed back into Euclidean space, we see them as exponential functions.

Sampling

Algorithm 5: ARS: Sampling Step

begin
  Sample new point $x* \sim s_k(x) = \frac{exp(u_k(x))}{\int_D exp(u_k(x')dx')}$
  Sample new point $w \sim Unif[0, 1]$
  if $w \leq exp(l_k(x*) - u_k(x*))$ **then**
    Accept $x*$
    Return to Sample Step
  end
  Evaluate $h(x*)$, $h'(x*)$
  if $w \leq exp(h(x*) - u_k(x*))$ **then**
    Accept $x*$
    Go to Update Step
  else
    Go to Update Step
  end
end

In the sampling step,(Algorithm 5) we sample a new point $x*$ from the normalized piecewise upper hull, $s_k(x)$. We generate another value w from a random uniform distribution, between 0 and 1. We compare w against the ratio of the lower hull and upper hull transformed back into Euclidean Space. The closer the difference is to 0, the closer the odds of acceptance are to 1. If w is less than that ratio, we accept the x value and continue sampling. If it is not, we compare w against the ratio of the target density and the upper hull. If w is less than this ratio, then we still accept the x value. If it is greater than this ratio, we reject x. Either way, because we have evaluated the target density at this point, we move on to updating the hulls and sampling distribution.

Updating

Algorithm 6: ARS: Updating the Hulls and Sampling Distribution

begin
- Reorder Abscissae in Ascending Order, Including $x*$
- Calculate new Tangent Lines to form upper hull
- Calculate new Intersection Points
- Calculate new $u_{k+1}(x)$ with new Intersection Points
- Calculate new $s_{k+1}(x)$ with new $u_{k+1}(x)$
- Calculate new $l_{k+1}(x)$ with new Abscissa
- Return new Functions to Sampling Step

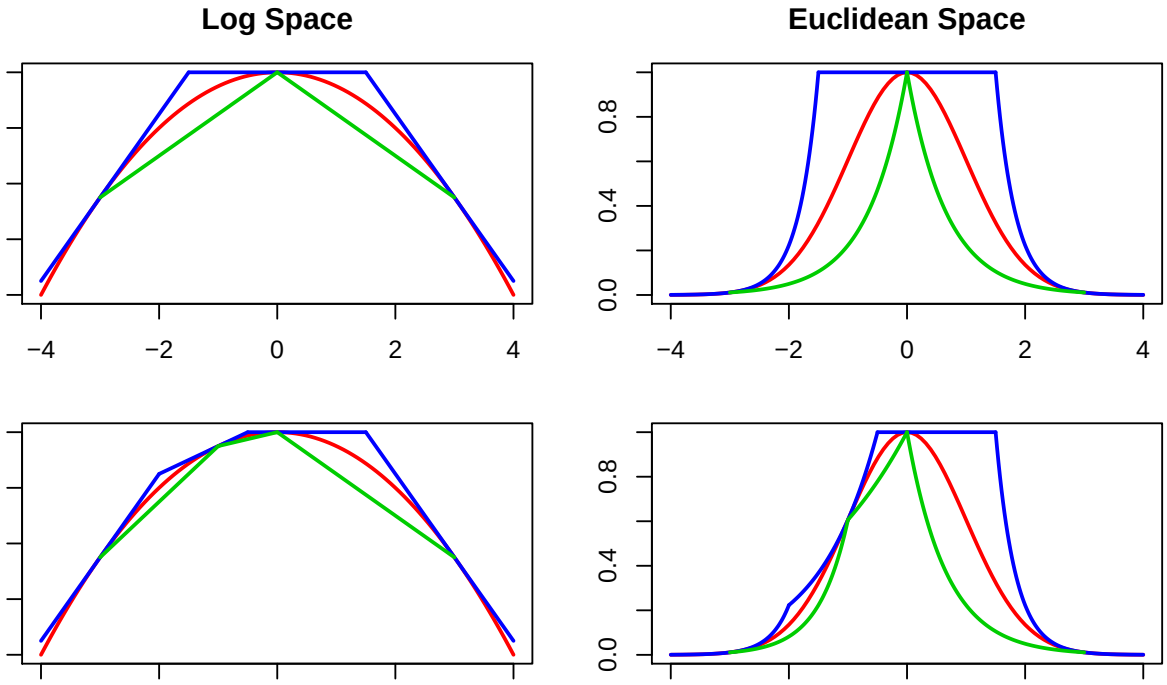
end

We move on to updating the hulls.(Algorithm 6) As $x*$ was added to the list of abscissae, we need to reorder it such that $x*$ is between its closest neighbors. We then calculate the tangent line at $x*$, and the new intersection points for its neighboring tangent lines. This forms the new upper hull, $u_{k+1}(x)$. We then integrate the total space under $u_{k+1}(x)$, and normalize it to form $s_{k+1}(x)$. We also form the new lower hull $l_{k+1}(x)$, adding the new abscissa to the intersection points, and calculating the associated intersection lines. Then we return to the sampling step.(Algorithm 5)

In this manufactured example,(Figure 2) we are trying to sample from a standard normal distribution. The upper hull is defined by 3 tangent lines, calculated at 3 abscissae. Those abscissae being -3, 0, and 3. Let's say we randomly sample and get $x* = -1$, and we reject it. We add another abscissa at -1, and update the upper and lower hulls to account for the new information.

Given sufficient calculation power and abscissae, we would eventually converge such that both the upper and lower hulls were equal to the target density.

Figure 2: Fitting Hulls to Normal Distribution



Examples

ARS Example 1: Truncated Normal Distribution

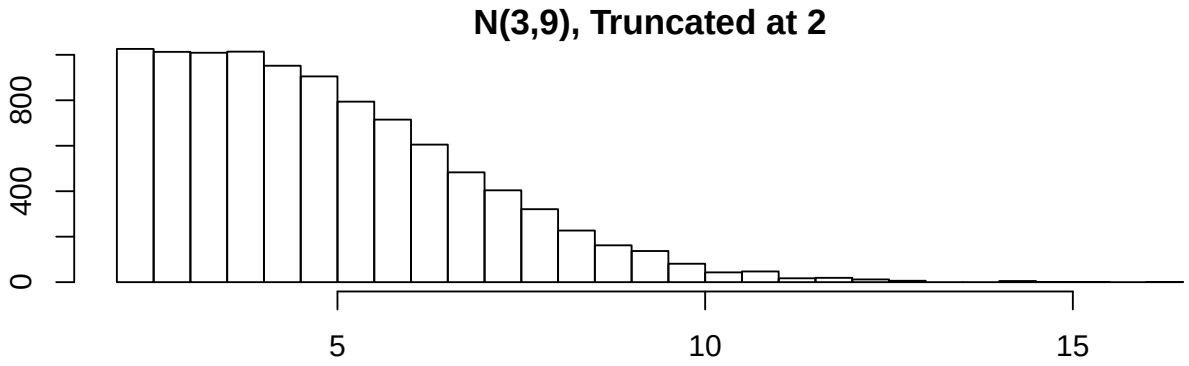
For these examples we'll be using the ARS package in R[5], which is written in C++ and based on the original fortran code by Gilks and Wild. Here we sample 10^4 random values from a $N(3, 9)$ distribution truncated at 2.(Figure 3)

$$f(x) = \frac{-1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(x-\mu)^2}$$
$$g(x) = e^{\frac{-1}{2\sigma^2}(x-\mu)^2}$$
$$h(x) = \frac{-1}{2\sigma^2}(x-\mu)^2$$
$$h'(x) = \frac{-1}{\sigma^2}(x-\mu)$$

As we can see, all the algorithm needs to work is the log of the kernel of the target density, and the derivative of that log-kernel. Defining some starting values makes this implementation more reliable and gives better results.

Figure 3: Truncated Normal Example

```
f = function(x,mu=0,sigma=1){-1/(2*sigma^2)*(x-mu)^2}
fprima = function(x,mu=0,sigma=1){-1/sigma^2*(x-mu)}
x = ars(10^4,f,fprima,mu=3,sigma=3,lb=TRUE,xlb=2,x = c(2,4,10))
```



ARS Example 2: Weibull Distribution

$$
\begin{aligned}
f(x) &= \frac{a}{b}(\frac{x}{b})^{a-1}e^{-(\frac{x}{b})^a} \\
&\propto (\frac{x}{b})^{a-1}e^{-(\frac{x}{b})^a} \\
h(x) &= log(\propto f(x)) \\
&= (a-1)log(\frac{x}{b}) - (\frac{X}{b})^a \\
h'(x) &= \frac{d}{dx}h(x) \\
&= \frac{a-1}{x} - \frac{a}{b}(\frac{x}{b})^{a-1}
\end{aligned}
$$

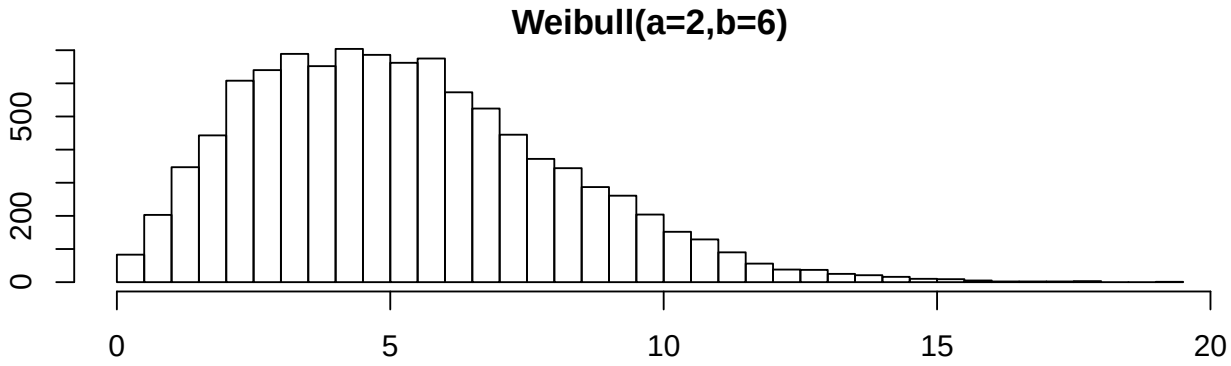
Here we sample 10^4 samples from a Weibull distribution.(Figure 4) The Weibull Distribution has a lower bound at 0, so we need to define that. I have given it some starting values in order to define an initial upper hull. The algorithm returns a sample of Weibull variates.

Conclusion

We can use ARS to give us independent and identical draws from any distribution that satisfies the requirements. This method is very powerful because we are not constrained by the requirements to know a great deal of information about the distribution prior to sampling from it. However, as described, it does have some weaknesses.

Figure 4: Weibull Example

```
f = function(x,a=1,b=1){(a-1)*log(x/b) - (x/b)^a}
fprima = function(x,a=1,b=1){(a-1)/x - a*(x/b)^(a-1)/b}
sample=ars(10^4,f,fprima,a=2,b=6,lb=T,xlb=0,x=c(1,4,12))
```



Drawbacks

As described the ARS algorithm 3 is limited to univariate distributions. I will discuss later about why it is difficult to expand this to multivariate distributions. One method that is commonly used to bypass this restriction is through Gibbs Sampling, where we would use ARS to approximate the target distribution in the Gibbs Sampler.[1] Additionally, the implementation in the ARS package[5] is somewhat unstable and for reliable and efficient operation requires user-defined starting points.

ARS is much more complex than more traditional methods of generation, like Accept/Reject. For a given situation, we need to evaluate if the advantages of this method are worth the additional complexity.

Advantages

ARS gives us independent and identical samples from any log-concave, continuous and differentiable distribution. The a majority of distributions in common use meet these requirements, so this method will allow us to create draws for the bulk of distributions in common use. It is thus a very powerful method for the right situations, where a more traditional method may be even more computationally expensive or complex.

Further Work

An area where we might expand this method would be the ability to draw from multidimensional distributions. In order to present the difficulty we'll use the relatively simple case of a 2 dimensional function. The problem with expanding this algorithm to higher dimensions is the ability to reliably define an integration space in order to find the density under the object.

Transforming a log-concave two dimensional distribution into log space would give us roughly a cone. Sampling and rejection will result in an abscissa being added to the cone. At that abscissa we calculate its density and normal line to define a plane tangent to the cone at that abscissa. That plane will intersect the planes around it, and the projection of those intersecting lines onto the xy plane will define the integration space of that piece of the upper hull. As this goes on, in log space the upper hull will begin to resemble a geodesic dome.

These integration space boundaries will not happen in a predictable fashion. Intersections will happen at all angles, and polygons defining the integration spaces can have a large and unpredictable number of vertices, making a method of defining and mathematically integrating them very difficult to implement. And yet such a method is absolutely necessary for a multidimensional ARS as described here.

References

[1] W.R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 41(2):337–348, 1992.

[2] Donald E. Knuth. *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[3] George Marsaglia. The squeeze method for generating gamma variates. *Comp. & Maths with Applications*, 3(4):321–325, 1977.

[4] Christian P. Robert and George Casella. *Introducing Monte Carlo Methods with R (Use R)*. Springer Science+Business Media, New York, NY, 1st edition, 2009.

[5] Paulio Perez Rodriguez and Arnost Komarek. ars: Adaptive rejection sampling, 3 2009.