STAT 243 Final Project: Adaptive-rejection Sampler

Due on Thursday, Dec 17, 2015

University of California, Berkeley
Fall 2015

Github repository: jJasonWang/stat243-fall-2015-FinalProject

Chao Mao
Xian Shi
Chih-Hui Wang
Luyun Zhao

1 Code Design

1.1 Pseudo code

Pseudo code of the main function is given in Algorithm 1. Auxiliary functions used in the main function are listed as follows:

- **initTk**$(h, h', a = -\infty, b = \infty)$: get the initial abscissae for the given function. (further discussed in section **1.2**)
- **genu**(T_k, h, h'): given the abscissae, log of the input function and its derivation function, generate a set of parameters (slopes and intercepts of each piece-wise linear functions) and breaking points $\mathbf{z}$ to construct the *upper hull* $\mathbf{u}(x)$ function.
- **samplex**(u, y): given the parameters and breaking points of $\mathbf{u}(x)$ function, and a vector of random numbers drawn from $U[0, 1]$, generate a vector of samples x^* from $s_k(x)$ using the inverse CDF method. (further discussed in section **1.3**)
- **evalu**$(x.temp, u_k)$: evaluate function $\mathbf{u}(x)$ at $x.temp$.
- **evall**$(x.temp, T_k)$: evaluate function $\mathbf{l}(x)$ at $x.temp$.

1.2 Optional input

If $h'(x)$ ($d\ \log f(x)/dx$) is known and written as a standard R function, it can be given as an optional input with keyword `hfun_deriv`.
If the input function $f(x)$ is a R built-in density function, e.g., dnorm, dlogis, their corresponding parameters can be written as input as well, e.g., mean = 1, sd = 1.

1.3 Initial Abscissae

As indicated in the research paper, initialization of the starting abscissae should satisfy the following conditions:

- When $x \in [a, \infty]$, $h'(x_k) > 0$
- When $x \in [-\infty, b]$, $h'(x_1) < 0$

which further imply

- When $x \in [-\infty, \infty]$, $h'(x_1) < 0, h'(x_k) > 0$

Without using any standard optimization technique which should be avoided according to the paper, we developed an iterative method with adaptive stepping to locate at least two x values that satisfy the above criterion. E.g. if $x \in [-\infty, \infty]$, a random x_1 is initialized. If $h'(x_1) < 0$, it continuously tries the next $x_{i+1} = x_i - \Delta(x_i)$, until a x_k is found with $h'(x_k) > 0$. Vice versa for $h'(x_1) > 0$. The adaptive step size $\Delta(x_i)$ is proportional to the absolute value of $h'(x_i)$ so that the final x_{k-1} and x_k are close to the mode. A maximum step size and maximum number of iterations are also set to ensure numerical robustness.

1.4 Inverse CDF Method

In order to sample x^* from $s_k(x)$, we used the inverse CDF method: Figure 1 shows the piece-wise function $exp(u_k(x))$. The area under each part of the function $(S_1, S_2, ..., S_5)$ represent unnormalized $s_k(x)$. The CDF of $s_k(x)$ can be therefore constructed by summing up piece-wise areas. A random number y was picked from $U[0, 1]$ and the corresponding area, $y \sum(S_i)$, indicates a particular area where x^* fall in. x^* can be then determined by applying inverse CDF method on this particular area with specific x interval.

```
Function: $x^\star \leftarrow$ **ars** $(n, f, a = -\infty, b = \infty, (h'), ...)$
**Input**  : $n \leftarrow$ Number of sampled data points;
             $f \leftarrow$ Density function, in the form of a standard R function, e.g. function(x);
             $a \leftarrow$ Lower bound of sampling interval;
             $b \leftarrow$ Upper bound of sampling interval;
             $h' \leftarrow$ Derivative of $\log(f)$. (Optional)
**Output**: $x^\star \leftarrow$ Sampled data points.

Checks for input validity;
$h \leftarrow \log(f)$
$T_k \leftarrow$ **initTk**$(h, h', a = -\infty, b = \infty)$;
// $k \geq 2$
$u \leftarrow$ **genu**$(T_k, h, h')$;
$i \leftarrow 0$;    $res \leftarrow \underbrace{(0, 0, ..., 0)}_{n}$;
**while** $i < n$ **do**
    $y \leftarrow runif(min(i, n - i + 1))$;
    $x.temp \leftarrow$ **samplex**$(u, y)$;
    // $s_k(x)$ is constructed within function samplex.
    // Piece-wise Inverse CDF method is used.
    $w \leftarrow$ **runif**$(1)$;
    $u.x \leftarrow$ **evalu**$(x.temp, u_k)$;
    $l.x \leftarrow$ **evall**$(x.temp, T_k)$;
    **if** $w \leq \exp(l.x - u.x)$ **then**
        $i \leftarrow i + 1$;
        res[i] $\leftarrow x.temp$;
    **else**
        $h.x \leftarrow h(x.temp)$;
        $h'.x \leftarrow h'(x.temp)$;
        **if** $w \leq \exp(h.x - u.x)$ **then**
            $i \leftarrow i + 1$;
            res[i] $\leftarrow x.temp$;
        **end**
        $T_k \leftarrow$ **sort**$(\{T_k, x.temp\})$;
        $u_k \leftarrow$ **genu**$(T_k, h, h')$;
        **if** $f$ *is not log-concave* **then**
            return Error;
        **end**
    **end**
**end**
return res
```

Algorithm 1: Adaptive-rejection sampler

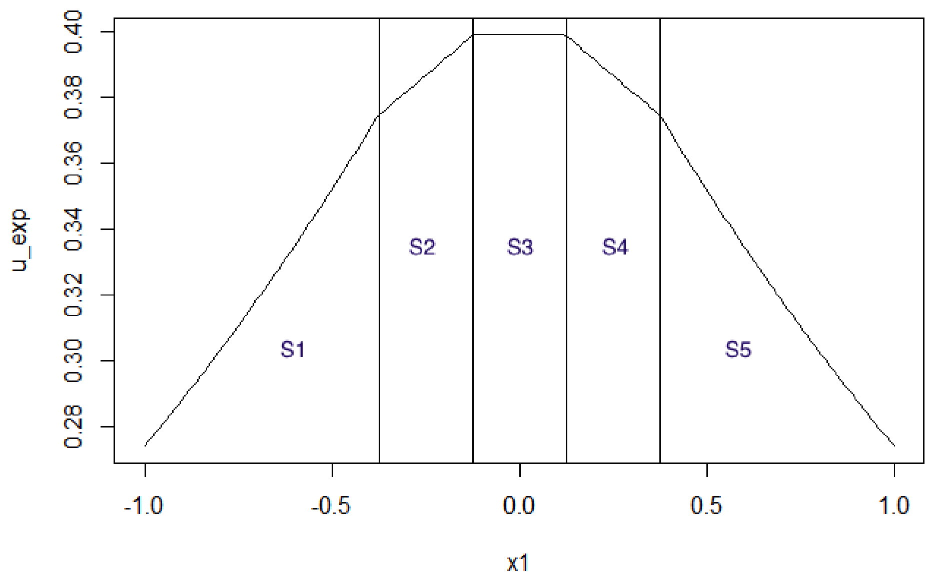

Figure 1: Plot of $exp(u_k(x))$

1.5 Performance Consideration

1.5.1 Evaluation of $h(x)$ and $h'(x)$

As indicated in the research paper, evaluations of $h(x)$ and $h'(x)$ can be computationally expensive. Therefore, evaluated $h(x)$ and $h'(x)$ when a new abscissa is added to T_k are also stored for future uses, e.g., updates of $u(x)$. It avoids re-evaluation of known T_k.

1.5.2 Vectorization and Dynamic Sample Size

According to the profiling of serial code, sampling of $x^\star$ was found as the most time-consuming step. Therefore, vectorization is applied to the inverse CDF sampling operation, i.e., function **samplex**. In the meanwhile, the number of $x^\star$ to be sampled at one step, $n_{x\star}$, is dynamically determined. Specifically, $n_{x\star}$ is the smaller value between i and $n-i+1$, as i is the number of already accepted $x^\star$ at current step. At first several steps, $n_{x\star}$ is most likely i so that $u(x)$ and $l(x)$ will be updated almost at every step. While at later steps, $u(x)$ and $l(x)$ have been updated enough and the sampling function only needs to sample the left amount of $x_\star$, which is $n-i+1$. The overall performance improvement is notably faster (approximately 8 times faster based on tests of sampling 10000 data points from standard normal distribution).

1.6 Checks and error messages

Generic checks, e.g. whether input parameters are valid, as well as log-concave checks have been applied. The log-concave check follows the definition "$h'(x)$ decreases monotonically with increasing x in D" and was numerically implemented at every sampling step.
For exponential-like density functions, the slopes of each piece $u_k(x)$ function are identical. As a result, break points z vector cannot be found through the general approach given in the paper. An *if* statement has been added to deal with this type of input function: instead of calculating $u_k(x)$ as a combination of different pieces, a single $u(x)$ function is used.

Sometimes abscissae initialization can be tricky when the input function has extreme parameters. Therefore a maximum number of searching steps is set (500) and an error message will appear if the subroutine fails to find satisfactory initial abscissae.

2 Test

2.1 Unit Test

We carried out some necessary tests for the auxiliary functions used in the main function, such as **initTk**(), **genu**(), **samplex**(), **evalu**() and **evall**(), to make sure they give us the right output. These tests can be found in *test-unit.R* under the *tests* directory.

2.2 Main Test

In the main test, we first test the functionality of *ars* to check the validity of the inputs, such as giving informative error messages when sample size is negative or the lower bound is bigger than upper bound. Then we use *ks.test* to test if the output sampled points follow the given distribution function for a significance level of 0.05. The distributions we have tried include: Normal distribution, Beta distribution, Gamma distribution, Logistic distribution, Weibull distribution, Uniform distribution and Exponential distribution. All the tests have passed and in Figure 2, we show some of our sampled results.

3 Work Distribution

It has been a fantastic experience working with each other in this group. Everyone has been dedicated to the project and made their indispensable contributions. The following table shows specific work distribution.

	Chao Mao	Xian Shi	Chih-hui Wang	Luyun Zhao
Algorithm	+++	+++	++	++
Code development	++	++	+++	+++
Performance	+++	+++	+	+
Test development	++	++	+++	+++
Package development	+	+	+++	+++
Report	+++	+++	++	++
Overall	+++	+++	+++	+++

Table 1: Work distribution

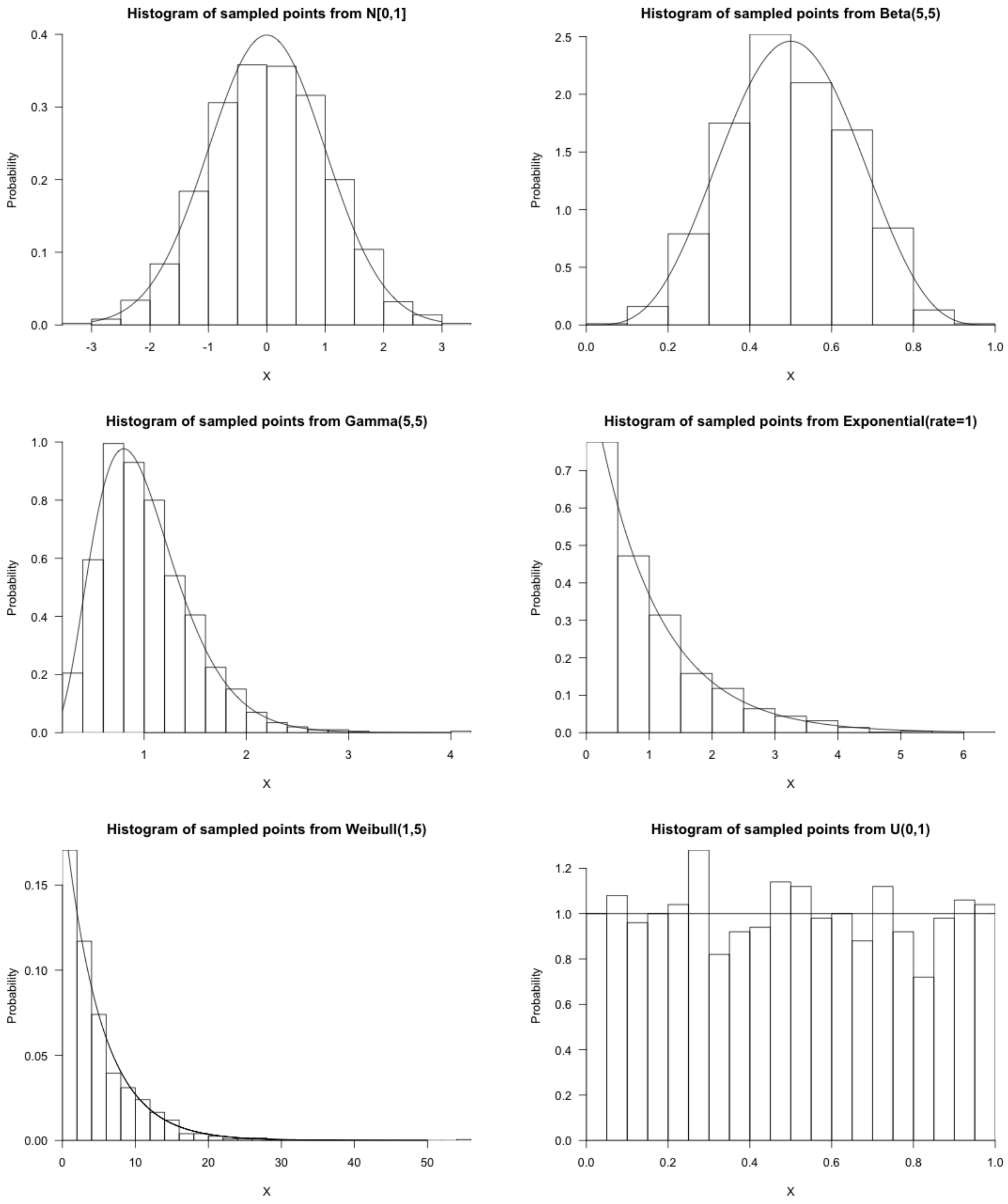

Figure 2: Sampled Results with Different Distribution Functions