Adaptive Rejection Sampling

Daniel Sunko, Inga Huld Ármann, Arm Wonghirundacha

2022-12-15

The Adaptive Rejection Sampling method implemented is based on
W. R. Gilks and P. Wild 1992, 'Adaptive Rejection Sampling for Gibbs Sampling', Journal of the Royal Statistical Society, Series C (Applied Statistics), Vol. 41, No. 2 (1992), pp. 337-348.

Github

Our project files are at: https://github.berkeley.edu/daniel-sunko/ars

Can use the following command to install the package: R CMD INSTALL ars_0.1.tar.gz

Our Approach

We split the sampling algorithm from the paper into a few key steps and built functions around those steps. These steps include:

1. Initialisation : Find starting points T, find rejection envelope $u(x)$ and $l(x)$ and intersection points Z.
2. Sampling : Sample by computing the CDF of the upper envelope piecewise, normalising and using the inverse CDF method to find x^*. Perform squeeze test, rejection test with computed values of $u(x)$, $l(x)$, and $h(x)$, $h^{'}(x)$.
3. Updating : Compute values of new $u(x)$, $l(x)$, and $h(x)$, $h^{'}(x)$ as needed to update lists of values from the initialisation step.

We then place all these functions together in a main function which carries out the algorithm until the desired sampling number has been reached.

Primary Function

Our primary function takes in the user inputs of a function to sample from, starting points (if provided), the bounds of the function, and the number of sample points required. Apart from using the main function to compile all of our helper functions, we perform a log transform on the input function $g(x)$ to get $h(x)$.

Initialisation

The first step in the initialisation is obtain the starting points. If starting points are provided, then we just proceed with the initialisation following the paper, computing $u(x)$, $l(x)$, T, Z, $h(x)$ and $h^{'}(x)$ storing them in one big key-value list. However, if starting points are not provided, we find the maximum of the function provided, using the base R function optim, then add and subtract 1 from either side to get starting points on opposite sides of the maximum and then proceed with the initialisation steps described above.

Computing Z, $u(x)$ and $l(x)$

In computing the upper and lower envelopes, we again follow the equations provided in the paper. However, to save time and computation, we notice that the values of our starting points at $u(x)$ and $l(x)$ are just those

points evaluated at $h(x)$ since these points touch the density curve. So we simply look up in our list of values of $h(x)$ evaluated at x to get the values of $u(x)$ and $l(x)$ at those points. For Z which are the values of the intersection points, we simply used the formula in the paper to obtain our intersections.

Sampling

To sample from $\exp(u(x))$ where $u(x)$ differs depending on the interval (piecewise), we compute the $\int \exp(u(x))dx$ where the bounds of integration are from the bounds of the function (either provided by user or assumed -Inf, Inf) to the point of intersection with another $u(x)$ line, denoted Z. After obtaining these CDFs, we normalise them to provide the probabilities of sampling from under that $u(x)$. Once we obtain the CDF of the $u(x)$ to sample from, we use a standard uniform and the inverse CDF method to obtain a point from the distribution $\exp(u(x))$. Since the inverse CDF function relies on dividing by a derivative, and taking logs, we try to avoid NA and Nan values. So if the derivative of $h(x)$ is zero we sample from a uniform density instead of the exponential. To address taking negative logs, we approximate the log with a Taylor expansion in the case that the value is negative.

After sampling a new point x^* we perform the squeezing test and rejection test following the procedure on the paper, where we return the value of x^* along with booleans to denote passing the squeezing test, and/or passing the rejection test.

Updating

Following the paper, if $h(x^*)$ was evaluated then we must add x^* into T, sort the list then evaluate $u(x)$, $l(x)$, Z again. We choose to recompute all of these points if a new x^* is introduced into the set T since we want to ensure alignment of the values between each of the lists, that is, we want to ensure that T[1] corresponds to u[1], l[1] and so on. Due to the sorting of T values, there could be a misalignment if we do not recompute the values from scratch each time a new x^* is added. Since only a few x^*'s would be added to T due to the adaptive nature of the algorithm, we believe the choice to recompute all values will not become a bottleneck for space or runtime.

Testing

Our testing considerations mainly focus on checking the user inputs and the expected outputs on each of the helper functions, based on some values and types of inputs and outputs. To test the performance of our overall function we use visual inspection through plotting the histogram of our samples and the actual density, and plotting $u(x)$ and $l(x)$ functions to check they are tangent and bounding our density function. Apart from visual testing, we considered...

We assume that the user has inputed appropriate bounds for their density function of choice so we have decided not to create tests to catch errors resulting from points being out of the appropriate bounds.

Can run test by using the function testthat::test_file('./tests/testthat/test-main.R')

Results

We run our function with different common density functions and provided visualizations which shows the histogram of our sampled points overlayed by the true density function. Our plots look pretty good!

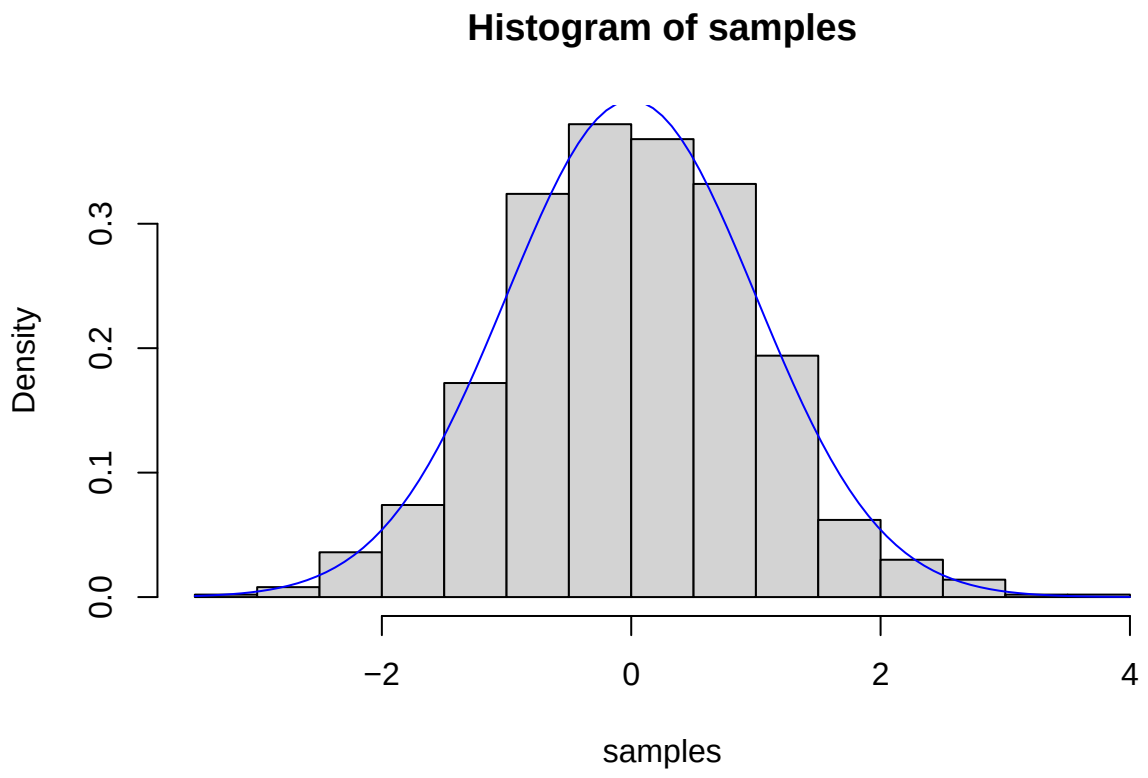
Standard Normal

```
set.seed(47)
source("main.r")
```

```
## Warning: package 'numDeriv' was built under R version 4.0.3
```

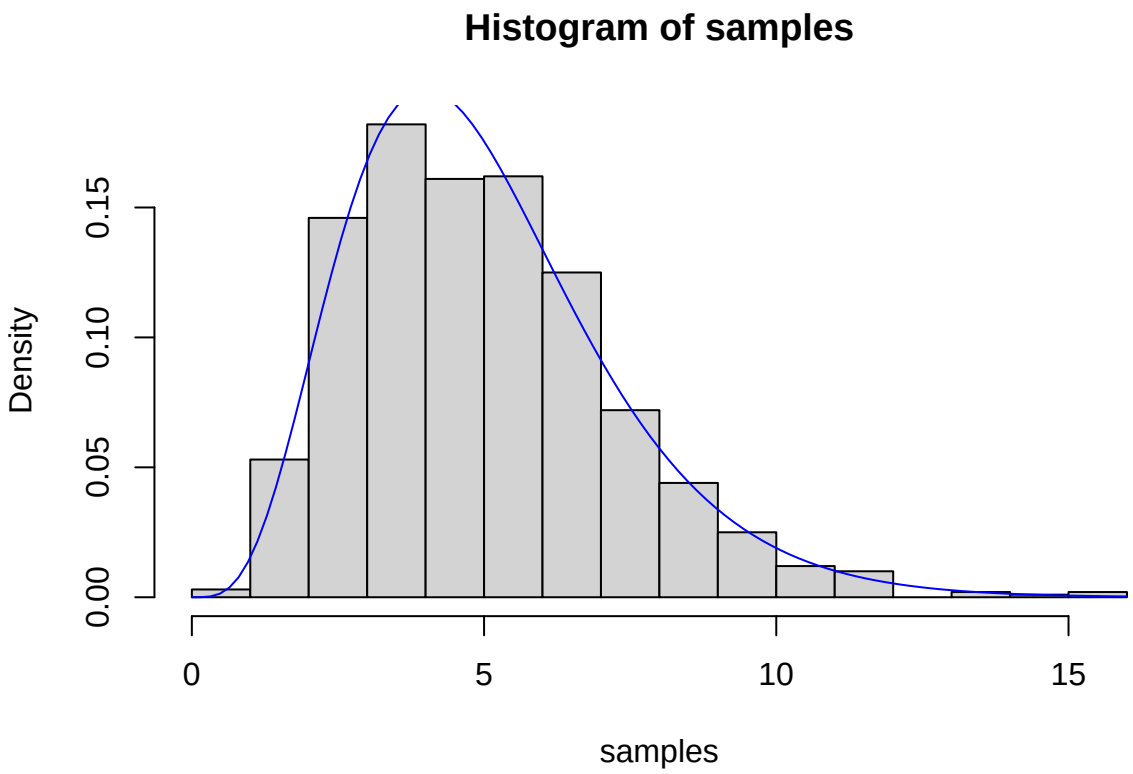
```
## Warning: package 'assertthat' was built under R version 4.0.5
```

```
f <- function(x) dnorm(x)
samples <- ars(f, n=1000)
hist(samples , probability = TRUE)
curve(f, add = TRUE, col = "blue")
```



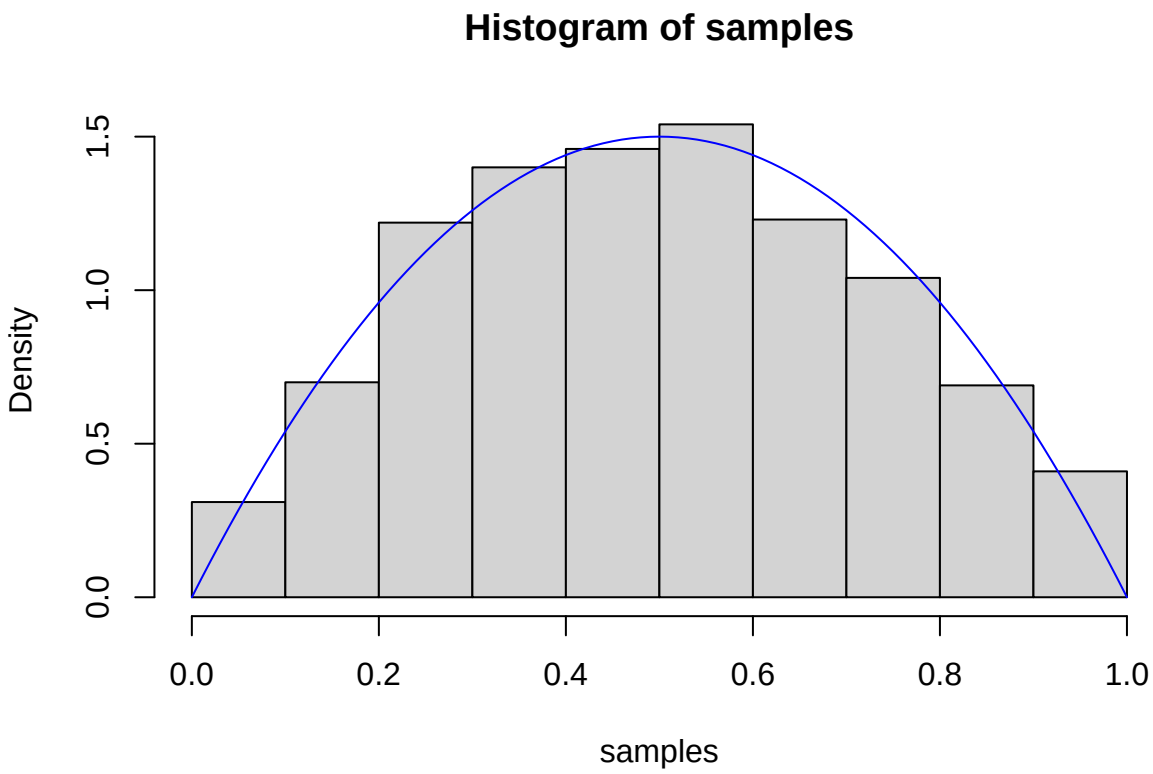
Gamma

```
f <- function(x) dgamma(x, 5)
samples  <- ars(f, n=1000, bounds = c(0, Inf))
hist(samples , probability = TRUE)
curve(f, add = TRUE, col = "blue")
```



Beta

```
f <- function(x) dbeta(x,2,2)
samples  <- ars(f, n=1000, bounds = c(0, 1))
hist(samples , probability = TRUE)
curve(f, add = TRUE, col = "blue")
```



Contributions

Note: our contributions are more like primary responsibilities than standalone tasks since we all check each others code, help debug and brainstorm ideas for our implementations and solve git merge conflicts together.

Daniel Sunko

Algorithm functions other than sampling, testing, package

Inga Huld Ármann

Testing, code organization and commenting

Arm Wonghirundacha

Sampling function, visual testing, write up