ARS Final Project
`github.com/dylandaniels/ars`

Dylan Daniels, Jiang Zhu, Tian Xia, Birce Tezel

December 17, 2015

In this project, we implemented the Adaptive Rejection Sampling algorithm and made it into an R package.

Implementation

The core part of the algorithm is implemented in the `ars()` function; helper functions are implemented in small, modular components. The more complicated part of the algorithm involves sampling from the distribution $s_k(x) = u_k(x)/\int_D u_k(x')dx' = c\ u_k(x)$. The basic strategy was to use inverse transform sampling. To do this, we first need to find the normalizing constant $c := 1/\int_D u_k(x')dx'$:

$$\int_D u_k(x')dx' = \sum_{j=0}^{k+1} I_j,$$

where

$$I_j := \int_{z_{j-1}}^{z_j} \exp(h(x_j) + (x' - x_j)h'(x_j))dx'.$$

Notice that the explicit form of I_j depends on whether $h'(x_j) = 0$ or not. Therefore we have,

$$I_j = \begin{cases} (z_j - z_{j-1})\exp(h(x_j)) & \text{if} \quad h'(x_j) = 0, \\ \frac{\exp u_k(z_j) - \exp u_k(z_{j-1})}{h'(x_j)} & \text{otherwise.} \end{cases}$$

Next, in order to use the inverse CDF method for sampling, we must find the CDF for $s_k(x)$, $S_k(x) = c\int_{z_0}^{x} u_k(x')dx'$:

$$S_k(x) = c\left(\sum_{j=0}^{t-1} I_j + \int_{z_{t-1}}^{x} \exp(h(x_t) + (x' - x_t)h'(x_t))dx')\right),$$

where t is the index of which interval of z's that x lies in. Formally, it is $t(x) = \{1 \le i \le k+1 : x \in (z_{i-1}, z_i)\}$. For convenience, let

$$\texttt{partialSums[t-1]} := \sum_{j=1}^{t-1} I_j$$

and notice that our normalizing constant can be expressed as $c =$ `1/partialSums[k]`. Moreover, let us define

$$J_{t-1}(x) := \int_{z_{t-1}}^{x} \exp(h(x_t) + (x' - x_t)h'(x_t))dx').$$

Then we have,

$$S_k(x) = c\left(\texttt{partialSums[t-1]} + J_{t-1}(x)\right),$$

where

$$J_{t-1}(x) = \begin{cases} \exp\big(h(x_t)\big)(x - z_{t-1}) & \text{if} \quad h'(x_t) = 0, \\ \frac{\exp u_k(x) - \exp u_k(z_{t-1})}{h'(x_t)} & \text{otherwise.} \end{cases}$$

Now, we can determine the inverse transform $S_k^{-1}(U)$, where $U \sim$ Uniform[0,1]. Because t is actually a function of x, it too must be inverted. Intuitively, we want to pick the biggest t such that $S_k(z_{t-1}) < U$. Formally, $t(U) = \{1 \leq i \leq k+1 : U \in (c \times \texttt{partialSums[i - 1]}, c \times \texttt{partialSums[i]})\}$. Solving for the inverse, we have:

$$S_k^{-1}(U) = \begin{cases} \frac{\frac{U}{c} - \texttt{partialSums[t-1]}}{\exp\left(h(x_t)\right)} + z_{t-1} & \text{if} \quad h'(x_t) = 0, \\ \frac{\log\left(h'(x_t)(\frac{U}{c} - \texttt{partialSums[t-1]}) + \exp u_k(z_{t-1})\right) - h(x_t)}{h'(x_t)} + x_t & \text{otherwise.} \end{cases}$$

In order to be more efficient, at each step of the algorithm, we only update the specific intersections points and integrals (`partialSums`) which change. After implementing this approach, we observed an order of magnitude improvement in speed!

Testing

We have written unit tests for each individual helper function, as well as tests for our main `ars()` function. Due to the stochastic nature of some of our functions, many tests are explicitly set with a seed for the pseudo-random number generator so that their output becomes deterministic, and thus, verifiable by an `expect_equal()` assertion.

In addition to automated testing, we checked if the samples we created are statistically correct. We first compared the histograms of our samples to the corresponding density function and visually checked if there were any unexpected behavior. These histograms along with the density function curves are presented in Figure 1. Second, we ran a series of Kolmogorov-Smirnov (KS) tests to verify that the empirical CDF of our generated samples matched the CDF of the true distribution. The KS test tests the null the hypothesis that the samples drawn from our algorithm match the target distribution. Thus, for the following tests, if our algorithm is working correctly, we expect high p-values. We ran the following KS tests:

```
#Normal distribution with mean 0 and standard deviation 1
set.seed(0)
samples <- ars(n=1e4, g=dnorm, dg=NULL, initialPoints=NULL, leftbound=-Inf, rightbound=Inf)
print(ks.test(samples,pnorm)$p.value)

## [1] 0.5158495

#Uniform disribution between 0 an 1
set.seed(0)
samples <- ars(1e4, g=dunif, leftbound=0, rightbound=1)
print(ks.test(samples,punif)$p.value)

## [1] 0.4646052

#Gamma distribution with shape parameters alpha = 4, beta = 5
set.seed(0)
samples <- ars(1e4, g=ddgamma, leftbound=0, rightbound=Inf)
print(ks.test(samples,ppgamma)$p.value)

## [1] 0.4975701

#Beta distribution with shape parameters alpha = 2, beta = 2
set.seed(0)
samples <- ars(1e4, g=ddbeta, leftbound=0, rightbound=1)
print(ks.test(samples,ppbeta)$p.value)
```

```
## [1] 0.5179732

#Chi-squared distribution with d.f. = 4
set.seed(0)
samples <- ars(1e4, g=ddchisq, leftbound=0, rightbound=Inf)
print(ks.test(samples,ppchisq)$p.value)

## [1] 0.5026843
```

Contributions

Jiang Zhu wrote the squeezing function, the precheck function and some part of the ars function. Jiang Zhu also wrote tests for squeezing function, precheck function and for common cases of the ars function.

Tian Xia wrote the envelope function, returning the value evaluated by the $u_k(x)$, test code for testing envelope function, intersects function, update function and part of the test code of the main function under some abnormal input functions.

Birce wrote the code for finding initial set of abscissae points if not provided by the user, finding and updating the intersection points of the envelope function, updating abscissae and vectors `hx` and `dhx` and finally the accept/reject function that takes returns the decision regarding a new sampled point.

Dylan Daniels wrote the code for sampling from the envelope function, including its tests. In addition, he spearheaded the auxiliary tasks, such as writing the help manual for the `ars()` function, debugging & style assistance, setting up the git repo and creating the package.

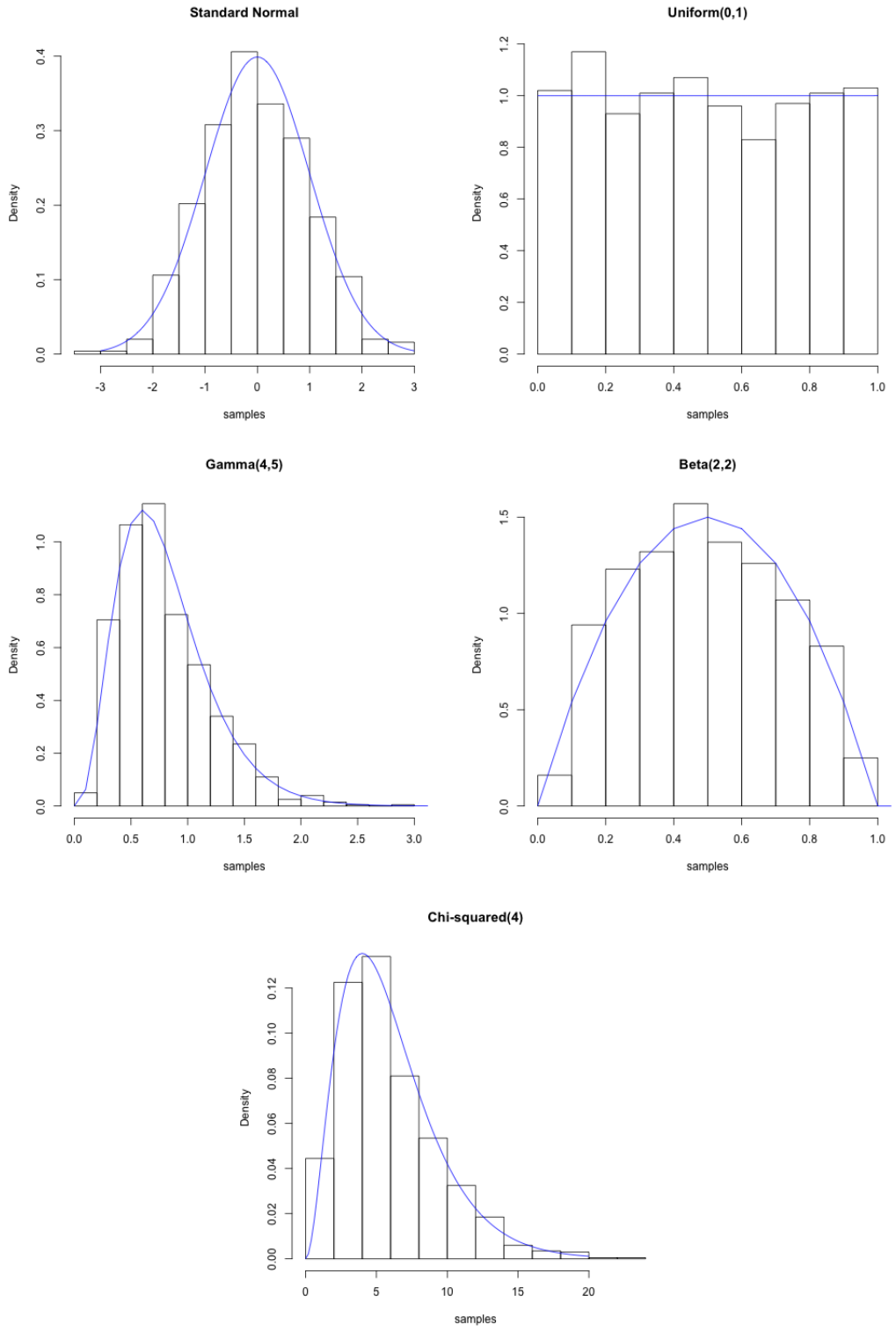

Figure 1: Histograms of the samples generated by `ars` compared to the true density functions.