Stat 243 Group Project

A.J. Torre, Chenxing (Esther) Wu, Phil Ryjanovsky

December 6, 2018

Introduction

Our approach to this problem was to follow the ARS paper in a straightforward manner to write a package to do Adaptive Rejection Sampling. We explain our approach and code below in the following order:

- Checking Assumptions/ Conditions of ARS
- Applying ARS
- Testing
- Group Member Contributions
- Package Details & Examples

Checking Assumptions/ Conditions of ARS

Our code does checks upfront to make sure the ARS from the Gilks and Wild paper can be applied to the domain and function that the user inputs. First, we take the log of the user's input function (outputting another function) and check the boundary points provided by the user. If the user did not input boundary points, the default lower boundary is set to negative infinity and the upper boundary to positive infinity. The user can input their own lower and upper bound in the ars function, but we also do a check on their bounds in case users inputted a very large or small upper or lower bound that causes issues in R with numerical accuracy. We then check whether the user's inputted lower and upper bounds give a value of negative or positive infinity when our `h_x` (the log of the inputted function) and/ or the derivative of `h_x` functions are evaluated at those points, in which case our function narrows the boundaries down until `h_x` and the derivative of `h_x` are finite. This step ensures that our code won't break in later steps, like in the `u_k` and `calc_zj` functions that need to use `h_x` and its derivative at a given point.

Then, we check if the user inputted an exponential or uniform distribution because we cannot follow the ARS for these distributions. For example, when calculating z_j's (see formula below), the denominator will equal 0 for uniform and exponential distributions as the derivatives of the logs of these functions are constants. So, if first two slopes are equal within machine numerical accuracy (8 digits) and then after, we check two cases: 1) if the first slope is approximately equal to 0 and then see if every other slope is equal to 0 as well (within 8 digits), which will be a uniform distribution, and 2) if the first two slopes are equal to each other but not 0, we save the first slope as our lambda and check every other slope to see if its equal within 8 digits of lambda. Then, if the user did indeed input a uniform or exponential distribution, we sample by using either runif in R's base package or the Rgeode package to sample from a truncated exponential.

Next, if the user did not input an exponential or uniform, our function then does a "rough/ approximate" check at the beginning to see if slope of h(x) is always decreasing (not strictly) and tell the user to try a different function if their's cannot pass this check. Our code compares consecutive slopes and if ever one slope is greater than the previous one, we return a message to the user that their function is not fully concave on the given domain. If a user's function passes through these initial checks and is log-concave but not exponential or uniform, then we continue with applying Adaptive Rejection Sampling, described below.

Applying ARS

Once the user's function passes these checks, we can then apply ARS as described in the Gilks and Wild paper. From the paper, the upper hull and lower hull are piecewise within given ranges. In our functions, we

wrote the $u_k(x)$and $l_k(x)$ functions in the form of wrapper functions that output a line that we can then plug different x values into. To make it simpler to calculate our $s_k(x)$, we solved for $u_k(x)$ and $l_k(x)$ in the form of $y = mx + b$. Formatting the $u_k(x)$ and $l_k(x)$ equations in the paper to the $y = mx + b$ form:

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

Above is our equation for our upper-hull line, with slope $h'(x_j)$ and intercept $h(x_j) - x_j h'(x_j)$, where x_j is the jth element in T_k, which is an ordered list of length k of x values that have been plugged into $h(x)$ and $h'(x)$. To code this function, we followed this formula, and in order to evaluate this integral, we used the `grad` function in the `pracma` package as it seemed the most efficient way to do numerical differentiation in R.

$$\begin{aligned} l_k(x) = \frac{(x_{j+1} - x)h(x_j) + (x - x_j)h(x_{j+1})}{x_{j+1} - x_j} &= \frac{x_{j+1}h(x_j) - xh(x_j) + xh(x_j + 1) - x_j h(x_{j+1})}{x_{j+1} - x_j} \\ &= \frac{(h(x_{j+1}) - h(x_j))}{x_{j+1} - x_j}x + \frac{x_{j+1}h(x_j) - x_j h(x_{j+1})}{x_{j+1} - x_j} \end{aligned}$$

Above is our equation for our lower-hull line, with slope $\frac{(h(x_{j+1}) - h(x_j))}{x_{j+1} - x_j}$ and intercept $\frac{x_{j+1}h(x_j) - x_j h(x_{j+1})}{x_{j+1} - x_j}$.

From the Gilks and Wild paper, in order to apply the $u_k(x)$ and $l_k(x)$ functions, we need to first check the range that x falls in. For $u_k(x)$, we need to check if x falls in $[z_{j-1}, z_j]$, where $j = 1...k$.

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

For applying, $l_k(x)$, checking the range for x is simpler as it involves no calculations, and we check if x is in $[x_j, x_{j+1}]$ for $j = 1, ...k - 1$. Also, following the paper, we defined $l_k(x)$ to be $-\infty$ when $x < x_1$ or if $x > x_k$. To check for log-concavity throughout the sampling and in the main ARS function (after our approximate check in the beginning of the function), we check the inequality that $l(x) \leq h(x) \leq u(x)$ and return a message that the user's function is not log-concave if this inequality is broken during the sampling process.

We calculate the normalized piece-wise $s_k(x)$ function by simply summing all the k integrals of $exp(u_k(x))$ to obtain the total integration constant, and then dividing a specific $exp(u_k(x))$ line by the constant to obtain a normalized $s_k(x)$ value, following the paper's equation:

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

The `sampler` function allows us to sample the values from density function $s_k(x)$ which we will test under the conditions in the "Sampling step" of the paper on page 341 in section 2.2.2.

Using the integrals of each of the k piecewise $exp(u_k(x))$ equations returned by the $s_k(x)$ function, we weight each of the k sections accordingly and use a randomly generated value from a uniform(0,1) to select one of them. We then obtain the normalized density of that section j by taking $exp(u_j(x))$, where j corresponds to the function where the jth ordered T_k value is and dividing it by the integral of $exp(u_j(x))$ returned by $s_k(x)$. We can now sample randomly from that section by taking the inverse CDF, $\frac{1}{IC_j}\int_{z_{j-1}}^{x} s_j(x)dx$, where IC_j is the integration constant corresponding to $s_j(x)$. This is equivalent to $\frac{1}{IC_j}\int_{z_{j-1}}^{x} exp(m_j x + b_j)dx$.

Then, we set this CDF equal to u, a value from a Unif(0,1) and solve for $x*$, and get

$$x* = \frac{log[(u)(IC_j)(m_j) + exp(m_j z_{j-1} + b_j)] - b_j}{m_j}$$

The main ARS function follows the paper on how to accept sample values or reject them. To initialize, we either use the starting points the user has provided, and if the user has not provided any, we look at the lower

and upper boundaries. From the lower boundary, we slowly increase until we find an x_1 such that $h'(x_1) > 0$, and from our upper boundary, we slowly decrease until we find an x_k until $h'(x_k) < 0$. Next, as we sample $x*$'s, given w, a value sampled independently from Unif(0,1), if $w \leq exp(l_k(x*) - u_k(x*))$, then we accept $x*$. If not, check $h(x*)$ and $h'(x*)$, and see if $w \leq exp(h(x*) - u_k(x*))$ and accept $x*$ if this inequality holds and if not, we reject it. The ars function also updates according to the paper: in the above case, our ars function will add the $x*$ to the T_k (the list of ordered x_j's where $h(x)$ and $h'(x)$ have been evaluated) and then use `sort` to give us our new re-ordered T_k.

Our main ARS function takes in six inputs: a function, number of samples, vector of starting points, lower bound and upper bound. If starting points are not inputted by the user, the default is NA and we will find suitable starting points within our function. If the upper and lower bounds are not input by the user, the default is positive and negative infinity and we will narrow our boundary (as described in the Checking Assumptions section) depending on the function the user input. The `ars` function will output the n values that we accepted while sampling. We test our function's sample values against outputs from rnorm, rchisq, and rgamma using ks.test, and our package has fairly high p-value and works well when the user supplies reasonable starting points and lower and upper bounds.

Testing

We carried out several tests including both simple checks on the type of output returned by our internal functions, as well as for the proper values that should be returned given concrete functions input by the user, by giving mock examples to our algorithm.

We tested all of our simpler functions for proper return types and proper values given a mock example where the input function was a normal distribution with varying initial values. We checked that our internal functions return proper indices, and functions for different sections of the s_k and u_k functions. We verified that our functions properly checked for assumptions, such as overall concavity, validity of the boundary points and starting points - that they caught proper and improper cases. We verified that uniform and exponential distributions are properly identified. Lastly, the most comprehensive tests checked the algorithm's results versus the truth. We sampled values from our package and used the Kolmogorov–Smirnov test (ks.test) to verify distribution sameness. We used the normal, chi-squared and gamma distributions as mock examples. (note that although these tests have all been consistently successful, theoretically, a test may give a false negative at some point).

Group Member Contributions

We split up tasks between group members and have several functions within our package to help our main function as each group member wrote different functions. AJ wrote the `u_k`, `l_k`, and `calc_zj`, and `is_unif_exp` functions and also did some of the work of the `get_xj_uk` and `get_xj_lk` functions and helped write the group paper. Phil wrote the `s_k` and `sampler` functions and also worked on the testing file for our package and helped write the group paper. Esther wrote `check_boundary`, `is_concave`, `find_sp` and the main `ars` function and also wrote the `ars` package and help page for the `ars` function. We all did some debugging/ testing along the way, and all team members wrote documentation and comments for their own functions/ parts.

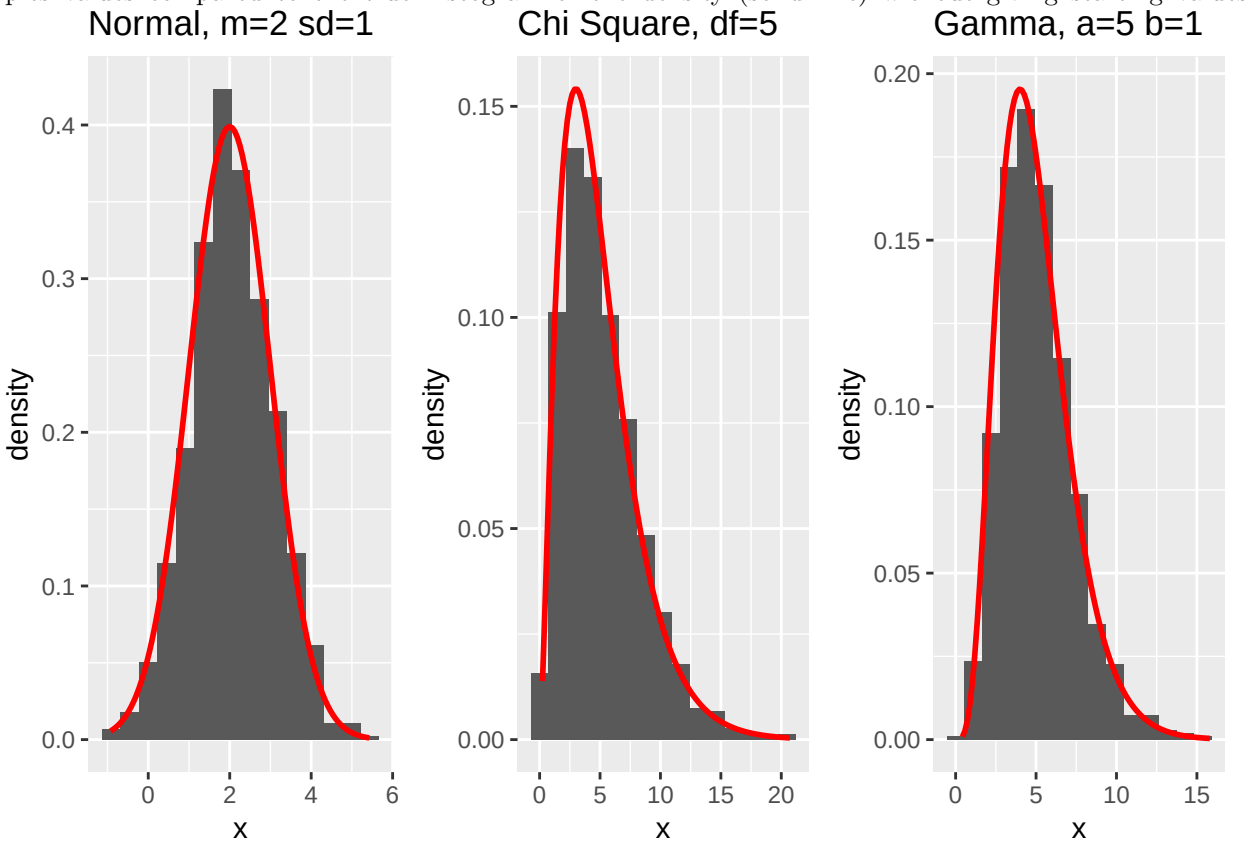
Package Details & Examples

Our package is called `ars`, stored at in Chenxing (Esther)'s GitHub at: https://github.com/ChenxingWu211/ars. Our main function is called `ars` and our pacakge uses the pracma (for the `grad` function) and RGeode (to sample from a truncated exponential) packages so the user needs to install these packages. To use and test this package, the user should use install and load the pracma, RGeode, and testthat packages. Below is the code the user needs to install the package (2 methods) and how to load it, access the help page, and test it.

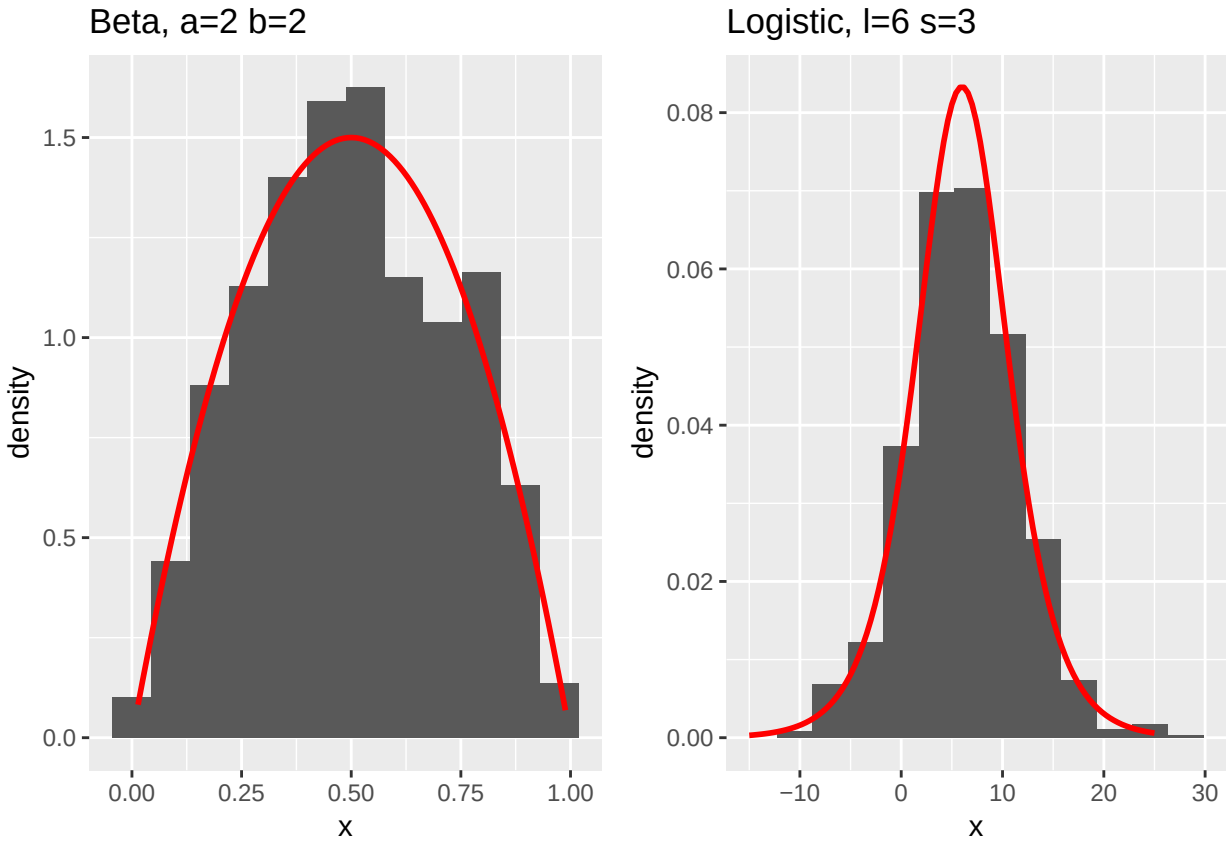
```r
#how to install our package
devtools::install_github('ChenxingWu211/ars')

#another way to install it to make sure
#the tests load
install_github("ChenxingWu211/ars", INSTALL_opts = '--install-tests')

library(ars)
#for help on main function
man(ars)

#the other packages used in our package
#user needs to install & load them
library(pracma)
library(RGeode)

#to test from our package
library(testthat)
test_package('ars')
```

Below are graph examples comparing results of our function to true distributions/ densities.

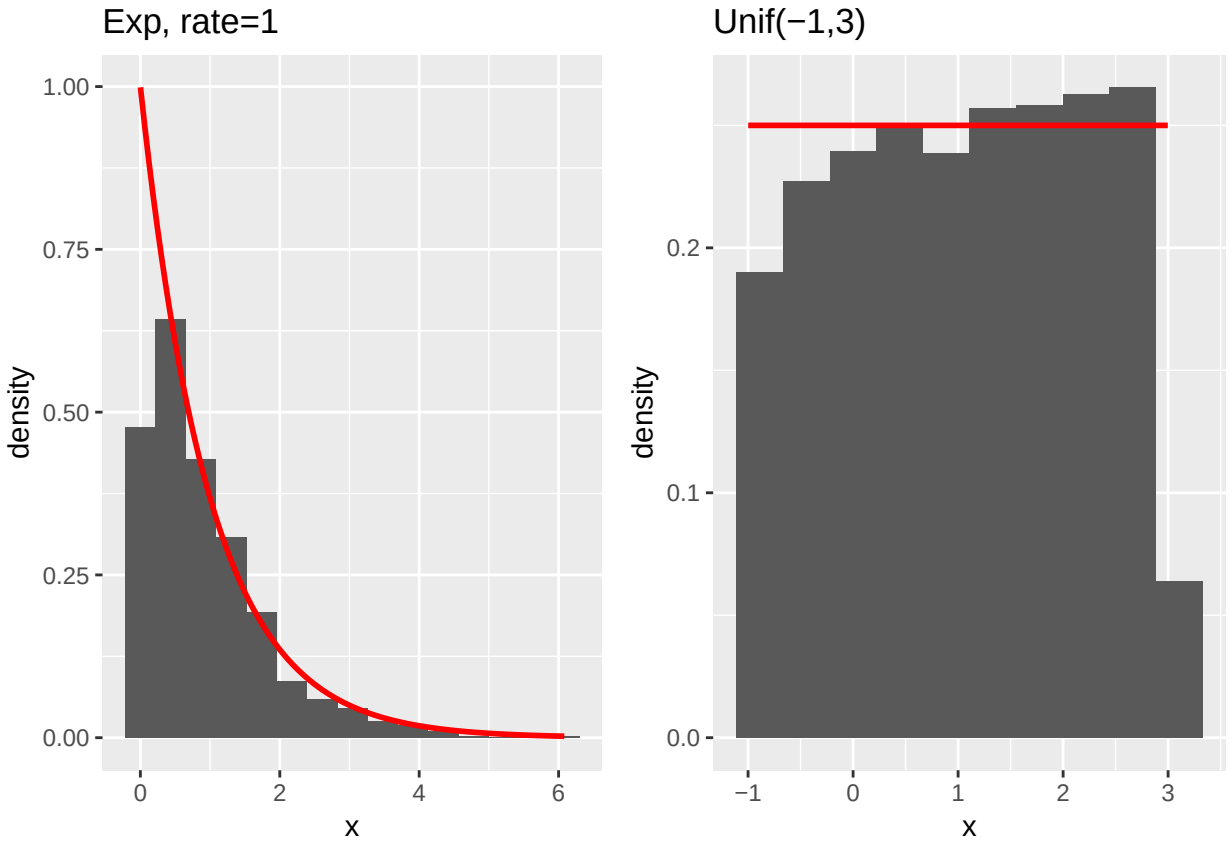
Examples of results for normal, chi-squared, and gamma distributions with the histograms of our samples values compared to the true histogram of the density (solid line) without giving starting values:



Examples of results for beta and logistic distributions with the histograms of our samples values compared to the true histogram of the density (solid line):



Examples of results for exponential and uniform distributions with the histograms of our samples values compared to the true histogram of the density (solid line). (note that our package recognizes when the user inputs a uniform or an exponential and runs runif or RGeode for the truncated exponential for sampling):


Exp, rate=1
Unif(−1,3)
density
x
1.00
0.75
0.50
0.25
0.00
0.2
0.1
0.0
0
2
4
6
−1
0
1
2
3