Adaptive Rejection Sampling – STAT 243

Brittney Benchoff, Christian Carmona, Hairong Xie, Thomas McCann

December 12, 2013

1 Description of Work

The group was given the task of implementing an adaptive rejection sampling algorithm to take user inputs and sample fectively, according to Gilks, et al, 1992. Initial development of pseudo-code led to a relatively natural modular form. Functions would be made to carry out individual tasks, and larger wrapper functions would encompass several of these individual functions in order to run a full simulation or run tests from a single command.

Object-oriented programming was used for the core functions to create a class called 'abscissae' with print and plot methods. One plot method is used to create a panel of plots to show the functions of interest (i.e. user-input $f(x)$, and function-defined $s(x)$ and $S(x)$.) A second plot function is used to create a panel many plots to create a visualization of the ARS algorithm as it runs through the loop. These plots portray the upper and lower bound functions as well as the accepted and rejected sampled points. The print method is used to print the sampled values from a single call by the user.

The function has initial tests to check that user-inputs are valid. For example, the user can input either a function/call or an expression that can be coerced into a function/call, but if any other type of object is entered as an input, a warning message will be displayed and the algorithm will not continue. Another check is placed at each iteration of the adaptive rejection sampling to ensure that the upper and lower bounds contain the entirety of the functhe R package, this is the file ars.R. The abscissae class constructor is found in abscissae_class.R, and the methods of that class are in abscissae_methods.R. Minor auxiliary function are found in auxiliar_fun.R. Rd format help files were used to compile the help manual for all functions.

2 Functions and Flow Chart

On the following page is a flow chart of the functions used in the ARS code. The following paragraphs will briefly describe each of these functions. For further information, please see the help manual.

Ars is called by the user with inputs for how many sampled points are desired, the function from which the sample should come from, the support of that function, an epsilon value used for numeric differentiation, and how many points should be sampled in a single iteration. There is an additional input used to save plots of each iteration of acceptance/rejection to a pdf file.

The ars function first checks that the user input function f is of the class 'function'. If it is not, it automatically converts expressions to function form before proceeding, or else returns an error for the user. Initial abscissae for the algorithm are input using *as.abscissae()* and then validated using *check.abscissae()*, which takes at least two user-input initial x values and runs eight different checks (i.e. at least 2 unique points; $log(f)$ is finite at x ; log-concavity).

The simulation is then run until the desired number of points have been sampled from the user's function. The simulation begins with sampling from a random uniform over $[0, 1]$. X^* is then found by sampling from $s(x)$. S_{inv}, *S()*, *s()*, and $int_s()$ will be discussed below.

plot.abscissae() then generates three plots in the panel - one showing the upper and lower bounds with $h(x)$ included if opted by the user; and the other two plots show the pdf $s(x)$ and the cdf $S(x)$. User can see that $s(x)$ is analogous to the original function $f(x)$. After this check, $e^{(h(x)-u(x))}$ and $e^{(l(x)-u(x))}$ are plotted to depict the acceptance and rejection areas. The lower bound function and upper bound function are defined methods named *l()* and *u()* respectively. *l()* and *u()* will be discussed below. The X^* sampled points are then plotted on top of this existing visual so it can be observed if they will be accepted or rejected.

The formal squeeze and rejections tests are performed by comparing the sampled uniform, w, to $e^{(x^*-u(x^*))}$, where $u(x)$ is the upper bound function $u()$. If w is less than this value, then the sampled point x^* is accepted. Any points that are not accepted at this stage are passed through the rejection test, where $h()$ and $h'()$ are evaluated at the rejected x^* values. Uniform w's are again compared to $e^{(x^*-u(x^*))}$, with $u()$ reevaluated for new $h()$ and $h'()$ information. From the squeeze and rejection tests, all accepted points are added to the set of simulated values, and the x values are added to the abscissae list which is then reordered along with the associated $h(x)$ and $h'(x)$ using the function *add.points.abscissae()*. The next iteration then begins with this information.

Several functions and methods mentioned above are utilized as support function to *ars()*. One key auxiliary function is *bucket()*. *bucket()* is called from the upper bound and lower bound functions as well as S and S_{inv}. An x value and set of domains are passed through *bucket()*, and it returns the index number showing which domain the x falls into. This was useful in working with the piecewise functions. *l()* is a function of x and an object of class abscissae (generated by *as.abscissae()*). It defines the lower bound of $h(x)$ - in this case, it is the secant lines between each of the abscissae points. From Gilks et al, 1992, this lower bound is defined as:

$$\text{For } x \in [x_j, x_{j+1}],\ \ j = 1, ..., k-1\ \ l_k(x) = \frac{(x_{j+1}-x)h(x_j) + (x-x_j)h(x_{j+1})}{x_{j+1}-x_j}\ .\ \text{For } x < x_1 \text{ or } x > x_k\ \ l_k(x) = -\infty.$$

Similarly, u is a function that defines the envelope function - tangent lines at the abscissae points, connected at points z_i. Below is the formula for this upper bound:

$$\text{For } x \in [z_{j-1}, z_j],\ j = 1, ..., k\ \ u_k(x) = h(x_j) + (x - x_j)h'(x_j)$$
$$\text{where } z_0 \text{ is the lower bound of } D \text{ (or } -\infty \text{ if } D \text{ is not bounded below)}$$
$$\text{and } z_k \text{ is the upper bound of } D \text{ (or } +\infty \text{ if } D \text{ is not bounded above).}$$

get.zi.abscissae() is method of the class abscissae and uses the abscissae list of x, $h(x)$, and $h'(x)$ values to return the intersection points of the upper envelope lines. *z()* is defined as:

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

To sample from *s()* using the ARS algorithm, several functions are used - int_s, s, S, and S_{inv}. int_s calculates the integration portion of the equation used to determine *s()*. The equation is shown below:

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

s(x) is a pdf. The cdf is *S()*. The key equation used is:

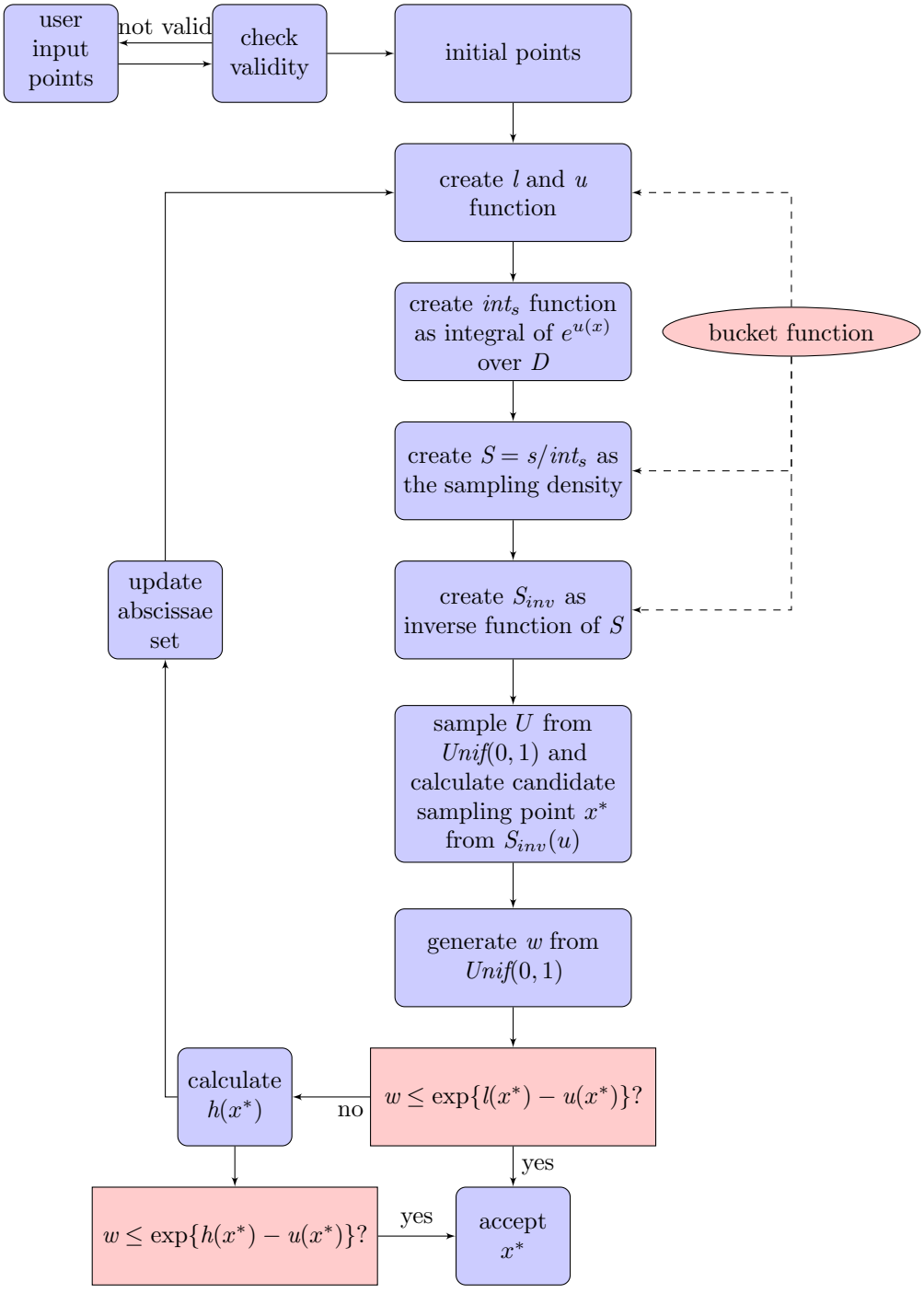
$$\text{For } x \in [z_{j-1}, z_j]\ \ S(x) = \frac{1}{R}\left(\sum_{i=1}^{j-1}\frac{1}{h'(x_i)}[e^{u(z_i)} - e^{u(z_{i-1})}] + \frac{1}{h'(x_j)}[e^{u(x)} - e^{u(z_{j-1})}]\right)\ \ \text{, where } R = \sum_{i=1}^{k}\frac{1}{h'(x_i)}[e^{u(z_i)} - e^{u(z_{i-1})}]$$

$S_{inv}()$ is then used to find the inverse of this cdf. This allows for sampling from *s(x)*. This was done using:

$$\text{For } x \in [z_{j-1}, z_j]\ \ S_{inv}(s) = \frac{log[(sR - \sum_{i=1}^{j-1}\frac{1}{h'(x_i)}[e^{u(z_i)} - e^{u(z_{i-1})}])h'(x_j) + e^{u(z_{j-1})}] - h(x_j)}{h'(x_j)} + x_j$$
$$\text{where } R = \sum_{i=1}^{k}\frac{1}{h'(x_i)}[e^{u(z_i)} - e^{u(z_{i-1})}]$$

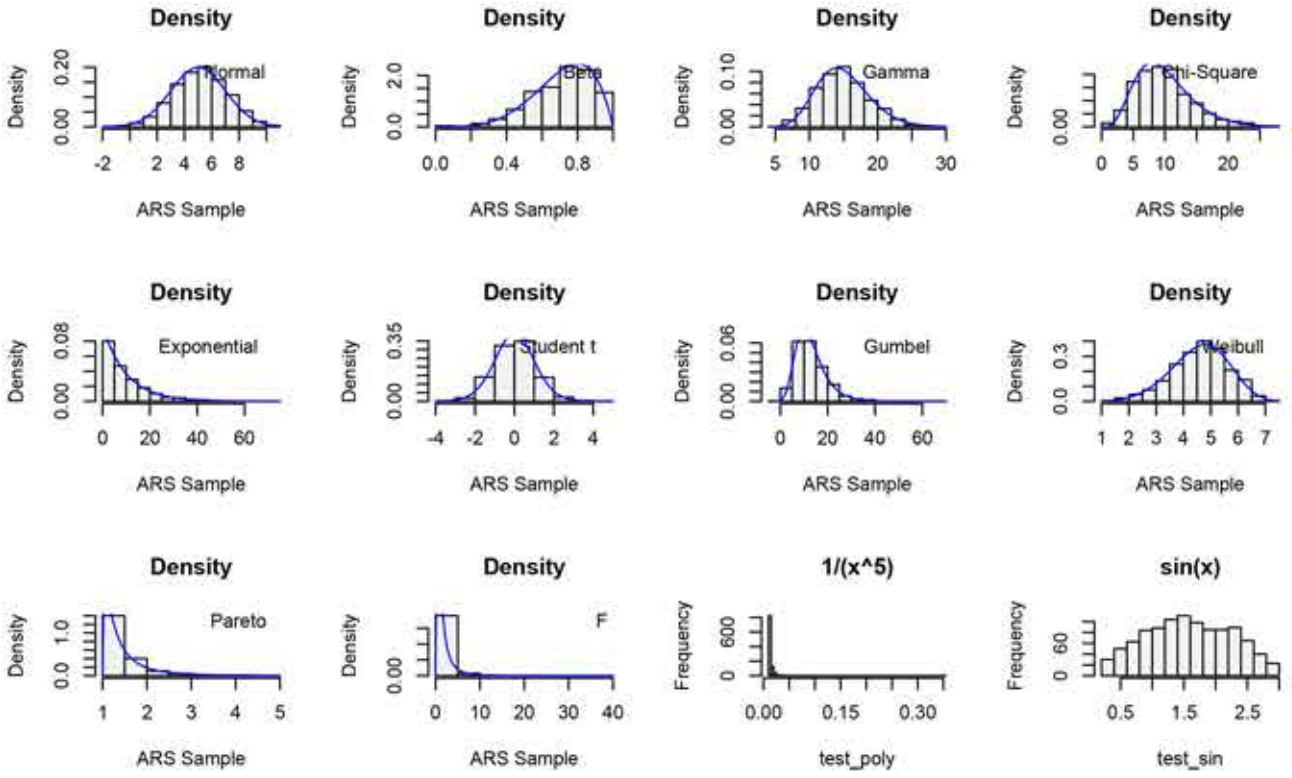
For additional information, see Help Manual.

Flow Chart


user input points
not valid
check validity
initial points
create l and u function
create int_s function as integral of $e^{u(x)}$ over D
bucket function
create $S = s/int_s$ as the sampling density
update abscissae set
create S_{inv} as inverse function of S
sample U from $Unif(0,1)$ and calculate candidate sampling point x^* from $S_{inv}(u)$
generate w from $Unif(0,1)$
calculate $h(x^*)$
no
$w \leq \exp\{l(x^*) - u(x^*)\}$?
yes
$w \leq \exp\{h(x^*) - u(x^*)\}$?
yes
accept x^*

3 Results

The implementation of adaptive rejection sampling was successful. A testing function was used to compare the algorithm to known distributions (i.e. dnorm). Resulting print-outs and plots can be seen below.



4 Contributions of Members

The group worked together through six face-to-face meetings. At early stages of our work, we discussed the problem and all made contributions to our general approach and the pseudo-code of the algorithm for adaptive rejection sampling. In later meetings, we all coded together and collaborated in areas that represented our individual strengths. We made our work as cohesive as possible by asking questions and ensuring that all members were following the same line of analysis and coding style.

Chris wrote the core functions of the *ars()* function and developed the updating plotting method for the simulation. He also pulled all of our final files into an R package.

Tom worked on the lower bound function, as well as vectorizing and debugging of *check.abscissae()* the (initial abscissae) function. Tom also wrote the test function for various known distributions and formatted the equations used in the paper in latex syntax.

Hairong helped work on the initial abscissae function (check.abscissae) used to create the starting x values for the ARS algorithm, and did preliminary work on the code for the sampling step. She also wrote .Rd help files for half of the functions in our code.

Brittney helped work on the initialization (*check.abscissae()*) function with Hairong, and wrote the other half of the .Rd help documentation. Additionally, she drafted the written documentation to submit for the project.

Note: The project was submitted to bspace by Christian Carmona.