Adaptive Rejection Sampling

Name: SID :
Name:SID:
Name:SID:
Name:Hao Lyu SID:26966632

December 16, 2015

1 Intoduction

By refering to the paper: "Rejection Sampling for Gibbs Sampling", we used the adaptive rejection sampling algorithm to simulate the samples for different log concave distributions. It mainly includes four different parts. First is Initialization, in this part we need to find the initial points for the grid bound: x1 and xk, then we generated the grid. Then we need to compute the upper hull and lower hull using the generated grid. After that we need to do the sampling part by using the density S_k which is defined by the upper hull and update our grid after sampling each point and we need to check the log concavity of our functions. Finally we will repeat the sampling step to get the number of samples we want. We also tried as much as we can to vectorize and implemented some tricks of generating a vector of samples but it is biased or inefficient compared to generating one point each time. We tested our results in four kinds of distributions and got the good results as shown in the tests part.
Zhenyuan Liu mainly focused on the organizing the algorithm of sampling,rejection sampling and the Initialization. Thibault Doutre focused on testing and the Initialization part of searching for x1 and xk. Hao Lyu focused on the searching for x1 and xk, checking for log concavity and the trick function of sampling. PAUL CHO focused on vectorzing the functions and doing the R package.
Github information: we worked on "gitpcho/stat243-finalPJ" for our editing and the final version is on "doutib/ars".

2 Initialization

In this section our main purpose is how to search for x1 and xk of our grid bound. We set the default values for x1 and xk to be null and search for a starting value x0 where h(x0) is finite in our search function. If the domain of the distribution is infinite, we will use this starting point to generates the x1 and xk by shifting x0 to the gradient direction in order to get x1 and xk whose signs of the gradients are different. If the domain of the distribution is bounded then we can use the bound to be our initial points. The trick here is that when it comes to chi square distribution or some distributions with small vatriance, the constant step will fail so we need ot build the automatically changed steps. We need to make our step smaller if it fails to find the reasonable x1,xk here.
For the upper hull, lower hull and envelop density, we just used the definitions in the paper.

3 Sampling and updating

We used the Inverse CDF method to generate the sample from our envelop desity S_k and do the two rejection tests to choose whether to keep the point or not. If we accept it we will use the update function to uppdate the grid again repeat until we get enough points.

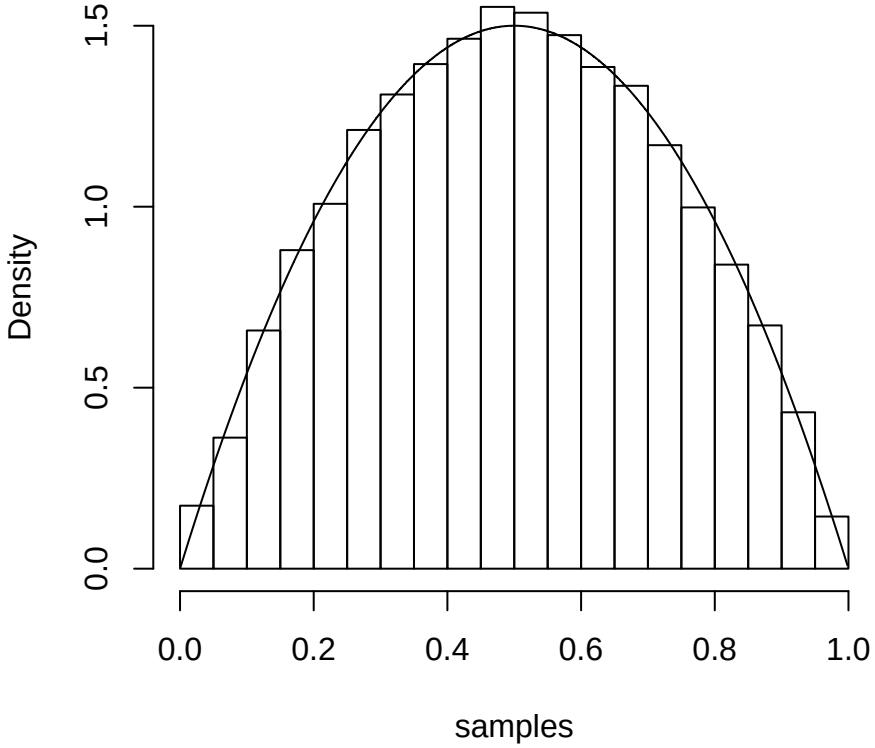
4 Main function

In our final version we just give the version of generatine one point each time we also tried generating several points each time to check whether it improves or not. If we generate m points each time and check squeezing test first and then let the rejected ones do the rejection test. Keep those pass the first tests or second. This algorithn will include more bias because it generates many samples based on one grid without updating it. If we generate m points then stop and drop the remainings while we came into a point which can not pass the squeezing test. In this case it will decrease the bias and increase the time for computing. So in this case we think the generating one by one is better than other tricks

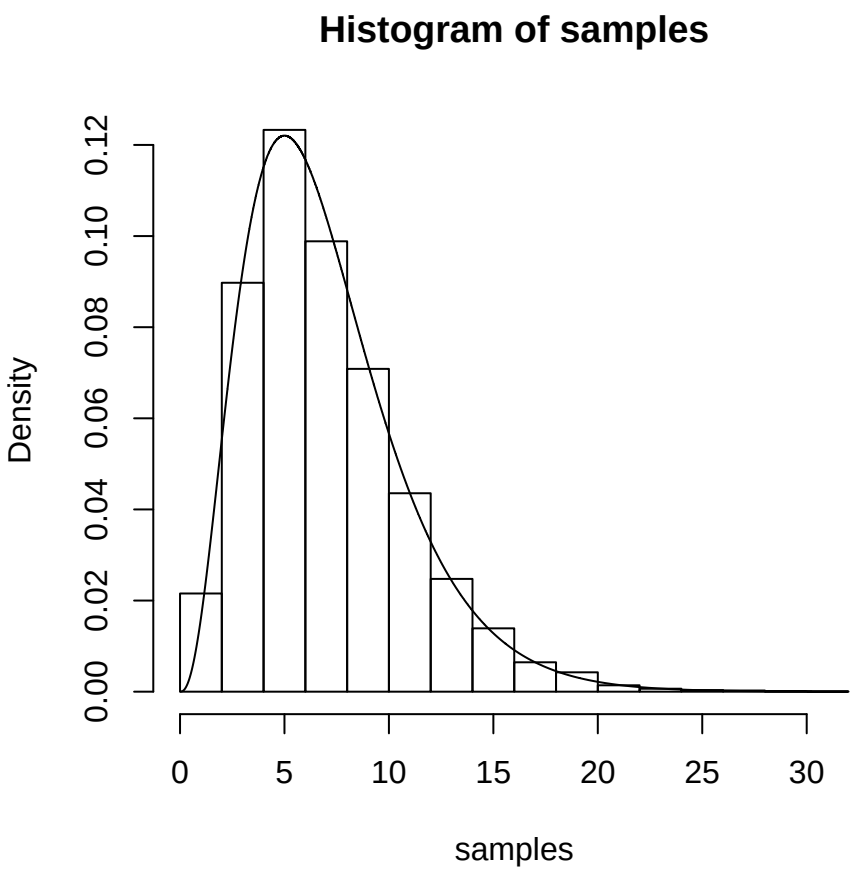
5 Tests and results

We used normla distribution,gamma, beta and chi square to test our samples and we campared the sample mean and variance with the expected ones. We also plot some histograms in which the line means the true value and the histgram is our samples to make a clearer comparison. Finally we used the KS tests to check whether the samples belongs to the true distribution and the larger p values here means better sampler in that we can not reject the assumption that the samples are from the true distribution we assigned.

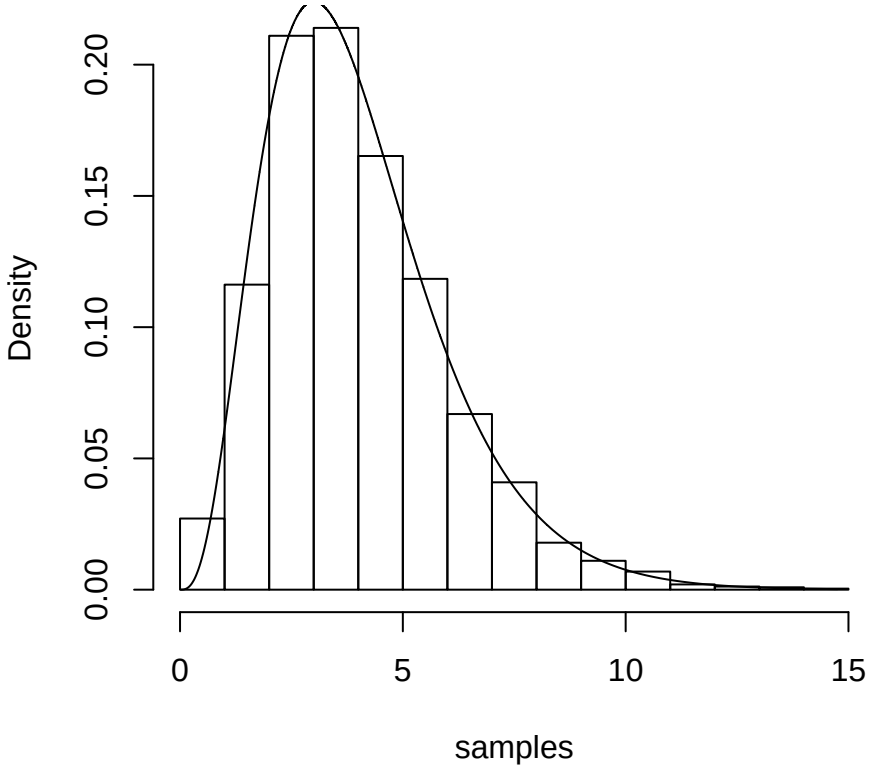
```
## [1] "/Users/haolyu/Desktop/sta243/ars"

## Loading required package:  testthat

##
## Computing test:  Beta(2,2).
```



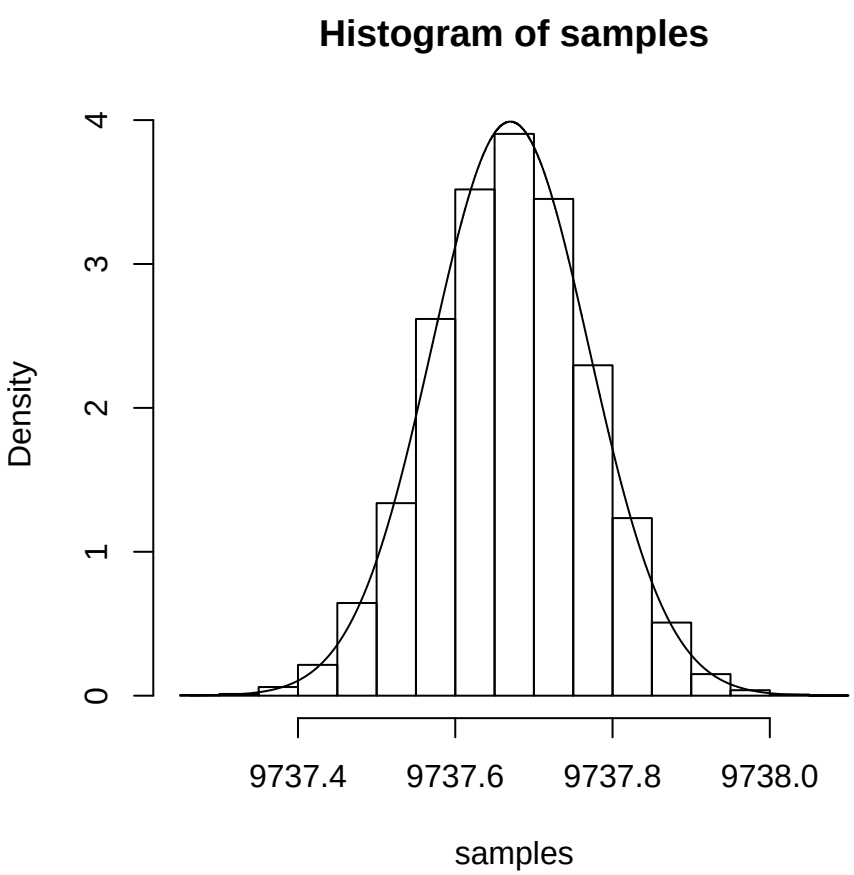

```
##
## Result:
## D =  0.005620696
## p-value =  0.9101855
## ------------
## Computing test:  chisq(7).
```




```
##
## Result:
## D =  0.01005637
## p-value =  0.2640061
## ------------..
## Computing test:  Gamma(shape=4).
```




```
##
## Result:
## D =  0.008759512
## p-value =  0.4267783
## ------------
## Computing test:  N(9737.67,0.1).
```




```
##
## Result:
## D =  0.007381251
## p-value =  0.6471818
## ------------
## DONE
```