Stat 243 Final Project

Yuchao Guo   Alanna Iverson   Auyon Siddiq   Qingyuan Zhang
github:amandazhang

December 17, 2015

The approach we took to implement an adaptive rejection sampling algorithm, unsurprisingly, began with and mirrored the Gilks paper. The first task was to understand what our algorithm should do, and then begin to use R to implement a solution. To accomplish this required some manipulation of the given formulas to allows us to use the inverse CDF method in order to get the $x*$ values. Below are the calculations we used and then implemented in R:

First thing we need to do is to calculate the area under different $Upper$ functions. Since $Upper$ is a piece-wise function, we have to calculate the areas piece by piece. For the jth piece, we can calculate each piece by the following common expressions for $Upper$ by using the equation for z_j below:

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

$$ExpUpper[j] = \int_{z_{j-1}}^{z_j} exp(h(x_j) + (x - x_j)h'(x_j)) \tag{2}$$

After some simple algebra, we can get the final expression as follows:

$$\frac{exp(h(x_j) - x_j h'(x_j))}{h'(x_j)} \cdot (exp(z_j h' x_j) - exp(z_{j-1} h'(x_j))) \tag{3}$$

We can find each piece of the area by changing different js. In our method, we use this expression to give us a vector that stores all the pieces and calculates a cumulative sum of this vector to stand for the cumulative area sums.

For the sampling process, we can use the inverse CDF method. But for this question, since the CDF is a piece wise function, we need to figure out which section the random number falls in. Then we should take the previous areas out of the overall CDF and use an inverse CDF method to draw samples in the section we are in.

More explicitly, the inverse CDF method first gets a uniform random number from 0 to 1 and then multiplies this number by the sum of the area, matching the total area. For simplicity, we call the $\int_{ub}^{lb} exp(u_k(x'))dx'$ as c, where c is a constant. Then we call the random number we generated as u as below:

$$uc - cum[j] = \int_{z_j}^{x} exp(h(x_j) + (x^* - x_j)h'(x_j))dx \tag{4}$$

We can easily calculate the $x*$ values as x_all:

$$x^* = \frac{1}{h'(x_j)} \cdot log(e^{(h'(x_j))} + (\frac{(uc - cum[j])h'(x_j)}{e^{h(x_j)} - x_j h'(x_j)})) \tag{5}$$

Once we had all the equations we needed, we began implementing them in R. In order to keep the code modular, each formula was created as a single function that we could call within the final ars algorithm. The functions we wrote based on the Gilks paper are $ConstructZ$ function, $Envelope$ function (S function from the paper), $Upper$ function (U_k function from the paper), $ExpUpper$ (exponential Upper function) and $Lower$ function (l_k function from the paper). We chose to write the code modularly for a a few reasons: to increase the readability of the code, to decrease probability of bugs and to follow general best practices.

In order to handle the different inputs, and verify the ars function is used properly, we also added a few checks at the beginning of the function to check for the uniform and exponential density cases, since these can be more difficult to handle than other densities. First, we find the maximum of the log of the input density using the optimize function. We also find the mode of the function if the user does not provide initial x values (x_init) in order to initialize the starting values. After running these initial values through the various smaller functions, we use GenCandidates to get some initial values of $x*$ to then pass to the RejectionTest function. This Rejection function will determine which $x*$ values to keep in the final sample and which to throw out based on the two inequalities in Gilks. Finally, we place the entire process into a while loop that ends when we hit the requisite number of $x*$ values in the returned samples.

With our main ars function completed, we moved on to informal, formal and unit testing. We informally tested our algorithm and the functions used within it. One way we found very helpful in making sure we were actually calculating what we needed was by graphing samples of results along the way. Doing this allowed us to catch several indexing errors as well some other bugs we had missed.

To formally test our algorithm, we decided to use the Kolmogorov-Smirnov Test (KS Test) to measure the maximum difference between the empirical CDF we got from our algorithm and the theoretical CDF (a known truth). We set tolerances to be relatively low to ensure that passing was only possible if our algorithm worked correctly. Our formal tests include 7 continuous distributions that are easily called within R:

Normal
Gamma
Uniform
Beta
Chi-Square
Exponential
Weibull

Our tests also report the KS statistic for each distribution tested in order to see relative accuracy among them.

Our unit tests cover the *ConstructZ* function, *logconc* function, $findmax$ function, *utest* function, and the *Upper* and *Lower* functions. In order to test these, we used known truths about the function output and tested them with the actual function output. If the two were essentially equal (with a low tolerance) then we can assume our functions work as intended. For example, to test the *utest* function, which detects whether the input density is uniform, we made sure the output was TRUE when the input was infact uniform and FALSE otherwise. Since the *GenCandidates* and *Rejection* functions are the main engines behind the *ars* algorithm, the formal tests essentially serve as their unit tests. It is also very difficult to come up with known truths about the output of these functions by themselves, and not in the context of the algorithm, which also contributed to our decision not to use the formal tests as their unit testing.

Finally, the individual team members contributed as follows:
Yuchao: Yuchao contributed to the projects with the main functions build-up. Read the paper to convert the algorithm into R codes. Wrote out the small function that the main function would call. Wrote the main function drafts. Also, did some unit tests for the small functions.

Auyon: Auyon contributed functions to the main sampling algorithm, performed debugging and wrote the code for computing KS statistics to compare the empirical distributions produced by the ars() function to Rs built in sampling functions.
Qingyuan: Qingyuan contributed to the project with the discussion of the algorithm listed in the reference paper and debugging as the algorithm was implemented using R code. She also helped with the unit test and contributed to the write up of the final paper.
Alanna: Alanna contributed to the project with the discussion of the approach to the algorithm, code review as the algorithm was being constructed and bug fixes throughout. She also helped write some of the informal tests/checks within the algorithm (check for log concavity for example) as well as the unit tests. Finally, she contributed to the final project output by writing the the first draft of the algorithm overview as well as placing the ars function and its tests into an R package.

The github username that holds the final .tar.gz file is: amandazhang (https://github.com/amandazhang)

Below we do a quick test to show the output of the *Upper* and *Lower* functions we have created both graphically and by printing results.

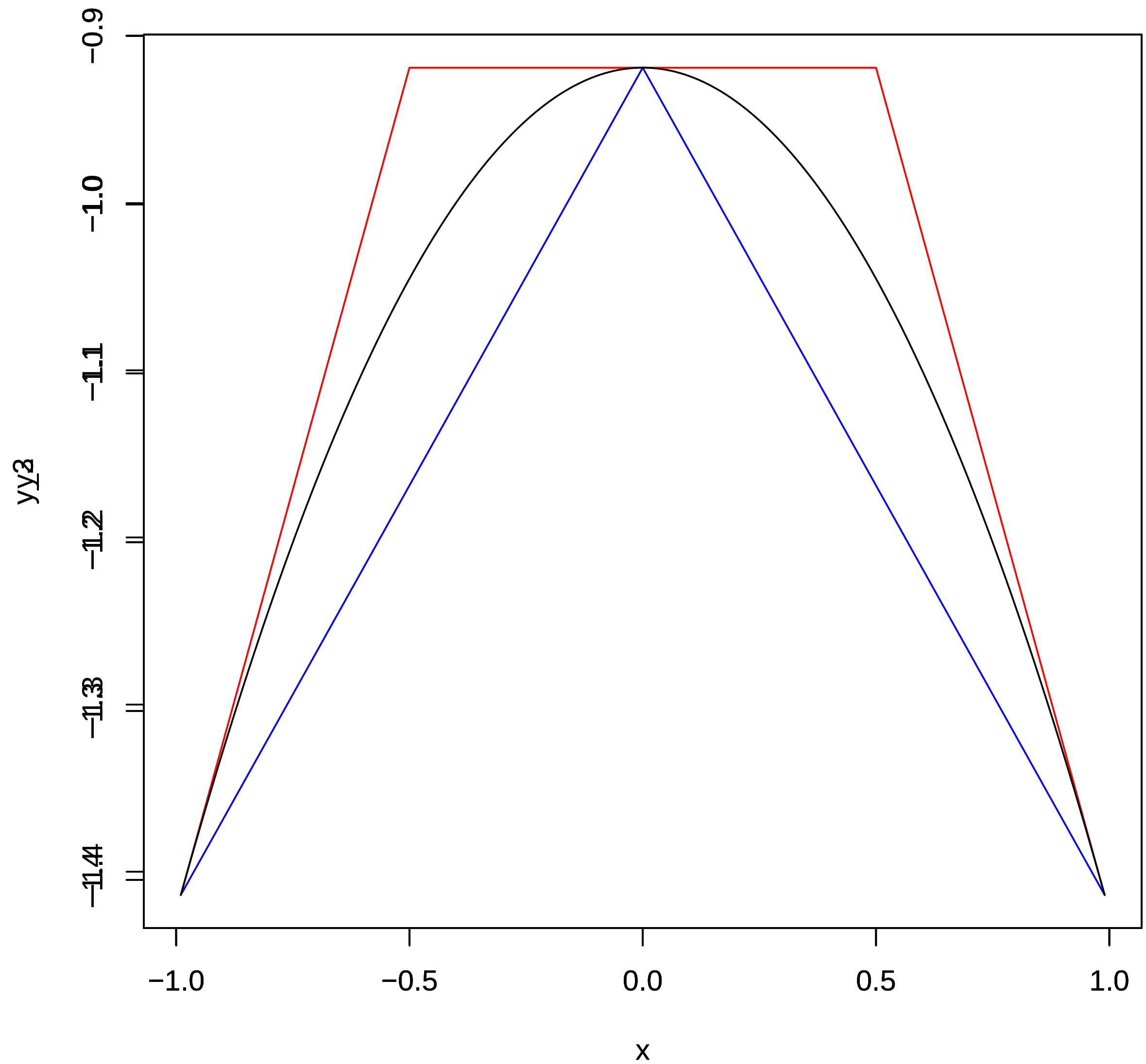
```
## [1] "Values of the upper hull below:"
##   [1] -1.4089385 -1.3989385 -1.3889385 -1.3789385 -1.3689385 -1.3589385
##   [7] -1.3489385 -1.3389385 -1.3289385 -1.3189385 -1.3089385 -1.2989385
##  [13] -1.2889385 -1.2789385 -1.2689385 -1.2589385 -1.2489385 -1.2389385
##  [19] -1.2289385 -1.2189385 -1.2089385 -1.1989385 -1.1889385 -1.1789385
##  [25] -1.1689385 -1.1589385 -1.1489385 -1.1389385 -1.1289385 -1.1189385
##  [31] -1.1089385 -1.0989385 -1.0889385 -1.0789385 -1.0689385 -1.0589385
##  [37] -1.0489385 -1.0389385 -1.0289385 -1.0189385 -1.0089385 -0.9989385
##  [43] -0.9889385 -0.9789385 -0.9689385 -0.9589385 -0.9489385 -0.9389385
##  [49] -0.9289385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [55] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [61] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [67] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [73] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [79] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [85] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [91] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
##  [97] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [103] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [109] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [115] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [121] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [127] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [133] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [139] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [145] -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385 -0.9189385
## [151] -0.9289385 -0.9389385 -0.9489385 -0.9589385 -0.9689385 -0.9789385
## [157] -0.9889385 -0.9989385 -1.0089385 -1.0189385 -1.0289385 -1.0389385
## [163] -1.0489385 -1.0589385 -1.0689385 -1.0789385 -1.0889385 -1.0989385
## [169] -1.1089385 -1.1189385 -1.1289385 -1.1389385 -1.1489385 -1.1589385
## [175] -1.1689385 -1.1789385 -1.1889385 -1.1989385 -1.2089385 -1.2189385
## [181] -1.2289385 -1.2389385 -1.2489385 -1.2589385 -1.2689385 -1.2789385
## [187] -1.2889385 -1.2989385 -1.3089385 -1.3189385 -1.3289385 -1.3389385
## [193] -1.3489385 -1.3589385 -1.3689385 -1.3789385 -1.3889385 -1.3989385
## [199] -1.4089385
```

```
## [1] "Values of the lower hull below:"
##   [1] -1.4139385 -1.4089385 -1.4039385 -1.3989385 -1.3939385 -1.3889385
##   [7] -1.3839385 -1.3789385 -1.3739385 -1.3689385 -1.3639385 -1.3589385
##  [13] -1.3539385 -1.3489385 -1.3439385 -1.3389385 -1.3339385 -1.3289385
##  [19] -1.3239385 -1.3189385 -1.3139385 -1.3089385 -1.3039385 -1.2989385
##  [25] -1.2939385 -1.2889385 -1.2839385 -1.2789385 -1.2739385 -1.2689385
##  [31] -1.2639385 -1.2589385 -1.2539385 -1.2489385 -1.2439385 -1.2389385
##  [37] -1.2339385 -1.2289385 -1.2239385 -1.2189385 -1.2139385 -1.2089385
##  [43] -1.2039385 -1.1989385 -1.1939385 -1.1889385 -1.1839385 -1.1789385
##  [49] -1.1739385 -1.1689385 -1.1639385 -1.1589385 -1.1539385 -1.1489385
##  [55] -1.1439385 -1.1389385 -1.1339385 -1.1289385 -1.1239385 -1.1189385
##  [61] -1.1139385 -1.1089385 -1.1039385 -1.0989385 -1.0939385 -1.0889385
##  [67] -1.0839385 -1.0789385 -1.0739385 -1.0689385 -1.0639385 -1.0589385
##  [73] -1.0539385 -1.0489385 -1.0439385 -1.0389385 -1.0339385 -1.0289385
```

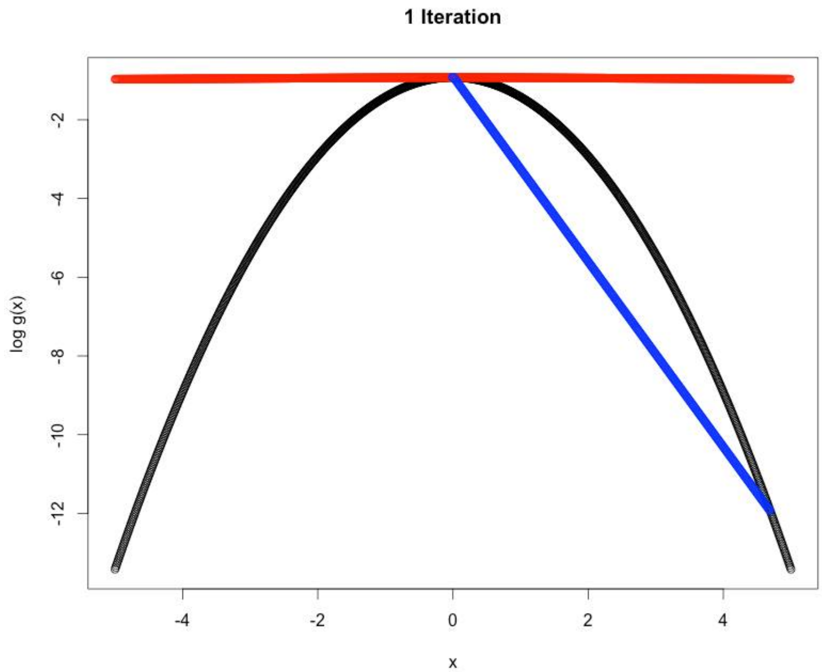
```
##  [79] -1.0239385 -1.0189385 -1.0139385 -1.0089385 -1.0039385 -0.9989385
##  [85] -0.9939385 -0.9889385 -0.9839385 -0.9789385 -0.9739385 -0.9689385
##  [91] -0.9639385 -0.9589385 -0.9539385 -0.9489385 -0.9439385 -0.9389385
##  [97] -0.9339385 -0.9289385 -0.9239385 -0.9189385 -0.9239385 -0.9289385
## [103] -0.9339385 -0.9389385 -0.9439385 -0.9489385 -0.9539385 -0.9589385
## [109] -0.9639385 -0.9689385 -0.9739385 -0.9789385 -0.9839385 -0.9889385
## [115] -0.9939385 -0.9989385 -1.0039385 -1.0089385 -1.0139385 -1.0189385
## [121] -1.0239385 -1.0289385 -1.0339385 -1.0389385 -1.0439385 -1.0489385
## [127] -1.0539385 -1.0589385 -1.0639385 -1.0689385 -1.0739385 -1.0789385
## [133] -1.0839385 -1.0889385 -1.0939385 -1.0989385 -1.1039385 -1.1089385
## [139] -1.1139385 -1.1189385 -1.1239385 -1.1289385 -1.1339385 -1.1389385
## [145] -1.1439385 -1.1489385 -1.1539385 -1.1589385 -1.1639385 -1.1689385
## [151] -1.1739385 -1.1789385 -1.1839385 -1.1889385 -1.1939385 -1.1989385
## [157] -1.2039385 -1.2089385 -1.2139385 -1.2189385 -1.2239385 -1.2289385
## [163] -1.2339385 -1.2389385 -1.2439385 -1.2489385 -1.2539385 -1.2589385
## [169] -1.2639385 -1.2689385 -1.2739385 -1.2789385 -1.2839385 -1.2889385
## [175] -1.2939385 -1.2989385 -1.3039385 -1.3089385 -1.3139385 -1.3189385
## [181] -1.3239385 -1.3289385 -1.3339385 -1.3389385 -1.3439385 -1.3489385
## [187] -1.3539385 -1.3589385 -1.3639385 -1.3689385 -1.3739385 -1.3789385
## [193] -1.3839385 -1.3889385 -1.3939385 -1.3989385 -1.4039385 -1.4089385
## [199] -1.4139385
```

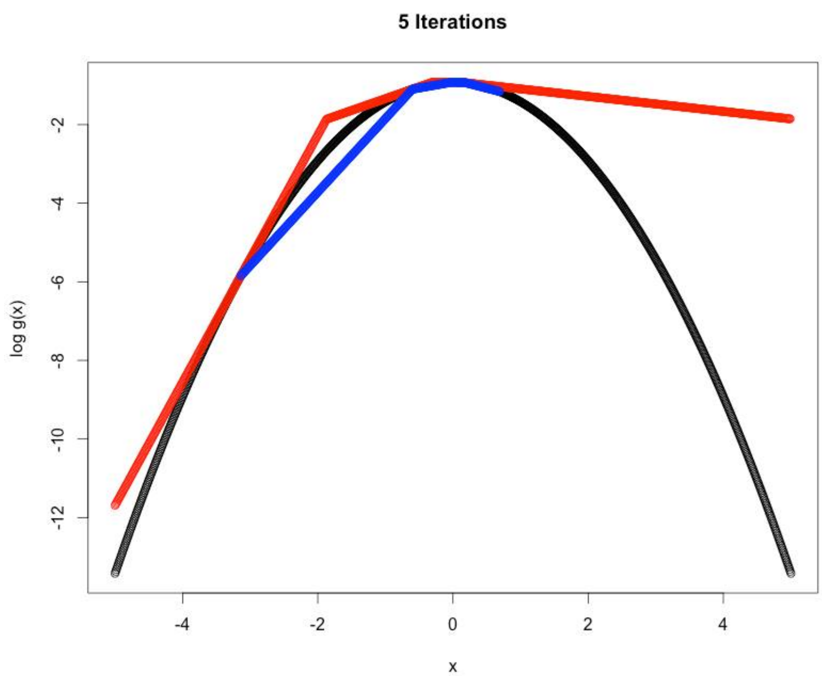
The upper hull created by the *Upper* function graph is the red line below, the lower hull created by the *Lower* function graph is the blue line below and the given theoretical distribution we are to draw samples from is the black line below. You can clearly see by the graph below that the theoretical curve is bounded by the *Upper* and *Lower* functions as desired.

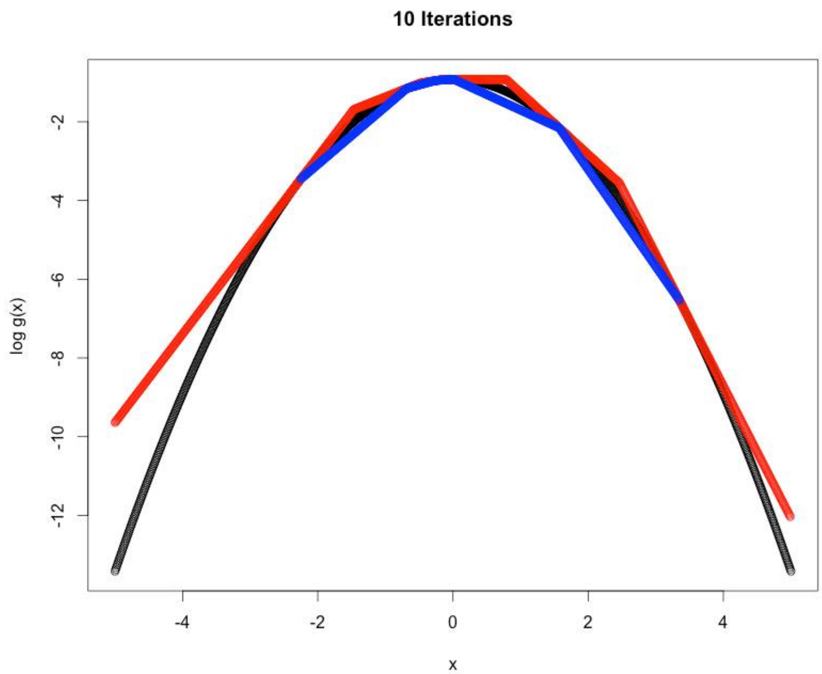
```
## [1] "Graph of upper hull = red, lower hull = blue, true curve = black"
```

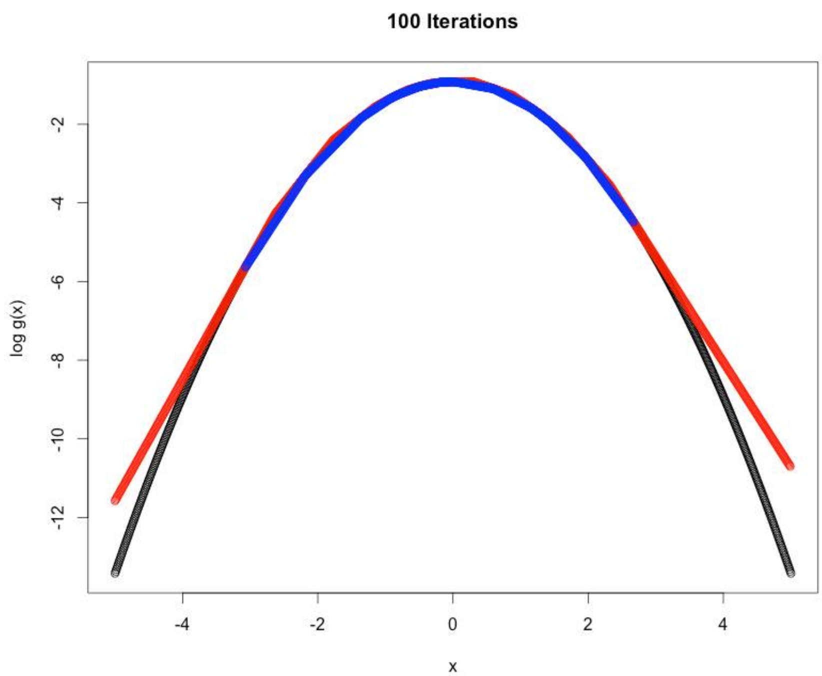

−0.9
−1.0
−1.1
−1.2
−1.3
−1.4
yy2
−1.0
−0.5
0.0
0.5
1.0
x

The plots below visualizes the algorithm by showing the standard normal, the lower bound and the upper bound as the algorithm progresses from 1 to 100 iterations (with each iteration consisting of 1 sample).






10 Iterations
log g(x)
-2
-4
-6
-8
-10
-12
-4
-2
0
2
4
x


100 Iterations
log g(x)
-2
-4
-6
-8
-10
-12
-4
-2
0
2
4
x