Stat 243: Final Group Project

Sicun Huang, Cheng Ju, Fengshi Niu, Yun Zhou
(in alphabetical order)

December 17, 2015

Please find the final version of the project in Cheng Ju's (https://github.com/jucheng1992/stat243final) Github repository .

1 Overview

The goal of this project is to implement an adaptive-rejection sampler for a log-concave function $f(x)$ with domain D using the tangent approach. The adaptive-rejection sampling method is especially effective when function $g(x) = cf(x)$ is costly to evaluate. To reduce the number of direct computions of $g(x)$, we introduce an envelope function (i.e. the upper hull) and a squeeze function (i.e. the lower hull) and use them to construct tests to determine whether or not to accept a sample point x. After each test, we update the upper and lower hulls to further lower the probability of needing to evaluate $g(x)$ in the later tests. Repeat the above steps until we get a sample of n points.

2 Functions

2.1 Primary function ars to implement the simulation

Given a log-concave function $g(x)$ with $g(x) = cf(x)$ for some possibly unknown c, function ars returns a sample of size n.

We start by computing $h(x)$, the *log* of $g(x)$. Then, use function $findInitAbsc$ to find the abscissae matrix comprised of the initial values of the abscissae $x_1, ...x_k$ (in ascending order), together with their corresponding $h(x)$'s and $h'(x)$'s.

From these values, we derive the unique upper and lower hull functions for $h(x)$ with functions $upperHull$ and $lowerHull$. To vectorize the following procedures, we introduce a variable *stepsize* to denote the number of samples generated in each iteration; it increses with each iteration to accommodate for the rise in acceptance rate as test proceeds. With the *sampleUpper* function, we generate *stepsize* many samples from the upper hull function; compute the values of upper and lower hulls at these sample points on the original (i.e. not log) scale (denoted by g_u and g_l). We check for log-concavity of $g(x)$ by making sure that $g_u > g_l$. Then, generate *stepsize* many $U(0, 1)$ random variables w's and perform the following tests for each point:

- (1) If $w \leqslant \frac{g_l}{g_u}$, then accept x. If all x's are accepted, add them to the final samples of interest; otherwise, proceed to test (2).
- (2) Discard the x's that come after the first rejected x. Evaluate $h(x)$ and $h'(x)$ for the first rejected x. If $w \leqslant \frac{exp(h(x))}{g_u}$, then accept. Add it back into the vector of accepted x's and append the vector to the final samples.
- (3) If all abscissae have zero derivatives, then the given distribution is very likely to be uniform. Thus, we will sample directly from the uniform distribution.

Use function *updateAbscissae* to update the abscissae matrix and repeat the above steps until final sample reaches the predetermined size n.

2.2 Function $findInitAbsc$ to construct the initial abscissae matrix

Given a log-concave function $g(x) = cf(x)$, the maximum size of the abscissae k, and the left and right bounds of the interval on which function g is defined, function $findInitAbsc$ returns a matrix comprised of the abscissae $x_1, ..., x_k$ (in ascending order) and their corresponding $h(x)$'s and $h'(x)$'s, where $h(x) = log(g(x))$.

First, we use the helper function $findMode$ to find the mode of function h. Notice that since the *optim* function returns the minimum by default, we need to plug in -$h(x)$. If function h is defined on $-\infty$ to ∞, we choose 0.5 (near center of the domain) to be the starting value (see explainations below), then use the *optim* function with the general BFGS method to find the mode. If only one of the bounds is finite, we start near the finite bound; if both bounds are finite, we start at the middle point of the specified interval. Then we use the *optim* function with the L-BFGS-B method to solve for the mode. To avoid complications when g is distributed chi-squared, we set the upper and lower bounds to be right bound - 1e-15 and left bound + 1e-10 respectively. Notice that the global supremum can be outside of or on the boundaries of the given interval; in that case, we return the result with a warning.

Then, we proceed to construct the abscissae. If function is defined on $-\infty$ to ∞, then the mode is guaranteed to be within the given interval. So we sample from both sides of the mode to obtain the desired abscissae with $h'(x_1) > 0$ and $h'(x_k) < 0$. If only one of the bounds is finite, we sample from the finite bound side up/down to include the mode (if mode is in the provided interval) or until we have k points (if mode is not in the provided interval). If both bounds are finite, we sample with equal distance from the left bound to the right bound regardless of whether the mode is in the interval or not. With these k abscissae, we compute the $h(x)$'s and $h'(x)$'s, leaving out the x's with infinite $h(x)$ or $h'(x)$ values. Return the results as a matrix.

Caveats:

- When g is distributed chi-squared with a small degree of freedom, to avoid error with the *optim* function, we need to make sure that the starting value we plug in is not zero; if an interval is given, we use the left bound plus a small value and the right bound minus a small value as the lower and upper bounds instead of the provided values.
- When the global supremum of h is outside of or on the boundaries of the interval provided, the abscissae computed from the above method may not satisfy $h'(x_1) > 0$ and $h'(x_k) < 0$. Although it will still work in the later calculations.
- When sampling from the given interval, stay away from the very edge to avoid the case where function is not defined on the boundaries.
- It is possible for $h(x)$ and $h'(x)$ to be infinite; exclude those results to ensure that the abscissae matrix is suitable for later computations.
- The abscissae vector needs to have at least two elements to be useful in the later calculations. If the resulted abscissae vector contains less than two points after deleting the ones with infinite $h(x)$'s or $h'(x)$'s, user may need to choose a larger k value and try again.

2.3 Function *computeZ* to find intersections of the tangent lines

Given the abscissae matrix, function *computeZ* returns the intersections of the tangent lines (i.e. the kinks in the upper hull). The formula used in this function can be found in the paper by Gilks et al.

Caveats:

- If all derivatives $h'(x)$'s are equal, then function $h(x)$ is not strictly concave; we return a warning message as this contradicts the assumption made in Gilks et al.'s paper. However, with a user defined support interval, our package can still handle this case.

- Some functions can be locally constant (i.e. not strictly concave), which will result in infinite z's (x-coordinates for intersections of the tangent lines). In this case, we remove all infinite values from the z vector, since two of the same tangent line provide no extra information.

2.4 Functions *upperHull*/*lowerHull* to find the upper/lower hull

Given a vector x, functions *upperHull*/*lowerHull* return the corresponding values of upper/lower hull that are uniquely defined by the abscissae matrix.

- For upper hull: First, we determine the intersections of the tangent lines $z_j, j = 1, 2...k-1$ where z_j is the intersection between support points x_j and x_{j+1}. We use *index* to represent position in the input vector x, e.g. index[5] = 3 means x[5] $\in$ [z[3 − 1], z[3]]. After knowing the interval an x falls in, we can use the corresponding piecewise linear formula to calculate its upper hull value.

- For lower hull: Similar to upper hull, we denote position in the input vector by *index*, e.g. index[5] = 3 means x[5] $\in$ [$\mathbf{x}$[3], $\mathbf{x}$[3 + 1]] with $\mathbf{x}$ being the vector of support points and x being the input vector. After knowing the interval an x falls in, we can use the corresponding piecewise linear formula to calculate its lower hull value.

- Vectorization: To make calculations more efficient, we construct vectors *tempX*1 and *tempX*2 from the original abscissae matrix, where *tempX*1 is all support points except the first one and *tempX*2 is all support points except the last one. Then substitute them into the formula.

Problems encountered:

- In the *upperHull* function, when the log density $h(x)$ is uniform or piecewise constant, we have infinite intersection z values since we are dividing by 0. To see this, when the derivatives are the same at two adjacent support points, the tangent lines at the two points overlap, which result in no specific (or infinitly many) intersections. To solve this problem, we set the intersection between support points $x[i]$ and $x[i+1]$ to be $z[i] = (x[i] + x[i+1])/2$. Note that fraction $\frac{1}{2}$ is chosen arbitrarily because the intersection lies anywhere between $x[i]$ and $x[i+1]$. (See test for log piecewise constant distribution in *Appendix 1*)

- In the *lowerHull* function, when a value of the input vector falls outside of the support interval, the corresponding lower hull value is negative infinity; this will cause an index overflow. To avoid this issue, we set the input values that are outside of the given interval (positions denoted by *mark*) to be in the interval [$\mathbf{x}$[1], $\mathbf{x}$[2]], where $\mathbf{x}$ is a vector of the support points. This will enable us to compute all the lower hull values. We then change the lower hull values in the $mark^{th}$ position back to $-\infty$.

- In the test function *testUpLowLogDensity*, we need to make sure that $h(x)$, upper hull, and lower hull have the same value at support points since tangent line and secant line coincide with function curve at those points. Choose 1000 random points in the support interval, and test whether or not the corresponding upper/lower hull values satisfy $upperHull \geq functionh \geq lowerHull$. If not, function h may not be log concave. The test cases we chose to include are Normal, Chisquare, Uniform, Piecewise constant, Exponential, and Beta distributions. (See *Appendix 1* for a few examples)

- In the case where the distribution is truncated, we use the *abscissaeSummary* function to find the adjusted range of x. Note that due to the precision problem in R, we need to be careful when comparing values of the three functions. We set the comparision precision level to be 1e10-5, which can be later adjusted in the test function.

2.5 Function *sampleUpper* to sample from the upper hull

To sample from the envelope function $s(x)$ (i.e. the upper hull), we pick a random sample from the uniform distribution and transform it with the standard inverse CDF method (explained below).

We begin by generating the normalized density $exp(s(x))$, a piece-wise exponential function, from $s(x)$. In

Gilks et al.'s paper, the left and right end points of the initial abscissae have positive and negative derivatives respectively. If the initial abscissae satisfy such a condition, we can solve for the inverse CDF analytically. However, we also want to deal with cases where the derivatives $h'(x)$'s are all positive or all negative over the support interval (e.g. $exp(1)$ or truncated normal). In these cases, $exp(s(x))$ diverges on the left or the right end. A workaround for this issue is to ask the user to provide the domain of $h(x)$ and set the support of $exp(s(x))$ to be the given interval. The calculations are a little bit different for the left and right end cases; we will not discuss the difference in detail here. In the end, we get the analytic expression of the inverse CDF.

The main idea for testing the inverse CDF is: for any distribution, as the range and number of elements in the abscissae increase, the envelope function grows closer to the true $h(x)$. As a result, the estimated inverse CDF gets closer to the true inverse CDF; in the limit, it converges to the true inverse CDF. Therefore, we graph the inverse CDF's derived from the envelope function and from the true distribution. If they seem to converge on the plot, then the inverse CDF passes the test. We conducted this test for exponential, chi-square (with df = 5), truncated normal, and several other distributions. The graph for truncated chi-square distribution with df = 5 is shown in $Appendix 2$.

To test the accuracy of results produced by the random sampling function, we plot the empirical CDF of a random sample with size n (generated from the envelope function) and the true CDF (constructed with a different set of initial values). If they converge on the graph as range and number of elements in the abscissae increase, the function for random sampling passes the test. We performed the test on Cauchy, chi-square, truncated normal, and some other distributions. The graph for truncated chi-square distribution with df = 5 is shown in $Appendix 3$. The figure shows that our sampling function works as expected.

2.6 Function *updateAbscissae* to update the abscissae matrix

After each iteration, if we evaluated any h values (e.g. in the rejection step), we want to utilize them in the future tests. For an updated set of abscissae x, we compute $h(x)$ and $h'(x)$, then insert them into the abscissae matrix; check that the abscissae is in increasing order with some simple tests.

Note:

- In the updating step, we can update lower/upper hull and inverse CDF locally. However, this does not save much time, as we are only evaluating functions that are piecewise linear or exponential in these functions.
- To avoid potential bugs, we only update the abscissae matrix, and use the updated matrix as input for related functions in the following iteration.

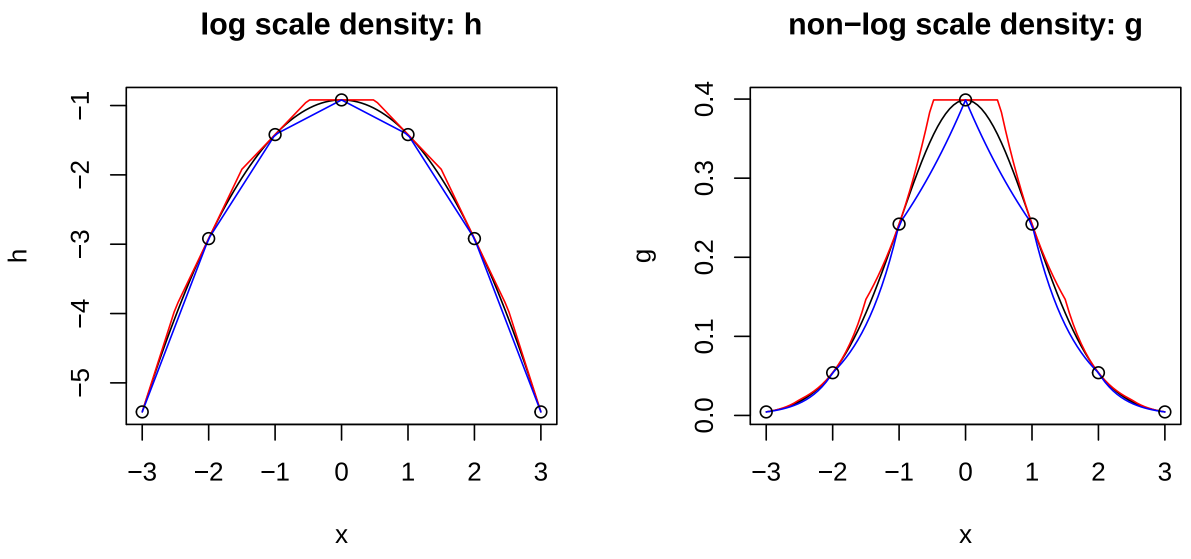
3 Team member responsibilities

- Sicun Huang (sicihuang): Write function $findInitAbs$ and its helper functions to construct the initial abscissae matrix. Write overview, ars, and findInitAbs sections in this report. Aggregate and edit report. Write and edit documentations for the package.
- Cheng Ju (jucheng1992): Write the main adaptive rejection sampling function *ars* and some helper functions (e.g. *computeZ*). Write test cases for the *ars* function and other helper functions. Write report on helper and *ars* functions. Build and test the R package.
- Fengshi Niu (FengshiNiu): Write and test the function $upperCDFInverse$ and *sampleUpper* to generate random variables from upper hull.
- Yun Zhou (YZhouEntheos): Write functions *upperHull*, *lowerHull* and their helper functions to calculate bound values for arbitrary input vectors, *testUpLowDensity* to inspect different cases and plot their graphs. Write test functions, documentation and report for above functions.

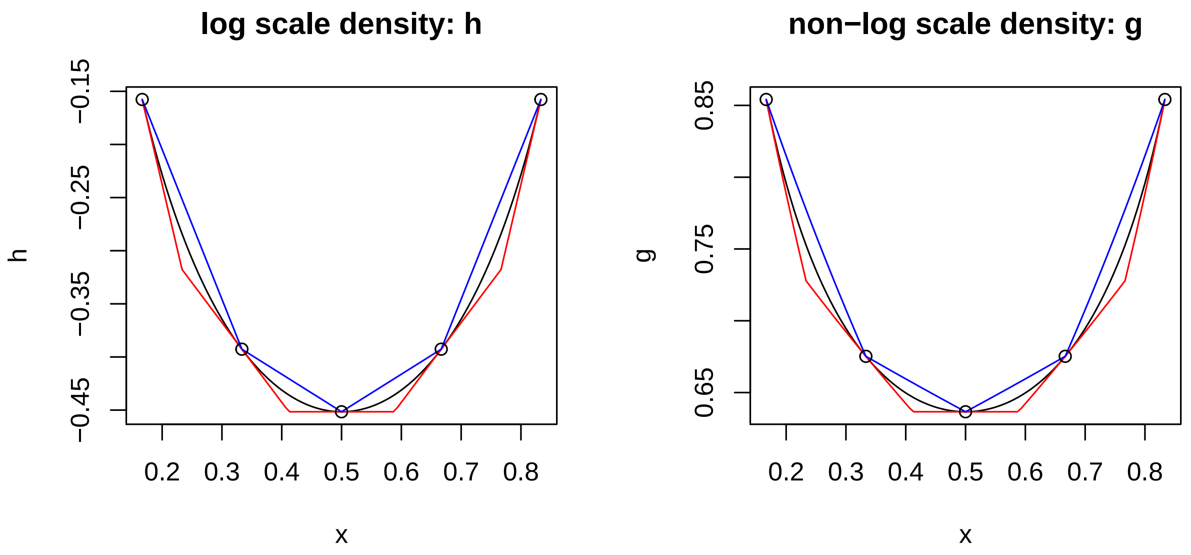
4 Appendices

4.1 Appendix 1: tests

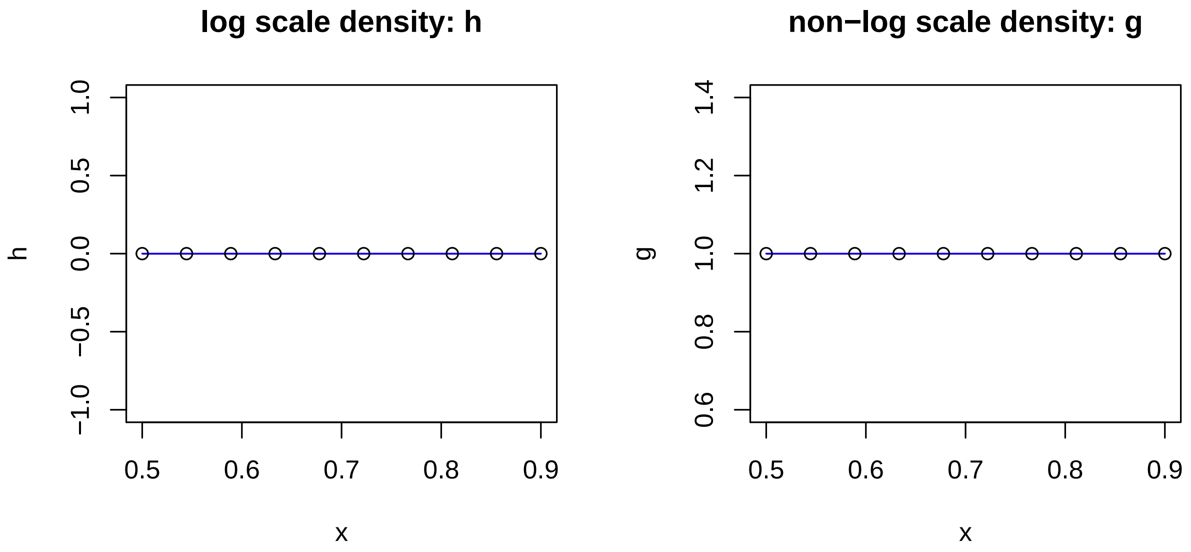
```
## [1] "Test: log of normal distribution"
## [1] "Test passed: h function, upperhull, and lowerhull have the same value at support points"
## [1] "Test passed: upper hull always lie above the h function"
## [1] "Test passed: lower hull always lie below the h function"
## [1] "All tests passed"
## [1] "The graph is: "
```



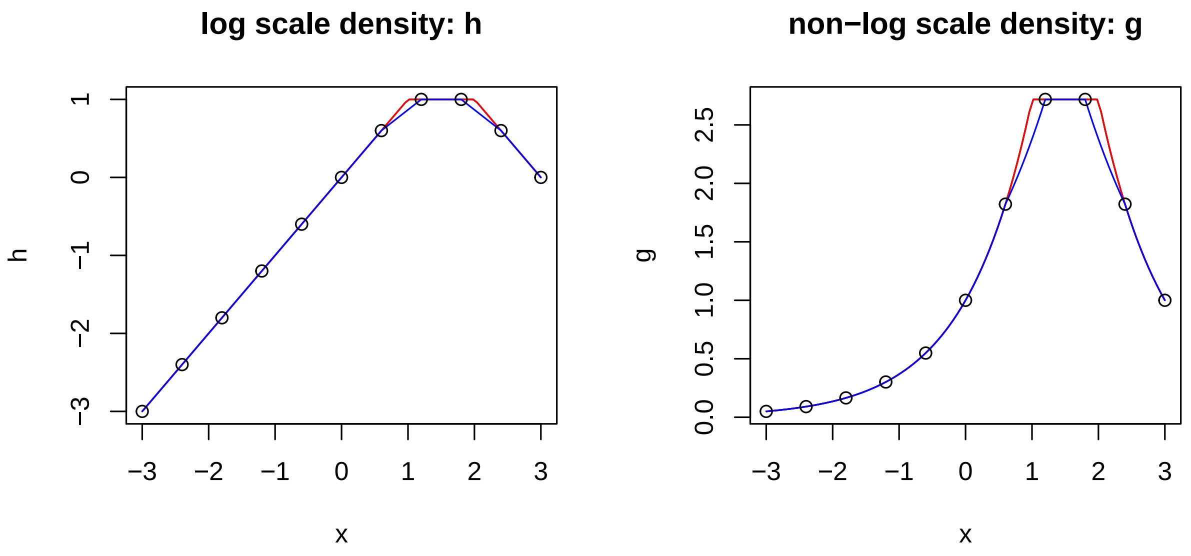
```
## [1] "Test: log of beta distribution"
## [1] "Test passed: h function, upperhull, and lowerhull have the same value at support points"
## [1] "Test failed: upper hull does not always lie above the h function; h function may not be log-conc
## [1] "Test failed: lower hull does not always lie below the h function; h function may not be log-conc
## [1] "Some tests did not pass"
## [1] "The graph is: "
```



```
## [1] "Test: log of uniform distribution"
## [1] "Test passed: h function, upperhull, and lowerhull have the same value at support points"
## [1] "Test passed: upper hull always lie above the h function"
## [1] "Test passed: lower hull always lie below the h function"
## [1] "All tests passed"
## [1] "The graph is: "
```



```
## [1] "Test: log of piecewise-constant distribution"
## [1] "Test passed: h function, upperhull, and lowerhull have the same value at support points"
## [1] "Test passed: upper hull always lie above the h function"
## [1] "Test passed: lower hull always lie below the h function"
## [1] "All tests passed"
## [1] "The graph is: "
```



4.2 Appendix 2

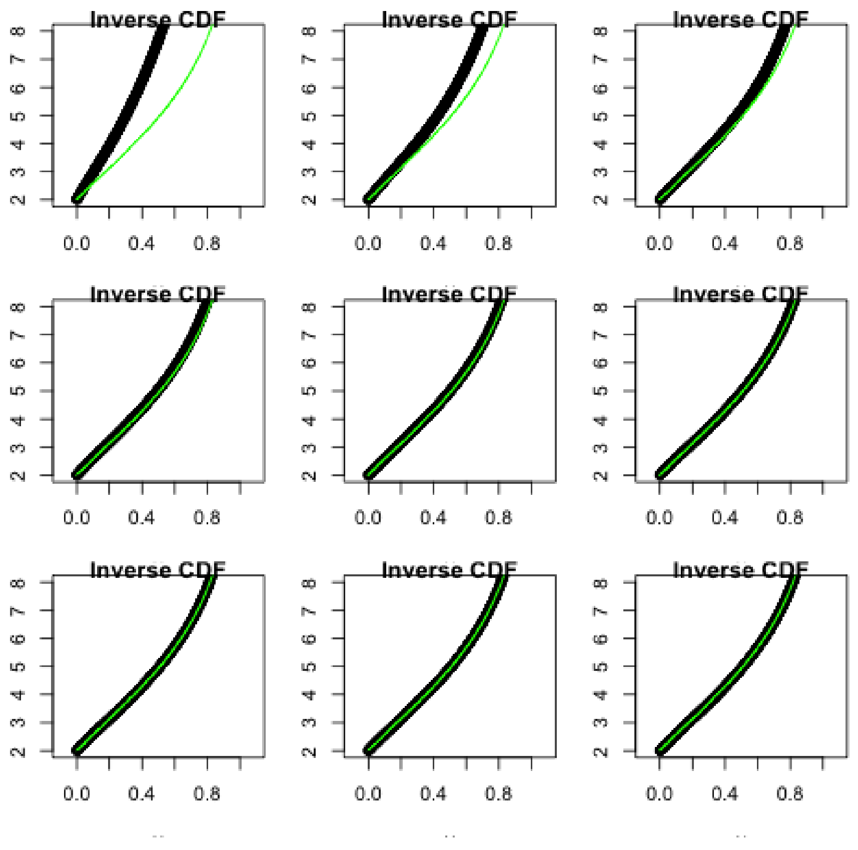


Figure. Inverse CDF's generated from the envelope function and the true distribution chisq 5

4.3 Appendix 3

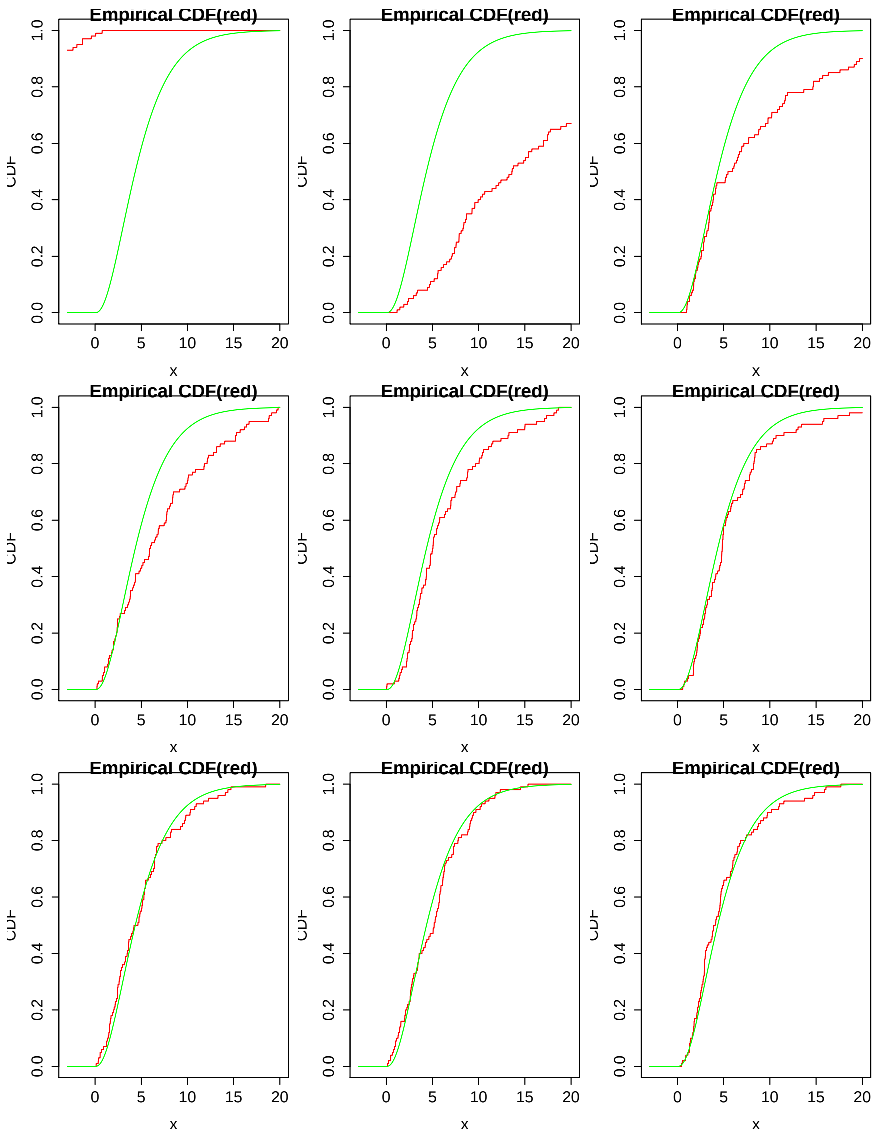


Figure. The empirical CDF (generated from the envelope function) and the true CDF