STAT 243 Final Project: Adaptive Rejection Sampling Package

Mengfei Jiang, Ji Wang, Alexander Samuel Fred Ojala, Zhuangdi Li

December 16, 2015

Contents

1 Package location and testing requirements

Package github repository:

The ars package named *ars.tar.gz* is located in Mengfei Jiang's Github repository:
`https://github.com/MengfeiJiang/stat243-project`.

1.1 Packages required for testing

To test our ars package the external packages listed below are required (links to CRAN website):

testthat

evd

PtProcess

smoothmest

truncdist

truncnorm.

N.B. To install all packages listed above:
Run bash/R script ***run_first*** in project root folder.

2 Description

The main function *ars* takes 4 inputs, namely (1) target log-concave density function, (2) sample size, (3) lower bound and (4) upper bound, and returns a sample by Adaptive Rejection Sampling. The lower bound and the upper bound define the support for the target density, and it is assumed that the user input the **analytically correct bounds** when defining the density. $-/+\infty$ are default values respectively. The function will then draw samples from the target density after validity checks.

3 Approach

The general approach is described by the below pseudo-code:

Let A be a set of points $x_i, h(x_i), h'(x_i)$. The tangents at these points define the upper hull h_u, the chords define the lower hull h_l.
Repeat
Sample x from $s(x)$, the normalized exponential of the upper hull
Sample u from a $Unif(0,1)$
    if $u < exp(h_l(x) - h_u(x))$ then accept x (squeezing)
    else perform the rejection test
        evaluate function value h at x
        if $u < exp(exp(h(x) - h_u(x)))$ *then* accept x

else reject x
endif
update upper and lower hull by adding $x, h(x), h'(x)$ to A
endif
until the user required number of points are sampled

Descriptions of each function are detailed as follows.

3.1 Input Validity Check

Check support boundaries Three checks are performed: lower bound is smaller than upper bound, density is neither $-/+\infty$ or smaller than 0 at bounds, and density is not 0 everywhere within the bounds.

Check density convergence The function stops when the integral of the unnormalized density diverges within the bounds.

Check log-concavity of the density Log-concavity check is performed locally at the neighbourhood of each of the abscissae, and we require $h_u(x_i)$ be greather than or equal to $h_l(x_i)$. The function stops whenever $h_l(x_i)$ exceeds $h_u(x_i)$. The check is performed as sampling proceeds.

3.2 Distribution Shape Determination

Mode finding Mode of the target density is to be used to determine the density shape and as one of the initialization points. R's *optim* function is called to find such mode as well as checks for logical upper and lower bounds. To make the optim function stable, we check where the density has support (by evaluating some function values) and optimize in the region where the density has support and the starting point is the largest function value found for the density.

Shape determination We categorize the shape of a distribution into 4 categories, (1) uniform distribution with constant density within bounds, (2) monotonely decreasing density, (3) monotonely increasing density, and (4) density mode occurring within bounds. R's built-in *runif* function is called to generate samples in the first shape category, while the following steps of ARS are performed to generate samples for the rest of the categories.

3.3 Boundary Shrink

To avoid numerical issues, valid boundaries are shrinked to such an extent that the target density is not numerically zero at and within bounds if it is in the first place.

3.4 Initialization

Initial abscissae are the lower bound, the upper bound, the mode, one point to the left of the mode and another point to the right of the mode. h value, i.e. the $log(density)$ value, and h' value is calculated for each of the initial point.

3.5 Upper hull and lower hull

Vectorized auxiliary functions u and l are such that take a vector of x's and output the corresponding values at the upper hull and the lower hull. Each time $h(x)$ is evaluated, the two functions are also updated. Log-concavity check is performed for all abscissae every time the upper and lower hulls are updated.

3.6 Intersections of upper hulls

Intersections of the upper hulls are determined by the following formula:

$$z_i = x_i + \frac{h(x_i) - h(x_{i+1}) + h'(x_{i+1})(x_{i+1} - x_i)}{h'(x_{i+1}) - h'(x_i)}$$
$$h_u(z_i) = h'(x_i)(z_i - x_i) + h(x_i)$$

, for $i = 1, 2, \ldots, n$, and $z_0 =$ lower bound.
z's are updated each time the upper hull and the lower hull are updated.

3.7 Sampling

CDF CDF up to each intersection point is calculated as follows:

$$cdf(z_0) = 0$$
$$cdf(z_i) = \frac{1}{constant} \sum_{j=0}^{i} \frac{1}{h'(x_{j+1})} exp(h_u(z_{j+1})) - exp(h_u(z_j))$$

, for $i = 1, 2, \ldots, n$ and

$$constant = \sum_{j=0}^{n-1} \frac{1}{h'(x_{j+1})} exp(h_u(z_{j+1})) - exp(h_u(z_j))$$

Inverse CDF To sample from the customized distribution, we need to inverse the cdf. First, draw a vector of $Unif(0, 1)$ random variables $\boldsymbol{u}$. Then, for each u_j, we find the largest z_i such that cdf(z_i) is smaller than u_j. Then, the x_j sampled from u_j is generated as such to form candidate sample points $\boldsymbol{x}$:

$$x_j = z_i + \frac{1}{h'(x_{i+1})} log\Big[1 + \frac{h'(x_{i+1}) * constant * (u_j - cdf(z_i))}{exp(h_u(z_i))}\Big]$$

, for $i = 0, 1, 2, \ldots, n-1$.

Squeeze test and rejection test A vector of $Unif(0, 1)$ random variables $\boldsymbol{w}$ is generated to perform squeeze test. All the candidate points up to the first rejected point are accepted and stored in the final result to be returned. Then, rejection test is performed on the rejected point, which will be added to the final result if accepted. The upper hulls and lower hulls are then updated, and log-concavity check is performed. If the number of accepted points in the result is smaller than the sample size, go back to the sampling step and sample until enough.

4 Testing

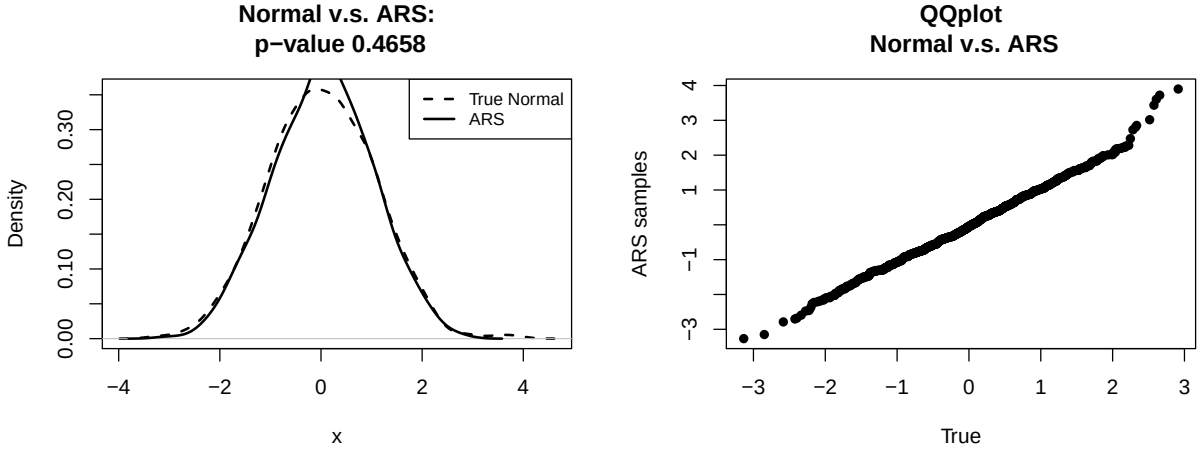
Only representative test results of the main function are detailed in the report. You may run the full test on the package to see more corner cases that we used to test the main function. Unit tests are performed while the package is being developed and are not shown here. Both log-concave and non-log-concave distributions are used to test the main function: in the former case, the test checks that the sampler correctly samples from some well-known log-concave distributions, while in the latter, an error message should be displayed to the user.

4.1 Log-concave Distributions

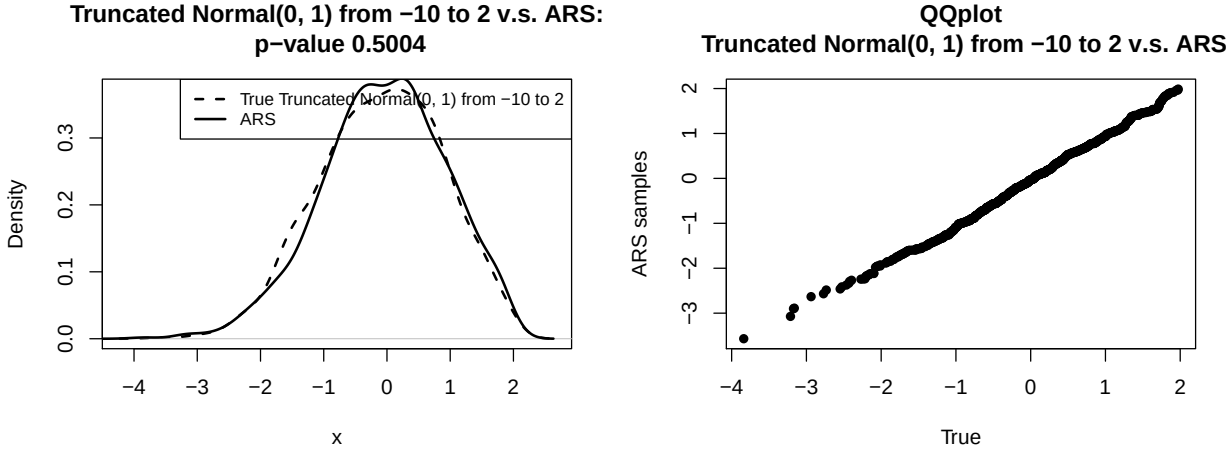
The test output comprises of (1) p-value from Kolmogorov-Smirnov test, (2) graphical comparison of the empirical density with the true one, and (3) QQ-plot of the empirical quantiles and the true.

Two-sample Kolmogorov-Smirnov test (or KS test) is used to compare a sample from a target unnormalized density generated by ARS with one generated by R's built-in function, assuming R's built-in function correctly samples from the underlying true distribution. It checks whether two data samples come from the same distribution. The test statistic is $D_{n,n'} = sup_x|F_{1,n}(x) - F_{2,n'}(x)|$, where $F_{1,n}(x)$ and $F_{2,n'}(x)$ are the empirical distribution functions of the first and the second sample respectively. We would accept that ARS correctly samples from the target distribution if the p-value of the statistic is greater than 0.05. In 20 iterations all of our 35 tests carried out with a sample size of 1000 samples passes this test 685/700, also it usually performs a lot better than the boundary limit, as can be seen below. However, since the testing sometimes returned error message for the p-value for sample size 1000, in the formal testing we put the limit to p¿0.01, just to see that our samples are not deviating too much (but still not giving an error message in rare cases, for small sample sizes).
Note, the 35 tests runs quickly on a sample size of 1000 that has been used throughout below when presenting the results from the testing.

4.1.1 Normal

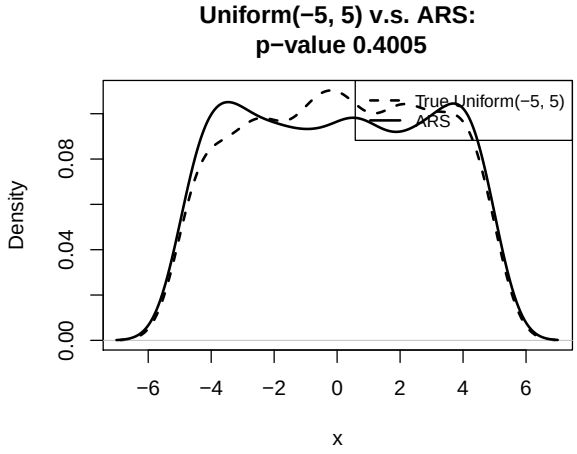


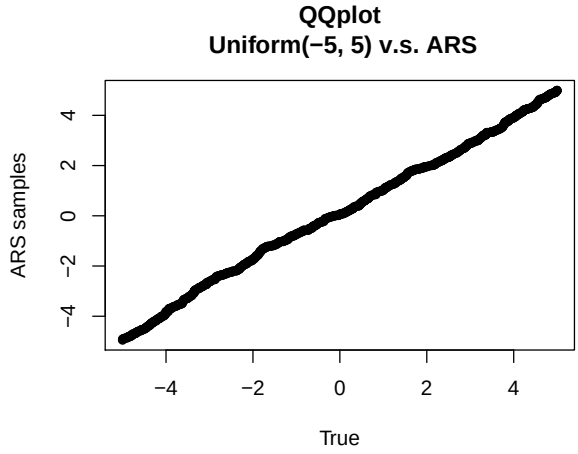
4.1.2 Truncated Normal



4.1.3 Uniform

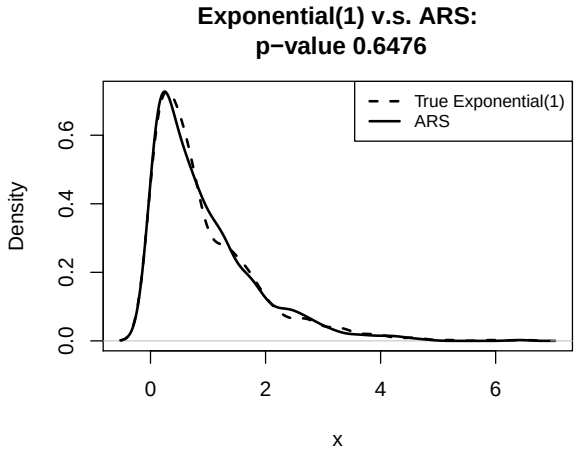
```
## [1] "Uniform distribution: runif is used to generate sample"
```

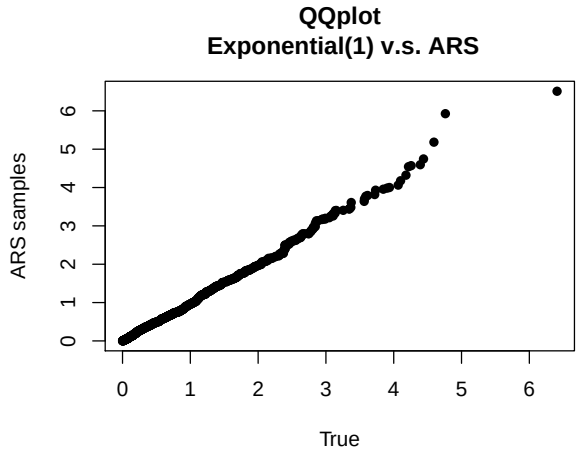




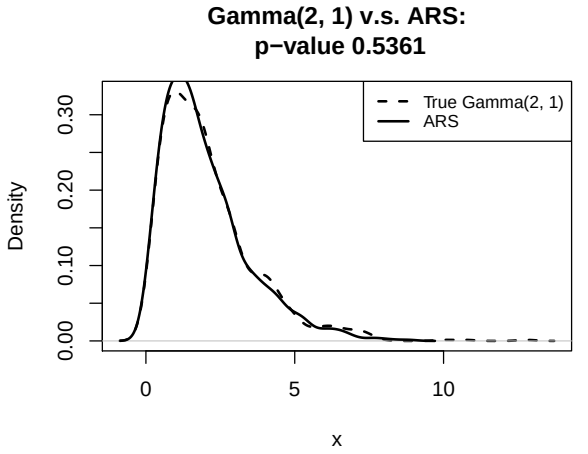
4.1.4 Exponential

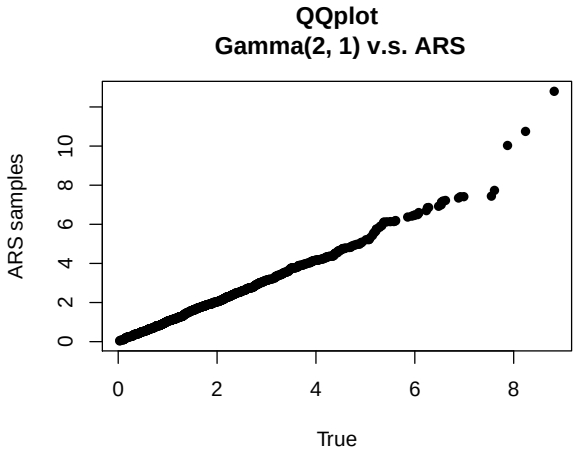
```
## [1] "Truncated distribution: the leftmost point is the mode."
```



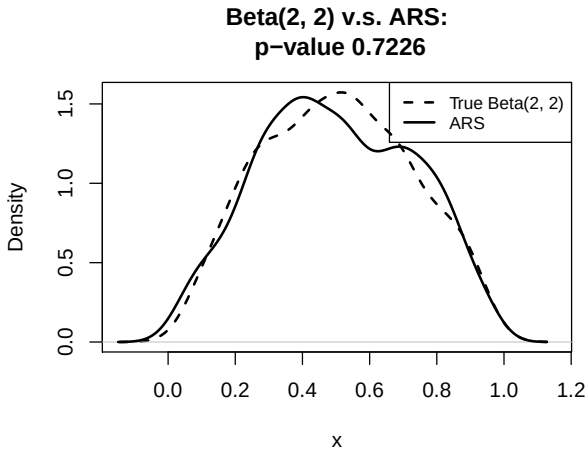


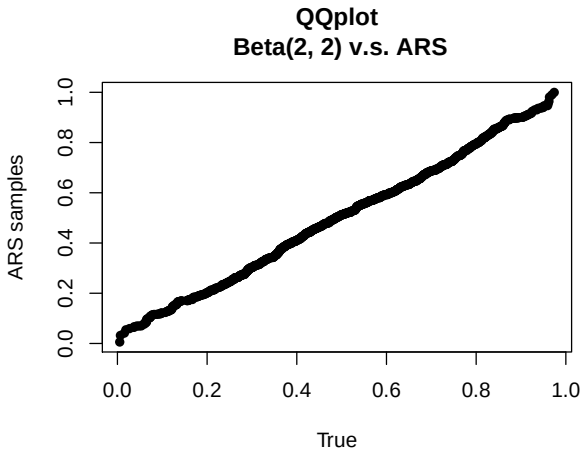
4.1.5 Gamma



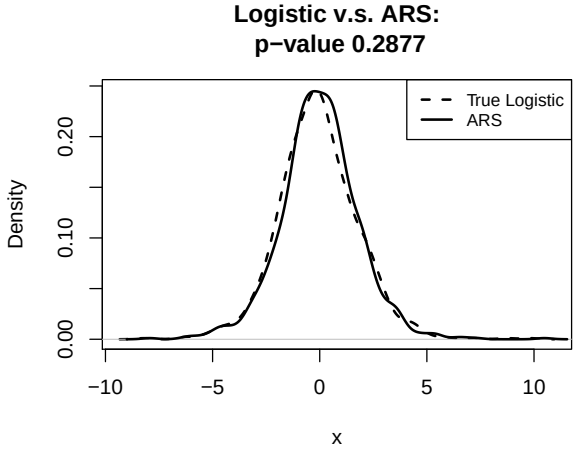


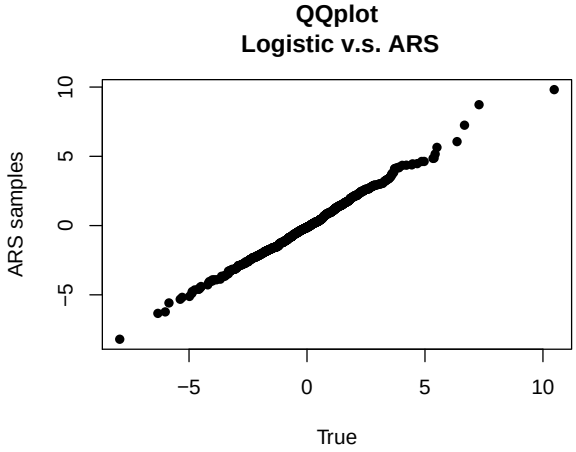
4.1.6 Beta



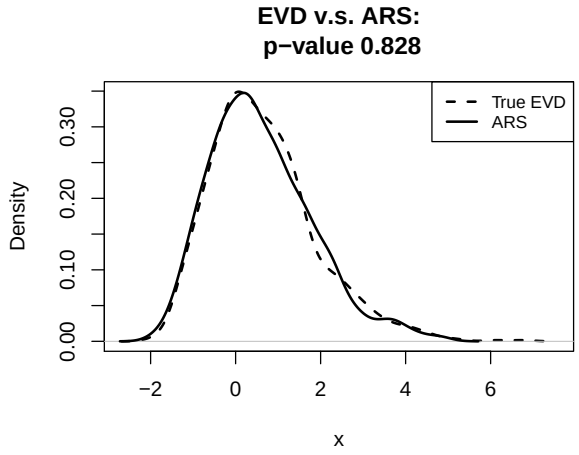


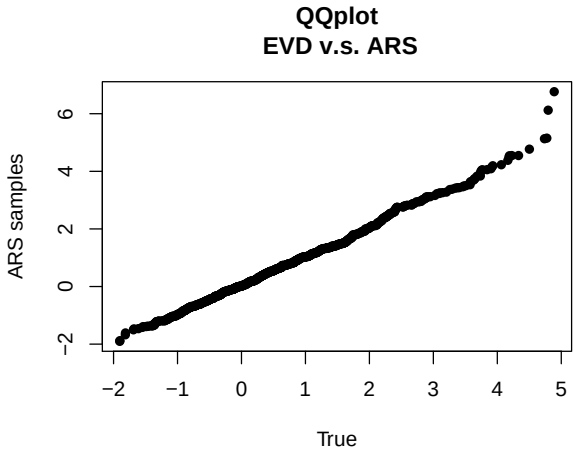
4.1.7 Logistic Distribution



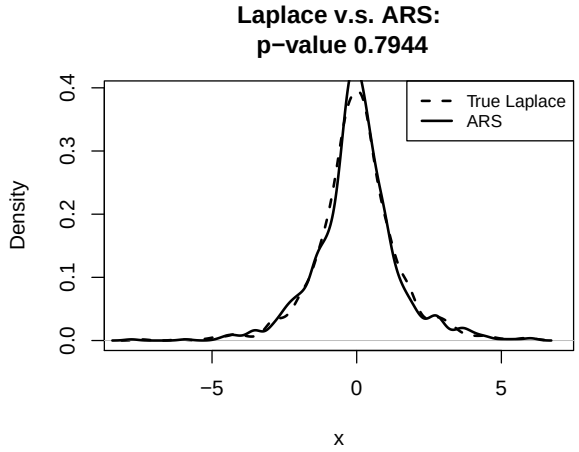


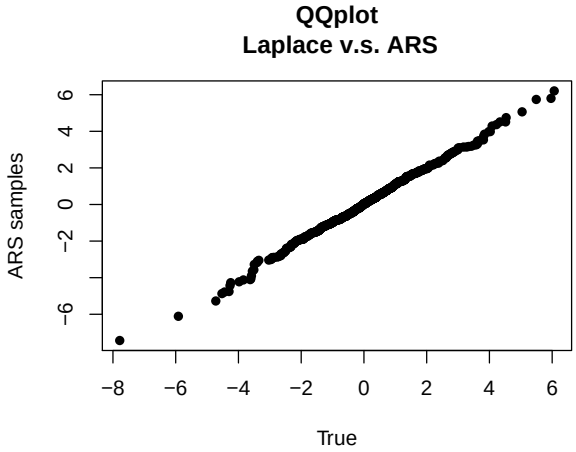
4.1.8 Extreme Value Distribution



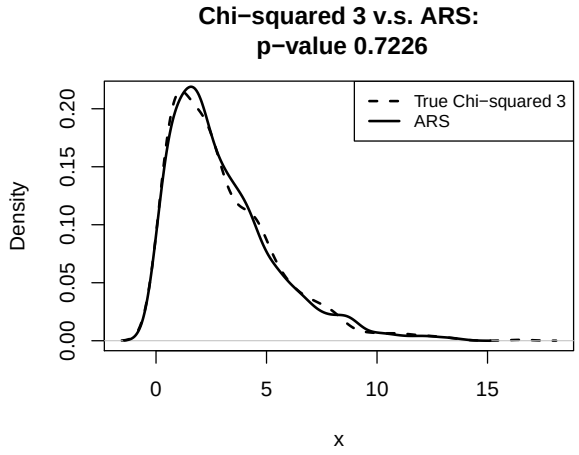


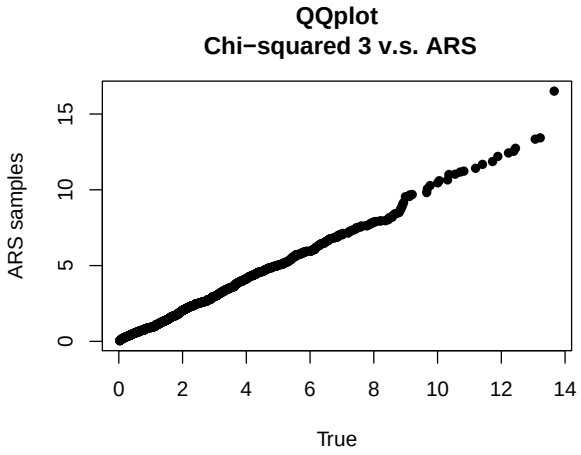
4.1.9 Laplace





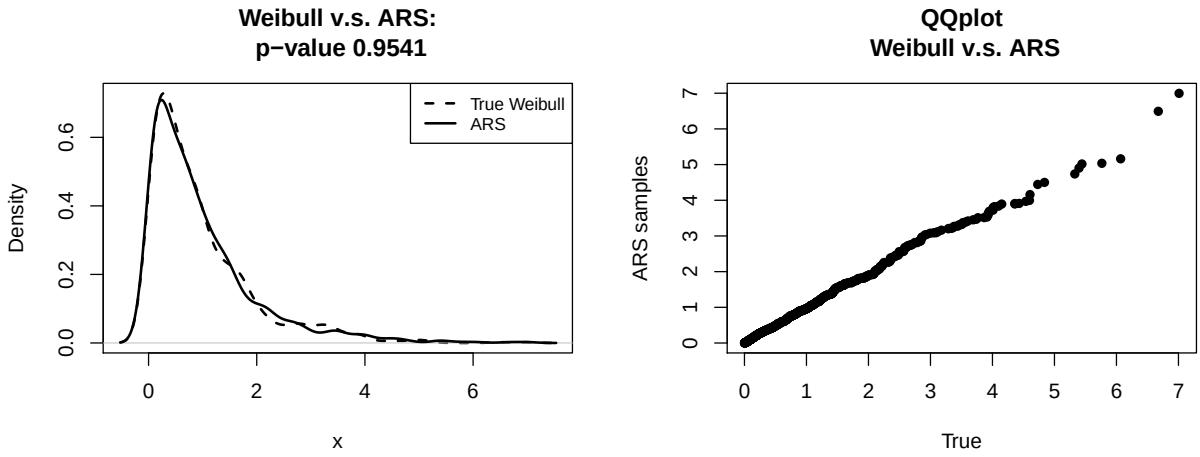
4.1.10 Chi-squared with dof >= 2





4.1.11 Weibull Distribution

```
## [1] "Truncated distribution: the leftmost point is the mode."
```



4.2 Non Log-concave Distributions

For the below non-log-concave distributions, **it is expected that the sampler should return an error message** about the density not being log-concave and stops.

4.2.1 Chi-squared with dof = 1

N.B. The function works if it produces an error below.

```
## [1] "Truncated distribution: the leftmost point is the mode."

## Error in ars(chisq_pdf, n, 0.001, Inf):  Bad density:  not log-concave
```

4.2.2 Student's t Distribution

N.B. The function works if it produces an error below.

```
## Error in ars(t_pdf, n, -50, 50):  Bad density:  not log-concave
```

4.2.3 Cauchy distribution

N.B. The function works if it produces an error below.

```
## Error in ars(cauchy_pdf, n, -Inf, Inf):  Bad density:  not log-concave
```

4.2.4 Pareto Distribution

N.B. The function works if it produces an error below.

```
## [1] "Truncated distribution: the leftmost point is the mode."

## Error in ars(pareto_pdf, n, 3, Inf):  Bad density:  not log-concave
```

4.2.5 F-Distribution

N.B. The function works if it produces an error below.

```
## Error in ars(f_pdf, n, 1e-05):  Bad density:  not log-concave
```

5 Logistics

Code development: Mengfei Jiang, Ji Wang, Alexander Samuel Fred Ojala
Testing: Mengfei Jiang, Ji Wang, Alexander Samuel Fred Ojala, Zhuangdi Li
Report: Mengfei Jiang
Package compilation: Alexander Samuel Fred Ojala, Ji Wang