STAT 243 Final Project

Vincent Myers, Yanting Pan, and Zhenni Ye

December 10, 2018

Section 0: Package Location

The ars Package is located in Yanting Pan's Github repository: https://github.com/yantingpan/ars

Section1: Function Overview

The main function *ars()* takes 2 required inputs, namely (1) sample size and (2) target log-concave density function, as well as 4 other optional inputs, namely (3) lower bound, (4) upper bound, (5) center and (6) step. The lower bound and upper bound define the domain of the target function to be evaluated with the default of negative infinity and positive infinity, respectively. The center is the mode of the target function with default of zero, and the step is the bandwidth around the center used to find the starting abscissae with default of 0.5. The *ars()* function draws samples from the target function using Adaptive Rejection Sampling along with appropriate validity tests, and returns numbers of observations.

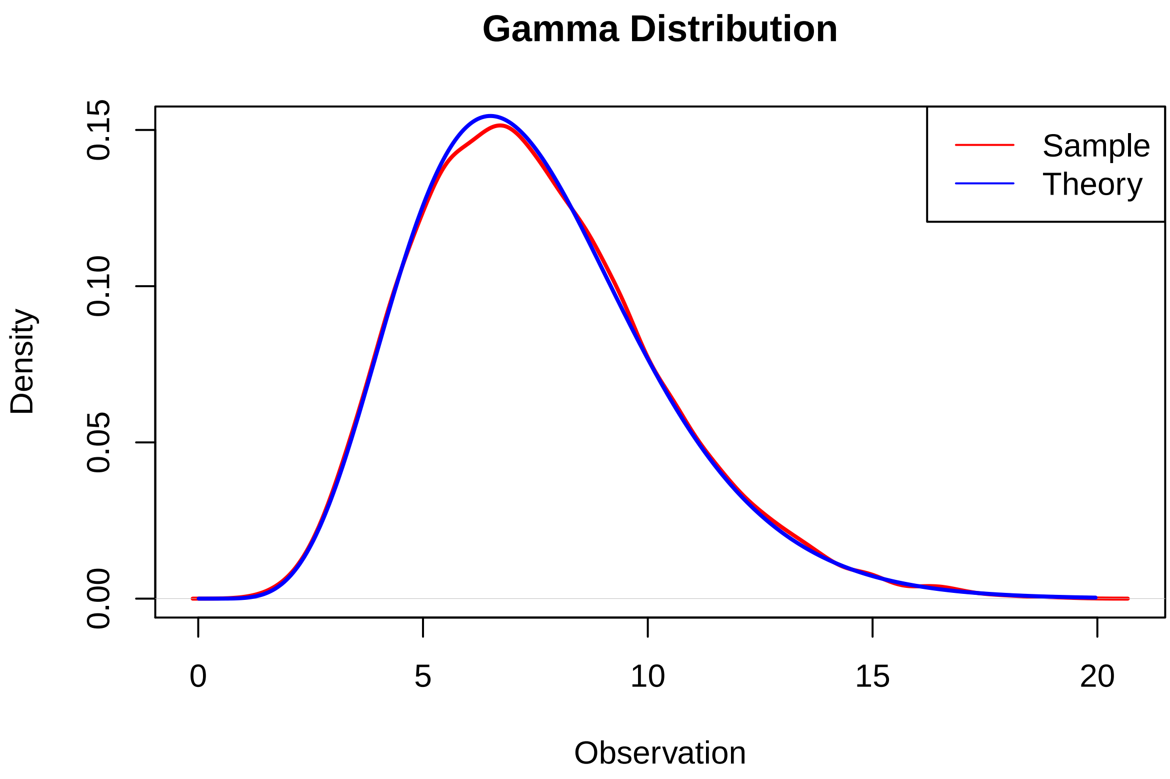
1.1 Main Function

We use gamma distribution to demonstrate how *ars()* works. Suppose a user wants to sample 10,000 observations from gamma distribution, the user needs to input (1) the sample size N = 10000, (2) the target function of the specified gamma distribution, as well as optional inputs for a lower bound and upper bound for the sampling, a center of the distribution, and a step value to assist the function for extremely narrow or wide distributions.

```
set.seed(0)
## custom function for density of gamma distribution
gamma_test <- function(x){
  k <- dgamma(x, shape = 7.5, scale = 1.0)
  return(k)
}

## run ars() function
vals <- ars(10000, gamma_test, l = 0.01, u = 20)

## graph output
z <- seq(0.01, 20, by=0.05)
plot(density(vals), xlab = "Observation", ylab = "Density",
     main = "Gamma Distribution", col = "red", lwd = 2)
lines(z, gamma_test(z), type="l", col = "blue", lwd = 2)
legend("topright", legend=c("Sample", "Theory"),
       col=c("red", "blue"), lty=1)
```



1.2 Efficiency

All the algorithm is vectorized to improve the efficiency.

```
system.time(ars(1000, dnorm))
```

```
##    user  system elapsed
##   0.124   0.004   0.131
```

Section 2: Algorithm/Approach

The implementation contains 4 parts: Initialization step, Sampling step, Updating step, and Validity checks. In order to create modular codes, we have various auxiliary functions used to implement discrete tasks, which serves the main *ars()* function.

2.1 Initialization

If the boundary is specified, we will use it as the starting abscissae. If not, we will find the abscissae starting from the center of function. We assume the center is 0, and check the derivative of candidate abscissae generated by gap, 0.5, from the center.

After 50 iterations, error message is designed to ask the user to input the estimated center of the function that helps to find reasonable starting points. Before assigning the value of starting abscissae, we will check whether accepted point(s) are defined on the function.

2.2 Sampling and Updating

Six sub-functions are involved: *setParams()*, *calcDeriv()*, *lowerPDF()*, *expPDF()*, *expCDF()*, and *invCDF()*.

The structure of the functions is:

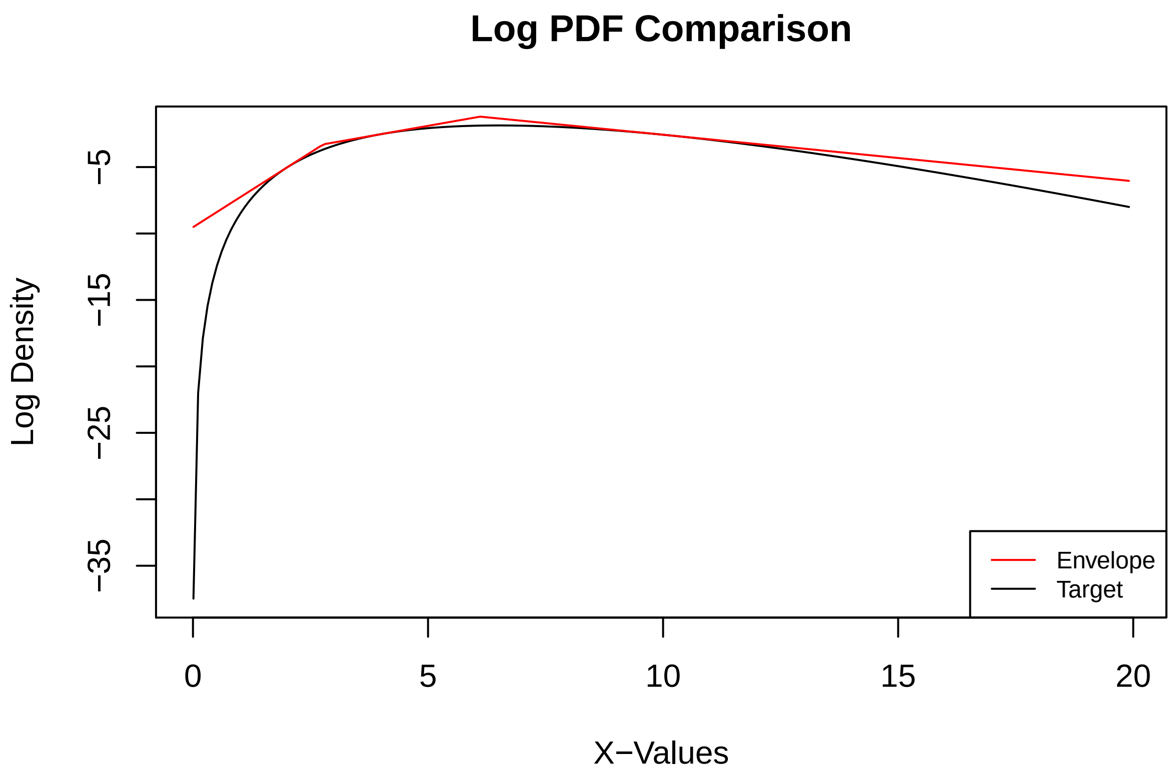
- *ars()*: main function; samples from the CDF of the enveloping function, performs the squeeze test and the envelope test, and either (i) accepts or (ii) rejects and updates the fixed points.
    - *setParams()*: determines the parameters for the enveloping function and specific set of fixed points.
        - *calcDeriv()*: calculates the derivative of a function at a given point.
        - *expCDF()*: calculates the piecewise exponential CDF of the enveloping function and returns the parameters. Each line segment in the enveloping log PDF, each in the form of $y = mx + b$, is exponentiated and integrated using the form $(1/m)e^{mx+b}$, and then shifted vertically to create a continuous CDF. The CDF is then normalized, and the parameters used for the shifts and the normalization are returned to be used by other functions.
    - *lowerPDF()*: calculates the value of the squeezing function for a specified evaluation point.
    - *expPDF()*: calculates the value of the piecewise exponential enveloping function for a specified evaluation point. For a given x-value, the function finds the related piece of the enveloping log function and calculates the y-value using $y = mx + b$ form. It then exponentiates the result, which is returned.
    - *invCDF()*: inverts the piecewise exponential CDF of the enveloping function; takes as input a number sampled from the Unif(0,1) distribution, finds the corresponding piece of the enveloping function, and inverts the equation used in the *expCDF()* function in order to return a sample from the CDF.

Since the *ars()* function does not provide any intermediate output, we've used some of the sub-functions below to demonstrate the code. The code below uses the gamma function above to graphically demonstrate the relationship between the target function and the enveloping function.

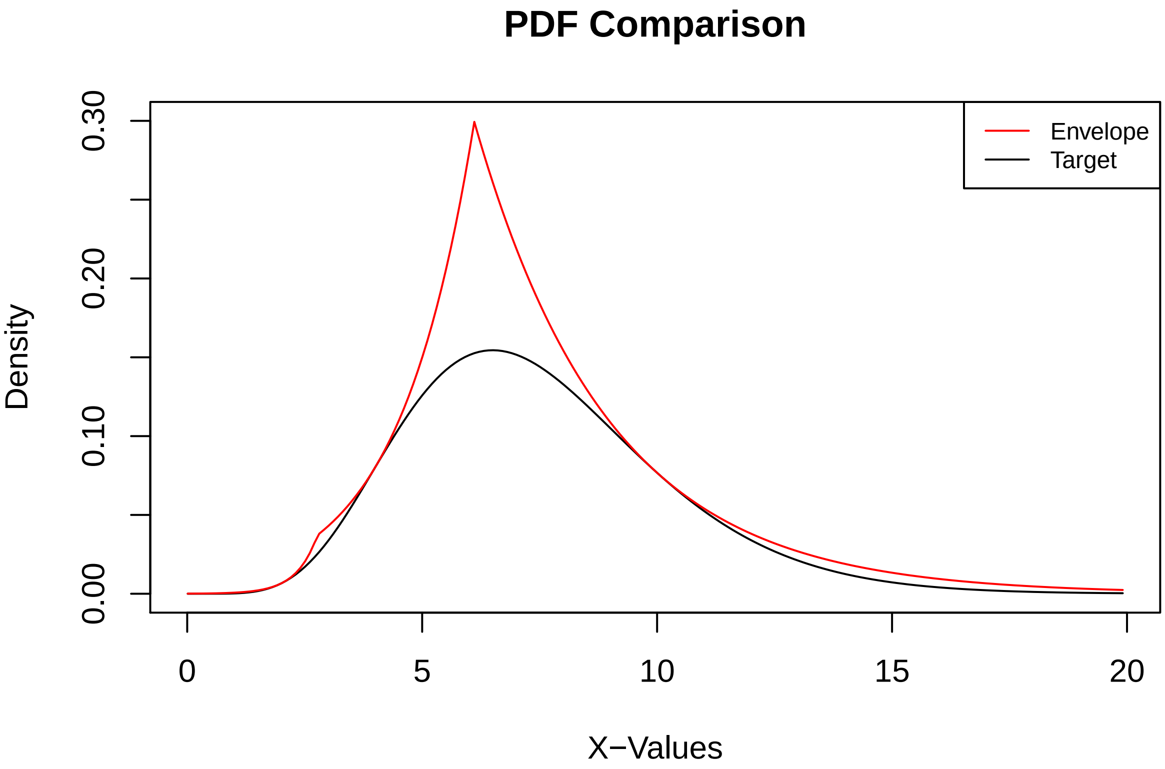
```r
## define function inputs
min <- 0.01
max <- 20
p <- c(2, 4, 10) # define the initial fixed points
xvec <- seq(min, max, by=0.1)
target <- gamma_test(xvec)

## calculate corresponding envelope function
par <- setParams(gamma_test, 0.01, 20, p) # parameters for function
envelope <- expPDF(xvec, p, par$int_x, par$m_p, par$lf_p) # enveloping PDF

## plot log PDFs
plot(xvec, log(target), type='l', xlab="X-Values", ylab="Log Density",
     main="Log PDF Comparison")
lines(xvec, log(envelope), col="red")
legend("bottomright", legend=c("Envelope", "Target"),
       col=c("red", "black"), lty=1, cex=0.75)
```



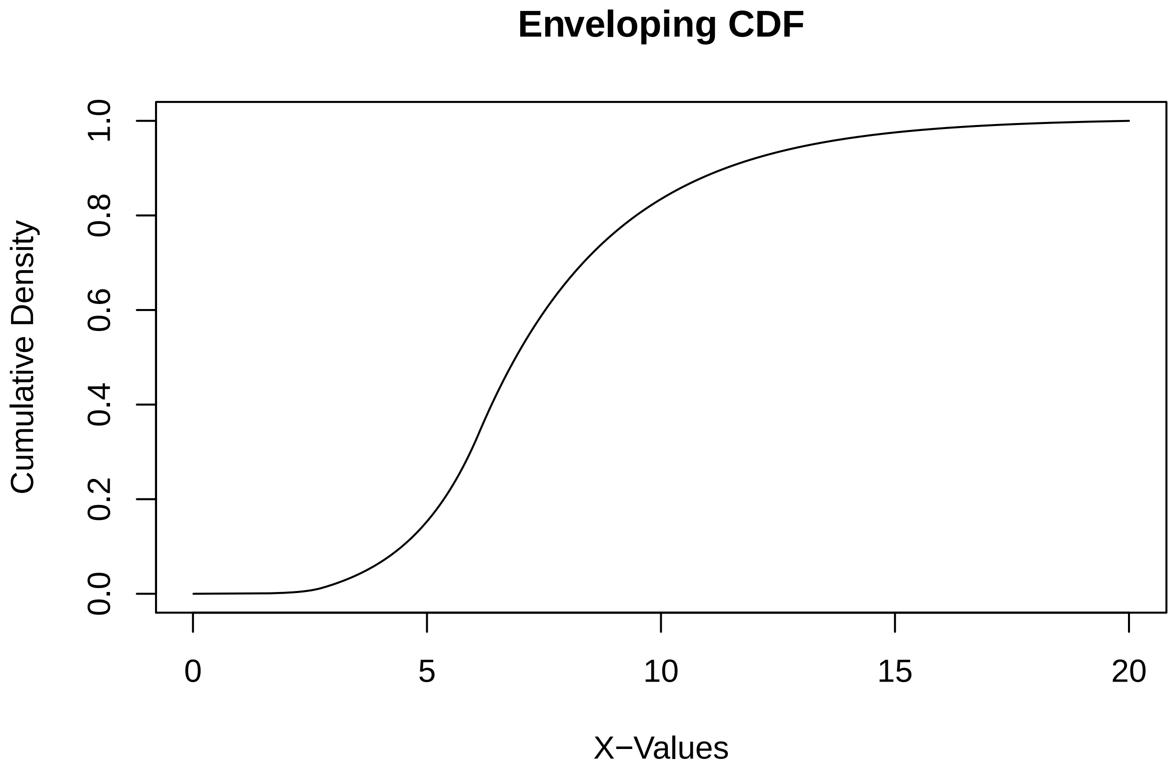
```
## plot PDFs
plot(xvec, target, type='l', ylim=c(0,0.3), xlab="X-Values",
     ylab="Density", main="PDF Comparison")
lines(xvec, envelope, col="red")
legend("topright", legend=c("Envelope", "Target"),
       col=c("red", "black"), lty=1, cex=0.75)
```



The chart below shows the CDF of the enveloping function shown above.

```
yvec <- seq(0, 1, by=0.001)
envelope_CDF <- invCDF(yvec, par$int_x, par$m_p, par$b_vec, par$nc, par$shift, par$adj)

plot(envelope_CDF, yvec, type='l', xlab="X-Values", ylab="Cumulative Density",
     main = "Enveloping CDF")
```



2.3 Validity checks

- Check for sample size: the input sample size should be a positive numeric integer. Stop the function otherwise.
- Check for the boundary: the boundary input should be numeric, and the lower bound must less than the upper bound. In case of the user mistakenly enters the value of lower bound as the upper bound and the value of upper bound as the lower bound, provide the warning message and swap the values automatically for further implementation. Stop the function otherwise.
- Check for density function: the input density function should be a function firstly.
  - Check for existence: if the function is not defined (i.e. not finite) on the boundary points, and updating fixed points, stop the function and give the error message.
  - Check for differentiability: if the function is not differentiable (i.e. the calculated derivative is 'NaN' or not finite) on the boundary points, starting points, and updating fixed points, stop the function and give the error message.
  - Check for log-concave: the log-concave function should obey the property of the log-concave, that is, the derivatives of the log-scale of the function decreases monotonically with increasing abscissae within the domain. This test is conducted on sorting the staring points and updating fixed points as the calculations proceed. The function stops once the derivatives of the log-scale function do not follow the property stated above, and gives the error message.

Section 3: Testing

3.1 Testthat

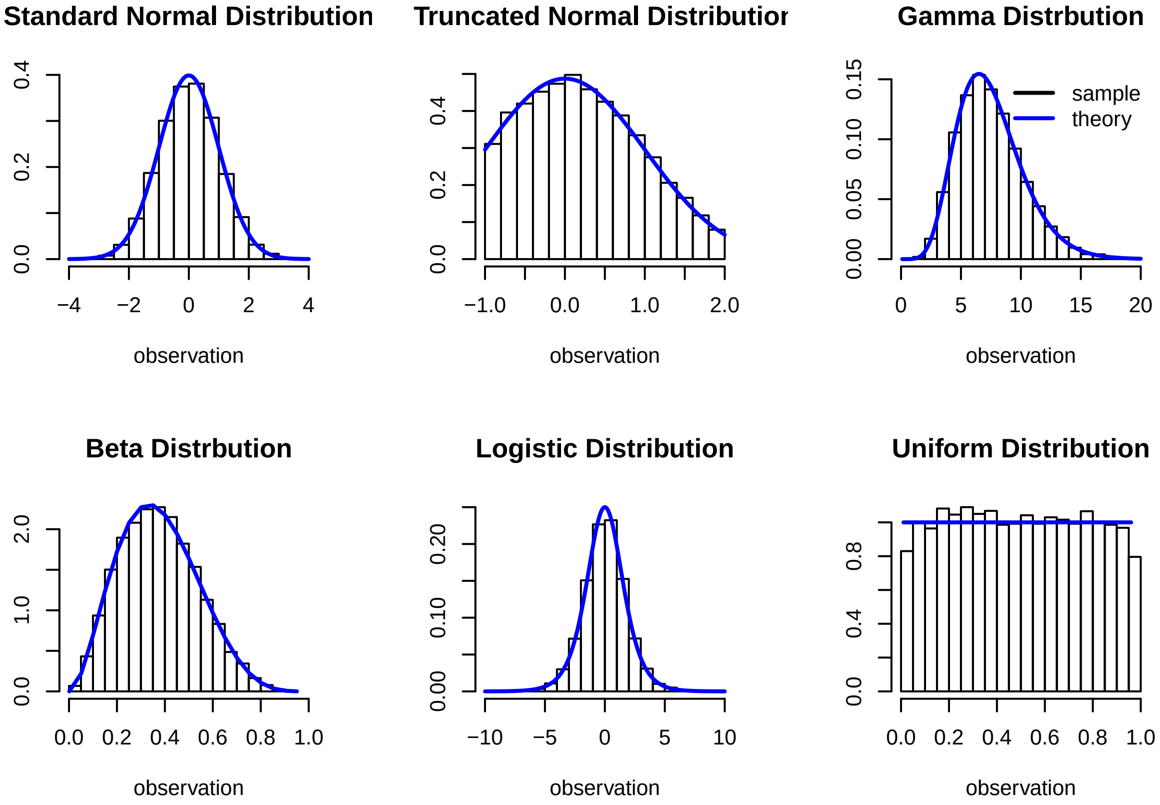
- Final Tests:

  - To test our algorithm, we compared our resulting samples with the known distributions using Kolmogorov-Smirnov Tests. We tested our function against the log-concave distributions including the Normal distribution, Gamma distribution, Beta distribution, Logistic distribution, Uniform distribution. Our algorithm consistently passed for all those functions with the significance level of 0.05. Furthermore, we also tested our function against the non-log-concave distributions including the Student t distribution, Chi-square distribution with one degree of freedom, and F distribution. Error message we designed appeared correctly as expected.
- Module Tests: we tested all auxiliary functions to make sure the output of each individual function was correctly computed and formatted.

3.2 Results/Examples

3.2.1 Log-concave Distributions

Below is the plot of some distributions. The histogram shows the density of observations generated by *ars()*, and the blue line shows the theoretical density.



3.2.2 Non Log-concave Distributions

Student t Distribution, Chi-square Distribution with one degree of freedom, and F Distribution are not a log-concave density. The function works if it produces the error message “Please provide the log-concave density.”

```
# Student t Distribution
t_test <- function(x){return(dt(x, 1))}
ars(100, t_test, l = -10, 10)
```

```
## Error: Please provide the log-concave density
```

```r
# Chi-square Distribution with dof = 1
chi_test <- function(x){return(dchisq(x, 1))}
ars(1000, chi_test, l=1)
```

```
## Error: Please provide the log-concave density
```

```r
# F Distribution
f_test <- function(x){return(df(x, 9, 11))}
ars(100, f_test, l=1)
```

```
## Error: Please provide the log-concave density
```

Section 4: Contributions

Vincent is responsible for the algorithm of sampling and updating steps. Yanting and Zhenni are together responsible for the initialization step, validity checks, tests, and R packaging. All group members reviewed, discussed and revisied the codes, and all members contributed to the final report.

Section 5 : References

Gilks, W. R., and P. Wild. “Adaptive Rejection Sampling for Gibbs Sampling”. Journal of the Royal Statistical Society. Series C (Applied Statistics) 41.2 (1992): 337-348.