Programming Approach

Code Structure

The coding of the project follows a functional approach. Functionality and modularity comes in two facets; first, reusable operations are defined to be auxiliary functions (and sub-functions) that are accessible by other functions; second, lengthy pieces of computation are out sourced into auxiliary function to improve main function readability. Specifically, the project code includes 5 functions (including 1 wrapper function, 1 main function, 3 auxiliary functions), and 2 sub-function objects (all 7 shown below).

Functional Programming

Here the flow of the adaptive rejection sampling process is broken down into core units and operation groups. There are 3 main groups of operations. First, to check the validity of input and decide whether to sample or to abort (Wrapper Function). Second, to initialize the starting X-grid (vector) by identifying and breaking up an efficiently sample-able bound (First Helper Auxiliary Function). Third, in a loop, sample X's, based on calculated acception criterion, determine whether to accept X and whether to add X to the X-grid (Main Function and other Auxiliary Functions).

Object Oriented Programming

The target density functions and other helper functions (including log density and the derivative of the log density) are passed into other functions as the actual function object. To pass them around not only avoids repeating code but also avoids potentially harmful parameter overrides. While they are passed around, their namespace (environment) is encapsulated in the wrapper functions, hence they are always initialized with proper input density parameters despite local variable names.

Code Explanation

```r
# Wrapper Function
ars = function(target_density, n, x_domain, num_iter_allowed, ...)
# Reusable Sub-Function (Inside Wrapper Function) Objects
h_of = function(x_vec){ log(target_density(x_vec, ...)) }
h_prime_of = function(x_vec){ dx=1e-8 ; derivative=(h_of(x_vec+dx)-h_of(x_vec))/dx }
```

'ars' is the wrapper function that takes in the user input, checks input validity in terms of the domain's correctness, operability, and the density function's validity and log concavity with multiple test. if any of the test fails, stop function will be called and error message thrown.

Also inside the wrapper function, the log density function object, h(x), and the derivative of the log density function object, h'(x), are made. If input validity check is passed, the wrapper 'ars' function will call the main function, 'get_samples_from_density', to perform sampling.

```r
# Main Function
get_samples_from_density = function(target_density, h_of, h_prime_of,
                                    x_domain, n, num_iter_allowed, ...)
```

The main function, 'get_samples_from_density', takes in pre-validated inputs from the wrapper 'ars' function, and first calls auxiliary 'get_initial_x_vec_and_D' function to determine staring vector for the sampling process. It then, in a while loop, acquire n samples with the iterative process described by the adaptive rejection sampling process from the paper. In the loop, to enhance the readability, grouped operations, 'get_z_vec_and_I_vec', and repeated operations, 'if_all_unique_append', are outsourced as an additional auxiliary function as shown below.

```r
# Auxiliary Function 1
get_initial_x_vec_and_D = function(target_density, h_of, h_prime_of, x_domain, ...)
```

The aforementioned, 'get_initial_x_vec_and_D', is in charge of determining a efficiently sample-able starting location in the domain and numerically operable upper and lower bounds. First, the start point, x_mid, is set to be the point with the highest density where h_prime_of(x_mid) is numerically zero. Second, the operable bounds is determined by comparing the user specified 'x_domain' and where in such domain does, x = D = c(lower, upper), have zero h_prime_of(x). Here D is returned. Then, the domain is split into parts to cut up the density function and cuts return as 'x_vec'.

```r
# Auxiliary Function 2
get_z_vec_and_I_vec = function(target_density, h_of, h_prime_of, x_vec, d, ...)
```

The auxiliary helper function, 'get_z_vec_and_I_vec', is an extension to the main function where it take out the lengthy operation of finding the the-z-vector, z_vec, intercepts of the tangent lines to the h(x) function at the vector x_vec. Along the way, the integral of exp(u(x)) as defined in the paper is computed for each interval of the points in z_vec. This is stored in the integral-vector, I_vec, and is also returnd after normalizing by the sum of integrals.

```r
# Auxiliary Function 3
if_all_unique_append = function(x_vec, x)
```

The auxiliary helper function, 'if_all_unique_append', check whether input 'x' exist in the input vector 'x_vec' with a threshold of tolerance (1e-10). 'x' will be added to 'x_vec' if it's not already in it; 'x_vec' will be returned.

Testing Procedure and Examples

Output Correctness Tests

```
expect_equal(ks.test(ars(dchisq, 1000, x_domain = c(0, 100), df = 2),
                     rchisq(1000, df = 2))$p.value > 0.01, TRUE)
```

The ks test checks similarity to ensure that samples belong to sampling distribution by comparing the 'ars' empirical distribution with base R sampling empirical distribution. Note that these tests are stochastic and may occasionally fail; since with the give p-value of 0.01 and we have 7 tests, we have a $0.99^7 \approx 93.2\%$ chance to pass all 7 tests, but is a good sanity check.

Output Length Tests

```
expect_equal(length(ars(dnorm, x_domain=c(0, Inf), n=100000)), 100000)
```

We want to make sure sample vector generated is of valid length requirement.

Input Domain Tests

```
expect_error(ars(dchisq, 1000, x_domain=c(-Inf, -1000), df=10))
```

The input domain is simply wrong, we expect error message from checking in 'ars'.

Input Density Tests

```
expect_error(ars(dt, 1000))
```

The input density is not log concave, we expect error message from checking in 'ars'.

Auxiliary Function Tests

```
expect_equal(if_all_unique_append(c(1,2,3,4,5),5.0000000001) , c(1,2,3,4,5))
```

This is one example test specifically for auxiliary functions.