ST407 Monte Carlo Methods Assignment 1

Student Number: 1983162

Question 1

The density of interest with parameters is as follows;



*Part a Evaluate density*

The function f.density was used to evaluate in R as follows;

f.density <- function(x, a, b){

y = (x^(a - 1)\*(1-(x^a))^(b - 1))

y

}

Note that the normalizing constant of in 1 is given by however the function was evaluated only up to its normalizing constant. To implement the function f.density and produce the required plot for say , the following code was executed;

x = seq(0,1,length=1000)

a = 1/5

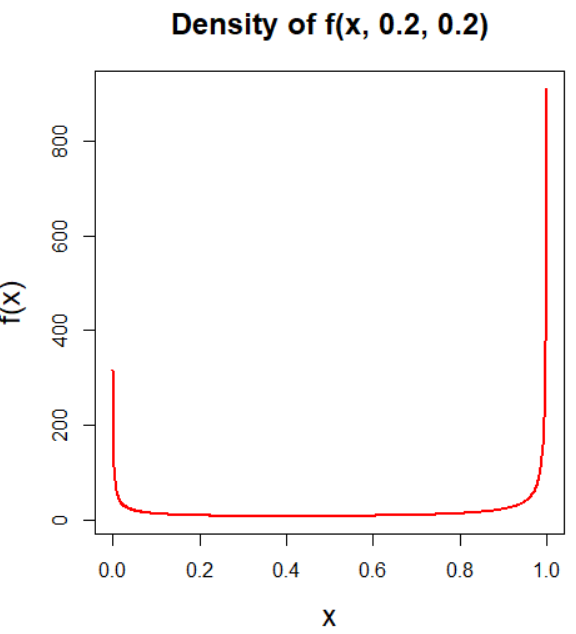
b = 1/5

fx1 = f.density(x, a, b)

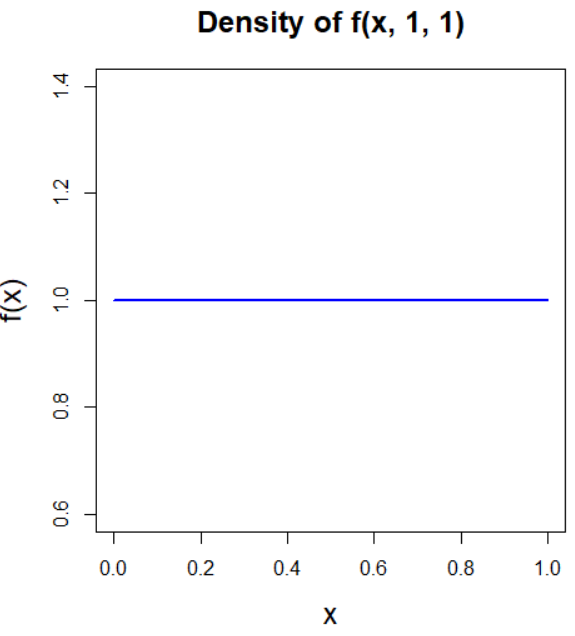
plot(x, fx1)

The plots for all parameterisations are displayed below;

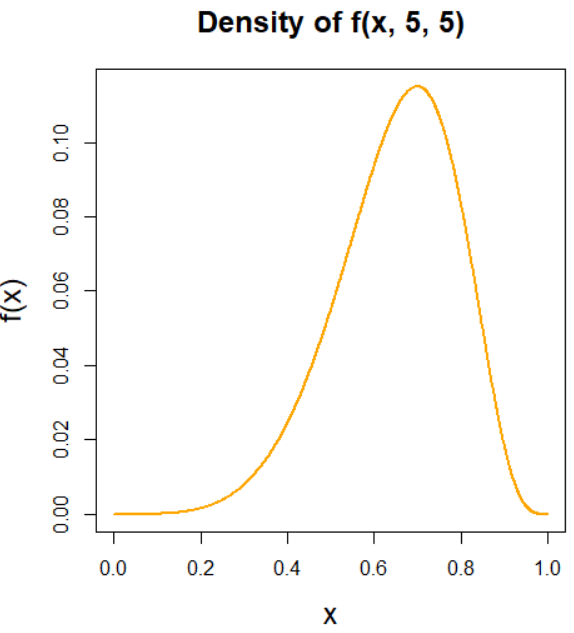
Plot I:



Plot II:



Plot III:



*Part b Inversion Sampling*

Inversion sampling was used to return samples from . Corresponding to the algorithm for inversion sampling, this was achieved by firstly determining the CDF of , namely (See 2), subsequently finding the inverse of the CDF (See 3), generating uniform samples U[0, 1] and obtaining the desired density by means of

CDF

(2)

Inverse CDF

(3)

The function inversion\_sampler was written in R to execute this as below, i.e to generate *n* samples from the target distribution for any given .

inversion\_sampler = function(alpha, beta, n){

#Setup

u = runif(n) #uniform samples [0,1] of sample size n

fu = (1 - (1-u)^(1/b))^(1/a) #Inverse cdf

fu #Return fu

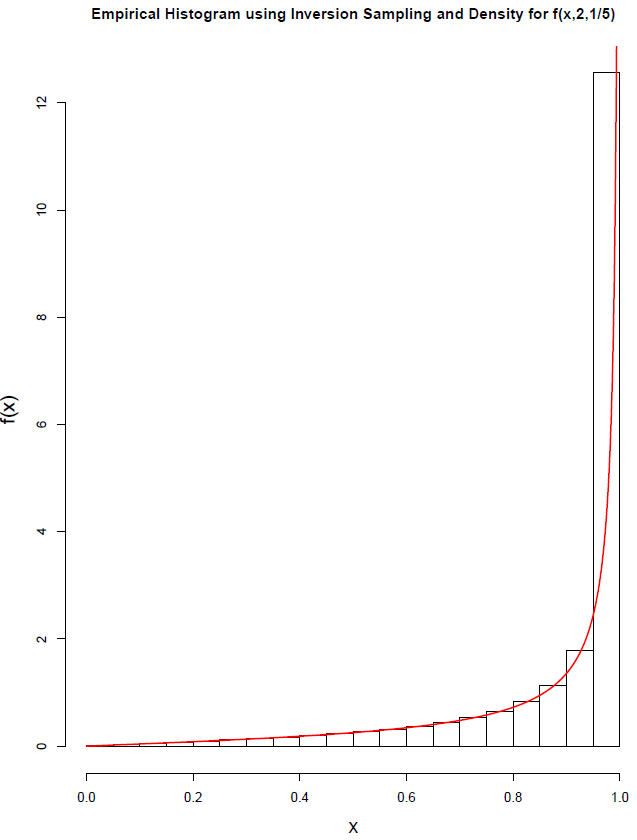
}

A possible assumption in using inversion sampling is that we can get an analytic form of the inverse of the cdf, which is not always feasible, but is in this case.

(ii) To generate 1,000,000 samples from the density involves simply executing the following;

samples = inversion\_sampler(2, 1/5, 1000000)

The required density can be visualised using the hist command. The empirical histogram of the density and the true target density are plotted together in figure 1b illustrating the application of inversion sampling.

 ­­­­

*Figure 1b. Inversion sampling for (n samples = 1,000,000)*

*Part c Rejection Sampling*

Rejection sampling was also used to return samples from using the function rejection\_sampler as below. A uniform distribution was chosen as the proposal distribution. In rejection sampling, a sample is accepted with probability given by;

(4)

Whereby is the target distribution in question (i.e (1)), ) is the proposal distribution (uniform) and is some constant bound given by;

Whereby is the normalizing constant found to be , therefore;

This is maximised when is equal to zero, i.e;

This derivative can be evaluated for any given using the function evaluate\_fx\_derivative as below. To get the maximum of the curve, the roots of this equation were found using the R function uniroot.all (x = 0 was disregarded in this case). The maximum was then found by evaluating the density at the root, i.e;

bound = f\_density(root\_fx, a, b)

Whereby;

f\_density <- function(x, a, b){

norm\_constant = a\*b

y = norm\_constant\*(x^(a - 1)\*(1-(x^a))^(b - 1))

y

}

As previously stated for n given proposal samples, samples corresponding to the target distribution were accepted with probability given by (4) as shown below in the rejection\_sampler function

rejection\_sampler = function(a, b, num\_req\_samps){

#Step 1: Get bound

evaluate\_fx\_derivative = function(a, b){

function (x) (a-1)\*x^(a-2)\*(1-x^a)^(b-1) -a\*(b-1)\*(x^(2\*a-2)\*(1-x^a)^(b-2))

}

fx\_deriv = evaluate\_fx\_deriv(a,b)

#Roots

roots\_all <- uniroot.all(fx\_deriv, c(0, 1))

root\_fx = roots\_all[-1]

bound = f\_density(root\_fx, a, b)

print(bound)

#Num proposals

num\_proposals = num\_req\_samps + round(num\_req\_samps\*(bound))

#Rejection algorithm

x = runif(num\_proposals)

u = runif(num\_proposals)

accept <- u <= (f\_density(x, a, b)/bound)

x\_accept = x[accept]

x\_accept = x\_accept[1:num\_req\_samps]

x\_accept

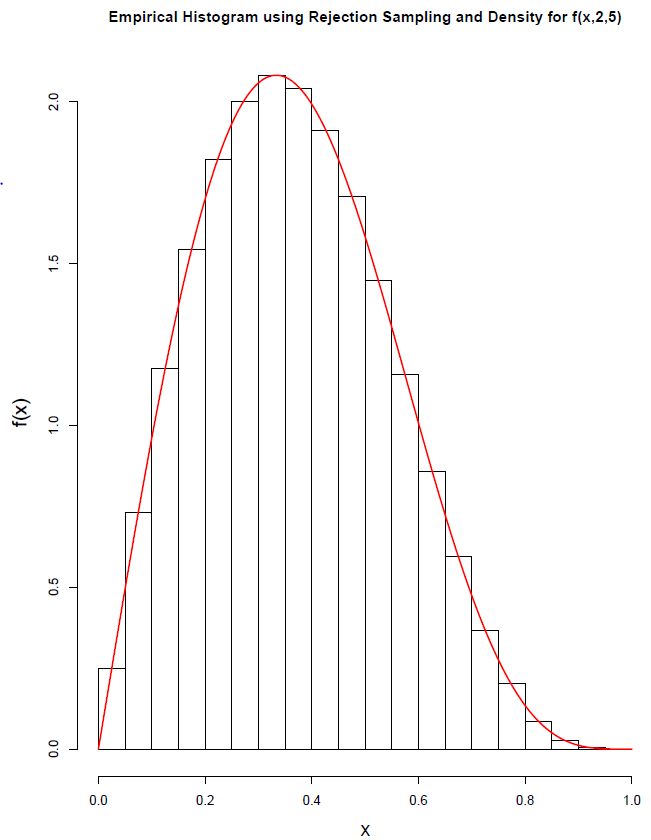
}

Note that an assumption made in this implementation of rejection sampling is that we can analytically compute the bound by finding the maximum of the curve which may not always be feasible.

ii. To generate 1,000,000 samples from the density involves simply executing the following;

fx\_is = rejction\_sampler(2, 5, 1000000)

The required density can be visualised using the hist command. The empirical histogram of the density and the true target density are plotted together in figure 1c illustrating the application of rejection sampling.



*Figure 1c. Rejection sampling for (n samples = 1,000,000)*

*Part d Comment*

It could be argued that both the inversion\_sampler function and the rejection\_sampler function are relatively efficient given that they generate a sample size to the order of 6 (1 million samples) in under 3 seconds as displayed in Table 1d. The functions were created without the use of any explicit loops in a effort to reduce the overall run time. The elapsed times were found in R using the the sys.time() function. Each time corresponds to the average elapsed time of three runs of the function. As an example, the time required to execute one run of the inversion\_sampler would be;

start\_time = Sys.time()

fu = inversion\_sampler(a\_is, 5, 1000000)

end\_time = Sys.time()

time\_elapsed = end\_time - start\_time

In comparing the two functions, it seems the inversion\_sampler function is more efficient than the rejection\_sampler function. The latter function contains significantly more computations including determining the roots of the function, which could explain this.

\*\* What to do for no roots

*Table 1d. Elapsed time required to run the sampling functions*

|  |  |  |
| --- | --- | --- |
| Function | Inversion Sampling timing  (s) | Rejection Sampling timing  (s) |
|  | 0.25 | 0.29 |
|  | 0.15 |  |

*Part e.*

To get a sample from as below, the function h\_sampler was written.

 (5)

Let

Given that it is a sample from the max of two normalised densities the following must hold;

Whereby the lower bound is given when there is no overlap between the two functions and upper bound is given where there is complete overlap.

To sample from the principles of a ‘coin toss’ will be used, i.e that of a Bernoulli distribution whereby the probability that we will draw a sample from is ½ and the is also ½. The pseudo code of the h\_sampler function is given below;

Pseudocode

***Repeat;***

1. Generate variables according to following distributions;

1. Implement ‘Coin Toss’ to take a samples from either or ;

***Until;***

Evaluate functions at w and accept sample if above holds

***Then***

z = w

The function in R was implemented as follows;

h\_sampler2 = function(g, num\_samps){

#1. Get two target densities

h1 = inversion\_sampler(2, 1/g, num\_samps)

h2 = inversion\_sampler(2, g, num\_samps)

#Coin toss between two distributions (Bernoulli)

i = runif(num\_samps)

w = h1\*(i>1/2) + h2\*(i<1/2)

#Which density is max\*

u = runif(num\_samps)

threshold=(max(f\_density(w, 2, 1/g), f\_density(w, 2, g)))/(f\_density(w, 2, 1/g) +f\_density(w, 2, g))

accept\_w <- u <= threshold

#Accepted samples

x\_samp = w[accept\_w]

x\_samp

}

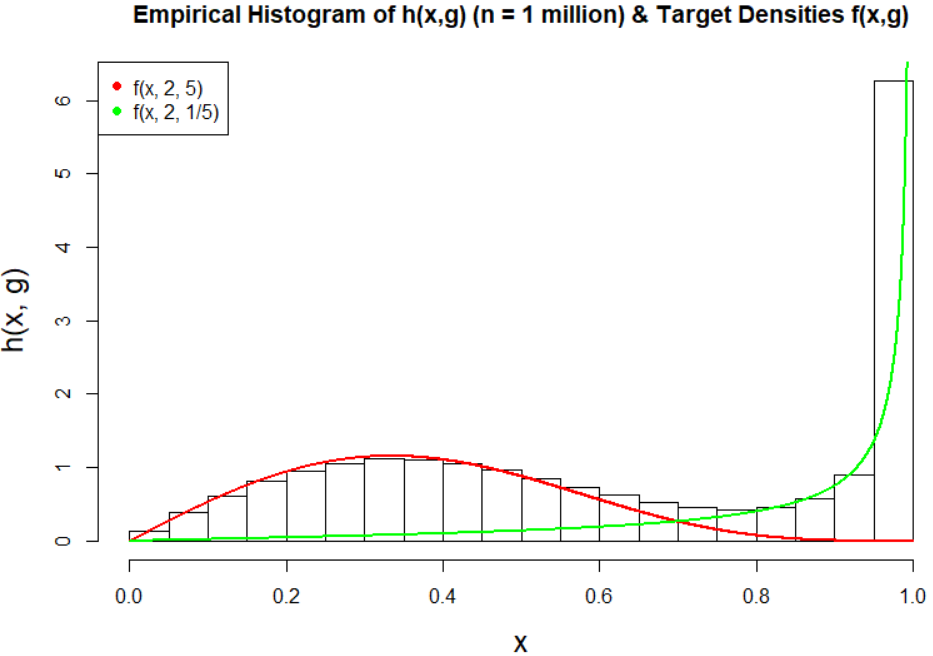
(ii).The empirical histogram of the density and the true target densities are plotted together in figure 1e. The density plots are scaled according to the total mass of the two plots given by;

Normalizing factor = = 1.83950

The CDF is given by (2) above while is the point of intersection of the two functions found analytically, by equating *f1* and *f2* and solving for x, to be;

In this case for the point of intersection was found to be 0.699 for which the normalizing factor above was found.

As shown in figure 1e the h\_sampler effectively samples from ,



*Figure 1e. Empirical histogram of and target densities*

Part e (iii)

It seems that the function h\_sampler effectively samples from as shown in figure 1e. The function was firstly written in a loop and then recreated without in order to reduce the run time. The total run time is shown in table 1e. Note that as we increasing gamma the timing increases. This is perhaps unsurprising, as if we liken the function to a gamma distribution, the second parameter ( is the scale parameter since most of it’s influence is on the spread of the data, therefore you would suspect it would take more time to sample from a distrubtion of greater spread (larger gamma) perhaps.

Table 1e

|  |  |
| --- | --- |
|  | h\_sampler timing  (s) |
|  | 1.00 |
|  | 1.04 |
|  | 1.16 |

-