Gibbs Sampling from Weibull Distribution

Vijay Nadimpalli - 140123042

Talasila Saiteja-140123040

Lakkimsetti Sujay - 140123020

A V N K Sai Charan - 140123004

B Santosh Reddy - 140123010

13 April 2016

1 Theory

In probability theory and statistics, the Weibull distribution is a continuous probability distribution. It is named after Waloddi Weibull, who described it in detail in 1951, although it was first identified by Frechet (1927) and first applied by Rosin & Rammler (1933) to describe a particle size distribution. The Weibull Distribution is given by the following density function:-

$$f(x;\beta,\theta) = \beta\theta^{\beta}x^{\beta-1}e^{-(\theta x)^{\beta}}, \qquad x,\theta,\beta > 0 \tag{1.1}$$

Its cumulative distribution function is given by:-

$$F(x;\beta,\theta) = 1 - e^{-(\theta x)^{\beta}}, \qquad x,\theta,\beta > 0 \tag{1.2}$$

Our objective is to generate from this distribution. To achieve this, we have used the inverse transform method. As a start, we generate $u \sim U(0,1)$ and obtain the Weibull variate x by

$$x = (1/\theta)[-\log(1-u)]^{1/\beta} \tag{1.3}$$

In the process of generating sets of the same, we fix θ, β Our next immediate concern is to use the simulated Weibull variates to sample θ and β. To achieve this, we apply the Gibbs sampling algorithm which happens to be a special case of the "Metropolis - Hastings algorithm" which in turn is an extension of the basic rejection sampling algorithm. The essence of this algorithm lies in the fact that it is easier to sample from a conditional distribution than to marginalize by integrating over a joint distribution. We obtain full conditionals on θ and β upto proportionality from these equations:-

$$p_1(\theta|\beta,x) \quad \propto \quad \theta^{\beta n}e^{-(\theta)^{\beta}\sum_{i=1}^{n}x_i^{\beta}} \tag{1.4}$$

$$p_2(\beta|\theta,x) \quad \propto \quad \beta^{n}\theta^{\beta n}\left(\prod_{i=1}^{n}x_i^{\beta-1}\right)e^{-(\theta)^{\beta}\sum_{i=1}^{n}x_i^{\beta}} \tag{1.5}$$

The Metropolis-Hastings algorithm is a Markov Chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution for which direct

sampling is difficult.

The Metropolis-Hastings algorithm can draw samples from any probability distribution $P(x)$, provided you can compute the value of a function $f(x)$ which is proportional to the density of P.The lax requirement that $f(x)$ should be merely proportional to the density, rather than exactly equal to it, makes the Metropolis-Hastings algorithm particularly useful, because calculating the necessary normalization factor is often extremely difficult in practice.

The Metropolis-Hastings algorithm works by generating a sequence of sample values in such a way that, as more and more sample values are produced, the distribution of values more closely approximates the desired distribution, $P(x)$. These sample values are produced iteratively, with the distribution of the next sample being dependent only on the current sample value (thus making the sequence of samples into a Markov chain). Specifically, at each iteration, the algorithm picks a candidate for the next sample value based on the current sample value. Then, with some probability, the candidate is either accepted (in which case the candidate value is used in the next iteration) or rejected (in which case the candidate value is discarded, and current value is reused in the next iteration) - the probability of acceptance is determined by comparing the likelihoods of the current and candidate sample values with respect to the desired distribution $P(x)$.

The Metropolis algorithm (which is a special case of the Metropolis-Hastings Algorithm) is given below :
Let $f(x)$ be a probability distribution that is proportional to the desired distribution $P(x)$.

1. Initialization: Choose an arbitrary point x_0 to be the first sample, and choose an arbitrary probability density $Q(x|y)$ which suggests a candidate for the next sample value x, given the previous sample value y. For the Metropolis algorithm, Q must be symmetric; in other words, it must satisfy $Q(x|y) = Q(y|x)$. A usual choice is to let $Q(x|y)$ be a Gaussian distribution centered at y, so that points closer to y are more likely to be visited next-making the sequence of samples into a random walk. The function Q is referred to as the proposal density or jumping distribution.

2. For each iteration t:

   - Generate a candidate $x^\star$ for the next sample by picking from the distribution $Q(x^\star|x_t)$.
   - Calculate the acceptance ratio $\alpha = \frac{f(x^\star)}{f(x_t)}$,which will be used to decide whether to accept or reject the candidate. Because f is proportional to the density of P , we have $\alpha = \frac{f(x^\star)}{f(x_t)} = \frac{P(x^\star)}{P(x_t)}$.
   - If $\alpha \geq 1$, then the candidate is more likely than x_t. Automatically accept the candidate by setting $x_{t+1} = x^\star$. Otherwise, accept the candidate with probability α; if the candidate is rejected, put $x_{t+1} = x_t$ instead. (To "accept a candidate with probability α", pick a uniformly distributed random number u between 0 and 1. If $u \leq \alpha$, accept; otherwise, reject).

This algorithm proceeds by randomly attempting to move about the sample space, sometimes accepting the moves and sometimes remaining in place. Note that the acceptance ratio a indicates how probable the new proposed sample is with respect to the current sample, according to the distribution $P(x)$. If we attempt to move to a point that is more probable than the existing point (i.e. a point in a higher-density region of $P(x)$), we will always accept the move. However, if we attempt to move to a less probable point, we will sometimes reject the move, and the more the relative drop in probability, the more likely we are to reject the new point. Thus, we will tend to stay in (and return large numbers of samples from) high-density regions of $P(x)$, while only occasionally visiting low-density regions. Intuitively, this is why this algorithm works, and returns samples that follow the desired distribution $P(x)$.

When using this algorithm for sampling from the target distributions above, the envelope distribution used is the "Exponential distribution" given by the density function

$$f(x;\theta) = \theta e^{-\theta x}, \quad x \geq 0 \tag{1.6}$$

Our aim is to achieve a simulation which displays maximum randomness of the generated θ and β. We expect a step function in each case (for θ and β) but the length of each step should be really small for a good sampling. This in turn means that the values should be accepted more frequently. To achieve a good acceptance condition, we have tried exponential distributions with different parameters. Finally we have picked the one that displays the desired randomness.

We have sampled from both the target distributions (for θ and β) simultaneously. For this, the following steps were used:-

1. We have assumed initial seeds θ_0 and β_0.
2. Using the generated set of Weibull variates and the seeds, we generate θ_1 from (1.5a).
3. We then use the generated θ_1 to sample from (1.5b) to obtain β_1.
4. β_1 is then in turn used to generate θ_2 and so on. This goes on in cycles until the desired number of values have been generated.

The algorithm is described in greater detail in the following section.

2 Algorithm

1. Generate $x^{\sim}$ from 1.4.
2. Assume β_0 and θ_0.
3. Generate $x^{\star}$ from the exponential distribution.
4. Calculate the acceptance ratio$a = min\left\{1, \frac{p_1(x^{\star}|\beta_t, x^{\sim})e^{-\theta x}}{p_1(\theta_t|\beta_t, x^{\sim})e^{-\theta x^{\star}}}\right\}$, where $p_1(\theta|\beta, x)$ is given by (1.5). We accept $x^{\star}$ with this probability. If $x^{\star}$ is accepted, then $\theta_{t+1} = x^{\star}$. Otherwise, $\theta_{t+1} = \theta_t$. (Here, θ_t and β_t refer to the θ and β generated in the previous iteration).
5. Generate $x^{\star}$ from the exponential distribution.
6. Calculate the acceptance ratio$a = min\left\{1, \frac{p_2(x^{\star}|\theta_t, x^{\sim})e^{-\theta x}}{p_2(\beta_t|\theta_t, x^{\sim})e^{-\theta x^{\star}}}\right\}$, where $p_2(\beta|\theta, x)$ is given by (1.6). We accept $x^{\star}$ with this probability. If $x^{\star}$ is accepted, then $\beta_{t+1} = x^{\star}$. Otherwise, $\beta_{t+1} = \beta_t$. (Here, θ_t refer to the θ generated in step 4 and β_t refers to the β generated in the previous iteration).
7. Go to step 3 till we get the number of values desired.

3 Code

Code for generating the distribution:-

```
# This function generates from the Weibull Distribution
# using the inverse transform method .
weibull <- function (l , b , N )
{
   set.seed(1221);
   u = runif ( N );
   ret = (1/l)*((-log(1-u))^(1/b));
      return ( ret );
}
# Markov Chain method .
theta_beta <- function( l0 , b0 , x_vec , l_exp) {
   l_t = l0
   b_t = b0
   ret_l <- vector()
   ret_b <- vector()
   ret_l[1] = l0
   ret_b[1] = b0
   N = length ( x_vec )
   e_dist_l = rexp (N , rate = l_exp )
   e_dist_b = rexp (N , rate = l_exp )
   u_l = runif ( N )
   u_b = runif ( N )
   for ( i in 1:( N -1) ) {
      tmp = (p1 (e_dist_l[i] , ret_b[i] , x_vec ) / p1 ( ret_l[ i ] , ret_b[ i ] , x
          _vec) ) * ( dist_q ( ret_l[ i ] , l_exp ) / dist_q ( e_dist_l[i] , l_exp )
           )
            if ( is.nan( tmp ) )
               tmp = 1
            alpha = min (1 , tmp )
            if ( u_l [ i ] <= alpha )
               ret_l [ i +1] = e_dist_l [ i ]
            else
               ret_l [ i +1] = ret_l [ i ]
      tmp = (p2 ( ret_l[ i +1] , e_dist_b[ i ] , x_vec ) / p2 ( ret_l[ i +1] , ret_b
          [i] , x_vec) )*( dist_q (ret_b[ i ] , l_exp ) / dist_q ( e_dist_b[ i ] , l
          _exp ) )
            if ( is.nan( tmp ) )
               tmp = 1
```

```
            alpha = min (1 , tmp )
            if ( u_b[ i ] <= alpha )
               ret_b[ i +1] = e_dist_b[ i ]
            else
               ret_b[ i +1] = ret_b[ i ]
      }
   return ( list ( ret_l , ret_b ) )
}

# Returns the value of the target distribution function p1 .
p1 <- function (l , b , x_vec ) {
   N = length ( x_vec )
   s_exp = sum ( exp ( x_vec^b ) )
   tmp = ( l^(b*N) ) * exp (-(l^b) * s_exp )
   return ( tmp )
}
# Returns the value of the exponential distribution q .
dist_q <- function (v , l_exp ) {
   return ( l_exp * exp ( - l_exp * v ) )
   }
# Returns the value of the target distribution function p2 .
p2 <- function (l , b , x_vec ) {
   N = length ( x_vec )
   s_exp = sum ( exp ( x_vec^b ) )
   tmp = (b^N)*( l^(b*N) )* ( prod ( x_vec )^(b -1) ) * exp (-(l^b) * s_exp )
   return ( tmp )
}
```

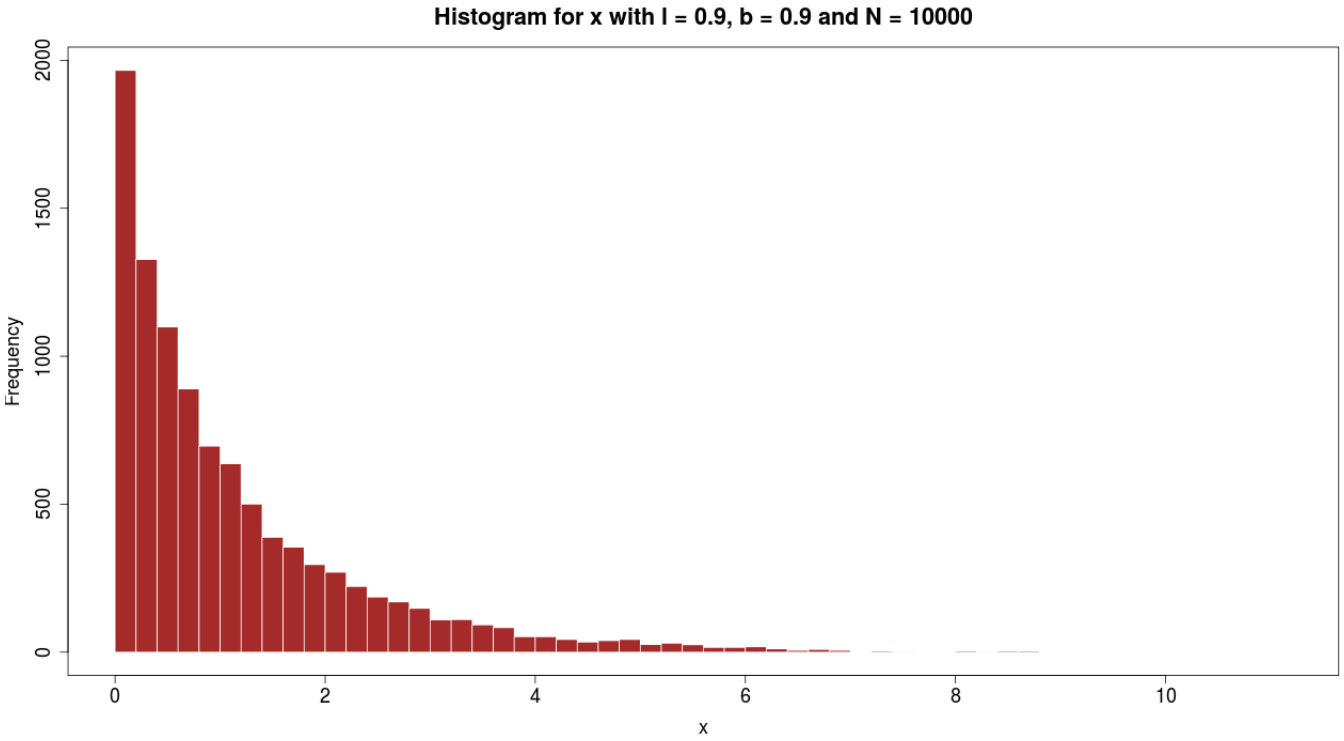
The graphs are plotted using the following piece of code:-

```
# Plots the histograns and the theta , beta distributions
# for a given value of the exponential distribution .
plot_theta_betas <- function ( exp_theta ) {
   theta_bet = c (0.9 , 0.9 , 2 , 2)
   let = c ( "A" , "B" , "C" , "D" )
   for ( i in 1:4) {
   x_vec = weibull(theta_bet[i] , theta_bet [ i ] , 10000)
   name = paste ( "Histogram" , let [ i ] , ".png" , sep = "" )
   png ( filename = name , bg = "white" , width =1280 , height =720)
   par ( ps =18)
   tit = paste ( "Histogram for x with l = " , theta_bet[ i ] , ", b = ", theta_bet
      [ i ] , " and N = 10000" , sep = "" )
```

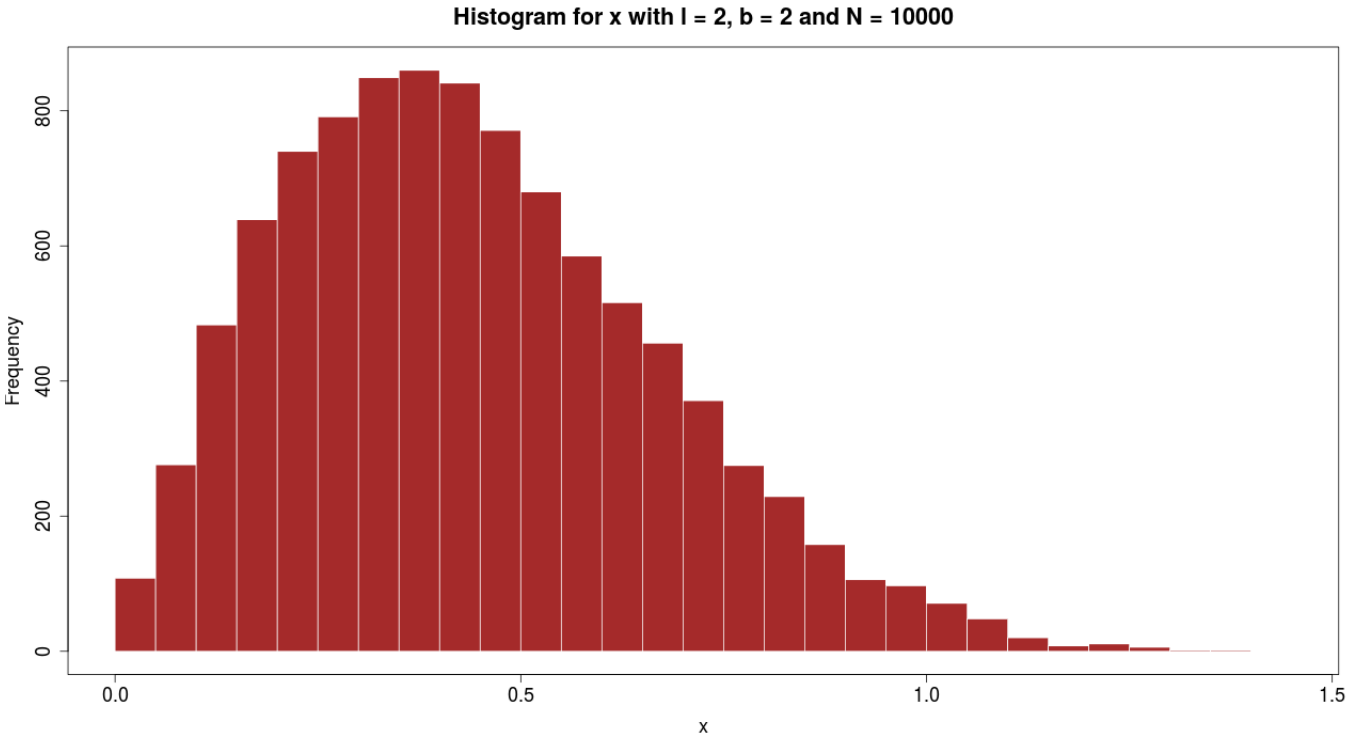
```
12      hist ( x_vec , breaks =49 , col = "black" , border = "white" , xlab = "x" , main=
            tit )
13      box()
14      dev.off ()
15      x_vec = x_vec [1:100]
16      #x_vec = weibull ( theta_bet [ i ] , theta_bet [ i ] , 100)
17      l_b = theta_beta ( theta_bet [ i ] , theta_bet [ i ] , x_vec , exp_theta )
18      name = paste ( "Plot" , let[ i ] , "1" , ".png" , sep = "" )
19       png ( filename = name , bg = "white" , width =1280 , height =720)
20      par ( ps =18)
21      tit = paste ( "Plot of theta vs i with exponential distribution s l= " , exp_
            theta , sep = "" )
22      plot ( l_b[[1]] , type = "l" , ylab = "theta" , xlab = "i" , main = tit )
23      box()
24      dev.off()
25      name = paste ( "Plot" , let[i] , "2" , ".png" , sep = "" )
26      png ( filename = name , bg = "white" , width =1280 , height =720)
27      par ( ps =18)
28      tit = paste ( "Plot of beta vs i with exponential distribution s l =" , exp_
            theta , sep = "" )
29      plot ( l_b[[2]] , type = "l" , ylab = "beta" , xlab = "i" , main = tit )
30      box()
31      dev.off()
32  }
33  }
```

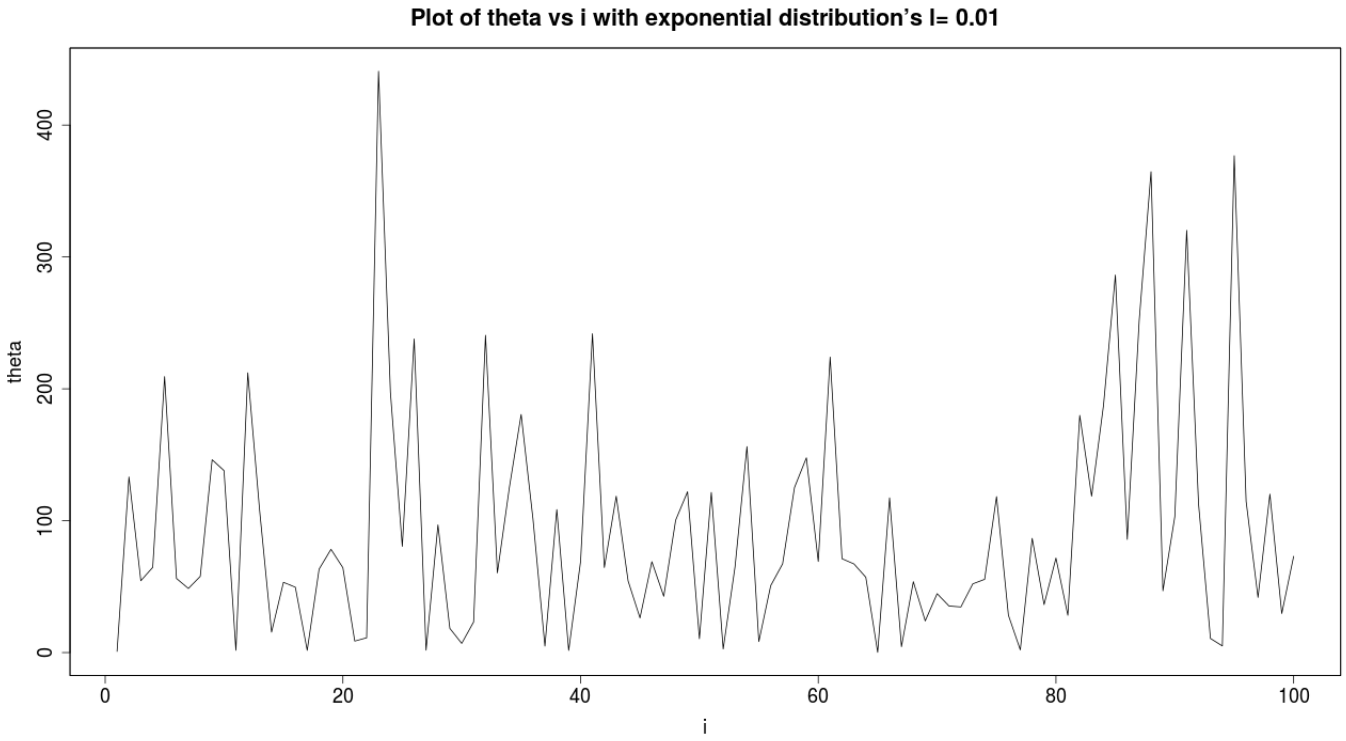
4 Results

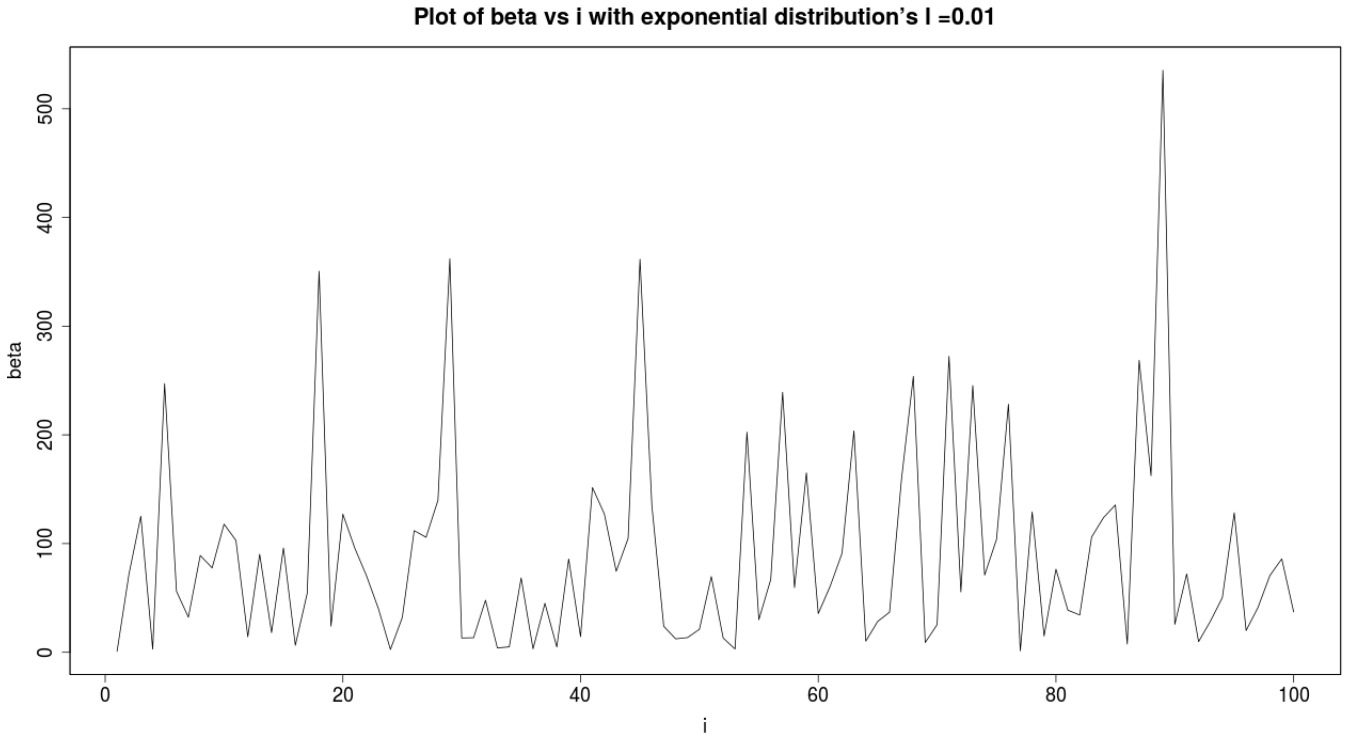
Histogram for x which follows the Weibull Distribution with $\theta = 0.9, \beta = 0.9$,N=10000:-

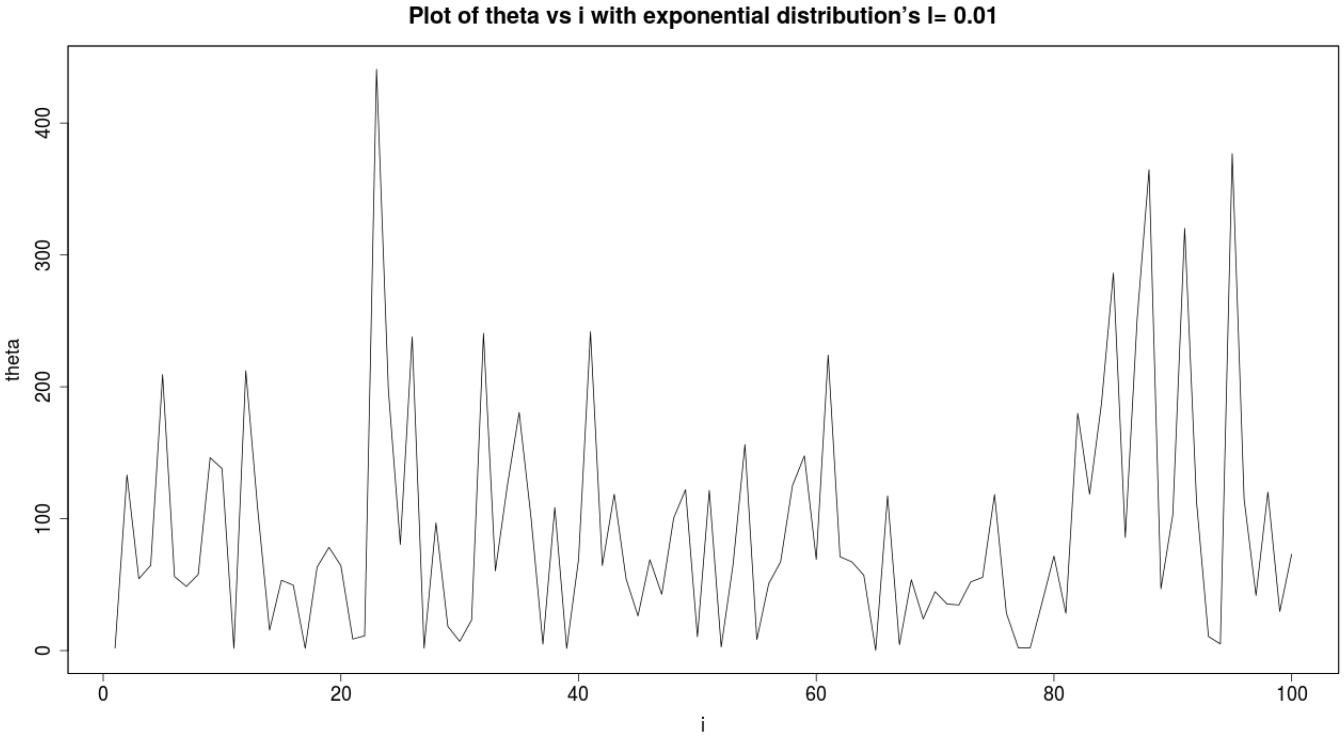


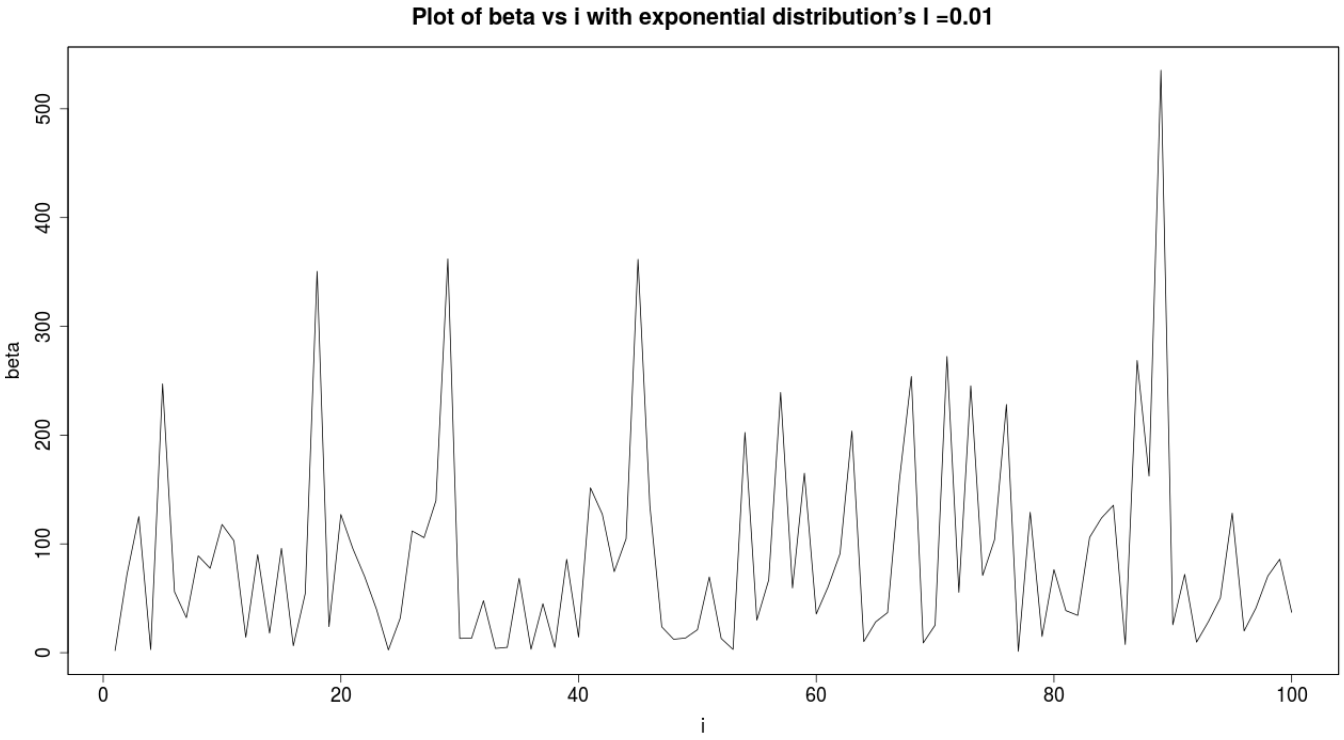
Histogram for x which follows the Weibull Distribution with $\theta = 0.9, \beta = 0.9$,N=10000:-




Plot of theta vs i with exponential distribution's l= 0.01
theta
i
0
100
200
300
400
0
20
40
60
80
100


Plot of beta vs i with exponential distribution's l =0.01
beta
i
0
100
200
300
400
500
0
20
40
60
80
100


Plot of theta vs i with exponential distribution's l= 0.01
theta
i
0
100
200
300
400
0
20
40
60
80
100


Plot of beta vs i with exponential distribution's l =0.01
beta
i
0
100
200
300
400
500
0
20
40
60
80
100