MATH48122 - Coursework 2

Student ID: 10876957

Question 1

The code for obtaining a sample of size 10000 from the posterior distribution of (ϵ, λ) is shown in Figure 1 below:

```
obs=c(52,10,25,8,7,3,0) #n=(n_0,n_1,...,n_6)
#Independent sampler
widowInd=function(obs,run,eps,lam,p) #p is the vector of parameters from proposals
{
output=matrix(0,ncol=2,nrow=run)
for(i in 1:run)
{
epsnew=rbeta(1,p[1], p[2]) # New value for epsilon
lamnew=rgamma(1,p[3], p[4]) # New value for lambda

liknew0= (epsnew+(1-epsnew)*(exp(-lamnew)) ) ^ (obs[1]) # the first part of the likelihood
    function for epsnew and lamnew
liknew=0;
for (k in 1:6) {
liknew[k]= ( (1-epsnew)* ( (lamnew^(k))/(factorial(k)) ) * (exp(-lamnew)) )^ (obs[k+1]) #the
    second part of the likelihood function for epsnew and lamnew
}
lik0= (eps+(1-eps)*(exp(-lam)) ) ^ (obs[1]) #the first part of the likelihood fucntion for eps
    and lam
lik=0;
for (m in 1:6) {
lik[m]= ( (1-eps)* ( (lam^(m))/(factorial(m)) ) * (exp(-lam)) )^ (obs[m+1]) #the second part of
     the likelihood function for eps and lam
}

num=liknew0*prod(liknew[1:k]) #numerator of the likelihood ratio
den=lik0*prod(lik[1:m]) #denominator of the likelihood ratio
ratio=num/den # Likelihood ratio
ratio=ratio*dbeta(eps,p[1], p[2])/dbeta(epsnew,p[1], p[2]) # proposals for epsilon
ratio=ratio*dgamma(lam,p[3], p[4])/dgamma(lamnew,p[3], p[4]) # proposals for lambda
u=runif(1)
if(u<ratio)
{
eps=epsnew
lam=lamnew
} # If proposed value is accepted, set epsilon and lambda equal to the proposed values.
output[i,]=c(eps,lam)
}
output
}
```

Figure 1: R code using Independent sampler to obtain samples from the posterior distribution of (ϵ, λ)

It is given that the likelihood can be written as:

$$L(\epsilon, \lambda|\mathbf{n}) \propto (\epsilon + (1-\epsilon)exp(-\lambda))^{n_0} \prod_{k=1}^{6}((1-\epsilon)\frac{\lambda^k}{k!}exp(-\lambda))^{n_k} \quad (1)$$

Code implementation: First, note that out vector p of parameters from the two proposals is equal to $p = (a, b, \alpha, \beta)$, where $Beta(a, b)$ is the proposal for ϵ and $Gamma(\alpha, \beta)$ is the proposal for λ. On lines 8 and 9 in Figure 1 we set the new values for ϵ and λ. On line 11 we implement just the first part: $(\epsilon+(1-\epsilon)exp(-\lambda))^{n_0}$ from Equation 1 for the new values we defined above. On lines 12 to 15 we implement the rest of Equation 1 for the same values. Therefore, on line 22 we find the full likelihood Equation 1 for the new values. Similarly, we do the same for the initial ϵ and λ on lines 16 to 20 to get their likelihood equation too, found on line 23. On line 24 it is implemented the final likelihood ratio needed. On lines 25 and 26 we update the ratio by adding the according proposals and priors for both ϵ and λ, provided in the question. On lines 27 to 32 we use "if statement" to check if the proposed ratio value is accepted and if it is, then we set ϵ and λ to the proposed new values we defined. On line 33 we define a matrix in which we will store the outputs for ϵ and λ after running the algorithm for the chosen number of times.

Now, let us run the algorithm by the following code:

```
1 p <- c(4,6,4,2) #set initial values for a,b, alpha and beta respectively
2 # Run the independent sampler
3 indsamp <- widowInd(obs, 10100, 0.5, 1, p) #run the independent sampler, where we set 0.5 and 1
      for initial values for epsilon and lambda respectively
4 indsamp<- indsamp[101:10100,] #discard the first 100 iterations as burn-in
5 plot.ts(indsamp) #generate a trace plot
6 plot(indsamp[,1],pch=20) #analysing epsilon for all the iterations
7 plot(indsamp[,2],pch=20) #analysing lambda for all iterations
```

Figure 2: R code for running the Independent sampler

Tuning the parameters of our proposals can be done by analysing the trace plot and the plots for ϵ and λ for different parameters of their proposals for all the iterations. These plots are produced by the code shown on lines 5 to 7 in Figure 2. Note that on line 1, the values in vector p are the ones we have chosen for our final ones after the process of tuning, which was as follows: In our first test, we choose the vector for our parameters p as $p = (1, 1, 4, 4)$. As a result, we get the plots shown in Figures 3, 4 and 5.

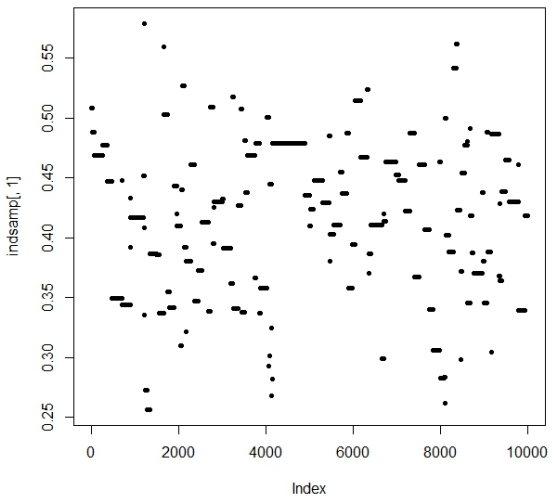

Figure 3: Proposal for ϵ

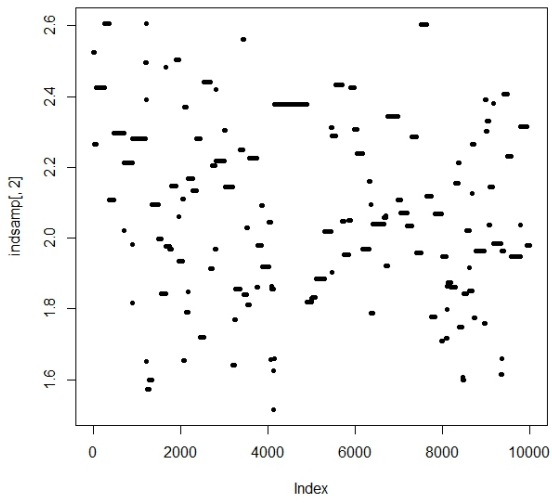

Figure 4: Proposal for λ

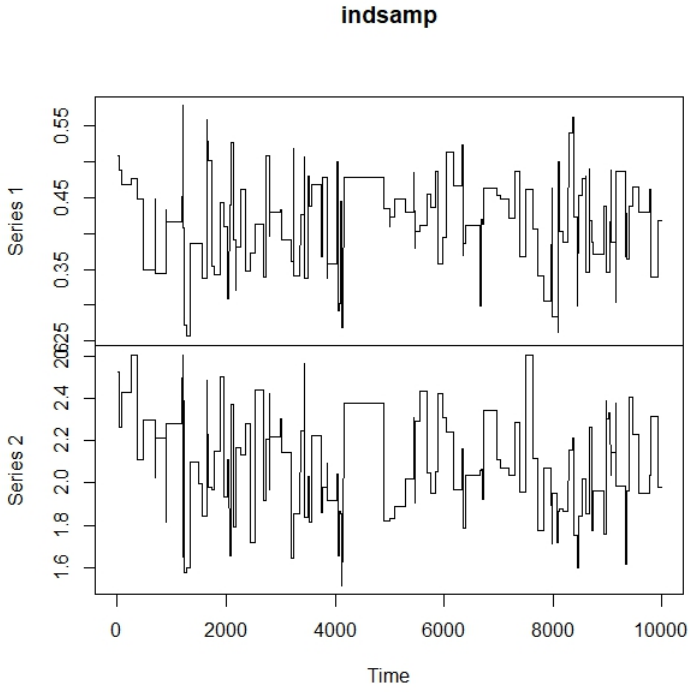

Figure 5: Trace plot

However, it can be seen that in both Figures 3 and 4, there are some long stretches and the algorithm remains at the same value for long periods of time. In Figure 5 we can also observe that the algorithm does not converge well. Therefore, we can conclude that this choice for p is quite inefficient. We need to keep looking for another one, which moves around the space more frequently. In order to do that, we should note that the means for ϵ and λ are approximately 0.4 and 2, respectively. Based on this information, in our second test we will set the means of the proposals to be close to the means of the posterior distributions.

Hence, our next choice is $p = (4, 6, 4, 2)$, which we actually used to run our algorithm in Figure 2. In this case, our plots for the proposals for ϵ and λ are shown in Figures 6 and 7:

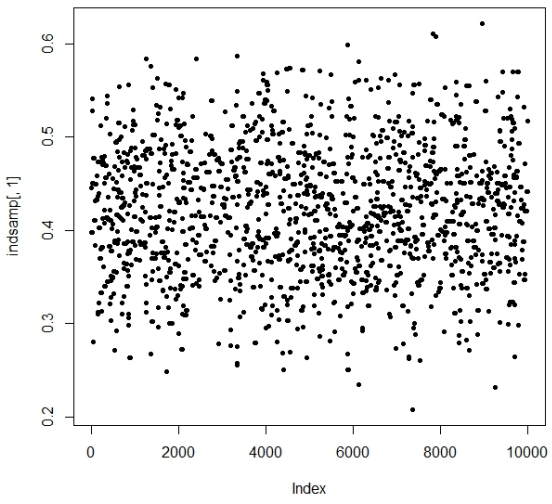

Figure 6: Proposal for ϵ

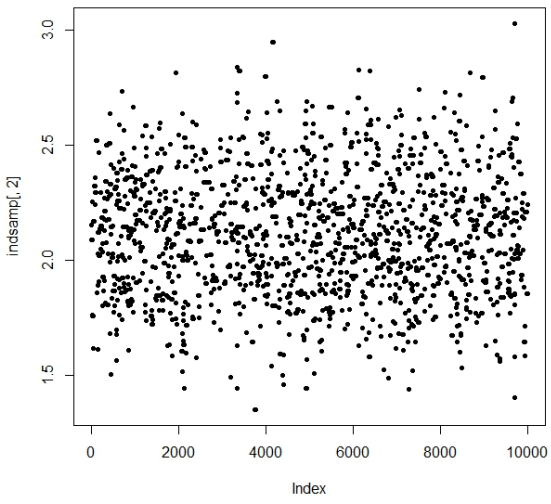

Figure 7: Proposal for λ

Here, the algorithm for both graphs moves around the space quicker, explores the distribution faster and it is more representative, since it captures the variability of the distribution better. It can be concluded that it is quite efficient, because the proposal distribution reflects the posterior distribution.

Now, let us observe the trace plots, presented in Figure 8:

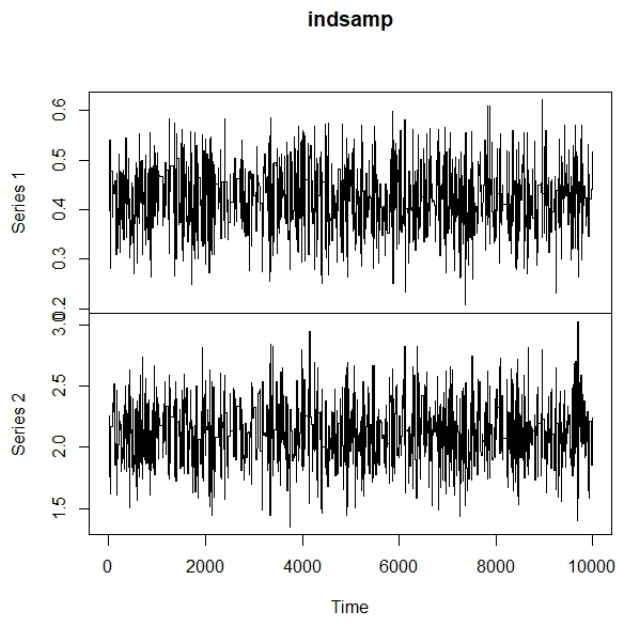

Figure 8: Trace plot

From the plot, we could say that MCMC mixing looks good, since it shows movement around the possible values without obvious patterns and trends. It also presents convergence, indicating that the sampler has reached a stable distribution and therefore the Independent sampler is efficient in exploring the space. Moreover, the sampled values for ϵ and λ fluctuate around their means. Now, plotting the posterior vs the proposal distribution can help us check the dispersion. The graph is shown in Figure 9:

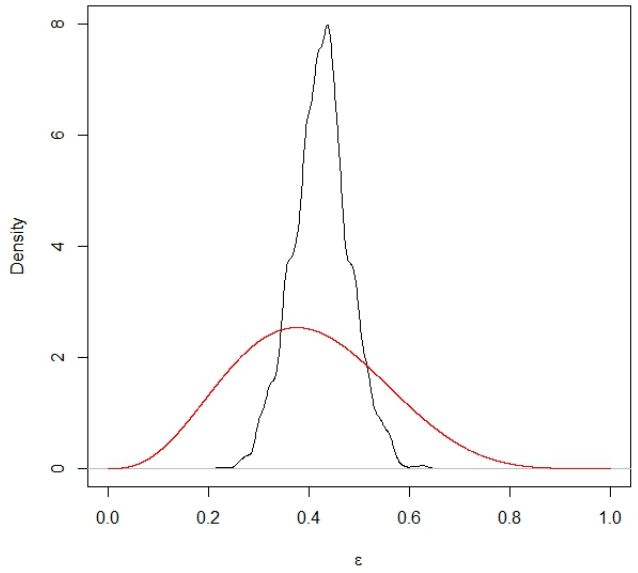

Figure 9: Posterior vs proposal distribution

As we aim for making the proposal less dispersed, the acceptance rate here looks to be reasonably high, which is what we want to achieve with the Independent sampler.

Question 2

```
1  obs=c(52,10,25,8,7,3,0) #n=(n_0,n_1,...,n_6)
2  #RWM
3  widowRMW=function(obs,run,eps,lam,sigma1,sigma2) #sigma1 and sigma2 are the proposal sds
4  {
5  accnum <- 0 #number of accepted values
6  output=matrix(0,ncol=2,nrow=run)
7  for(i in 1:run)
8  {
9  epsnew=rnorm(1,eps,sigma1)
10 lamnew=rnorm(1,lam,sigma2)
11
12 liknew0= (epsnew+(1-epsnew)*(exp(-lamnew)) ) ^ (obs[1]) #the first part of the likelihood
       function for epsnew and lamnew
13 liknew=0;
14 for (k in 1:6) {
15 liknew[k]= ( (1-epsnew)* ( (lamnew^(k))/(factorial(k)) ) * (exp(-lamnew)) )^ (obs[k+1]) #the
       second part of the likelihood function for epsnew and lamnew
16 }
17 lik0= (eps+(1-eps)*(exp(-lam)) ) ^ (obs[1]) #the first part of the likelihood fucntion for eps
       and lamnew
18 lik=0;
19 for (m in 1:6) {
20 lik[m]= ( (1-eps)* ( (lam^(m))/(factorial(m)) ) * (exp(-lam)) )^ (obs[m+1]) #the second part of
        the likelihood function for eps and lam
21 }
22 num=liknew0*prod(liknew[1:k]) #numerator of the likelihood ratio
23 den=lik0*prod(lik[1:m]) #denominator of the likelihood ratio
24 ratio=num/den # Likelihood ratio
25
26 u=runif(1)
27 if(u<ratio)
28 {
29 eps=epsnew
30 lam=lamnew
31 accnum <- accnum + 1 #counting the accepted values
32 }
33 output[i,]=c(eps,lam)
34 }
35 return(list(output, accnum / run)) #returning both the output and acceptance rate in a list
36 }
```

Figure 10: R code using a RWM algorithm to obtain samples from the posterior distribution of (ϵ, λ)

Code implementation: The main difference with the RWM algorithm compared to the Independence sampler is that here we use proposed standard deviations, denoted by σ_1 and σ_2, which we incorporate in the function created on line 3. On line 5 we initiate a value, which will count the number of accepted values from the acceptance probability. On line 6 we set a matrix in which we will store the outputs for ϵ and λ from the posterior distribution. On lines 9 and 10 we define the new proposed values for ϵ and λ. On lines 12 to 16 we implement Equation 1 for these new values in a similar way as we did in Question 1. On lines 17 to 21 we do the same for the original values and then we can find the likelihood ratio, defined on line 24. On lines 26 to 32 we use an if statement to see which values are accepted and we count their number. On line 33 we store the values for ϵ and λ and on line 35 we set the function to return a list, which contains the

above output as its first element and the acceptance rate /calculated by dividing the number of accepted values over the number of iterations/ as its second one.

In order to select our proposal variances, we run the algorithm with some different values for σ_ϵ and σ_λ until the acceptance rate is approximately 25% or above, but less than 50%. First let us choose them to be 1's. We use the code shown in Figure 11 /Note that this is not the final code we will use for running the algorithm, since these values are not tuned yet./:

```
1 rwm <- widowRMW(obs, 10000, 0.5, 1, 1, 1)
2 accrate <- rwm[[2]] #calculating the acceptance rate
3 accrate
4 plot.ts(rwm[[1]]) #creating the trace plot
```

Figure 11: R code for calculating the acceptance probability and creating trace plots for the first choice of proposal standard deviations.

After running the code in Figure 11, we see that the acceptance rate is 0.2512, which is good. However, the trace plots show us that there is something wrong when looking at the samples for σ_1. They are shown in Figure 12. Therefore, we should try some lower values for the standard deviations.

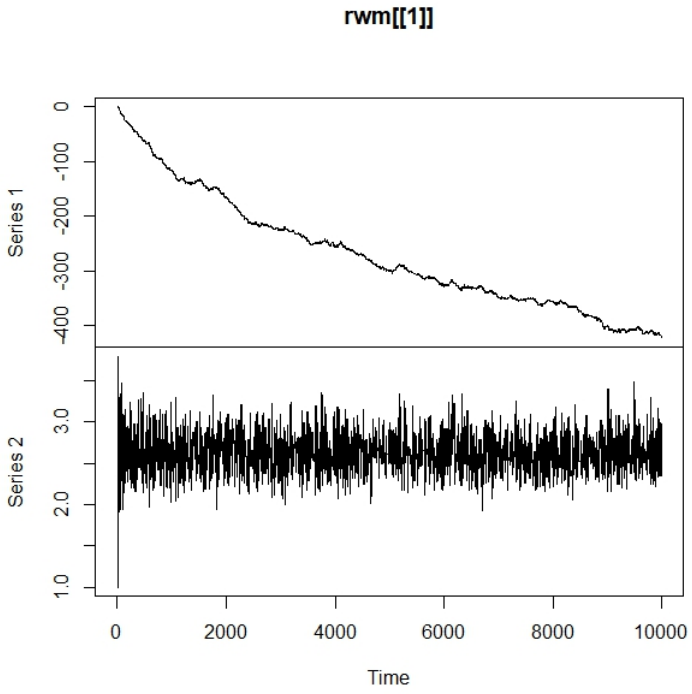

Figure 12: Trace plot - test 1

For our second test, let us choose 0.05 for both σ_1 and σ_2. Consequently, we observe that the calculated acceptance rate is too high - 0.7172 so we should make the values higher. The according trace plot is the one shown in Figure 13:

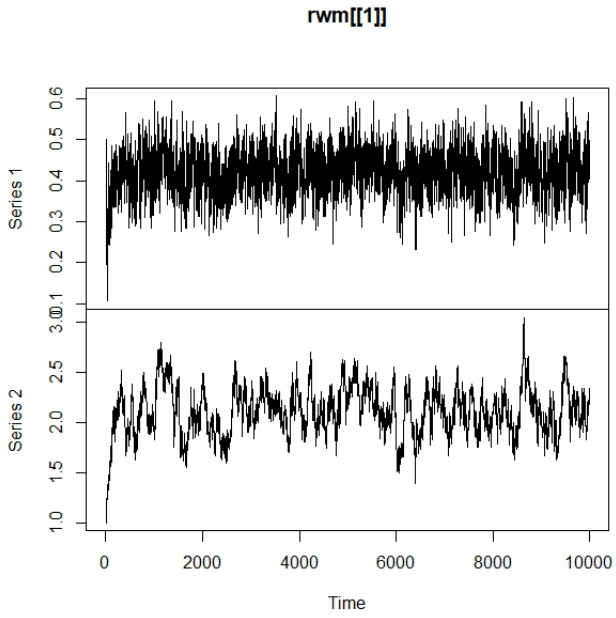

Figure 13: Trace plot - test 2

It looks relatively good for σ_1, but maybe the value needs to be set a bit higher in order to lower the acceptance rate. For σ_2, the algorithm does not converge well and it behaved better when the value was higher. Therefore, we can choose a compromised value of 0.5 for σ_2. This explains our final tuning of the parameters.

Finally, choosing σ_1 as 0.1 and σ_2 as 0.5, gives us an acceptance rate 0.2799, which is in the desired range. Moreover, the trace plots show that the algorithm converges well, since it explores the space efficiently and there are no long periods where it has the same value. They are presented in Figure 14:

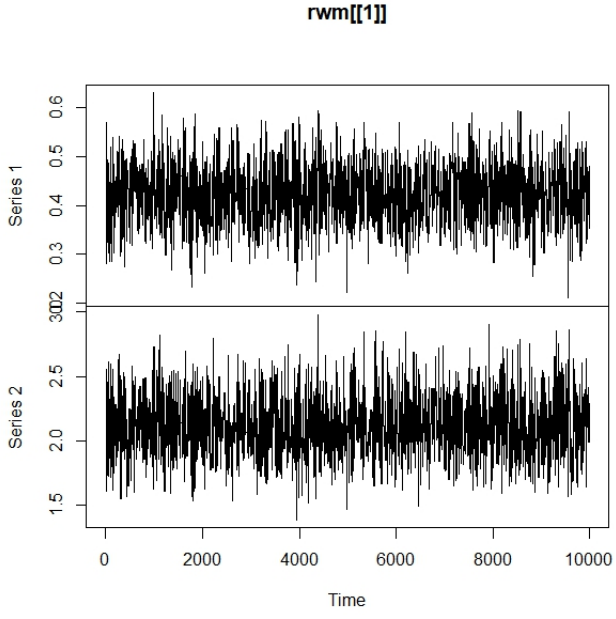

Figure 14: Trace plot - final test

To conclude, we should use the final code in Figure 15 to run our RWM algorithm:

```
rwm <- widowRMW(obs, 10100, 0.5, 1, 0.1, 0.5) #running the algorithm with our final values
rwm<- rwm[[1]][101:10100,] #burn-in
plot.ts(rwm) #creating the trace plot
```

Figure 15: R code for running the RWM algorithm for our final choice of proposal standard deviations.

Question 3

The code for obtaining the posterior means and standard deviations for ϵ and σ for both algorithms are presented in Figure 16:

```
mean(indsamp[,1]) #mean for epsilon using Independent sampler
sd(indsamp[,1]) #standard deviation for epsilon using Independent sampler
mean(indsamp[,2]) #mean for lambda using Independent sampler
sd(indsamp[,2]) #standard deviation for lambda using Independent sampler

mean(rwm[, 1]) #mean for epsilon using RWM algorithm
sd(rwm[, 1]) #standard deviation for epsilon using RWM algorithm
mean(rwm[, 2]) #mean for lambda using RWM algorithm
sd(rwm[, 2]) #standard deviation for lambda using RWM algorithm
```

Figure 16: R code for calculating the means and standard deviations for ϵ and λ for both algorithms.

The code output we get is visible in Figure 17.

```
> mean(indsamp[,1]) #mean for epsilon using Independent sampler
[1] 0.4221315
> sd(indsamp[,1]) #standard deviation for epsilon using Independent sampler
[1] 0.05717314
> mean(indsamp[,2]) #mean for lambda using Independent sampler
[1] 2.106076
> sd(indsamp[,2]) #standard deviation for lambda using Independent sampler
[1] 0.2203875
>
> mean(rwm[, 1]) #mean for epsilon using RWM algorithm
[1] 0.4234345
> sd(rwm[, 1]) #standard deviation for epsilon using RWM algorithm
[1] 0.05781134
> mean(rwm[, 2]) #mean for lambda using RWM algorithm
[1] 2.117957
> sd(rwm[, 2]) #standard deviation for lambda using RWM algorithm
[1] 0.2220783
```

Figure 17: R output for the means and standard deviations for ϵ and λ.

As it can be observed, the mean for ϵ is 0.4221315 when using the Independent sampler algorithm and 0.4234345 when using the RWM one /lines 2 and 11 in Figure 17/. Its standard deviations for both algorithms are 0.05717314 and 0.05781134 /lines 4 and 13/, respectively. Regarding λ, its means are 2.106076 and 2.117957, whereas its standard deviations are 0.2203875 and 0.2220783 for Independent sampler and RWM algorithm, respectively. In conclusion, when approximated, these values are almost identical to each other for both IS and RWM. Hence, in our case, the two algorithms are efficient and it does not make a great difference which one of them we will choose.

Question 4

Let us now write a program which will estimate the number of widows with 0 to 6 children, where the total number of widows is 105. The code used is shown in Figure 18:

```
#Create a function to find the pmf for the Zero-Poisson distribution
zp <- function(k, eps, lam) {
  pmf <- ( (1-eps)* ((lam^k)/(factorial(k)) ) * (exp(-lam)) ) + eps*(k==0)
  return(pmf)
}

#Create a function to find the predictive distribution and the mean, where we define the
    samples for epsilon and lambda we found after running the algorithms with epssamp and
    lamsamp
pd <- function(epssamp, lamsamp) {
  preddistr <- rep(0, 7)
  for (i in 1:length(epssamp)) {
    for (k in 0:6) {
      preddistr[k + 1] <- preddistr[k + 1] + zp(k, epssamp[i], lamsamp[i])
    }
  }
  return(preddistr/ length(epssamp))
}
indeps <- indsamp[,1] #samples from Independent sampler for epsilon
indlam <- indsamp[,2] #samples from Independent sampler for lambda
rwmeps <- rwm[,1] #samples from RWM algorithm for epsilon
rwmlam <- rwm[,2] #samples from RWM algorithm for lambda
indest = 105*pd(indeps,indlam) #estimated number of widows with 0 to 6 children respectively
    for the Independent sampler
rwmest = 105*pd(rwmeps,rwmlam) #estimated number of widows with 0 to 6 children respectively
    for the RWM algorithm
```

Figure 18: R code for estimating the number of widows with 0 to 6 children

The output we get for both algorithms is quite similar and it is shown in Figure 19:

```
> indest
[1] 51.9473971 15.6350591 16.2084724 11.3235393 5.9969668 2.5679273 0.9260479
> rwmest
[1] 51.9670514 15.4902057 16.1597179 11.3619199 6.0563381 2.6102470 0.9474072
```

Figure 19: R code for estimating the number of widows with 0 to 6 children

Approximating the numbers we got from both algorithms and putting them in Table 1 results in:

No. of children	0	1	2	3	4	5	6
Observed no. of widows for IS	52	16	16	11	6	3	1
Observed no. of widows for RWM	52	15	16	11	6	3	1

Table 1: Estimated number of widows with the corresponding number of children

Comparing it with the original one, we can observe that the estimated number of widows with 0 children is identical - probably because of the higher value. However, the approximated number of widows having

just one child is not that accurate and there is a substantial difference - it is 16 for IS and 15 for RWM, whereas the true value is 10. Almost the same amount of difference is observed for widows, having 2 children, where the algorithms estimated a smaller value. For the rest of the numbers, there are also some discrepancies, but they are small ones.

The zero-inflated Poisson model approximates most of the values accurately, taking into account the fact that the sample size of 105 is relatively small. Comparing the model estimates and the true values, we can draw similar conclusions - almost half of the widows do not have any children and a small number of widows have more than 3 children. Hence, it is quite a good fit.