**MA5750: APPLIED STATISTICS**

**ASSIGNMENT 4**

**E Naveen**

**ME16B077**

**9.1**

**R Code:**

# Set of values of pi for Monte-Carlo Simulation  
pi\_values = seq(from = 0.1, to = 0.9, by = 0.1)  
  
# Initialize a few required variables with zeros  
bias\_f = numeric(length = length(pi\_values))  
bias\_b = numeric(length = length(pi\_values))  
variance\_f = numeric(length = length(pi\_values))  
variance\_b = numeric(length = length(pi\_values))  
MS\_f\_1 = numeric(length = length(pi\_values))  
MS\_b\_1 = numeric(length = length(pi\_values))  
MS\_f\_2 = numeric(length = length(pi\_values))  
MS\_b\_2 = numeric(length = length(pi\_values))  
  
# Iterate over the different values of pi  
for (pi\_index in 1:length(pi\_values)) {  
   
 # Initialize a few required variables with zeros  
 pi\_f = numeric(length = 5000)  
 pi\_b = numeric(length = 5000)  
 square\_dist\_f = numeric(length = 5000)  
 square\_dist\_b = numeric(length = 5000)  
   
 # (i)  
 # Iterate over 5000 instances for Monte-Carlo Simulations  
 for (N in 1:5000) {  
 # Generate uniform random varible from 0 to 1.  
 pi\_montecarlo = runif(10, 0, 1)  
 # y is the count of r.v.'s out of 10 from above that are less than  
 # the pi value.  
 y = sum(pi\_montecarlo < pi\_values[pi\_index])  
 n = 10  
 # Store the Frequentist and Bayesian estimates for Pi.  
 # (ii)  
 pi\_f[N] = (y)/(n)  
 # (iii)  
 pi\_b[N] = (y+1)/(n+2)  
 # Calculate squared errors of estimates from actual value.  
 square\_dist\_f[N] = (pi\_f[N] - pi\_values[pi\_index])\*\*2  
 square\_dist\_b[N] = (pi\_b[N] - pi\_values[pi\_index])\*\*2  
 }  
   
 # Find Bias and Variance of sample distribution.  
 # (iv)  
 bias\_f[pi\_index] = mean(pi\_f) - pi\_values[pi\_index]  
 bias\_b[pi\_index] = mean(pi\_b) - pi\_values[pi\_index]  
 # (v)  
 variance\_f[pi\_index] = var(pi\_f)  
 variance\_b[pi\_index] = var(pi\_b)  
 # (vi)  
 # Find Mean squared Error using Bias and Variance.  
 MS\_f\_1[pi\_index] = (bias\_f[pi\_index]\*\*2) + (variance\_f[pi\_index])  
 MS\_b\_1[pi\_index] = (bias\_b[pi\_index]\*\*2) + (variance\_b[pi\_index])  
 # Find Mean Squared Error by taking average of squared errors.  
 MS\_f\_2[pi\_index] = mean(square\_dist\_f)  
 MS\_b\_2[pi\_index] = mean(square\_dist\_b)  
   
}  
  
# Print the Mean squared error calculated by the two methods to check they are same.  
cat('Frequentist \n','First Method', MS\_f\_1,'\nSecond Method', MS\_f\_2)

**## Frequentist   
## First Method** 0.009073815 0.01553711 0.02138228 0.02376475 0.02461492 0.02406081 0.02107821 0.01574315 0.009191838   
**## Second Method** 0.009072 0.015534 0.021378 0.02376 0.02461 0.024056 0.021074 0.01574 0.00919

cat('Bayesian \n','First Method', MS\_b\_1,'\nSecond Method', MS\_b\_2)

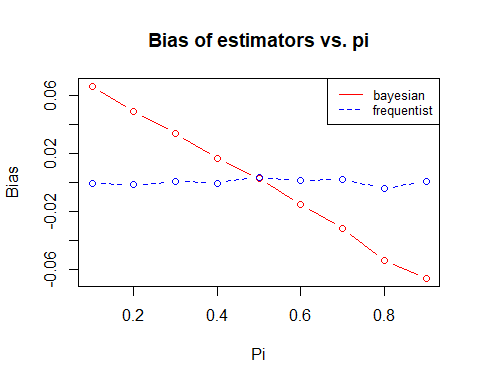
**## Bayesian   
## First Method** 0.01068348 0.01316132 0.01601214 0.01677108 0.01709369 0.01693779 0.01564098 0.01378607 0.01071878   
**## Second Method** 0.01068222 0.01315917 0.01600917 0.01676778 0.01709028 0.01693444 0.01563806 0.01378389 0.0107175

* The mean squared of the estimator produce the same result when done using either bias and variance (or) mean of squared error as seen in the output printed above.
* The small changes in values are purely due to the numerical approximation done by the computer.

**(b)**

**R Code:**

# Plot Biases of the two estimators versus pi  
plot(pi\_values, bias\_b, type='b', col = 'red', main = 'Bias of estimators vs. pi',  
 xlab = 'Pi', ylab = 'Bias')  
lines(pi\_values, bias\_f, type='b', col='blue', lty=2)  
legend(x= "topright", legend=c("bayesian","frequentist"),  
 col=c("red", "blue"), lty=1:2, cex=0.8)

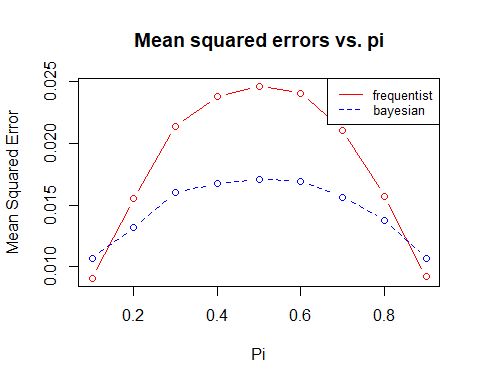


1. By observation we can see that Frequentist estimator is Unbiased owing to values of Bias close to 0.
2. On the other hand, the Bayesian estimate is Biased and we see a downwards trend with . This is in accordance to the theory that we studied earlier.

**(c)**

**R Code:**

# Plot mean squared errors of the two estimators versus pi  
plot(pi\_values, MS\_f\_1, type='b', col = 'red', main = 'Mean squared errors vs. pi',  
 xlab = 'Pi', ylab = 'Mean Squared Error')  
lines(pi\_values, MS\_b\_1, type='b', col='blue', lty=2)  
legend(x= "topright", legend=c("frequentist","bayesian"),  
 col=c("red", "blue"), lty=1:2, cex=0.8)



1. The Graph plotted above does resemble the one (Fig 9.2) from the textbook.
2. When takes on values in the range the Bayesian estimator have smaller mean squared error than that of Frequentist estimator.

**10.2**

Jeffrey’s prior for Poisson parameter is given by,

We know any gamma distribution is given by,

Comparing the above two we have, and . Hence will give this prior.

**R Code:**

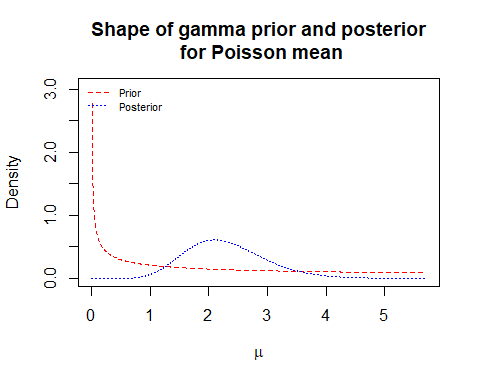
# Import the necessary library  
library('Bolstad')

##   
## Attaching package: 'Bolstad'

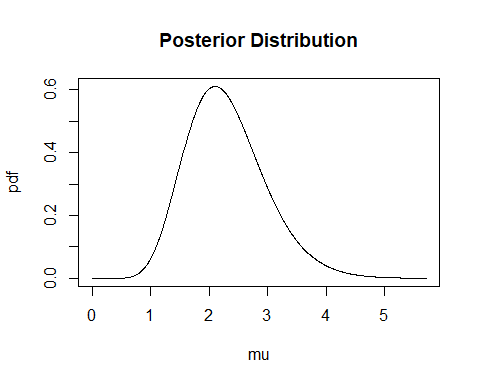
## The following objects are masked from 'package:stats':  
##   
## IQR, sd, var

# Store the observation (data) in y  
y = c(3,4,3,0,1)  
  
# (b)  
# Use posgamp command to find posterior for the given prior distribution  
result = poisgamp(y, 1/2, 0)

## Summary statistics for data  
## ---------------------------  
## Number of observations: 5   
## Sum of observations: 11   
##   
## Summary statistics for posterior  
## --------------------------------  
## Shape parameter (r): 11.5   
## Rate parameter (v): 5   
## 95% credible interval for mu: [1.17, 3.81]



plot(result$mu, result$posterior, type='l', main = 'Posterior Distribution',  
 xlab = 'mu', ylab = 'pdf')



**R Code:**

# (c)  
# Calculate Mean and Median  
E\_posterior = sintegral(result$mu, result$mu\*result$posterior, n.pts = length(result$mu))  
median\_posterior = result$mu[which.max(result$posterior)]  
# Print Mean and Median  
cat('Posterior\nMean =', E\_posterior$value, '\nMedian =', median\_posterior)

## Posterior  
## Mean = 2.299401   
## Median = 2.100347

* Posterior Mean = 2.299401
* Posterior Median = 2.100347

**R Code:**

# (d)  
# Find 95% Bayesian credible interval using 'which.max' function  
cdf\_1 = sintegral(result$mu, result$posterior, n.pts = length(result$mu))  
cdf = cdf\_1$cdf  
lower\_bound = cdf$x[with(cdf,which.max(x[y<=0.025]))]  
upper\_bound = cdf$x[with(cdf,which.max(x[y<=0.975]))]  
cat(paste("Approximate 95% Bayesian credible interval using 'which.max' function: \n[", round(lower\_bound, 4),  
 " ", round(upper\_bound, 4), "]\n", sep = ""))

**## Approximate 95% Bayesian credible interval using 'which.max' function:   
## [1.1603 3.8002]**

# Find 95% Bayesian credible interval using 'approxfun' function  
Finv = approxfun(cdf$y,cdf$x)  
lower\_bound = Finv(c(0.025))  
upper\_bound = Finv(c(0.975))  
cat(paste("Approximate 95% Bayesian credible interval using 'approxfun' function: \n[", round(lower\_bound, 4),  
 " ", round(upper\_bound, 4), "]\n", sep = ""))

**## Approximate 95% Bayesian credible interval using 'approxfun' function:   
## [1.166 3.8047]**

* 95% Bayesian Credible interval = [1.166 3.8047]

**10.6**

**(a)**

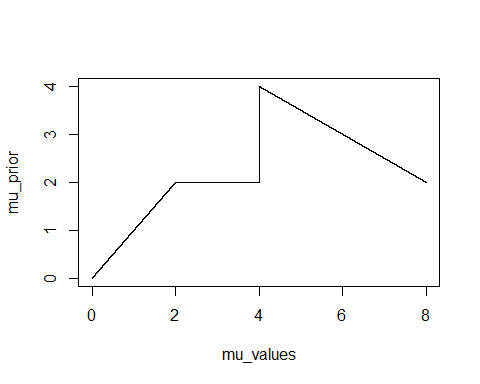
**R Code:**

# Import the necessary library  
library('Bolstad')

##   
## Attaching package: 'Bolstad'

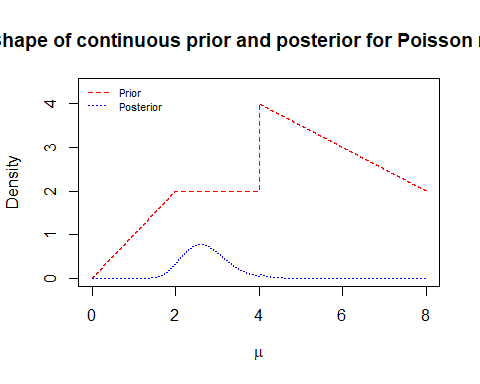
## The following objects are masked from 'package:stats':  
##   
## IQR, sd, var

# Store the observation (data) in y  
# NOTE: This time we'll be taking 10 observations as mentioned in Q)10.6  
y = c(3,4,3,0,1,1,2,3,3,6)  
  
mu\_values = seq(from = 0, to = 8, by=0.0001)  
  
mu\_prior = rep(0,length(mu\_values))  
mu\_prior[mu\_values<=2] = mu\_values[mu\_values<=2]  
mu\_prior[mu\_values>2 & mu\_values<=4] = 2  
mu\_prior[mu\_values>4 & mu\_values<=8] = 6 - mu\_values[mu\_values>4 & mu\_values<=8]/2  
mu\_prior[mu\_values>8] = 0  
  
plot(mu\_values,mu\_prior, type='l')

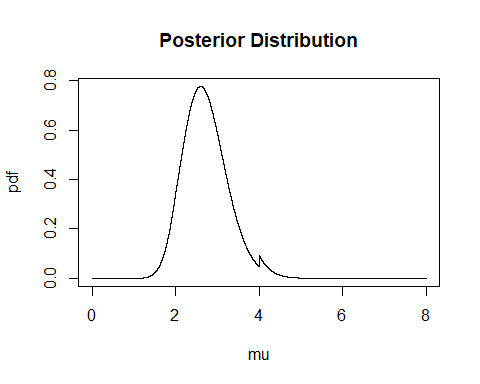


# (a)  
# Use posgcp command to find posterior for the given prior distribution  
result = poisgcp(y = y, density = "user", mu = mu\_values, mu.prior = mu\_prior)

## Summary statistics for data  
## ---------------------------  
## Number of observations: 10   
## Sum of observations: 26



plot(result$mu, result$posterior, type='l', main = 'Posterior Distribution',  
 xlab = 'mu', ylab = 'pdf')



**(b)**

# (b)  
# Calculate Mean, Median and Standard deviation  
E\_posterior = sintegral(result$mu, result$mu\*result$posterior, n.pts = length(result$mu))  
median\_posterior = result$mu[which.max(result$posterior)]  
# Print Mean and Median  
cat('Posterior\nMean =', E\_posterior$value, '\nMedian =', median\_posterior)

## Posterior  
## Mean = 2.724635   
## Median = 2.6

variance = sintegral(result$mu, ((result$mu-E\_posterior$value)^2)\*result$posterior, n.pts = length(result$mu))  
# Print Standard Deviation  
cat('\nStandard Deviation =', variance$value^0.5)

##   
## Standard Deviation = 0.5369072

* Posterior Mean = 2.724635
* Posterior Median = 2.6
* Posterior Standard Deviation = 0.5369072

**(c)**

# (c)  
# Find 95% Bayesian credible interval using 'approxfun' function  
cdf\_1 = sintegral(result$mu, result$posterior, n.pts = length(result$mu))  
cdf = cdf\_1$cdf  
Finv = approxfun(cdf$y,cdf$x)  
lower\_bound = Finv(c(0.025))  
upper\_bound = Finv(c(0.975))  
cat(paste("Approximate 95% Bayesian credible interval using 'approxfun' function: \n[", round(lower\_bound, 4),  
 " ", round(upper\_bound, 4), "]\n", sep = ""))

## Approximate 95% Bayesian credible interval using 'approxfun' function:   
## [1.8114 3.975]

* 95% Bayesian Credible interval = [1.8114 3.975]