Problem Set 10 – MATH392

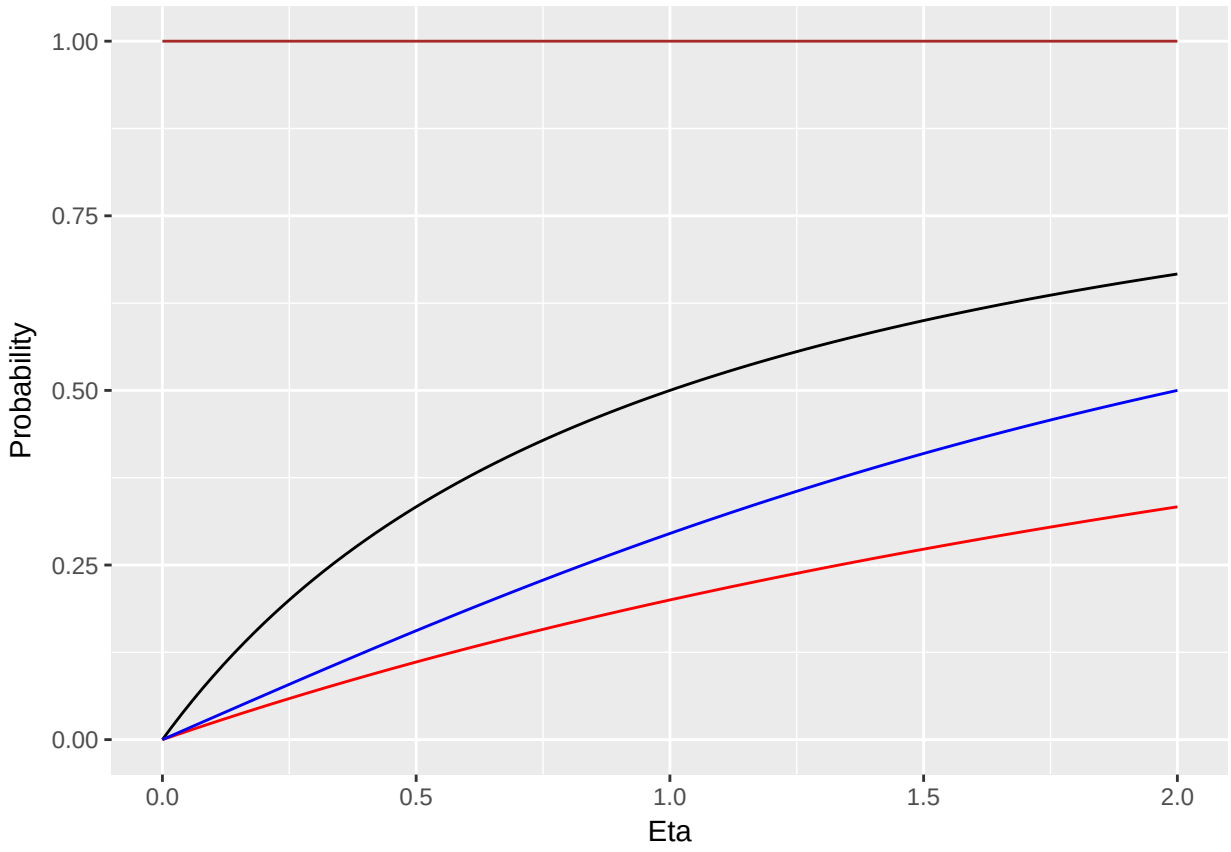
Theodore Dounias

4/24/2018

PART I

A

```
model1 <- function(eta){
  eta/(1+eta)
}

model2 <- function(eta){
  eta/(4+eta)
}

model3 <- function(eta){
  atan(eta/2)/(atan(eta/2) + atan(2/eta))
}

#Placeholder
model4 <- function(eta){
  1
}

ggplot(data.frame(x = c(0, 2)), aes(x)) +
  stat_function(fun = model1) +
  stat_function(fun = model2, col = "red") +
  stat_function(fun = model3, col = "blue") +
  labs(x = "Eta", y = "Probability") +
  stat_function(fun = model4, col = "brown")
```



B

```
likelihood <- function(model){
  a <- rep(0, 11)
  for(i in 1:11){
    res <- results$Face[i]
    n <-  results$Total[i]
    prob <- model(coin_info$eta[i])
    a[i] <- dbinom(res, n, prob)
  }
  sum(log(a))
}

likelihood(model3)
```

```
## [1] -623.4315
```

```
dt <- data.frame(c("Model 1", "Model 2", "Model 3", "Model 4"), c(likelihood(model1), likelihood(model2)

names(dt) <- c("Model", "Log Likelihood")

dt %>%
  kable()
```

Model	Log Likelihood
Model 1	-371.1590
Model 2	-778.7845
Model 3	-623.4315
Model 4	-Inf

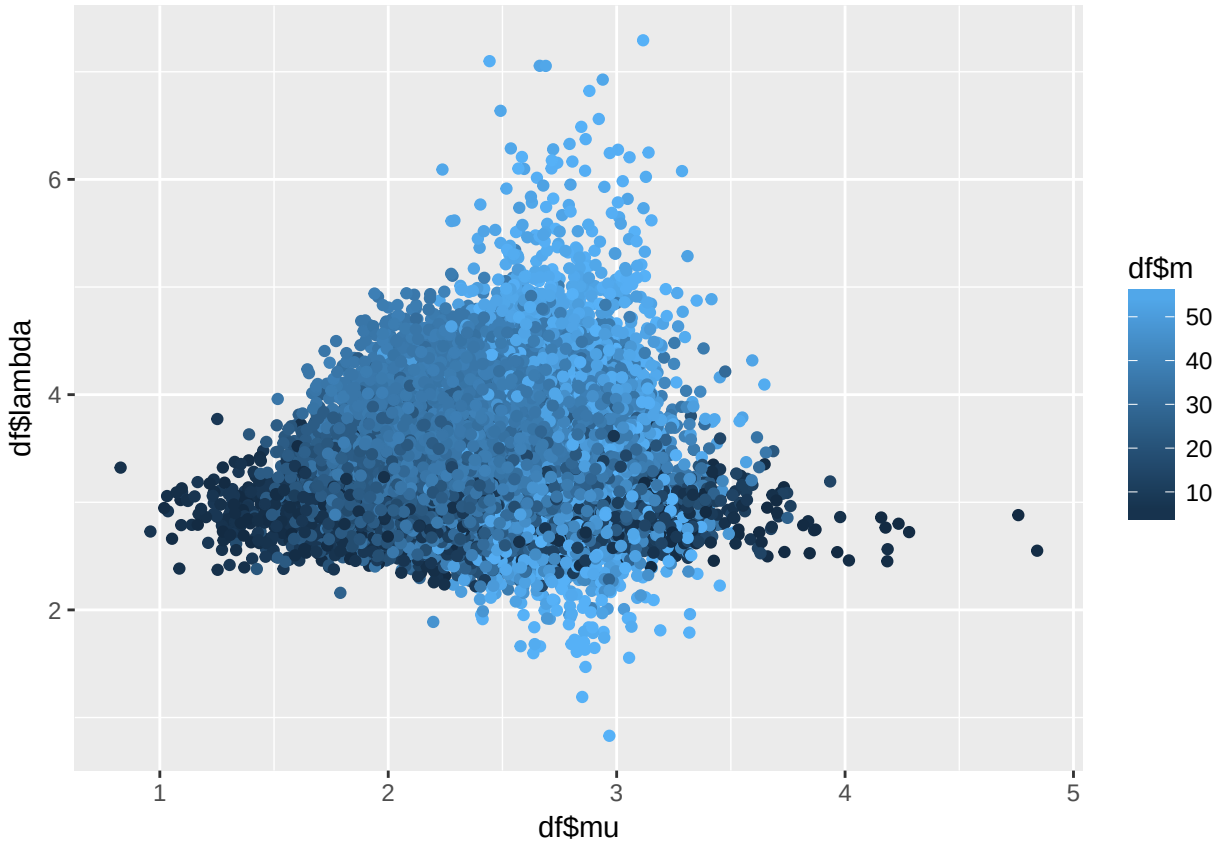
I use the inverse here since R would not print small enough numbers. The results indicate that has a higher likelihood than the other three.

PART II

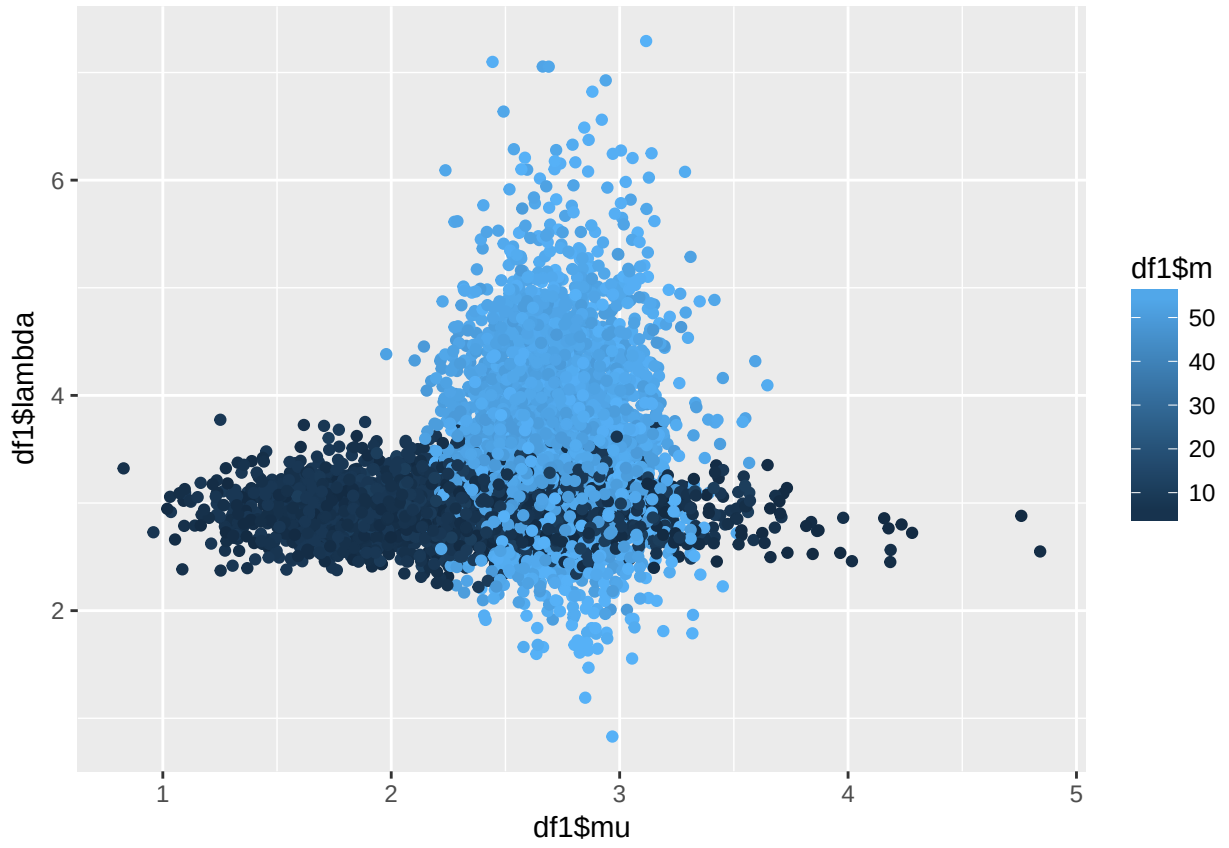
Copying code from the notes:

```
set.seed(497)
n <- 60
m <- 38
mu <- 2
lambda <- 4
y_mu <- rpois(m, lambda = mu)
y_lambda <- rpois(n - m, lambda = lambda)
y <- c(y_mu, y_lambda)
alpha <- 10
beta <- 4
nu <- 8
phi <- 2
it <- 50000
post_samples <- matrix(rep(NA, it * 3), ncol = 3)
colnames(post_samples) <- c("mu", "lambda", "m")
m_j <- 2 # initialize m
for (j in 1:it) {
  # sample mu
  mu_j      <- rgamma(1, alpha + sum(y[1:m_j]), beta + m_j)
  # sample lambda
  lambda_j  <- rgamma(1, nu + sum(y[(m_j+1):n]), phi + (n - m_j))
  # sample m
  m_vec <- rep(NA, n - 1)
  for (k in 1:(n - 1)) {
    m_vec[k] <-  mu_j^(alpha + sum(y[1:k]) - 1) *
      exp(-(beta + k) * mu_j) *
      lambda_j^(nu + sum(y[(k+1):n]) - 1) *
      exp(-(phi + n - k) * lambda_j)
  }
  p <- m_vec/sum(m_vec)
  m_j <- sample(1:(n - 1), size = 1, prob = p)
  # store results
  post_samples[j, "mu"]     <- mu_j
  post_samples[j, "lambda"] <- lambda_j
  post_samples[j, "m"]      <- m_j
}
df <- data.frame(post_samples) %>%
  mutate(index = 1:it)

#Scatterplot
```

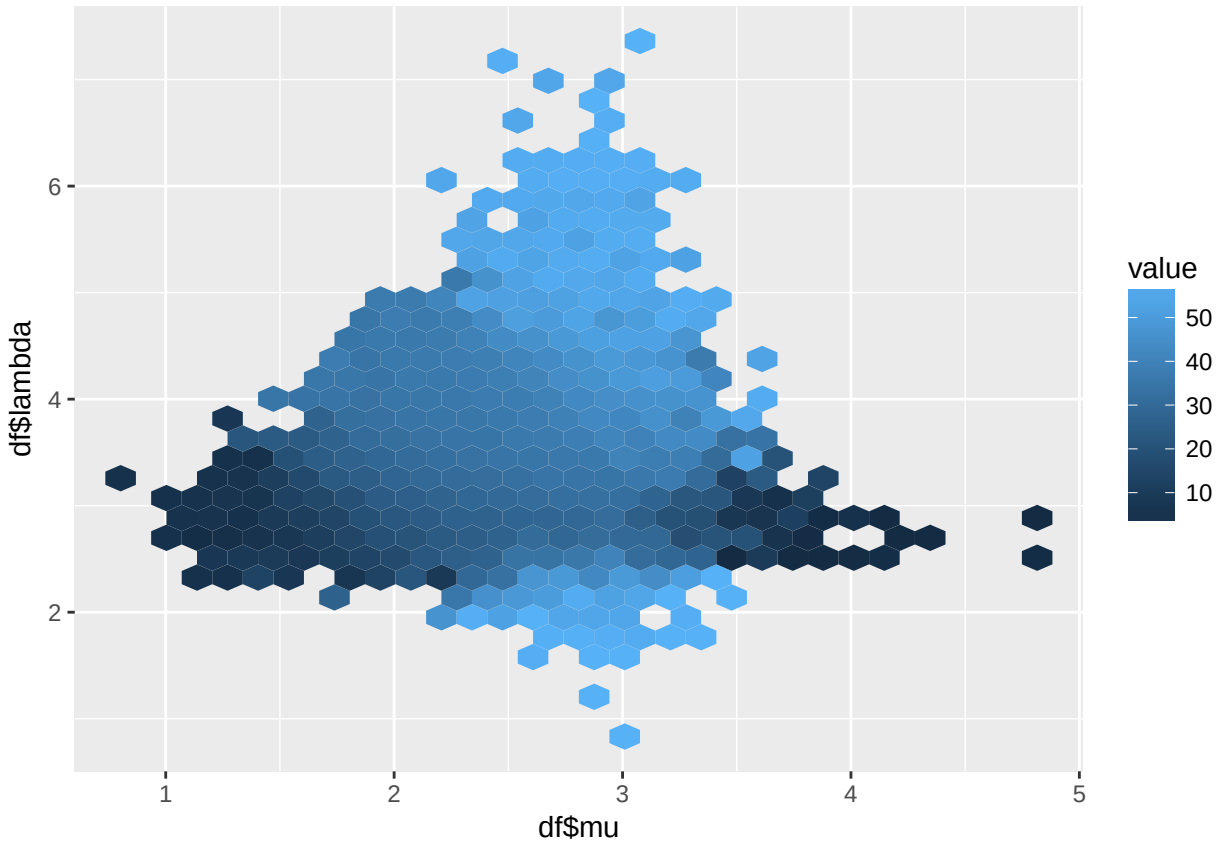
```r
ggplot(df, aes(x = df$mu, y = df$lambda, col = df$m)) +
  geom_point()
```




```r
#Filtered Scatterplot

df1 <- df %>%
  filter(m <= 10 | m >= 50)

ggplot(df1, aes(x = df1$mu, y = df1$lambda, col = df1$m)) +
  geom_point()
```



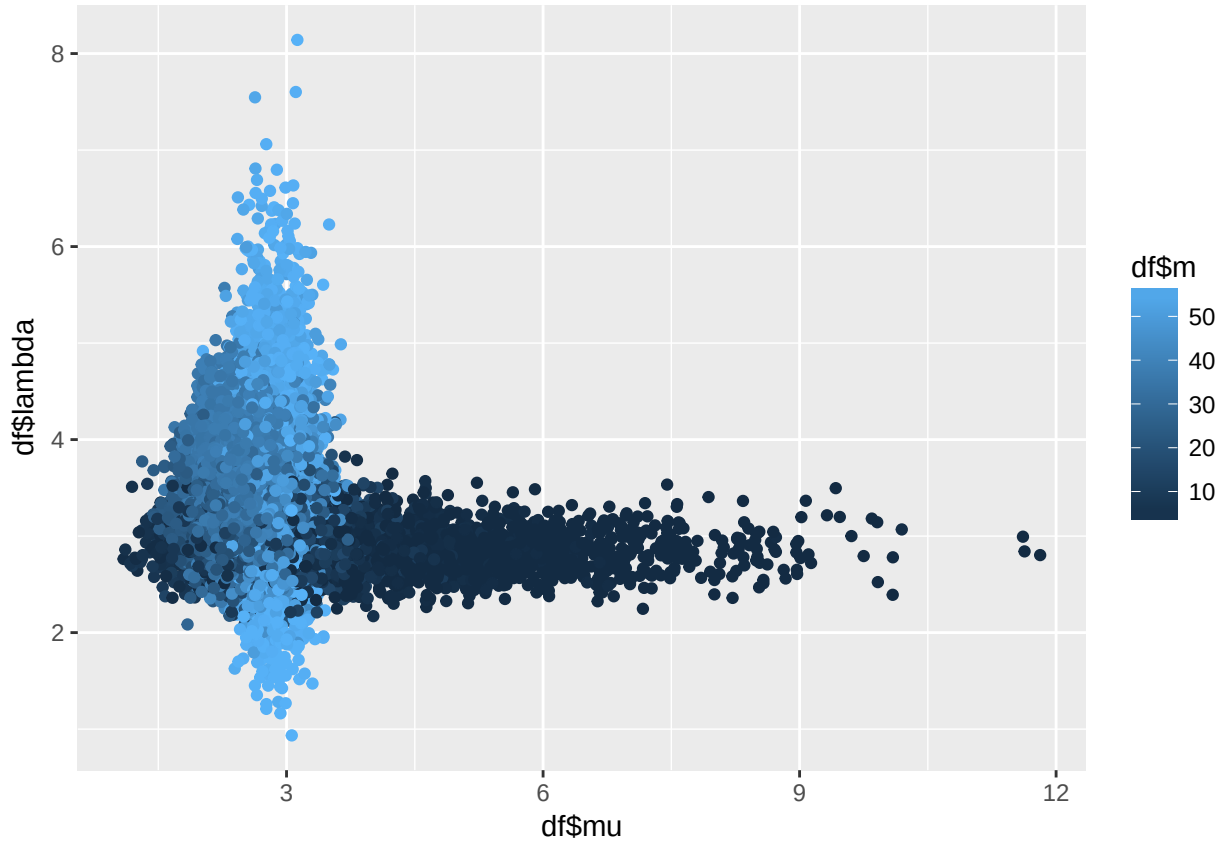

```
ggplot(df, aes(x = df$mu, y = df$lambda)) +
  stat_summary_hex(aes(z = df$m), fun = "mean")
```



It is clear, particularly from the filtered scatterplot, that when m is closer to its extreme values one of the two distributions suffers from data limitations on making any inference on its value; for example, for a low m value, lambda follows a sharp distribution while mu has high variability. For middling values of m, the plots show a large area of potential values for mu, lambda, since both have an approximately similar amount of draws to work with.

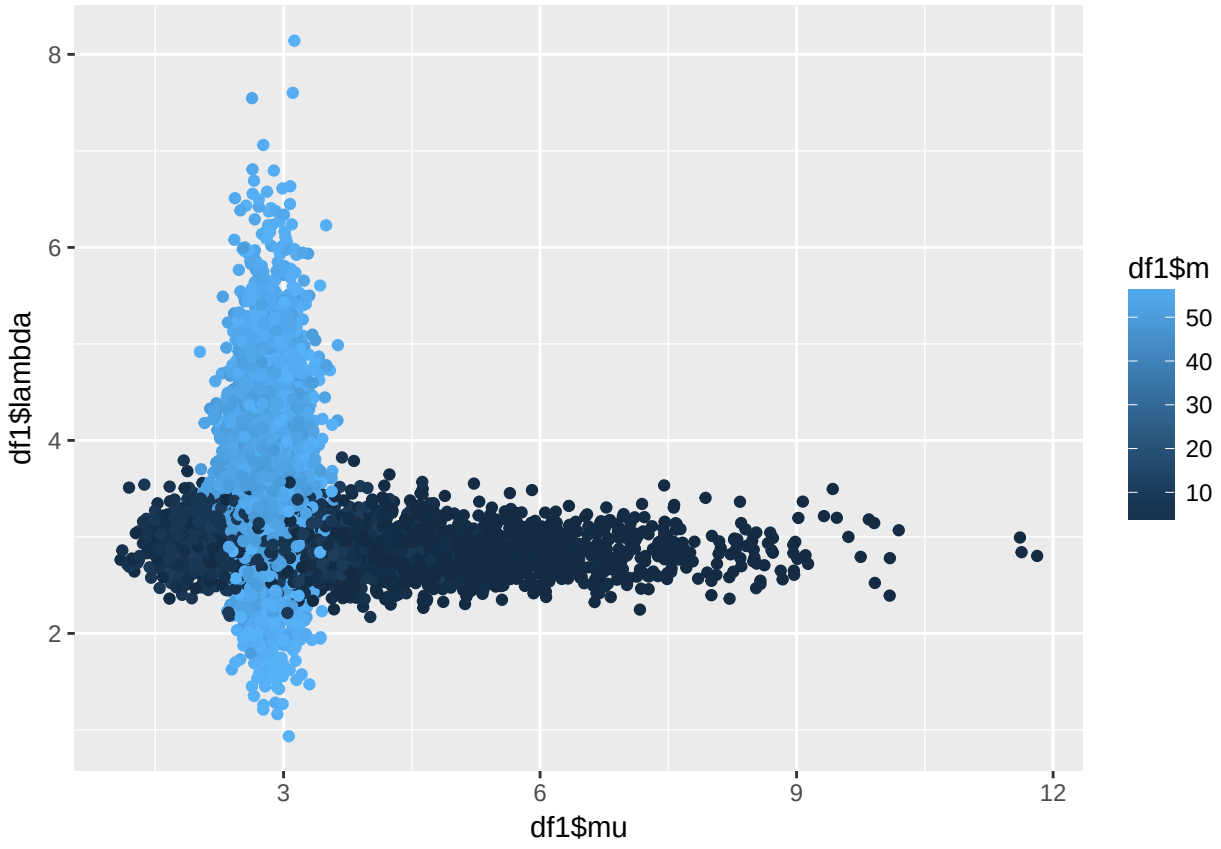
```
#I change the values of alpha and beta on the prior
set.seed(497)
n <- 60
m <- 38
mu <- 2
lambda <- 4
y_mu <- rpois(m, lambda = mu)
y_lambda <- rpois(n - m, lambda = lambda)
y <- c(y_mu, y_lambda)
alpha <- 7
beta <- 1
nu <- 8
phi <- 2
it <- 50000
post_samples <- matrix(rep(NA, it * 3), ncol = 3)
colnames(post_samples) <- c("mu", "lambda", "m")
m_j <- 2 # initialize m
for (j in 1:it) {
  # sample mu
  mu_j      <- rgamma(1, alpha + sum(y[1:m_j]), beta + m_j)
  # sample lambda
```

```r
  lambda_j  <- rgamma(1, nu + sum(y[(m_j+1):n]), phi + (n - m_j))
  # sample m
  m_vec <- rep(NA, n - 1)
  for (k in 1:(n - 1)) {
    m_vec[k] <-  mu_j^(alpha + sum(y[1:k]) - 1) *
      exp(-(beta + k) * mu_j) *
      lambda_j^(nu + sum(y[(k+1):n]) - 1) *
      exp(-(phi + n - k) * lambda_j)
  }
  p <- m_vec/sum(m_vec)
  m_j <- sample(1:(n - 1), size = 1, prob = p)
  # store results
  post_samples[j, "mu"]     <- mu_j
  post_samples[j, "lambda"] <- lambda_j
  post_samples[j, "m"]      <- m_j
}
df <- data.frame(post_samples) %>%
  mutate(index = 1:it)

#Scatterplot
ggplot(df, aes(x = df$mu, y = df$lambda, col = df$m)) +
  geom_point()
```

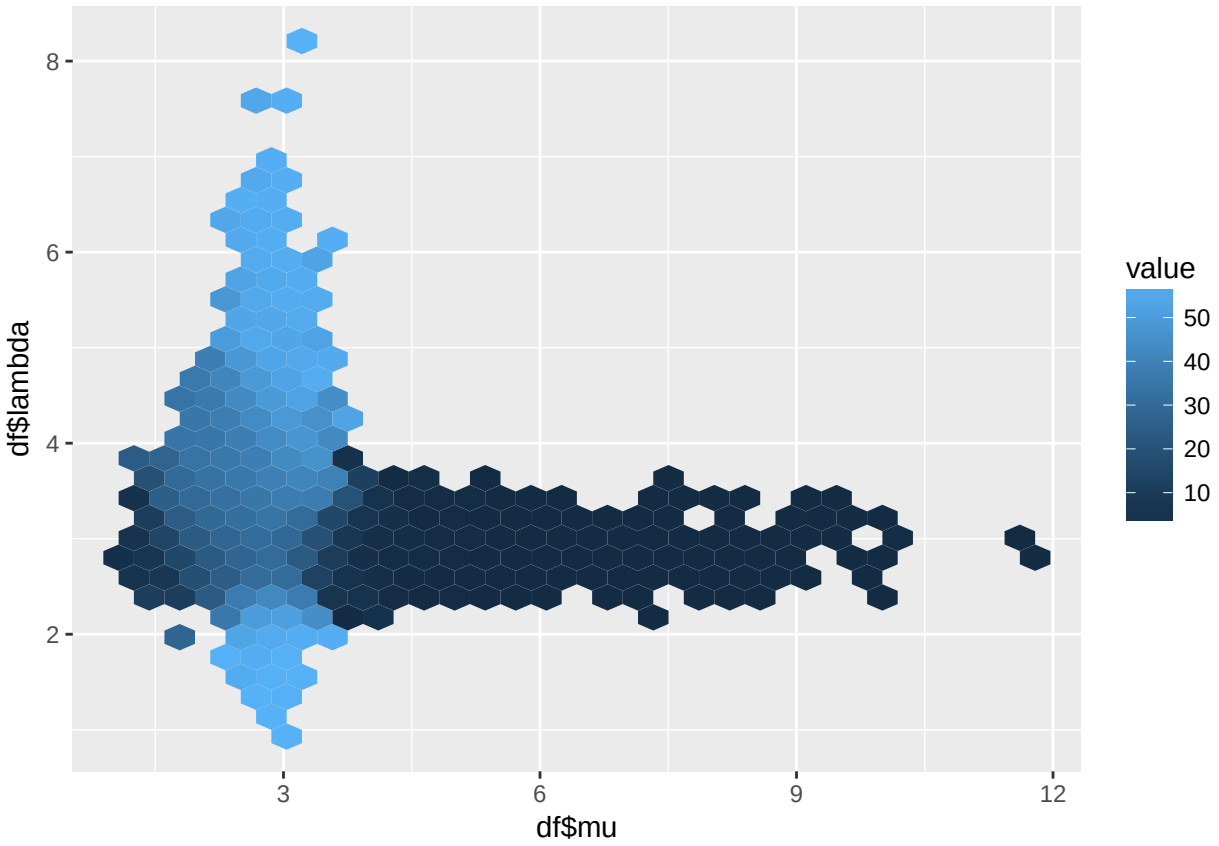



```r
#Filtered Scatterplot

df1 <- df %>%
  filter(m <= 10 | m >= 50)
```

```
ggplot(df1, aes(x = df1$mu, y = df1$lambda, col = df1$m)) +
  geom_point()
```




```
ggplot(df, aes(x = df$mu, y = df$lambda)) +
  stat_summary_hex(aes(z = df$m), fun = "mean")
```

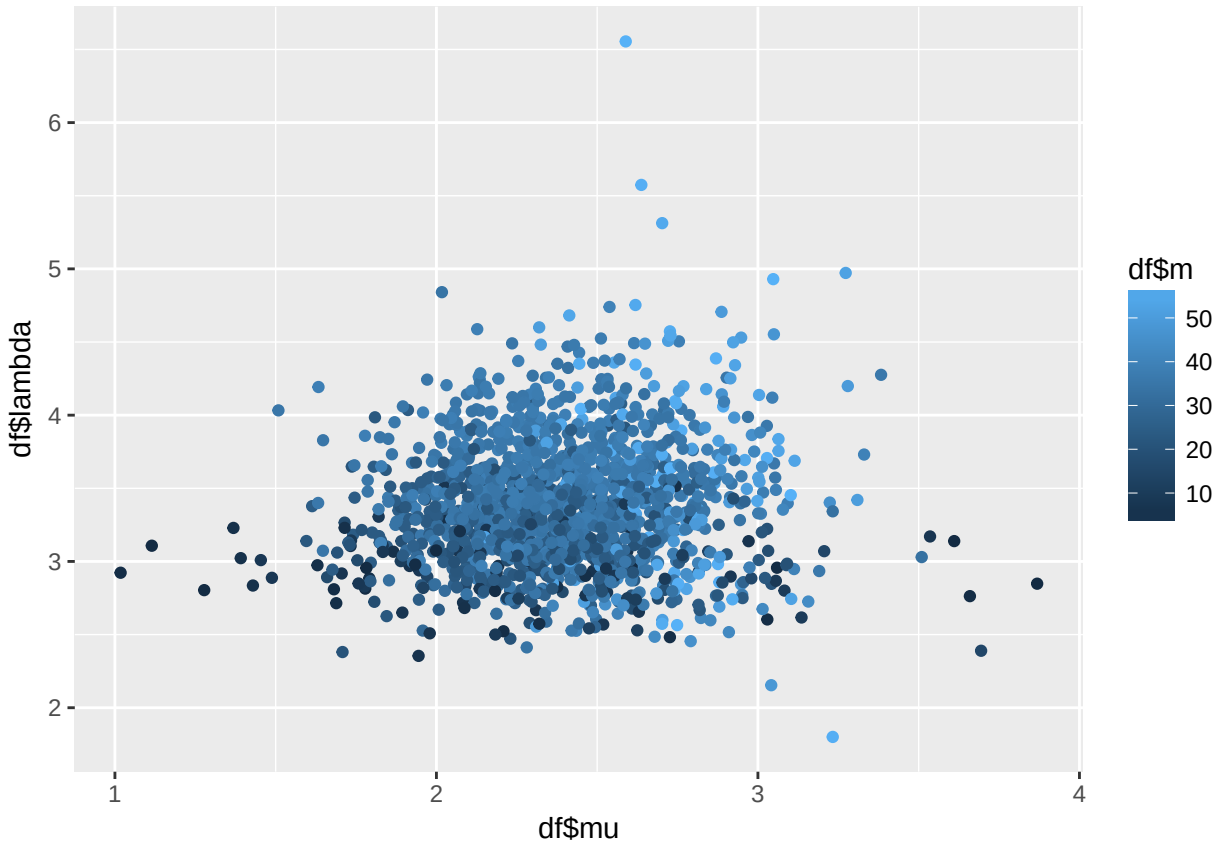


Both distributions for lambda and mu are more sharply distributed and centered a bit off of their true values.

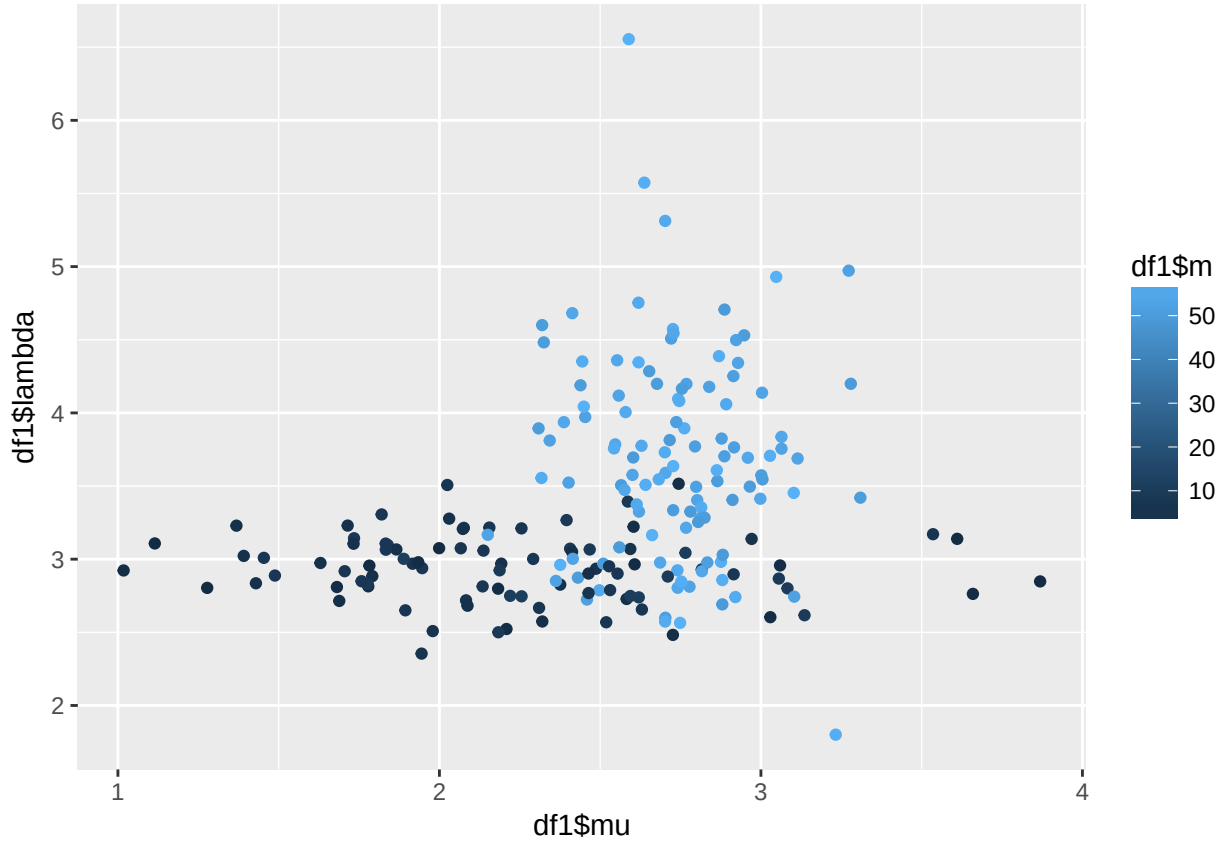
PART III

```
set.seed(497)
n <- 60
m <- 38
mu <- 2
lambda <- 4
y_mu <- rpois(m, lambda = mu)
y_l <- rpois(n - m, lambda = lambda)
y <- c(y_mu, y_l)
alpha <- 10
beta <- 4
ni <- 8
fi <- 2

joint_post <- function(lambda, mu, m, y){
  mu^(alpha + sum(y[1:m]) - 1) *
    exp(-(beta + m) * mu) *
    lambda^(ni + sum(y[(m+1):n]) - 1) *
    exp(-(fi + n - m) * lambda)
}

params_df <- function(it){
```

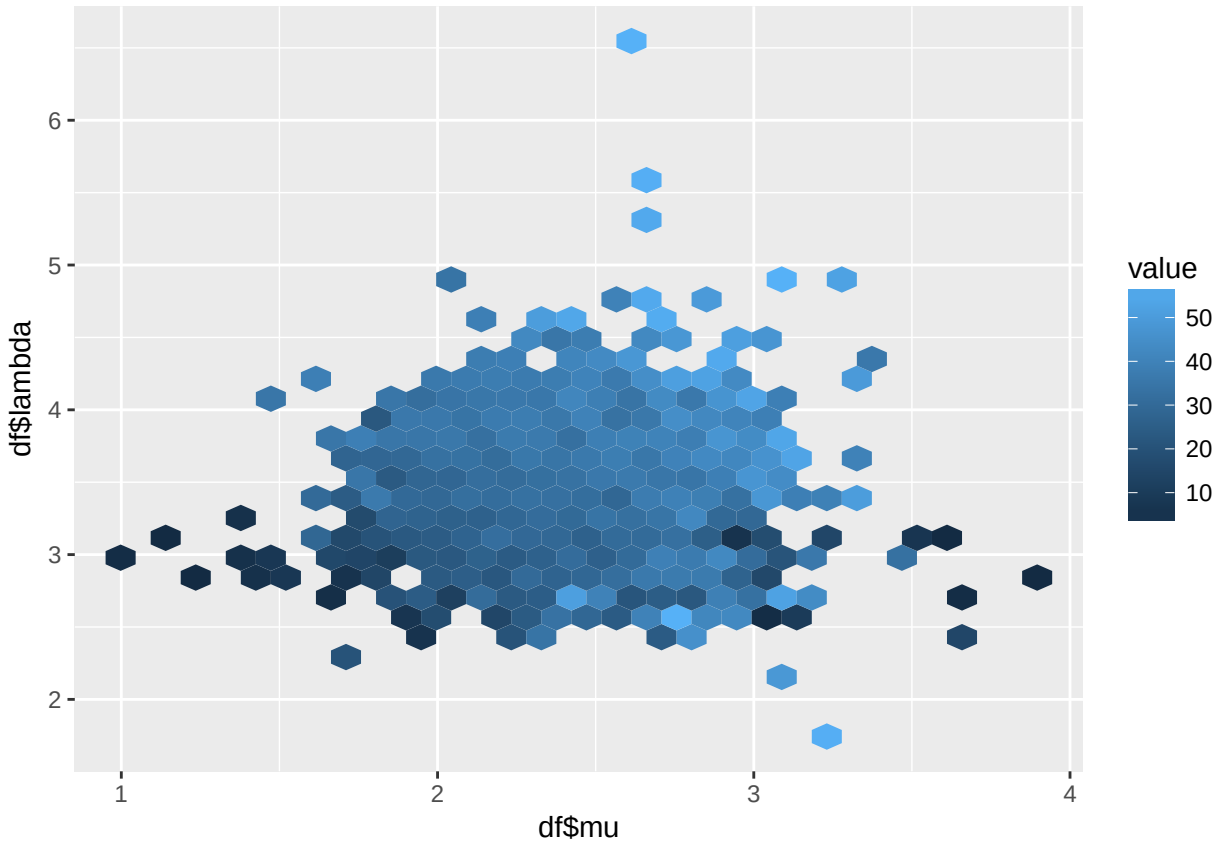
```r
  lambda <- runif(it, 0, 7)
  mu <- runif(it, 0, 7)
  m <- base::sample(1:59, size = it, replace = TRUE)
  df <- data.frame(lambda, mu, m)
  df
}

it <- 1e6

params <- params_df(it)

joint_post_prob <- rep(0, it)
for(i in 1:it){
joint_post_prob[i] <- joint_post(params$lambda[i], params$mu[i], params$m[i], y)
}

keep <- joint_post_prob/max(joint_post_prob) > runif(it)

params$keep <- keep

df <- params %>%
  filter(keep)

#Scatterplot
ggplot(df, aes(x = df$mu, y = df$lambda, col = df$m)) +
  geom_point()
```



```r
#Filtered Scatterplots

df1 <- df %>%
  filter(m <= 10 | m >= 50)

ggplot(df1, aes(x = df1$mu, y = df1$lambda, col = df1$m)) +
  geom_point()
```




```r
ggplot(df, aes(x = df$mu, y = df$lambda)) +
  stat_summary_hex(aes(z = df$m), fun = "mean")
```



The plots produced here look very similar to those from the Gibbs sampler. The benefits from the rejection sampler are that it is less intensive on the calculus side, and does not require a full calculation of all the posteriors. It is, however, more computationally intensive for my processor to handle. Both methods need some for of knowledge on the parameters we wish to calculate. If, for example, I would have chosen a different range for lambda or mu, the results for the rejection sampler would have been innacurrate, while the Gibbs sampler needs definition of priors. The Gibbs might have the advatage here, as what matters most is the support of the prior distributions, not having some sort of semi-accurate guess as to the real values.