MATH 392 Problem Set 10

Sam D. Olson

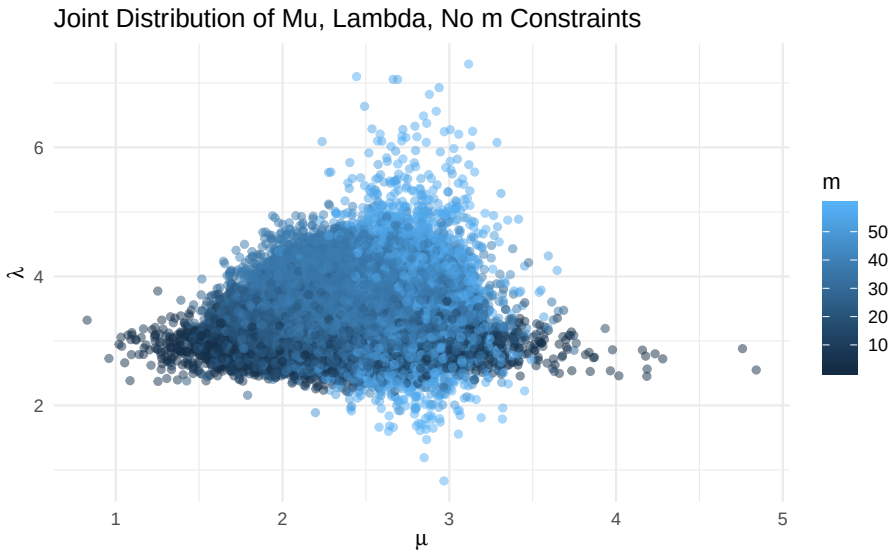
Part I: Changing the changepoint

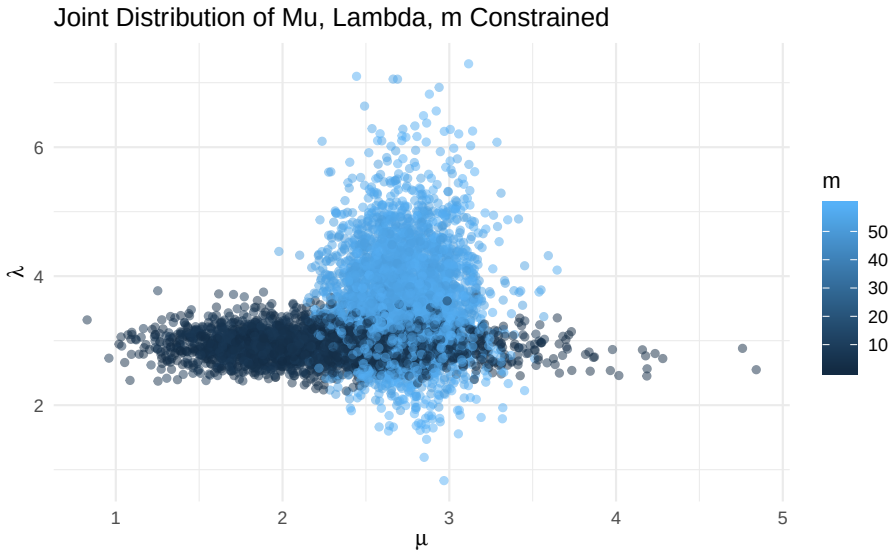
You're encouraged to start these exercise by copying into your problem set code from the notes that implements the Gibbs Sampler with `set.seed(497)`.

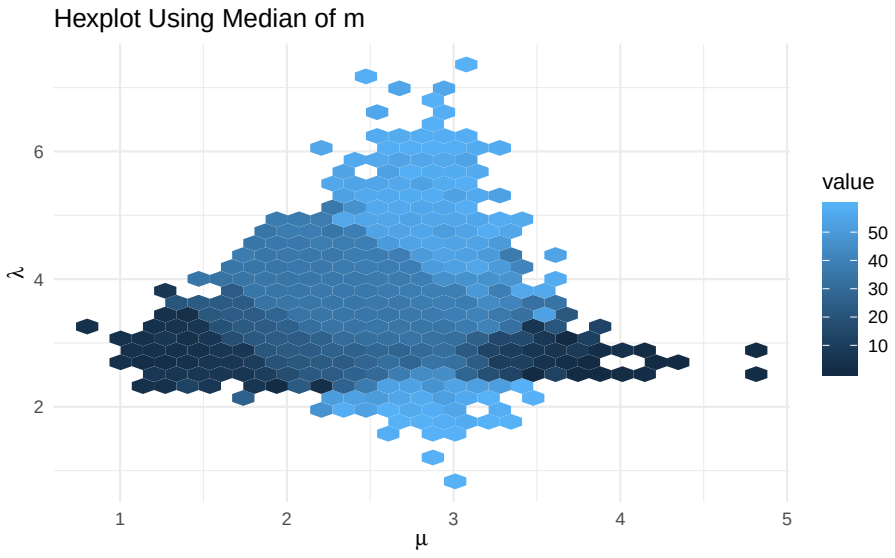
1. Using the `post_samples` matrix (or corresponding dataframe `df`), investigate the shape of the joint posterior distribution by constructing several plots. After contructing these plots, summarize in a few sentences the relationship between the three parameters in the posterior. *Please put all three plots in a single 1 by 3 frame using `library(patchwork)`.*
    - A scatterplot of λ on μ with color mapped to the value of m.
    - The same scatterplot but where you only include samples with values of m that are less than 10 or greater than 50.
    - A hexplot where each hex is filled with either the mean or median value of m.

```
# Set Seed
set.seed(497)
# Initialize Data/Parameters
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
# Post-Samples Matrix
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)
# Visuals
# Part a
p4 <- ggplot(df, aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Mu, Lambda, No m Constraints") +
  theme_minimal()
# p4
# Part b
# $m$ that are less than 10 or greater than 50.
df2 <- subset(df, m > 50 | m < 10,)
p5 <- ggplot(df2, aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Mu, Lambda, m Constrained") +
  theme_minimal()
# Part c
# A [hexplot](http://ggplot2.tidyverse.org/reference/stat_summary_2d.html) where each hex is filled wit
p6 <- ggplot(df, aes(x = mu, y = lambda, z = m)) +
  # geom_hex(fun = ~ quantile(m, 0.5)) +
  stat_summary_hex(fun = "quantile", fun.args = list(probs = 0.5)) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Hexplot Using Median of m") +
  theme_minimal()
p4 / p5 / p6
```


Joint Distribution of Mu, Lambda, No m Constraints
λ
μ
m
50
40
30
20
10
1
2
3
4
5
2
4
6


Joint Distribution of Mu, Lambda, m Constrained
λ
μ
m
50
40
30
20
10
1
2
3
4
5
2
4
6


Hexplot Using Median of m
λ
μ
value
50
40
30
20
10
1
2
3
4
5
2
4
6

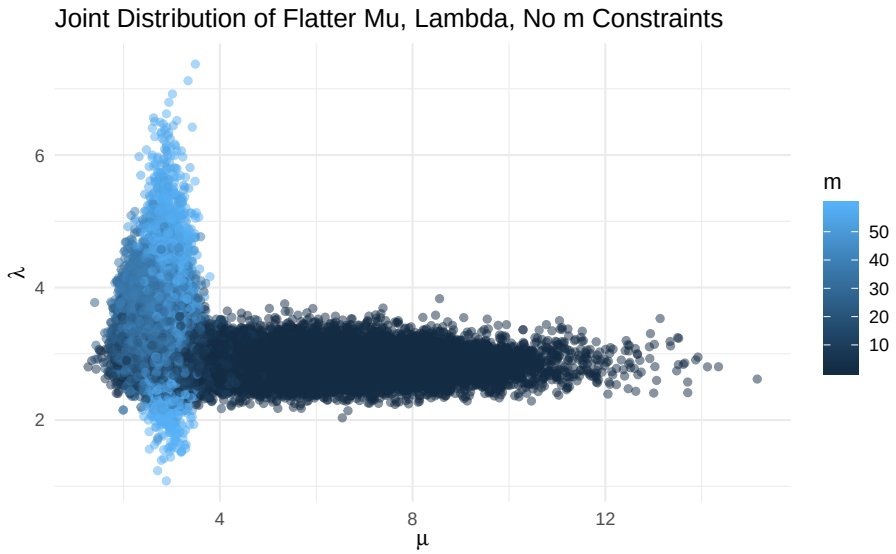
Commentary For low values of m, there is greater spread across μ. By comparison, for higher values of m, there is greater spread across λ.

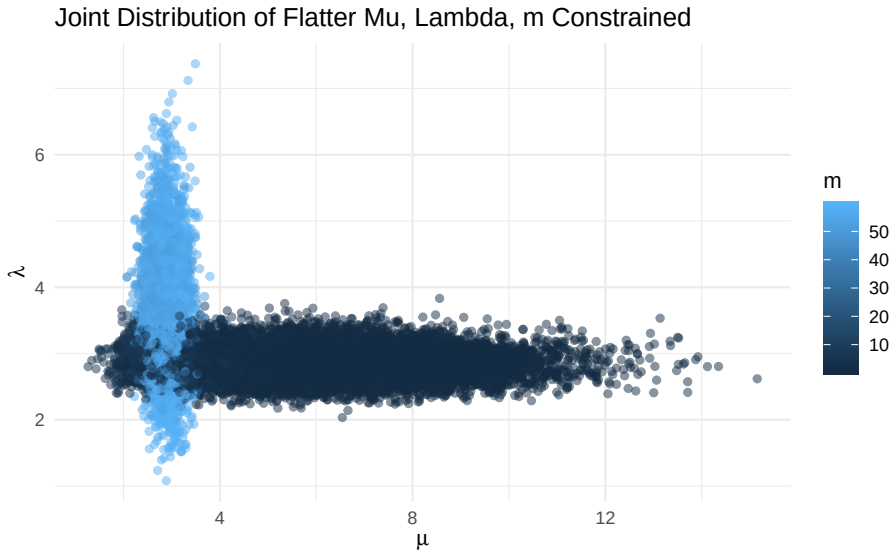
If μ or λ is more extreme, or is a larger value, the other one tends to be much less extreme (a smaller value).

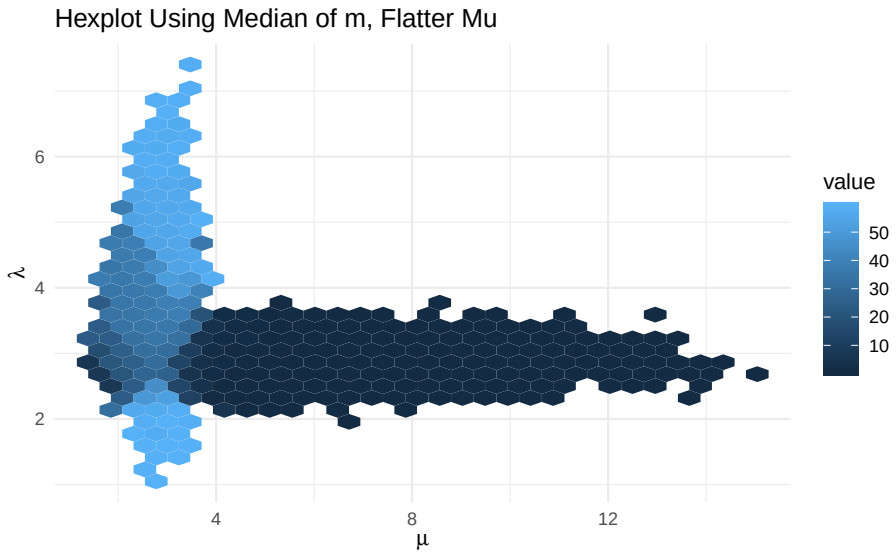
2. Alter one of the prior distributions of the Poisson rate parameters so that they are much more flat. What is the effect on the joint posterior distribution?

```
# Set Seed
set.seed(497)
# Initialize Data/Parameters
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
# Modify the value of beta, increasing the variance of the distribution
# Increasing the variance makes the curve flatter
beta <- 1
nu <- 8
phi <- 2
# Post-Samples Matrix
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)
# Visuals
# Part a
p7 <- ggplot(df, aes(x = mu, y = lambda, color=m)) +
```

```
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Flatter Mu, Lambda, No m Constraints") +
  theme_minimal()
# p4
# Part b
# $m$ that are less than 10 or greater than 50.
df2 <- subset(df, m > 50 | m < 10,)
p8 <- ggplot(df2, aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Flatter Mu, Lambda, m Constrained") +
  theme_minimal()
# Part c
# A [hexplot](http://ggplot2.tidyverse.org/reference/stat_summary_2d.html) where each hex is filled wit
p9 <- ggplot(df, aes(x = mu, y = lambda, z = m)) +
  # geom_hex(fun = ~ quantile(m, 0.5)) +
  stat_summary_hex(fun = "quantile", fun.args = list(probs = 0.5)) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Hexplot Using Median of m, Flatter Mu") +
  theme_minimal()
p7 / p8 / p9
```


Joint Distribution of Flatter Mu, Lambda, No m Constraints
λ
μ
m
50
40
30
20
10
2
4
6
4
8
12


Joint Distribution of Flatter Mu, Lambda, m Constrained
λ
μ
m
50
40
30
20
10
2
4
6
4
8
12


Hexplot Using Median of m, Flatter Mu
λ
μ
value
50
40
30
20
10
2
4
6
4
8
12

Commentary Flattening μ makes the distribution of λ wider.

Part II: The Metropolis algorithm

To validate the results from the Gibbs sampler, get a second estimate of the joint posterior (using the original priors) using the Metropolis algorithm. You're encouraged to start these exercise by copying into your problem set code from the notes that implements the Metropolis Algorithm. You will need to put some thought into how to build your symmetric proposal distribution since it combines continuous and discrete parameters.

1. Generate the same 3 x 1 plot as you did for the Gibbs sampler. Do these tell the same story? (these plots should represent your final conclusions after considering questions 2 and 3 below).
2. What is your acceptance rate? Tinker with the variance of the proposal distribution to bring it within the target range.

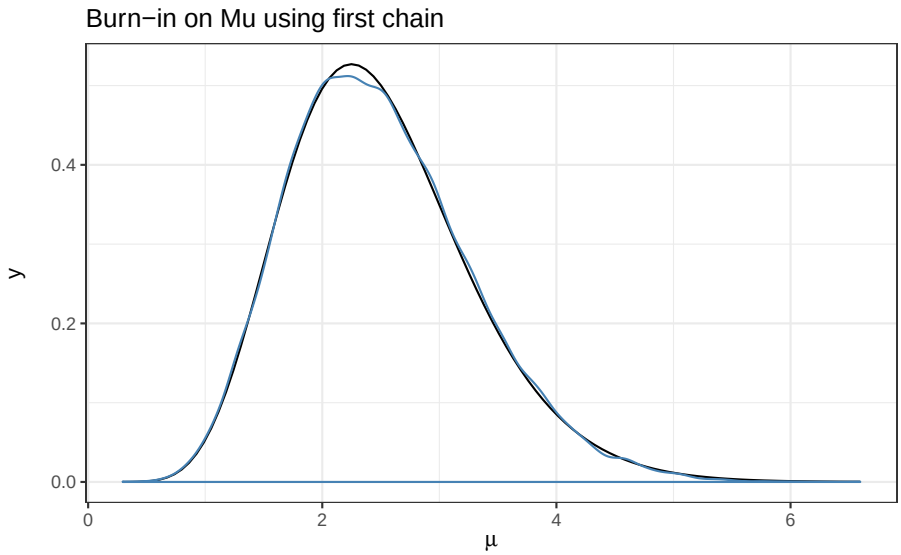
```
# Set Seed
set.seed(497)
# Adapted Metro-Hast
# Set Parameters
n <- 60
alpha <- 10
beta <- 4
nu <- 8
phi <- 2
# Create Chain
chain <- matrix(NA, nrow = it+1, ncol = 3)
# Set Initial Value
theta_0 <- matrix(c(3,5,30), nrow = 1, ncol = 3)
chain[1,] <- theta_0
# Simulate/Run Algorithm
val1 <- .4
val2 <- .2
it <- 50000
for (i in 1:it) {
  # Bound the proposals, otherwise infinite
  proposal <- matrix(c(min(rnorm(1, chain[i,1], val1),10), min(rnorm(1, chain[i,2], val2),10), sample((1
  p_move <- min((dgamma(proposal[,1], alpha, beta) * dgamma(proposal[,2], nu, phi) * dunif(proposal[,3]
                 1)
  # Move up the chain
  test_value <- runif(1)
  chain[i + 1, 1] <- ifelse(test_value < p_move,
                            proposal[,1],
                            chain[i,1])
  chain[i + 1, 2] <- ifelse(test_value < p_move,
                            proposal[,2],
                            chain[i,2])
  chain[i + 1, 3] <- ifelse(test_value < p_move,
                            proposal[,3],
                            chain[i,3])
}
colnames(chain) <- c("mu", "lambda", "m")
```

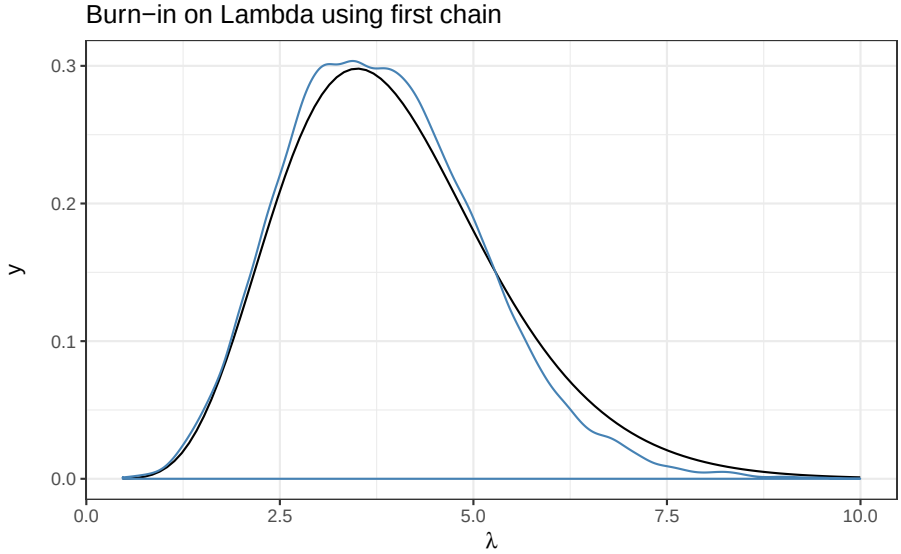
```r
# Using Bray Notes
# Set Seed
set.seed(497)
# Adapted Metro-Hast
# Set Parameters
B0 <- 0
B1 <- 5
sigma <- 10
n <- 60
alpha <- 10
beta <- 4
nu <- 8
phi <- 2
# Create x's, y's
x <- (-(n-1)/2):((n-1)/2)
y <-  B0 + B1 * x + rnorm(n, mean = 0, sd = sigma)
# Metropolis Algorithm
it <- 50000
chain_alt <- matrix(rep(NA, (it + 1) * 3), ncol = 3)
theta_0 <- matrix(c(3,5,30), nrow = 1, ncol = 3)
chain_alt[1,] <- theta_0
for (i in 1:it){
  proposal <- c(max(rnorm(1, mean = chain_alt[i, 1],
                    sd = c(0.5)), 1), max(rnorm(1, mean = chain_alt[i, 2],
                    sd = c(0.2)), 1),  sample((1:n-1), 1, rep(dbinom(chain_alt[i,3], n-1, chain_alt[i,3]
  p_move <- exp(posterior(proposal) - posterior(chain_alt[i, ]))
  if (runif(1) < p_move) {
    chain_alt[i + 1, ] <- proposal
  } else {
    chain_alt[i + 1, ] <- chain_alt[i, ]
  }
}
```

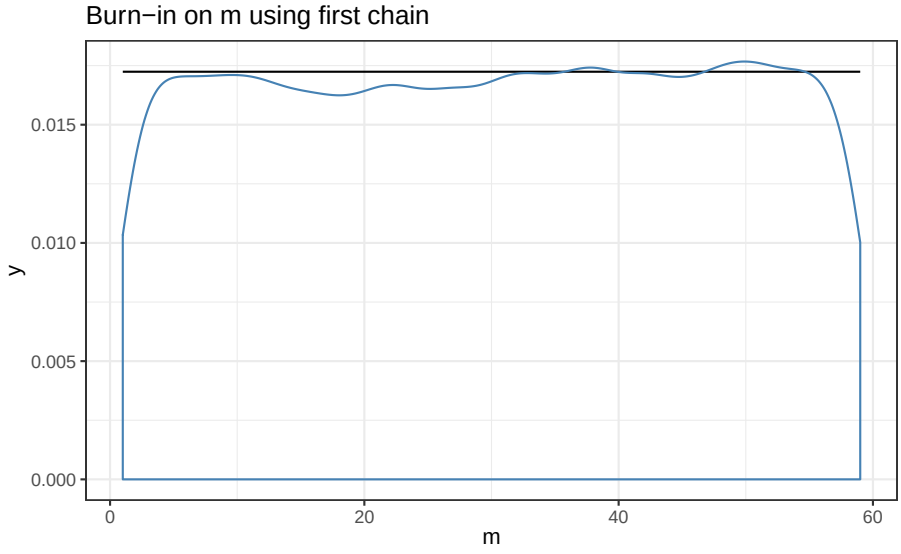
```r
# Another Alternative
# Not Functioning
# Write Functions
posterior <- function(theta){
  mu <- theta[1]
  lambda <- theta[2]
  m <- theta[3]
  m_alt <- n - m
  y_mu <- rpois(n = m, lambda = mu)
  y_lambda <- rpois(n = m_alt, lambda = lambda)
  (((mu^(sum(y_mu)))* exp(-m* mu))/(sum(factorial(y_mu)))) *
  (((lambda^(sum(y_lambda)))* exp(-(m_alt) * lambda))/(sum(factorial(y_lambda)))) *
  (((beta)^(alpha))/(gamma(alpha)))*(mu^(alpha -1)) *exp(-beta*mu) *
  (((phi)^(nu))/(gamma(nu)))*(lambda^(nu -1)) * exp(-phi*lambda) *
  (1/(n-1))
}
# Metropolis Algorithm
n <- 60
alpha <- 10
beta <- 4
```

```r
nu <- 8
phi <- 2
# Simulation
it <- 50000
chain3 <- matrix(rep(NA, (it + 1) * 3), ncol = 3)
theta_0 <-c(2,4, 35)
chain3[1,] <- theta_0
colnames(chain3) <- c("mu", "lambda", "m")
for (i in 1:it){
  m_proposal <-  max(sample((1:n-1), 1, rep(dbinom(chain3[i,3], n-1, chain3[i,3]/(n-1)), replace=TRUE))
   param_proposal <- rnorm(2, chain3[i,1:2], sd=sqrt(chain3[i,1:2]*0.01))
   param_proposal[param_proposal<=0] <- 1e-10
   proposal <- c(param_proposal, m_proposal)
   p_move <- min(posterior(proposal)/posterior(chain3[i, ]),1)
  ifelse(runif(1) < p_move,
    chain3[i + 1, ] <- proposal,
    chain3[i + 1, ] <- chain3[i, ])
}
chain3_df <- as.data.frame(chain3)
```

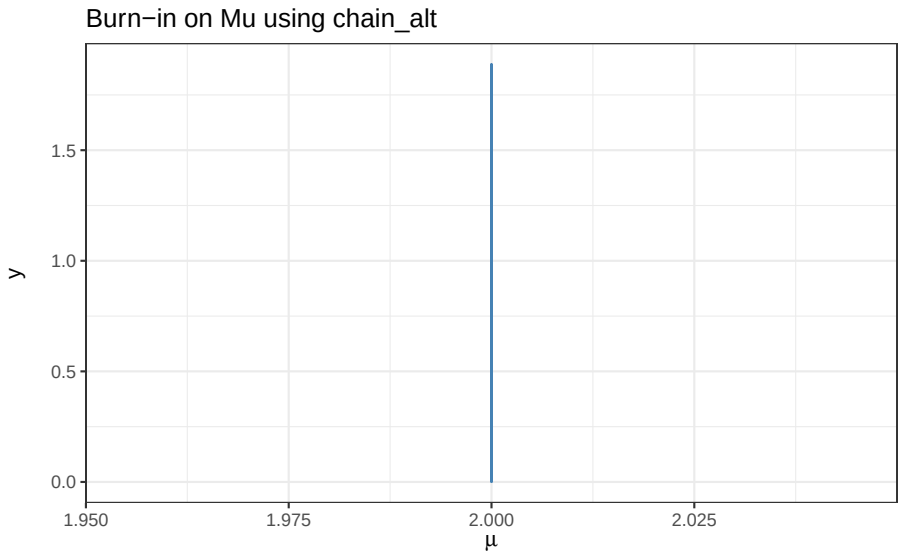
```r
# Burn-in & Other Considerations
burn_in <- 5000
viz1 <- data.frame(chain, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = mu)) +
  stat_function(fun = dgamma, args = list(10, 4)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on Mu using first chain") +
  labs(x = expression(mu))

viz2 <- data.frame(chain, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = lambda)) +
  stat_function(fun = dgamma, args = list(8, 2)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on Lambda using first chain") +
  labs(x = expression(lambda))

viz3 <- data.frame(chain, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = m)) +
  stat_function(fun = dunif, args = list(1, 59)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on m using first chain") +
  labs(x = expression(m))

viz1 / viz2 / viz3
```

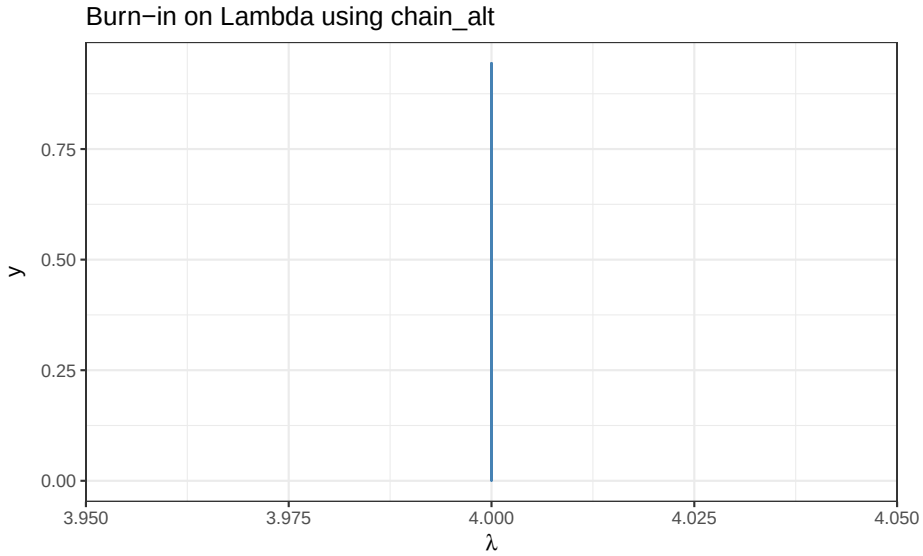

Burn−in on Mu using first chain
0.4
0.2
0.0
y
0
2
4
6
μ

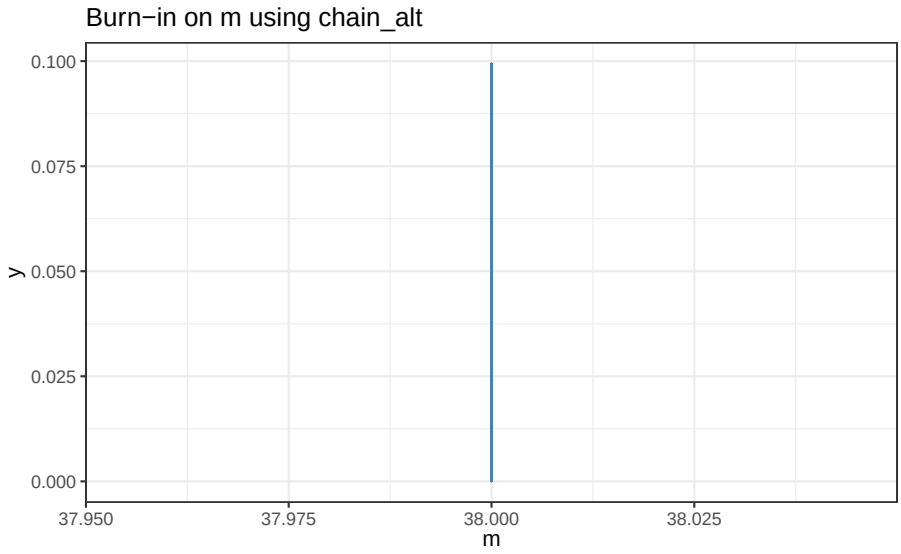

Burn−in on Lambda using first chain
0.3
0.2
0.1
0.0
y
0.0
2.5
5.0
7.5
10.0
λ


Burn−in on m using first chain
0.015
0.010
0.005
0.000
y
0
20
40
60
m

```r
# Burn-in & Other Considerations
burn_in <- 5000
viz1 <- data.frame(chain_alt, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = mu)) +
  stat_function(fun = dgamma, args = list(10, 4)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on Mu using chain_alt") +
  labs(x = expression(mu))

viz2 <- data.frame(chain_alt, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = lambda)) +
  stat_function(fun = dgamma, args = list(8, 2)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on Lambda using chain_alt") +
  labs(x = expression(lambda))

viz3 <- data.frame(chain_alt, index = 1:(it + 1)) %>%
  filter(index > burn_in) %>%
  ggplot(aes(x = m)) +
  stat_function(fun = dunif, args = list(1, 59)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Burn-in on m using chain_alt") +
  labs(x = expression(m))

viz1 / viz2 / viz3
```


Burn−in on Mu using chain_alt
1.5
1.0
0.5
0.0
y
1.950
1.975
2.000
2.025
μ


Burn−in on Lambda using chain_alt
0.75
0.50
0.25
0.00
y
3.950
3.975
4.000
4.025
4.050
λ


Burn−in on m using chain_alt
0.100
0.075
0.050
0.025
0.000
y
37.950
37.975
38.000
38.025
m

```r
# Acceptance Rate
# Part 2
# Initial Chain
(acceptance <- 1 - mean(duplicated(chain[-(1:burn_in)])))
```

```
## [1] 0.5359199
```

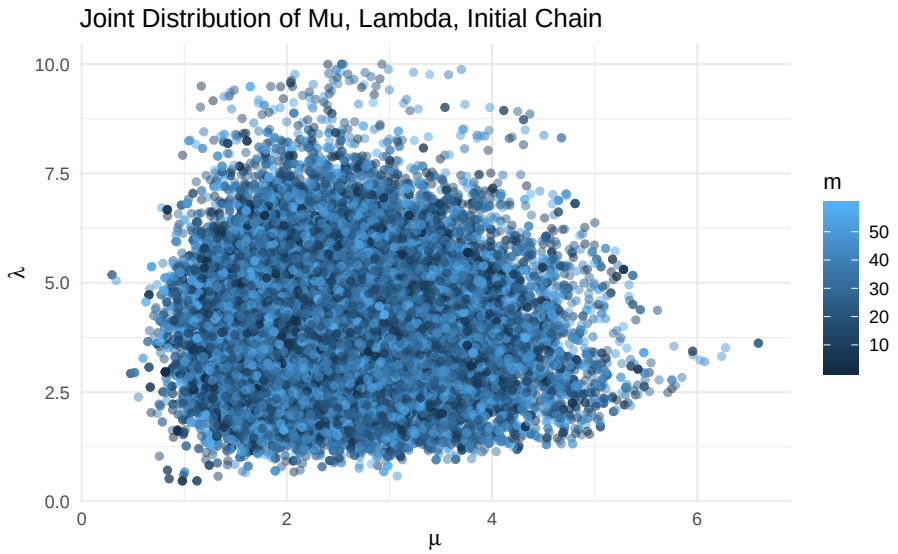
```r
# Alternative Chain
(acceptance_alt <- 1 - mean(duplicated(chain_alt[-(1:burn_in)])))
```

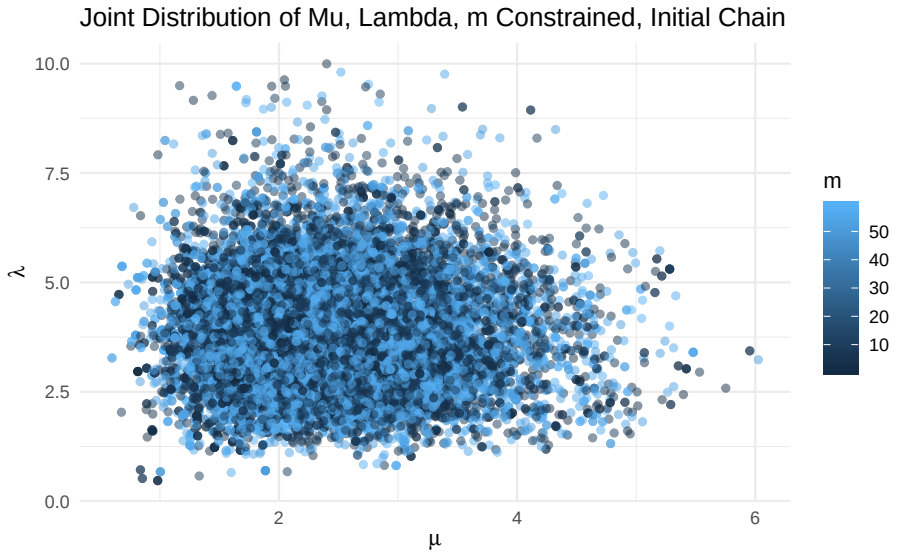
```
## [1] 0.01467556
```

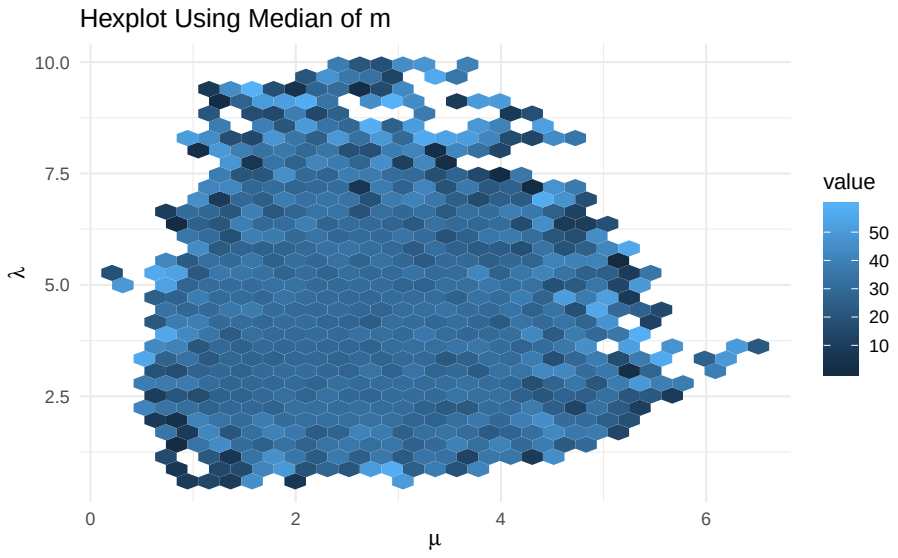
Commentary

Under the initial formulation, our acceptance rate is much higher than using the Metropolis algorithm in conjunction with the posterior/prior functions. Encouragingly, the initial formulation's acceptance rate is ~35%, which is around where we'd hope it to be (~30%). Some further adjustments could be made to change this via adjusting the initial chain values/standard deviation of the proposal distribution.

```r
# Plot Comparison
# Part 1
p10 <- ggplot(data.frame(chain), aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Mu, Lambda, Initial Chain") +
  theme_minimal()

chain2 <- chain[which(chain[,3] > 50
| chain[,3] < 10), ]

p11 <- ggplot(data.frame(chain2), aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Joint Distribution of Mu, Lambda, m Constrained, Initial Chain") +
  theme_minimal()

p12 <- ggplot(data.frame(chain), aes(x = mu, y = lambda, z = m)) +
  # geom_hex(fun = ~ quantile(m, 0.5)) +
  stat_summary_hex(fun = "quantile", fun.args = list(probs = 0.5)) +
  labs(y = expression(lambda), x = expression(mu)) +
  ggtitle("Hexplot Using Median of m") +
  theme_minimal()

p10 / p11 / p12
```


Joint Distribution of Mu, Lambda, Initial Chain
10.0
7.5
5.0
2.5
0.0
λ
0
2
4
6
μ
m
50
40
30
20
10


Joint Distribution of Mu, Lambda, m Constrained, Initial Chain
10.0
7.5
5.0
2.5
0.0
λ
2
4
6
μ
m
50
40
30
20
10

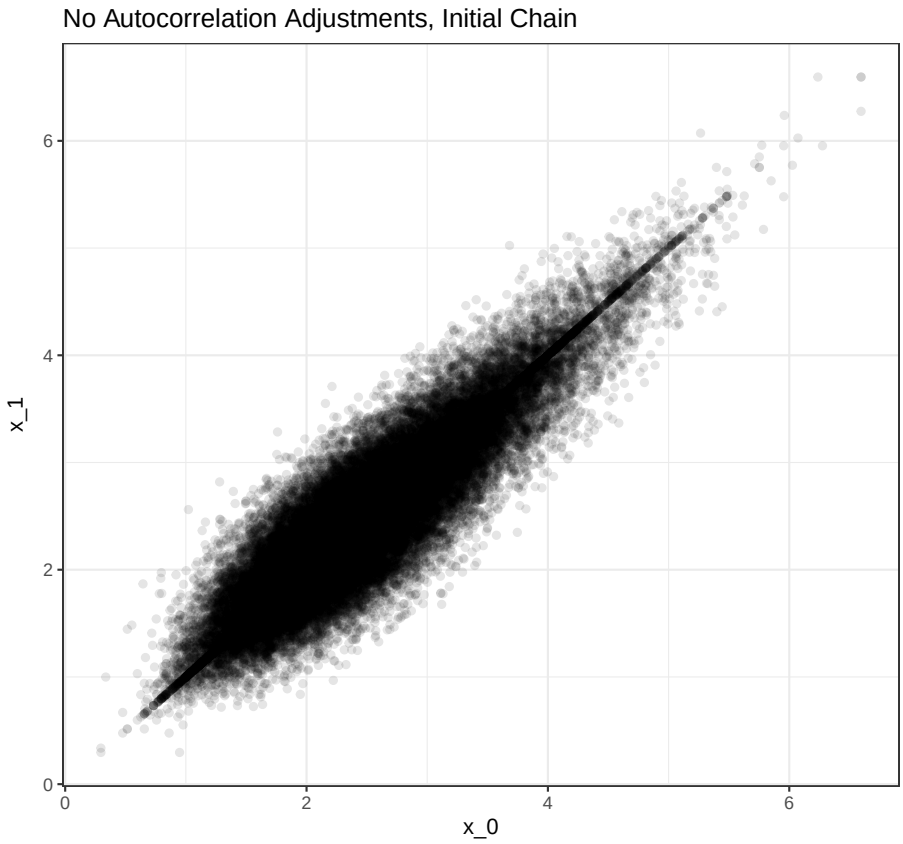

Hexplot Using Median of m
10.0
7.5
5.0
2.5
λ
0
2
4
6
μ
value
50
40
30
20
10

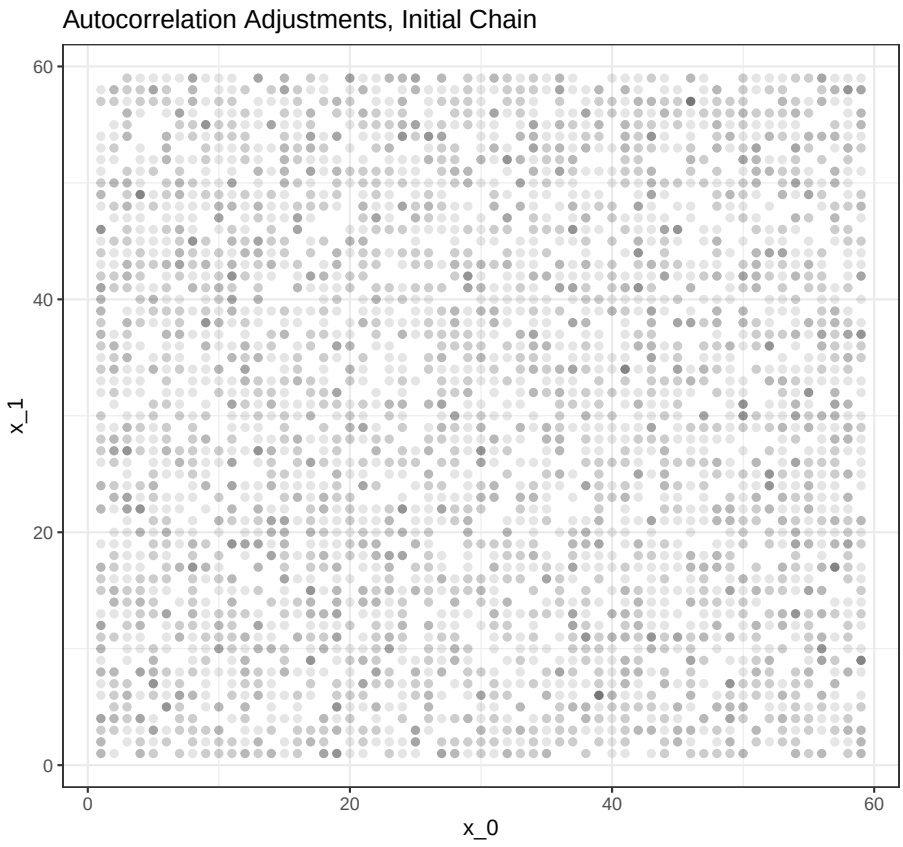
Commentary

Compared to the Gibbs Sampling method, the Metropolis algorithm doesn't appear to have values of μ, λ clustered around particular values of m. This is encouraging, as it validates our decimation to remove autocorrelation. However, it is important to note that without corrections to autocorrelation, we would anticipate the above graphs to be similar to their Gibbs counterparts, as the algorithm generates values that are autocorrelated (subject to the initial conditions but primarily based on the previous values in the chain).

3. To understand the Markov dependence in this chain (or in fact any form of dependence in index), a helpful tool is an autocorrelation plot. Use the `autoplot()` function in `library(ggfortify)` to get a sense of the correlation in your chain (see example of a correlogram). How heavily do you need to thin the chain to diminish most of this dependence? Go ahead and do so to produce a final set of plots for question 1 above.

```
# Autocorrelation
# Part 3
df1 <- data.frame(x_0 = chain[1:(it-1)],
                  x_1 = chain[2:it])

chain_thinned <- data.frame(chain) %>%
  slice(seq(from = 1, to = it, by = 10)) %>%
  pull()

df2 <- data.frame(x_0 = chain_thinned[1:(length(chain_thinned)-1)],
                  x_1 = chain_thinned[2:length(chain_thinned)])

p1 <- ggplot(df1, aes(x = x_0, y = x_1)) +
  geom_point(alpha = .1) +
  ggtitle("No Autocorrelation Adjustments, Initial Chain") +
  theme_bw()

p2 <- ggplot(df2, aes(x = x_0, y = x_1)) +
  geom_point(alpha = .1) +
  ggtitle("Autocorrelation Adjustments, Initial Chain") +
  theme_bw()

p1 / p2
```


No Autocorrelation Adjustments, Initial Chain
x_1
x_0
0
2
4
6


Autocorrelation Adjustments, Initial Chain
x_1
x_0
0
20
40
60

Part III: Rejection Sampling

In this part, you will apply to the same problem a third approach: rejection sampling. You may want to refer to past slides for more details, but in broad strokes rejection sampling for the changepoint problem will involve,

- Writing a function called `joint_post()` that takes as input values of the three parameters (and the data) and returns a value of the joint posterior density.

```
## [1] 0.3678794
```

```
## [1] 0.3678794
```

```
## [1] 0.01724138
```

```
## [1] 0.002333367
```

- Formulating a 3D proposal distribution that you can take random samples from that covers the support of the parameters (or the support with the highest density).

```
# Using Bray Notes
# Some Functions Considered
# Set Seed
set.seed(497)
# Adapted Metro-Hast
# Set Parameters
n <- 60
alpha <- 10
beta <- 4
nu <- 8
phi <- 2
# Create x's, y's
x <- (-(n-1)/2):((n-1)/2)
y <-  B0 + B1 * x + rnorm(n, mean = 0, sd = sigma)
# Write Functions
likelihood <- function(theta) {
    B0 <- theta[1]
    B1 <- theta[2]
    sigma <- theta[3]
    y_fit <- B0 + B1 * x
    logLik_vec <- dnorm(y,
                        mean = y_fit,
                        sd = sigma,
                        log = T)
    sum(logLik_vec)
}

prior <- function(theta) {
    B0 <- theta[1]
    B1 <- theta[2]
    sigma <- theta[3]
    B0_prior <- dnorm(B0, sd = 5, log = T)
```

```
    B1_prior <- dunif(B1, min = 0, max = 10, log = T)
    sigma_prior <- dunif(sigma, min = 0, max = 30, log = T)
    B0_prior + B1_prior + sigma_prior
}

posterior <- function(theta) {
   likelihood(theta) + prior(theta)
}

joint_density <- function(mu, lambda, m) {
  posterior(mu) + posterior(lambda) + posterior(m)
}
```

```
# generate random numbers
random_values <- matrix(NA, nrow= 500000, ncol=3)
for(i in 1:500000){
  random_values[i,] <- c(round(runif(1, min=0, max=10),1) , round(runif(1, min=0, max=10),1), sample(1:5

}
alpha <- 10
beta <- 4
nu <- 8
phi <- 2
joint_post <- function(vec){
  mu <- vec[1]
  lambda <- vec[2]
  m <- vec[3]
  y_mu <- rpois(m, lambda = mu)
  y_lambda <- rpois(n - m, lambda = lambda)
  exp(-mu) * prod(mu^(y_mu)/factorial(y_mu)) *
    exp(-lambda) * prod(lambda^(y_lambda)/factorial(y_lambda)) *
    (beta^(alpha))/gamma(alpha) * mu^(alpha - 1) * exp(-beta*mu) *
    (phi^(nu))/gamma(nu) * lambda^(nu - 1) * exp(-phi * lambda) *
    (1/ (n-1))
}

joint_post_alt <- function(vec){
  # Initial Parameters
  n <- 60
  alpha <- 10
  beta <- 4
  nu <- 8
  phi <- 2
  # Generate Data
  y_mu <- dpois(vec[3], lambda = vec[1])
  y_lambda <- dpois(n - vec[3], lambda = vec[2])
  # sample mu
  mu_j      <- dgamma(1, alpha + sum(y_mu), beta + vec[3])
  # sample lambda
  lambda_j  <- dgamma(1, nu + sum(y_lambda), phi + (n - vec[3]))
  m_j <- dunif(1, 1, n-1)
  k <- vec[3]
  # Calculate Probability
```

```
  posterior(mu_j) + posterior(lambda_j) * m_j * y_mu * y_lambda
}

joint_values <- rep(NA, 500000)
for(i in 1:500000){
  joint_values[i] <- joint_post_alt(random_values[i,])
}
max(joint_values)
```

```
## [1] NA
```

```
q_func <- function(vec){
  dunif(vec[1], 0, 10) * dunif(vec[2], 0, 10) * dunif(vec[3], 1, 59)
}

U <- runif(1, 0, 1) # *runif(1, 0, 1)*runif(1, 0, 1)
# Need to multiply by a constant
# M <-

q_func(c(1,1,1)) > max(joint_values)
```

```
## [1] NA
```

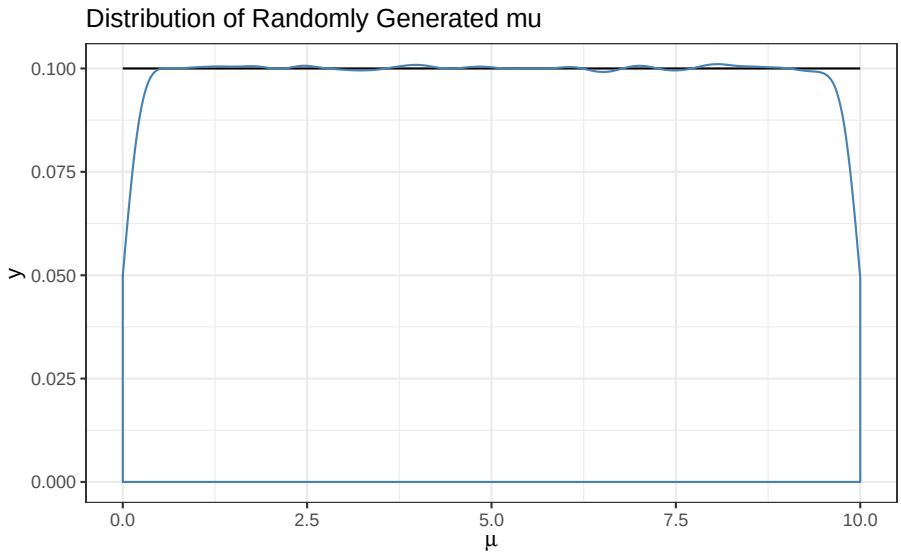
```
vec_accepted <- random_values[U < (joint_post(random_values) / (q_func(random_values)))]
length(vec_accepted)
```

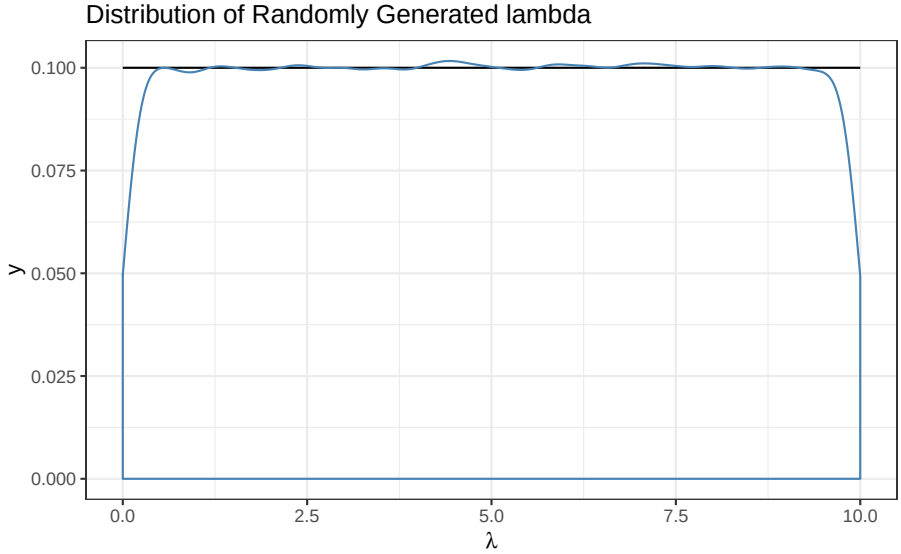
```
## [1] 1500000
```

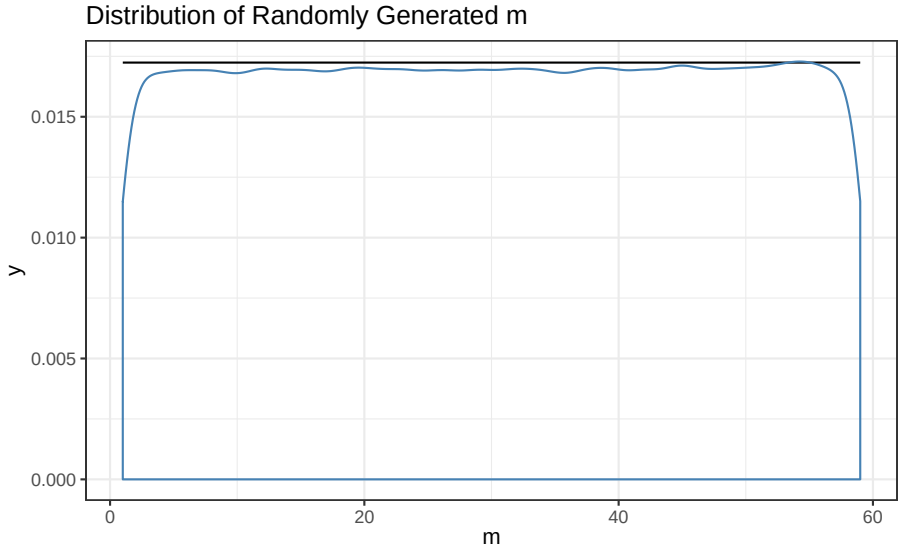
- Taking draws from the proposal distribution for your three parameters and retaining it with probability equal to the ratio of the proposal density at the point to the value of `joint_post()` at that point. Note that you will likely need to normalize either the proposal density or the joint density so that the proposal density is always greater than or equal to the joint density, ensuring acceptance probabilities less than or equal to 1.
- Repeat the previous step many many times until you have accepted a suitable number of draws from the joint posterior.

```
colnames(random_values) <- c("mu", "lambda", "m")
viz1 <- data.frame(random_values, index = 1:500000) %>%
  ggplot(aes(x = mu)) +
  stat_function(fun = dunif, args = list(0, 10)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Distribution of Randomly Generated mu") +
  labs(x = expression(mu))

viz2 <- data.frame(random_values, index = 1:500000) %>%
  ggplot(aes(x = lambda)) +
  stat_function(fun = dunif, args = list(0, 10)) +
  geom_density(col = "steelblue") +
  theme_bw() +
```

```
  ggtitle("Distribution of Randomly Generated lambda") +
  labs(x = expression(lambda))

viz3 <- data.frame(random_values, index = 1:500000) %>%
  ggplot(aes(x = m)) +
  stat_function(fun = dunif, args = list(1, 59)) +
  geom_density(col = "steelblue") +
  theme_bw() +
  ggtitle("Distribution of Randomly Generated m") +
  labs(x = expression(m))

viz1 / viz2 / viz3
```


Distribution of Randomly Generated mu
0.100
0.075
0.050
0.025
0.000
y
0.0
2.5
5.0
7.5
10.0
μ


Distribution of Randomly Generated lambda
0.100
0.075
0.050
0.025
0.000
y
0.0
2.5
5.0
7.5
10.0
λ


Distribution of Randomly Generated m
0.015
0.010
0.005
0.000
y
0
20
40
60
m

```r
# Visuals
# Part a
p7 <- ggplot(accepted, aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  theme_minimal()
# p4
# Part b
# $m$ that are less than 10 or greater than 50.
accepted2 <- subset(accepted, m > 50 | m < 10,)
p8 <- ggplot(accepted2, aes(x = mu, y = lambda, color=m)) +
  geom_point(alpha=0.5) +
  labs(y = expression(lambda), x = expression(mu)) +
  theme_minimal()
# Part c
# A [hexplot](http://ggplot2.tidyverse.org/reference/stat_summary_2d.html) where each hex is filled wit
p9 <- ggplot(accepted, aes(x = mu, y = lambda, z = m)) +
  # geom_hex(fun = ~ quantile(m, 0.5)) +
  stat_summary_hex(fun = "quantile", fun.args = list(probs = 0.5)) +
  labs(y = expression(lambda), x = expression(mu)) +
  theme_minimal()
p7 / p8 / p9
```

Implement such a rejection sampling scheme for the changepoint problem and create the same plots from part I with the original priors and the data from `set.seed(497)`. Is the structure in the new sample from the joint posterior the same as when we used a Gibbs Sampler (up to Monte Carlo variability)? What about the Metropolis Algorithm? What are the downsides, as you see it, to the rejection sampling approach? What about the upsides?

Commentary

Though unfortunately I was unable to get working code for the Rejection Sampling method, please note the following observations—made as though the rejection sampling worked. We can at least be thankful that we generated the data correctly for the sample distribution.

Rejection sampling ensures no autocorrelation, but at the expense of lower efficiency (greater computation requirements). In-line with this, a lower efficiency comes with a lower acceptance rate for the Rejection Sampling method compared to Gibbs Sampling and the Metropolis Algorithm.

Conversely, Gibbs sampling and the Metropolis Algorithm are more efficient, but at the expense of autocorrelation, hence the usage of a burn-in period, thinning, and decimation.

Taken together, Rejection Sampling is the most flexible method considered in this problemset, but there are more efficient methods available–namely Gibbs Sampling and the Metropolis Algorithm. However, the more efficient methods come at the expense of data issues, namely autocorrelation, which then calls for additional processes to consider to remove the issues brought about by data issues, though once Mark Hopkins shares the paper he discussed, perhaps these aren't issues after all?

Note: this problem set has some serious computation in it, so I suggest cacheing your results by adding `cache = TRUE` to the r chunk options.