# MATH 611 HW7

Jeff Gould

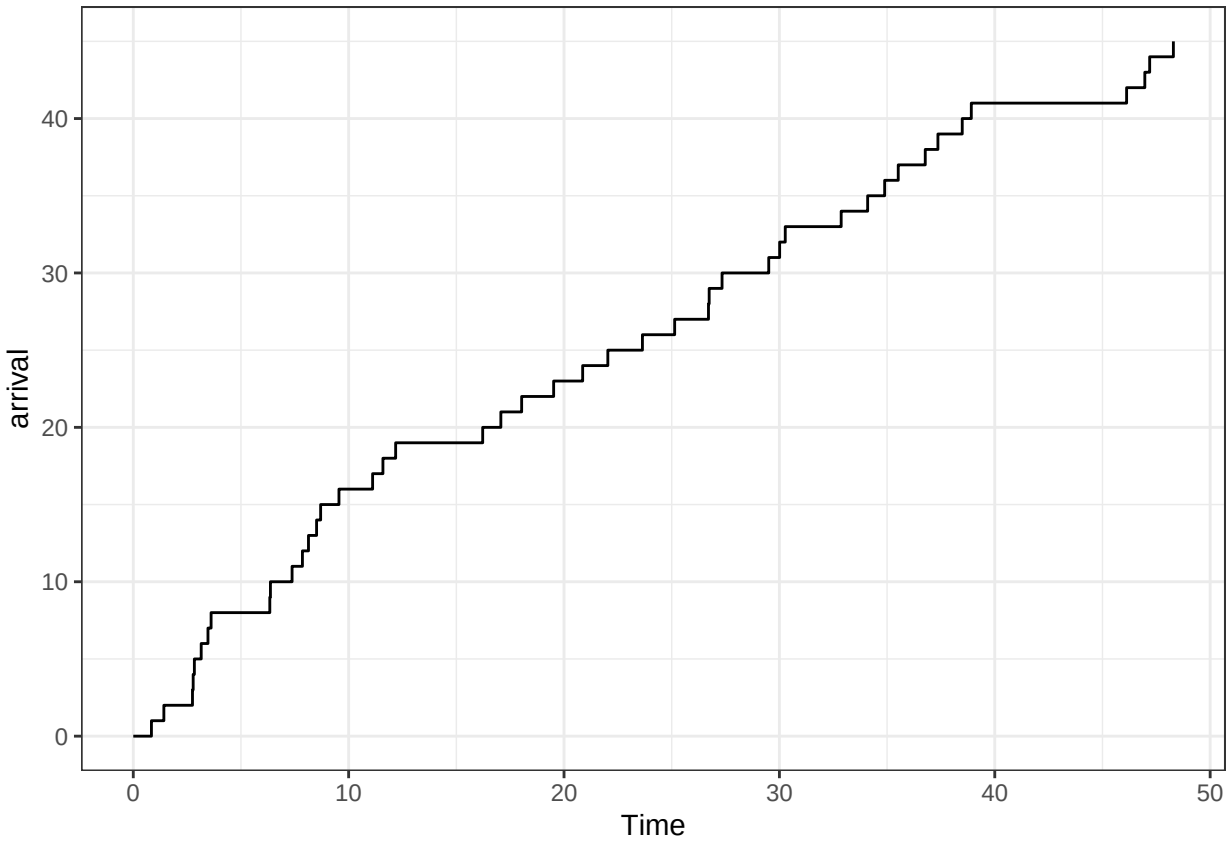
10/17/2020

## 1

$X(t)$ be a poisson process with $\lambda(t) = a + be^{ct}$ where $a = 5 \cdot 10^{-4}$, $b = 7.5858 \cdot 10^{-5}$, $c = \log(1.09144)$

**a) As a warm-up, write a function that samples the jump times of a Poisson process with constant rate $\lambda$ that occur prior to some specified time $t$.**

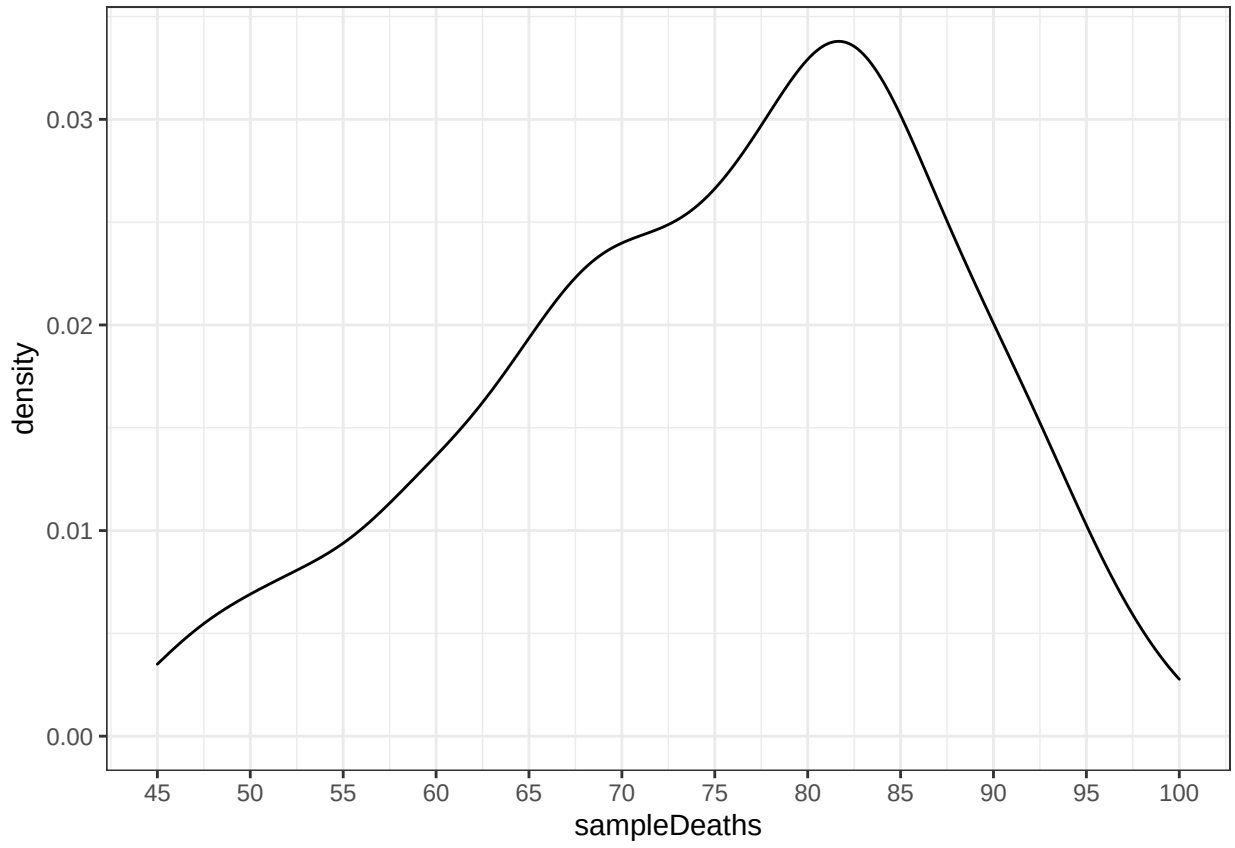
```
poisSample <- function(lambda, t, start_t = 0){

  T_i <- c(start_t)
  while(sum(T_i) < t){

    newT <- rexp(1, rate = lambda)
    T_i <- c(T_i, newT)

  }

  arrivals <- data.frame(arrival = seq(0, length(T_i)-1),
                         timeSinceLast = T_i) %>%
    mutate(Time = cumsum(timeSinceLast)) %>%
    filter(Time <= t) %>%
    select(-timeSinceLast)

  return(arrivals)

}

set.seed(123)
PoissonProcess <- poisSample(lambda = 1, t = 50)

ggplot(PoissonProcess, aes(x = Time, y = arrival)) +
  geom_step() +
  theme_bw()
```




**b) Write a sampler that generates the Poisson process $X(t)$ up to $t = 100$. Use the accept-reject algorithm that we talked about in class based on a constant rate Poisson process. Show that the accept-reject algorithm is valid. (Hint: to show the validity of the accept-reject algorithm show that it will sample a Poisson process with a jump in $[t, t + \Delta t]$ with probability approximately $\lambda(t)\Delta t$. We did this in class.)**

```
a <- 5e-04
b <- 7.5858e-05
c <- log(1.09144)



poisDeathProcess <- function(start_age = 45, max_age = 100){

  lambda_bar <- a + b * exp(c * max_age)

  ## First we draw from a poisson process using the function in the previous problem, and constant lamb
  deathSample <- poisSample(lambda = lambda_bar, t = max_age, start_t = start_age) %>%
    select(-arrival) %>% ## drop arrival time since it's not relevant
    rename(Age = Time) %>% ## rename Time to Age
    mutate(lambda_t = a + b * exp(c * Age)) %>% ## calcualte lambda(t)
    mutate(lambda_ratio = lambda_t / lambda_bar) ##lambda(t) / lambda_bar

  deathSample$u <- runif(n = nrow(deathSample)) ## draw from uniform(0,1) for each process
```

```
  deathSample %<>%
    mutate(accept = case_when(
      u < lambda_ratio ~ "yes",
      u > lambda_ratio ~ "no"
    )) %>%
    filter(accept == "yes") ## filter down to all the jumps that we would accept

  return(min(deathSample$Age, 100)) ## since you can only die once, return first jump. If no jump, they

}

sampleDeaths <- sapply(rep(45, 1000), poisDeathProcess)

sampleDeaths %>% as.data.frame() %>%
  ggplot(aes(x = sampleDeaths)) +
  geom_density() +
  theme_bw() +
  scale_x_continuous(breaks = seq(45, 100, 5))
```




```
mean(sampleDeaths)
```

```
## [1] 75.26139
```

**c) Derive a formula for the cdf of the age upon death of a 60 year old. (Don't just quote the result I presented in class, derive it yourself. Use the idea of splitting up time into small time intervals and taking a limit as those intervals go to 0 in size.)**

$\lambda(t) = a + b \cdot e^{ct}$ We want to find the probabilty of no jump in $[60, 90)$

In general: $P(\text{not jumping in time interval} [j\Delta t, (j+1)\Delta t)) = 1 - \lambda(j\Delta t)\Delta t$

$P(\text{no jump by time} t) = \prod_{j=1}^{t/\Delta t}(1 - \lambda(j\Delta t)\Delta t) \approx \lim_{\Delta t \to 0} e^{\sum \log(1-\lambda(j\Delta t)\Delta t)} = \lim_{\Delta t \to 0} e^{\sum_{j=1}^{t/\Delta t} -\lambda(j\Delta t)\Delta t} =$

$e^{-\int_{60}^{t} \lambda(s)ds} = e^{-\int_{60}^{t} a+b\cdot e^{cs}ds}$

Since this is the probabilty of not dying, then the cdf will be

$$1 - e^{-\int_{60}^{t} a+b\cdot e^{cs}ds}$$

**d) Compute the probability a 60 year old lives to be 90 in two ways**

- Use your languages integrate function to evaluate the formula for this probability based on your cdf in (c).

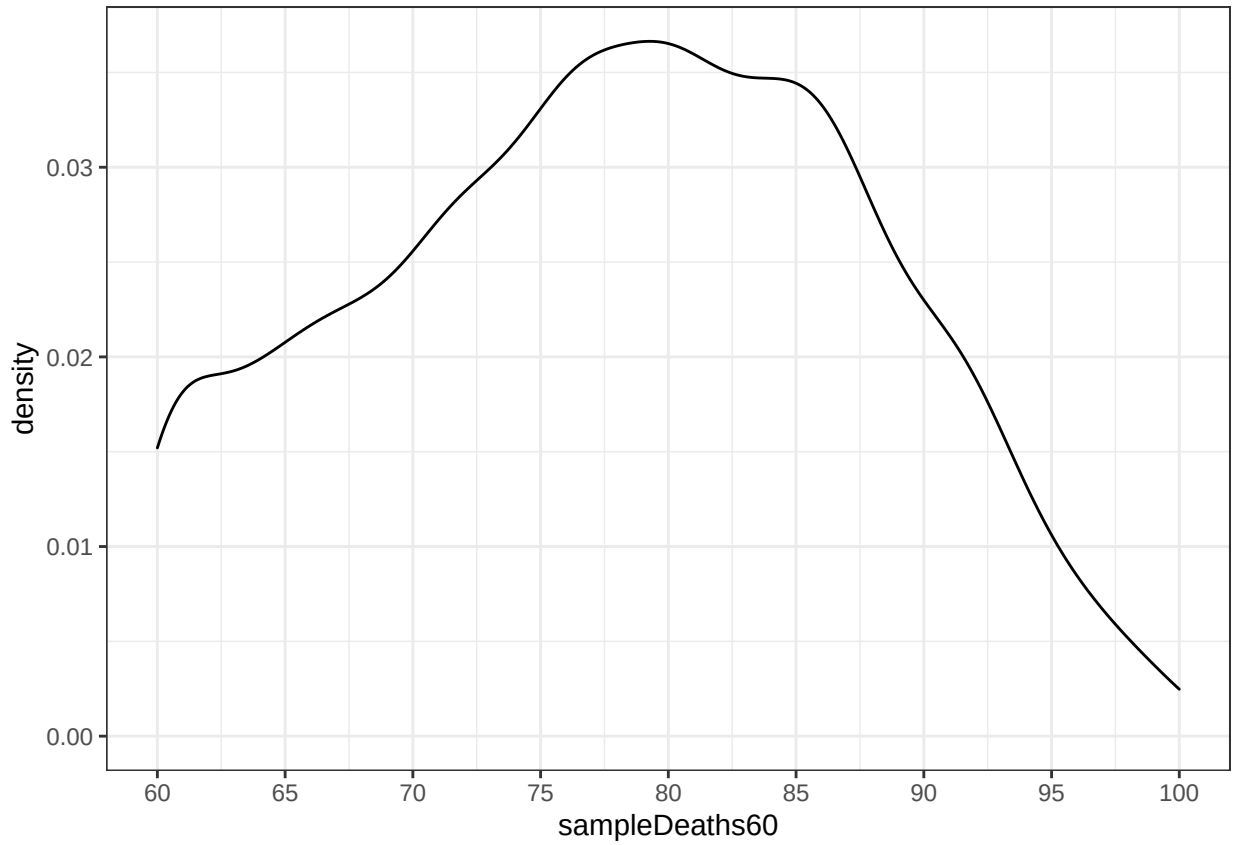
$$1 - (1 - e^{-\int_{60}^{t} a+b\cdot e^{cs}ds}) = e^{-\int_{60}^{t} a+b\cdot e^{cs}ds}$$

```
integrand <- function(x){a + b * exp(c * x)}
exp(-pracma::integral(integrand, 60, 90))
```

```
## [1] 0.118825
```

- Use a Monte Carlo integration approach and your sampler from (b)

```
sampleDeaths60 <- sapply(rep(60, 5000), poisDeathProcess)

sampleDeaths60 %>% as.data.frame() %>%
  ggplot(aes(x = sampleDeaths60)) +
  geom_density() +
  theme_bw() +
  scale_x_continuous(breaks = seq(60, 100, 5))
```




```
mean(sampleDeaths60 >= 90)
```

```
## [1] 0.1178
```

**2) Consider a Poisson process with rate $\lambda(t) = 1/\sqrt{t}$. Notice that $\lambda(t)$ is unbounded. Calculate the pdf of the first jump time.**

The probability of not having a jump by time $t$ is

$$e^{-\int_0^t \lambda(s)ds} = e^{-\int_0^t \frac{1}{\sqrt{s}}ds} = e^{-2\sqrt{t}}$$

$$P(T_1 > t) = e^{-2\sqrt{t}} \rightarrow P(T_1 < t) = 1 - e^{-2\sqrt{t}}$$

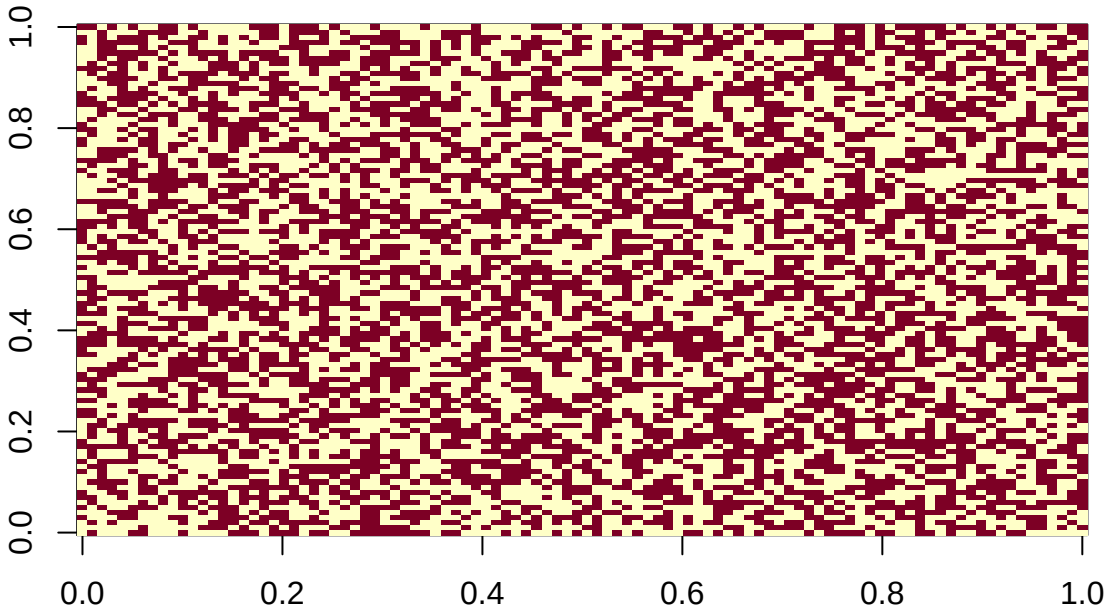
Is the cdf for the first time jump. To find the pdf, simply differentiate with respect to $t$

$$\frac{d}{dt}(1 - e^{-2\sqrt{t}}) = \frac{e^{-2\sqrt{t}}}{\sqrt{t}}$$

## 3)

**i) Redo problem $4b$ of homework $4$, but this time use a Gibbs sampler to sample from $Y$ rather than a Metropolis-Hastings sampler**

```
Y <- matrix(0, 100, 100)
grid_points <- expand_grid(x = 1:100, y = 1:100) %>% as.matrix()


iteration <- 0
while(iteration < 2){

  for (point in 1:10000) {
    i <- grid_points[point, 1]
    j <- grid_points[point, 2]
    omega <- Y
    omega[i,j] = 1

    omega_prime <- Y
    omega_prime[i,j] = 0

    nu_omega <- sum(omega)^2
    nu_omega_prime <- sum(omega_prime)^2

    if(runif(1) < (nu_omega / (nu_omega + nu_omega_prime))){
      Y <- omega
    }else{
      Y <- omega_prime
    }
  }


  iteration <- iteration + 1
}

image(Y)
```




```
gibbsSampleFy <- function(iter = 25, n = 100){

  Y <- matrix(0, n, n)
  grid_points <- as.matrix(tidyr::expand_grid(x = 1:n, y = 1:n))


  iteration <- 0
  while(iteration < iter){

    for (point in 1:n^2) {
      i <- grid_points[point, 1]
      j <- grid_points[point, 2]
      omega <- Y
      omega[i,j] = 1

      omega_prime <- Y
      omega_prime[i,j] = 0

      nu_omega <- sum(omega)^2
      nu_omega_prime <- sum(omega_prime)^2

      if(runif(1) < (nu_omega / (nu_omega + nu_omega_prime))){
        Y <- omega
      }else{
        Y <- omega_prime
      }
```

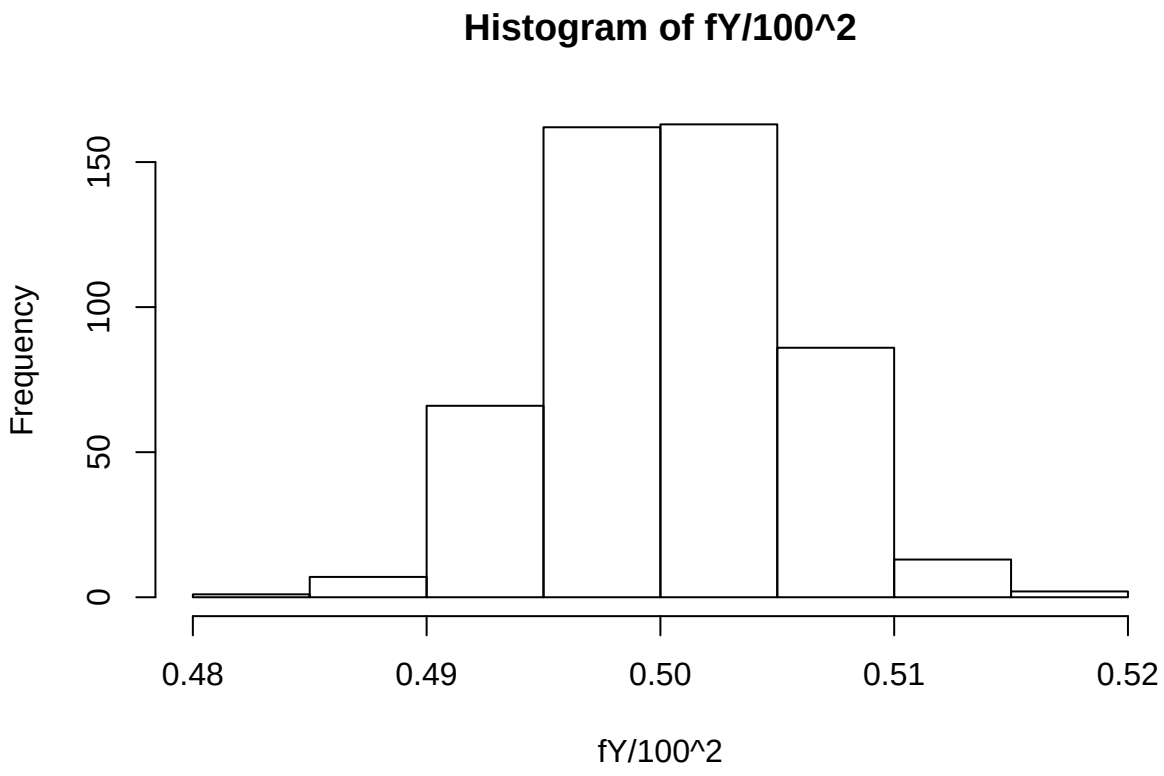
```
    }


    iteration <- iteration + 1
  }

  return(sum(Y))

}

library(parallel)
cl <- parallel::makeCluster(detectCores() - 1)
tictoc::tic()
fY <- parallel::parSapply(cl, rep(1, 500), gibbsSampleFy)
tictoc::toc()
```

```
## 198.379 sec elapsed
```

```
parallel::stopCluster(cl)

mean(fY/100^2)
```

```
## [1] 0.5005474
```

```
hist(fY/100^2)
```



We see that the hisotgram is very similar to the one we got previously, using the Metropolis-Hastings algorithm

**ii) Explain why you cannot use a Gibbs sampler based on switching a single coordinate (i.e. die role) to sample die rolls for problem 2 of homework 6**

We can't use a Gibbs Sampler for switching a single coordinate in the die problem because we would never be able to change the coordinate. If we have 100 die that sum to 450, and pull 1 die, and let's say that die is a 4. If we change the die to any number other than a 4, then the sum will no longer equal 450 and we will reject. So the die we put back will have to equal 4, and there will never be a change to the sample.

**iii) Construct a Gibbs sampler to sample die rolls for problem 2 of homework 6 by choosing two dice, computing their marginal distribution given all other die rolls, and sampling from the marginal to produce the Markov chain update. Such a sampler is not a Gibbs sampler in the sense that it considers one coordinate at a time, but it is a Gibbs sampler in the sense that it uses marginal distributions to update the chain**

For the marginal distribution of which dice we can sample from, consider this example:

We know the sum of the dice must sum to 450. If we pull two die, and their sum is 7, then the replacing dice we can sample from are:

$(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)$

Each of these have the same probabilty, and we assign the porbabilty of 0 to any other die combo.

For the function below, we will create a small data frame of every die combo. Then when we pull two dice to then sample replacements, we will draw from all the dice combos with the same sum, and replace.

We will redraw for half of dice pair combos

```
GibbsSamplerDice <- function(x){

  rolls <- sample(1:6, 100, replace = TRUE)

  ### Get to a sample of dice that sum to 450
  i <- 1
  while(sum(rolls) != x){

    new_sample <- sample(1:6, 10, replace = TRUE)

    rolls_new <- rolls
    rolls_new[(10*i - 9):(10*i)] <- new_sample

    if(sum(rolls_new) >= sum(rolls) & sum(rolls_new) <=x){rolls <- rolls_new}
    if(sum(rolls_new) <= sum(rolls) & sum(rolls_new) >=x){rolls <- rolls_new}

    i <- i + 1
    if(i == 11){i = 1}

  }

  ## Create a dataframe of all dice combos
  coords <- tidyr::expand_grid(a = 1:100, b = 1:100) %>%
    #filter(a != b) %>%
```

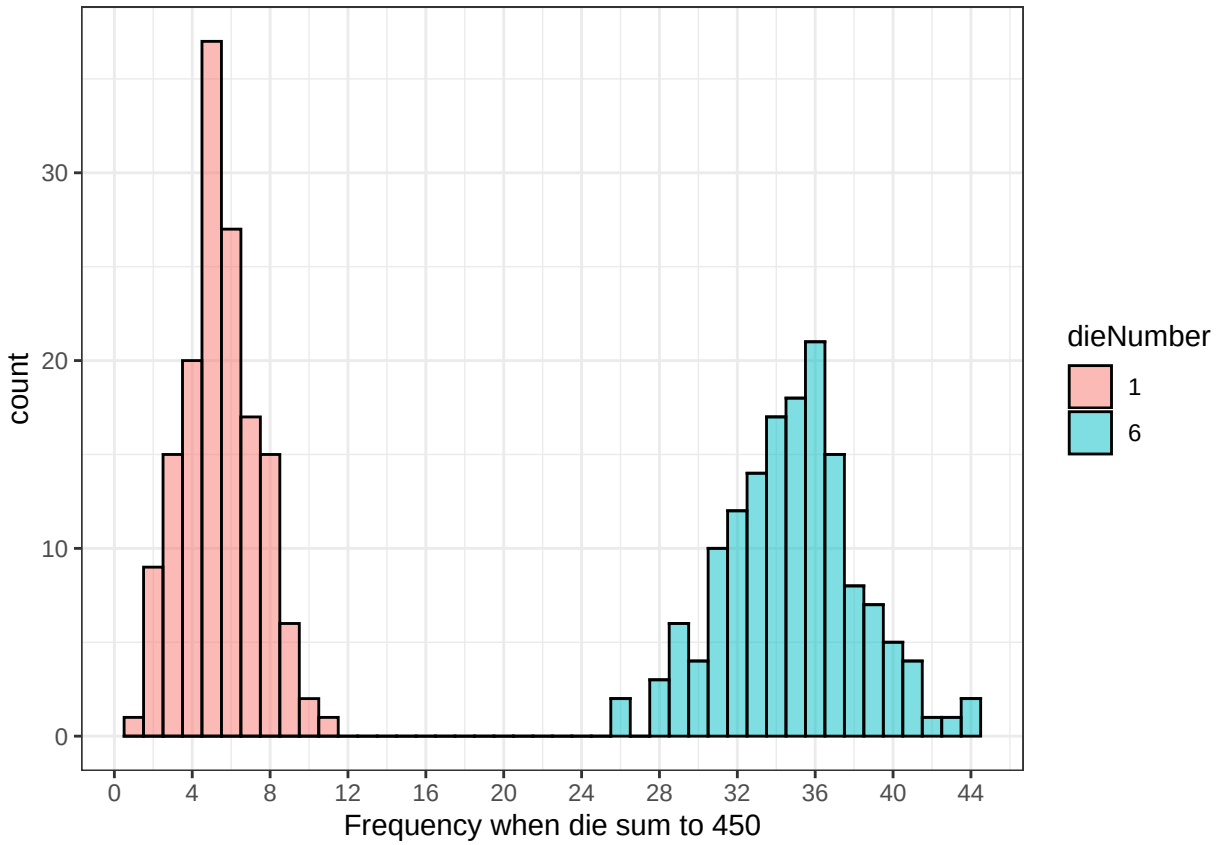
```
    filter(a < b) %>%
    sample_frac(0.55)

  roll_combos <- expand_grid(a = 1:6, b = 1:6) %>%
    mutate(rollSum = a + b)

  for(i in 1:nrow(coords)){

    a <- coords$a[i]
    b <- coords$b[i]

    replaceSum <- rolls[a] + rolls[b]

    newRoll <- roll_combos %>%
      filter(rollSum == replaceSum) %>% sample_n(1)

    rolls[a] = newRoll$a
    rolls[b] = newRoll$b

  }
  return(rolls)
}
```

```
cl <- makeCluster(detectCores() - 1)
clusterEvalQ(cl, library(tidyverse, quietly = TRUE))
```

```
## [[1]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[2]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[3]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[4]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[5]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[6]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
```

```
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[7]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[8]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[9]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[10]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
##
## [[11]]
##  [1] "forcats"   "stringr"   "dplyr"     "purrr"     "readr"     "tidyr"
##  [7] "tibble"    "ggplot2"   "tidyverse" "stats"     "graphics"  "grDevices"
## [13] "utils"     "datasets"  "methods"   "base"
```

```
rollSamples <- parSapply(cl, rep(450, 150), GibbsSamplerDice)
stopCluster(cl)

E_6 <- apply(rollSamples, 2, function(x)mean(x == 6)*100)
E_1 <- apply(rollSamples, 2, function(x)mean(x == 1)*100)

EVs <- data.frame(NumberInRoll = c(E_6, E_1),
                  dieNumber = c(rep("6", length(E_6)), rep("1", length(E_1))))

ggplot(EVs, aes(x = NumberInRoll, fill = dieNumber)) +
  geom_histogram(alpha = 0.5, color = "black", binwidth = 1) +
  theme_bw() +
  labs(x = "Frequency when die sum to 450") +
  scale_x_continuous(breaks = seq(0,48, 4))
```


count
30
20
10
0
0
4
8
12
16
20
24
28
32
36
40
44
Frequency when die sum to 450
dieNumber
1
6