Chutes and Ladders and MCMC

Prompt

Original Prompt can be found here. A copy of the prompt along with the completed exercise can be found under /Applications.

Summary

This investigation concerns the boardgame Chutes and Ladders. Detailed instructions and some code have been provided in original prompt; For detailed explaination, please be sure to read the full prompt carefully.

The board has 100 spaces, labeled 1, 2, …, 100. A player starts off the board. A player generally moves on the board according to the roll of a fair six-sided die. For example, if the player is currently on space 13 and they roll a 5, then they move to space 18. However, the board also has 9 ladders which help the player climb the board and 10 chutes (slides) which knock the player back down. The game ends when the player makes it to space 100. (We'll assume only one player.)

Problem 1

We are interested in T, the number of moves (rolls) needed until spot 100 is reached (the player doesn't need to land on 100 exactly). The position of the player after the nth move can be modeled as a Markov chain with transition matrix `Pgame` defined in the code from the prompt.

Problem 2

Suppose you were designing a new Chutes and Ladders board. How does the placement of the chutes and ladders on the board affect the expected value of T? In particular, is there a way to place the chutes/ladders to minimize the expected number of moves? In this problem, you'll write an MCMC algorithm to find the board which minimizes $E(T)$.

Application

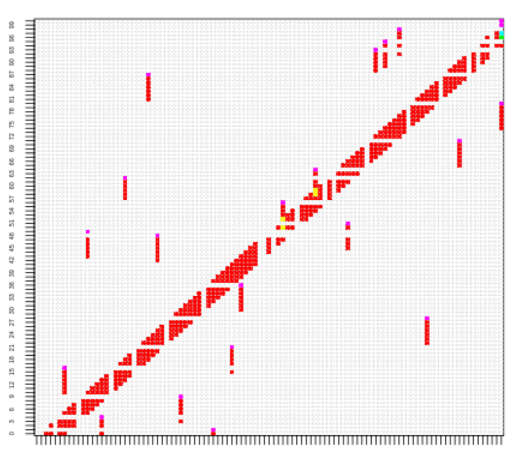
1.

First, create the Pgame matrix to use.

```
#use this to allow for running R within Python
%load_ext rpy2.ipython
```

```
%%R
N = 100 # number of spaces on board

s = N + 1  # number of states

k = 6  # number of sides on die

# P0 is the transition matrix if there were no chutes/ladders
P0 = matrix(rep(0, s * s), nrow = s)

for (i in 1:(N - 1)){
  for (j in min(i + 1, N):min(i + k, N)){
    if (j == N){
      P0[i, j] = (i - N + k + 1) / k
      # don't need to land on 100 exactly
    } else {
      P0[i, j] = 1 / k
    }
  }
}

P0[N, N] = 1  # absorbing state

P0[s, 1:k] = 1 / k  # initial state
```

```
%%R
# The make_board function takes as an input the starting spaces
# for chutes and ladders and outputs the transition matrix
# add the chutes/ladders by swapping appropriate columns
# with an annoying little detail for the two short chutes
# e.g. you can get from 50 to 53 by rolling a 3
# or by rolling a 6 and then sliding down the chute from 56 to 53

make_board <- function(ladder_start, chute_start, plot = FALSE){
  ladder_length = c(8, 10, 16, 20, 20, 21, 22, 37, 56)
  ladder_end = ladder_start + ladder_length
  chute_length = c(3, 4, 10, 20, 20, 20, 22, 38, 43, 63)
  chute_end = chute_start - chute_length
  P = P0
  for (j in 1:length(ladder_start)){
    i = which(P[, ladder_start[j]] > 0)
    P[i, ladder_end[j]] = P[i, ladder_start[j]]
    P[i, ladder_start[j]] = 0
    P[ladder_start[j], ] = rep(0, s)
    P[ladder_start[j], ladder_end[j]]=1
  }
  for (j in 1:length(chute_start)){
    i = which(P[, chute_start[j]] > 0)
    i1 = i[which(i <= chute_end[j])]
    P[i1, chute_end[j]] = P[i1, chute_start[j]] +
      P0[i1, chute_end[j]]
    P[i1, chute_start[j]] = 0
    i2 = i[which(i > chute_end[j])]
    P[i2, chute_end[j]] = P[i2, chute_start[j]]
    P[i2, chute_start[j]] = 0
    P[chute_start[j], ] = rep(0, s)
    P[chute_start[j], chute_end[j]] = 1
  }
  if (plot == TRUE){
    image(1:s, 1:s, t(P[c(s, 1:(s - 1)), c(s, 1:(s - 1))]),
          xlab = "", ylab = "",
          zlim = c(1 / k, 1), xaxt = "n", yaxt = "n",
          col = rainbow(k))
    axis(1, at = 1:s, labels = 0:(s - 1), cex.axis=0.4)
    axis(2, at = 1:s, labels = 0:(s - 1), cex.axis=0.4)
    grid(s, s)
  }
  return(P)
}


#run in R environment but export output variable to python
%%R -o Pgame
# generate the transition matrix for the actual game
Pgame = make_board(
  ladder_start = c(36, 4, 51, 71, 80, 21, 9, 1, 28),
  chute_start = c(56, 64, 16, 93, 95, 98, 48, 49, 62, 87),
  plot = TRUE)
```



```
%%R
# Check that all row sums are 1
Invalid_rows_n = which(!rowSums(Pgame) == 1)
print(Invalid_rows_n)
```

```
integer(0)
```

a.

Solve for $E(T)$ without first finding the distribution of T.

```
%%R
mean_time_to_absorption <- function(transition_matrix, state_names = NULL) {

  absorbing_states = which(diag(transition_matrix) == 1)

  if (length(absorbing_states) == 0) stop("There are no absorbing states.")

  n_states = nrow(transition_matrix)

  transient_states = setdiff(1:n_states, absorbing_states)

  Q = transition_matrix[transient_states, transient_states]

  mtta = solve(diag(nrow(Q)) - Q, rep(1, nrow(Q)))

  if (is.null(state_names)) state_names = 1:n_states

  data.frame(start_state = state_names[transient_states],
             mean_time_to_absorption = mtta)
}
```

```
%%R
mu = mean_time_to_absorption(Pgame)
mu[100,]
```

```
    start_state mean_time_to_absorption
100         101                36.19307
```

Above we can see the mean absorbtion time, $E(T)$, from off the board to spot 100.

b.

Solve for the exact distribution of T and plot it. (Technically, T can take infinitely many values, but feel free to cut off when the probabilities become sufficiently small.) Find $E(T)$ based on this distribution. Compare the expected value to the previous part.

```
%%R
install.packages('expm')
install.packages('kableExtra')
install.packages('tidyverse')
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: =
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: downloaded 688 KB


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: The downloaded source packages are in
        '/tmp/Rtmpt3r79s/downloaded_packages'
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
```

```
%%R
library(expm)
library(kableExtra)

pmf_of_time_to_absorption <- function(transition_matrix, state_names = NULL, start_state) {

  absorbing_states = which(diag(transition_matrix) == 1)

  if (length(absorbing_states) == 0) stop("There are no absorbing states.")

  n_states = nrow(transition_matrix)

  transient_states = setdiff(1:n_states, absorbing_states)

  if (is.null(state_names)) state_names = 1:n_states

  if (which(state_names == start_state) %in% absorbing_states) stop("Initial state is an absorbing state; absorption at time 0.")

  n = 1

  TTA_cdf = sum(transition_matrix[which(state_names == start_state), absorbing_states])

  while (max(TTA_cdf) < 0.999999) {

    n = n + 1

    TTA_cdf = c(TTA_cdf, sum((transition_matrix %^% n)[which(state_names == start_state), absorbing_states]))
  }

  TTA_pmf = TTA_cdf - c(0, TTA_cdf[-length(TTA_cdf)])

  data.frame(n = 1:length(TTA_pmf),
             prob_absorb_at_time_n = TTA_pmf)
}
```

```
WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: Loading required package: Matrix

WARNING:rpy2.rinterface_lib.callbacks:R[write to console]:
Attaching package: 'expm'


WARNING:rpy2.rinterface_lib.callbacks:R[write to console]: The following object is masked from 'package:Matrix':

    expm
```

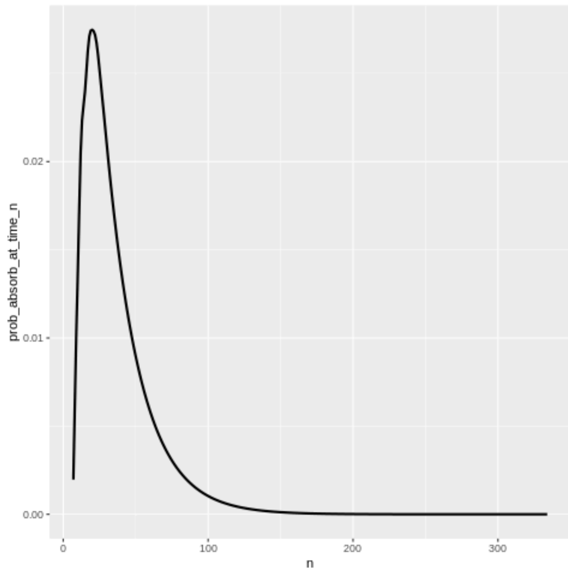
```
%%R
T_pmf = pmf_of_time_to_absorption(Pgame, start_state = 101)

T_pmf |> head(100)
```

```
      n prob_absorb_at_time_n
1     1           0.000000000
2     2           0.000000000
3     3           0.000000000
4     4           0.000000000
5     5           0.000000000
6     6           0.000000000
7     7           0.001971879
8     8           0.006174626
9     9           0.010176731
10   10           0.013461394
11   11           0.017124516
12   12           0.020461774
```

13	13	0.022348204
14	14	0.023169028
15	15	0.023949501
16	16	0.025097382
17	17	0.026294356
18	18	0.027100283
19	19	0.027427515
20	20	0.027474270
21	21	0.027397815
22	22	0.027177541
23	23	0.026731643
24	24	0.026059559
25	25	0.025252409
26	26	0.024409494
27	27	0.023574142
28	28	0.022739104
29	29	0.021888860
30	30	0.021027647
31	31	0.020176073
32	32	0.019353038
33	33	0.018564695
34	34	0.017806949
35	35	0.017074119
36	36	0.016364529
37	37	0.015680137
38	38	0.015023213
39	39	0.014394041
40	40	0.013791061
41	41	0.013212330
42	42	0.012656620
43	43	0.012123458
44	44	0.011612542
45	45	0.011123253
46	46	0.010654628
47	47	0.010205616
48	48	0.009775312
49	49	0.009362995
50	50	0.008968023
51	51	0.008589733
52	52	0.008227417
53	53	0.007880368
54	54	0.007547920
55	55	0.007229467
56	56	0.006924440
57	57	0.006632287

```
%%R
library(tidyverse)

ggplot(T_pmf |>
         filter(prob_absorb_at_time_n > 0),
       aes(x = n,
           y = prob_absorb_at_time_n)) +
  geom_line(linewidth = 1)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.4.4     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ──────────────────────────────────────────── tidyverse_conflicts() ──
✖ tidyr::expand()     masks Matrix::expand()
✖ dplyr::filter()     masks stats::filter()
✖ dplyr::group_rows() masks kableExtra::group_rows()
✖ dplyr::lag()        masks stats::lag()
✖ tidyr::pack()       masks Matrix::pack()
✖ tidyr::unpack()     masks Matrix::unpack()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```




```
%%R
sum(T_pmf[, 1] * T_pmf[, 2])
```

```
[1] 36.19272
```

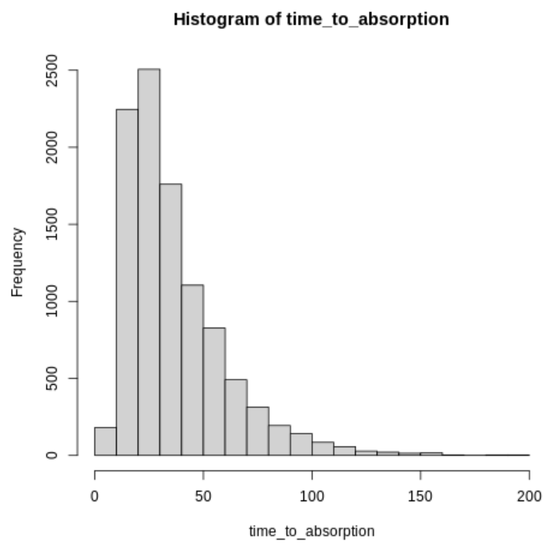
Both computation through absorbing state and cumulative PMF to find average result in values that agree. They both come out to be about 36.2 steps.

C.

Write code to run the chain and simulate the distribution of T. Plot the simulated distribution, use it to estimate the expected value, and compare to the previous part.

```
%%R
simulate_single_DTMC_path <- function(initial_distribution, transition_matrix, last_time){

  n_states = nrow(transition_matrix) # number of states

  states = 1:n_states # state space

  X = rep(NA, last_time + 1) # state at time n; +1 to include time 0

  X[1] = sample(states, 1, replace = TRUE, prob = initial_distribution) # initial state

  for (n in 2:(last_time + 1)){

    X[n] = sample(states, 1, replace = TRUE, prob = transition_matrix[X[n-1], ])

  }

  return(X)

}
```

```
%%R
pi0 <- rep(0, 101)
pi0[101] <- 1

absorbing_states = which(diag(Pgame) == 1)

n_rep = 10000
time_to_absorption = rep(NA, n_rep)

for (i in 1:n_rep) {
  x = simulate_single_DTMC_path(pi0, Pgame, last_time = 200)
  time_to_absorption[i] = min(which(x %in% absorbing_states))
}

hist(time_to_absorption)
```




```
%%R
summary(time_to_absorption)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      8      21      31     Inf      47     Inf 
```

```
%%R
mean(time_to_absorption)
```

```
[1] Inf
```

```
%%R
sd(time_to_absorption)
```

```
[1] NaN
```

2.

First, think about what the optimal placement might look like. Then, write an MCMC algorithm to find the board that minimizes $E(T)$. Your MCMC algorithm should involve:

- Proposing a new state, that is, proposing a new board. A board is identified by the starting spaces of the chutes and the starting spaces of the ladders (that is, the inputs to the `make_board` function).
- Finding the expected value of T for the proposed board and then deciding whether or not to accept the proposed board. Note: if the proposed board is not valid (e.g., chutes/ladders land off the board), then it should be rejected.

Run the algorithm until you think it has converged and you have found the optimal board. Identify the the starting spaces for the chutes and ladders for this board.

MCMC Algorithm

```
# Function - create proposal and verify it is valid
%%R

propose_starting_locations <- function(ladder_start, chute_start) {
  # Find end for chutes and ladders
  ladder_length <- c(8, 10, 16, 20, 20, 21, 22, 37, 56)
  ladder_end <- ladder_start + ladder_length
  chute_length <- c(3, 4, 10, 20, 20, 20, 22, 38, 43, 63)
  chute_end <- chute_start - chute_length

  # Randomly choose chutes or ladders to adjust
  adjust_type <- sample(c("ladder", "chute"), 1)

  # Randomly pick which element to adjust
  if (adjust_type == "ladder") {
    element_index <- sample(length(ladder_length), 1)
    start <- 1
    end   <- 100 - ladder_length[element_index]
    new_start <- sample(start:end, 1)  # Generate new starting position
    new_end   <- new_start + ladder_length[element_index]
    while (new_start %in% c(ladder_start, ladder_end, chute_start, chute_end) ||
           new_end %in% c(ladder_start, ladder_end, chute_start, chute_end)) {
      new_start <- sample(start:end, 1)  # Regenerate if position is not valid
      new_end   <- new_start + ladder_length[element_index]
    }
    new_ladder_start <- replace(ladder_start, element_index, new_start)
    new_chute_start  <- chute_start  # Chute positions remain unchanged
  } else {
    element_index <- sample(length(chute_length), 1)
    start <- 4 + chute_length[element_index]
    end   <- 99
    new_start <- sample(start:end, 1)  # Generate new starting position
    new_end   <- new_start - chute_length[element_index]
    while (new_start %in% c(ladder_start, ladder_end, chute_start, chute_end) ||
           new_end %in% c(ladder_start, ladder_end, chute_start, chute_end)) {
      new_start <- sample(start:end, 1)  # Regenerate if position is not valid
      new_end   <- new_start - chute_length[element_index]
    }
    new_chute_start  <- replace(chute_start, element_index, new_start)
    new_ladder_start <- ladder_start  # Ladder positions remain unchanged
  }

  # Return a list containing the new starting locations for ladders and chutes
  return(list(new_ladder_start = new_ladder_start, new_chute_start = new_chute_start))
}


# Example usage:
ladder_start <- c(36, 4, 51, 71, 80, 21, 9, 1, 28)
chute_start <- c(56, 64, 16, 93, 95, 98, 48, 49, 62, 87)

result <- propose_starting_locations(ladder_start, chute_start)
new_ladder_start <- result$new_ladder_start
new_chute_start <- result$new_chute_start

# Print the new starting locations
ladder_length <- c(8, 10, 16, 20, 20, 21, 22, 37, 56)
chute_length <- c(3, 4, 10, 20, 20, 20, 22, 38, 43, 63)

print(ladder_start)
print(new_ladder_start)
print(ladder_length)
print(new_ladder_start+ladder_length)

print(chute_start)
print(new_chute_start)
print(chute_length)
print(new_chute_start-chute_length)
```

```
[1] 36  4 51 71 80 21  9  1 28
[1] 36  4 51 71 80 61  9  1 28
[1]  8 10 16 20 20 21 22 37 56
[1]  44  14  67  91 100  82  31  38  84
 [1] 56 64 16 93 95 98 48 49 62 87
 [1] 56 64 16 93 95 98 48 49 62 87
 [1]  3  4 10 20 20 20 22 38 43 63
 [1] 53 60  6 73 75 78 26 11 19 24
```

```
%%R
Pgame = make_board(new_ladder_start, new_chute_start, plot = FALSE)

# Check that all row sums are 1
Invalid_rows_n = which(!rowSums(Pgame) == 1)
print(Invalid_rows_n)
```

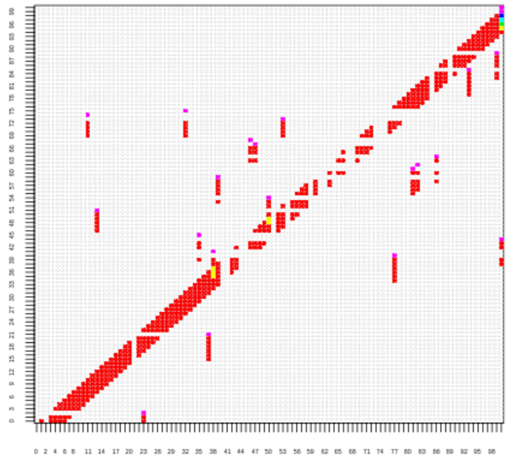
```
integer(0)
```

```
#run MCMC algorithm and find min E(T)
%%R

#start with original board
ladder_start = c(36, 4, 51, 71, 80, 21, 9, 1, 28)
chute_start = c(56, 64, 16, 93, 95, 98, 48, 49, 62, 87)
Pgame = make_board(ladder_start, chute_start, plot = FALSE)

mu = mean_time_to_absorption(Pgame)
T_curr = mu[100,2]

T_min = T_curr
Pgame_min = Pgame


#run simulation for number of steps and find min T
sim_length = 50000
for (i in 1:sim_length){

    #propose new board
    result <- propose_starting_locations(ladder_start, chute_start)
    new_ladder_start <- result$new_ladder_start
    new_chute_start <- result$new_chute_start

    #make the board and find T
    Pgame = make_board(new_ladder_start, new_chute_start, plot = FALSE)
    mu = mean_time_to_absorption(Pgame)
    T_prop = mu[100,2]

    #test if it is a minimum
    if (T_prop < T_min)
    {
        T_min = T_prop
        ladder_min = new_ladder_start
        chute_min = new_chute_start
    }

    #determine acceptance probability and get determination
    a = min((1/T_prop)/(1/T_curr),1)
    action = sample(c("reject", "accept"), 1, prob = c(1 - a, a))

    if (action == "accept"){
      ladder_start = new_ladder_start
      chute_start = new_chute_start
      T_curr = T_prop
    }

}
```

Comparison

Repeat Problem 1 for your optimal board, and compare the distribution of T for your board to the one from the actual game. Write a few sentences summarizing your results.

```
%%R
Pgame_min = make_board(ladder_min, chute_min, plot = TRUE)
```



```
%%R
# Check that all row sums are 1
Invalid_rows_n = which(!rowSums(Pgame_min) == 1)
print(Invalid_rows_n)
```

```
integer(0)
```

a.

Solve for $E(T)$ without first finding the distribution of T.

```
%%R
mu = mean_time_to_absorption(Pgame_min)
T = mu[100,]
print(T)
```

```
    start_state mean_time_to_absorption
100         101                17.83413
```

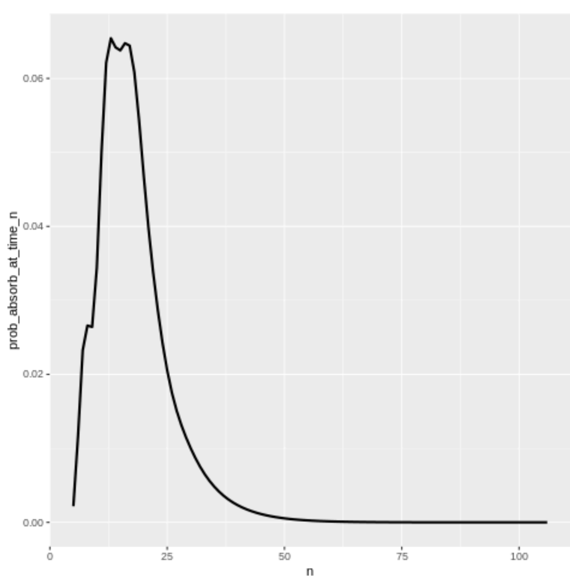
b.

Solve for the exact distribution of T and plot it. Find $E(T)$ based on this distribution.

```
%%R
T_pmf = pmf_of_time_to_absorption(Pgame_min, start_state = 101)

T_pmf |> head(100)
```

```
70   70          2.933893e-05
71   71          2.535667e-05
72   72          2.191493e-05
73   73          1.894035e-05
74   74          1.636951e-05
75   75          1.414763e-05
76   76          1.222733e-05
77   77          1.056768e-05
78   78          9.133297e-06
79   79          7.893608e-06
80   80          6.822186e-06
81   81          5.896191e-06
82   82          5.095884e-06
83   83          4.404204e-06
84   84          3.806409e-06
85   85          3.289754e-06
86   86          2.843226e-06
87   87          2.457306e-06
88   88          2.123769e-06
89   89          1.835504e-06
90   90          1.586365e-06
91   91          1.371043e-06
92   92          1.184948e-06
93   93          1.024111e-06
94   94          8.851056e-07
95   95          7.649676e-07
96   96          6.611363e-07
97   97          5.713984e-07
98   98          4.938408e-07
99   99          4.268103e-07
100 100          3.688781e-07
```

```
%%R
library(tidyverse)

ggplot(T_pmf |>
         filter(prob_absorb_at_time_n > 0),
       aes(x = n,
           y = prob_absorb_at_time_n)) +
  geom_line(linewidth = 1)
```




```
%%R
sum(T_pmf[, 1] * T_pmf[, 2])
```

```
[1] 17.83402
```

c.

Write code to run the chain and simulate the distribution of T. Plot the simulated distribution, use it to estimate the expected value.