Mark Stanley: RNG,Markov Chains, Drunkards Walk

2024-03-30

Linear Congruence Method for pseudo random numbers (PRN):

```r
lin_congruence <- function(n, m, a, c, X0) {
  X <- numeric(n)  # store sequence here
  X[1] <- X0  # first value is seed
  for (i in 2:n) {
    X[i] <- (a * X[i - 1] + c) %% m  # find next val in sequence
  }
  return(X)
}

# testing
m <- 8
a <- 5
c <- 1
X0 <- 2
n <- 100
random_sequence <- lin_congruence(n, m, a, c, X0)
print(random_sequence)
```

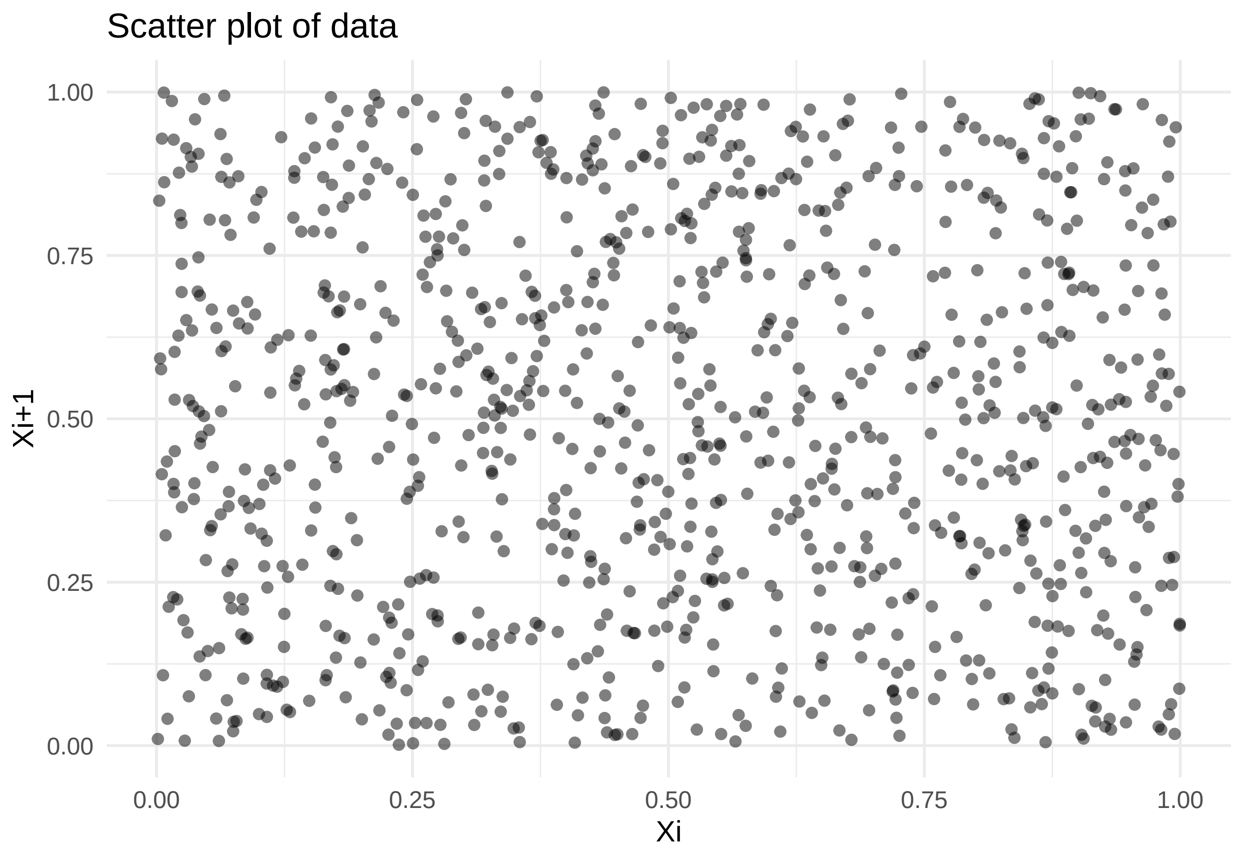
```
##   [1] 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6
##  [38] 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3
##  [75] 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1 6 7 4 5 2 3 0 1
```

Here the method produces a cycle, which occurs when c is not very large. This is the reason that most computers use c = 2^31.

Given a sequence X1, X2, . . . , Xn of U[0,1]- distributed PRN, we can use a scatter plot of (Xi, Xi+1) for i = 1, . . . , n - 1 in order to try to assess whether the Xi are independent.

Plot using the built in random number generator in R:

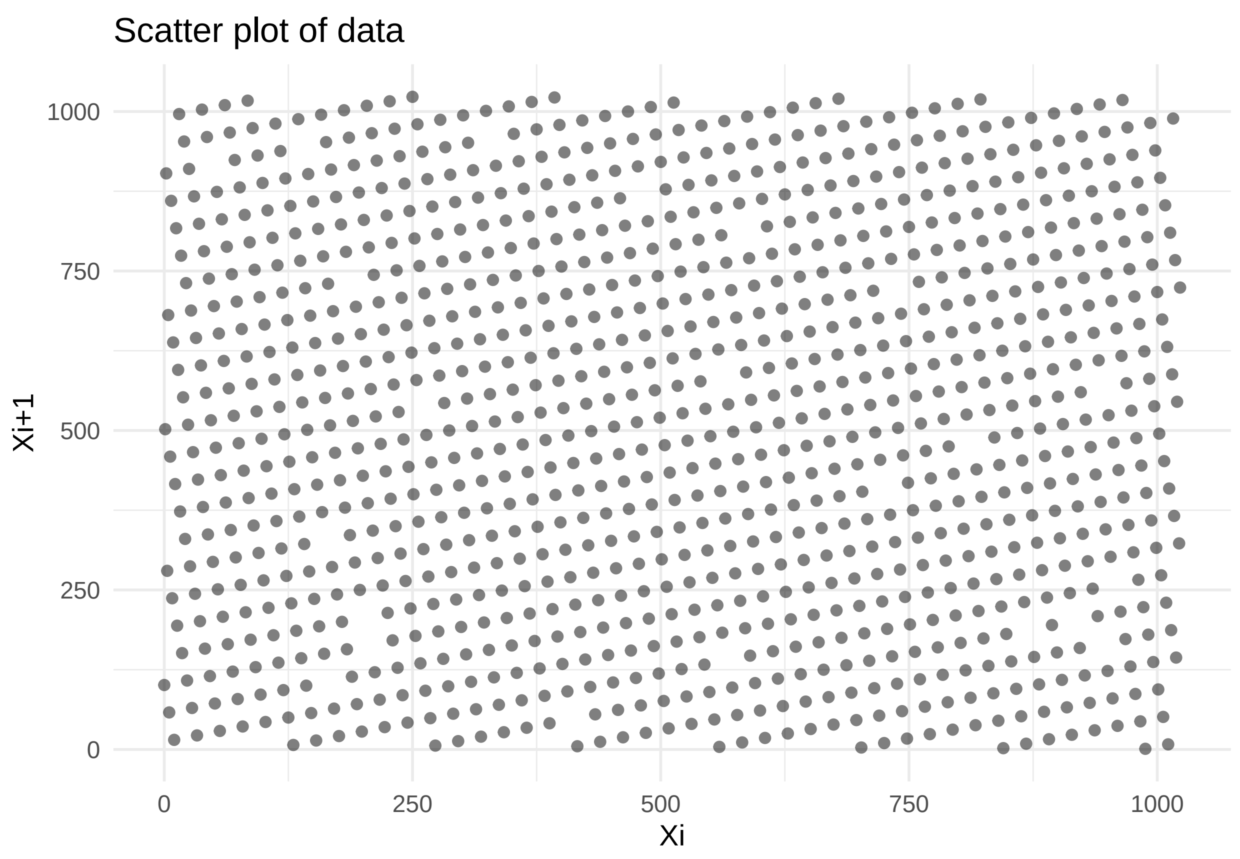
```r
library(ggplot2)

n <- 1000
random_sequence <- runif(n) # generate distributions

data <- data.frame(Xi = random_sequence[1:(n-1)], Xi1 = random_sequence[2:n])

# create scatter plot
ggplot(data, aes(x = Xi, y = Xi1)) +
  geom_point(alpha = 0.5) +
  labs(x = "Xi", y = "Xi+1", title = "Scatter plot of data") +
  theme_minimal()
```



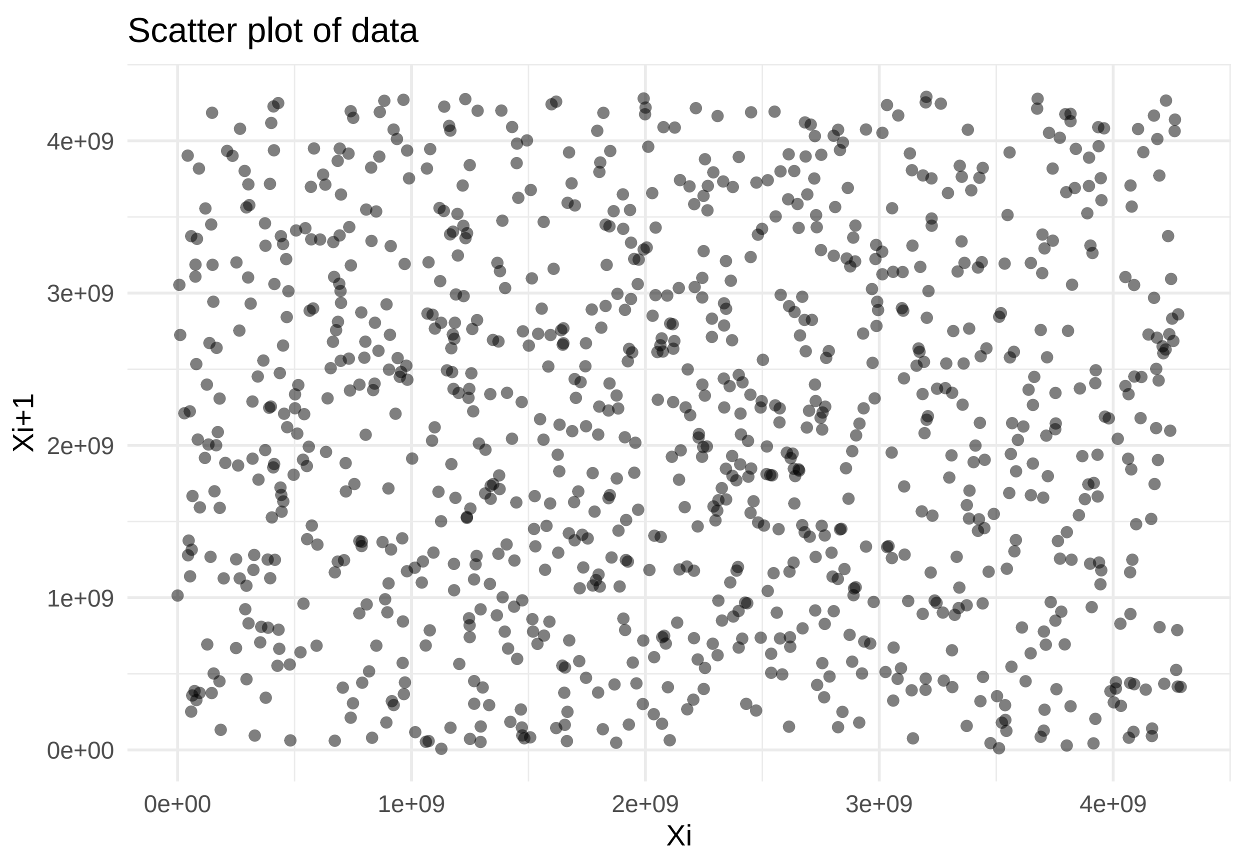
If each value is generated randomly, then pairing up coordinates will be random all the way and create a plot of points that are evenly dispersed all around.

Previous experiment using the parameters m = 1024, a = 401, c = 101 and m = 232, a = 1664525, c = 1013904223:

```
random_sequence <- lin_congruence(1000, 1024, 401, 101, 0)


data <- data.frame(Xi = random_sequence[1:(n-1)], Xi1 = random_sequence[2:n])

ggplot(data, aes(x = Xi, y = Xi1)) +
  geom_point(alpha = 0.5) +
  labs(x = "Xi", y = "Xi+1", title = "Scatter plot of data") +
  theme_minimal()
```



```
random_sequence <- lin_congruence(1000, 2^32, 1664525, 1013904223, 0)

data <- data.frame(Xi = random_sequence[1:(n-1)], Xi1 = random_sequence[2:n])

# make scatter
ggplot(data, aes(x = Xi, y = Xi1)) +
  geom_point(alpha = 0.5) +
  labs(x = "Xi", y = "Xi+1", title = "Scatter plot of data") +
  theme_minimal()
```



In the first test case here, there is a pattern, but the numbers are still distributed pretty evenly. In the second case, the mod is so large that the data is very random, this is why computers use this value to generate values as random as possible. In both these examples, I was not sure if it was required to shrink the data between 0 and 1, but if that were the case, the data would still be evenly distributed.

Here is a drunkard's walk simulation in two dimensions. Each step, the walker randomly chooses one of eight possible directions (North, South, East, West, Northeast, Southeast,Southwest, Northwest) with varying probabilities

Here are some probabilities that I will use: North: 10% South: 15% East: 15% West: 20% Northeast: 10% Southeast: 10% Southwest: 10% Northwest: 10%

```
probabilities <- c(North = 0.1, South = 0.15, East = 0.15, West = 0.2,
                   Northeast = 0.1, Southeast = 0.1, Southwest = 0.1, Northwest = 0.1)

# need probabilities vector
```

We write a function to simulate a step. It returns a vector representing the step in the chosen direction, where the first element indicates the change in the x-coordinate (East/West) and the second element indicates the change in the y-coordinate (North/South):

```
simulate_step <- function() {
  direction <- sample(names(probabilities), 1, prob = probabilities)

  step <- switch(direction,
                 North = c(0, 1),
                 South = c(0, -1),
                 East = c(1, 0),
                 West = c(-1, 0),
                 Northeast = c(1, 1),
                 Southeast = c(1, -1),
                 Southwest = c(-1, -1),
                 Northwest = c(-1, 1))

  # change in x and y decided by vector
  # used a simple switch statement

  return(step)
}

# test function
step <- simulate_step()
print(step)
```

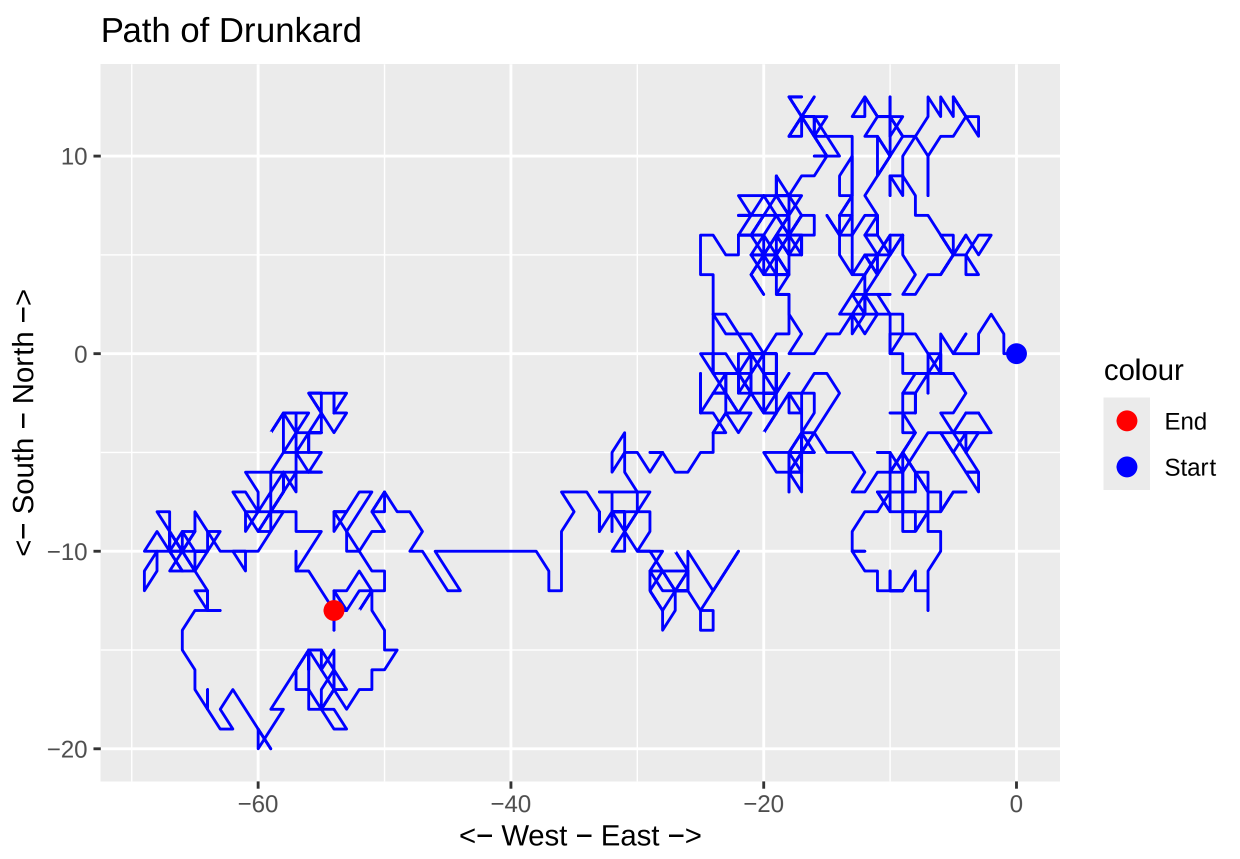
```
## [1] 1 0
```

1000 step simulation: We initialize a matrix named positions with dimensions (1001, 2) to store the positions of the drunkard at each step. Use a for loop to iteratively call the simulate_step function and update the drunkard's position accordingly:

```
num_steps <- 1001

positions <- matrix(0, nrow = num_steps, ncol = 2)


positions[1, ] <- c(0, 0) # start position

for (i in 2:(num_steps)) {
  step <- simulate_step()
  positions[i, ] <- positions[i - 1, ] + step
  # update step positions
}
```

Plot the path of the walk in two dimensions:

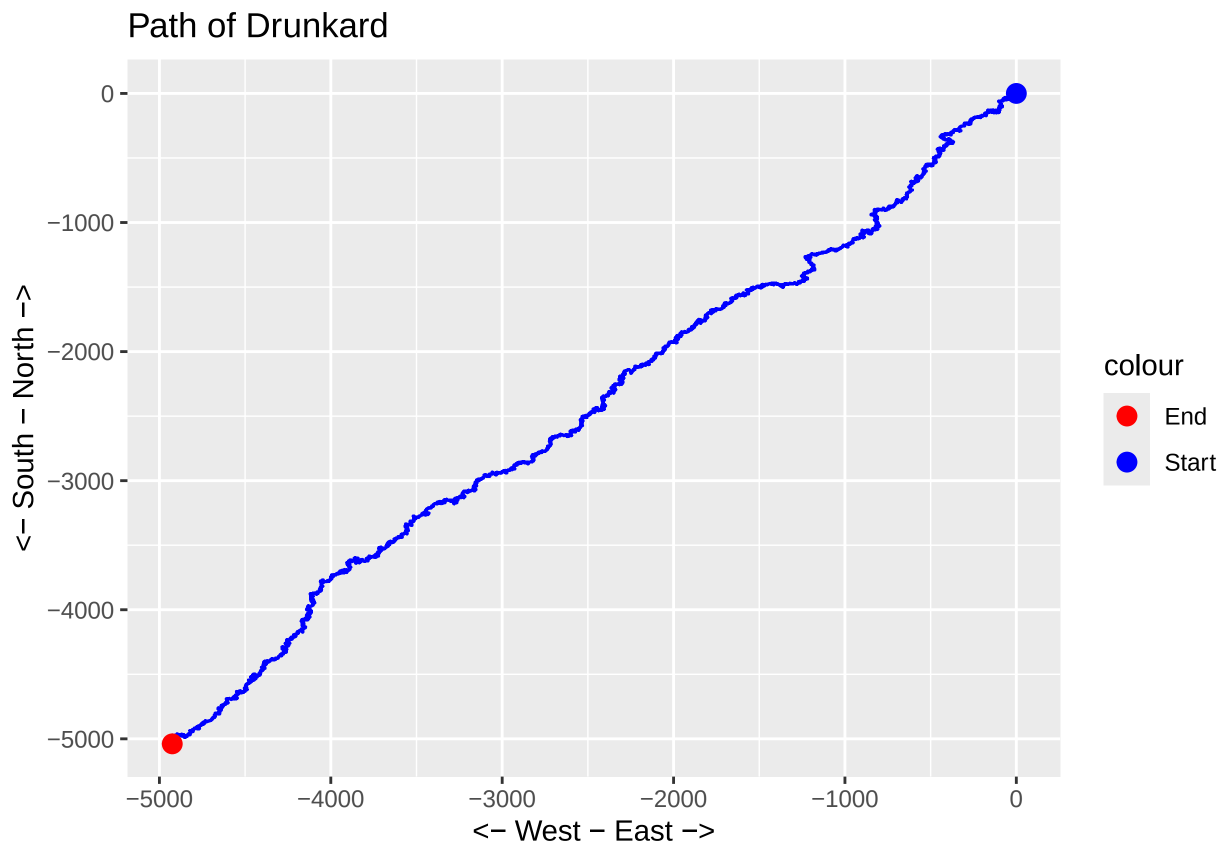
```
positions_df <- data.frame(Xcoord = positions[, 1], Ycoord = positions[, 2])

# use ggplot

ggplot(positions_df, aes(x = Xcoord, y = Ycoord)) +
  geom_path(color = "blue") +
  geom_point(data = positions_df[1, , drop = FALSE], aes(color = "Start"), size = 3) +
  geom_point(data = tail(positions_df, n = 1), aes(color = "End"), size = 3) +
  scale_color_manual(values = c("red", "blue")) +
  labs(x = "<- West - East ->", y = "<- South - North ->", title = "Path of Drunkard")
```



If you add up the probabilities, the walker has a 30% chance of going North, 35% South, 35% East, and 40% West (The probabilities do not add up to 100% as several of the outcomes are possible ex. Northeast will move the drunkard both North AND East). From this, over a longer period of time with more steps, the drunkard should move Southwest, with a slope of approximately 1.

Same simulation but for 100000 steps. You will notice the clear Southwest line and the slope of 1:

```r
num_steps <- 100001

positions <- matrix(0, nrow = num_steps, ncol = 2)


positions[1, ] <- c(0, 0) # start position

for (i in 2:(num_steps)) {
  step <- simulate_step()
  positions[i, ] <- positions[i - 1, ] + step
  # update step positions
}

positions_df <- data.frame(Xcoord = positions[, 1], Ycoord = positions[, 2])

# use ggplot

ggplot(positions_df, aes(x = Xcoord, y = Ycoord)) +
  geom_path(color = "blue") +
  geom_point(data = positions_df[1, , drop = FALSE], aes(color = "Start"), size = 3) +
  geom_point(data = tail(positions_df, n = 1), aes(color = "End"), size = 3) +
  scale_color_manual(values = c("red", "blue")) +
  labs(x = "<- West - East ->", y = "<- South - North ->", title = "Path of Drunkard")
```



Consider a simple stock price model with two states: up and down. The transition probabilities are: If the stock is up today, there's a 60% chance it will be up tomorrow and a 40% chance it will be down. If the stock is down today, there's a 50% chance it will be up tomorrow and a 50% chance it will be down. Here is a simulation of this Markov Chain to predict the stock price movement over a certain number of days:

```
multiply_matrix_n_times <- function(matrix, n) { # create a simple function to multiply a
↪  matrix by itself n times.
  result <- matrix

  for (i in 2:n) {
    result <- result %*% matrix
  }

  return(result)
}


simulate_stock_price <- function(initial_state, num_days) {

  transition_matrix <- matrix(c(0.6,0.5,0.4,0.5),nrow = 2) # make a 2x2 transition matrix
  simulate_n_times_matrix <- multiply_matrix_n_times(transition_matrix,num_days)
  # now multiply the initial vector

  state_after_n_days <- initial_state %*% simulate_n_times_matrix

  return(state_after_n_days)
}

# testing

initial_vector <- c(1,0)
# note here that the first item of the vector is the probability of
#the stock being up, and the second is the probability of the stock being down.

result <- simulate_stock_price(initial_vector,10) # after 10 days probabilities

print("Percent chance the stock will go up")
```

```
## [1] "Percent chance the stock will go up"
```

```
print(result[1])
```

```
## [1] 0.5555556
```

```
print("Percent chance the stock will go down")
```

```
## [1] "Percent chance the stock will go down"
```

```
print(result[2])
```

```
## [1] 0.4444444
```

As expected, the percent chance the stock will go up/down converged.