Homework 5

Daniel Lee

November 4, 2016

1. Random Walk in one dimension

```
# Description: counts the number of steps required to go to certain location on x-axis
# Input
# x: numeric value
# Output
# number_of_steps_required: numeric value
count_steps <- function(x){
stopifnot(x %% 1 == 0)
  number_steps_required <- 0
  positions <- 0
  #consider positive final positions
  if(x > 0){
  while(positions < x){
  left_or_right <- c(-1, 1)
  move <- sample(left_or_right, 1, replace = TRUE, prob = c(0.5, 0.5))
  positions <- positions + move
  number_steps_required <- number_steps_required +1
  }
  print(number_steps_required)
  } else{
  #consider the non-positive final positions
  while(positions > x){
  left_or_right <- c(-1, 1)
  move <- sample(left_or_right, 1, replace = TRUE, prob = c(0.5, 0.5))
  positions <- positions + move
  number_steps_required <- number_steps_required +1
  }
  print(number_steps_required)
  }
}

count_steps(100)
```

```
## [1] 461242
```

```
count_steps(-4)
```

```
## [1] 24
```

2. Area of pi

```
# Description: calculates part of the area of the upper half of a unit circle
# spanned by -x and x
# Input
```

```r
# x: x-coordinate (numeric value)
# Output
# Area of the covered region (numeric value)
f1 <- function(x) sqrt(1-x^2)

# Description: approximates the area under the curve of f and above axis x
# between x = a and x = b
# Input
# f: function (numeric value)
# a: the lower region (numeric value)
# b: the upper region (numeric value)
# n: number of intervals used to approximate the area (numeric value)
# m: maximum y value (numeric value)
# Output
# area: the approximated area under the curve f (numeric value)
mc_area <- function(f, a, b, n, m) {

  below <- 0
  x <- runif(n, min = a, max = b)
  y <- runif(n, min = 0, max = m)
  below <- sum(f1(x) > y)
  area <- (b - a) * m * (below / n)
  area

}

#approximate area of a half unit circle
half_circle <- mc_area(f1, -1, 1, 5000, 1)
half_circle
```

```
## [1] 1.5728
```

```r
# approximate area of a the entire unit circle
estimate_pi <- half_circle*2
estimate_pi
```

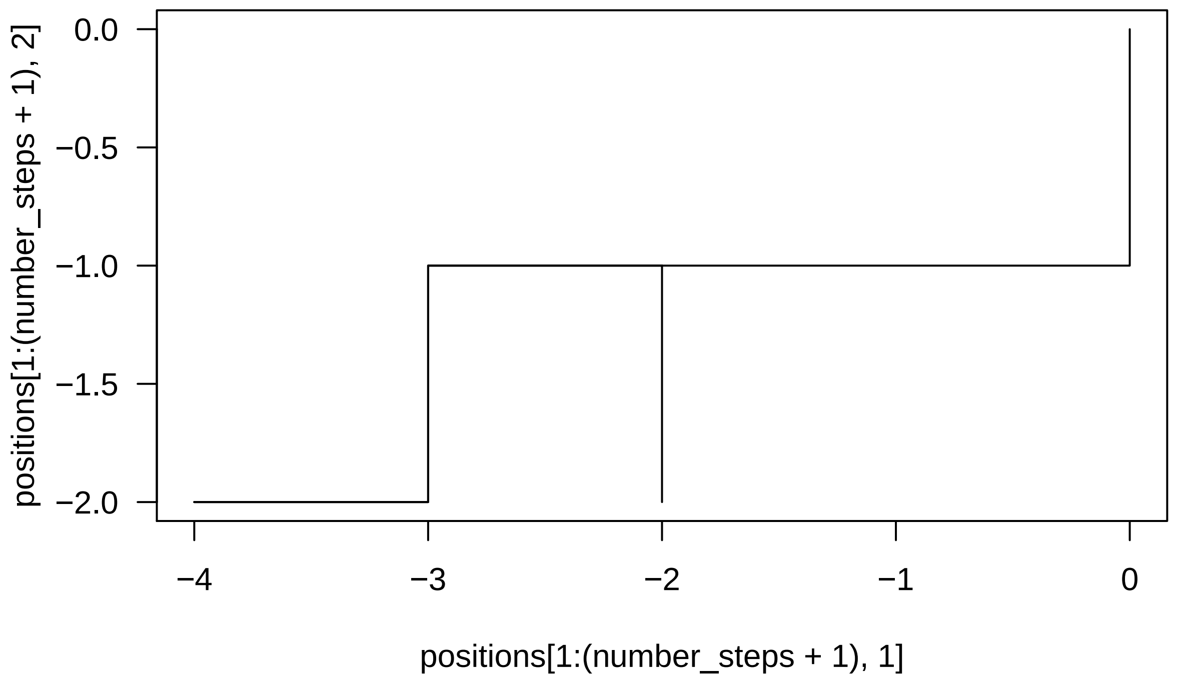
```
## [1] 3.1456
```

```r
# the difference between the approximated area of a unit circle
# and the value of pi in R
estimate_pi - pi
```

```
## [1] 0.004007346
```

3. Random Walk in two space dimensions

```r
# Description: generates a random walk for one particle in two dimensions
# Input
# number_steps: number of steps to be moved (numeric value)
# full.path: Boolean
# Output
# if full.path = TRUE, returns a matrix of all the positions
# if full.path = TRUE, creates a plot of all the random walks
# if full.path = FALSE, returns only the final position
random_walk <- function(number_steps, full.path = TRUE){

  #check conditions
  if ((number_steps < 0) | (number_steps %% 1 != 0)) {

    stop("\n argument 'number_steps' must be a non-negative integer")

  } else if (class(full.path) != "logical") {

    stop("\n argument 'full.path' must be a logical")

  }

  # define directions, down = -2, up = 2, left = -1, right = 1
  up_dn_lf_ri <- c(-2, -1, 1, 2)
  # start in position (0,0)
  positions <- matrix(rep(0, number_steps*2+2), nrow = number_steps+1, ncol = 2)

  moves <- sample(x = up_dn_lf_ri, number_steps, replace = TRUE, prob = c(0.25, 0.25 ,0.25 ,0.25))

  vertical_moves <- horizontal_moves <- moves

  vertical_moves[which(abs(moves) == 1)] <- 0
  horizontal_moves[which(abs(moves) == 2)] <- 0

  #calculates cumulative sum of horizontal moves and vertical moves
  positions[1:(number_steps+1), 1] <- cumsum(c(0, horizontal_moves))
  positions[1:(number_steps+1), 2] <- cumsum(c(0, vertical_moves/2))

  if(full.path == TRUE){

    # plot final position after number_steps steps
    plot(positions[1:(number_steps+1),1], positions[1:(number_steps+1),2], las = 1, type = "s")

    return(positions)

    } else{

      return(positions[number_steps+1,])

    }
}

#test run
random_walk(10)
```



```
##       [,1] [,2]
##  [1,]    0    0
##  [2,]    0   -1
##  [3,]   -1   -1
##  [4,]   -2   -1
##  [5,]   -3   -1
##  [6,]   -3   -2
##  [7,]   -4   -2
##  [8,]   -3   -2
##  [9,]   -3   -1
## [10,]   -2   -1
## [11,]   -2   -2
```

4. Object Oriented Programming

```
# Description: generates random walks in the vertical and horizontal directions
# Input
# num_steps: number of steps to be moved (numeric value)
# Output
# moves: vector of all the moves up, down, left, and right
# -2, -1, 1, 2 indicate down, left, right, up
move.randwalk <- function(num_steps){

  up_dn_lf_ri <- c(-2, -1, 1, 2)
  moves <- sample(up_dn_lf_ri, num_steps, replace = TRUE, prob = c(0.25, 0.25 ,0.25 ,0.25))
  return(moves)

}

# Description: Checks to see if num_steps is the right value
# Input
```

```r
# num_steps: number of steps to be moved (numeric value)
# Output
# TRUE or an error message
check_steps <- function(num_steps){

  if (num_steps < 0 | num_steps %% 1 != 0) {

    stop("\n argument 'num_steps' must be a non-negative integer")

  } else {

    TRUE
  }
}

# Description: Checks to see if start_position is the right value
# Input
# start_position: location to start the random walk vector of two numbers
# Output
# TRUE or an error message
check_start_position <- function(start_position){

  if (class(start_position) != "numeric") {

    stop("\n argument 'start_position' must be of class 'numeric'")

  } else if (length(start_position) != 2) {

    stop("\n argument 'start_position' must be a vector of length 2")

  } else if (start_position[1] %% 1 != 0 | start_position[2] %% 1 != 0) {

    stop("\n x and y values of argument 'start_position' must be an integer")

  } else {

    TRUE

  }
}

# Description: Generates a matrix of all the movements in random walk
# Input
# moves: vector of moves (left = -1, right = 1, up = 2, down = -2)
# start_position: location to start the random walk vector of two numbers
# Output
# position: matrix of all the steps taken (two columns)
all_positions <- function(moves, start_position) {

  x <- length(moves)

  vertical_moves <- horizontal_moves <- moves

  vertical_moves[which(abs(moves) == 1)] <- 0
  horizontal_moves[which(abs(moves) == 2)] <- 0

  positions <- matrix(rep(0, x*2+2), nrow = x+1, ncol = 2)

  positions[1:(x+1), 1] <- cumsum(c(start_position[1], horizontal_moves))
  positions[1:(x+1), 2] <- cumsum(c(start_position[2], vertical_moves/2))

  return(positions)

}

# Description: Returns the final position of the random walk
# Input
# positions: matrix of all the positions
# Output
# Last row of the positions matrix (vector of length 2)
final_position <- function(positions) {

  return(positions[nrow(positions),])

}

# Description: Constructor function that create an object of class randwalk
# Input
# num_steps: numeric value
# moves: vector of random movements left = -1, right = 1, up = 2, down = -2
# start_position: numeric vector of length 2
# Output
# res: list containing the attributs and outputs of randwalk object
make_randwalk <- function(num_steps, moves, start_position){

 res <- list(
    num_steps = num_steps,
    start_position = start_position,
    position_matrix = all_positions(moves, start_position),
    final_position = final_position(all_positions(moves, start_position)),
    total_right = sum(moves == 1),
    total_left = sum(moves == -1),
    total_up = sum(moves == 2),
    total_down = sum(moves == -2))
  class(res) <- "randwalk"
  res

}

# Description: function that the user uses to create the object randwalk
# Input
# num_steps: numeric value
# start_position: numeric vector of length 2 (default value = c(0, 0))
# Output
# list from constructor function make_randwalk containing
# the attributs and outputs of randwalk object
```

```r
randwalk <- function(num_steps, start_position = c(0, 0)){

  check_steps(num_steps)
  check_start_position(start_position)
  moves <- move.randwalk(num_steps)
  make_randwalk(num_steps, moves, start_position)

}

# Description: prints summary information for 'randwalk' object
# Input
# x: object of class 'randwalk'
# Output
# the summary of randwalk object
print.randwalk <- function(x) {

  cat('object "randwalk" \n')
  cat('start position:', x$start_position, "\n")
  cat('final position:', x$final_position, "\n")
  cat('total steps:', x$num_steps, "\n")
  cat('total steps to the right:', x$total_right, "\n")
  cat('total steps to the left:', x$total_left, "\n")
  cat('total steps up:', x$total_up,"\n")
  cat('total steps down:', x$total_down,"\n")
  invisible(x)

}

# Description: plots the entire random walk path in 2D
# Input
# x: object of class 'randwalk'
# ...: can specify parameters of a plot
# Output
# the plot of randwalk object
plot.randwalk <- function(x, ...){

    plot(x$position_matrix[1:(x$num_steps+1),1], x$position_matrix[1:(x$num_steps+1),2],
         las = 1, type = "s")

}

# Description: gives the position of the ith walk
# Input
# x: object of class 'randwalk'
# i: numeric value
# Output
# the ith position of the random walk
`[.randwalk` <- function(x, i){

  x$position_matrix[i, ]

}
```

```r
# Description: gives the position of the ith walk
# Input
# x: object of class 'randwalk'
# value: numeric vector that indicates the new starting location
# Output
# x: updated object of class 'randwalk' with new starting position
"start.randwalk<-" <- function(x, value){

  x <- randwalk(x$num_steps, value)
  return(x)

}

#test codes:
a <- randwalk(10)
a
```

```
## object "randwalk"
## start position: 0 0
## final position: 0 2
## total steps: 10
## total steps to the right: 1
## total steps to the left: 1
## total steps up: 5
## total steps down: 3
```

```r
a$position_matrix
```

```
##       [,1] [,2]
##  [1,]    0    0
##  [2,]    0    1
##  [3,]    0    2
##  [4,]    0    1
##  [5,]   -1    1
##  [6,]   -1    0
##  [7,]   -1    1
##  [8,]    0    1
##  [9,]    0    2
## [10,]    0    1
## [11,]    0    2
```

```r
a[4]
```

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

```r
#change starting location of a
start.randwalk(a) <- c(10, 20)
a
```

```
## object "randwalk"
## start position: 10 20
## final position: 12 16
## total steps: 10
## total steps to the right: 3
## total steps to the left: 1
## total steps up: 1
## total steps down: 5
```

```
a$position_matrix
```

```
##       [,1] [,2]
##  [1,]   10   20
##  [2,]   10   19
##  [3,]   10   20
##  [4,]   10   19
##  [5,]    9   19
##  [6,]    9   18
##  [7,]    9   17
##  [8,]   10   17
##  [9,]   10   16
## [10,]   11   16
## [11,]   12   16
```