Week 3: Simulation, Loops, and if/else Statements

January 25, 2018

Simulation in R

What is Simulation

A few questions about simulation:

1. What does statistical simulation mean to you?
2. Describe a setting where simulation can be used.

Simulation of Roulette

Consider the casino game Roulette.

Figure 1: Roulette Wheel

We can use simulation to evaluate gambling strategies.

Roulette Simulation in R


```
RouletteSpin <- function(num.spins){
  # function to simulate roulette spins
  # ARGS: number of spins
  # RETURNS: result of each spin
  outcomes <- data.frame(number = c('00','0',
                as.character(1:36)),
                color=c('green','green','red','black','red','black',
                        'red','black','red','black','red','black',
                        'black','red','black','red','black',
                        'red','black','red','red','black',
                        'red','black','red','black','red',
                        'black','red','black','black','red',
                        'black','red','black','red','black','red'))
  return(outcomes[sample(38,num.spins,replace=T),])
}
kable(RouletteSpin(2), row.names=F)
```

number	color
15	black
11	black

Exercise: Probability of Red, Green, and Black

1. Calculate/Derive the probability of landing on green, red, and black.
2. How can the `RouletteSpin()` function be used to compute or approximate these probabilities?

Solution: Probability of Red, Green, and Black

In this situation, it is easy to compute the probabilities of each color analytically. However, consider simulating this process many times to estimate these probabilities.

```
num.sims <- 1000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims
```

Analytically $P[red] = \frac{18}{38} = 0.4737$, this is estimated as 0.446.
Similarly, $P[black] = \frac{18}{38} = 0.4737$, this is estimated as 0.506
and $P[green] = \frac{2}{38} = 0.0526$, this is estimated as 0.048

Exercise: Simulation Questions - Part 2

Now what happens if we:

1. run the simulation again with the same number of trials?
2. run the simulation with more trials, say 1 million?

Solution: Simulation Questions - Part 2

Now what happens if we:

1. run the simulation again with the same number of trials?

```
num.sims <- 1000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims
```

The simulated results are different Analytically $P[red] = \frac{18}{38} = 0.4737$, this is estimated as 0.476. Similarly, $P[black] = \frac{18}{38} = 0.4737$, this is estimated as 0.483 and $P[green] = \frac{2}{38} = 0.0526$, this is estimated as 0.041

Solution: Simulation Questions - Part 2

2. run the simulation with more trials, say 1 million?

```
num.sims <- 1000000
spins <- RouletteSpin(num.sims)
p.red <- sum(spins[,2] == 'red') / num.sims
p.black <- sum(spins[,2] == 'black') / num.sims
p.green <- sum(spins[,2] == 'green') / num.sims
```

Analytically $P[red] = \frac{18}{38} = 0.4737$, this is estimated as 0.474.
Similarly, $P[black] = \frac{18}{38} = 0.4737$, this is estimated as 0.4733
and $P[green] = \frac{2}{38} = 0.0526$, this is estimated as 0.0527

Conditional Expressions in R

Exercise: Conditions in R

We have touched on many of these before, but here are some examples of expressions (conditions) in R. Evaluate these expressions:

```
pi > 3 & pi < 3.5
c(1,3,5,7) %in% 1:3
1:3 %in% c(1,3,5,7)
rand.uniform <- runif(n = 1, min = 0, max = 1)
rand.uniform < .5
```

Solutions: Conditions in R: Evaluated


```
pi > 3 & pi < 3.5
```

```
## [1] TRUE
```

```
c(1,3,5,7) %in% 1:3
```

```
## [1]  TRUE  TRUE FALSE FALSE
```

```
1:3 %in% c(1,3,5,7)
```

```
## [1]  TRUE FALSE  TRUE
```

```
rand.uniform <- runif(n = 1, min = 0, max = 1); rand.uniform
```

```
## [1] 0.659815
```

```
rand.uniform < .5
```

```
## [1] FALSE
```

Conditional Expression: If and Else


```
print(rand.uniform)
```

```
## [1] 0.659815
```

Now what does this return?

```
if (rand.uniform < .5){
  print('value less than 1/2')
} else {
    print('value greater than or equal to 1/2')
}
```

Conditional Expression: If and Else


```
print(rand.uniform)
```

```
## [1] 0.659815
```

```
if (rand.uniform < .5){
  print('value less than 1/2')
} else {
    print('value greater than or equal to 1/2')
}
```

```
## [1] "value greater than or equal to 1/2"
```

Conditional Expression: Vectorized?

Does this function accept a vector as an input?

```
rand.uniform2 <- runif(2)
print(rand.uniform2)
if (rand.uniform2 < .5){
  print('value less than 1/2')
} else {
    print('value greater than or equal to 1/2')
}
```

Conditional Expression: Vectorized?


```
rand.uniform2 <- runif(2)
print(rand.uniform2)
```

```
## [1] 0.6225445 0.9048026
```

```
if (rand.uniform2 < .5){
  'value less than 1/2'
} else {
    'value greater than 1/2'
}
```

```
## Warning in if (rand.uniform2 < 0.5) {: the condition has lengt
## only the first element will be used
```

```
## [1] "value greater than 1/2"
```

Conditional Expression, ifelse


```
print(rand.uniform2)
```

```
## [1] 0.6225445 0.9048026
```

```
ifelse(rand.uniform2 < .5,'less than 1/2',
       'greater than 1/2')
```

```
## [1] "greater than 1/2" "greater than 1/2"
```

The `ifelse()` function is vectorized and generally preferred with a single set of if/else statements.

Exercise: Conditional Expression

Write a conditional statement that takes a playing card with two arguments:

- (`card.number <- '4'` and `card.suit <- 'C'` = 4 of clubs or `card.number <-'K'` and `card.suit <- 'H'` = King of hearts) and
- Print `Yes` if the card is a red face card and `No` otherwise.

Verify this works using the following inputs:

- `card.number <- 'J'` and `card.suit <- 'D'`
- `card.number <- 'Q'` and `card.suit <- 'S'`

Solution: Conditional Expression


```
card.number <- 'J'
card.suit <- 'D'
ifelse(card.number %in% c('J','Q','K') &
         card.suit %in% c('H','D'),'Yes','No')
```

```
## [1] "Yes"
```

```
card.number <- 'Q'
card.suit <- 'S'
ifelse(card.number %in% c('J','Q','K') &
         card.suit %in% c('H','D'),'Yes','No')
```

```
## [1] "No"
```

Loops

FOR LOOPS

We have seen a few instances of for loops in this class, now consider these two statements.

```
rand.uniform2
```

```
## [1] 0.6225445 0.9048026
```

```
for (i in 1:length(rand.uniform2)){
  print(i)
}
```

and

```
for (i in rand.uniform2){
  print(i)
}
```

- what will each of these print?

FOR LOOPS

We can loop through a sequence or a vector.

```
for (i in 1:length(rand.uniform2)){
  print(i)
}
```

```
## [1] 1
## [1] 2
```

```
for (i in rand.uniform2){
  print(i)
}
```

```
## [1] 0.6225445
## [1] 0.9048026
```

While loops

An alternative to for loops is to use the while statement.

```
set.seed(02012017)
total.snow <- 0
while (total.snow < 36){
  print(paste('need more snow, only have',
              total.snow, 'inches'))
  total.snow <- total.snow + rpois(1,15)
}
```

```
## [1] "need more snow, only have 0 inches"
## [1] "need more snow, only have 18 inches"
## [1] "need more snow, only have 31 inches"
```

```
  print(paste('okay, we now have', total.snow, 'inches'))
```

```
## [1] "okay, we now have 48 inches"
```

Repeat Loops

Repeat loops are similar to while loops, but with the requirement that the expressions are evaluated at least once and require an explicit `break` statement.

```
repeat{
  total.snow <- total.snow + rpois(1,15)
  print(paste('we have', total.snow, 'inches'))
  if (total.snow >= 36)
    break
}
```

```
## [1] "we have 65 inches"
```

Exercise: Loops

Assume you plan to wager $1 on red for ten roulette spins. If the outcome is red you win a dollar and otherwise you lose a dollar. Write a loop that simulates ten rolls and determines your net profit or loss.

```
#hint: to get color from a single spin use
RouletteSpin(1)[2]
```

```
##    color
## 10 black
```

Solution: Loops

Assume you plan to wager $1 on red for ten roulette spins. If the outcome is red you win a dollar and otherwise you lose a dollar. Write a loop that simulates ten rolls and determines your net profit or loss.

```
#hint: to get color use
profit <- 0
for (i in 1:10){
  ifelse(RouletteSpin(1)[2] == 'red',
         profit <- profit + 1,
         profit <- profit - 1 )
}
profit
```

```
## [1] -2
```

Why not loops?

Some of you have seen or heard that loops in R should be avoided.

- **Why**: it has to do with how code is compiled in R. In simple terms, vectorized operations are much more efficient than loops.
- **How**: we have seen some solutions to this, explicitly using the apply class of functions. We can also write vectorized functions, consider the earlier roulette example where number of spins was an argument for the function.

Monte Carlo Procedures

Motivation for Monte Carlo procedures

Some probabilities can easily be calculated either intuitively or using pen and paper; however, as we have seen we can also simulate procedures to get an approximate answer.

Consider poker, where players are dealt a hand of five cards from a deck of 52 cards. What is the probability of being dealt a full house?

Full House Probability Calculation

Could we analytically compute this probability? **Yes** Is it an easy calculation? not necessarily. So consider a (Monte Carlo) simulation.

```
DealPoker <- function(){
  # Function returns a hand of 5 cards




}
```

Full House Probability Calculation

Could we analytically compute this probability? **Yes** Is it an easy calculation, not necessarily. So consider a (Monte Carlo) simulation.

```
DealPoker <- function(){
  # Function returns a hand of 5 cards
  deck <- data.frame( suit = rep(c("H", "S","C","D"),each=13),
          card = rep(c('A',2:10,'J',"Q",'K'),4) )
  return(deck[sample(52,5),])
}
```

Full House Probability Calculation

Next write another function to check if the hand is a full house.

```
IsFullHouse <- function(hand){
  #determines whether a hand of 5 cards is a full house
  #ARGS: data frame of 5 cards
  #RETURNS: TRUE or FALSE
  cards <- hand[,2]
  num.unique <- length(unique(cards))
  ifelse(num.unique == 2, return(TRUE),return(FALSE))
}
```

Does this work? If not, what is the problem and how do we fix it?

Full House Probability Calculation


```
IsFullHouse <- function(hand){
  #determines whether a hand of 5 cards is a full house
  #ARGS: data frame of 5 cards
  #RETURNS: TRUE or FALSE
  cards <- hand[,2]
  num.unique <- length(unique(cards))
  num.appearances <- aggregate(rep(1,5),
                               list(cards),sum)
  max.appearance <- max(num.appearances[,2])
  ifelse(num.unique == 2 & max.appearance ==3,
         return(TRUE),return(FALSE))
}
```

Full House Probability Calculation


```
num.sims <- 1e5
sim.hands <- replicate(num.sims,DealPoker()
                       ,simplify=F)
results <- lapply(sim.hands,IsFullHouse)
prob.full.house <- sum(unlist(results))/num.sims
```

Analytically the probability of getting a full house *can* be calculated as approximately 0.00144, with our simulation procedure we get 0.00139.

Closing Thoughts on Monte Carlo

A few facts about Monte Carlo procedures:

- They return a random result due to randomness in the sampling procedures.
- The run-time (or number of iterations) is fixed and typically specified.
- Mathematically, Monte Carlo procedures are computing an integral or enumerating a sum.
- They take advantage of the law of large numbers
- They were given the code name Monte Carlo in reference to the Monte Carlo Casino in Monaco.