Intro to Simulation

Prof. Donges

9/30/2019

Using Monte Carlo simulation to estimate the probability of winning, the expected value, and variance in roulette.

Example: Bet on a single number in roulette (American wheel)

1. Estimate the probability of winning.

The first step is to start by setting the seed. This ensures you'll get the same random process every time you run the code/knit the document. I've somewhat arbitrarily chosen 35; you should choose your own, different number.

```
#seed is set!
set.seed(35)
```

You need to choose the number of iterations (i.e., the number of spins of the wheel), too. When you are first writing a simulation it's a good idea to choose a small number of iterations until you're sure the code is working correctly. Then, you can increase the number of iterations to a more reasonable number. When I first wrote this I iterated by hand (I set i=1 and ran the code, then I set i=2 and ran the code, then I set i=3 and rand the code). Then I ran it for 10 iterations, followed by 100, followed by 1000. Finally, I settled in on 60,000 iterations. Crawl, walk, then run!

```
#define and set the number of spins/iterations
n.sim <- 60000
```

I'm not sure this needs to be done, but I'm going to initialize some storage vectors. Taking the time to do this upfront guarantees these are available later when I need them.

```
#n.wins = the number of wins (this is used to count the number of wins)
n.wins <- 0

#p.hat = the running estimated prob of winning (that is, p.hat = n.wins/n.sim)
p.hat <- 0
```

You need to choose a number upon which to bet. I've arbitrarily chosen 10. If you pick anything other than 00, you can just set it as the number upon which you've bet. But, R treats 0 and 00 as 0. The reality is the number you pick does not matter since the distribution is the same. I did write some code below to handle the 00, however.

```
#choose a number upon which to bet
n.bet <- 10
```

I need to "build" my roulette wheel. I need 38 equally likely possibilities. I chose to use the numbers 1, 2, . . . , 38. You could just as easily use 0, 1, 2, . . . , 37. In essence, I'm mapping the numbers 1 through 36 on the wheel to the numbers 1 through 36 and I'm mapping 0 and 00 to 37 and 38, respectively (or vice versa). So, if the number upon which you bet is 0 or 00, I'm mapping it to 37 below; by default that maps the other zero to 38.

```
#the n.bet==0 is equal to a 1 in the mult if n.bet is 0 and is equal to 0 o.w.
#hence, the line below maps 0 to 37 and leaves all other numbers as entered
#and maps the other zero to 38
n.bet <- n.bet + 37*(n.bet==0)
```

Now, I construct the wheel.

```
#build the wheel
numbers <- c(1:38)
```

Again, I'm going to construct a storage vector for the winning numbers. I don't necessarily need to store the winning numbers, but I decided to do so in order to see if the code was working.

```
#create an empty vector to put the winning numbers in
#eventually, this will be the length of n.sim
win.numbers <- vector()
```

I'm going to write a loop that will run for 60,000 spins. Each time it will spin the wheel (sample from the numbers 1, 2, 3, . . . , 38 with equal probability - this is an example of the discrete uniform at work), store the number spun, determine whether or not a win took place, update the running tally of wins, and record the running estimated probability of a win.

```
#set up the loop
for (i in 1:n.sim){
  #i=3 <- this is a remnant from the aforementioned iteration by hand

  #generate a winning number for spin i by randomly choosing a number from 1,2, ..., 38
  #store the ith winning number in spot i in win.numbers vector
  win.numbers[i] <- sample(numbers,1)

  #if you won on spin i add 1 to n.wins, if not add 0
  n.wins <- n.wins + 1*(n.bet==win.numbers[i])

  #compute and store est'd win prob, phat, at iteration i
  p.hat[i] <- n.wins/i

  #close the loop!
}
```

The estimated probability of a win can be obtained by either extracting the last entry in the p.hat vector or by dividing the number of wins (n.wins) by the number of iterations/spins (n.sim).

```
#extract the final entry of p.hat
round(p.hat[n.sim], 4)
```

```
## [1] 0.0261
```

```
#divide the number of wins by the number of iterations
round(n.wins/n.sim, 4)
```

```
## [1] 0.0261
```

While the Monte Carlo method is very useful when a probability must be estimated/cannot be computed, in this case I actually know the true win probability. Here, we'll use this knowledge to compute the error and absolute error of the estimated win probability. When you're estimating a probability in lieu of computing it this is not possible.

```
#define the true win prob
true.win.prob <- 1/38

#compute the error in the estimate by subtracting the true value from the est.
error <- round(p.hat[n.sim] - (true.win.prob), 6)
error
```

```
## [1] -0.000182
```

```
#take the absolute value to the absolute error
abs.error <- abs(error)
abs.error
```

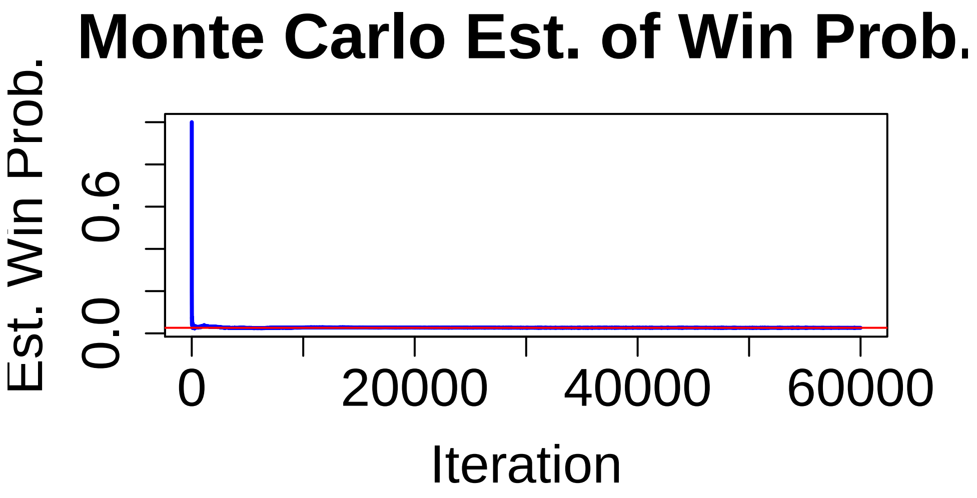
```
## [1] 0.000182
```

I can represent the process of convergence of the estimated probability to the true probability by constructing a line plot of the estimated win prob by iteration. I'll superimpose a horizontal line at the value of the true win probability.

```
#note that I'm controlling the size and location of the figure
par(ps=20)

#plot function says to plot the (x,y) with the x coming from 1:nsim by 1
#the y coming from the p.hat vector, type='l' says do a line plot
#lwd controls the width (higher means thicker), and I'm making it blue
plot(1:n.sim, p.hat, type='l',lwd=2, col='blue',xlab='Iteration',
     ylab='Est. Win Prob.',main='Monte Carlo Est. of Win Prob.')

#add a horizotal red line at 1/38, make it thinner than above
abline(h=true.win.prob, col='red', lwd=1)
```



2. Estimate the expected win/loss

Start by setting the seed to the same value as above; this ensures you'll get the same random sequence as above. Doing this here allows you to run just the code chunks which follow.

```
#seed is set!
set.seed(35)
```

Now, you should choose an amount to bet each time (same bet on each spin). To keep it realistic, choose a positive integer between $1 and $500.

```
#choose the amount; I'm calling it bet
bet <- 25
```

As above, choose the number of iterations.

```
#set the number of spins; this is redundant with the above code
#but it allows part 2 to be run independently of part 1
n.sim <- 60000
```

As above, I'm going to initialize some storage vectors. Taking the time to do this upfront guarantees these are available later when I need them.

```
#win = the amount won each spin
win <- 0

#win.total = running total of amount won
win.total <- 0

#avg.win = running avg of amount won
avg.win <- 0
```

You need to choose a number upon which to bet. I've arbitrarily chosen 10. If you pick anything other than 00, you can just set it as the number upon which you've bet. But, R treats 0 and 00 as 0. The reality is the number you pick does not matter since the distribution is the same. I did write some code below to handle the 00, however.

```
#choose a number upon which to bet
n.bet <- 10
```

I need to "build" my roulette wheel. I need 38 equally likely possibilities. I chose to use the numbers 1, 2, . . . , 38. You could just as easily use 0, 1, 2, . . . , 37. In essence, I'm mapping the numbers 1 through 36 on the wheel to the numbers 1 through 36 and I'm mapping 0 and 00 to 37 and 38, respectively (or vice versa). So, if the number upon which you bet is 0 or 00, I'm mapping it to 37 below; by default that maps the other zero to 38.

```
#the n.bet==0 is equal to a 1 in the mult if n.bet is 0 and is equal to 0 o.w.
#hence, the line below maps 0 to 37 and leaves all other numbers as entered
#and maps the other zero to 38
n.bet <- n.bet + 37*(n.bet==0)
```

Now, I construct the wheel.

```r
#build the wheel
numbers <- c(1:38)
```

Again, I'm going to construct a storage vector for the winning numbers. I don't necessarily need to store the winning numbers, but I decided to do so in order to see if the code was working.

```r
#create an empty vector to put the winning numbers in
#eventually, this will be the length of n.sim
win.numbers <- vector()
```

I'm going to write a loop that will run for 60,000 spins. Each time it will spin the wheel (sample from the numbers 1, 2, 3, . . . , 38 with equal probability - this is an example of the discrete uniform at work), store the number spun, determine whether or not a win took place, update the running tally of wins, and record the running estimated probability of a win.

```r
#set up the loop
for (i in 1:n.sim){
  #generate a winning number for spin i by randomly choosing a number from 1,2, ..., 38
  #store the ith winning number in spot i in win.numbers vector
  win.numbers[i] <- sample(numbers,1)

  #store win total add 35 times the bet amount to the win total
  #if not, subtract the bet amount
  win[i] <- 35*bet*(n.bet==win.numbers[i]) - bet*(n.bet!=win.numbers[i])

  #compute and store running win total
  win.total[i] <- sum(win)

  #compute and store avg running win total
  avg.win[i] <- sum(win)/i

  #close the loop!
}
```

The estimated expected value can be obtained by either extracting the last entry in the avg.win vector or by dividing the total amount won (last entry of win.total) by the number of iterations/spins (n.sim).

```r
#extract the final entry of avg.win
round(avg.win[n.sim], 2)
```

```
## [1] -1.48
```

```r
#divide the total amount won by the number of iterations
round(win.total[n.sim]/n.sim, 2)
```

```
## [1] -1.48
```

Just as in the case above where we estimated a known probability, here we know the true expected value $(35(bet)(\frac{1}{38}) - (bet)(\frac{37}{38}))$ and, as a result, can compute the error and absolute error of estimate. This is not always possible.

```r
#compute the true expected value
true.exp.value <- 35*bet*(1/38) - bet*(37/38)

#compute the error in the estimate by subtracting the true value from the est.
error.ev <- round(avg.win[n.sim] - (true.exp.value), 4)
error.ev
```

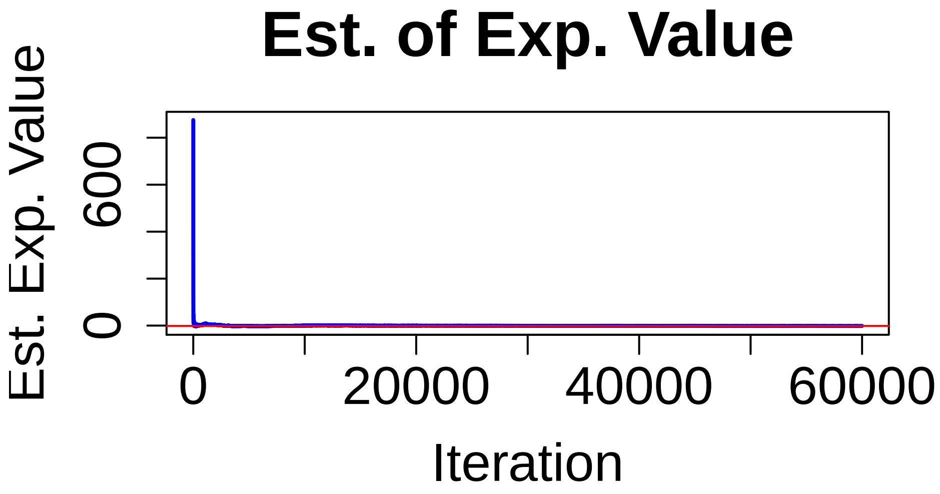
```
## [1] -0.1642
```

```r
#take the absolute value to get the absolute error
abs.error.ev <- abs(error.ev)
abs.error.ev
```

```
## [1] 0.1642
```

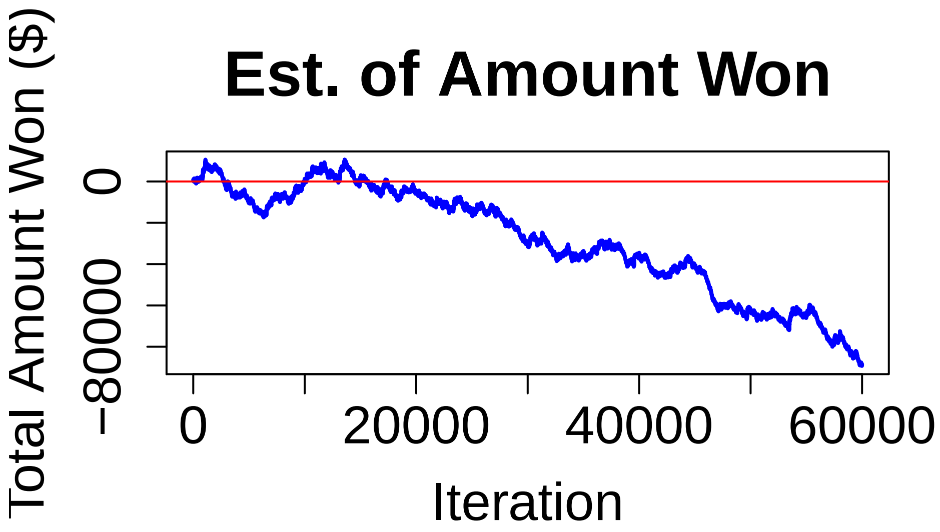
As above, I can represent the process of convergence of the estimated expected value to the true expected value by constructing a line plot of the estimated expected value by iteration. I'll superimpose a horizontal line at the value of the true expected value.

```r
#make a line plot of the est'd avg win by iteration
par(ps=20)
plot(1:n.sim, avg.win, type='l',lwd=2, col='blue',
     xlab='Iteration', ylab='Est. Exp. Value',main='Est. of Exp. Value')
#add a line at true exp value
abline(h=true.exp.value, col='red', lwd=1)
```



I can also do this for the total amount win. This shows the progression of your "win" over the 60,000 spins. I'm including a horizontal line at $0 to show the break even point.

```r
#make a line plot of the total amount won by iteration
par(ps=20)
plot(1:n.sim, win.total, type='l',lwd=2, col='blue',
     xlab='Iteration', ylab='Total Amount Won ($)',main='Est. of Amount Won')
abline(h=0, col='red', lwd=1)
```



3. Estimate the standard deviation

We can use the work above to quickly estimate the variance/standard deviation.

```
#compute the est'd expected value
est.ex.value <- 35*bet*p.hat[n.sim] - bet*(1-p.hat[n.sim])
est.ex.value
```

```
## [1] -1.48
```

```
#or
est.ex.value2 <- avg.win[n.sim]
est.ex.value2
```

```
## [1] -1.48
```

```
#compute the est'd variance

#first, estimate E(h(X)^2)
est.ex.value.sq <- ((35*bet)^2)*p.hat[n.sim] + (bet^2)*(1-p.hat[n.sim])
est.ex.value.sq
```

```
## [1] 20617
```

```
#now compute the est'd variance
est.var <- est.ex.value.sq - est.ex.value

#compute the est'd standard deviation
#don't round until the end!
est.sd <- round(sqrt(est.var), 2)
est.sd
```

```
## [1] 143.59
```