Simulations Pre-Class Project

Due March 13, 2017 at 5:00pm

# Project Goals:

With this project we will simulate a famoues probability problem. This will not require knowledge of probability or statistics but only the logic to follow the steps in order to simulate this problem. This is one way to solve problems by using the computer.

1. **Gambler's Ruin**: Suppose you have a bankroll of $1000 and make bets of $100 on a fair game. By simulating the outcome directly for at most 5000 iterations of the game (or hands), estimate:
   1. the probability that you have "busted" (lost all your money) by the time you have placed your one hundredth bet.

gamblers<-function(initial,bet){  
 counter=0  
a<-matrix(nrow=500,ncol=10)  
for ( i in 1:5000 ){  
 current<-initial  
 x<-rbinom(bet,1,prob=0.50) # let p be the probaility of loosing  
  
 for ( j in 1:bet){  
 if(x[j]==1 && current>0){ # a value of 1 means I won the bet  
 current<-current+(100)  
 }else if (x[j]==0 && current>0){  
 current<-current-(100)  
 }else if( current==0){  
 a[i]=1 # if you loose all your money it counts that   
 break  
 }   
 }  
}  
  
 answer<-sum(a,na.rm=TRUE)/5000  
 print(answer)  
}

b. the probability that you have busted by the time you have placed your five hundredth bet by simulating the outcome directly.

gamblers(1000,500)

## [1] 0.6618

c. the mean time you go bust, given that you go bust within the first 5000 hands.

counter=0  
initial<-1000 # inital amount of money you have  
  
a<-matrix(nrow=500,ncol=10)  
for (i in 1:5000){  
 current<-initial  
 x<-rbinom(5000,1,prob=0.50) # let p be the probaility of loosing  
 if(x[i]==1 && current>0){ # a value of 1 means I won the bet  
   
 current<-current+(100)  
 }else if (x[i]==0 && current>0){  
 current<-current-(100)  
 }else if( current==0){  
 a[i]=1 # if you loose all your money it counts that   
 break  
 }   
 }  
  
  
 answer<-sum(a,na.rm=TRUE)/5000 \*5000

d. the mean and variance of your bankroll after 100 hands (including busts).

counter=0  
current<-rep(0,100)  
initial<-1000 # inital amount of money you have  
  
  
 current[1]<-initial  
 x<-rbinom(100,1,prob=0.50) # let p be the probaility of loosing  
  
 for ( j in 2:100){  
 if(x[j]==1 && current[j-1]>0){ # a value of 1 means I won the bet  
 current[j]<-current[j-1]+(100)  
 }else if (x[j]==0 && current[j-1]>0){  
 current[j]<-current[j-1]-(100)  
 }else if( current[j-1]==0){  
 break  
 }   
 }  
  
  
 mean(current)

## [1] 181

var(current)

## [1] 104180.8

e. the mean and variance of your bankroll after 500 hands (including busts).

counter=0  
current<-rep(0,500)  
initial<-1000 # inital amount of money you have  
  
  
 current[1]<-initial  
 x<-rbinom(500,1,prob=0.50) # let p be the probaility of loosing  
  
 for ( j in 2:500){  
 if(x[j]==1 && current[j-1]>0){ # a value of 1 means I won the bet  
 current[j]<-current[j-1]+(100)  
 }else if (x[j]==0 && current[j-1]>0){  
 current[j]<-current[j-1]-(100)  
 }else if( current[j-1]==0){  
 break  
 }   
 }  
  
  
mean(current)

## [1] 529.2

var(current)

## [1] 461310

Note: you *must* stop playing if your player has gone bust. How will you handle this in the for loop?

1. **Markov Chains**. Suppose you have a game where the probability of winning on your first hand is 48%; each time you win, that probability goes up by one percentage point for the next game (to a maximum of 100%, where it must stay), and each time you lose, it goes back down to 48%. Assume you cannot go bust and that the size of your wager is a constant $100.

starting\_prob=0.48  
current\_prob=starting\_prob  
wager=100  
tally=0  
bankroll=0  
  
Markov<-function(wager,starting\_prob,percentage,iterations){  
for ( i in iterations){  
 if(rbinom(1,1,current\_prob)==1 && current\_prob<1){ # if you win and the current probability is less than 100  
 bankroll<-bankroll+wager  
 current\_prob<-current\_prob+percentage  
   
 if (rbinom(1,1,current\_prob)==1 && current\_prob==1){ # if you win and the current probability is already at 100  
 bankroll<-bankroll+wager  
 current\_prob=1  
 }   
   
 else{  
 bankroll<-bankroll-wager # if you loose you go back to original probability  
 current\_prob=starting\_prob}  
}  
}  
  
  
}

1. Is this a fair game? Simulate one hundred thousand sequential hands to determine the size of your return. Then repeat this simulation 99 more times to get a range of values to calculate the expectation

Markov(100,0.48,0.01,100000)  
simulated<-replicate(100,Markov(100,0.48,0.01,100000))  
avg<-mean(simulated)

## Warning in mean.default(simulated): argument is not numeric or logical:  
## returning NA

This is not a fair game because the expected value is negative.

b. Repeat this process but change the starting probability to a new value within 2% either way. Get the expected return after 100 repetitions. Keep exploring until you have a return value that is as fair as you can make it. Can you do this automatically?

for ( i in seq(0.45,0.51,0.01)){  
 process<-(c(i,mean(replicate(100,Markov(100,i,0.01,100000)))))  
}

## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, i, 0.01, 1e+05))):  
## argument is not numeric or logical: returning NA

If the probability is somewhat around p=0.489, then the replication can be more fair. c. Repeat again, keeping the initial probability at 48%, but this time change the probability increment to a value different from 1%. Get the expected return after 100 repetitions. Keep changing this value until you have a return value that is as fair as you can make it.

mean(replicate(100, Markov(100, .48, .02, 100000)))

## Warning in mean.default(replicate(100, Markov(100, 0.48, 0.02, 1e+05))):  
## argument is not numeric or logical: returning NA

## [1] NA

for ( i in seq (.01, .015, .001)){  
 process2<-(c( i, mean(replicate(100, Markov(100, .48, i, 100000)))))  
}

## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA

## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA  
  
## Warning in mean.default(replicate(100, Markov(100, 0.48, i, 1e+05))):  
## argument is not numeric or logical: returning NA

When we replicate , If I use an increment near 0.0123 I can get a somewhat fair return value