R Simulations: The Role of Probability in Casino Games

Stat133, Fall17

Arthur Huynh

The Secret of Casinos

Real people at a casino, totally not staged at all

Want to learn how to beat the casinos and rake in that cash money? Well you can't. Sucks bruh, I know. Gonna have to find some other way to make it rain on the weekends, cause *Random Number Simulations* say in the end, you're always going to take an L. Don't believe me? Think that lucky sock of yours helps you win all day every day? Let's take a look at the stats then.

Random Number Simulation Basics

Before I can even show you that super cool stuff that'll blow your mind away (who am I kidding, its stats lol) you gotta learn the basics of random simulations to even begin to appreciate the beauty that is squiggly graphs which show you a some overhyped trends that so-called "statician" people make inferences off of and make up some meaning out of wriggly lines. So, lets get to it!

First off, what exactly is a random number simulation? If you don't know or can't tell from the name, you need some books thrown in your face. Nothing personal, just a necessary for your own benefit. It's basically just a virtual simulation in which random numbers are generated, so nothing too difficult. Take this example. It's literally just a program that spits out a random number using the command "sample". Pretty much like a die.

Here's some examples and usages of random number commands in R.

```
sample(1:6,1)
```

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

We can do it with decimals too to get a random percentage using command "runif".

```
runif(1)
```

```
## [1] 0.1345675
```

Then we can take it to the next level and generate a whole bunch of random integers. In case you need a ton of random numbers for some odd reason. No worries, I don't judge.

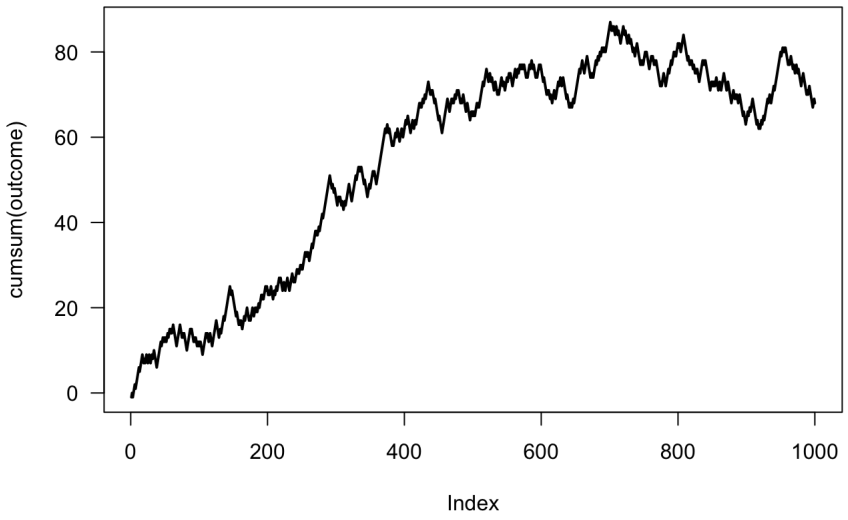
```
head(.Random.seed,10)
```

```
##  [1]         403           2 -451100276  760407759 1017838213  201494844
##  [7] -902022126 1422856973 1851318343 1217794606
```

I know, I know, all these numbers must be hard to look at. We need something more visual, more instant gratification. Like a graph! So let's apply

this in a game where you win if you roll at least a six in 4 dice rolls, and put the outcome on a graph and see what it looks like. You can throw say, oh, a 1000 times and see what happens (do not try at home, or you'll waste your life).

```
library('ggplot2')
set.seed(2)
games <- 1000
outcome <- rep(0, games)
die <- 1:6
roll <- function(x, times = 1) {
  sample(x, size = times, replace = TRUE)
}

for (g in 1:games) {
  rolls <- roll(die, times = 4)
  # count number of sixes
  sixes <- sum(rolls == 6)
  # win or lose?
  if (sixes >= 1) {
    outcome[g] <- 1  # win
  } else {
    outcome[g] <- -1  # lose
  }
}

plot(cumsum(outcome), type = 'l', lwd = 2, las = 1)
```



reference code: https://github.com/ucb-stat133/stat133-fall-2017/blob/master/tutorials/11-more-simulations.md

Congrats, in the end, the more you play, the more likely you are to win! But there's no prize! Suuucks!

So that's just one application of random number simulations; it can tell you a probable future if you're dealing in games of chance. It can tell you how likely you are to win. Cause gambling is a number's game after all. Luck is an illusion; probability is the master of this realm. There's more practical applications, don't worry, I'm not wasting your time (probably am since it's a graded assignment though, sorry).

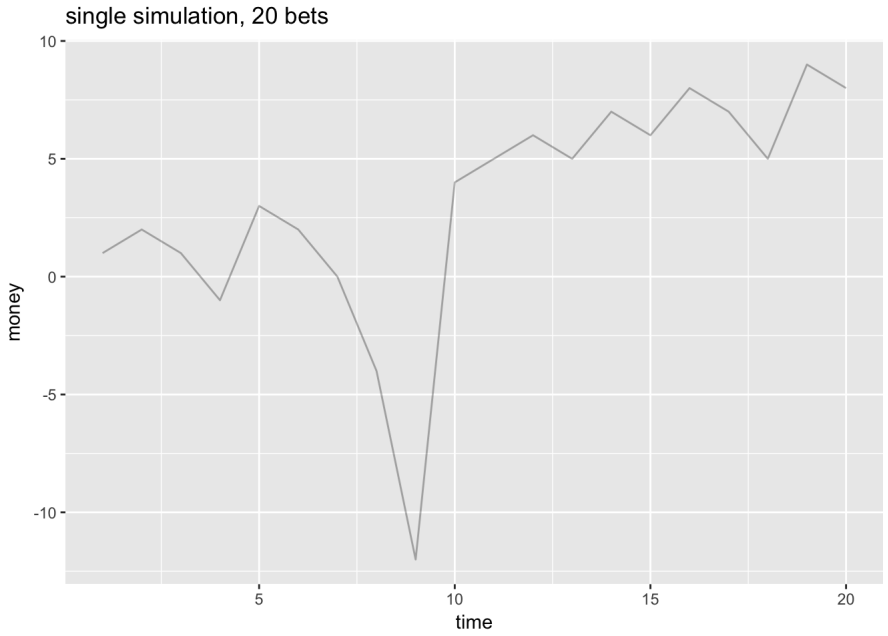
Casino Time

Ah yes, a picture of your wallet losing a ton of weight

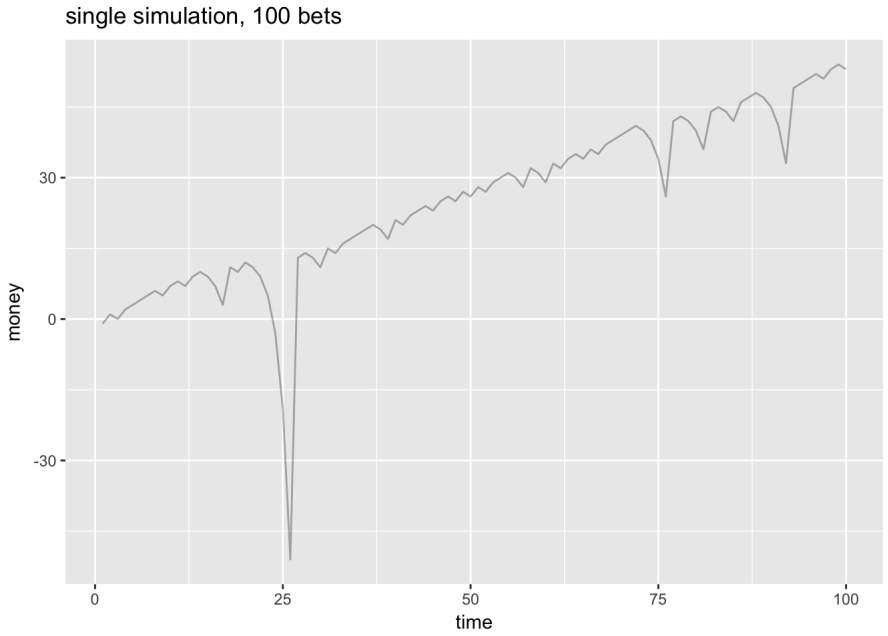
A more real life example of the use of these simulations is in a classic casino game: roulette. Not Russian roulette, stats won't help you when you're facing down the barrel of a .45, but the spinny red and black one. They say there's a flawless winning strategy to bring the house down. You bet double your last bet every round. This way, even if you have a series of losses, once you eventually get that one win, you'll make a profit. This is called the Martingale Betting System which you can find more about on this website. But this won't save your broke butt.

A simulation for such a system in roulette is simulated below in varying number of bets.

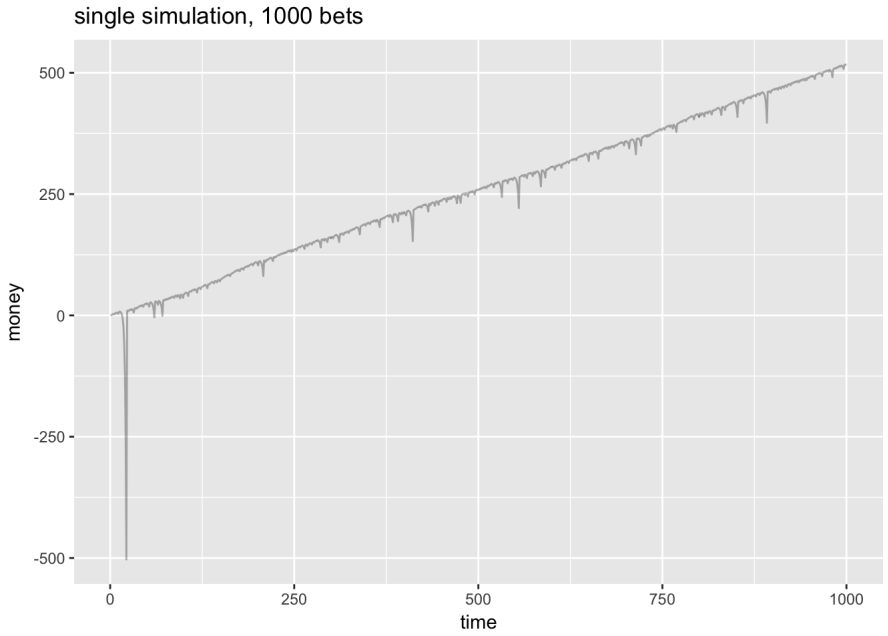
```
gamble = function(moneyin){
  if( runif(1) < 0.5 ) return(moneyin) # result is black
  -moneyin # result is red
}

nextm = function(gamble, outcome){
  if(outcome < 0) return(2*gamble) #on lose we double the money
  1
}

simulate = function(maxt){
  df = data.frame(time=as.numeric(c()), money=as.numeric(c()))
  move = 0
  outcome = 0
  for(i in 1:maxt){
    move = nextm(move, outcome)
    outcome = gamble(move)
    df = rbind(df, data.frame(time=i, money=outcome))
  }
  df$money = cumsum(df$money)
  df
}

gamblersruin = function(num, maxsim){
  p = ggplot()
  for(i in 1:num){
    df = simulate(maxsim)
    p = p + geom_line(data=df, aes(time,money), alpha=0.3)
  }
  p
}
set.seed(1)
gamblersruin(1,20) + ggtitle('single simulation, 20 bets')
```



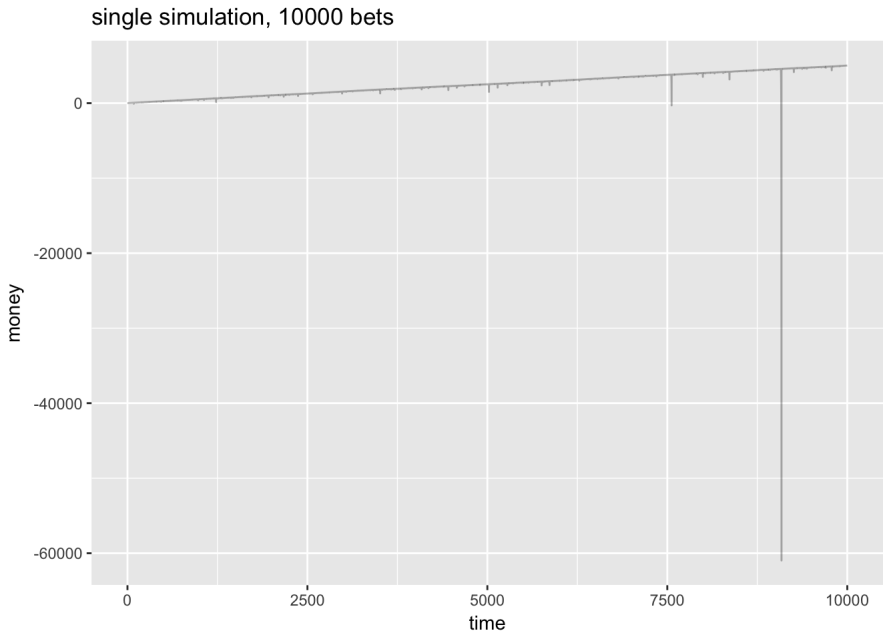

```
gamblersruin(1,100) + ggtitle('single simulation, 100 bets')
```




```
gamblersruin(1,1000) + ggtitle('single simulation, 1000 bets')
```



```
gamblersruin(1,10000) + ggtitle('single simulation, 10000 bets')
```



reference code: http://koaning.io/casino-gambling-simulations-in-r.html

As you can see, it looks like you're gonna make bank off casinos at first, but don't be fooled. Yes, there is a trend upwards, indicating you'll make that cash money in all simulations. And that's what casinos want you to think. But it is not without a huge risk. If you look closer, past your dreams of weekends swimming in a pool benjamins, you'll see that huge dip where you lose everything. And the longer you play, the more likely the event becomes. Because realistically, when you do eventually hit that dip, you're going to have to compensate with cold hard cash that you probably don't have. And that's when you have to sell the house and kids to pay off your debts. And even if you did have a ton of cash and sold those whiny brats, casino's tend to apply a maximum bet limit to gain even more of an advantage so that you can't double your bet forever.

And this doesn't even take into account the odds of the game itself which favors the house. In American roulette, the house edge is nearly 5.26% on all bets. Let's run a simulation to see if this is true.

```r
# Simulated random walks were drawn for red-black roulette.
# In each game of red-black roulette, the house has a 10/19 (=1-9/19) chance
# of winning. Simulated betting histories were drawn to illustrate the
# idea that whereas the proportion of games won approaches 9/19,
# the number of games won in a typical history when N games are played does not
# approach 9N/19. This is essentially a property of the
# binomial distribution.

# Red - Black Roulette
#   - Wheel has 38 slots: 18 black, 18 red, 2 green
#   - Bettor bets $1 on a color (red or black)
#   - If bettor's color comes up, bettor gains $1. Otherwise he loses his $1 bet.
#   - p(win) = 18/38 = 9/19
#   - game outcomes are independent if they are based upon different spins of the wheel
#   - If n is the number of games bet upon, and X = number of games won, X ~ binomial(n,18/38)
#   - Total earnings are X - (n-X) = 2X - n

# Recall: binomial distribution
#   - n 'trials' (trial is a game; n = number of trials = number of games)
#   - trials are independent
#   - each trial has two possible outcomes: success or failure (success is bettor wins)
#   - p = probability of success in a single trial
#    (p = probability bettor wins a single game = 18/38)
#   - X = number of successes in the n trials (= number of times bettor wins in the n games)

# E(X) = np, Var(X) = np(1-p)
# p^hat = X/n = proportion of successes in the n trials (estimator of p)
# E(p^hat) = p, Var(p^hat) = p(1-p)/n

pwin<-18/38

r<-runif(10)

# Bettor wins i-th game if r[i]<pwin. In this case earn[i]=1;
# otherwise earn[i]=-1. cum.earn[i] is the cumulative earnings through
# the ith game.

# if r < pwin, then earn = 1, otherwise earn = -1
# earn will be a data vector of length 10

earn<-ifelse(r<pwin,1,-1)

# find cumulative sum - to get cumulative earnings
# cum.earn is also a data vector tracking cumulative earnings after every game

cum.earn<-cumsum(earn)

# print out values
pwin
```

```
## [1] 0.4736842
```

```r
r
```

```
##  [1] 0.02934563 0.80655363 0.57308665 0.48257013 0.07266470 0.49797523
##  [7] 0.96907498 0.37165046 0.34280974 0.67186158
```

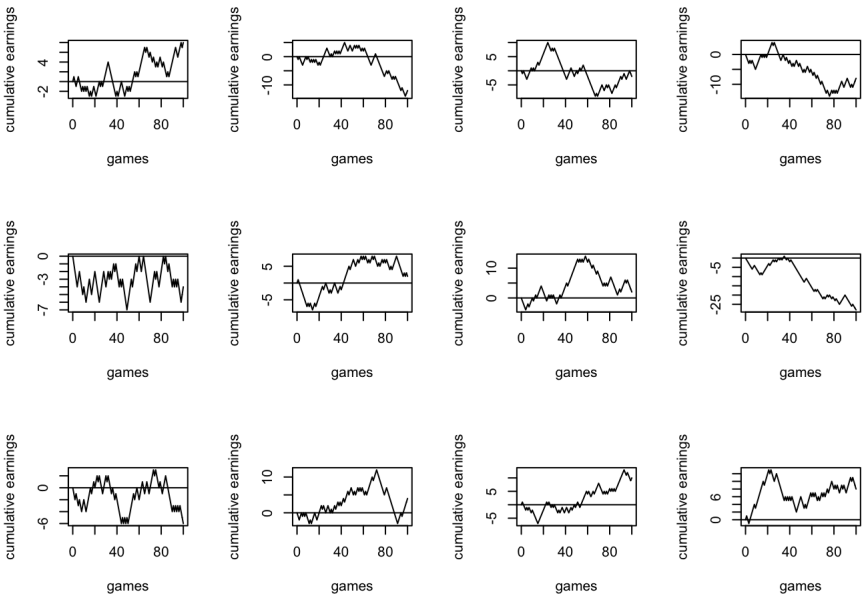
```r
earn
```

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

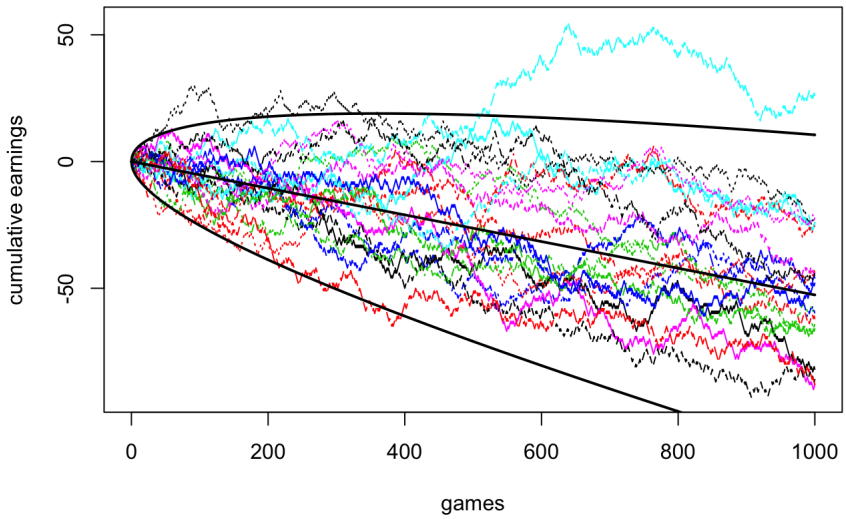
```r
cum.earn
```

```
##  [1]  1  0 -1 -2 -1 -2 -3 -2 -1 -2
```

```r
# start plot device with 3 rows and 4 columns

opar<-par(mfrow=c(3,4))

# We plot 12 simulated histories of 100 games each.
# page 86 for options for plot - type of "l" means draw lines
# each is a path of a random walk

for (i in 1:12) {
  r <- runif(100)
  earn<-ifelse(r<pwin,1,-1)
  cum.earn<-c(0,cumsum(earn))  # starts all plots at (0,0)
  plot(0:100,cum.earn,type="l",xlab="games",ylab="cumulative earnings")
  abline(h=0)  # draws a horizontal line at 0
}
```



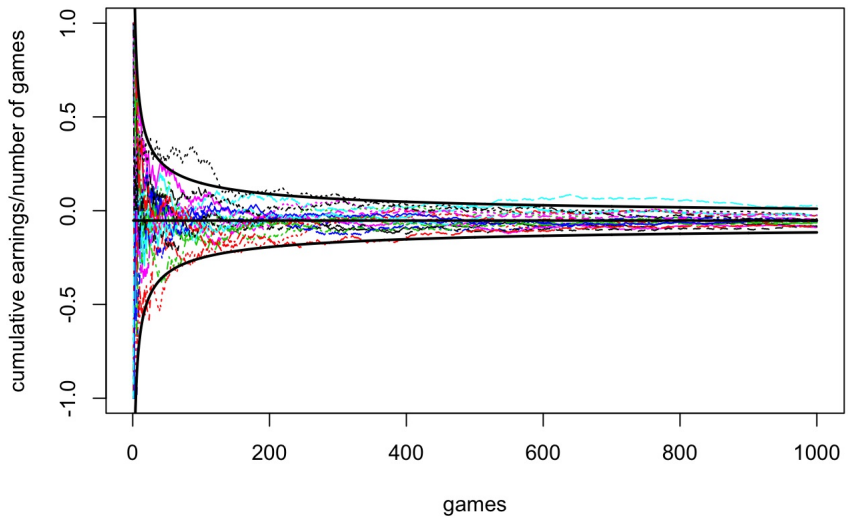
```r
# The coding adds, at the beginning of each history, that the
# history starts at 0 games and 0 earnings. The horizontal line
# represents a "break even" history.

# Next, we plot, on one graph, 20 histories of length 1000. These
# histories are stored as the columns of cum.earn so as to use the
# command matplot (which plots matrices by columns).

par(opar)
cum.earn<-matrix(NA,nrow=1000,ncol=20)
for (i in 1:20) {
  r<-runif(1000)
  earn<-ifelse(r<pwin,1,-1)
  cum.earn[,i]<-cumsum(earn)
}

# In previous 12 runs of 100 games each, cum.earn was a vector that was overwritten
# for each of the 12 runs. Here above in the 20 runs of 1000 games each, cum.earn is
# a matrix, with each column storing the cumulative earnings for the 1000 games of the
# corresponding run, e.g. column 1 keeps the values for run 1.

# Add a row of all 0s.
cum.earn<-rbind(0,cum.earn)

# matplot: Plot the columns of one matrix against the columns of another.
# Says the arguments x, y of matplot(x,y) can be vectors or matrices of data for plotting.
# Here x is a vector, 0:1000, and y is a matrix. And so 20 graphs are produced with
# the y-axis being each of the 20 columns of matrix cum.earn.

matplot(0:1000,cum.earn,type="l",xlab="games",ylab="cumulative earnings")

# If X represents the number of games won, X~binomial(n,p) with
# p=18/38. If Y represents the cumulative earnings through n
# games, Y=2X-n. Thus E(Y)=2E(X)-n=n(2p-1)= -2n/38 and
# Var(Y)=4Var(X)=4np(1-p). It is 4 because there is a squaring operation.
# We plot the line of expected earnings
# and the curves E(Y) +/- 2 sqrt(Var(Y)) versus number of games played.

# To see expected earnings as a vector as the 1000 games are played use i(2p-1) after ith game
exp.earn<-2*pwin*(1:1000) - 1:1000
exp.earn<-c(0,exp.earn)
lines(0:1000,exp.earn,lwd=2)

# the above draws in the expected earnings as a bold line into the matplot

sd.earn<-2*sqrt(pwin*(1-pwin)*(1:1000))
sd.earn<-c(0,sd.earn)
lines(0:1000,exp.earn+2*sd.earn,lwd=2)
lines(0:1000,exp.earn-2*sd.earn,lwd=2)
```



```
# There are some histories which are more unlucky than what
# would be predicted by a E(Y)+/- 2 sqrt(Var(Y)) guideline. We notice
# however that, as predicted by the guideline, actual histories
# tend to spread away from the -2n/38 expected earnings line. This
# is contrary to popular misconceptions of what should happen when
# one plays a large number of games and is due to Var(Y) increasing
# with n.

# If we plot instead Y/n = the earnings as a proportion of n (the total amount
# bet in n games), we see that histories converge to their
# expected value E(Y/n) = -2/38. This is because Var(Y/n) decreases with n.
# Var (Y/n) will have an n squared term in the denominator, which cancels out one n in
# the numerator and hence one n remains in the denominator.

# Since first row is all 0, we drop that row, and plot proportions

prop.earn<-cum.earn[-1,]/(1:1000)
matplot(1:1000,prop.earn,type="l",xlab="games",ylab="cumulative earnings/number of games")

# As E[Y/n] mean earnings as a proportion of the number of games is (2p-1)

exp.prop.earn<-2*pwin-1
exp.prop.earn
```

```
## [1] -0.05263158
```

```
# Make it a vector of length 10000
exp.prop.earn<-rep(exp.prop.earn,1000)

sd.prop.earn<-2*sqrt(pwin*(1-pwin)/(1:1000))

lines(1:1000,exp.prop.earn,lwd=2)
lines(1:1000,exp.prop.earn+2*sd.prop.earn,lwd=2)
lines(1:1000,exp.prop.earn-2*sd.prop.earn,lwd=2)
```




```
# We calculate the probability that a bettor wins or ties. If for
# example he plays 9 games, he loses if he wins 4 or fewer games.
# If he plays 10 games, he also loses if he wins 4 or fewer games
# and ties if he wins exactly 5 games. pbinom(x,n,p) is the
# probability that a binomial(n,p) variable is less than or equal
# to x.

pwin
```

```
## [1] 0.4736842
```

```
n<-c(1:10,20,30,40,60,80,100)
n
```

```
##  [1]   1   2   3   4   5   6   7   8   9  10  20  30  40  60  80 100
```

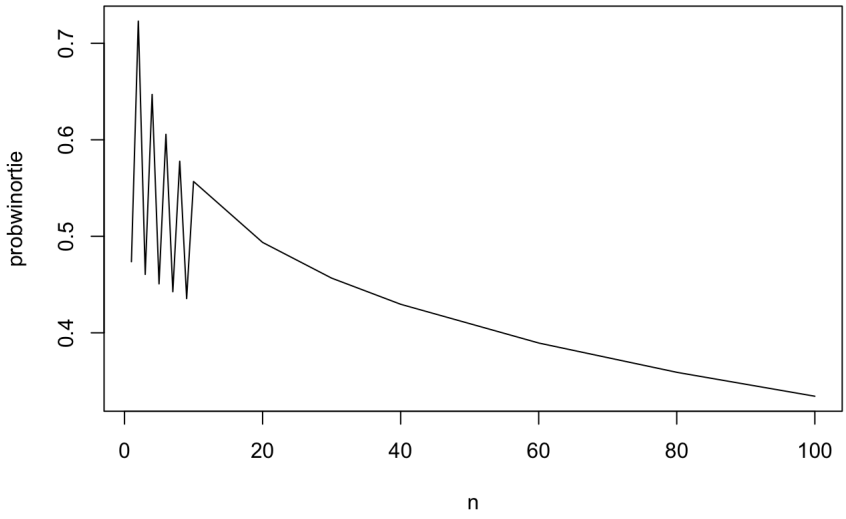
```
# Using help(floor), we see
# floor takes a single numeric argument x and returns a numeric vector containing the
# largest integers not greater than the corresponding elements of x.

lose<-floor(n/2-1/2)
lose
```

```
##  [1]  0  0  1  1  2  2  3  3  4  4  9 14 19 29 39 49
```

```
# pbinom(q, size, prob) - q: quantile; size: n and prob is p
# probwinortie is a vector of the same length as n, which is 16
# Win or Tie - i.e., do not lose

probwinortie<-1-pbinom(lose,n,pwin)

# column bind
cbind(n,probwinortie)
```

```
##          n probwinortie
##  [1,]    1    0.4736842
##  [2,]    2    0.7229917
##  [3,]    3    0.4605628
##  [4,]    4    0.6470254
##  [5,]    5    0.4507489
##  [6,]    6    0.6057041
##  [7,]    7    0.4425934
##  [8,]    8    0.5778036
##  [9,]    9    0.4354771
## [10,]   10    0.5568291
## [11,]   20    0.4937186
## [12,]   30    0.4566960
## [13,]   40    0.4296142
## [14,]   60    0.3894660
## [15,]   80    0.3591117
## [16,]  100    0.3343053
```

```
# Plot - see better probabilities when n is even - chance for ties
plot(n,probwinortie,type="l")
```




```
# We see that if the bettor plays 100 games, he has a 1/3
# probability of winning or tying.

# A bettor plays relatively few games. To analyze the situation from
# the casino's viewpoint we need to consider a large number of
# games and for this purpose we use the normal approximation to the binomial.

n<-c(100,1000,10000,100000,1000000)
temp<-(.48-pwin)/sqrt(pwin*(1-pwin))
temp
```

```
## [1] 0.01264911
```

```
cbind(n,pnorm(temp*sqrt(n)))
```

```
##          n
## [1,] 1e+02 0.5503284
## [2,] 1e+03 0.6554217
## [3,] 1e+04 0.8970484
## [4,] 1e+05 0.9999683
## [5,] 1e+06 1.0000000
```

reference code: http://www.ece.virginia.edu/~mv/edu/D2K/lectures/roulette/roulette.R

In 100,000 games, there is at least a 99.997% chance that the casino's edge is at least 4%. Thus, this simulation agrees with the above claim and I'm not just spouting crap. The attraction of betting is that the bettor has a "good" chance of winning, but real talk here, it's actually less than 50%. Which is worse than a coin toss. Why lose your money when you don't even have a 50/50 chance you'll win? So if you find yourself caught in a casino's greedy hands and haul your butt out of there and bail fat. Don't give them your money. Spend it on yourself, gotta look out for number 1.

Conclusions

Basically, if you didn't understand any of these fancy graphs and eloquent words, if you want to earn a lot of money gambling, you gotta have a lot of money in the first place. And at that point, why even gamble to make bank if you already have a swiss account. But be warned, the more you bet, the more likely you are to lose because the odds are against you.

Morale of the story, you're better off just keeping your money than try to make some quick cash at a casino. Play for fun, but don't expect to make a profit. Or, use that money and take a loved one out to dinner. Or even better, go out and buy a puppy. The best use of money, after all, is in experiences. That's something stats could never simulate.

…

But if you *have* to be a baller, at least double check that the casino you play at has a homeless shelter nearby.