Week 07

Nicole Hardy

31 de octubre de 2018

Gambler’s Ruin:

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:

Part A/D:

The probability that you have “busted” (lost all your money) by the time you have placed your one hundredth bet. The mean and variance of your bankroll after 100 hands (including busts).

```
#---------------------------------------------------------------------
# Building of various functions
#---------------------------------------------------------------------
# Keeps track of wins and losses up to max number of bets or no bankroll
play = function(game, bankroll, win_prob, num_bets, hands, bet, incr)
{
  wins = 0
  losses = 0
  start = win_prob
  while (bankroll > 0 & (num_bets < hands) & (win_prob <= 1))
  {
    roll = rbinom(1, 1, win_prob)
    if((roll == 1) & (win_prob < 1))
    {
      bankroll = bankroll + bet
      wins = wins + 1
      num_bets = num_bets + 1
      win_prob = win_prob + incr
    } else if(roll == 0)
    {
      bankroll = bankroll - bet
      losses = losses + 1
      num_bets = num_bets + 1
      win_prob = start
    } else if(roll == 1 & win_prob == 1)
    {
      bankroll = bankroll + bet
      wins = wins + 1
      num_bets = num_bets + 1
    }
  }
  game_matrix(game, wins, losses, bankroll)
}

# Saves a matrix with the individual trials as rows.
#Saves in GLOBAL environment
game_matrix = function(entry, num_win, num_loss, money_left)
{
  trial[entry,1] <<- num_win + num_loss
  trial[entry,2] <<- num_win/trial[entry,1]
  trial[entry,3] <<- num_loss/trial[entry, 1]
  trial[entry, 4] <<- money_left
}

#Calculates probability of busting for a given number of maximum bets
calc_prob_busted = function(data, iteration)
{
  data %>%
```

```
    filter(bankroll <= 0) %>%
    summarise(busted_prob = n()/iteration)
}

#calculates the mean time you go bust
mean_time_busted = function(data)
{
  data %>%
    filter(bankroll <= 0) %>%
    summarise(mean_time = mean(total))
}

#Creates empty trial matrix saves to GLOBAL environment
empty_trial_mat = function(iter)
{
  trial <<- as.data.frame(matrix(data = 0, nrow = iter, ncol = 4))
  colnames(trial) <<- c("total", "prob of winning", "prob of losing", "bankroll")
}

#Repeats everything for a set # of iterations. Saves results in matrix
#Prints probabilities, mean and variance
replicate_hands = function(bankroll, win_prob, bet, num_bets, hands, iter, incr, mean_
fun, var_fun)
{
  for (game in 1:iter)
  {
    losses = 0
    wins = 0
    play(game, bankroll, win_prob, num_bets, hands, bet, incr)
  }
  return(c(calc_prob_busted(trial, iteration = iter), mean_time_busted(trial),
           mean = mean_fun(trial$bankroll), var = var_fun(trial$bankroll)))
}
```

Parts A/D

```
#Creates empty matrix and runs everything for 5000 trials, 100 hands
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 0.5, bet = 100, num_bets = 0, hands = 10
0, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0.3274
##
## $mean_time
## [1] 53.51985
##
## $mean
## [1] 991.48
##
## $var
## [1] 859531.3
```

```
head(trial)
```

```
##   total prob of winning prob of losing bankroll
## 1   100       0.5100000      0.4900000     1200
## 2   100       0.5700000      0.4300000     2400
## 3   100       0.5400000      0.4600000     1800
## 4   100       0.5000000      0.5000000     1000
## 5    28       0.3214286      0.6785714        0
## 6    92       0.4456522      0.5543478        0
```

Part B/E:

The probability that you have busted by the time you have placed your five hundredth bet by simulating the outcome directly. The mean and variance of your bankroll after 500 hands (including busts).

```
#Creates empty matrix and runs everything for 5000 trials, 500 hands
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 0.5, bet = 100, num_bets = 0, hands = 50
0, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0.6392
##
## $mean_time
## [1] 145.8467
##
## $mean
## [1] 1041.48
##
## $var
## [1] 2718111
```

```
head(trial)
```

```
##   total prob of winning prob of losing bankroll
## 1   360       0.4861111      0.5138889        0
## 2   222       0.4774775      0.5225225        0
## 3   284       0.4823944      0.5176056        0
## 4   500       0.5340000      0.4660000     4400
## 5    48       0.3958333      0.6041667        0
## 6    44       0.3863636      0.6136364        0
```

Part C:

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

```
#Mean time of bust 5000 hands
#Creates empty matrix and runs everything for 5000 trials, 5000 hands
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 0.5, bet = 100, num_bets = 0, hands = 500
0, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0.879
##
## $mean_time
## [1] 515.4712
##
## $mean
## [1] 1082.92
##
## $var
## [1] 11225313
```

```
head(trial)
```

```
##   total prob of winning prob of losing bankroll
## 1   158       0.4683544      0.5316456        0
## 2   826       0.4939467      0.5060533        0
## 3    28       0.3214286      0.6785714        0
## 4   250       0.4800000      0.5200000        0
## 5   516       0.4903101      0.5096899        0
## 6  5000       0.5098000      0.4902000    10800
```

American Roulette:

Part A/D:

```
#Creates empty matrix and runs everything for 5000 trials, 100 hands
set.seed(1234)
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 18/38, bet = 100, num_bets = 0, hands = 10
0, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0.5112
##
## $mean_time
## [1] 52.7457
##
## $mean
## [1] 587.76
##
## $var
## [1] 578605.9
```

Part B/E:

```
#Creates empty matrix and runs everything for 5000 trials, 500 hands
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 18/38, bet = 100, num_bets = 0, hands = 50
0, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0.9188
##
## $mean_time
## [1] 128.5272
##
## $mean
## [1] 158.72
##
## $var
## [1] 393158.6
```

Part C:

```
#Mean time of bust 5000 hands
#Creates empty matrix and runs everything for 5000 trials, 5000 hands
empty_trial_mat(5000)
replicate_hands(bankroll = 1000, win_prob = 18/38, bet = 100, num_bets = 0, hands = 50
00, iter = 5000, incr = 0, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 1
##
## $mean_time
## [1] 194.0684
##
## $mean
## [1] 0
##
## $var
## [1] 0
```

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.

Part a:

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 example. win_prob incr. by 1% for each win. Goes back to 48% for losses
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.48, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.01, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 297524
##
## $var
## [1] 986864469
```

```
head(trial)
```

```
##   total prob of winning prob of losing bankroll
## 1 1e+05         0.48982        0.51018   296400
## 2 1e+05         0.49154        0.50846   330800
## 3 1e+05         0.49054        0.50946   310800
## 4 1e+05         0.48998        0.51002   299600
## 5 1e+05         0.48874        0.51126   274800
## 6 1e+05         0.48784        0.51216   256800
```

```
range(trial$bankroll)
```

```
## [1] 233000 383200
```

No, this does not seem to be fair since the expectation is less than the starting bankroll of 500,000.

Part 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?

```
#start at win_prob = 50%
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.50, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.01, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 718063
##
## $var
## [1] 1257415486
```

```
#start at win_prob = 46%
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.46, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.01, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 1
##
## $mean_time
## [1] 80386.3
##
## $mean
## [1] 0
##
## $var
## [1] 0
```

```
#sets inital values before loop
prob = seq(from = .46, to = .5, by = 0.001)
count = 1
money = 500000
empty_trial_mat(100)
test = replicate_hands(bankroll = money, win_prob = prob[1], bet = 100, num_bets = 0,
hands = 100000, iter = 100, incr = 0.01, mean_fun = mean, var_fun =  var)

#searches for probability that gives the expectation to be +/- 4% of the starting bank
roll
while(test[[3]] > 1.04*money | test[[3]] < 0.96*money)
{
  count = count + 1
  test = replicate_hands(bankroll = money, win_prob = prob[count], bet = 100, num_bet
s = 0, hands = 100000, iter = 100, incr = 0.01, mean_fun = mean, var_fun =  var)
  print(prob[count])
}
```

```
## [1] 0.461
## [1] 0.462
## [1] 0.463
## [1] 0.464
## [1] 0.465
## [1] 0.466
## [1] 0.467
## [1] 0.468
## [1] 0.469
## [1] 0.47
## [1] 0.471
## [1] 0.472
## [1] 0.473
## [1] 0.474
## [1] 0.475
## [1] 0.476
## [1] 0.477
## [1] 0.478
## [1] 0.479
## [1] 0.48
## [1] 0.481
## [1] 0.482
## [1] 0.483
## [1] 0.484
## [1] 0.485
## [1] 0.486
## [1] 0.487
## [1] 0.488
## [1] 0.489
```

The return value closest to a fair game is 48.9%. Yes, it can be done automatically.

Part 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.

```
#start at incr = 0.05%
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.48, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.005, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 194756
##
## $var
## [1] 1177234408
```

```
#start at incr = 2%
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.48, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.02, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 506468
##
## $var
## [1] 191012299
```

```
#sets inital values before loop
increment = seq(from = 0.005, to = 0.02, by = 0.0001)
money = 500000
count = 1
empty_trial_mat(100)
test = replicate_hands(500000, 0.48, 100, 0, 100000, 100, incr = increment[1], mean, v
ar)

#searches for increment that gives the expectation +/- 4% of the starting bankroll
while(test[[3]] > 1.04*money | test[[3]] < 0.96*money)
{
  count = count + 1
  test = replicate_hands(bankroll = money, win_prob = 0.48, bet = 100, num_bets = 0, h
ands = 100000, iter = 100, incr = increment[count], mean_fun = mean, var_fun = var)
  print(increment[count])
}
```

```
## [1] 0.0051
## [1] 0.0052
## [1] 0.0053
## [1] 0.0054
## [1] 0.0055
## [1] 0.0056
## [1] 0.0057
## [1] 0.0058
## [1] 0.0059
## [1] 0.006
## [1] 0.0061
## [1] 0.0062
## [1] 0.0063
## [1] 0.0064
## [1] 0.0065
## [1] 0.0066
## [1] 0.0067
## [1] 0.0068
## [1] 0.0069
## [1] 0.007
## [1] 0.0071
## [1] 0.0072
## [1] 0.0073
## [1] 0.0074
## [1] 0.0075
## [1] 0.0076
## [1] 0.0077
## [1] 0.0078
## [1] 0.0079
## [1] 0.008
## [1] 0.0081
## [1] 0.0082
## [1] 0.0083
## [1] 0.0084
## [1] 0.0085
## [1] 0.0086
## [1] 0.0087
## [1] 0.0088
## [1] 0.0089
## [1] 0.009
## [1] 0.0091
## [1] 0.0092
## [1] 0.0093
## [1] 0.0094
## [1] 0.0095
## [1] 0.0096
## [1] 0.0097
## [1] 0.0098
```

```
## [1] 0.0099
## [1] 0.01
## [1] 0.0101
## [1] 0.0102
## [1] 0.0103
## [1] 0.0104
## [1] 0.0105
## [1] 0.0106
## [1] 0.0107
## [1] 0.0108
## [1] 0.0109
## [1] 0.011
## [1] 0.0111
## [1] 0.0112
## [1] 0.0113
## [1] 0.0114
## [1] 0.0115
## [1] 0.0116
## [1] 0.0117
## [1] 0.0118
## [1] 0.0119
## [1] 0.012
## [1] 0.0121
## [1] 0.0122
## [1] 0.0123
## [1] 0.0124
## [1] 0.0125
## [1] 0.0126
## [1] 0.0127
## [1] 0.0128
## [1] 0.0129
## [1] 0.013
## [1] 0.0131
## [1] 0.0132
## [1] 0.0133
## [1] 0.0134
## [1] 0.0135
## [1] 0.0136
## [1] 0.0137
## [1] 0.0138
## [1] 0.0139
## [1] 0.014
## [1] 0.0141
## [1] 0.0142
## [1] 0.0143
## [1] 0.0144
## [1] 0.0145
## [1] 0.0146
## [1] 0.0147
```

```
## [1] 0.0148
## [1] 0.0149
## [1] 0.015
## [1] 0.0151
## [1] 0.0152
## [1] 0.0153
## [1] 0.0154
## [1] 0.0155
## [1] 0.0156
## [1] 0.0157
## [1] 0.0158
## [1] 0.0159
## [1] 0.016
## [1] 0.0161
## [1] 0.0162
## [1] 0.0163
## [1] 0.0164
## [1] 0.0165
## [1] 0.0166
## [1] 0.0167
```

The increment value that produces the most fair game is 0.0167.

Bootstrap Function and Test:

```
boot_ci = function(data, n, func)
{
  boot_stat = apply(replicate(n, sample(data[,4], dim(data)[1], replace = TRUE)), MARG
IN = 2, func)
  quantile(boot_stat, c(0.025, 0.975))
}
```

```
#Finds 95% confidence interval of expected bankroll based on first Markov question wit
h win_prob = .489
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.489, bet = 100, num_bets = 0, hands =
100000, iter = 100, incr = 0.01, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 493614
##
## $var
## [1] 1093676974
```

```
boot_ci(trial, 1000, mean)
```

```
##     2.5%    97.5%
## 487051.8 499988.4
```

Bootstrap variance:

3b ultimately has a much wider confidence interval and higher variance than 3c.

```
#Bootstrap variance based on 3b
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.489, bet = 100, num_bets = 0, hands =
100000, iter = 100, incr = 0.01, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 484124
##
## $var
## [1] 1239219216
```

```
boot_ci(trial, 1000, var)
```

```
##       2.5%      97.5%
##  904127198 1610509438
```

```
#Bootstrap variance based on 3c
#3b has a much higher variance and wider confidence intervals
empty_trial_mat(100)
replicate_hands(bankroll = 500000, win_prob = 0.48, bet = 100, num_bets = 0, hands = 1
00000, iter = 100, incr = 0.0167, mean_fun =  mean, var_fun = var)
```

```
## $busted_prob
## [1] 0
##
## $mean_time
## [1] NaN
##
## $mean
## [1] 483494
##
## $var
## [1] 834034307
```

```
boot_ci(trial, 1000, var)
```

```
##       2.5%      97.5%
##  562241312 1132140269
```