STAT 206 Homework 6_LIHUA_XU

Due Monday, November 13, 5:00 PM

General instructions for homework: Homework must be completed as an R Markdown file. Be sure to include your name in the file. Give the commands to answer each question in its own code block, which will also produce plots that will be automatically embedded in the output file. Each answer must be supported by written statements as well as any code used. (Examining your various objects in the "Environment" section of RStudio is insufficient – you must use scripted commands.)

Part I - Gambler's Ruin

1. 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 the following. (You must stop playing if your player has gone bust.)
   a. the probability that you have "busted" (lost all your money) by the time you have placed your one hundredth bet.
   b. the probability that you have busted by the time you have placed your five hundredth bet by simulating the outcome directly.
   c. the mean time you go bust, given that you go bust within the first 5000 hands.
   d. the mean and variance of your bankroll after 100 hands (including busts).
   e. the mean and variance of your bankroll after 500 hands (including busts).

```
set.seed(1)
#Assuming, every time,the person wins, the bet would be 0.11*(total money)
Gambler <- function(mon_bet,money,p)
{i <- 0
  while(TRUE) {
    PV <- runif(1)
    if(PV <= p)
    {money <- mon_bet + money
     mon_bet <- (mon_bet + money)*0.11}
    else{money <- money-mon_bet
         mon_bet <- mon_bet}
    i=i+1
    if(money<mon_bet){break}}
  num_time <- i
  return(num_time)}

times <- c()
num <- 5000
for(i in 1:num)
{following_turn <- Gambler(100,1000,0.5)
times <- c(times,following_turn)}
```

```
##1(a)
N <- length(times)
A <- length(which(times<=100))
Prob_1 <- A/N
#The probability would be:
Prob_1
```

```
## [1] 0.2156
```

```
##1(b)
B <- length(which(times<=500))
Prob_2 <- B/N
#The probability would be:
Prob_2
```

```
## [1] 0.7042
```

```
##1(c)
C <- times[which(times<=5000)]
mean_c <- mean(C)
mean_c
```

```
## [1] 408.3572
```

```
##1(d)
set.seed(1)
Gambler_100 <- function(mon_bet,money,p,n)
{i<-0
  while(mon_bet>0) {
    PV=runif(1)
    if(PV <= p)
    {money <- mon_bet + money
    mon_bet <- (mon_bet + money)*0.11
    }else{money <- money-mon_bet
    mon_bet <- mon_bet}
    i=i+1
    if(i==n){break}
    if(money<mon_bet){break}}
  num_time <- i
  return(money)}

money_whole <- c()
num <- 5000
for(i in 1:num)
{folloing_turn_m = Gambler_100(100,1000,0.5,100)
 money_whole <- c(money_whole,folloing_turn_m)}

#The mean and variance of your bankroll after 100 hands (including busts) are
M_d <- c(mean=mean(money_whole))
V_d <- c(variance=var(money_whole))
M_d
```

```
##     mean
## 983.1206
```

```
V_d
```

```
## variance
##  5928989
```

```
##1(e)
money_whole <- c()
num <- 5000
for(i in 1:num)
```

```
{folloing_turn_m = Gambler_100(100,1000,0.5,500)
 money_whole <- c(money_whole,folloing_turn_m)}

#The mean and variance of your bankroll after 500 hands (including busts) are
M_e <- c(mean=mean(money_whole))
V_e <- c(variance=var(money_whole))
M_e
```

```
##     mean
## 568.4773
```

```
V_e
```

```
## variance
## 29735724
```

2. Repeat the previous problem with betting on black in American roulette, where the probability of winning on any spin is 18/38 for an even payout.

```
times <- c()
num <- 5000
for(i in 1:num)
{following_turn <- Gambler(100,1000,18/38)
times <- c(times,following_turn)}
```

```
#2(a)
N <- length(times)
A <- length(which(times<=100))
Prob_1 <- A/N
#The probability would be:
Prob_1
```

```
## [1] 0.2838
```

```
##2(b)
B <- length(which(times<=500))
Prob_2 <- B/N
#The probability would be:
Prob_2
```

```
## [1] 0.8322
```

```
##2(c)
C <- times[which(times<=5000)]
mean_c <- mean(C)
mean_c
```

```
## [1] 286.9228
```

```
##2(d)
money_whole <- c()
num <- 5000
for(i in 1:num)
{folloing_turn_m = Gambler_100(100,1000,18/38,100)
 money_whole <- c(money_whole,folloing_turn_m)}

#The mean and variance of your bankroll after 100 hands (including busts) are
M_d <- c(mean=mean(money_whole))
```

```
V_d <- c(variance=var(money_whole))
M_d
```

```
##     mean
## 460.3944
```

```
V_d
```

```
## variance
##  1328166
```

```
##2(e)
money_whole <- c()
num <- 5000
for(i in 1:num)
{folloing_turn_m = Gambler_100(100,1000,18/38,500)
 money_whole <- c(money_whole,folloing_turn_m)}

#The mean and variance of your bankroll after 500 hands (including busts) are
M_e <- c(mean=mean(money_whole))
V_e <- c(variance=var(money_whole))
M_e
```

```
##     mean
## 43.94803
```

```
V_e
```

```
## variance
## 393227.8
```

3. For the American roulette problem in the previous question, you calculated a mean value. Because you saved these final results in a vector, use the bootstrap to estimate the variance of the return in each case for your final answer.

```
bootstrap <- sample(money_whole,5000,replace=T)
mean(bootstrap)
```

```
## [1] 50.06168
```

```
#The variance is:
var(bootstrap)
```

```
## [1] 736017.5
```

Part II - Elo Ratings

One of the earliest examples of a convergent, adaptive Markov process was the rating system devised by Arpad Elo to rank chess players. It has endured for so long as a simple system for so long that it is used as a primary ranking system in many other scenarios, including the NBA team rankings (Nate Silver) and Scrabble (NASPA).

The main idea is two players have ratings R_A and R_B. The estimated probability that player A will win is modeled by a logistic curve,

$$P(A) = \frac{1}{1 + \exp(R_B - R_A)}$$

and once a game is finished, a player's rating is updated based on whether they won the game:

$$R_A(\text{new}) = R_A(\text{old}) + K(1 - P(A))$$

or if the lost the game:

$$R_A(\text{new}) = R_A(\text{old}) - KP(A)$$

for some factor K. (Note that both player ratings change.) Our goal is to simulate a repetitive tournament with 10,000 games to see if it converges on the true values.

4. Create a "true" vector of ratings for 13 players whose ratings range from -2 to 2 in even intervals. Create another vector with the current ratings which will be updated on a game-by-game basis, and a matrix with 13 rows and 10,000 columns into which we will deposit the ratings over time.

```
true_ratings <- seq(from=-2,to=2,by=1/3)
true_ratings
```

```
##  [1] -2.0000000 -1.6666667 -1.3333333 -1.0000000 -0.6666667 -0.3333333
##  [7]  0.0000000  0.3333333  0.6666667  1.0000000  1.3333333  1.6666667
## [13]  2.0000000
```

```
#I will assume the current ratings
current_rating <- c(1,0.5,-1.3,0.8,1.7,-1.9,-1.7,-0.5,1.2,0.6,-0.4,-0.7,1.0)
Matrix_deposite <- matrix(0,nrow=13,ncol=10000)
```

5. Write a function that simulates a game between players i and j given their true underlying ratings. This should be a simple draw from `rbinom(1,1,p)` with the appropriate probability.

```
#p should equal to 0.5, when rbinom(1,1,p) equal to 0, i win.
#When rbinom(1,1,p) equal to 1, j win.
simulat_i_j_true <- function(i,j,true_ratings,p=0.5,K){
  O <- rbinom(1,1,p)
  if (O==0){
    Prob_wi <- 1/(1+exp(true_ratings[j]-true_ratings[i]))
    Prob_wj <- 1/(1+exp(true_ratings[i]-true_ratings[j]))
    true_ratings[i] <- true_ratings[i]+K*(1-Prob_wi)
    true_ratings[j] <- true_ratings[j]-K*Prob_wj}
    else{
    Prob_wi <- 1/(1+exp(true_ratings[j]-true_ratings[i]))
    Prob_wj <- 1/(1+exp(true_ratings[i]-true_ratings[j]))
    true_ratings[j] <- true_ratings[j]+K*(1-Prob_wi)
    true_ratings[i] <- true_ratings[i]-K*Prob_wj}
 return(true_ratings)}
```

6. Write a function that, given a value of K, replaces the ratings for the two players who just played a game with their updated ratings given the result from the previous question.

```
#p should equal to 0.5, when rbinom(1,1,p) equal to 0, i win.
#When rbinom(1,1,p) equal to 1, j win.
simulat_i_j <- function(i,j,current_rating,p=0.5,K){
  O <- rbinom(1,1,p)
  if (O==0){
    Prob_wi <- 1/(1+exp(current_rating[j]-current_rating[i]))
    Prob_wj <- 1/(1+exp(current_rating[i]-current_rating[j]))
    current_rating[i] <- current_rating[i]+K*(1-Prob_wi)
    current_rating[j] <- current_rating[j]-K*Prob_wj}
    else{
    Prob_wi <- 1/(1+exp(current_rating[j]-current_rating[i]))
```

```
    Prob_wj <- 1/(1+exp(current_rating[i]-current_rating[j]))
    current_rating[j] <- current_rating[j]+K*(1-Prob_wi)
    current_rating[i] <- current_rating[i]-K*Prob_wj}
 return(current_rating)}

simulat_i_j(i=3,j=11,current_rating,p=0.5,K=0.02)
```

```
##  [1]  1.000000  0.500000 -1.314219  0.800000  1.700000 -1.900000 -1.700000
##  [8] -0.500000  1.200000  0.600000 -0.385781 -0.700000  1.000000
```

7. Write a function that selects two players at random from the 13, makes them play a game according to their true ratings, and updates their observed ratings.

```
current_rating <- c(1,0.5,-1.3,0.8,1.7,-1.9,-1.7,-0.5,1.2,0.6,-0.4,-0.7,1.0)

problem7 <- function(current_rating,p=0.5,K){
people_2 <- sample(1:13, size = 2)
current_rating <- simulat_i_j(people_2[1],people_2[2],current_rating,p,K)
return(current_rating)
}
```

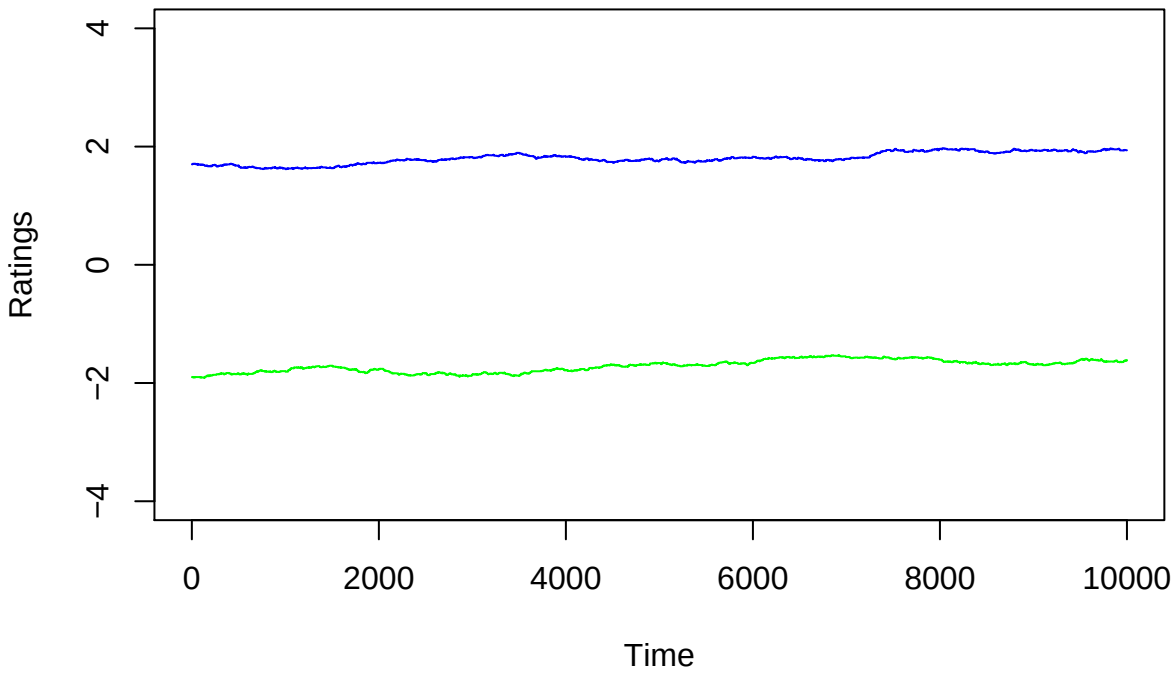
8. Finally, write a function that simulates a tournament as prescribed above: 10,000 games should be played between randomly chosen opponents, and the updated ratings should be saved in your rating matrix by iteration.

```
current_rating <- c(1,0.5,-1.3,0.8,1.7,-1.9,-1.7,-0.5,1.2,0.6,-0.4,-0.7,1.0)

problem8 <- function(n_times=10000,p=0.5,K){
  for (b in 1:n_times){
  current_rating <- problem7(current_rating,p=0.5,K)
  Matrix_deposite[,b] <- current_rating
  }
  return(Matrix_deposite)
}
```

9. Run this tournament with $K = 0.01$. Plot the rating for the best player over time using `plot(..., ty="l")`; add the rating for the worst player using `lines(...)`. Do they appear to converge to the true ratings?

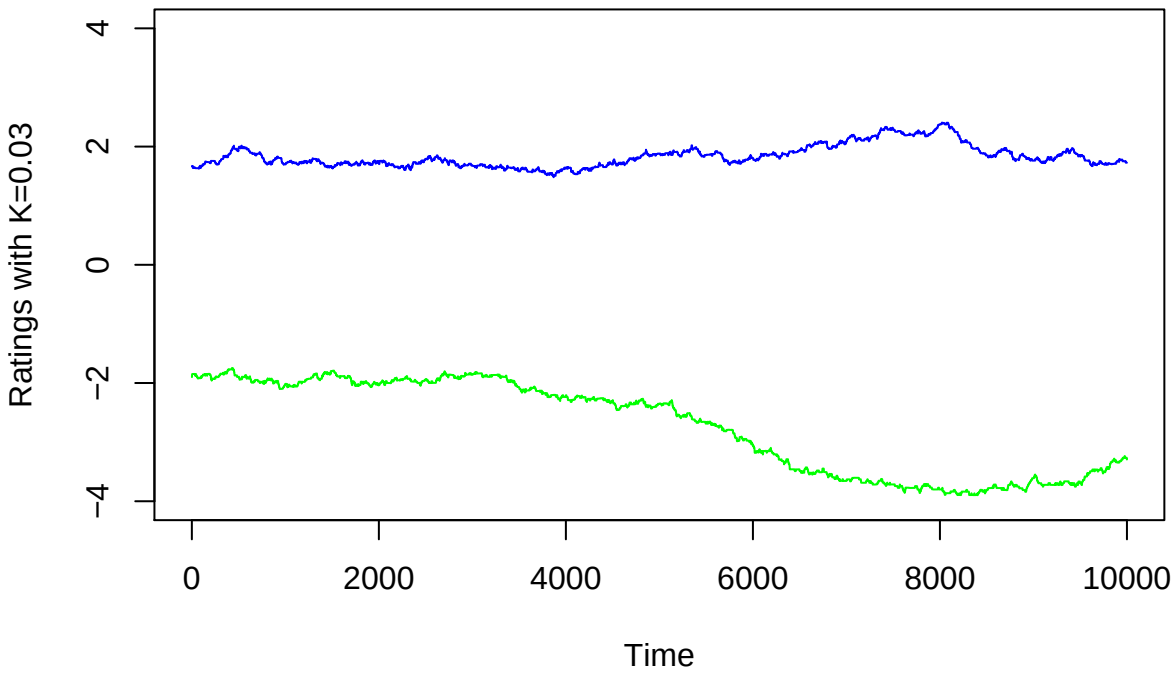
```
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.01)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings",col="blue",xlim=c(0,10000),ylim=c(-4,4))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```



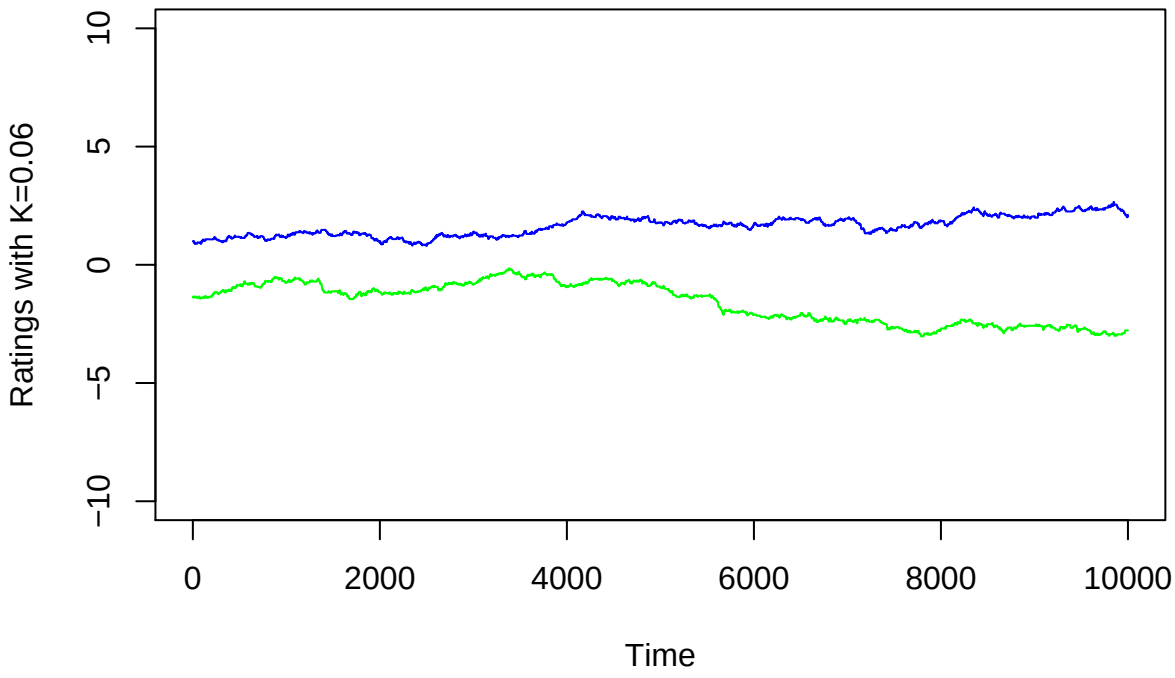

```
#They appear NOT to converge to the true ratings.There are a lot of uncertainties in side this problem.
#But from the plot, the value is kind of come closer to the true ratings.
```

10. Repeat the previous step with K equal to 0.03, 0.06, 0.1, 0.3, 0.6 and 1. Which appears to give the most reliable rating results?

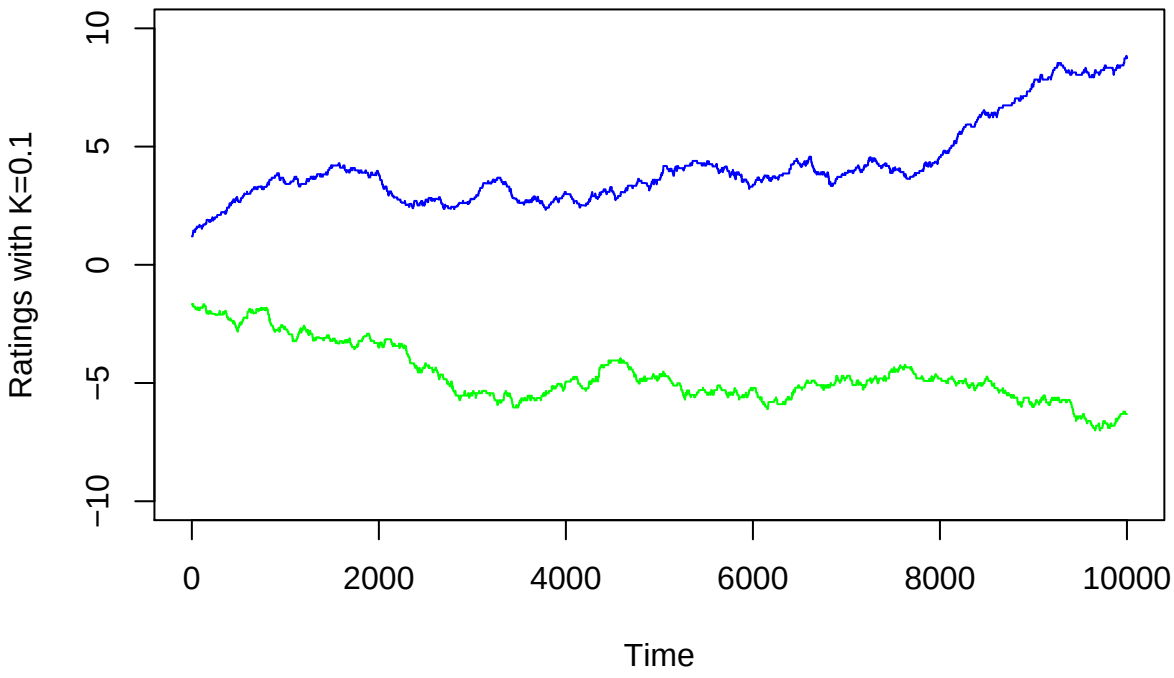
```
#K=0.03
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.03)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=0.03",col="blue",xlim=c(0,10000),ylim=c(-4,4))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```



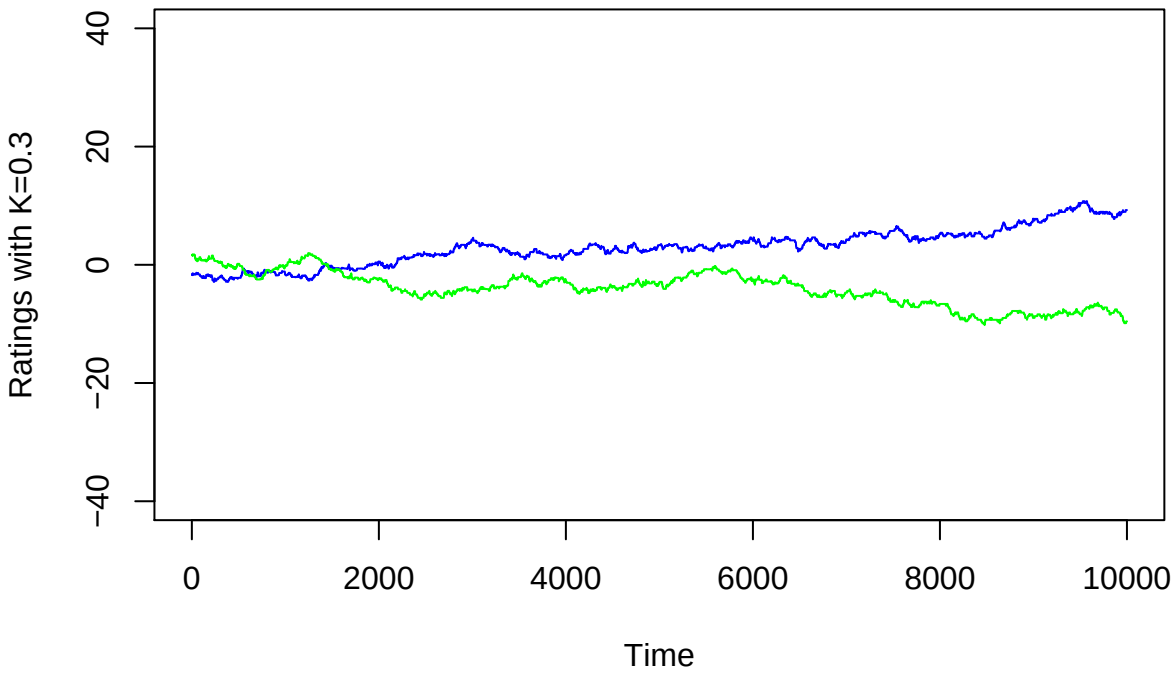

```
#K=0.06
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.06)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=0.06",col="blue",xlim=c(0,10000),ylim=c(-10,10))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```



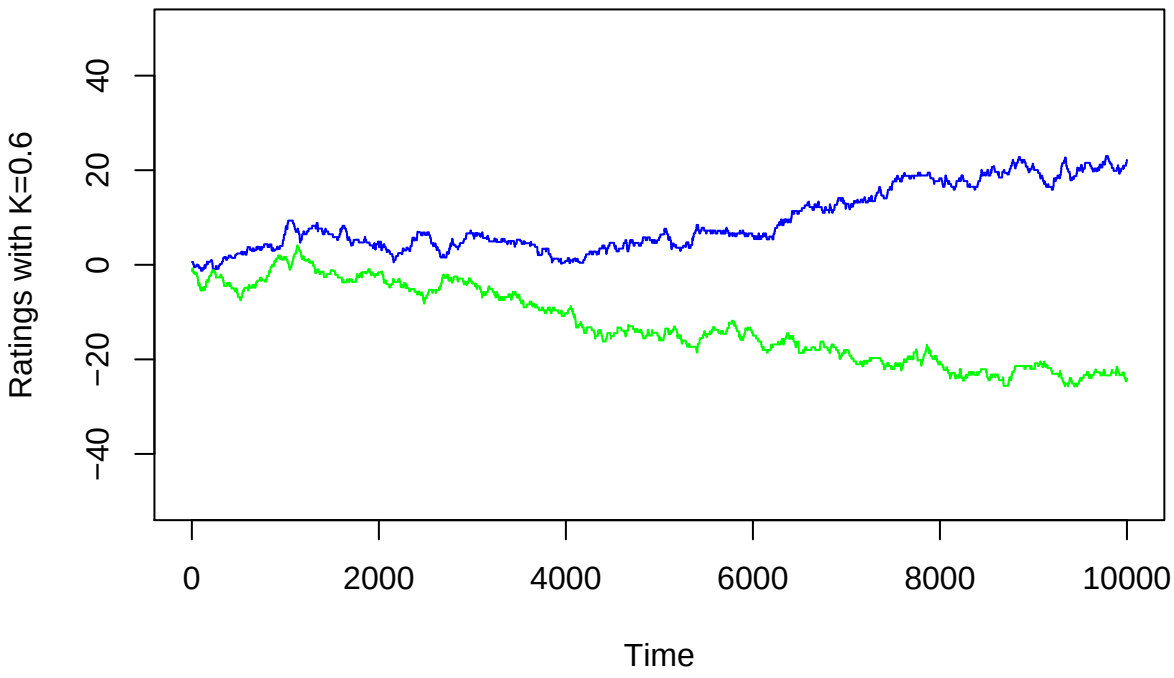

```
#K=0.1
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.1)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=0.1",col="blue",xlim=c(0,10000),ylim=c(-10,10))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```



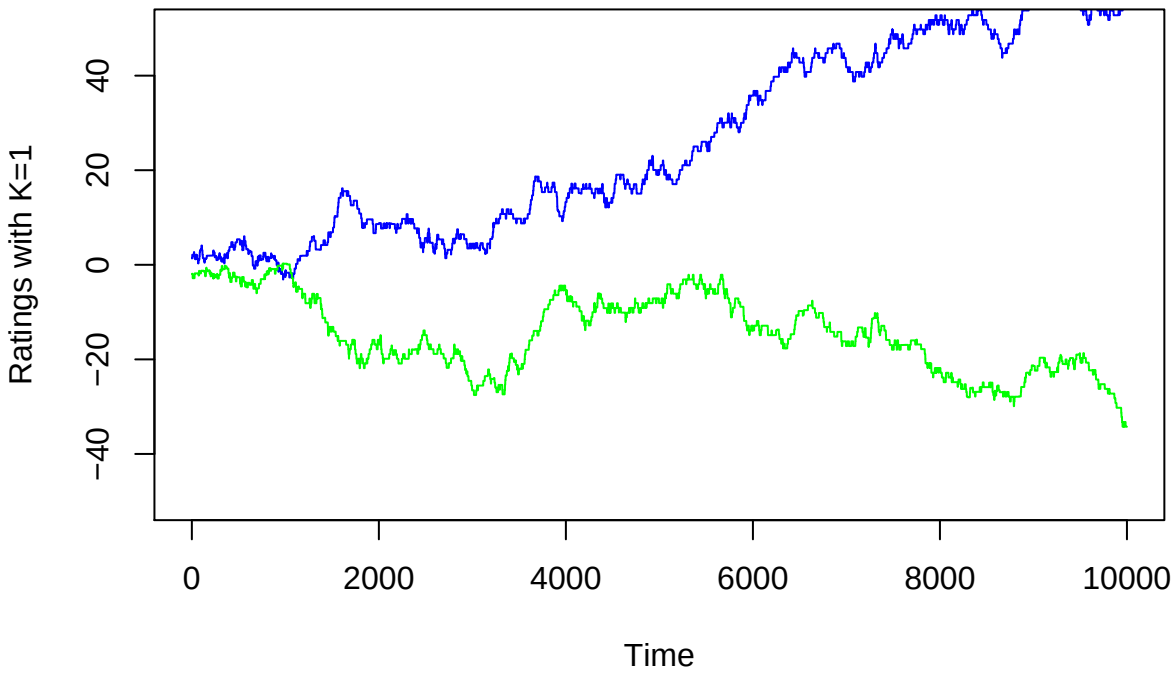

```
#K=0.3
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.3)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=0.3",col="blue",xlim=c(0,10000),ylim=c(-40,40))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```




```
#K=0.6
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=0.6)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=0.6",col="blue",xlim=c(0,10000),ylim=c(-50,50))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```




```
#K=1
Matrix_10000 <- problem8(n_times=10000,p=0.5,K=1)
best_n <- which(Matrix_10000[,10000]==max(Matrix_10000[,10000]))
worst_n <- which(Matrix_10000[,10000]==min(Matrix_10000[,10000]))
plot(Matrix_10000[best_n,], ty="l",xlab="Time",
     ylab="Ratings with K=1",col="blue",xlim=c(0,10000),ylim=c(-50,50))
lines(1:10000,Matrix_10000[worst_n,], ty="l",col="green")
```



From my point of view, when K equal to 0.03, it seems to give the most reliable rating results.