Tutorial

Solving the Monty Hall Problem with Simulations:

A super-simple intro to loops and *if* statements in R

Bodo Winter[1]

Last updated: January 12, 2012

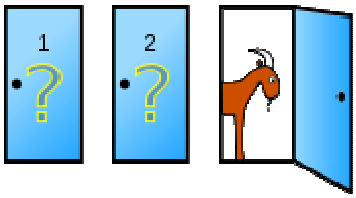
Introduction

Have you heard of the Monty Hall problem? In this simple tutorial, we're going to solve this problem by means of simulation – if you haven't worked with loops and *if* statements in R before, this will be useful for you. If you already know this stuff – stay tuned… the tutorial might still be fun!

So, what's the Monty Hall problem[2]? Monty Hall was the host of the famous gameshow *Let's make a deal* which was popular in the 70s and had many offshoots in other countries (e.g. *Geh aufs Ganze!* in Germany). After a series of deals, each competitor in this show would be confronted with three doors, only one of which lead to the desired prize – usually a car. Behind the other two doors were things that nobody would want, e.g. a goat.

[1] For updates and other tutorials, check my webpage www.bodowinter.com. If you have any suggestions, please write me an email: bodo@bodowinter.com

[2] For an extensive overview, check Wikipedia: http://en.wikipedia.org/wiki/Monty_Hall_problem



(Source: http://en.wikipedia.org/wiki/Monty_Hall_problem)

The competitor would initially choose a door and then the host would open one of the remaining doors (always one that *didn't* conceal the prize, of course). After that, the participant was given the choice to switch doors or not to switch doors. So, in the above picture, let's say you initially chose door 1. Monty Hall opens door 3 and there is – lo and behold – a goat. Now you're given the choice to stay at door 1 or to switch to door 2. What is a better strategy, to switch or not to switch? Or does it matter at all? Before you read on, think about this for a second: What is your initial and intuitive response?

Marilyn vos Savant, once in the Guinness Book of Records for having the highest measured IQ, published a solution to the Monty Hall problem in her column in the magazine *Parade*… This solution indicated that it's actually a better strategy to switch doors. To many, this seemed extremely counterintuitive because at first sight, the probability behind each of the doors should be the same: 1/3 a chance of winning the prize. A deluge of letters to the editors flooded in, some 10,000 responses, stating how Ms. vos Savant was wrong, she just had to be! How could she dare to publish such a simple logical mistake, how preposterous!! Many of the letters were from highly educated people. In the end, they were all proven wrong and Marilyn's solution stood the test of time.

So, why don't we try to answer this question for ourselves, using R. We will solve the Monty Hall problem in a brute force fashion, simply by simulating a 10,000 runs of choices between three doors… and calculating whether switching or not switching is better.

The R code that we need to do for this is super-simple[3]. Have a quick look at the R code in the box, that's all we're going to need! The tutorial will guide you through this code.

```
doors <- c("A","B","C")

xdata=c()
for(i in 1:1000)
{
prize <- sample(doors)[1]
pick <- sample(doors)[1]
open <- sample(doors[which(doors != pick & doors != prize)])[1]
switchyes <- doors[which(doors != pick & doors != open)]
if(pick==prize){xdata=c(xdata,"noswitchwin")}
if(switchyes==prize){xdata=c(xdata,"switchwin")}
}

length(which(xdata == "switchwin"))
length(which(xdata == "noswitchwin"))
```

So, let's get started!!! First, we need to make a vector "doors" with three doors: "A", "B" and "C".

```
doors <- c("A","B","C")
```

Then, we need to create an object into which we store all the information that is generated by the loop.

[3] It'll be really good for you if you have Tinn-R (http://sciviews.org/Tinn-R/) or any other external device to store your R code (e.g. a text editor). Then you can type everything in, and as soon as you're finished with the loop, copy it into the R console.

```
xdata=c()
```

Now, we need the loop. The general structure of loops in R is the following: You put everything that should be executed repeatedly into the curly brackets. Into the regular brackets, you define how often the loop should be executed. The expression *i in 1:1000* means that your so-called "running variable" *i* will be chosen from the vector of numbers between 1 and 1000, so the variable *i* will be "1" in the first run-through, "2" in the next one and so on.

```
for(i in 1:1000)
{
}
```

Now, let's go into the loop. For each particular game show, we need to place the prize behind one of the doors. We do this by picking one door at random from the character vector *doors*. The command *sample( )* simply puts the three doors into a random order, and then we index this randomly assorted vector with [1], taking the first element of that vector. The result of this is saved into the object *prize*.

```
prize <- sample(doors)[1]
```

Next, we need to simulate how the participant in the game show picks a random door. For this, we do exactly the same thing as in the last step, except that we save it into the object *pick*. Because we sample two times, *prize* and *pick* can be the same door (e.g. "A") – this simulates a success for the gameshow participant.

```
pick <- sample(doors)[1]
```

Now comes the most crucial step. The game master chooses a door other than the one that was picked by the participant… and of course, he doesn't choose

the door with the prize. So, we sample from the *doors* vector, but we don't take an element that equals *prize*. Also, we don't want our simulation to pick the door the participant has chosen, so we should something that is not equal to *pick* either. The command *which(doors != pick)* returns a number, which is the position of the vector *door* that is not equal to *pick*. You can combine multiple conditions with the *&* operator. So, *which(doors != pick)* gives you a number, and you can use that number to index the *doors* vector. This is done by the composite command *doors[which(doors != pick & doors != prize)]*, which will give you a character (e.g. "B") which is the door that is *not* chosen by the participant and that is also *not* the door with the prize. Now, in case the vectors *prize* and the *pick* are equal, *which( )* will actually return two numbers (because Monty Hall could open either of two doors). For this case, we need to pick one element by indexing *[1]*. However, this indexing should be done on the randomized vector – otherwise we end up picking always the order of the *doors* vector, therefore we need to use *sample( )*, too, and we end up with the following command:

```
open <- sample(doors[which(doors != pick & doors != prize)])[1]
```

We already have the door that is picked under the no-switching condition, let's turn to simulating another participant that – on the same prize/goat layout – would have switched.

```
switchyes <- doors[which(doors != pick & doors != open)]
```

Simple, isn't it? We simulate switching by taking one element of the vector *doors* which was not the original pick and not the door that was opened.

Now, we use two *if* commands to see who wins: the switching or the non-switching participant. Within the curly brackets of the *if* command, you put what ought to be executed if a certain condition is met. Within the regular brackets, you state what the condition is. In this case, if *pick==prize*, we assign the character

"noswitchin" to that position in the *xdata* results vector. Importantly, we need to concatenate this new element with the old vector – if we don't do this, our *xdata* vector would always only have one element after you completed the loop… namely, the element of the last cycle.

```
if(pick==prize){xdata=c(xdata,"noswitchwin")}
if(switchyes==prize){xdata=c(xdata,"switchwin")}
```

Now, it's time to enter the whole loop into R! So, copy everything from *xdata=c()* to the final curly bracket of the loop into the R console.

We conclude by counting how many instances of "switchwin" and "noswitchwin" occur in the *xdata* results.

```
length(which(xdata == "switchwin"))
length(which(xdata == "noswitchwin"))
```

That's it! The length of the switching should be approaching 666 in a series of 1,000 simulations. The more simulations you do (try 10,000!), the more you will approach a ratio of 2/3 for the switching case and 1/3 for the no-switching case. We proved Marilyn vos Savant right, switching *is* better!! Yay!!

But hey, let's just have a quick look at her solution, which was much more analytical and didn't use simulations at all. Let's simply enumerate all the possible arrangements (1-3). These will be:

	Door A	**Door B**	**Door C**
(1)	Prize	Goat	Goat
(2)	Goat	Prize	Goat
(3)	Goat	Goat	Prize

So, the prize will either be behind door A (arrangement 1), door B (arrangement 2) or door C (arrangement 3), and the goats will be behind the two other doors. Let's say our gameshow participant always picked Door A. If you *don't* switch, your chance of winning is exactly 1/3, as expected. Only the first of the three arrangements will give you the prize. If you do switch, then there's *two* arrangements (2) and (3) where you can win, thus your chances are 2/3 of winning. There only arrangement where you wouldn't win is arrangement 1.

Crucially, all of this only works because the gameshow host would *not* open the door that contains the prize. So in (2), Monty Hall would have opened door C… in (3), he would have opened door B. This means that the probability of winning the prize with the switching strategy is 2/3 as opposed to 1/3 in the no-switching case. The argument I made in the last paragraph for initially picking arrangement 1 also generalizes to initially picking arrangement 2 or 3.

Another way to wrap your head around this is that initially (on your first pick), you start with an exactly 1/3 probability. That means your chance of *not winning* is 2/3, because there are two other doors. When the gameshow master one of the two remaining doors, he halves the chances of *not winning* for these two doors. 2/3 divided by two equals 1/3, which is the chance of *not winning*. The chance of winning for the two doors is thus 1-(1/3) which is 2/3. The chance of your first pick being the winning door is unaffected by the game master's door-opening action and remains 1/3 …

Cool, isn't it?