Philipps-Universität Marburg

Fakultät für Mathematik und Informatik

Data Science in Biomedicine

Monte Carlo Verification of the Monty Hall Problem using Linear Congruential Generators

Machine Learning

Hannah Franziska Löchel, Jan Ruhland

April 28, 2023

Contents

1 Introduction

The Monty Hall problem is named after the host of the game show "Let's make a Deal" and was first scientifically discussed in the "Ask Marilyn" column [1]. Since the publication a lot of mathematicians have discussed the problem from different perspectives such as Bayesian interpretation or causal inference [2, 3, 4].
The problem describes the interaction between the player and the game master, resulting in the player's dilemma. The player is given the choice to open one of three doors. Behind one door is the win and behind the other two doors are losses. After the player has chosen one door, the game master will show one door with a loss behind it which isn't the door chosen by the player. The player is now faced with the dilemma of switching the door or staying with the door he had chosen in the beginning. The problem can be solved using conditional probabilities on the three events: the player's door chosen by the game master, and the winning door. Theoretical calculations show that the player doubles the chances of winning by switching the door [2, 3, 4].
The interpretation of probabilities always depends on the underlying paradigm. The use of the Bayesian or frequentist paradigm is philosophical in nature, yet, depending on the problem, one method might be favored [5]. The first solution to the Monty Hall problem used the Bayesian approach and in this report, a Monte Carlo simulation of the player's dilemma is used to verify the Bayesian results.
Monte Carlo simulations describe the repetitions of experiments to infer statistical properties of a system [6]. The first attempt at a computational Monte Carlo simulation was attributed to the theoretical physicist Enrico Fermi in 1930. The computing power at that time limited early attempts at Monte Carlo simulations. Due to the increase of commercially available high-performance computing services, Monte Carlo simulations are standard practice in modern research [7].
The aim of the thesis is to implement a Monte Carlo simulation of the Monty Hall problem in Python based on rudimentary operations and Python modules. The theoretical results should be derivable by only the simulation and hence show a modern approach to solving complex statistical problems.

2 Background

One of the first methods to generate pseudorandom numbers is given by equation (2.1) [8]. The equation describes a recursive function defined on the interval [0,m]. a, c, $m \in \mathbb{N}$ are parameters that define the period of the recursive function as well as the number of repeating patterns.

$$x_i = (a \cdot x_{i-1} + c) \mod m \tag{2.1}$$

The parameters a, c and m have to be chosen to obtain a full period with a minimal number of repeating patterns. A full period can only be achieved if the following conditions are met [9]:

i) m and c are coprime

ii) each prime divisor p of m divides $(a - 1)$

iii) if 4 divides m, then 4 divides $(a - 1)$

Repeating patterns are unavoidable but the frequency can be reduced by additional conditions for the parameters a, c and m [10]. The parameters given in table 2.1 fulfill these conditions and are used in other software applications [10].

Table 2.1: The table shows the values for the parameters a, c and m of the linear congruential generator taken from [10].

a	c	m
1664525	1013904223	2^{32}

The theoretical foundation of the Monte Carlo simulation lies within the law of large numbers. The theorem states that the normed sum of independently and identically distributed random variables X_i converges to the sample mean as described in equation (2.2) [11].

$$\mathbb{E}[X] = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} X_i \tag{2.2}$$

3 Materials and Methods

The Monte Carlo simulation was developed in the programming language Python. Due to the rudimentary implementation, the only additional module was Matplotlib for plotting and the corresponding versions of the software components are listed in Table 3.1.

Table 3.1: Software components used for the implementation of the Monte Carlo simulation of the Monty Hall Problem.

Software	Version
Python	3.7.6
Matplotlib	3.1.3

For the Monte Carlo simulation we needed three different random numbers representing the events P (door picked by player), W (winning door), and G (door revealed by the game master). Each event was defined by equation (2.1) with a different seed value. The parameter values are provided in table 2.1 and the seed values are listed in Table 3.2.

Table 3.2: The seed values for the different random number generators which represent the events P, W and G.

Event	seed
Player (P)	476756
Win (W)	1076
Game master (G)	90213

First, we drew the two random numbers for the events P and W. The interval $[0, 2^{32}-1]$ was subdivided into three disjoint and equidistant intervals. The three intervals represented the three doors, e.g. if a generated random number lands in the first interval, then the first door will be picked. If the doors for P and W are the same, then we have to generate the third random number for event G. For event G, the whole interval was subdivided into two disjoint and equidistant intervals corresponding to the remaining two doors. In the simulation, we also considered the case, that the player chooses one of the two remaining doors randomly. The player's random choice was implemented analogously to the event that the game master has to pick one of two doors.

4 Results

For the evaluation of the simulation, we simulated 10000 games and calculated the fraction of wins for every subinterval. Figure 4.1 shows the win rates with respect to the number of games for the three strategies: i) staying with the initial decision, ii) changing the door and iii) randomly choosing one of the two remaining doors. In the figure, a high fluctuation of the win rates can be observed for a small number of games. Moreover, the simulations suggest a convergence of the win rates for a sufficiently large number of games. For this case, the Monte Carlo simulation predicts the probability of winning for strategy i) to be 33.4%, for strategy ii) 66.6%, and for strategy iii) 50.0%. This is in accordance with theoretical considerations [2, 3, 4].

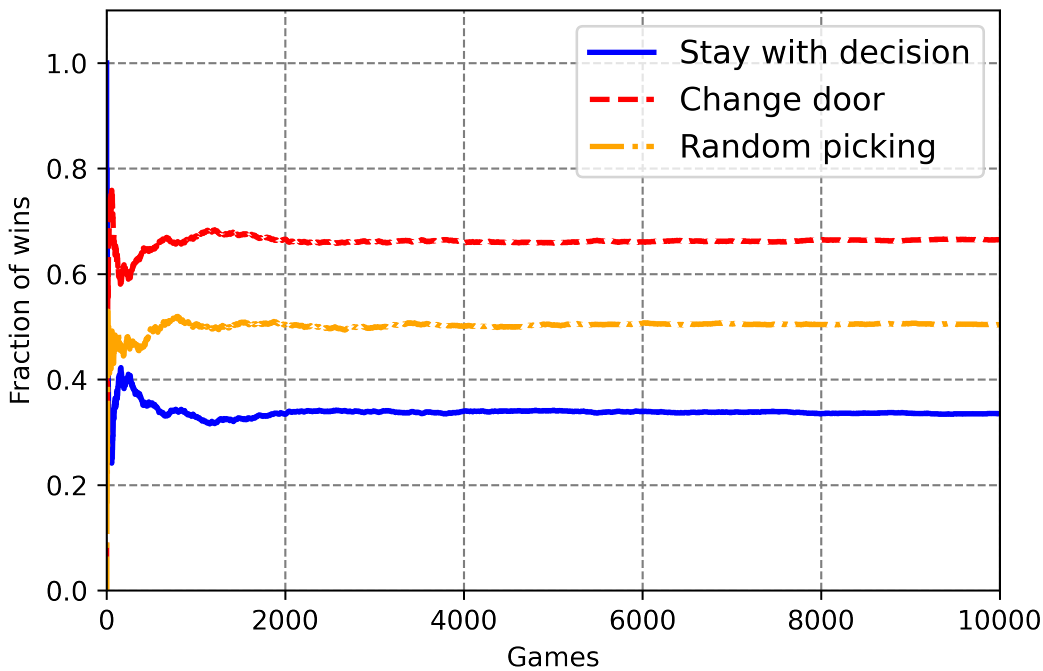

Figure 4.1: The figure shows the resulting win rates of the Monte Carlo simulation of the Monty Hall problem for 10000 games. The blue line shows the win rate if the player doesn't change the door after the game master revealed a loss door. The red line corresponds to the decision to change the door and the orange line to a random choice.

5 Discussion

In this thesis, a Monte Carlo simulation of the Monty Hall problem was implemented in Python using a linear congruential generator. The implementation was used to simulate 10000 games and compare the probability of winning to the theoretically derived values. The observed properties of the simulation, shown in figure 4.1, can be explained by the law of large numbers given by equation (2.2). For the explicit calculation, we have to consider the probability of winning $\mathbb{P}(X = 1)$ and losing $\mathbb{P}(X = 0)$, i.e. choosing the right door given our applied strategy.

$$\mathbb{E}[X] = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} X_i = 1 \cdot \mathbb{P}(X = 1) + 0 \cdot \mathbb{P}(X = 0) = \mathbb{P}(X = 1) \tag{5.1}$$

Equation (5.1) shows explicitly that the simulated win rates converge to the expected fraction of wins and hence to the probability of winning for the three strategies. This shows that both the theoretical consideration as well as the simulation lead to the same result. Despite having the same result, the adjustment of the theoretical framework to experimental alternations is more complicated than for the Monte Carlo method.

Bibliography

[1] Marilyn vos Savant. *Parade Magazine: Sept. 9*, page 22, 1990.

[2] Richard D. Gill. The Monty Hall Problem is not a Probability Puzzle (it's a challenge in mathematical modelling). *arXiv e-prints*, art. arXiv:1002.0651, February 2010. doi: 10.48550/arXiv.1002.0651.

[3] J. Rosenhouse. *The Monty Hall Problem: The Remarkable Story of Math's Most Contentious Brain Teaser*. Oxford University Press, USA, 2009. ISBN 9780199887927. URL `https://books.google.de/books?id=3M30ZPyLJVgC`.

[4] J. P. Morgan, N. R. Chaganty, R. C. Dahiya, and M. J. Doviak. Let's make a deal: The player's dilemma. *The American Statistician*, 45(4):284–287, 1991. ISSN 00031305. URL `http://www.jstor.org/stable/2684453`.

[5] M. J. Bayarri and J. O. Berger. The Interplay of Bayesian and Frequentist Analysis. *Statistical Science*, 19(1):58 – 80, 2004. doi: 10.1214/088342304000000116. URL `https://doi.org/10.1214/088342304000000116`.

[6] Isaac Todhunter. *A History of the Mathematical Theory of Probability: From the Time of Pascal to that of Laplace*. Cambridge Library Collection - Mathematics. Cambridge University Press, 2014. doi: 10.1017/CBO9781139923576.

[7] N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science (1987 Special Issue dedicated to Stanislaw Ulam)*, pages 125–130, 1987.

[8] W. E. Thomson. A Modified Congruence Method of Generating Pseudo-random Numbers. *The Computer Journal*, 1(2):83–83, 01 1958. ISSN 0010-4620. doi: 10.1093/comjnl/1.2.83. URL `https://doi.org/10.1093/comjnl/1.2.83`.

[9] T. E. Hull and A. R. Dobell. Random number generators. *SIAM Review*, 4(3):230–254, 1962. doi: 10.1137/1004061. URL `https://doi.org/10.1137/1004061`.

[10] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, USA, 3 edition, 2007. ISBN 0521880688.

[11] David R. Cox and David V. Hinkley. *Theoretical Statistics*. Chapman & Hall, London, England, 1974.