Gambler's Ruin Problem

Shreyes Madgaonkar

November 18, 2020

1 Introduction

Consider a gambler, say **A**, playing with an initial capital of i (in some unit of currency) and with a total capital N on the table. **A** random event such as a coin toss will decide whether **A** wins or not. If **A** wins, i increases by one and if he loses, i is reduced by 1. The game ends when either **A** loses all of his money i.e. $i = 0$ or wins enough rounds so that $i = N$.

If p is the probability that **A** wins a round, then the probability that **A** wins the whole game ($i = N$) is given by following distribution,

$$P(i) \begin{cases} \dfrac{1 - (\frac{q}{p})^i}{1 - (\frac{q}{p})^N}, & \text{if } p \neq 0.5 \\ \dfrac{i}{N}, & \text{if } p = 0.5 \end{cases} \tag{1}$$

where $q = 1 - p$

2 Analytical approach

We're given some values of initial capital for **A** to start gambling with. These are $i = 2x$ where $x = 1, 2 \ldots 9$. Also, the total capital, $N = 20$ and $p = 2/5$.
We use (1) to calculate the probability that **A** wins if he starts with initial capital i, where i takes the values given above.

i: Initial capital	$P(i)_{\text{analytical}}$: Probability of winning the whole game
2	3.760×10^{-4}
4	1.222×10^{-3}
6	3.125×10^{-3}
8	7.408×10^{-3}
10	1.704×10^{-2}
12	3.872×10^{-2}
14	8.751×10^{-2}
16	0.197
18	0.444

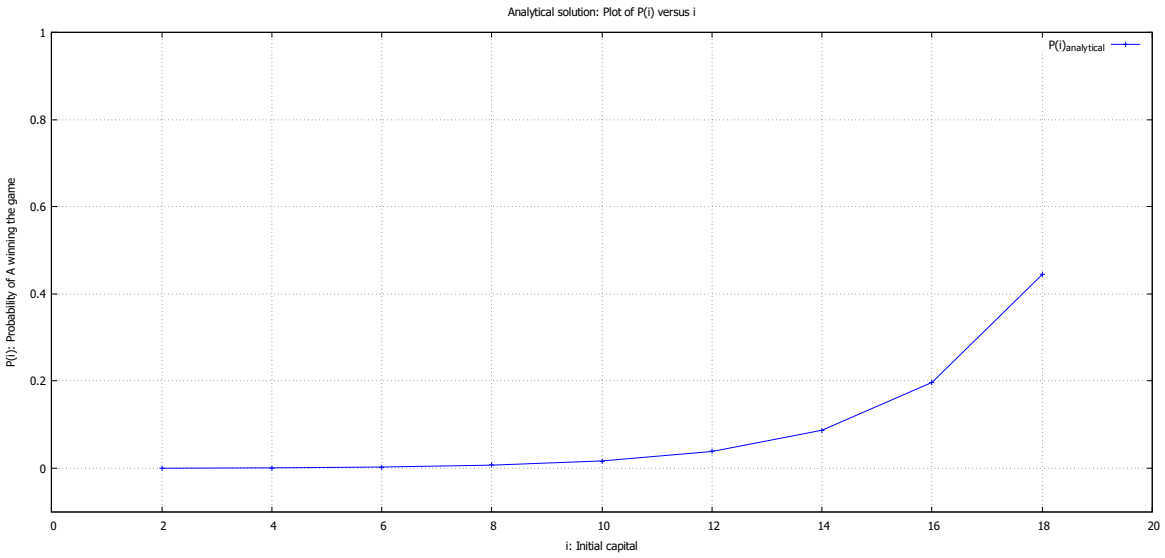

Figure 1: Analytical: Plot of P(i) versus i

3 Numerical approach

The numerical approach[1] to finding the winning probability involves playing several games, for the same value of i. Each game will end in either **A** winning or losing. So, each game (for a particular i) will have $N = 20$ and $p = 2/5$. In this simulation, 20000 games are being played. Say the number of games is set to some number m and out of n_{games}, **A** wins n_{wins} . Then,

$$P(i) = \frac{\text{Number of games won}}{\text{Number of games played}} = \frac{n_{wins}}{n_{games}} \tag{2}$$

Obtained result:

i: Initial capital	$P(i)_{\text{numerical}}$: Probability of winning the whole game
2	3.500×10^{-4}
4	1.200×10^{-3}
6	2.900×10^{-3}
8	8.150×10^{-3}
10	1.725×10^{-2}
12	3.755×10^{-2}
14	8.370×10^{-2}
16	0.1959
18	0.4352

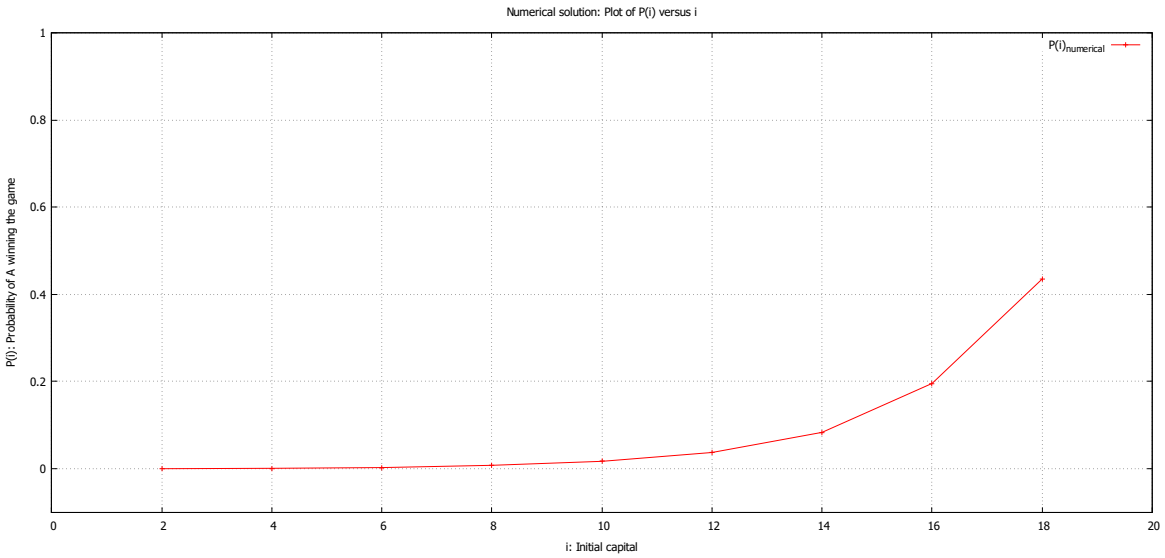

Figure 2: Numerical: Plot of P(i) versus i

[1]Github repo: https://github.com/mshreyes/Computational-Physics/tree/master/MCM

4 Comparison

The numerical approach implements a Monte Carlo method i.e. here, invoking a random number generator[2] to decide whether or not **A** wins.
Following chart (along with the plot) presents a comparison between both the methods and the error[3] (in numerical method):

i	$P(i)_{\text{numerical}}$	$P(i)_{\text{analytical}}$	Error
2	0.0003500	0.0003760	0.00002602
4	0.0012000	0.0012221	0.00002208
6	0.0029000	0.0031257	0.00022570
8	0.0081500	0.0074088	0.00074115
10	0.0172500	0.0170459	0.00020407
12	0.0375500	0.0387294	0.00117936
14	0.0837000	0.0875171	0.00381709
16	0.1959000	0.1972895	0.00138947
18	0.4352000	0.4442773	0.00907731

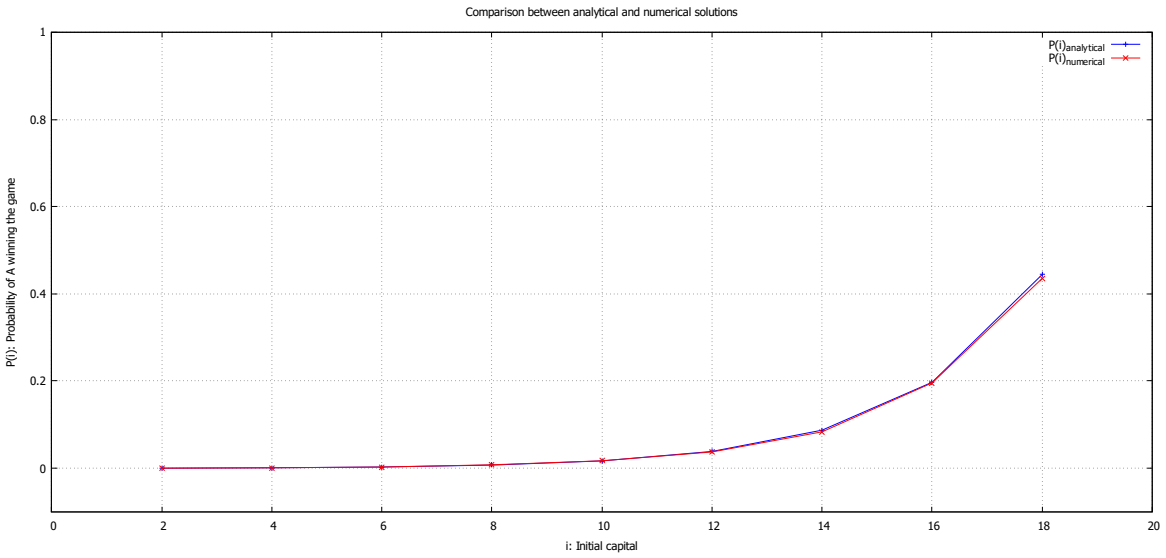

Figure 3: Comparison: Plot of P(i) versus i

[2]I've used gfortran's **random_number** subroutine which implements xoshiro256** PRNG.

[3]Error here is not static and will change every time the program is run. Also, the number of games played (for numerical approach) will affect this value. More games played will result in a better approximation of $P(i)$.