# Relatório Semestral

Orientando: Gustavo Silva Garone

Orientadora: Elisabeti Kira

2025-02-16

# PASSEIO ALEATÓRIO: SIMULAÇÕES PARA VALIDAÇÃO DE MODELOS

## Introdução

A Simulação de Monte Carlo é uma técnica baseada em amostragem aleatória para emular sistemas complexos, normalmente por meios computacionais (HARRISON, 2010). Dessa forma, é útil no estudo do comportamento de passeios aleatórios com parâmetros diversos.

Neste projeto, realizado conjuntamente com Eduardo Ishihara, analisamos um estimador proposto por esse para passeios mais flexíveis através do emprego dessas simulações na linguagem de programação Julia. A descrição desses passeios e de suas diferenças quando comparados ao problema clássico da Ruína do Jogador será apresentada ao longo do corpo do texto.

## Atividades Desenvolvidas

Inicialmente, realizamos uma revisão da literatura existente para identificar os principais resultados sobre o problema da Ruína do Jogador e suas variações. Na versão original do jogo proposta por Pascal (EDWARDS, 1983), dois jogadores, A e B, competem apostando certa quantia de dinheiro no lançamento de uma moeda honesta. Ao ganhar a aposta, o jogador A recebe R$1,00 do jogador B e vice-versa, descrevendo um martingal ("The Solution as a Fair Game (Martingale)", 1984). Além da questão proposta por Pascal sobre a probabilidade do jogador ir a ruína, ou seja, perder todo seu dinheiro, dado que começou com uma quantia $a$, também foram estudados o tempo esperado de duração do jogo (STERN, 1975) e, pelo cálculo de equações de diferenças finitas, a variância dessa duração (ANDĚL; HUDECOVÁ, 2012). Todavia, os resultados obtidos por esses autores funcionam apenas para ganho e perda de R$1,00 por aposta.

**Introdução do estimador**

Usando os resultados de STERN (1975), temos a esperança da duração $T$ de um jogo iniciado com saldo $a, \mu_a = E(T_a)$

$$\mu_a = E(T_a) = \begin{cases} a(N-a) & \textbf{se } p = q = 1/2, \\ \frac{a}{q-p} - \frac{N}{q-p}\left(\frac{1-(q/p)^a}{1-(q/p)^N}\right) & \textbf{se } p \neq q. \end{cases}$$

Vamos considerar um passeio com a chance de ganhar $p = 0.4, q = 1 - p$, o início $a = 30$ e final $N = 100$. Da fórmula, temos que

$$\mu_a = \frac{30}{0.2} - \frac{100}{0.2}\left(\frac{1-(0.6/0.4)^{30}}{1-(0.6/0.2)^{100}}\right) \approx 150$$

Utilizando o estimador proposto por Ishihara

$$\widehat{E}(T_a) = \begin{cases} \frac{N-a}{p-q} & \textbf{se } p > q \\ \frac{a}{q-p} & \textbf{se } p < q \end{cases}$$

Para $q \neq p \neq \frac{1}{2}$, temos $\frac{30}{0.2} = 150$. O que condiz com os resultados teóricos.

**Aplicações em casos flexíveis**

Por si só, o último resultado não diz muito sobre o estimador, uma vez que podem conferir pelo pequeno valor de $\left(\frac{1-(q/p)^a}{1-(q/p)^N}\right)$. Então, passamos a testar casos em que a fórmula da esperança original deixou de funcionar.

Adaptando o estimador acima, podemos trocar o denominador por uma esperança calculada com base nas regras do jogo. Seja X a variável aleatória que contém as regras do jogo:

$$X = \begin{cases} \text{ganho, com probabilidade } p \\ \text{perda, com probabilidade } q = 1 - p \end{cases}$$

Com esse recurso, podemos expandir o estimador proposto acima para funcionar com jogos mais complexos.

Um desses jogos foi uma simples variação do jogo anterior, ainda com $p = 0.4, q = 1-p, a = 30$ e $N = 100$, fizemos o apostador ganhar R$2,00 por vitória, mas ainda perder apenas R$1,00 por derrota. Nesse caso, temos nossa variável de regras X

$$X = \begin{cases} 2, \text{ com probabilidade } 0.4 \\ -1, \text{ com probabilidade } 0.6 \end{cases}$$

E a versão expandida do estimador

$$\widehat{E}(T_a) = \begin{cases} \frac{N-a}{E(X)} & \textbf{se } E(X) > 0 \\ \left|\frac{a}{E(X)}\right| & \textbf{se } E(X) < 0 \end{cases}$$

Desenvolvemos algoritmos de simulação baseados no método de Monte Carlo para testar o estimador. A primeira versão do algoritmo, implementada em Python, não satisfez as demandas de tempo e complexidade que desejavamos. Para solucionar essa limitação, migramos para a linguagem Julia, com velocidade computacional competitiva com o C (GODOY et al., 2023). Assim, conseguimos rodar um número maior de simulações em tempo razoável para melhorar a acurácia dos resultados (RITTER et al., 2011).

Aplicando a fórmula original, temos

$$\mu_a = 150$$

Isso desconsidera a nova regra de ganho do jogo, o que é refletido na discrepância com o valor simulado de 350 (veja o Apêndice A para o código dessa simulação assim como uma comparação com entre as linguagens Julia e Python)

Considerando que $E(X) = 0.2$, Aplicando o estimador, temos que

$$\widehat{E}(T_a) = \frac{70}{0.2} = 350$$

Durante o desenvolvimento dos algoritmos, introduzimos novas regras ao jogo para testarmos a flexibilidade do estimador. Nas primeiras simulações, definimos que o jogador A ganhava 1 real com probabilidade $\frac{1}{3}$, ganhava 3 reais com probabilidade $\frac{1}{3}$ e perdia 1 real com probabilidade $\frac{1}{3}$. Com $a = 30, N = 100$, temos as simulações (Apêndice B), que $\mu_a \approx 70.7$. Nessa configuração, $E(X) = 1$ e, pelo estiamdor, $\widehat{E}(T_a) = \frac{100-30}{1} = 70$.

Testamos o estimador sob diferentes regras e, na maioria dos casos, mostrou forte concordância com os valores simulados. Isso destaca a vantagem do estimador: estimar a duração de um jogo com regras diversas, o que, até onde encontramos na literatura, não foi descrito apenas com desenvolvimento teórico em probabilidade.

Estudamos também como simplificar a variável aleatória de regras $X$ através de médias quando esta pode assumir mais que dois valores.

O ponto fraco do estimador atualmente reside nas regras que levam a $E(X)$ próximo de 0, uma vez que, conforme $E(X)$ se aproxima de 0, descrevendo o passeio como um martingal, sua presença no denominador provoca divergências quando comparado com o valor simulado.

## Conclusão e Passos Seguintes

Os resultados obtidos indicam que o estimador proposto possui grande flexibilidade para diferentes configurações do jogo e mostra-se especialmente útil, pois as técnicas atuais não são capazes de fornecer um resultado teórico exato para as configurações de jogos propostas. Ademais, o estimador fornece estimativas mais precisas quando há maior circulação de dinheiro entre os jogadores. No entanto, observamos que o estimador tende a divergir rapidamente dos resultados simulados quando a esperança da variável aleatória que rege as regras do jogo aproxima-se de zero devido ao denominador.

Outro desafio identificado foi a complexidade computacional envolvida na simulação de jogos com muitas regras, isto é, quando a variável aleatória pode assumir um grande número de

valores. Essa limitação impõe a necessidade de soluções mais eficientes para lidar com cenários de alta complexidade, o que será um obstáculo a ser superado no estudo do caso não homogêneo.

Um interessante caminho de abordagem que buscaremos explorar é a conexão entre passeios aleatórios e redes elétricas como descrito por DOYLE; SNELL (1984). Queremos descobrir se existem conexões entre o estimador proposto e fórmulas da física, além de outras abordagens para estudarmos passeios aleatórios, como o emprego de equações de diferenças finitas e o estudo de martingais.

## Bibliografia


ANDĚL, J.; HUDECOVÁ, Š. Variance of the game duration in the gambler's ruin problem. **Statistics & Probability Letters**, v. 82, n. 9, p. 1750–1754, 2012.

DOYLE, P. G.; SNELL, J. L. **Random Walks and Electric Networks**. [s.l.] American Mathematical Soc., 1984.

EDWARDS, A. W. F. Pascal's Problem: The 'Gambler's Ruin'. **International Statistical Review / Revue Internationale de Statistique**, v. 51, n. 1, p. 73–79, 1983.

GODOY, W. F. et al. **Evaluating performance and portability of high-level programming models: Julia, Python/Numba, and Kokkos on exascale nodes**. 2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). **Anais**... Em: 2023 IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM WORKSHOPS (IPDPSW). maio 2023. Disponível em: <https://ieeexplore.ieee.org/abstract/document/10196600>. Acesso em: 14 fev. 2025

HARRISON, R. L. Introduction To Monte Carlo Simulation. **AIP conference proceedings**, v. 1204, p. 17–21, 5 jan. 2010.

RITTER, F. E. et al. Determining the Number of Simulation Runs: Treating Simulations as Theories by Not Sampling Their Behavior. Em: ROTHROCK, L.; NARAYANAN, S. (Eds.). **Human-in-the-Loop Simulations: Methods and Practice**. London: Springer, 2011. p. 97–116.

STERN, F. Conditional Expectation of the Duration in the Classical Ruin Problem. **Mathematics Magazine**, v. 48, n. 4, p. 200–203, 1 set. 1975.

The Solution as a Fair Game (Martingale). Em: DOYLE, P. G.; SNELL, J. L. (Eds.). **Random Walks and Electric Networks**. [s.l.] American Mathematical Soc., 1984. p. 11–14.

## Apêndice A - Comparação de um mesmo algoritmo em Julia e Python

Para validar nossa decisão de trocarmos de linguagem, comparamos o mesmo (ingênuo) algoritmo em Julia e Python que usamos para testar o estimador:

### Julia

```
using  LaTeXStrings, Random

function main()
  Random.seed!(1)
  duracoesSoma = 0
  M = 1_000_000
  p = 0.4
  teto = 100
  for i in 1:M
  saldo = 30
    duracao = 0
    while saldo > 0 && saldo < teto
      if p > rand()
        saldo += 2
      else
        saldo -= 1
      end
      duracao += 1
    end
    duracoesSoma += duracao
  end
  display(LaTeXString("Média da duração de $M passeios:
                       \$ $(round(duracoesSoma/M, digits = 2))\$"))
  display(LaTeXString("Tempo de execução em segundos:"))
end

@elapsed main()
```

Média da duração de 1000000 passeios: 350.29

Tempo de execução em segundos:

```
0.588439567
```

### Python

```python
import random
import time


def main():
    random.seed(1)
    duracoes = 0
    M = 1_000_000
    p = 0.4
    teto = 100
    for i in range(M):
        saldo = 30
        duracao = 0
        while saldo > 0 and saldo < teto:
            if p > random.random():
                saldo += 2
            else:
                saldo -= 1
            duracao += 1
        duracoes += duracao
    media = duracoes / M
    print(f"Média da duração de {M} passeios: {media:.2f}")


start_time = time.time()
main()
print(f"Tempo de execução: {time.time() - start_time} segundos")
```

```
Média da duração de 1000000 passeios: 350.45
Tempo de execução: 38.35879731178284 segundos
```

Para a simulação em Python nesse artigo, foi utilizado a biblioteca PyCall que nos permite criar um ambiente Python no Julia, possibilitando rodar código Python nessa linguagem o que foi necessário para inclusão de sua saída exata no relatório. A performance pode ter sido afetada minimamente nos tempos de chamada ao Kernel de execução do Python.

### Decisão de mudança

Desconsiderando o curto tempo de compilação de aproximadamente segundos do tempo total do código em Julia, que só precisa ser feito uma vez, temos que essa linguagem costuma ser consideravelmente mais rápida do que o Python, o que foi crucial nessas simulações e justificou nossas mudanças como previsto por resultados anteriores de GODOY et al. (2023)

## Apêndice B - Código de simulação com regras mais complexas

```
using LaTeXStrings, Random

function main()
  Random.seed!(1)
  duracoesSoma = 0
  M = 10_000_000
  p1 = 1/3
  p2 = 2/3
  teto = 100
  for i in 1:M
  saldo = 30
    duracao = 0
    while saldo > 0 && saldo < teto
      moeda = rand()
      if moeda < p2
        if moeda < p1
          saldo += 1
        else
          saldo += 3
        end
      else
        saldo -= 1
      end
      duracao += 1
    end
    duracoesSoma += duracao
  end
  display(LaTeXString("Média da duração de $M passeios:
                       \$ $(round(duracoesSoma/M, digits = 2))\$"))
end
main()
```

Média da duração de 10000000 passeios: 70.69