Tópicos em probabilidade (Arquivo 1)

Doriedson Rodrigues Neres Junior

19/03/2024

Problema 1:

Dado um número m produzir um vetor (e_1, e_2, ..., e_m) onde e_m é o nº de divisores até o número m. Em cada versão verificar m onde o algoritmo demora mais de um minuto para processar, m = 10^6, m = 10^7...

In [1]:
```
import time
```

Funções

In [2]:
```
def d1(n):
    """
    Algoritmo 1:
    O mais simples, dividir por todos.
    """

    #divisores = []
    for numeros in range(1, round(n) + 1):
        resto = n % numeros
        #if resto == 0:
        #    divisores.append(numeros)
        #    continue

def d2(n):
    """
    Algoritmo 2:
    Dividindo até a metade de 'n'.
    """
    for numeros in range(1, round(n/2) + 1):
        resto = n % numeros

def d3(n):
    """
    Algoritmo 3:
    Dividindo até a raiz quadrada de 'n'.
    """
    for numeros in range(1, round(n**(1/2)) + 1):
        resto = n % numeros

def d4(n):
    """
    Algoritmo 4:
    Supostamente, o melhor. Divide até a raiz quadrada de 'n', pulando de
    dois em dois.
    """
    if n%2 == 0: p_i = 2
    else: p_i = 3
```

```
    for numeros in range(p_i, round(n**(1/2)) + 1, 2):
        resto = n % numeros

def algoritmos(k, N=10**5):
    """
    Recebe 'k' (o código do algoritmo) e 'N' (o número inicial) definido, por padrã
    como 10^5. Faz os cálculos e computa o tempo.
    """

    if k == 1: algoritmo = d1
    if k == 2: algoritmo = d2
    if k == 3: algoritmo = d3
    if k == 4: algoritmo = d4

    while True:
        ini = time.time()
        algoritmo(N)
        fim = time.time()

        tempo = fim - ini    # em segundos

        if tempo < 5:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 10

        if tempo >= 5 and tempo < 10:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 7

        if tempo >= 10 and tempo < 20:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 5

        if tempo >= 20 and tempo < 30:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 2

        if tempo >= 30 and tempo < 45:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 1.2

        if tempo >= 45 and tempo < 60:
            print("De boa", '{:,}'.format(N), f", pois demorou {round(tempo, 3)}s."
            N *= 1.1

        if tempo >= 60:
            print("\n\n\nProblema!\nO número", '{:e}'.format(N), f"demorou {round(t
            f"no algoritmo {k}.\n")
            break
```

In [5]:
```
%%time
algoritmos(1)
```

```
De boa 100,000 , pois demorou 0.006s.
De boa 1,000,000 , pois demorou 0.059s.
De boa 10,000,000 , pois demorou 0.621s.
De boa 100,000,000 , pois demorou 6.301s.
De boa 700,000,000 , pois demorou 44.456s.
De boa 840,000,000.0 , pois demorou 58.789s.

Problema!
O número 924,000,000.0000001 demorou 63.817s
 no algoritmo 1.
CPU times: total: 2min 24s
Wall time: 2min 54s
```

In [4]:
```
%%time
algoritmos(2)
```

```
De boa 100,000 , pois demorou 0.004s.
De boa 1,000,000 , pois demorou 0.027s.
De boa 10,000,000 , pois demorou 0.209s.
De boa 100,000,000 , pois demorou 2.011s.
De boa 1,000,000,000 , pois demorou 20.582s.
De boa 2,000,000,000 , pois demorou 43.63s.
De boa 2,400,000,000.0 , pois demorou 57.786s.

Problema!
O número 2,640,000,000.0 demorou 67.495s
 no algoritmo 2.
CPU times: total: 0 ns
Wall time: 0 ns
```

In [6]:
```
%%time
algoritmos(3)
```

```
De boa 100,000 , pois demorou 0.0s.
De boa 1,000,000 , pois demorou 0.0s.
De boa 10,000,000 , pois demorou 0.0s.
De boa 100,000,000 , pois demorou 0.001s.
De boa 1,000,000,000 , pois demorou 0.002s.
De boa 10,000,000,000 , pois demorou 0.007s.
De boa 100,000,000,000 , pois demorou 0.022s.
De boa 1,000,000,000,000 , pois demorou 0.074s.
De boa 10,000,000,000,000 , pois demorou 0.232s.
De boa 100,000,000,000,000 , pois demorou 0.643s.
De boa 1,000,000,000,000,000 , pois demorou 2.227s.
De boa 10,000,000,000,000,000 , pois demorou 6.671s.
De boa 70,000,000,000,000,000 , pois demorou 19.438s.
De boa 350,000,000,000,000,000 , pois demorou 43.439s.
De boa 4.2e+17 , pois demorou 44.541s.
De boa 5.04e+17 , pois demorou 49.496s.
De boa 5.5440000000000006e+17 , pois demorou 50.864s.
De boa 6.098400000000001e+17 , pois demorou 53.098s.
De boa 6.708240000000003e+17 , pois demorou 56.628s.
De boa 7.379064000000004e+17 , pois demorou 58.571s.

Problema!
O número 8.116970400000005e+17 demorou 62.262s
 no algoritmo 3.
CPU times: total: 6min 21s
Wall time: 7min 28s
```

In [8]:
```
%%time
algoritmos(4)
```

```
De boa 100,000 , pois demorou 0.0s.
De boa 1,000,000 , pois demorou 0.0s.
De boa 10,000,000 , pois demorou 0.001s.
De boa 100,000,000 , pois demorou 0.0s.
De boa 1,000,000,000 , pois demorou 0.002s.
De boa 10,000,000,000 , pois demorou 0.004s.
De boa 100,000,000,000 , pois demorou 0.012s.
De boa 1,000,000,000,000 , pois demorou 0.034s.
De boa 10,000,000,000,000 , pois demorou 0.11s.
De boa 100,000,000,000,000 , pois demorou 0.355s.
De boa 1,000,000,000,000,000 , pois demorou 1.072s.
De boa 10,000,000,000,000,000 , pois demorou 3.421s.
De boa 100,000,000,000,000,000 , pois demorou 10.602s.
De boa 500,000,000,000,000,000 , pois demorou 24.246s.
De boa 1,000,000,000,000,000,000 , pois demorou 34.395s.
De boa 1.2e+18 , pois demorou 38.201s.
De boa 1.44e+18 , pois demorou 45.753s.
De boa 1.584e+18 , pois demorou 50.678s.
De boa 1.7424000000000003e+18 , pois demorou 54.881s.
De boa 1.9166400000000005e+18 , pois demorou 59.865s.

Problema!
O número 2.1083040000000008e+18 demorou 65.642s
 no algoritmo 4.
CPU times: total: 5min 31s
Wall time: 6min 29s
```

Resultados

Naturalmente, o tempo de execução muda um pouco a cada repetição dos cálculos, contudo, tratam-se de mudanças pequenas. A tabela a seguir resume a capacidade de cada variação do algoritmo dado um referencial de 60 segundos.

Algoritmo	Valor	Tempo (s)
Um	9.24e+08	63.81
Dois	2.64e+09	67.49
Três	8.11e+17	62.26
Quatro	2.10e+18	65.64

21/03/2024

Problema 2

Numa urna inicialmente vazia, serão adicionadas bolas verdes ou vermelhas. Para saber a quantidade de bolas adicionadas será lançados um dado honesto para bolas verdes e um dado viciado p/ bolas vermelhas. Antes disso, para saber a cor, será lançada uma moeda honesta.

- Dado viciado: P_1 = P_6 = 1/5 P_2 = P_3 = P_4 = P_5 = 3/20

Ao longo do tempo, qual a proporção das cores na urna?

In [2]:
```
import pandas as pd
import random as rd
import math

import matplotlib.pyplot as plt
import numpy as np
import matplotlib as mpl
```

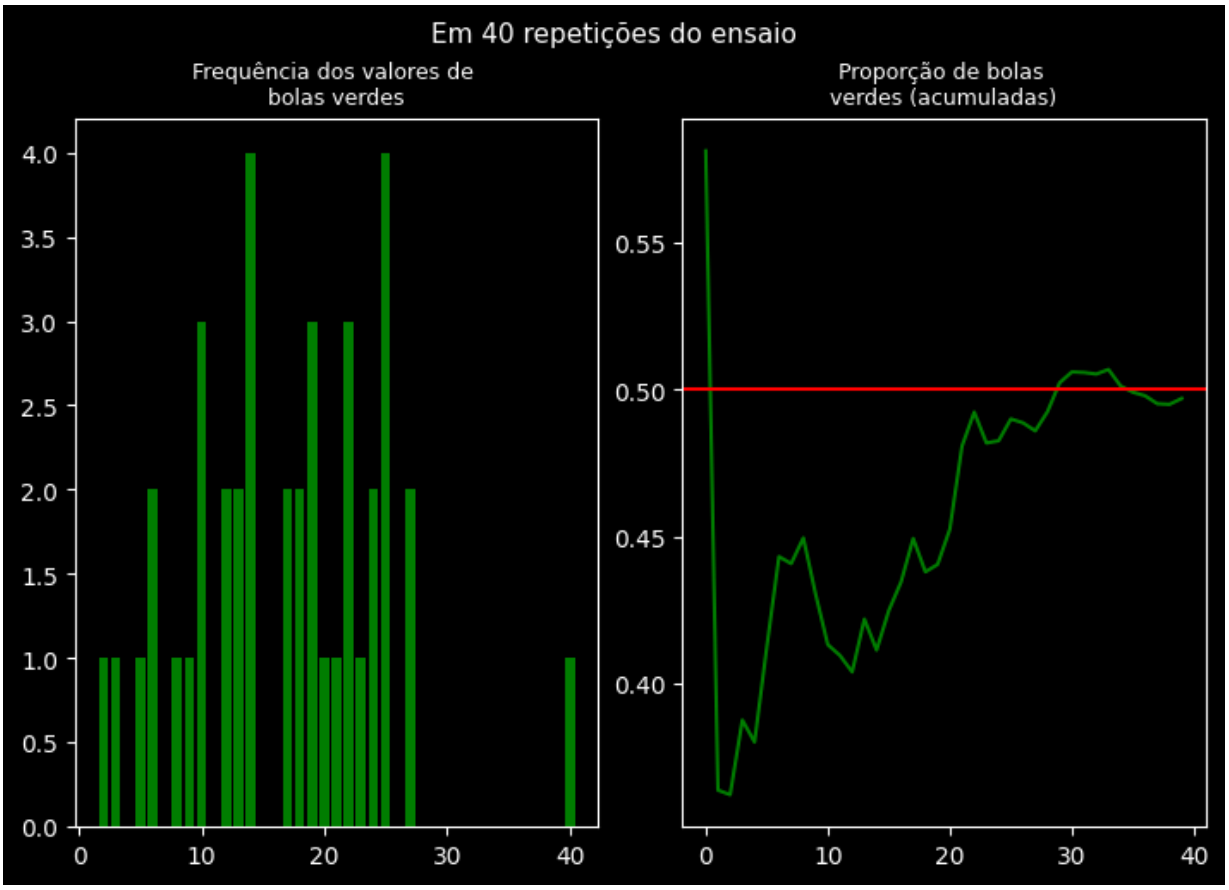
In [1]:
```
# FUNÇÕES
def dadoo(p1=1/6, p2=1/6, p3=1/6, p4=1/6, p5=1/6, p6=1/6):
    """
    Recebe as probabilidades de cada face em um dado.
    O padrão é um dado honesto.
    """
    n = rd.uniform(0, 1)

    s6 = math.fsum([p1, p2, p3, p4, p5, p6])
    s5 = math.fsum([p1, p2, p3, p4, p5])
    s4 = math.fsum([p1, p2, p3, p4])
    s3 = math.fsum([p1, p2, p3])
    s2 = math.fsum([p1, p2])

    if round(s6) == 1:
        if n < p1: face = 1
        if p1 < n and n < s2: face = 2
        if s2 < n and n < s3: face = 3
        if s3 < n and n < s4: face = 4
        if s4 < n and n < s5: face = 5
        if s5 < n and n < s6: face = 6
    else:
        print("A soma das probabilidades tem de ser 1 (um).")

    return face


def ensaio(k=10):
    """
```

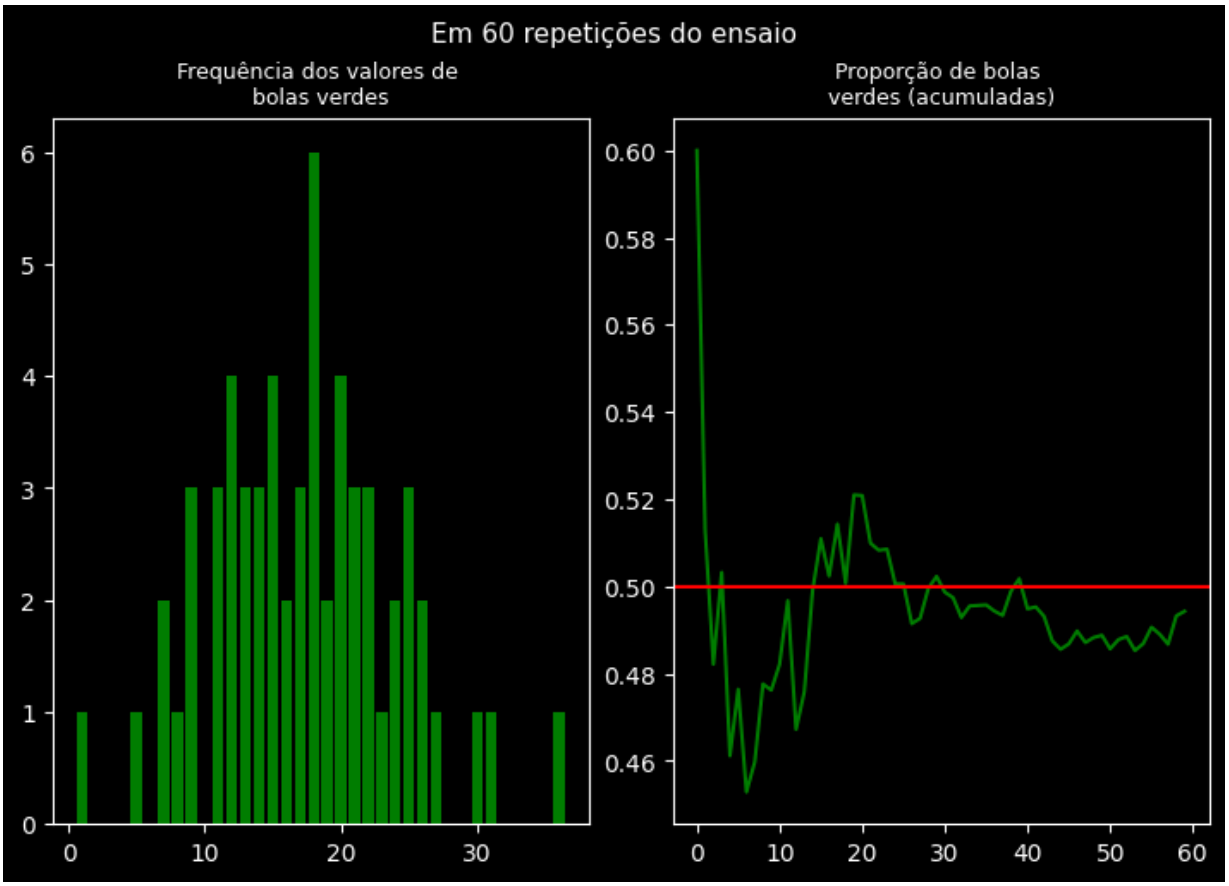
```
    Realiza um experimento, sendo ele 'k' repetições do lançamento de uma moeda
    e um dado.
    Utiliza 'dadoo()'.
    """
    urna = []
    for i in range(k):
        moeda = rd.randint(0, 1)    # Binomial
        if moeda == 0:
            numero = dadoo(.2, 3/20, 3/20, 3/20, 3/20, .2)    # Viciado
            for i in range(numero): urna.append('R')
        else:
            numero = dadoo()    # honesto
            for i in range(numero): urna.append('G')

    return urna


def repetir(k):
    """
    Faz 'k' repetições do 'ensaio()' e armazena os resultados.
    """
    tverdes, ttotal = [], []
    for i in range(k):
        res = pd.DataFrame(ensaio())
        conting = res.value_counts()

        try:
            v = conting['G']
        except KeyError:
            v = 0
            tverdes.append(v)
        else:
            tverdes.append(v)

        try:
            verm = conting['R']
        except KeyError:
            verm = 0
            ttotal.append(v + verm)
        else:
            ttotal.append(v + verm)

    resposta = pd.DataFrame({"Verdes": tverdes, "Total": ttotal})

    return resposta


def grafs(data):
    """
    Desenha o gráfico de barras dado um DataFrame do Pandas e também o gráf.
    da proporção acumulada, de bolas verdes, após todas repetições do ensaio.
    """
    conte = data["Verdes"].value_counts()

    plt.style.use('dark_background')
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7, 5), layout='constrained')
```

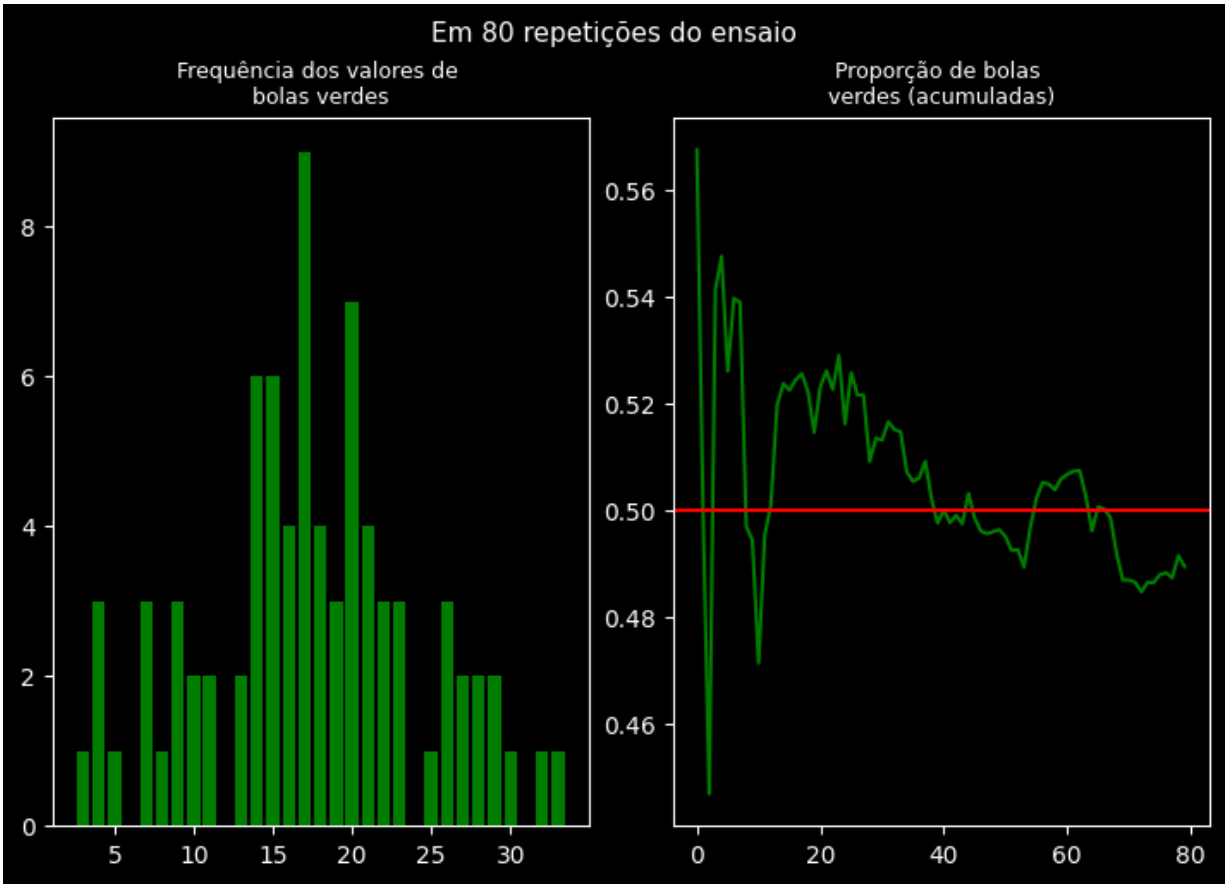
```
    ax1.bar(conte.index.tolist(), conte.tolist(), color='green')
    ax1.set_title("Frequência dos valores de \nbolas verdes",
                  {'fontsize': 9})

    cumulativo = data.cumsum()
    cumulativo["Verdes prop."] = cumulativo["Verdes"]/cumulativo["Total"]
    ax2.plot(cumulativo["Verdes prop."], color="green")
    plt.axhline(.5, color="red")
    #plt.ylabel("Núm")
    ax2.set_title("Proporção de bolas \nverdes (acumuladas)",
                  {'fontsize': 9})

    fig.suptitle(f"Em {len(cumulativo)} repetições do ensaio",
                 fontsize=11)
```

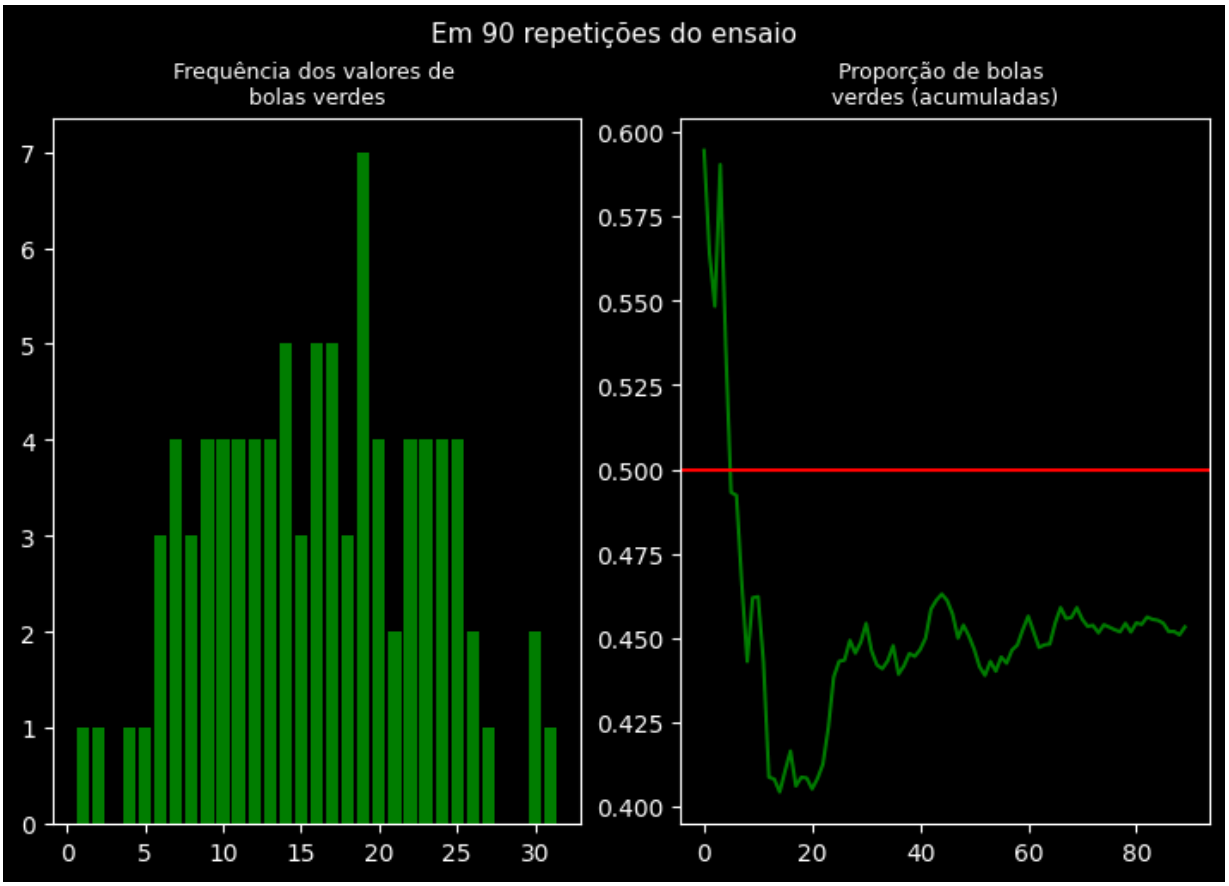
Parte 1:

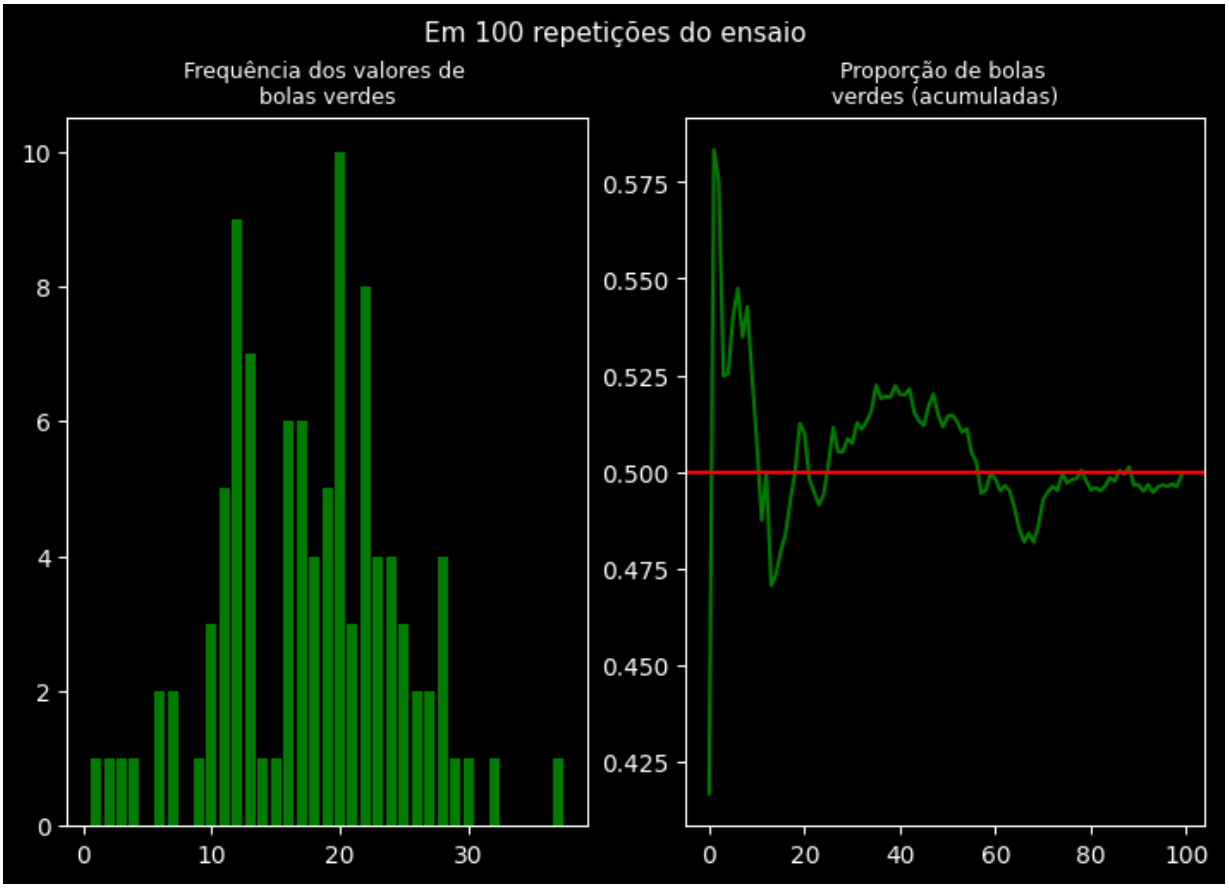
In [4]:
```
valores = [40, 60, 80, 90, 100, 300, 500, 700, 1000, 2000, 3000, 4000, 5000]

for v in valores:
    data = repetir(v)
    grafs(data)
```

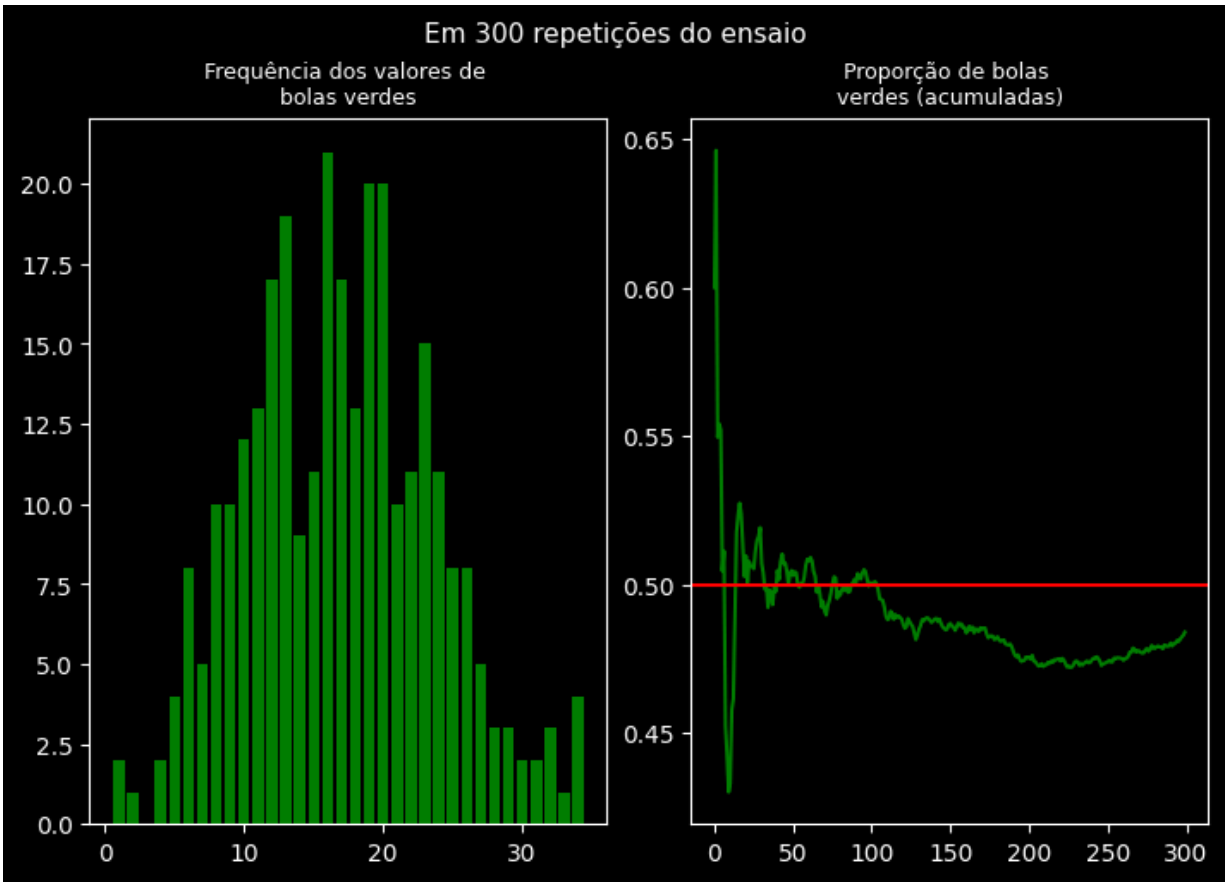


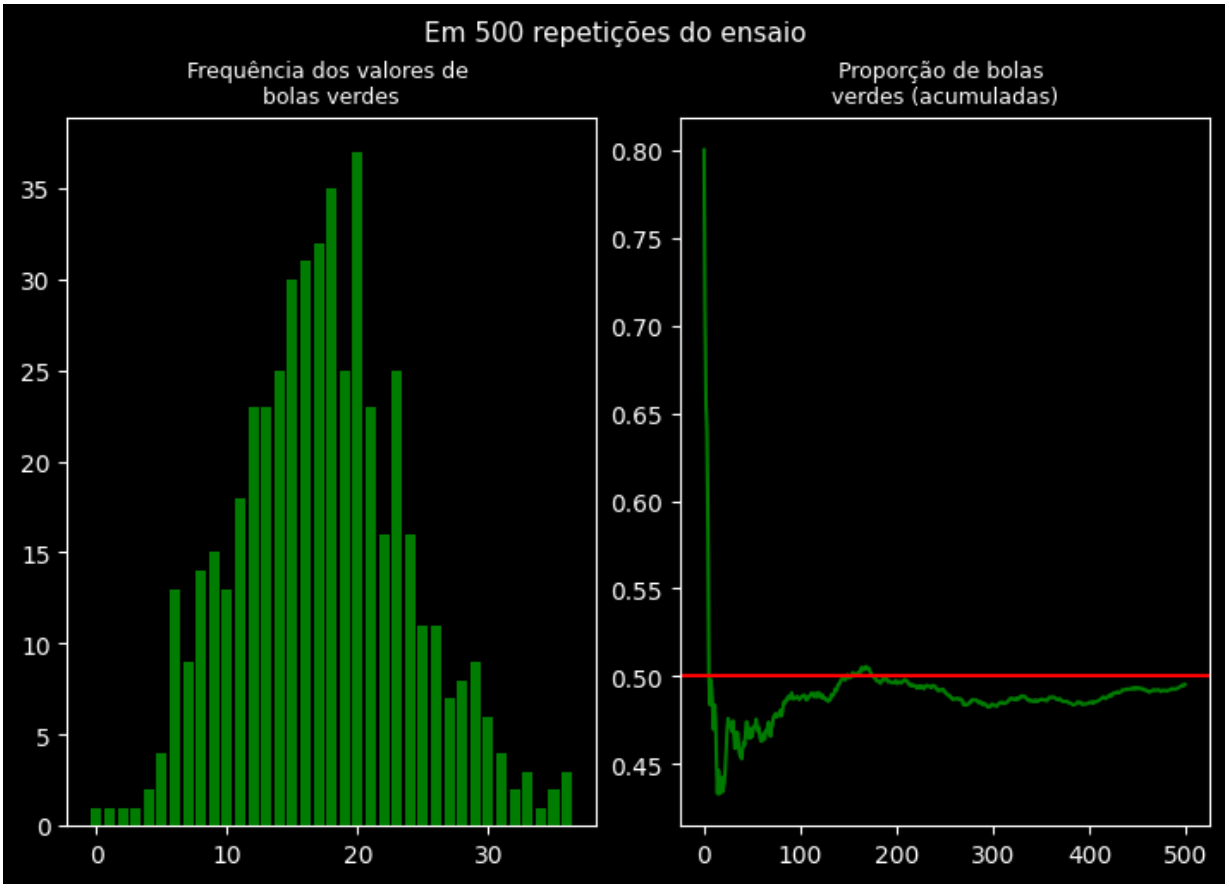

Em 60 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
1
2
3
4
5
6
0
10
20
30
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
0
10
20
30
40
50
60

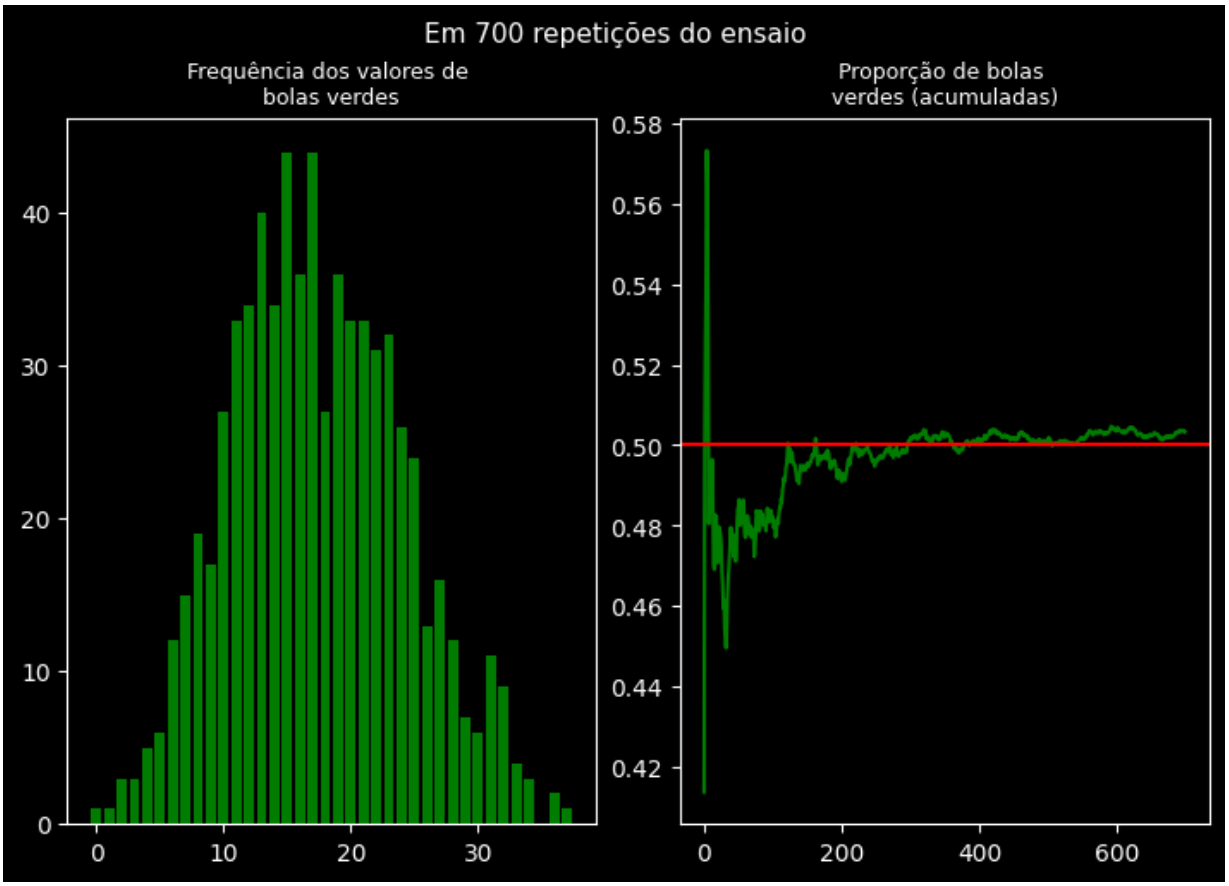

Em 80 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
2
4
6
8
5
10
15
20
25
30
0.46
0.48
0.50
0.52
0.54
0.56
0
20
40
60
80

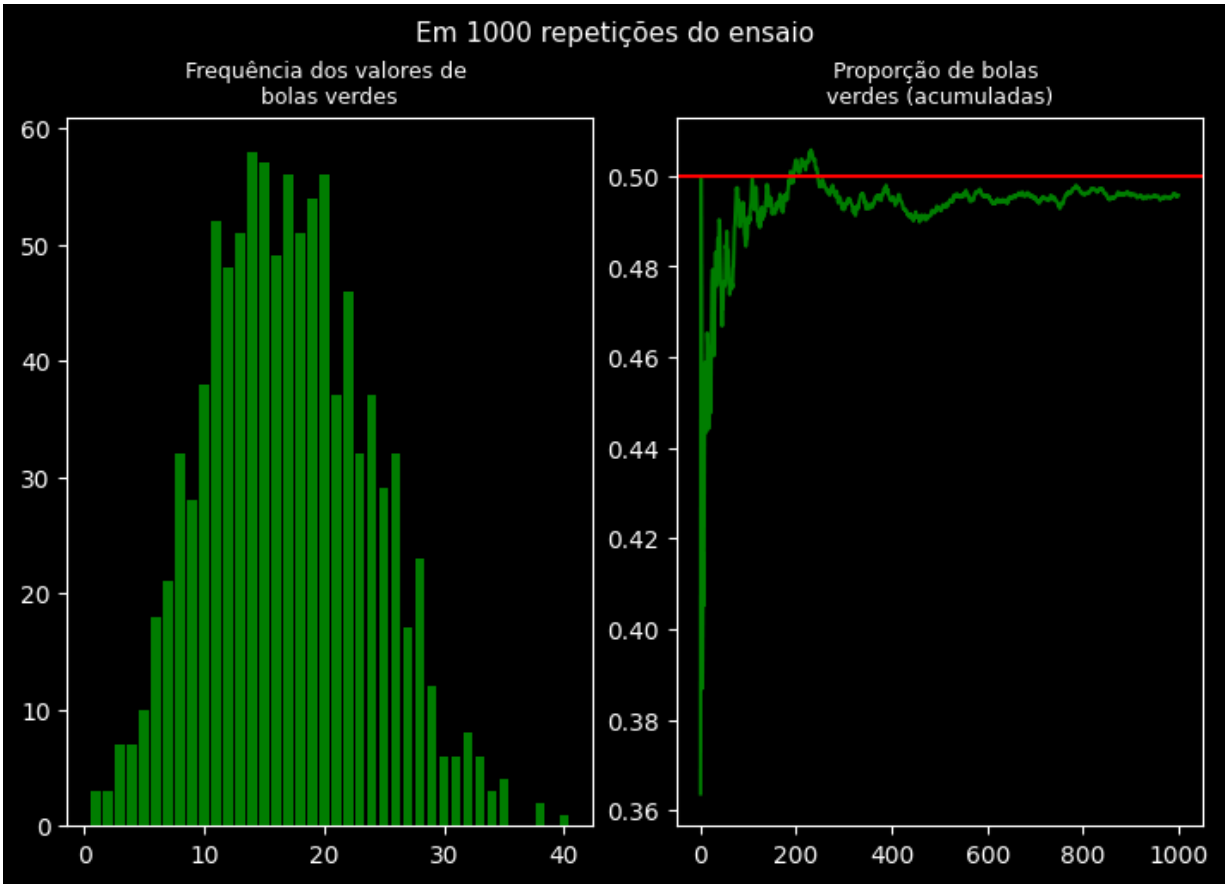

Em 90 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
0.400
0.425
0.450
0.475
0.500
0.525
0.550
0.575
0.600
0
20
40
60
80

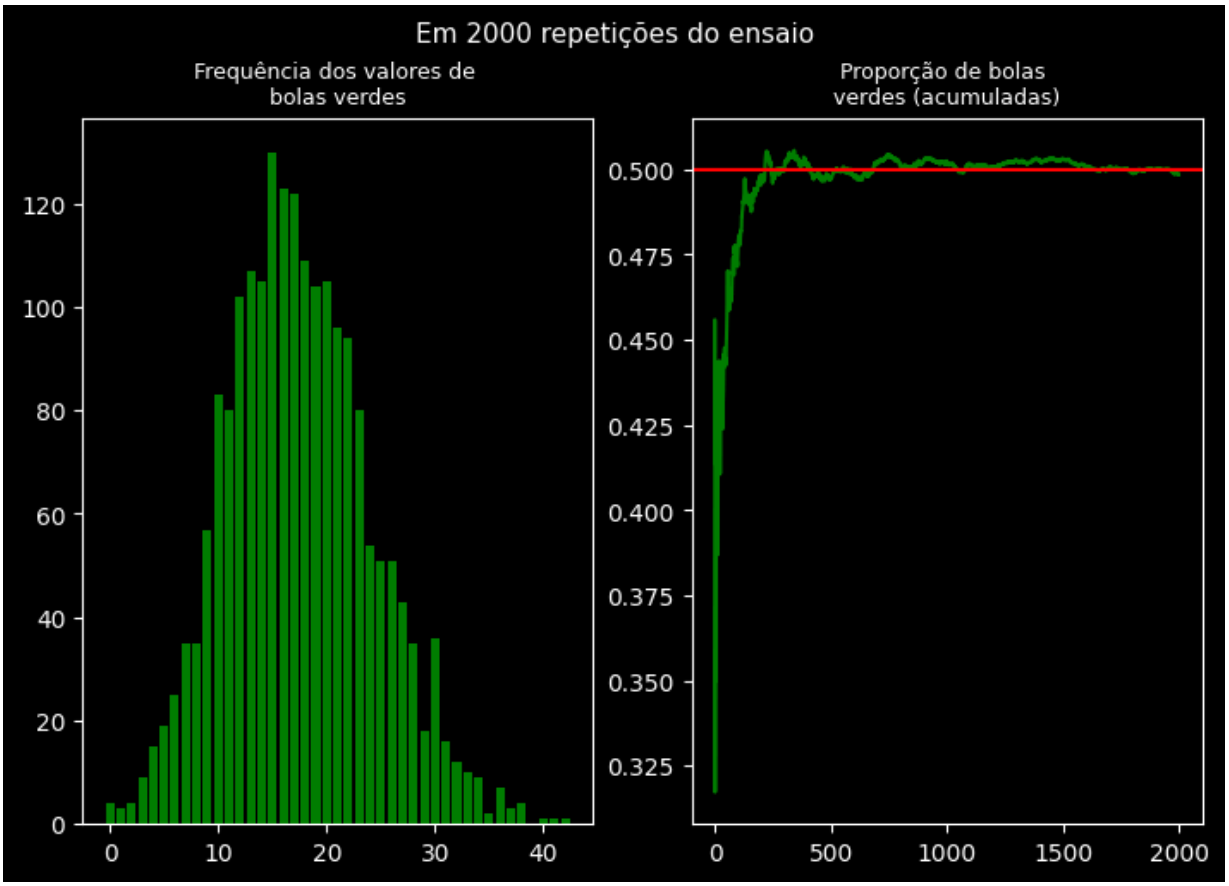

Em 100 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
2
4
6
8
10
0
10
20
30
0.425
0.450
0.475
0.500
0.525
0.550
0.575
0
20
40
60
80
100

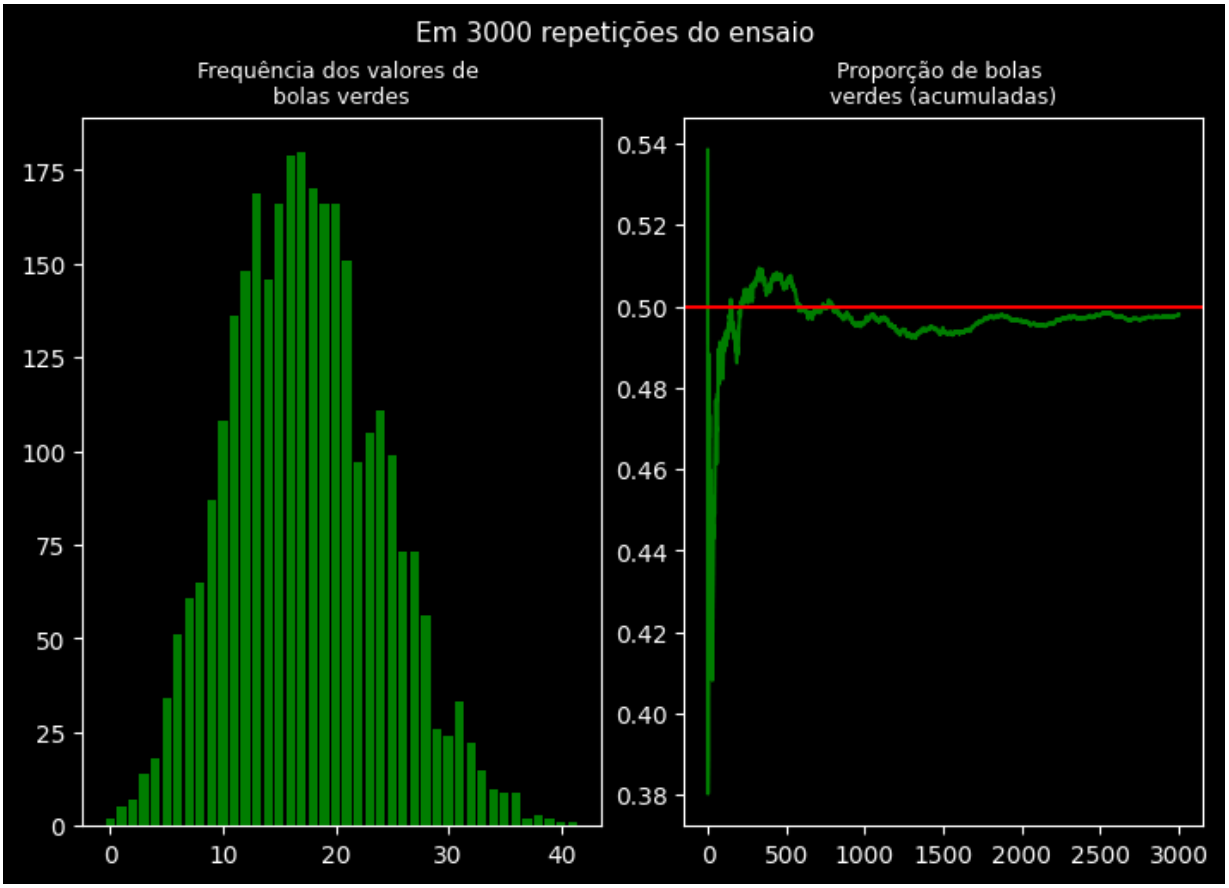

Em 300 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
10
20
30
0.45
0.50
0.55
0.60
0.65
0
50
100
150
200
250
300

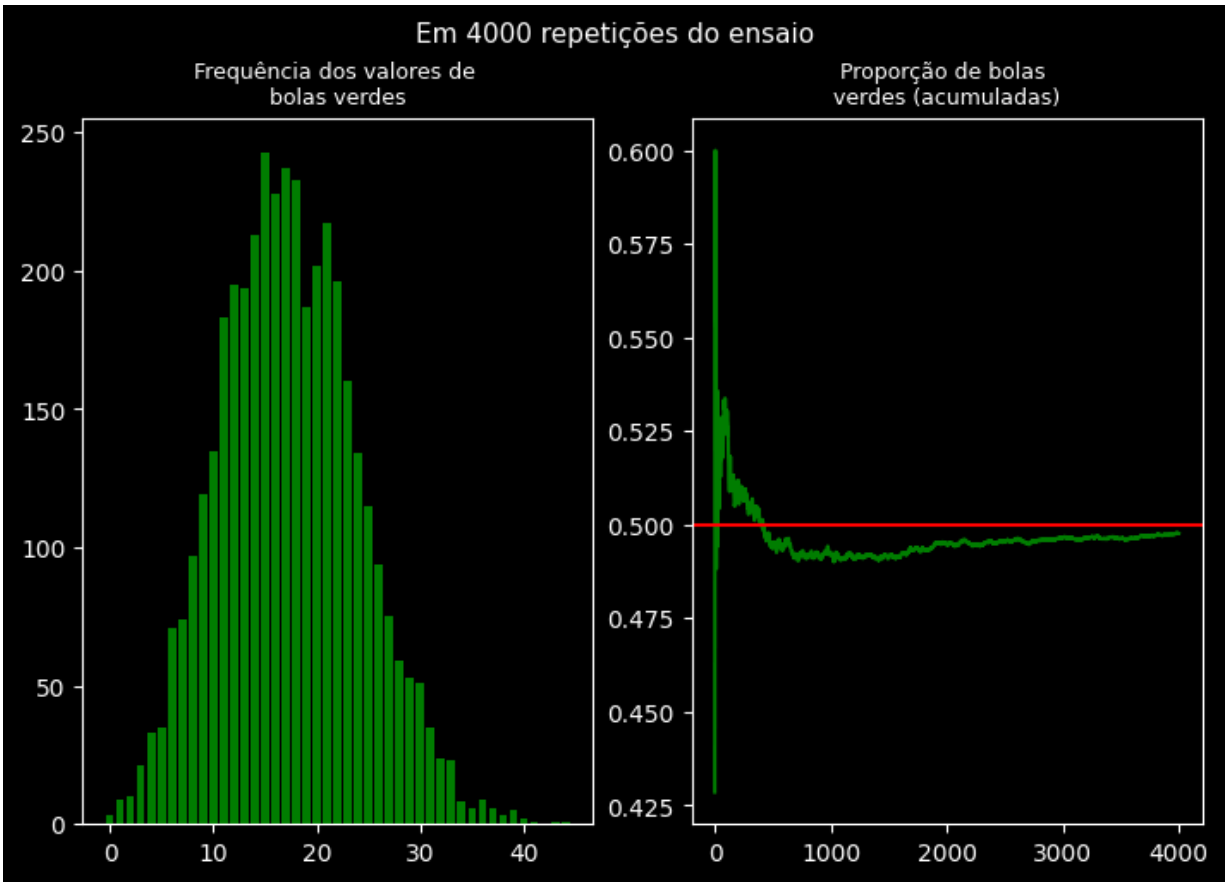

Em 500 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
5
10
15
20
25
30
35
0
10
20
30
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
0
100
200
300
400
500

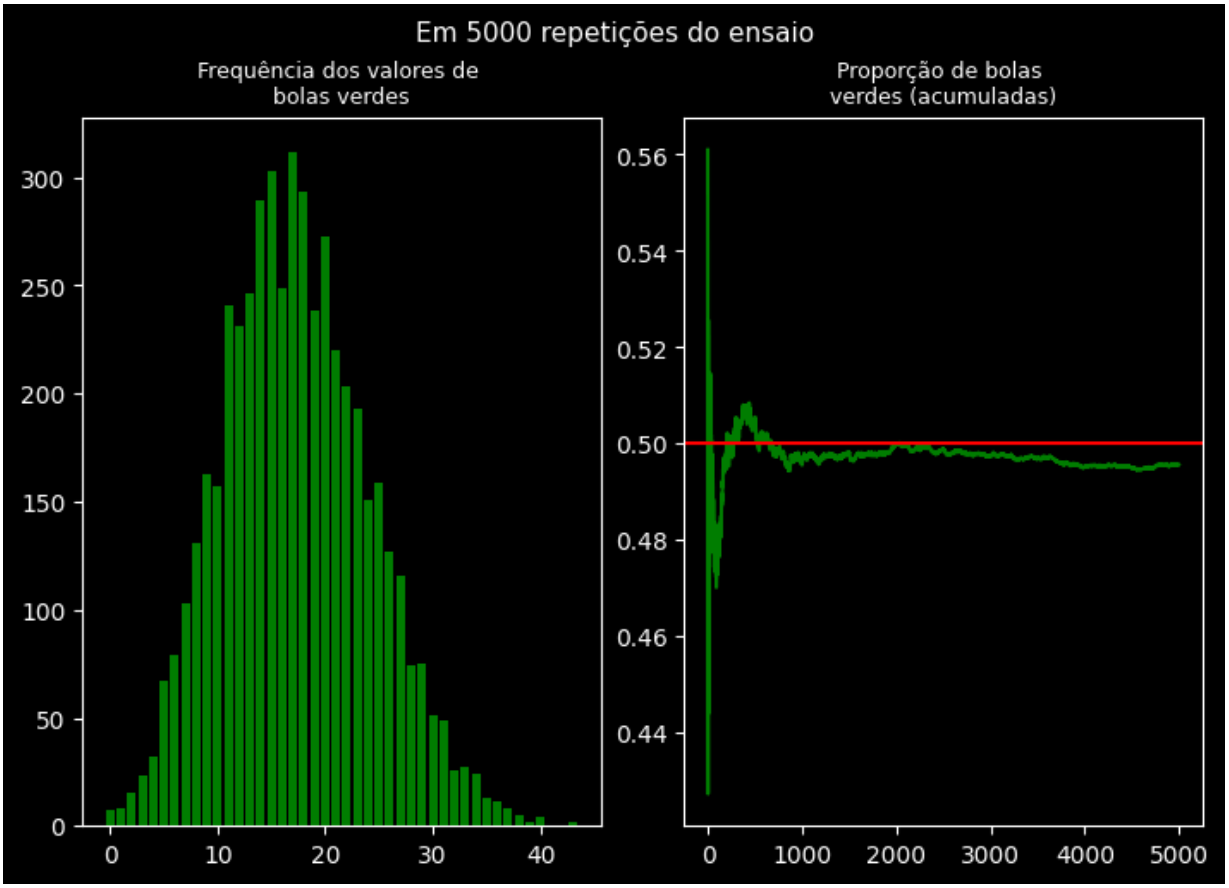

Em 700 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
10
20
30
40
0.42
0.44
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0
200
400
600


Em 1000 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
10
20
30
40
50
60
0.36
0.38
0.40
0.42
0.44
0.46
0.48
0.50
0
200
400
600
800
1000


Em 2000 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
20
40
60
80
100
120
0
10
20
30
40
0.325
0.350
0.375
0.400
0.425
0.450
0.475
0.500
0
500
1000
1500
2000


Em 3000 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
25
50
75
100
125
150
175
0
10
20
30
40
0.38
0.40
0.42
0.44
0.46
0.48
0.50
0.52
0.54
0
500
1000
1500
2000
2500
3000


Em 4000 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
50
100
150
200
250
0
10
20
30
40
0.425
0.450
0.475
0.500
0.525
0.550
0.575
0.600
0
1000
2000
3000
4000


Em 5000 repetições do ensaio
Frequência dos valores de bolas verdes
Proporção de bolas verdes (acumuladas)
0
50
100
150
200
250
300
0
10
20
30
40
0.44
0.46
0.48
0.50
0.52
0.54
0.56
0
1000
2000
3000
4000
5000

Resultados

Sabendo do Teorema Central do Limite, a simulação seguiu o desfecho esperado. A soma da repetição de ensaios discretos convergiu para $p = \frac{1}{2}$, que é a chance de sair n bolas verdes. Independentemente das probabilidades associadas a cada face, sendo o dado honesto ou não, um grande número de repetições, nesse caso, sempre convergirá para p (a média da Bernoulli associada ao lançamento da moeda).

Parte 2

In [4]:
```
# Novas funções
def ensaio2(k=100, dg=[1/6 for x in range(6)],
            dr=[1/6 for x in range(6)]):
    """
    Realiza um experimento, sendo ele 'k' repetições do lançamento de uma moeda
    e um dado.
    Utiliza 'dadoo()'.
    Recebe uma lista com as probabilidades das faces de cada dado.
    """
    if k < 100: k = 100
    urna = []
    for i in range(k*k):
        moeda = rd.randint(0, 1)    # Binomial
        if moeda == 0:
            numero = dadoo(dg[0], dg[1], dg[2], dg[3], dg[4], dg[5])
            for i in range(numero): urna.append('G')
        else:
            numero = dadoo(dr[0], dr[1], dr[2], dr[3], dr[4], dr[5])
            for i in range(numero): urna.append('R')

    return urna

def Munif(probs=[1/6 for i in range(6)]):
    """
    Calcula a média de uma distribuição uniforme discreta, dado uma lista com
    as respectivas probabilidades.
    """
    if round(sum(probs)) != 1: print("A soma das probabilidades deve ser um.")

    x = 1
    media = 0
    for valor in probs:
        media += x * valor
        x += 1

    return media
```

In [5]:
```
dadoverm = [3/8, 3/8, 1/16, 1/16, 1/16, 1/16]    # Grande chance de sair 1 ou 2.
dadoverd = [1/16, 1/16, 1/16, 1/16, 3/8, 3/8]    # Grande chance de sair 5 ou 6.

resp = ensaio2(k=1000, dg=dadoverd, dr=dadoverm)

respdf = pd.DataFrame(resp)
```

```
A = respdf.value_counts()
#print(A)

prop_est = A["G"]/(A["G"]+A["R"])
media = Munif(dadoverd)/(Munif(dadoverd)+Munif(dadoverm))

print(f"\nA proporção de bolas verdes, estimada pela simulação, foi: {round(prop_es
     f"\n\nEnquanto a proporção da média do dado verde* foi {round(media, 4)}")

#math.fsum([media, prop_est])
```

```
A proporção de bolas verdes, estimada pela simulação, foi: 0.6786.

Enquanto a proporção da média do dado verde* foi 0.6786
```

* Foi considerada a seguinte equação:

$$prop.\ \ média\ := \frac{E[X_i]}{E[X_i]+E[Y_i]}$$

onde, $X_i :=$ quantidade de bolas verdes

$Y_i :=$ quantidade de bolas vermelhas

In [ ]:

26/03/2024

Problema 3:

Montar uma tabela da distribuição normal a partir da aplicação do TCL em uma série de ensaios de Bernoulli e da aproximação numérica da integral da f.d.p da normal.

Teorema Central do Limite:

Dado $X_i's$ variáveis aleatórias i.i.d, com $E[X_i]$ finita e $Var(X_i)$ finita, não-nula.

$S_n = X_1 + X_2 + \ldots + X_n$

$Z_n = \frac{S_n - E[S_n]}{\sqrt{Var(S_n)}} = \frac{S_n - n \cdot E[X_1]}{\sqrt{n \cdot Var(X_1)}} \xrightarrow[n \to \infty]{d} N(0,1)$

No caso de X_i ter distribuição Bernouli com parâmetro p:

$(\Sigma_{i=1}^{n} S_n) \sim \ Binomial(n,p)$

$\therefore Z_n = \frac{S_n - np}{\sqrt{np(1-p)}}$

Aplicando o T.C.L:

$\mathbb{P}(Z_n \leq z) \approx \Phi(z)$

onde $\Phi(z)$ é, naturalmente, a área sob o gráfico da Normal(0, 1) até o ponto z.

Como $\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-x^2} dx$ não possui solução analítica, segue a numérica.

In [1]:
```
import numpy as np
import pandas as pd
import itertools
from scipy.stats import norm
import matplotlib.pyplot as plt
```

Funções:

In [2]:
```
# Não tem o attribute 'batched' no meu itertools
def batched(iterable, n):
    # batched('ABCDEFG', 3) → ABC DEF G
    if n < 1:
        raise ValueError('n must be at least one')
    it = iter(iterable)
    while batch := tuple(itertools.islice(it, n)):
        yield batch

def _(n=10**6, m=900, p=.5):
```

```
    """
    Estima a tabela da normal atráves de aproximações numéricas dadas
    pelo T.C.L.
    Recebe 'n', número de repetições,'m' e 'p' que são os parâmetros
    da binomial.
    """
    prob = []
    contador = [0 for x in range(400)]

    for i in range(n):
        aux = np.random.binomial(m, p)
        minimoz = (aux - m*p)/((m*p*(1-p))**(1/2))
        if minimoz < 0:
            minimo = 0
        else: pass

        pos_vet = -1 + minimoz * 100
        pos_vet = max(0, pos_vet)
        #prob.append(pos_vet)
        for j in range(round(pos_vet), 400):
            contador[j] += 1

    prob = [valor/n for valor in contador]

    return prob


def tabe(valores):
    """
    Converte a saída de '_()' para tuplas, depois para lista, e cria uma
    tabela da normal.
    Recebe a saída da função '_()'.
    """
    tuplas = list(batched(valores, 10))    # Lista de tuplas
    lista_lista = [list(t) for t in tuplas]

    t = pd.DataFrame(
        lista_lista,
        columns = [x/100 for x in range(10)],
        index = [x/10 for x in range(40)]
        )

    return t


def erros(tabela_normal):
    """
    Calcula o erro absoluto de cada valor estimado pelo T.C.L.; desenha o
    gráfico.
    """
    tab = tabela_normal
    indice = [i for i in range(40)]
    coluna = [j for j in range(10)]

    erros = []
    for linha in indice:
```

```
        for col in coluna:
            est = tab.loc[linha/10, col/100]
            prova = norm.cdf((col/100) + (linha/10))
            e = prova - est

            erros.append(abs(e))

    print("\nA média dos erros absolutos foi:", np.mean(erros))

    plt.style.use('dark_background')
    fig, ax = plt.subplots(figsize=(5, 5))
    ax.plot(erros, color="#bc1cd9")
    ax.set_title("Evolução do erro absoluto dos valores aproximados")
```

In [4]:
```
%%time
a=_()
tabela1 = tabe(a)
```

```
CPU times: total: 30.9 s
Wall time: 49.2 s
```

In [5]:
```
tabela1
```

Out[5]:

	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.0
0.0	0.513360	0.513360	0.513360	0.513360	0.513360	0.513360	0.539946	0.539946	0.53994
0.1	0.539946	0.539946	0.566258	0.566258	0.566258	0.566258	0.566258	0.566258	0.56625
0.2	0.592229	0.592229	0.592229	0.592229	0.592229	0.592229	0.618296	0.618296	0.61829
0.3	0.618296	0.618296	0.643117	0.643117	0.643117	0.643117	0.643117	0.643117	0.64311
0.4	0.667613	0.667613	0.667613	0.667613	0.667613	0.667613	0.691873	0.691873	0.69187
0.5	0.691873	0.691873	0.714712	0.714712	0.714712	0.714712	0.714712	0.714712	0.71471
0.6	0.736718	0.736718	0.736718	0.736718	0.736718	0.736718	0.757808	0.757808	0.75780
0.7	0.757808	0.757808	0.777931	0.777931	0.777931	0.777931	0.777931	0.777931	0.77793
0.8	0.797372	0.797372	0.797372	0.797372	0.797372	0.797372	0.815605	0.815605	0.81560
0.9	0.815605	0.815605	0.833038	0.833038	0.833038	0.833038	0.833038	0.833038	0.83303
1.0	0.849041	0.849041	0.849041	0.849041	0.849041	0.849041	0.864005	0.864005	0.86400
1.1	0.864005	0.864005	0.878148	0.878148	0.878148	0.878148	0.878148	0.878148	0.87814
1.2	0.891185	0.891185	0.891185	0.891185	0.891185	0.891185	0.902971	0.902971	0.90297
1.3	0.902971	0.902971	0.914089	0.914089	0.914089	0.914089	0.914089	0.914089	0.91408
1.4	0.923854	0.923854	0.923854	0.923854	0.923854	0.923854	0.932930	0.932930	0.93293
1.5	0.932930	0.932930	0.941308	0.941308	0.941308	0.941308	0.941308	0.941308	0.94130
1.6	0.948692	0.948692	0.948692	0.948692	0.948692	0.948692	0.955356	0.955356	0.95535
1.7	0.955356	0.955356	0.961149	0.961149	0.961149	0.961149	0.961149	0.961149	0.96114
1.8	0.966372	0.966372	0.966372	0.966372	0.966372	0.966372	0.971029	0.971029	0.97102
1.9	0.971029	0.971029	0.975213	0.975213	0.975213	0.975213	0.975213	0.975213	0.97521
2.0	0.978857	0.978857	0.978857	0.978857	0.978857	0.978857	0.981985	0.981985	0.98198
2.1	0.981985	0.981985	0.984765	0.984765	0.984765	0.984765	0.984765	0.984765	0.98476
2.2	0.987214	0.987214	0.987214	0.987214	0.987214	0.987214	0.989324	0.989324	0.98932
2.3	0.989324	0.989324	0.991037	0.991037	0.991037	0.991037	0.991037	0.991037	0.99103
2.4	0.992542	0.992542	0.992542	0.992542	0.992542	0.992542	0.993812	0.993812	0.99381
2.5	0.993812	0.993812	0.994922	0.994922	0.994922	0.994922	0.994922	0.994922	0.99492
2.6	0.995801	0.995801	0.995801	0.995801	0.995801	0.995801	0.996517	0.996517	0.99651
2.7	0.996517	0.996517	0.997174	0.997174	0.997174	0.997174	0.997174	0.997174	0.99717
2.8	0.997688	0.997688	0.997688	0.997688	0.997688	0.997688	0.998118	0.998118	0.99811
2.9	0.998118	0.998118	0.998488	0.998488	0.998488	0.998488	0.998488	0.998488	0.99848

	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.0
3.0	0.998769	0.998769	0.998769	0.998769	0.998769	0.998769	0.999019	0.999019	0.99901
3.1	0.999019	0.999019	0.999216	0.999216	0.999216	0.999216	0.999216	0.999216	0.99921
3.2	0.999370	0.999370	0.999370	0.999370	0.999370	0.999370	0.999495	0.999495	0.99949
3.3	0.999495	0.999495	0.999597	0.999597	0.999597	0.999597	0.999597	0.999597	0.99959
3.4	0.999684	0.999684	0.999684	0.999684	0.999684	0.999684	0.999756	0.999756	0.99975
3.5	0.999756	0.999756	0.999805	0.999805	0.999805	0.999805	0.999805	0.999805	0.99980
3.6	0.999849	0.999849	0.999849	0.999849	0.999849	0.999849	0.999882	0.999882	0.99988
3.7	0.999882	0.999882	0.999904	0.999904	0.999904	0.999904	0.999904	0.999904	0.99990
3.8	0.999936	0.999936	0.999936	0.999936	0.999936	0.999936	0.999956	0.999956	0.99995
3.9	0.999956	0.999956	0.999967	0.999967	0.999967	0.999967	0.999967	0.999967	0.99996

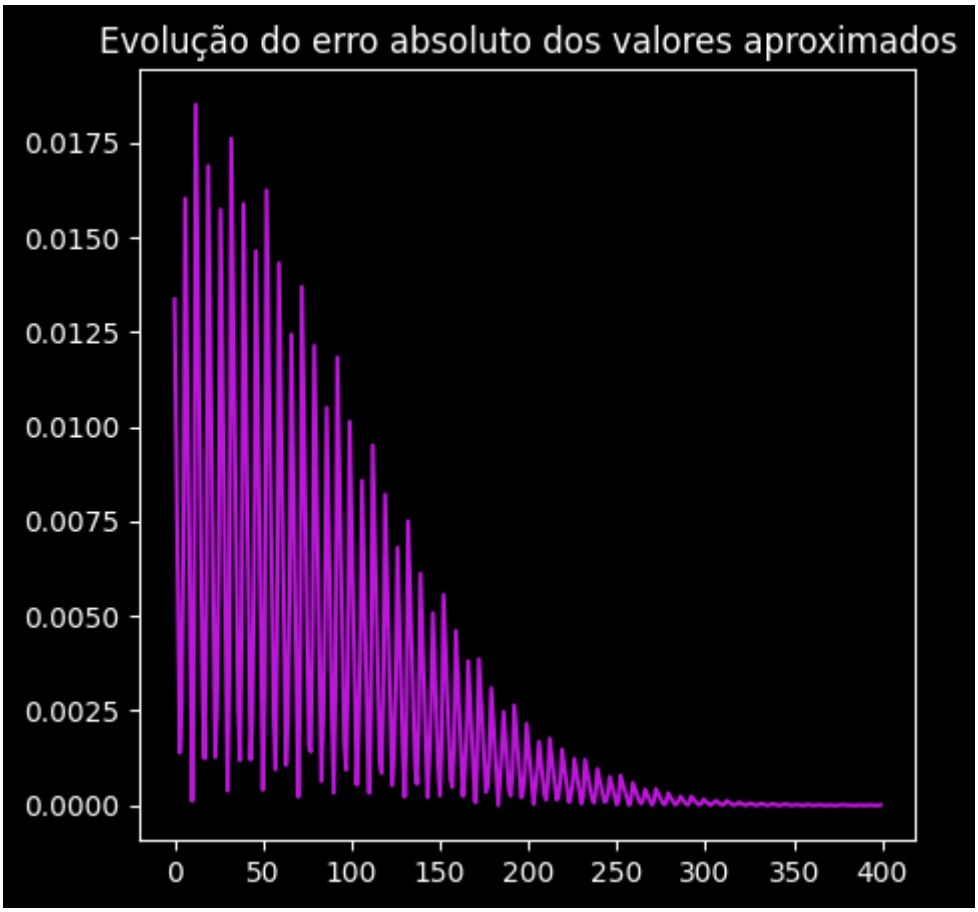
In [6]:
```
print("Estimado:", tabela1.loc[0/10, 0/100],
      "\nReal:", norm.cdf(0))
```

```
Estimado: 0.51336
Real: 0.5
```

Para $p = \frac{1}{2}$

In [7]:
```
erros(tabela1)
```

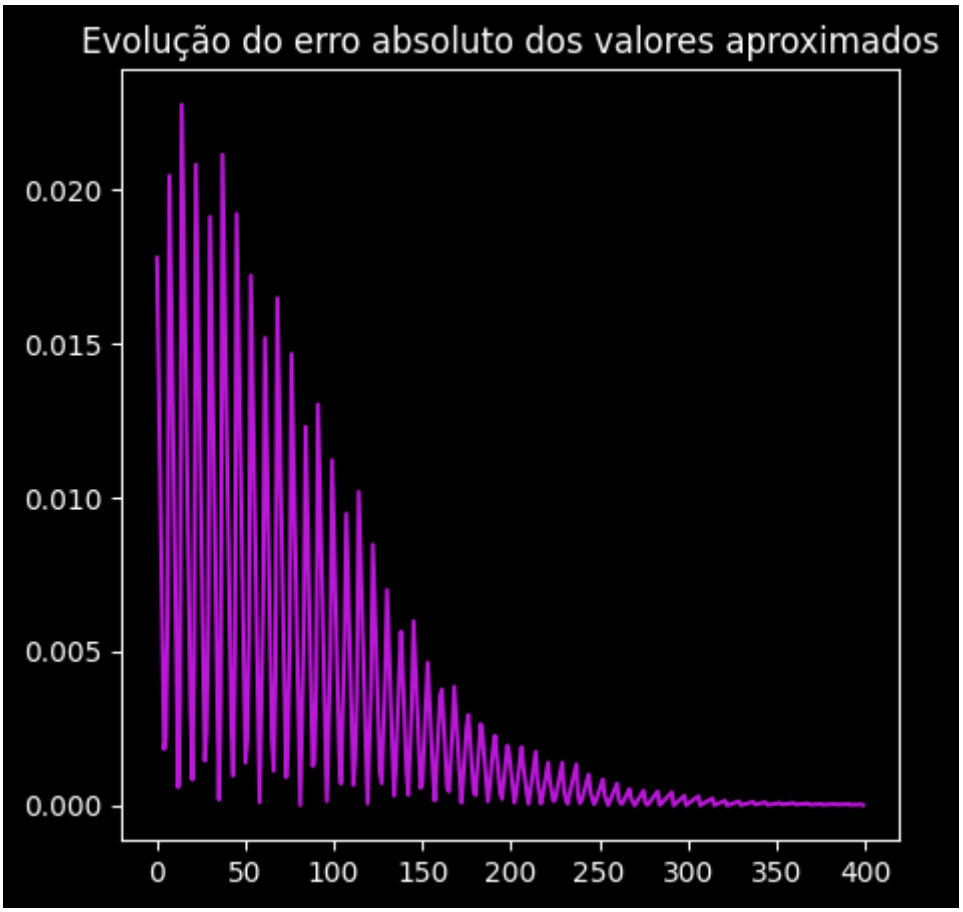
```
A média dos erros absolutos foi: 0.0024621511163030125
```



Para $p = \frac{1}{4}$

In [8]:
```
%%time
b=_(p=1/4)
tabela2 = tabe(b)
erros(tabela2)
```

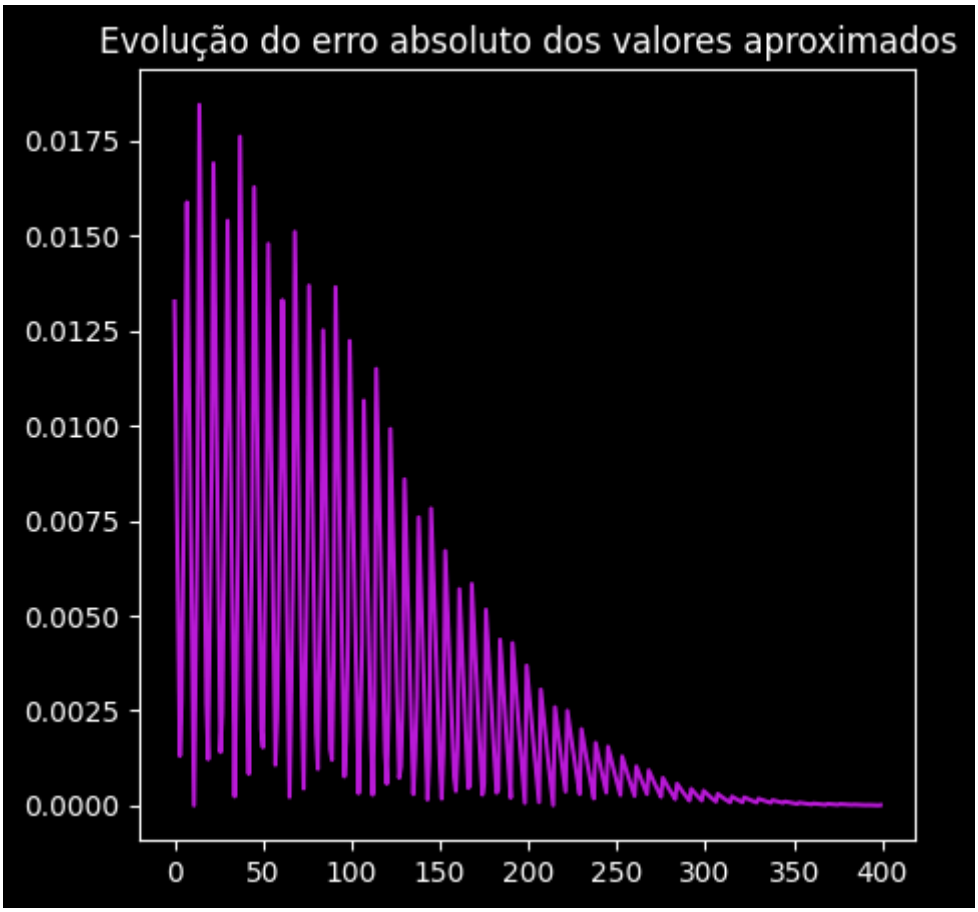
```
A média dos erros absolutos foi: 0.0028392946650582997
CPU times: total: 25.8 s
Wall time: 46.1 s
```



Para $p = \frac{3}{4}$

In [9]:
```
%%time
c=_(p=3/4)
tabela3 = tabe(c)
erros(tabela3)
```

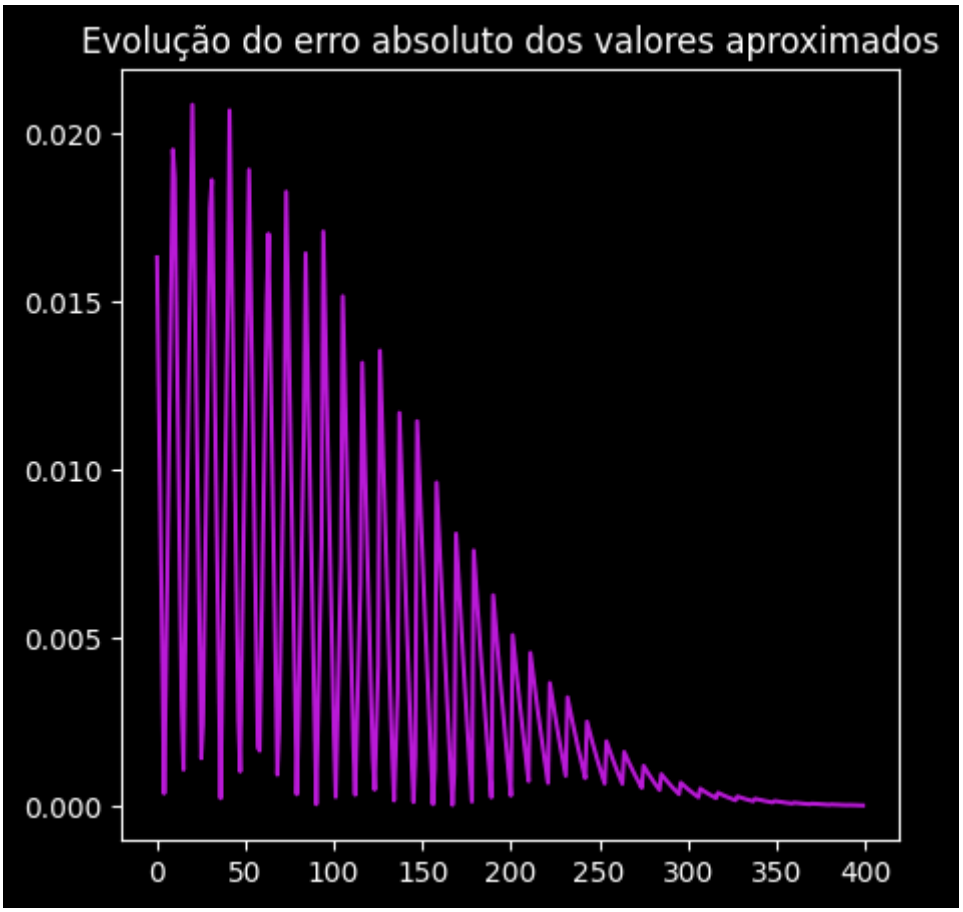
```
A média dos erros absolutos foi: 0.002902950542576037
CPU times: total: 26.3 s
Wall time: 44.6 s
```



Para $p = \frac{8}{9}$

In [10]:
```
%%time
c=_(p=8/9)
tabela3 = tabe(c)
erros(tabela3)
```

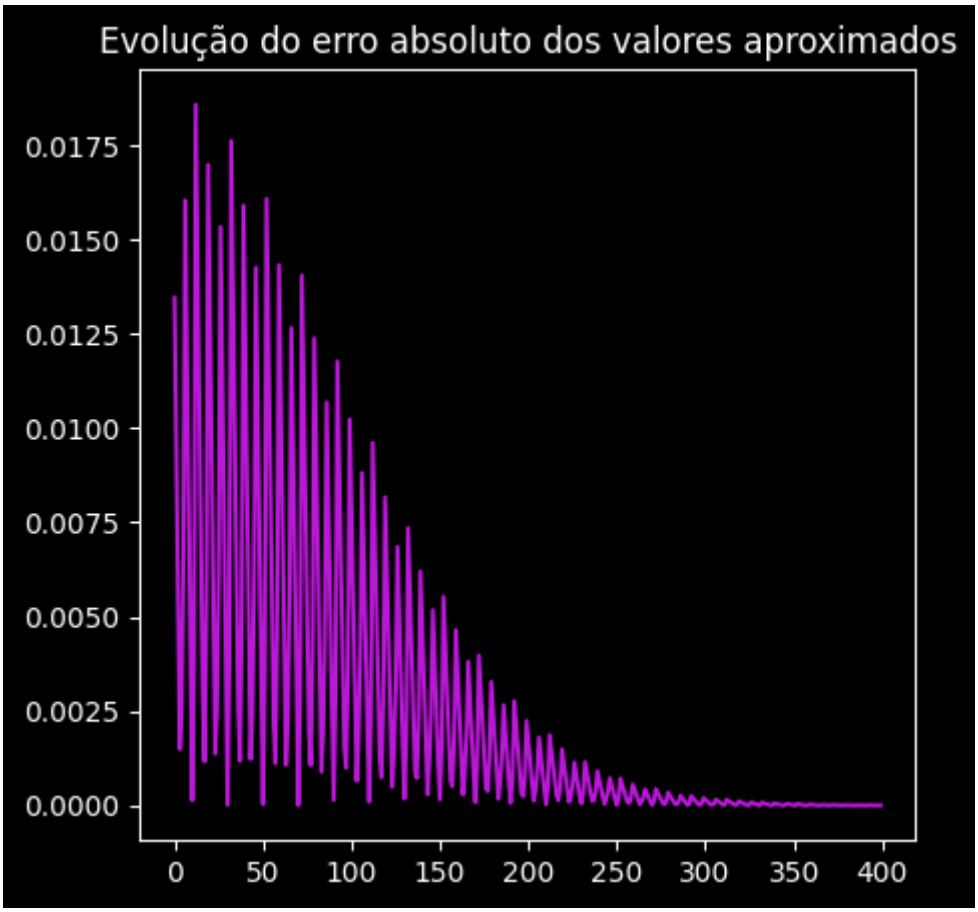
```
A média dos erros absolutos foi: 0.0039992784805598924
CPU times: total: 26.7 s
Wall time: 47.6 s
```



Para $p = \frac{1}{2} \ \wedge n = 10^7$

In [11]:
```
%%time
a=_(n=10**7)
erros(tabe(a))
```

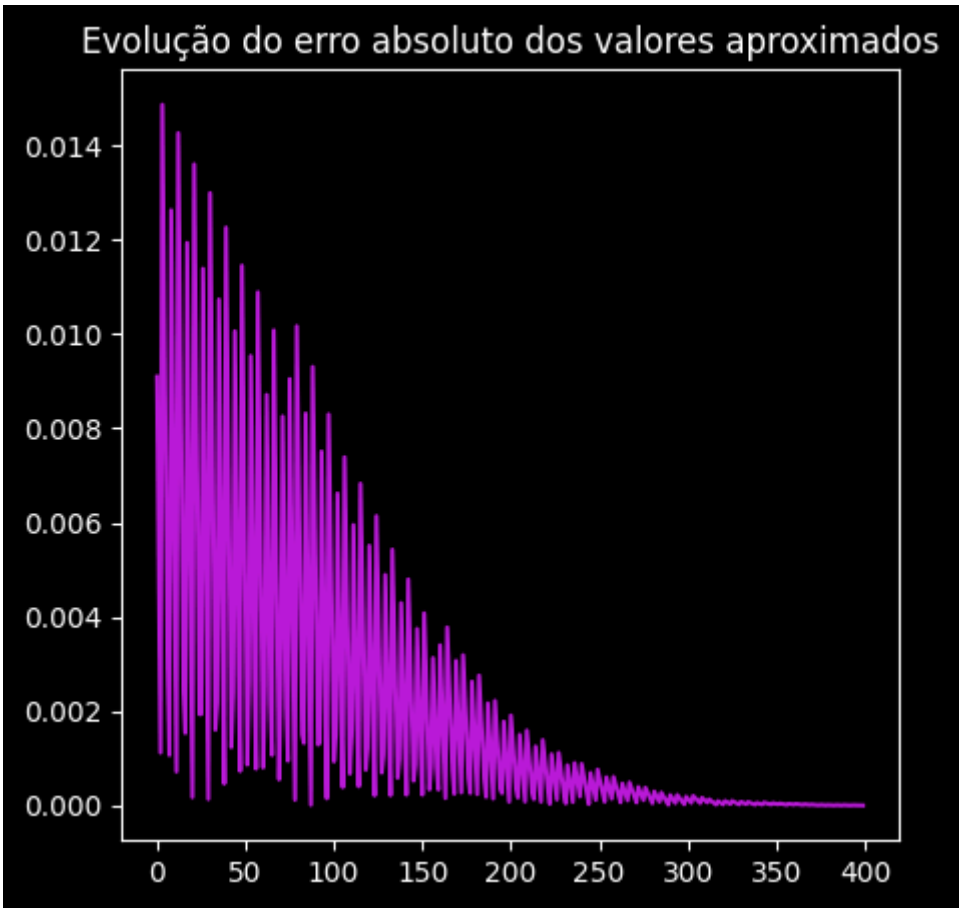
```
A média dos erros absolutos foi: 0.002469368879051753
CPU times: total: 4min 43s
Wall time: 8min 2s
```



Para $p = \frac{1}{2} \ \wedge m = 2000$

In [12]:
```
%%time
b=_(m=2000)
tabela21 = tabe(b)
erros(tabela21)
```

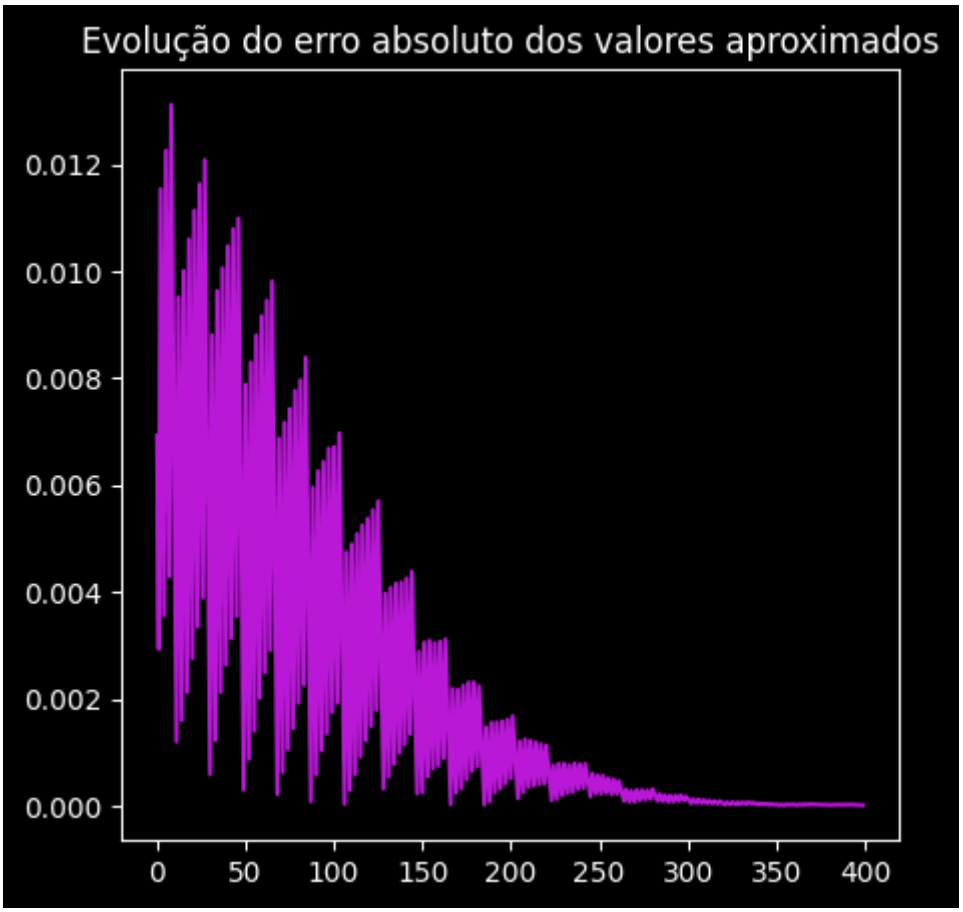
```
A média dos erros absolutos foi: 0.0019930525872372735
CPU times: total: 14.2 s
Wall time: 50.6 s
```



Para $p = \frac{1}{2} \ \wedge m = 4000$

In [13]:
```
%%time
b=_(m=4000)
tabela21 = tabe(b)
erros(tabela21)
```

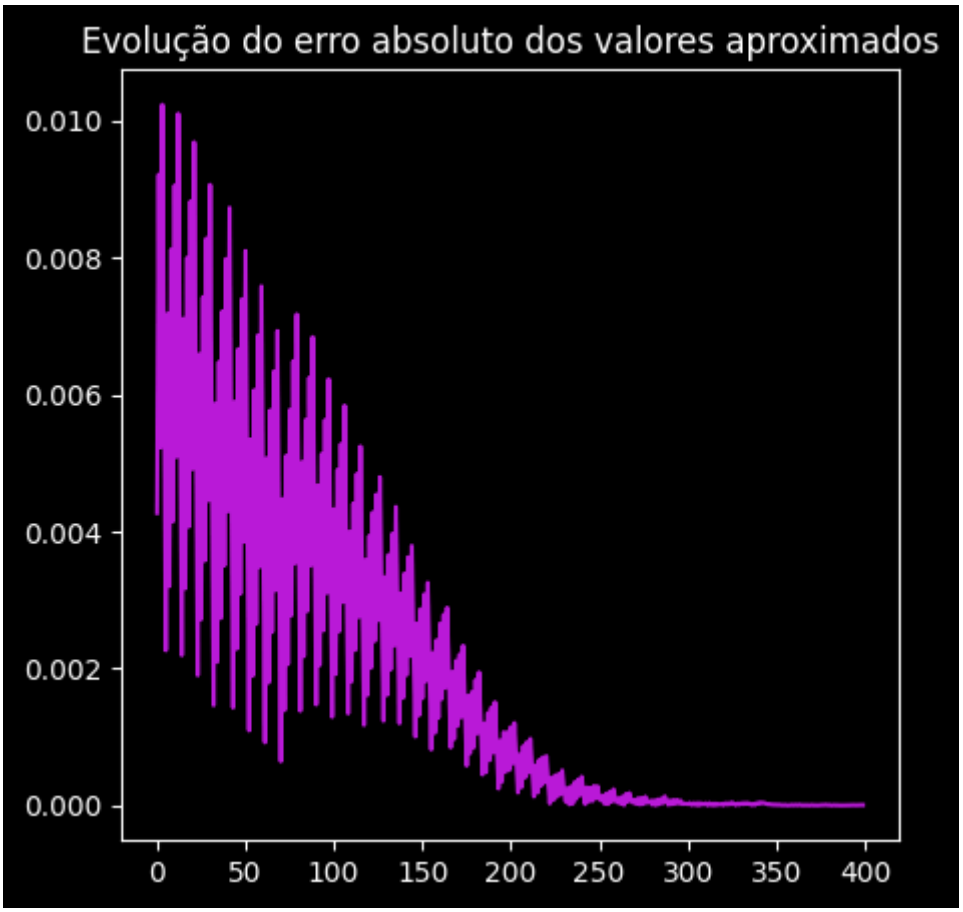
```
A média dos erros absolutos foi: 0.0020406275490278385
CPU times: total: 16.7 s
Wall time: 44.2 s
```



Para $p = \frac{1}{2} \ \wedge m = 8000$

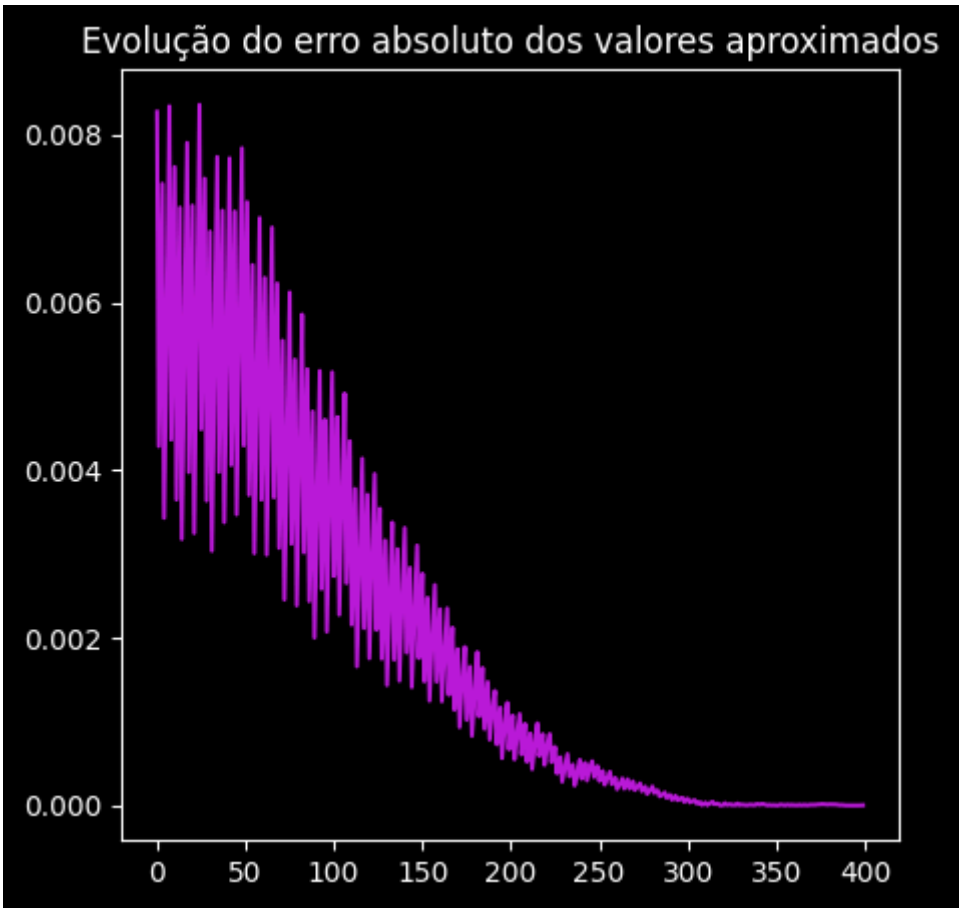
In [14]:
```
%%time
b=_(m=8000)
tabela21 = tabe(b)
erros(tabela21)
```

```
A média dos erros absolutos foi: 0.0018323905288714231
CPU times: total: 12.9 s
Wall time: 43.5 s
```



In [15]:
```
%%time
b=_(m=20000)
tabela21 = tabe(b)
erros(tabela21)
```

```
A média dos erros absolutos foi: 0.0018911861551679854
CPU times: total: 13.2 s
Wall time: 43.3 s
```



Conclusão

Vimos que o valor que apresentou menor erro absoluto médio foi $p = \frac{1}{2}$. Além disso, não houve melhora significativa ao aumentar o número de repetições. Por outro lado, aumentado o número de valores binomiais gerados em cada ensaio, foi possível diminuir a média dos erros.

03/04/2024

Lei Forte de Kolmogorov

Sejam $X_1, X_2, ..., X_n$ v.a.i.i.d. com média finita. Defina

$S_n = X_1 + X_2 + ... + X_n$.

$\bar{X_n} = \frac{S_n}{n} = \frac{X_1+...+X_n}{n}$

Então, $\frac{S_n}{n} \xrightarrow{qc} \mathbb{E}(X_i) = \mu$

$\forall\, \epsilon > 0$ com probabilidade 1, existe $N_0(\epsilon)$ aleatório tal que para $n > N_0(\epsilon)$

$\frac{S_n}{n} \in (\mu - \epsilon, \mu + \epsilon)$

```
import numpy as np
import scipy.stats as stata
import matplotlib.pyplot as plt
import pandas as pd
```

```
# Primeiro caso: binomial
p, m, n = .5, 10, 10**5
soma = 0    # soma acumulada
vetormedias = [0 for i in range(n)]

for i in range(n):
    valor = stata.binom.rvs(size=1, n = m, p = p)[0]
    soma = valor + soma

    if i==0: vetormedias[i]=soma
    else: vetormedias[i] = soma/i

plt.style.use('dark_background')
fig, ax = plt.subplots()
ax.plot(vetormedias, color='yellow')
ax.set_title("Lei Forte para Binomial(1, 1/2)")
```

```
Text(0.5, 1.0, 'Lei Forte para Binomial(1, 1/2)')
```

```
# Segundo caso: Normal(0, 1)
m, n = 10, 10**5
soma = 0    # soma acumulada
vetormedias = [0 for i in range(n)]

for i in range(n):
    valor = stata.norm.rvs()
    soma = valor + soma

    if i==0: vetormedias[i]=soma
    else: vetormedias[i] = soma/i

plt.style.use('dark_background')
fig, ax = plt.subplots()
ax.plot(vetormedias, color='yellow')
ax.set_title("Lei Forte para Normal(0, 1)")
```

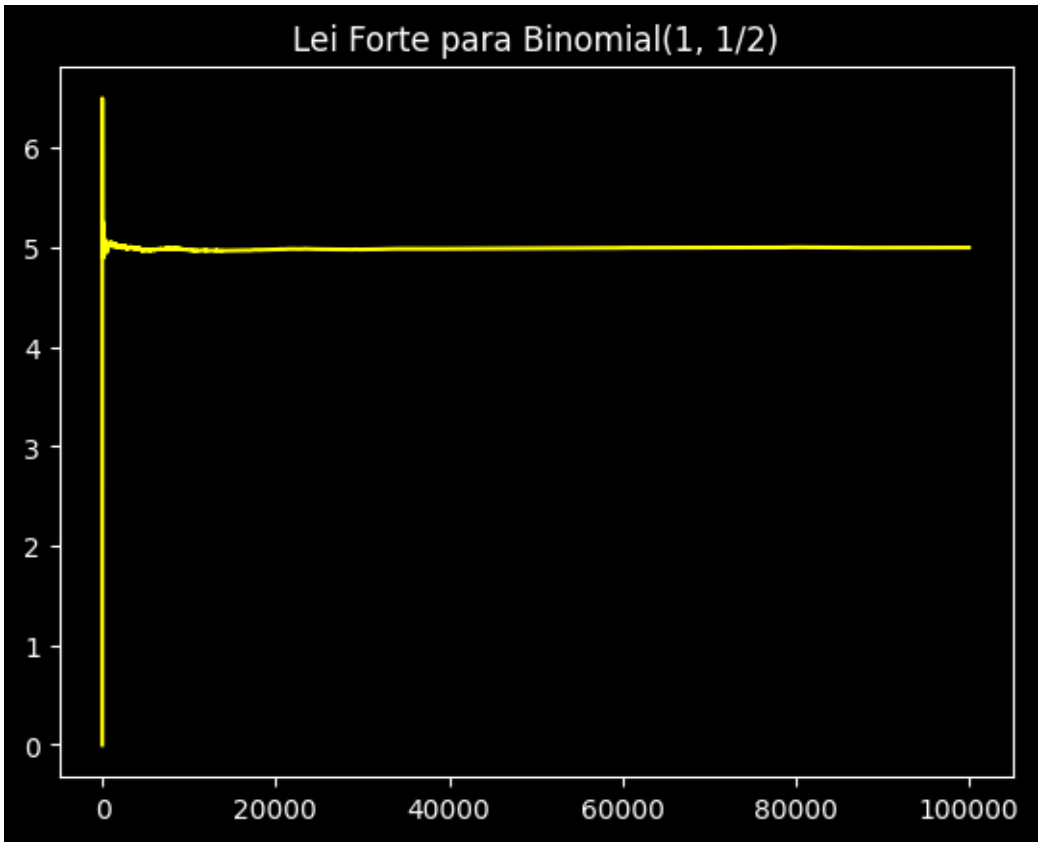

Lei Forte para Binomial(1, 1/2)
6
5
4
3
2
1
0
0
20000
40000
60000
80000
100000

Figure 1: png

```
Text(0.5, 1.0, 'Lei Forte para Normal(0, 1)')
```

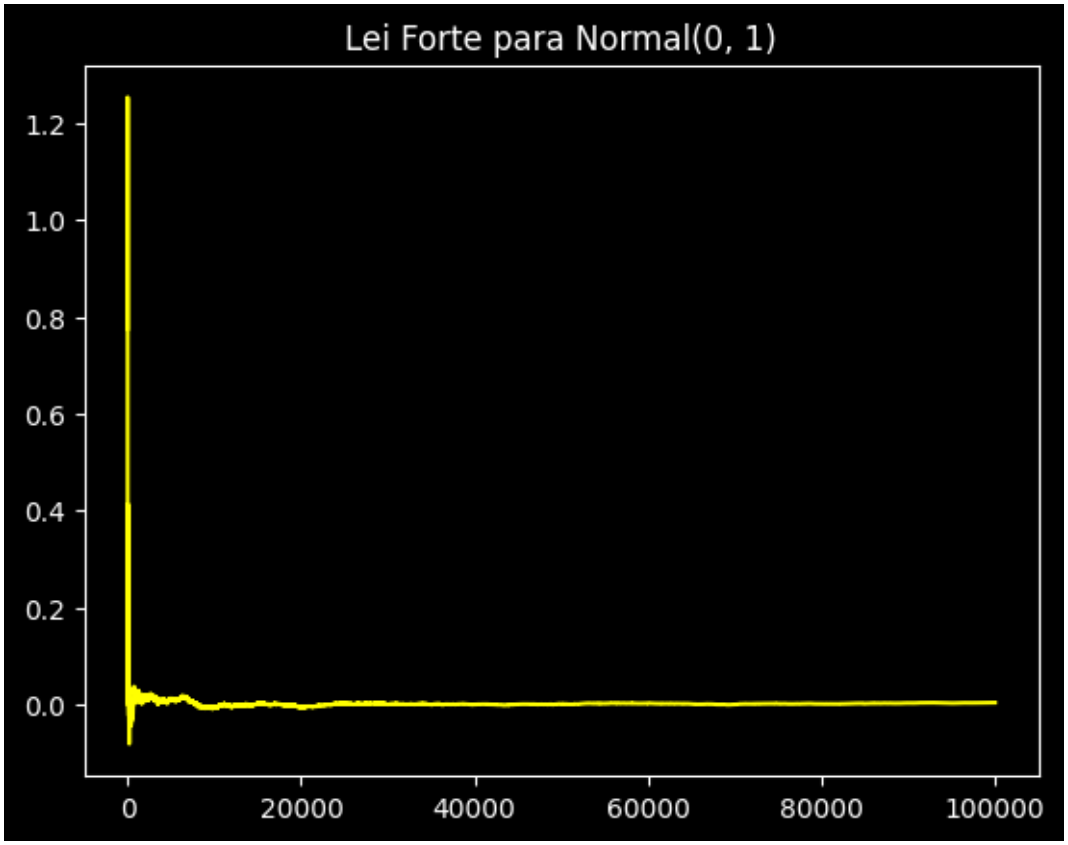


Figure 2: png

```python
# Terceiro caso: Cauchy(0, 1)

m, n = 10, 10**6
soma = 0    # soma acumulada
vetormedias = [0 for i in range(n)]

for i in range(n):
    valor = stata.cauchy.rvs()
    soma = valor + soma

    if i==0: vetormedias[i]=soma
    else: vetormedias[i] = soma/i

plt.style.use('dark_background')
fig, ax = plt.subplots()
ax.plot(vetormedias, color='yellow')
ax.set_title("Lei Forte para Cauchy(0, 1)")
```

```
Text(0.5, 1.0, 'Lei Forte para Cauchy(0, 1)')
```

Como era de se esperar, a distribuição Cauchy padrão, que não possui média finita, não apresentou a convergência descrita na Lei Forte.

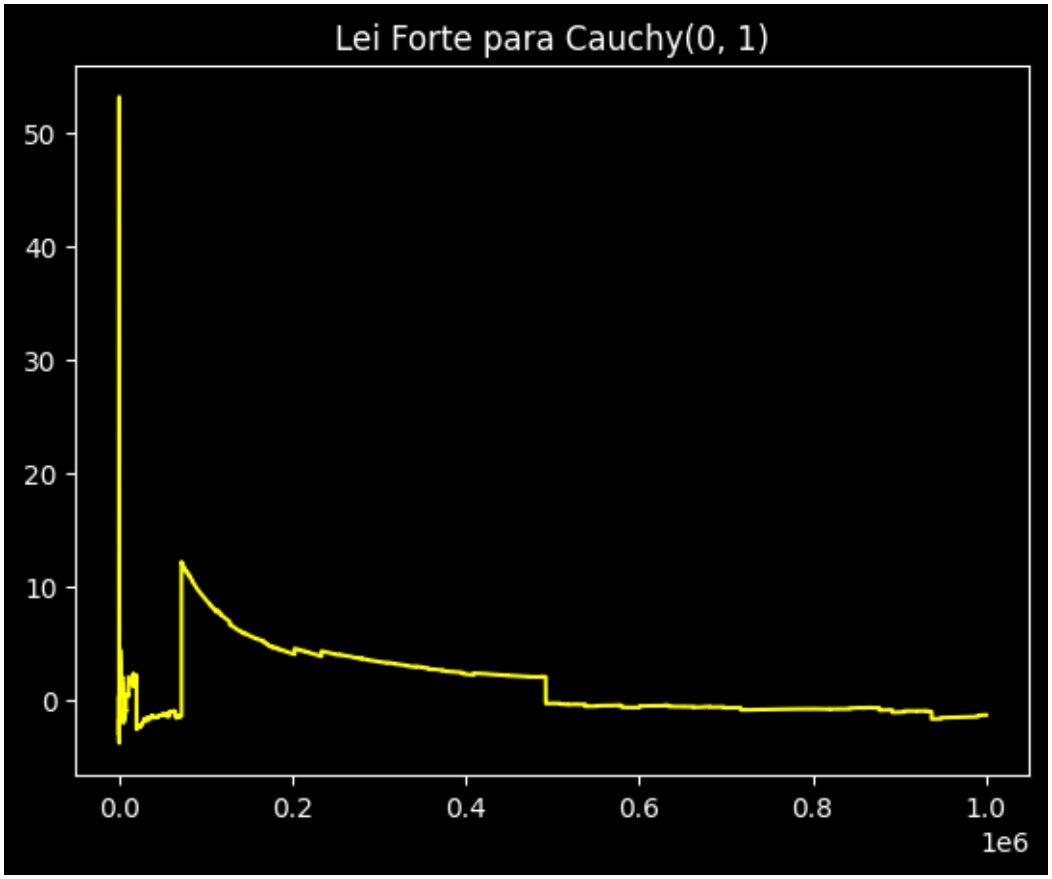


Figure 3: png

aula5

July 24, 2024

1 04/04/2024

Integral por Monte Carlo

$$Sejam\ X_1, X_2, ..., X_n\ v.a.i.i.d.com X_i \sim U(0,1) \tag{1}$$

$$\frac{X_1 + X_2 + ... + X_n}{n} \xrightarrow{qc} \frac{1}{2} = \mu_{X_i} \tag{2}$$

$$\tag{3}$$

$$Seja\ g : \mathbb{R} \to \mathbb{R}\ contnua\ de\ forma\ que\ g(X_1)\ tem\ mdia\ finita. \tag{4}$$

$$Ento,\ \mathbb{E}[g(x)] = \int_{-\infty}^{\infty} g(x) f(x)\, dx \tag{5}$$

[1]:
```
import numpy as np
import scipy.stats as stata
import matplotlib.pyplot as plt
import pandas as pd
import statistics
import math
```

$\int_0^1 x^2 dx$

[6]:
```
%%time

n=10**5

soma = 0
for i in range(n):
    uni = stata.uniform.rvs(size=1)[0]
    soma += uni**2

I = soma/n; I
```

```
CPU times: total: 2.31 s
Wall time: 5.63 s
```

[6]: 0.3335988162340694

[10]:
```
n=10**5
soma = 0
for i in range(n):
    uni = stata.uniform.rvs(loc=-np.pi/2, scale=np.pi/2)
    soma += uni**2

I = soma/n; I
```

[10]: 0.8234585116954918

[138]:
```
#92810

n=10**5
soma = 0
for i in range(n):
    uni = stata.uniform.rvs(size=1, scale=3)

    prod1 = uni**9
    prod2 = np.e**uni
    soma = prod1*prod2
    #print(soma)

I = 2*(soma/n)
print(I)
```

[0.19932086]

[140]:
```
uni = stata.uniform.rvs(loc=-np.pi/2, scale=np.pi/2)
uni
```

[140]: -0.10365402756462494

[159]:
```
result = []
for rep in range(10**3):
    soma = 0
    for i in range(1,10**3):
        u = np.random.uniform(1,3)
        soma = soma + (u**9 * np.e**u)

    I = 2*soma/n
    result.append(I)

np.mean(result)
```

[159]: 927.2469155608384

Construção da tabela da normal por MC

Não deu certo!

[ ]:
```
%%time

n = 10**3
tabela = [0 for i in range(400)]
for i in range(n):
    for j in range(400):
        u = stata.uniform.rvs(0, j/100)
        tabela[j] += (1/((np.pi*2)**(1/2)))*(np.e**((-u**2)/2))


tabela2 = [valor/n for valor in tabela]
tabela2
```

[251]:
```
# versão w
n = 10_000
soma = []
prob = []

# %%

for i in range(1, n):
    for j in range (1,400):
        u = np.random.uniform(0,j/100)
        soma.append(u + 1/math.sqrt(2*np.pi)*np.e**-(u**2)/2)


for i in range(1, 400):
    prob.append((i/i)*soma[i]/n+0.5)

print(prob)
```

```
[0.5000209466885378, 0.5000210279494267, 0.5000201080881641, 0.5000227067920486,
0.5000212112363603, 0.5000233667995095, 0.5000246902911727, 0.5000261520214259,
0.5000242107930224, 0.500027228438802, 0.5000269582195787, 0.5000283374555217,
0.5000249822375328, 0.5000234990592851, 0.5000325484092935, 0.5000345365076815,
0.5000367652822795, 0.500029537339573, 0.5000255306290645, 0.5000324665375965,
0.5000298606171819, 0.5000345089041633, 0.5000400657693062, 0.5000273932769049,
0.500032902713878, 0.5000200039658428, 0.5000346500135702, 0.500022088020149,
0.500031467754692, 0.5000229270958957, 0.5000455441216999, 0.5000407668018078,
0.5000469397471365, 0.5000517982458668, 0.5000522433971004, 0.5000520863968096,
0.5000407398460207, 0.5000314470618313, 0.5000441303087705, 0.5000346215452747,
0.5000382519205778, 0.5000537461698721, 0.5000207845652587, 0.5000398094615341,
0.5000294718054108, 0.5000478342886878, 0.5000559072462842, 0.5000413021113604,
0.5000612071809802, 0.5000479942791055, 0.500035593253822, 0.5000651259228028,
```

0.5000263294411831, 0.5000466417691581, 0.5000435642732306, 0.5000458637505928,
0.5000715009943775, 0.5000588482231297, 0.5000286853330922, 0.5000730723536977,
0.500034445053134, 0.5000716799159733, 0.5000491914843399, 0.5000670733235716,
0.5000467208792319, 0.5000700274992598, 0.5000372559719446, 0.5000248531289648,
0.5000355424486608, 0.5000323774520011, 0.5000363338299517, 0.5000513821367797,
0.5000509118519292, 0.5000721506330514, 0.5000792262972885, 0.5000387381827399,
0.5000334508482752, 0.500058875113119, 0.5000407066091548, 0.500038943834247,
0.5000700234905061, 0.5000589514896363, 0.500073208260062, 0.5000290869432128,
0.5000772615611685, 0.5000565862636256, 0.5000284791340994, 0.5000970263828665,
0.5000350508073527, 0.5000897213500465, 0.5000892864278795, 0.5000418349550186,
0.5000249235985774, 0.5000578567315275, 0.5001014935588546, 0.50007962910852,
0.5000920743206115, 0.5000660061935851, 0.5000377326280987, 0.5001043523817784,
0.5000299985990335, 0.5000809370038344, 0.5000955846924456, 0.5000420260657126,
0.5001088923582413, 0.5000575769697744, 0.500055712607719, 0.5000416490626591,
0.5000924372658121, 0.5000977719345008, 0.5001025296342095, 0.5000292196953253,
0.5000937204049576, 0.5000420713274613, 0.5001089060877912, 0.5000533385406348,
0.5000690409316053, 0.5001048987314695, 0.5000415811014871, 0.5001007661803194,
0.5001114672149924, 0.5000305738149283, 0.5001080610928642, 0.5000350739455898,
0.5000236314128318, 0.5000470693276164, 0.5000407684307432, 0.500094382956424,
0.500072400782314, 0.5000694821354047, 0.5000395512040932, 0.500034906905923,
0.5000809920253217, 0.5000242253267688, 0.5000633115453254, 0.5001110310345288,
0.500075658416193, 0.5000327166353806, 0.500026095939661, 0.5000718944570295,
0.5001343393094821, 0.5000755776295234, 0.5001027295478633, 0.5001260850772353,
0.5001374137543195, 0.5001067258200526, 0.5001477509839637, 0.5001499347513538,
0.5000778034895443, 0.5000851302030629, 0.5000916926558356, 0.5000722318907108,
0.5000816399642334, 0.5001275857372444, 0.5001154545619508, 0.5001206951897789,
0.5001522004740606, 0.5001268767953614, 0.5000661632144446, 0.5001277942649872,
0.5001096658628693, 0.5000658979870133, 0.500078643067219, 0.5001417226381104,
0.5001150767990292, 0.5000370704119568, 0.500137632658748, 0.5000363042692226,
0.5000690023211521, 0.5001549750085973, 0.5001280860908471, 0.5000670368002545,
0.500092394667814, 0.5001049663841589, 0.5001353612163125, 0.5001742418566675,
0.5000861652367107, 0.5000859943013453, 0.5001216551717492, 0.5000365734367361,
0.5001458415797071, 0.5001319138019822, 0.5000393898095764, 0.5000414471612803,
0.5000701340385281, 0.5001309472950902, 0.5001214719783137, 0.5000941770730943,
0.5001032513637932, 0.5001846311529874, 0.5000522910392557, 0.500156288876737,
0.5001163978156328, 0.5001726013544817, 0.5000923912697225, 0.5001378267604119,
0.5000984313629748, 0.5000481052242413, 0.5000969784987401, 0.5001558795490348,
0.5001239058474656, 0.5001967983398715, 0.5001872552216969, 0.5001800951684565,
0.5001347802536799, 0.5000780883766153, 0.500140534650785, 0.5000904648316775,
0.5001661254722208, 0.5000807469304585, 0.5000715566528845, 0.500034715011407,
0.5001302987884184, 0.5001229014132168, 0.5002015565477566, 0.5002018795147533,
0.5001122368460449, 0.5000454018742403, 0.5002095124235041, 0.500187700408739,
0.5000299774799647, 0.5000516615756155, 0.5002201778392715, 0.500128510193162,
0.5001228894530425, 0.5001134165941795, 0.5002145713415524, 0.50010892424316,
0.5000843140268296, 0.5001255910122003, 0.500172319629536, 0.500223714463439,
0.5001553425069571, 0.5000299975124183, 0.5001067258108223, 0.5001151334307123,
0.5002129957921987, 0.5000783603884303, 0.5002182254960689, 0.5001689192919322,
0.500186404801346, 0.5002143116556981, 0.5001350374589402, 0.5001185506575881,

```
0.5000303373076722, 0.5000536086833343, 0.5000339455597946, 0.5001723453802044,
0.500207837605054, 0.5001346235739652, 0.5002240428487648, 0.5002422434241947,
0.5001088465125881, 0.5001404438923516, 0.500225169639265, 0.500161757777503,
0.5000271359853793, 0.5000760980536887, 0.5001514237562104, 0.500101411124598,
0.5001379158569347, 0.5001213359426151, 0.500159257259606, 0.5001101086097545,
0.5001969989696244, 0.5001446526633717, 0.5002452064787616, 0.5000434717948707,
0.5002313295698251, 0.5001655320721046, 0.5001114693544387, 0.5000749226930332,
0.5000206376290943, 0.5001651960094919, 0.5000747963776734, 0.5001786819762766,
0.5000296272601864, 0.5001505682820503, 0.5002507363967995, 0.5002136419659848,
0.5000997919856247, 0.5002474608695705, 0.5000562593599946, 0.5002466420347272,
0.5001556684160836, 0.5001108318842111, 0.5001272676461131, 0.5001572250041562,
0.5000207827575478, 0.5001660955583971, 0.5000510641047329, 0.5002881873631255,
0.5000760663803536, 0.5001801933537025, 0.5002374760339986, 0.5000739021922174,
0.5002789436177555, 0.5001070479457984, 0.5000860297260612, 0.5000708303379862,
0.5000472491196161, 0.5002457193973578, 0.5001943448688979, 0.5001722991679527,
0.5001374927503587, 0.500020609485929, 0.5002792326414597, 0.5000735048592221,
0.5002967552828476, 0.5002984898028219, 0.5001936264456551, 0.5002913850710948,
0.5001516256466709, 0.5002021681790154, 0.5003088585837672, 0.5001014507330543,
0.5000729107869402, 0.5002161012627451, 0.5002461500479907, 0.500089306335451,
0.5000922454905214, 0.500223059588183, 0.5002625616195322, 0.5001533127861018,
0.5001931958367531, 0.5000557419235931, 0.5002157467269179, 0.5001947236414664,
0.5001266229567877, 0.5001007409922819, 0.500080806315732, 0.500073960694255,
0.500197004443015, 0.5002113097195888, 0.5001947436610749, 0.5002598139991817,
0.500177728679528, 0.5000805029473676, 0.5001194650122757, 0.5000803050600339,
0.5001436872445979, 0.5000507417337339, 0.5000649514325521, 0.5002928013567285,
0.5001115682359996, 0.5001221531202054, 0.5001540862736626, 0.5002450979575755,
0.5000202835028897, 0.5000373789500978, 0.5003252724961541, 0.5000600890304725,
0.5002044009573646, 0.5001715429950968, 0.5001740722909211, 0.5002953556091851,
0.500048255234635, 0.5001067878719002, 0.5000322919111903, 0.5002310621849075,
0.5001595172735626, 0.5000536515607631, 0.5002842447697513, 0.5001125833182785,
0.5000969239998252, 0.5001416847295523, 0.5001113617165376, 0.5002313619201648,
0.5002914658629686, 0.500025411453254, 0.5001853001007854, 0.5001263160959061,
0.5002010705032719, 0.500192919971878, 0.5003218838472803, 0.5000455329908245,
0.5003558029969933, 0.5003620818906094, 0.500223419819485, 0.5003155944526847,
0.5003319154598861, 0.5000753373532828, 0.5000249970041704, 0.5001527946039979,
0.5000437849919711, 0.5002848823046361, 0.5003569241935594, 0.5001497136548511,
0.5000368071867036, 0.5002915758953814, 0.5001165356901212, 0.5002967031436296,
0.5003858239269318, 0.5002443465139055, 0.5001580605051926, 0.5002923149417788,
0.5001069591343862, 0.5002441304320657, 0.5000208229171514]
```

[ ]:
```python
# Yield successive n-sized
# chunks from l.
def dividir_lista(l, n):
    """looping till length l """
    for i in range(0, len(l), n):
        yield l[i:i + n]
```

```
# A lista deve ser dividida em n listas
n = 10
x = list(dividir_lista(tabela2, n))

tab = pd.DataFrame(x,
        columns = [x/100 for x in range(10)],
        index = [x/10 for x in range(40)])
tab
```

09/04/2024

Passeio aleatório

$$S_n = S_0 + \Sigma_{i=1}^{n} X_i$$

Onde,

$S_n :=$ ¿'capital do jogador em n rodadas'

$S_0 :=$ ¿'capital inicial'

$\Sigma_{i=1}^{n} X_i :=$ ¿'lucro em n rodadas'

X_i's são v.a.i.i.d.

$P(X_i = 1) = p \wedge P(X_i = -1) = 1 - p$

Queremos $P_{CM} :=$ ¿'probabilidade do jogador entrar em ruína antes de atingir M'

Para sair no lucro o jogador deve ganhar $M - C$ (saldo)

Para sair no prejuízo o jogador deve perder C (saldo)

```
import numpy as np
import scipy.stats as stata
import pandas as pd
import matplotlib.pyplot as plt

def algoritmo1(p=1/2, c=10, m=20, t=0):
    """
    Ensaio inicial.
    """
    s = c
    while s!=0 and s!=m:
        uni = stata.uniform.rvs(size=1)
        if 0<uni<p: s = s + 1
        else: s = s - 1
        t = t + 1

    return t


def algoritmo2(p=1/2, r=0, c=10, m=20, dm=0, n=10**4):
    """
    .
    """
    s = c
```

```
    for i in range(n):
        t = 0
        s = 10
        while s!=0 and s!=m:
            uni = stata.uniform.rvs(size=1)
            if 0<uni<p: s = s + 1
            else: s = s - 1
            t = t + 1

        if s==0: r += 1
        dm += t

    return dm/n, r/n


def alg3(p=1/2, r=0, c=10, m=20, dm=0, n=10**4):
    """
    Para o gráfico.
    """
    s = c
    duracaom = [0 for i in range(n)]
    medias = [0 for i in range(n)]
    for i in range(n):
        t = 0
        s = 10
        while s!=0 and s!=m:
            uni = stata.uniform.rvs(size=1)
            if 0<uni<p: s = s + 1
            else: s = s - 1
            t = t + 1

        if s==0: r += 1; medias.append(r)
        dm += t
        duracaom.append(dm)

    fig, ax = plt.subplots()

    return dm/n, r/n


def algoritmo4(p=1/2, m=20, dm=0, n=10**4):
    """

    """
    for i in range(n):
        t = 0
        s = 10
        while s!=1:
            uni = stata.uniform.rvs(size=1)
            if 0<uni<p: s = s + 1
```

```
            else: s = s - 1
            t += 1

        #if s==0: r += 1
        dm += t
        print(dm/n)


algoritmo2()
```

```
(99.499, 0.4976)
```

```
1/2**30
```

```
9.313225746154785e-10
```

```
algoritmo2(p=3/5, m=30)
```

```
(96.9818, 0.0175)
```

```
#algoritmo2(n=10**6) Está demorando demais!

resp = [algoritmo1() for i in range(10**4)]

resp = pd.DataFrame(resp)

resp
```

```
        0
0      14
1      48
2      38
3      48
4      50
...   ...
9995   32
9996  202
9997  232
9998  526
9999  124

[10000 rows x 1 columns]
```

```
algoritmo4()
```

```
0.0123
0.022
0.0389
0.0548
0.0587
0.0642
0.1201
```

```
2.462
2.4667
2.4752
2.5047
2.528
2.5407
4.531
4.5461
4.574
5.9673
5.9848
6.0581
6.0626
6.1197
6.2824
6.2915
6.3074
6.3125
6.3166
6.3327

---------------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
~\AppData\Local\Temp\ipykernel_14688\828019275.py in ?()
----> 1 algoritmo4()

~\AppData\Local\Temp\ipykernel_14688\3962037353.py in ?(p, m, dm, n)
      5     for i in range(n):
      6         t = 0
      7         s = 10
      8         while s!=1:
----> 9             uni = stata.uniform.rvs(size=1)
     10             if 0<uni<p: s = s + 1
     11             else: s = s - 1
     12             t += 1

~\anaconda3\envs\ambiente_zero\Lib\site-packages\scipy\stats\
_distn_infrastructure.py in ?(self, *args, **kwds)
   1065             random_state = check_random_state(rndm)
   1066         else:
   1067             random_state = self._random_state
   1068
-> 1069         vals = self._rvs(*args, size=size,
random_state=random_state)
   1070
   1071         vals = vals * scale + loc
   1072
```

```
~\anaconda3\envs\ambiente_zero\Lib\site-packages\scipy\stats\
_continuous_distns.py in ?(self, size, random_state)
  10250     def _rvs(self, size=None, random_state=None):
> 10251         return random_state.uniform(0.0, 1.0, size)

KeyboardInterrupt:
```

```python
# Arquivo 2

def ruina_jogador(p=1/10, k=10, l=-2, c=20, m=40, dm=0, n=10**4):
    """
    .
    """
    s = c
    for i in range(n):
        t = 0
        s = 10
        while s!=0 and s!=m:
            uni = stata.uniform.rvs(size=1)
            if 0<uni<p: s = s + k
            else: s = s + l
            t = t + 1

        if s==0: r += 1
        dm += t

    return dm/n, r/n
```

11/04/2024

Passeio aleatório com meio aleatório

$S_n = S_0 + \Sigma_{i=1}^{n} X_i$

A lei de X_i vai ser sorteada

1º caso $X_i \vee P = p$ é tal que:

$P(X_i = 1 \vee P = p) = p \wedge i = 1, 2, \ldots$

$P(X_i = -1 \vee P = p) = 1 - p \wedge i = 1, 2, \ldots$

$P \sim U(0, 1)$

$N = min\{n, S_n = 1 \vee S_0 = 0\}$

$E(N) = ?$

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stata

def alg1(m=10**4, dt=0):
    """
    Tem o objetivo de não convergir.
    """
    for i in range(m):
        n = 0
        s = 0
        while s < 1:
            p = stata.uniform.rvs(0, 1)
            u = stata.uniform.rvs(0, 1)
            if u <= p: s += 1
            else: s -= 1
            n += 1

        dt += n
        print(dt)

    return dt/m

stata.uniform.rvs(0, 1)
```

```
0.10361942205227681
```

```
alg1()
```

```
1
6
117
118
369
372
373
374
375
378
387
392
401
402
403
404
407
672
673
674
677
910
913
920
959
960
961
982
987
988
989
990
1003
1006
1007
1008
1011
1014
1015
1016
1019
1050
1083
1086
1107
1108
1109
1148
1149
1150
```

```
1151
1152
1153
1168
1429
1432
1433
1434
1437
1444
1459
1460
1507
1514
1519
1520
1529
1582
1583
1586
1587
1588
1589
1628
1649
1650
62081
62082
62085
62086
62107
62142
62143
62144
62145
62146
62147
62150
62151
62152
62153
62154
62155
62156
62157
62180
62181
62182
62189
62190
```

```
62191
62222
62223
62226
62229
62230
62231
62238
62391
62392
62403
62422
62423
62426
62451
62454
62455
62456
62457
62532
62551
62596
62647
62648
62651
62654
62655
62656
62991
62992
62993
62998
62999
63022
63025
63026
63027
63030
63031
63068
63097
63160
63163
63188
63189

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
```

```
~\AppData\Local\Temp\ipykernel_106088\347275684.py in ?()
----> 1 alg1()

~\AppData\Local\Temp\ipykernel_106088\3392490419.py in ?(m, dt)
      5     for i in range(m):
      6         n = 0
      7         s = 0
      8         while s < 1:
----> 9             p = stata.uniform.rvs(0, 1)
     10             u = stata.uniform.rvs(0, 1)
     11             if u <= p: s += 1
     12             else: s -= 1

~\anaconda3\envs\ambiente_zero\Lib\site-packages\scipy\stats\
_distn_infrastructure.py in ?(self, *args, **kwds)
   1065             random_state = check_random_state(rndm)
   1066         else:
   1067             random_state = self._random_state
   1068
-> 1069         vals = self._rvs(*args, size=size,
random_state=random_state)
   1070
   1071         vals = vals * scale + loc
   1072

~\anaconda3\envs\ambiente_zero\Lib\site-packages\scipy\stats\
_continuous_distns.py in ?(self, size, random_state)
  10250     def _rvs(self, size=None, random_state=None):
> 10251         return random_state.uniform(0.0, 1.0, size)

KeyboardInterrupt:
```

Passeios aleatórios interagentes

```python
def p4i(n=100, m=10**4, p=.1):
    """
    Função que simula o seguinte processo: iniciando com uma partícula
    caso ela avance para a direita, tal partícula se dividirá em duas,
caso
    avance para a esquerda, tal partícula morre.
    p:='chance de ir para a direita'
    """
    sucessos = 0
    for i in range(m):
        pat = 1
        pos = 0
        cont = 0
        while pos <=100 and pat > 0:
            novasparticulas = 0
```

```
            for i in range(pat):
                u = stata.uniform.rvs(0, 1)
                if u <= p: novasparticulas += 2
            pos += 1
            pat = novasparticulas

        if pat > 0: sucessos += 1 #; print(sucessos, m)

    return sucessos/m

p4i(p=.5)
```

```
0.0196
```

```
p4i(p=.49)
```

```
0.0057
```

16/04/2024

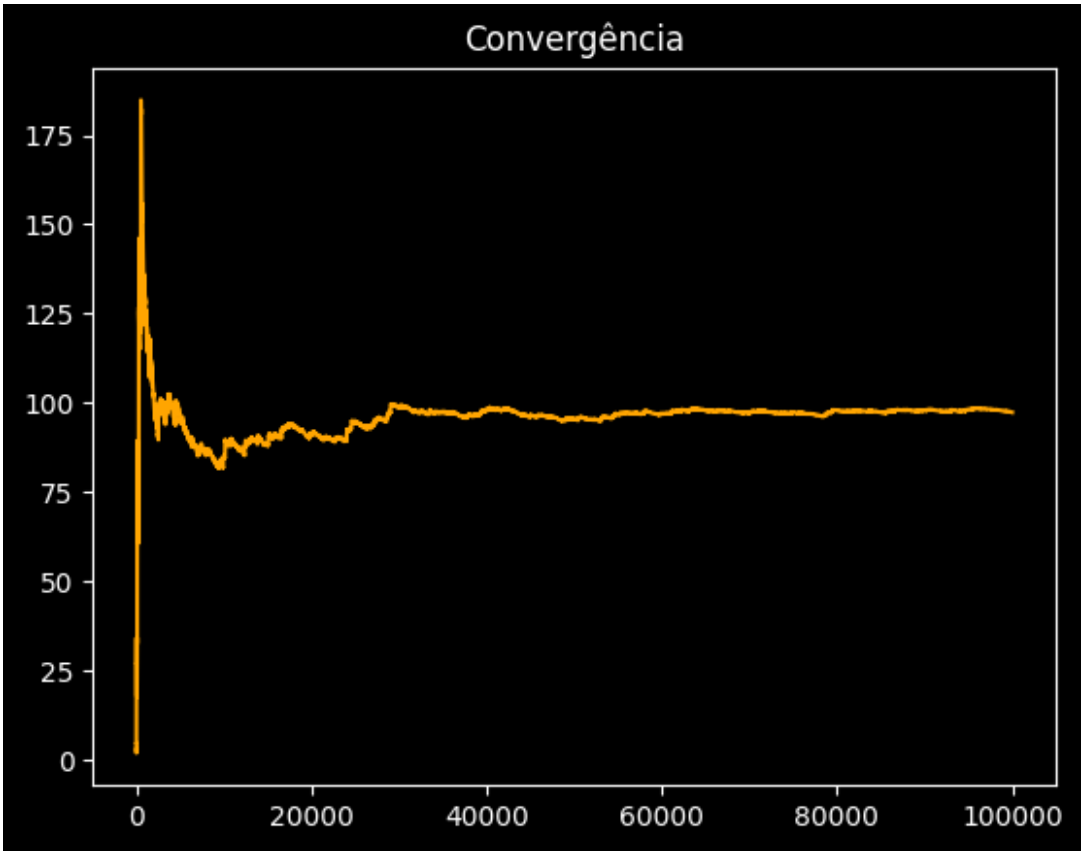
Processo de ramificação

```
import scipy.stats as stata
import numpy as np
import matplotlib.pyplot as plt

def alg1(L=.99, tempo=0, nind=1, tamger=1):
    """

    """
    while tanger > 0:
        cont = 0
        for i in range(tamger):
            y = stata.poisson.rvs(L)
            cont += y
        nind += cont
        tanger = cont
        tempo += 1

    return nind


def alg2(n=10**5, L=.99, tempo=0, nind=1, tamger=1):
    """

    """
    tt, nit = 0, 0
    for i in range(n):
        while tamger > 0:
            cont = 0
            for i in range(tamger):
                y = stata.poisson.rvs(L)
                cont += y
            nind += cont
            tamger = cont
            tempo += 1
        tt += tempo
        nit += nind
    tt = tt/n
    nit = nit/n

    return tt, nit

alg2(n=10**7, L=.999)
```

```
(2.0, 2.0)
```

```
def blah(n=10**5, L=.99):
    """

    """
    vetormedias = []
    tt, nit = 0, 0
    for j in range(1, n + 1):
        tempo, tamger, nind = 0, 1, 1
        while tamger > 0:
            cont = 0
            for i in range(tamger):
                y = stata.poisson.rvs(L)
                cont += y
            nind += cont
            tamger = cont
            tempo += 1
        tt += tempo
        nit += nind
        vetormedias.append(nit/j)
    #tt = tt/n
    #nit = nit/n

    plt.style.use('dark_background')
    fig, ax = plt.subplots()
    ax.plot(vetormedias, color='orange')
    ax.set_title("Convergência")
    #return(vetormedias)

#blah(n=1000


blah(n=10**4, L=.999)

%%time
blah()
```

```
CPU times: total: 4min 37s
Wall time: 5min 24s
```


Convergência
175
150
125
100
75
50
25
0
0
20000
40000
60000
80000
100000

04/06/2024

Teoria de filas

```
import scipy.stats as stata
import numpy as np
#import matplotlib.pyplot as plt
```

Teste da função que gera v.a.

```
x = stata.expon.rvs(scale=1, size=100000)

np.mean(x)
```

$T \sim exp(\lambda = 1)$

$E[T) = 1 = 60 min$

$A \sim exp(2)$

$E[A) = \frac{1}{2} = 30 min$

```
taxachegada <- 1
taxaatend <- 2
ociosidade <- exp(1)
usuarios <- 1
tempoanalise <- 1000
T <- ociosidade

Enquanto T <= tempoanalise faça
    Se usuarios > 0 faça
        contador <- exp(taxachegada + taxaatend)
        aux <- uniforme[0,1]
        T <- T + contador
        Se aux <= (taxachegada)/(taxachegada + taxaatend)
            Então usuarios <- usuarios + 1
        Senão usuarios <- usuarios - 1
    Senão faça
        contador <- exp(taxachegada)
        T <- T + contador
        ociosidade <- ociosidade + contador
    Fim senão
Fim enquanto

o <- ociosidade/tempoanalise
```

Por algum motivo, em python o código não funcionou como o esperado. Em R deu certo.

```
tc = 1
ta = 2
ocioso = stata.expon.rvs(scale=1, size=1)
usuarios = 1
tempoa = 1000
T = ocioso

while T <= tempoa:
    if usuarios > 0:
        contador = stata.expon.rvs(tc+ta)
        aux = stata.uniform.rvs()
        T += contador

        if aux <= (tc)/(tc+ta):
            usuarios += 1
        else:
            usuarios -= 1
    else:
        contador = stata.expon.rvs(tc)
        T += contador
        ocioso += contador

o = ocioso/tempoa

print(o)
```

```
[1.00132193]
```

```
stata.uniform.rvs()
```

```
0.21897674288761237
```

11/06/2024

Percolação

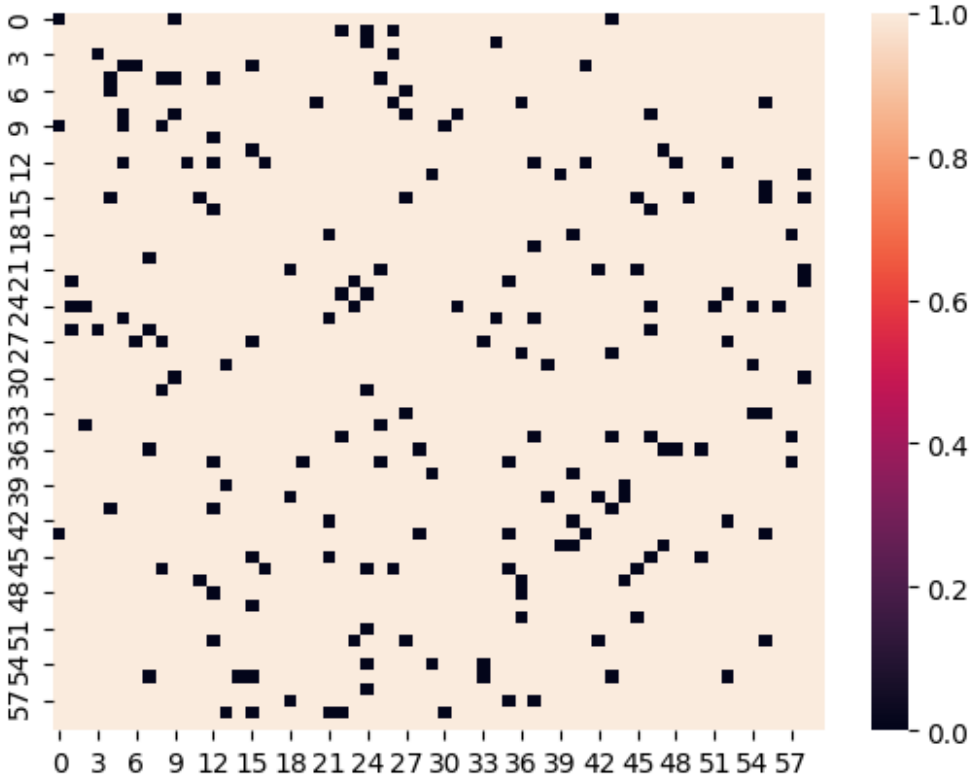
```
import scipy.stats as stata
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import networkx as nx

import itertools

def percolar(dimensao, probabilidade):
    """
    Recebe 'parteum()' e desenha o grafo com 'desenho()'.
    """
    x = parteum(dimensao, probabilidade)
    plt.figure(2)
    desenho(x)
    plt.show()

def parteum(n, p):
    """
    Realiza a percolação.
    n := dimensão da matriz
    p := probabilidade de sucesso
    """
    matriz_adj = matriz(n, p)

    no_em_comum = separar(matriz_adj[1])

    grupos_juntos = ajuntar(no_em_comum)

    #desenho(matriz_adj[1])

    return matriz_adj[1]


def matriz(n, p):
    """
    Forma uma matriz [de adjacências] que indica quando há conexão (1)
entre os
    nós e quando não há (0).
    """
```

```
    dados = np.zeros([n, n])
    matriz = pd.DataFrame(dados)

    for i in range(n):
        for j in range(i, n):
            sorteio = stata.uniform.rvs()
            if sorteio <= p:
                matriz.loc[i, j], matriz.loc[j, i] = 1, 1

    mapa_calor = sns.heatmap(matriz)
    #print(mapa_calor)

    return mapa_calor, matriz


def unico(eledigraph):
    """ NÃO UTILIZADO
    Recebe um elemento DiGraph com as arestas (edges) e unifica-as
tornando
    dispensável a ordem dos números, i.e. (1, 2) = (2, 1).
    Retorna outro elemento DiGraph, dessa vez, com arestas únicas.
    """
    copia = list(eledigraph.edges)
    lista = []
    for aresta in copia:
        aresta = sorted(aresta)
        if aresta not in lista:
            lista.append(aresta)

    teste2 = nx.DiGraph()
    teste2.add_edges_from(lista)

    return teste2.edges

def separar(matrizadj):
    """
    Lê uma matriz de adjacências e forma listas com as conexões
formadas por cada
    nó.
    """
    grupos = []
    for i in range(len(matrizadj)):
        atual = []
        for j in range(len(matrizadj)):
            ver = matrizadj.loc[i, j]
            if ver == 1:
                atual.append(j)
        atual.append(i)
        grupos.append(atual)
```

```
    return grupos


def ajuntar(listagrupos):
    """
    Recebe uma lista tal qual a formada pela função 'separar()'.
    Agrupa os nós que têm ao menos uma conexão entre si.
    """
    LL = set(itertools.chain.from_iterable(listagrupos))

    for each in LL:
        components = [x for x in listagrupos if each in x]
        for i in components:
            listagrupos.remove(i)
        listagrupos +=
[list(set(itertools.chain.from_iterable(components)))]

    contador, lmaior = 0, []
    for grupo in listagrupos:
        # Maior grupo
        if len(grupo) > len(lmaior):
            lmaior = grupo

        if len(grupo) > 0:
            contador += 1

    #Maior grupo


    print("\nFormaram-se", contador, "grupos com pelo menos um nó:")
    print(list(filter(None, listagrupos)))

    print("\nO maior grupo possui tamanho:", len(lmaior))
    print(lmaior)



def desenho(matrizdf):
    """
    Desenha o grafo [Erdos Renyi] a partir da matriz de adjacências.
    """
    G = nx.DiGraph()
    rotulo = {}
    for i in range(len(matrizdf)):
        for j in range(len(matrizdf)):
            if i != j and matrizdf[i][j] == 1:
                rotulo[j]=j
                G.add_edge(i,j)

    #print(set(list(G.edges)))
```

```
    pos = nx.spring_layout(G, seed=3000)  # positions for all nodes

    #return nx.draw_circular(G)
    return nx.draw_networkx(G, pos, arrows=False, labels=rotulo,
font_size=10)

resp8 = matriz(60, .95)
resp8[1]
```

```
{"type":"dataframe"}
```

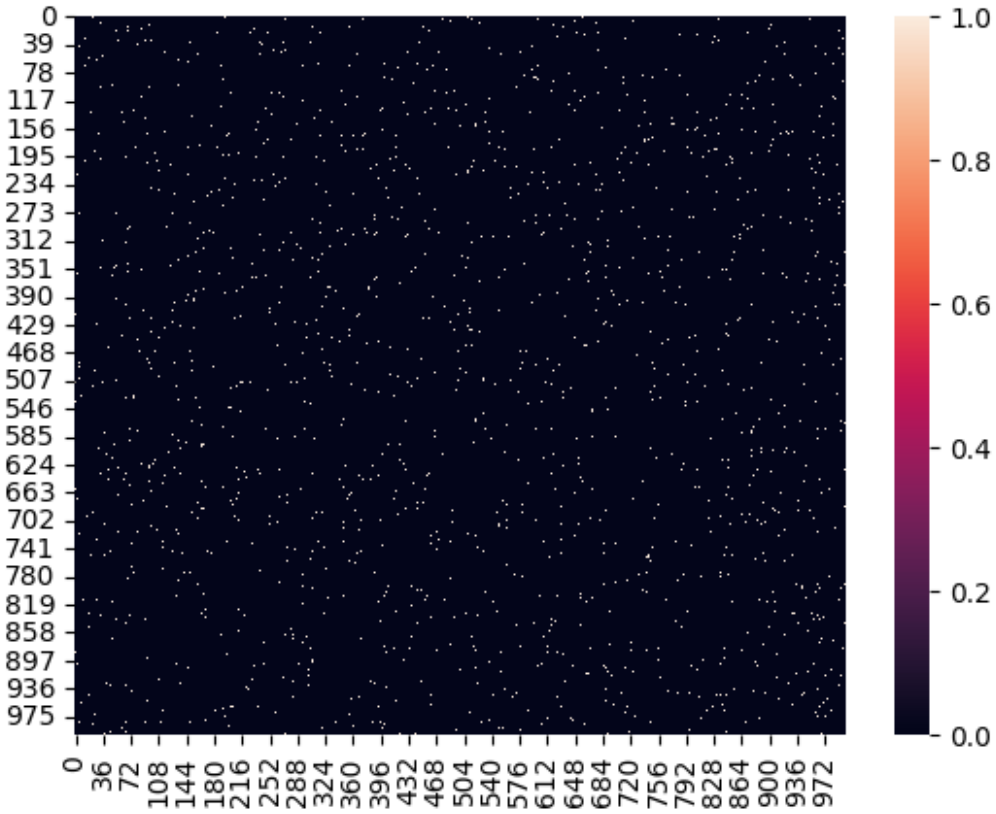


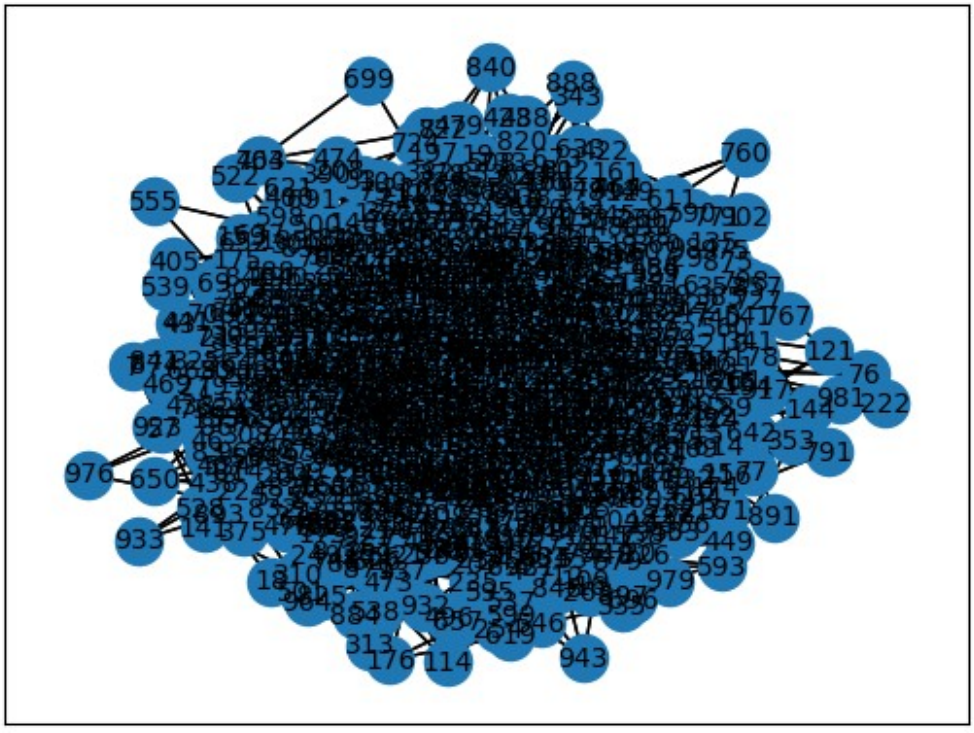

```
percolar(1000,.01)


Formaram-se 1 grupos com pelo menos um nó:
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52,
53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102,
103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158,
```

159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172,
173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200,
201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214,
215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228,
229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242,
243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256,
257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270,
271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284,
285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298,
299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312,
313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326,
327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340,
341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354,
355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368,
369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382,
383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396,
397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410,
411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424,
425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438,
439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452,
453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466,
467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480,
481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494,
495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508,
509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522,
523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536,
537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550,
551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564,
565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578,
579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592,
593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606,
607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620,
621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634,
635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648,
649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662,
663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676,
677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690,
691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704,
705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718,
719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732,
733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746,
747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760,
761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774,
775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788,
789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802,
803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816,
817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830,
831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844,

```
845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858,
859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872,
873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886,
887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900,
901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914,
915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928,
929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942,
943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956,
957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970,
971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984,
985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998,
999]]

O maior grupo foi: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65,
66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99,
100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141,
142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183,
184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197,
198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211,
212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225,
226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239,
240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253,
254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267,
268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281,
282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295,
296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309,
310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323,
324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337,
338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351,
352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365,
366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379,
380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393,
394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407,
408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421,
422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435,
436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449,
450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463,
464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477,
478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491,
492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505,
506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
```

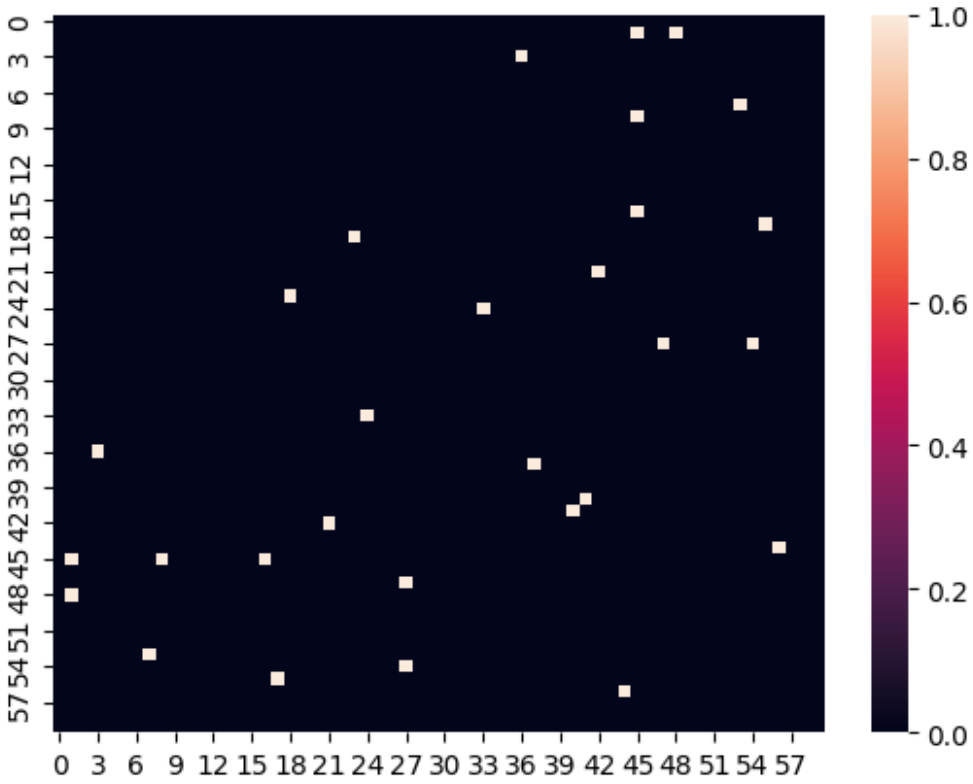
```
520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533,
534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547,
548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561,
562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575,
576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589,
590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603,
604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617,
618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631,
632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645,
646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659,
660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673,
674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687,
688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701,
702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715,
716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729,
730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743,
744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757,
758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771,
772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785,
786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799,
800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813,
814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827,
828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841,
842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855,
856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869,
870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883,
884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897,
898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911,
912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925,
926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939,
940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953,
954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967,
968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981,
982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995,
996, 997, 998, 999]
```

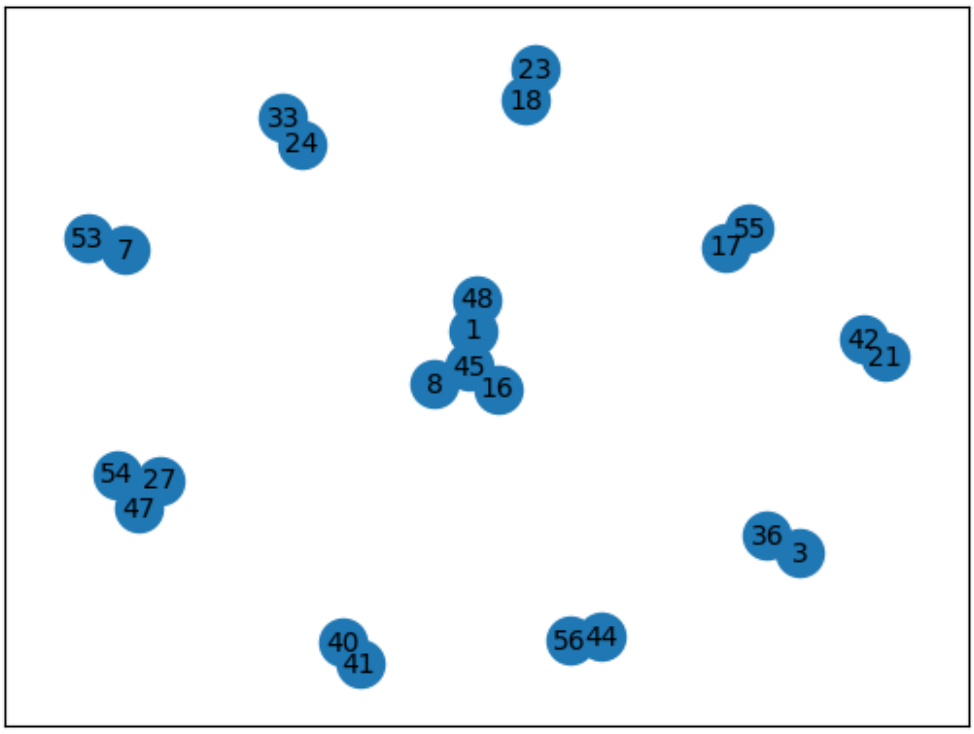





```
percolar(60, .01)
```

```
Formaram-se 46 grupos com pelo menos um nó:
[[0], [2], [4], [5], [6], [9], [10], [11], [12], [13], [14], [15],
[19], [20], [22], [18, 23], [25], [26], [28], [29], [30], [31], [32],
[24, 33], [34], [35], [3, 36], [37], [38], [39], [40, 41], [42, 21],
[43], [46], [1, 8, 45, 48, 16], [49], [50], [51], [52], [53, 7], [27,
54, 47], [17, 55], [56, 44], [57], [58], [59]]

O maior grupo possui tamanho: 5
[1, 8, 45, 48, 16]
```

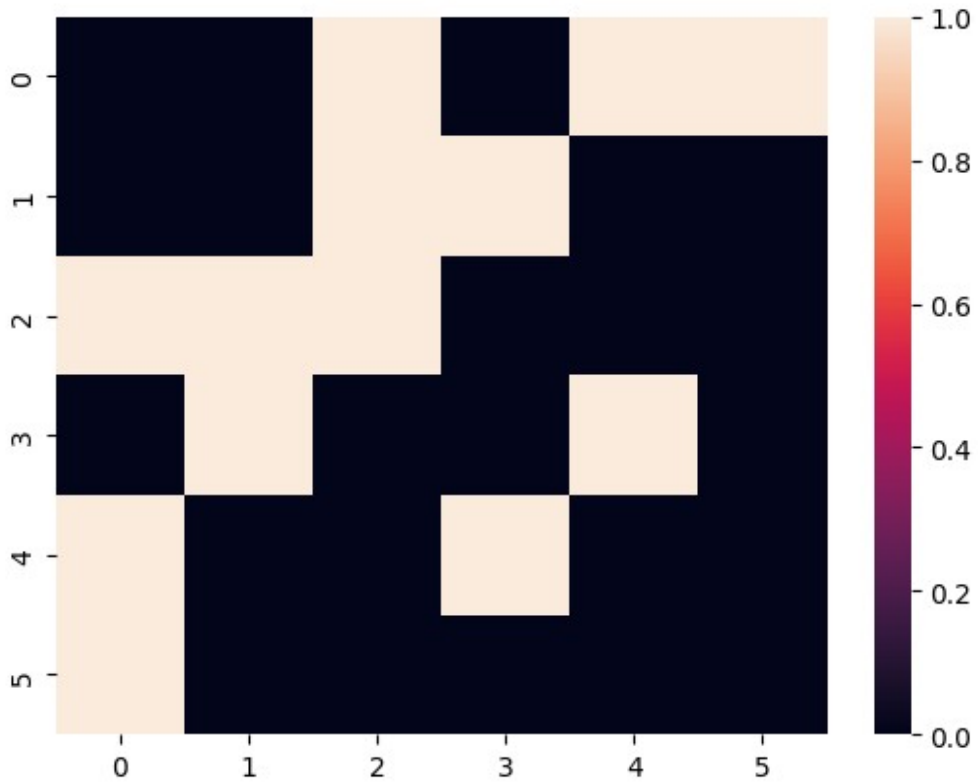





```
#print(resp8)
#desenho(resp8)
resp8=matriz(6,.5)
resp8[1]
#x=separar(resp8)
#ajuntar(x)
#print(x)

set
```


```
{"summary":"{\n  \"name\": \"#print(x)\",\n  \"rows\": 6,\n
\"fields\": [\n    {\n      \"column\": 0,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5477225575051661,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          1.0,\n          0.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": 1,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5163977794943223,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          1.0,\n          0.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": 2,\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.5477225575051661,\n
\"min\": 0.0,\n        \"max\": 1.0,\n        \"num_unique_values\":
2,\n        \"samples\": [\n          0.0,\n          1.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": 3,\n      \"properties\": {\n
```

\"dtype\": \"number\",\n   \"std\": 0.5163977794943223,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   1.0,\n   0.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": 4,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.5163977794943223,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.0,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": 5,\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.408248290463863,\n   \"min\": 0.0,\n   \"max\": 1.0,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   0.0,\n   1.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}



13/06/2024

Fazer a transmissão de informação no maior grupo.

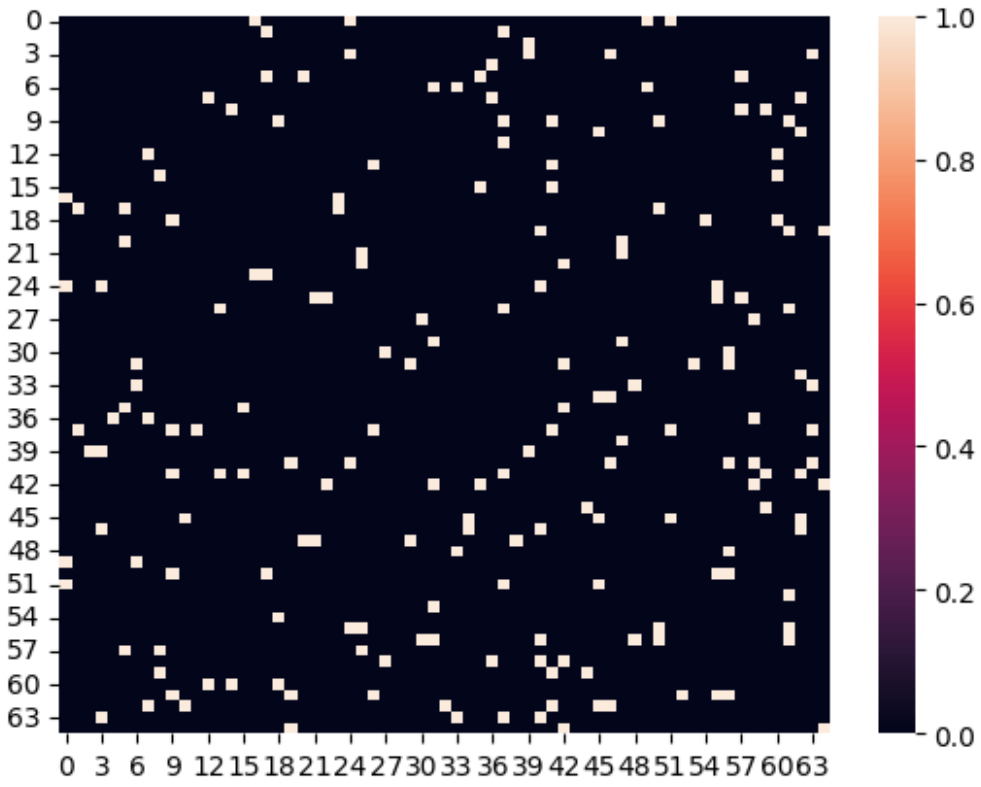
```
import scipy.stats as stata
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import networkx as nx

import itertools
from random import random as rd

def percolar(dimensao, probabilidade):
    """
    Recebe 'parteum()' e desenha o grafo com 'desenho()'.
    """
    x = parteum(dimensao, probabilidade)
    plt.figure(2)
    desenho(x[2])
    plt.show()

    # Faz o desenho e retorna a saída de 'parteum()'
    return x


def parteum(n, p):
    """
    Realiza a percolação.
    n := dimensão da matriz
    p := probabilidade de sucesso
    """
    matriz_adj = matriz(n, p)

    no_em_comum = separar(matriz_adj[1])

    grupos_juntos = ajuntar(no_em_comum)

    #desenho(matriz_adj[1])

    # Retorna OS GRUPOS JUNTOS, o MAIOR GRUPO e a MATRIZ DE ADJ.
    return grupos_juntos[0], grupos_juntos[1], matriz_adj[1]
```

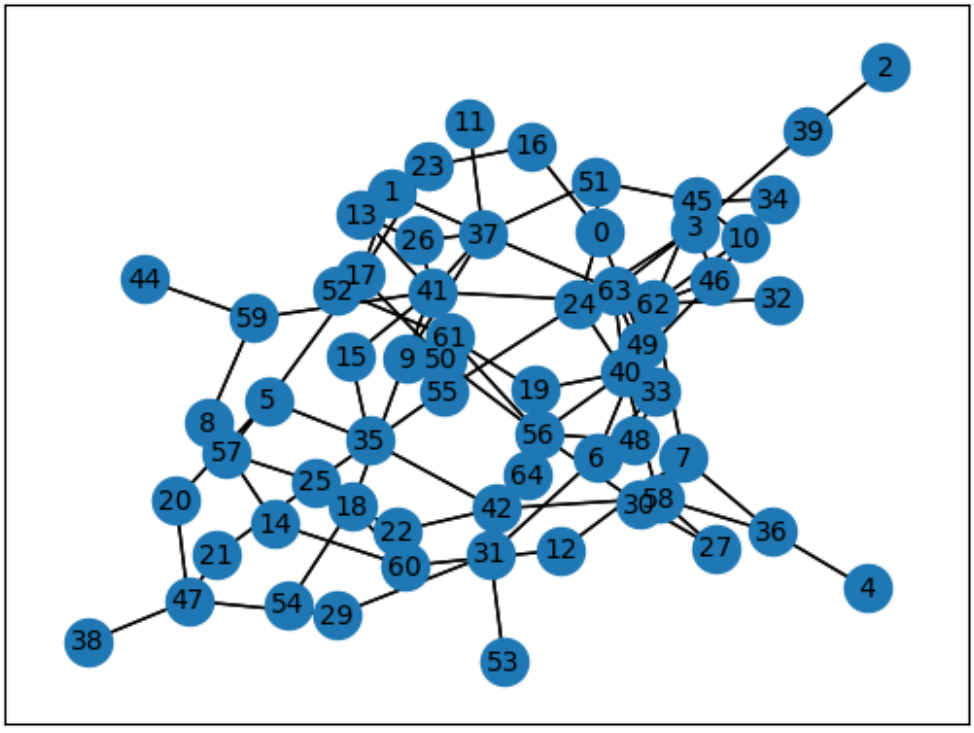
```
def matriz(n, p):
    """
    Forma uma matriz [de adjacências] que indica quando há conexão (1)
entre os
    nós e quando não há (0).
    """
    dados = np.zeros([n, n])
    matriz = pd.DataFrame(dados)

    for i in range(n):
        for j in range(i, n):
            sorteio = stata.uniform.rvs()
            if sorteio <= p:
                matriz.loc[i, j], matriz.loc[j, i] = 1, 1

    mapa_calor = sns.heatmap(matriz)
    #print(mapa_calor)

    return mapa_calor, matriz


def separar(matrizadj):
    """
    Lê uma matriz de adjacências e forma listas com as conexões
formadas por cada
    nó.
    """
    grupos = []
    for i in range(len(matrizadj)):
        atual = []
        for j in range(len(matrizadj)):
            ver = matrizadj.loc[i, j]
            if ver == 1:
                atual.append(j)
        atual.append(i)
        grupos.append(atual)

    return grupos


def ajuntar(listagrupos):
    """
    Recebe uma lista tal qual a formada pela função 'separar()'.
    Agrupa os nós que têm ao menos uma conexão entre si.
    """
    LL = set(itertools.chain.from_iterable(listagrupos))

    for each in LL:
        components = [x for x in listagrupos if each in x]
        for i in components:
```

```
            listagrupos.remove(i)
        listagrupos +=
[list(set(itertools.chain.from_iterable(components)))]

    contador, lmaior = 0, []
    for grupo in listagrupos:
        # Maior grupo
        if len(grupo) > len(lmaior):
            lmaior = grupo

        if len(grupo) > 0:
            contador += 1

    listaf = list(filter(None, listagrupos))
    print("\nFormaram-se", contador, "grupos com pelo menos um nó:")
    print(listaf)

    print("\nO maior grupo possui tamanho:", len(lmaior))
    print(lmaior)

    return listaf, lmaior


def desenho(matrizdf):
    """
    Desenha o grafo [Erdos Renyi] a partir da matriz de adjacências.
    """
    G = nx.DiGraph()
    rotulo = {}
    for i in range(len(matrizdf)):
        for j in range(len(matrizdf)):
            if i != j and matrizdf[i][j] == 1:
                rotulo[j]=j
                G.add_edge(i,j)

    #print(set(list(G.edges)))
    pos = nx.spring_layout(G, seed=3000)  # positions for all nodes

    #return nx.draw_circular(G)
    return nx.draw_networkx(G, pos, arrows=False, labels=rotulo,
font_size=10)

def percorrer1(percolar1, p_transmitir):
    """
    Transmissão de informação com probabilidade de aceitação.
    O transmissor pode contactar cada indivíduo de seu grupo de
maneira aleatória
    o indivíduo tem chance de recusar a informação uma quantidade
finita de vezes.
    """
```

```
    zeta = percolar1
    maiorgrupo = zeta.copy()
    informante = np.random.choice(maiorgrupo, 1, replace=False)[0]
    maiorgrupo.remove(informante)

    rede = [informante]
    iterar = 0
    while len(rede) < len(zeta):
        sorteio = stata.uniform.rvs()
        iterar += 1
        if sorteio < p_transmitir:
            vitima = np.random.choice(maiorgrupo, 1)[0]
            #maiorgrupo.remove(vitima)
            if vitima not in rede:
                rede.append(vitima)

    return iterar

alfa = percolar(65, .05)
```

```
Formaram-se 3 grupos com pelo menos um nó:
[[28], [43], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51,
52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64]]

O maior grupo possui tamanho: 63
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61, 62, 63, 64]
```






```
separar(zeta[2])[0]
```

```
[16, 24, 49, 51, 0]
```

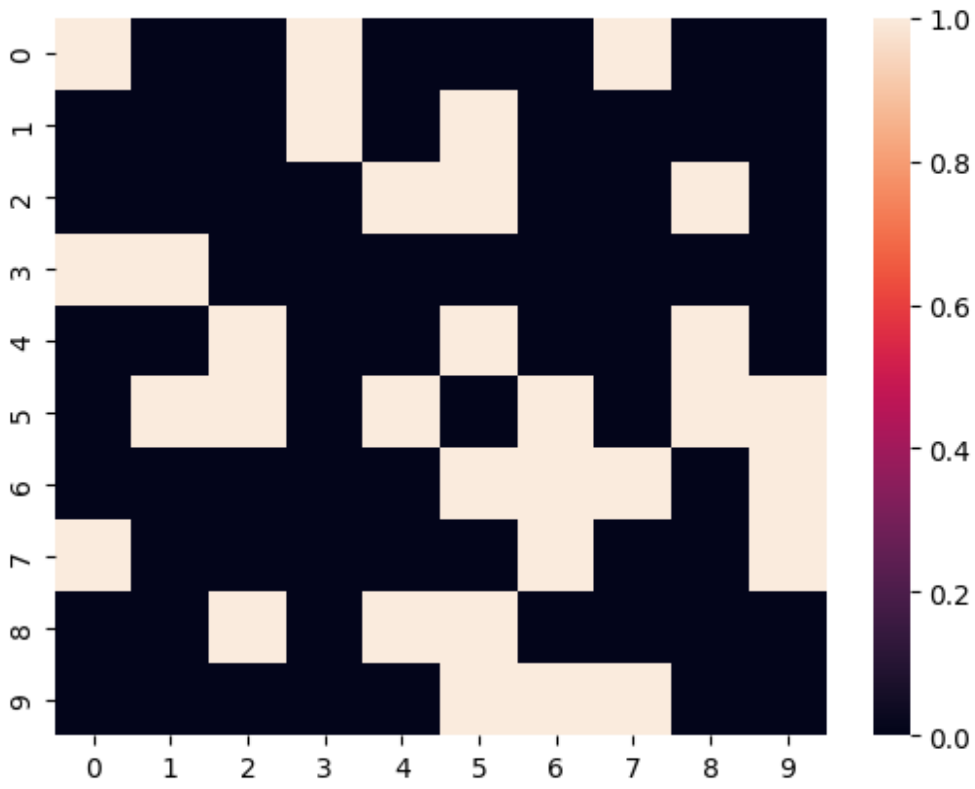
```
def percorrer(percolar, p_transmitir=1):
    """
    Transmissão de informação às ligações de primeiro grau.
    """
    zeta = percolar[1]
    maiorgrupo = zeta.copy()
    informante = np.random.choice(maiorgrupo, 1, replace=False)[0]
    maiorgrupo.remove(informante)

    lig_direta = separar(percolar[2])[informante]

    rede = [informante]
    iterar = 0
    while len(rede) < len(zeta):
        sorteio = stata.uniform.rvs()
        iterar += 1
        if sorteio < p_transmitir:
            vitima = np.random.choice(lig_direta, 1)[0]
            lig_vitima = separar(percolar[2])[vitima]
            lig_direta = [*lig_vitima, *lig_direta]    # PEP 448
            lig_direta.remove(vitima)
            if vitima not in rede:
                rede.append(vitima)

    return iterar

def percorrer3(percolar, p_transmitir=1):
    """
    Sorteio de uma das ligações de primeiro grau, caso um dos nós seja
um
    'informante' os dois passarão a ser.
    """
    zeta = percolar[1]
    maiorgrupo = zeta.copy()
    informante = np.random.choice(maiorgrupo, 1, replace=False)[0]
    maiorgrupo.remove(informante)

    lig_direta = separar(percolar[2])[informante]

    rede = [informante]
    iterar = 0
    while len(rede) < len(zeta):
        '''sorteio = stata.uniform.rvs()
        iterar += 1
        if sorteio < p_transmitir:
            vitima = np.random.choice(lig_direta, 1)[0]
            lig_vitima = separar(percolar[2])[vitima]
            lig_direta = [*lig_vitima, *lig_direta]    # PEP 448
```

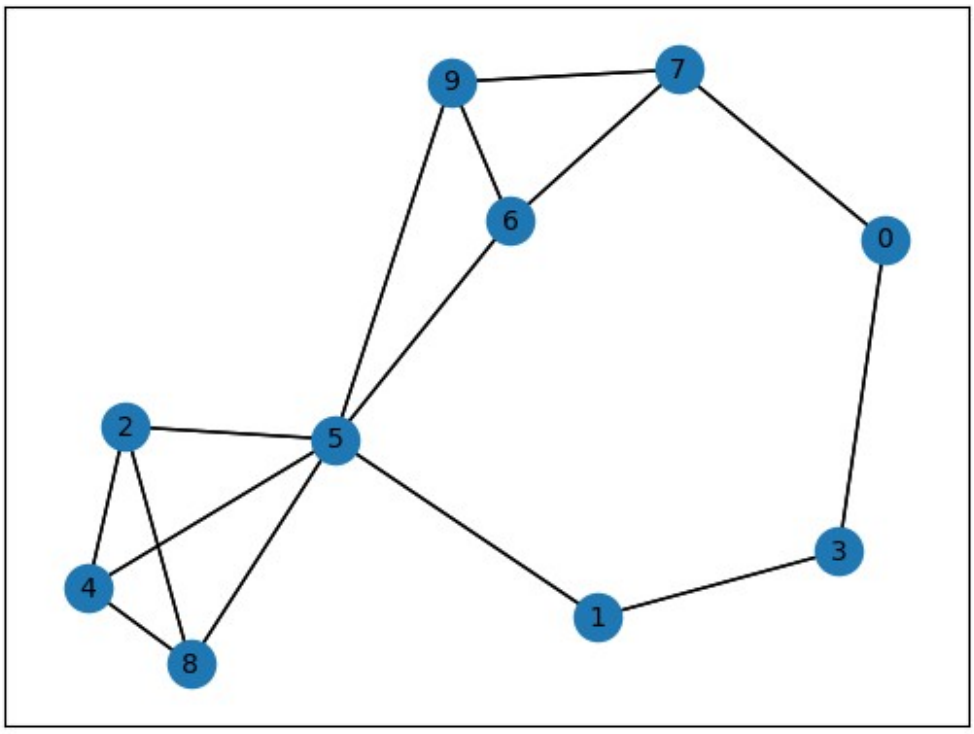
```
            lig_direta.remove(vitima)
            if vitima not in rede:
                rede.append(vitima)'''

    return iterar

resposta = [percorrer1(zeta[1], .5) for i in range(1000)]
#print(zeta)
np.mean(resposta)
```

```
586.068
```

```
alfa = percolar(10, .3)
```

```
Formaram-se 1 grupos com pelo menos um nó:
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]

O maior grupo possui tamanho: 10
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```






```
teste = [percorrer(alfa) for i in range(1000)]
np.mean(teste)
```

```
99.323
```

18/06/2024

Atualizando opiniões

A_1, A_2, A_3, ..., A_m (Tipo I)

B_1, B_2,B_3, ...,B_n (Tipo II)

(A_i, B_j) (Encontro)

Após o encontro (A_i, B_i):

$$\widetilde{A}_i = \frac{A_i + r_A B_i}{1 + r_A} \wedge \widetilde{B}_i = \frac{B_i + r_B A_i}{1 + r_B}$$

$r_A = r_B$ (tendência à média)

$r_A > r_B$ (A muda de opinião mais fácil)

$r_B > r_A$ (B muda de opinião mais fácil)

```
import scipy.stats as stata
import numpy as np
import pandas as pd
#import seaborn as sns
import matplotlib.pyplot as plt
#from scipy.cluster import hierarchy
import matplotlib.pyplot as plt
import networkx as nx

import itertools
from random import random as rd
```

Algoritmo:

```
entrada m, n, ra, rb, nsim
Ai~U(0; 0,1)
Bi~U(0,9; 1)

# intervalo (loc, loc+scale)
stata.uniform.rvs(scale=.1)

0.09916421548517328

def alg(m, n, ra, rb, nsim):
    """
    """
    Ai = list(stata.uniform.rvs(size=m, scale=.1))
```

```
    Bi = list(stata.uniform.rvs(size=n, loc=.9, scale=.1))
    Anovo, Bnovo, resultA, resultB = Ai.copy(), Bi.copy(), [], []

    if m == n:
        for i in range(nsim):
            # Encontro
            e = [np.random.choice([*range(m)], 1)[0],
                np.random.choice([*range(n)], 1)[0]]

            Atil = (Anovo[e[0]] + ra*Bnovo[e[1]])/(1 + ra)
            Btil = (Bnovo[e[1]] + rb*Anovo[e[0]])/(1 + rb)

            Anovo[e[0]], Bnovo[e[1]] = Atil, Btil
            resultA.append(Atil); resultB.append(Btil)

        df = pd.DataFrame(data={"A":Ai, "B":Bi, "Anovo": Anovo,
"Bnovo":Bnovo})
        print(df)

    else:
        davez = []
        for i in range(nsim):
            # Encontro
            e = [np.random.choice([*range(m)], 1)[0],
                np.random.choice([*range(n)], 1)[0]]

            Atil = (Anovo[e[0]] + ra*Bnovo[e[1]])/(1 + ra)
            Btil = (Bnovo[e[1]] + rb*Anovo[e[0]])/(1 + rb)
            davez.append([Anovo[e[0]], Bnovo[e[1]], Atil, Btil])
            resultA.append(Atil); resultB.append(Btil)

        df = pd.DataFrame(data=davez, columns=["A", "B", "Anovo",
"Bnovo"])
        print(df)

        sA = pd.Series([resultA])
        print(sA.cumsum()/pd.Series(np.arange(1, len(sA)+1),
sA.index))

    return np.mean(df["Anovo"]), np.mean(df["Bnovo"])

alg(400, 300, .5, .15, 10**4)
```

```
              A         B     Anovo     Bnovo
0     0.066233  0.923048  0.351838  0.811290
1     0.003761  0.935211  0.314245  0.813718
2     0.090218  0.929207  0.369881  0.819774
3     0.092249  0.922060  0.368853  0.813824
4     0.022394  0.999251  0.348013  0.871835
...        ...       ...       ...       ...
9995  0.022394  0.970572  0.338453  0.846896
```

```
9996  0.033703  0.995969  0.354458  0.870456
9997  0.067268  0.943786  0.359441  0.829458
9998  0.012019  0.996041  0.340026  0.867690
9999  0.051157  0.960883  0.354399  0.842223

[10000 rows x 4 columns]

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/ops/array_ops.py in _na_arithmetic_op(left, right, op, is_cmp)
    170     try:
--> 171         result = func(left, right)
    172     except TypeError:

/usr/local/lib/python3.10/dist-packages/pandas/core/computation/expressions.py in evaluate(op, a, b, use_numexpr)
    238             # error: "None" not callable
--> 239             return _evaluate(op, op_str, a, b)  # type: ignore[misc]
    240     return _evaluate_standard(op, op_str, a, b)

/usr/local/lib/python3.10/dist-packages/pandas/core/computation/expressions.py in _evaluate_numexpr(op, op_str, a, b)
    127     if result is None:
--> 128         result = _evaluate_standard(op, op_str, a, b)
    129

/usr/local/lib/python3.10/dist-packages/pandas/core/computation/expressions.py in _evaluate_standard(op, op_str, a, b)
     69         _store_test_result(False)
---> 70     return op(a, b)
     71

TypeError: unsupported operand type(s) for /: 'list' and 'int'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-4-38dbba81af02> in <cell line: 1>()
----> 1 alg(400, 300, .5, .15, 10**4)

<ipython-input-3-cd8604490b35> in alg(m, n, ra, rb, nsim)
     37
     38         sA = pd.Series([resultA])
---> 39         print(sA.cumsum()/pd.Series(np.arange(1, len(sA)+1), sA.index))
```

```
     40
     41     return np.mean(df["Anovo"]), np.mean(df["Bnovo"])

/usr/local/lib/python3.10/dist-packages/pandas/core/ops/common.py in
new_method(self, other)
     79         other = item_from_zerodim(other)
     80
---> 81         return method(self, other)
     82
     83     return new_method

/usr/local/lib/python3.10/dist-packages/pandas/core/arraylike.py in
__truediv__(self, other)
    208     @unpack_zerodim_and_defer("__truediv__")
    209     def __truediv__(self, other):
--> 210         return self._arith_method(other, operator.truediv)
    211
    212     @unpack_zerodim_and_defer("__rtruediv__")

/usr/local/lib/python3.10/dist-packages/pandas/core/series.py in
_arith_method(self, other, op)
   6110     def _arith_method(self, other, op):
   6111         self, other = ops.align_method_SERIES(self, other)
-> 6112         return base.IndexOpsMixin._arith_method(self, other,
op)
   6113
   6114

/usr/local/lib/python3.10/dist-packages/pandas/core/base.py in
_arith_method(self, other, op)
   1346
   1347         with np.errstate(all="ignore"):
-> 1348             result = ops.arithmetic_op(lvalues, rvalues, op)
   1349
   1350         return self._construct_result(result, name=res_name)

/usr/local/lib/python3.10/dist-packages/pandas/core/ops/array_ops.py
in arithmetic_op(left, right, op)
    230         # error: Argument 1 to "_na_arithmetic_op" has
incompatible type
    231         # "Union[ExtensionArray, ndarray[Any, Any]]"; expected
"ndarray[Any, Any]"
--> 232         res_values = _na_arithmetic_op(left, right, op)  #
type: ignore[arg-type]
    233
    234     return res_values

/usr/local/lib/python3.10/dist-packages/pandas/core/ops/array_ops.py
in _na_arithmetic_op(left, right, op, is_cmp)
    176             # Don't do this for comparisons, as that will
```

```
handle complex numbers
    177             #  incorrectly, see GH#32047
--> 178             result = _masked_arith_op(left, right, op)
    179         else:
    180             raise

/usr/local/lib/python3.10/dist-packages/pandas/core/ops/array_ops.py
in _masked_arith_op(x, y, op)
    114         # See GH#5284, GH#5035, GH#19448 for historical
reference
    115         if mask.any():
--> 116             result[mask] = op(xrav[mask], yrav[mask])
    117
    118     else:

TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

```
alg(19, 19, .05, .95, 10**2)
```

20/06/2024

Firework process (Valdivino)

Em uma fila, cada partícula consegue atingir k partículas posteriores (raio de ação). O interesse é saber quantas partículas são atingidas ao fim do experimento.

```
import scipy.stats as stata
from scipy.stats import geom as gge
import numpy as np
import pandas as pd
#import seaborn as sns
import matplotlib.pyplot as plt
#from scipy.cluster import hierarchy

#import networkx as nx
#import itertools
#from random import random as rd

x = list(gge.rvs(size=10, p=.5))
contagem(x,0)
```

```
10
```

```
# Teste importante
x = 1
cont = 0
while cont < 10:
    if x == 1:
        print("IF")
        x += 1
    elif x <= 2:
        print("ELIF")
        x += 1
    else:
        break
    cont += 1
    print(cont)
```

```
IF
1
ELIF
2
```

```
def geometrica(n, pe):
    """
    Gera uma amostra de tamanho n de uma distribuição Geométrica com
parâmetro 'pe'.
```

```
    Retorna uma lista com a amostra.
    Obs.: a distrib. geométrica do scipy é definida no suporte [1,
infinito),
    portanto houve o deslocamento para obter-se o suporte [0,
infinito).
    """
    x = list(gge.rvs(size=n, p=pe) - 1)
    return x

#geometrica(10, .9)

def falta_suc(lista, indice):
    """
    Checa se um valor k de uma lista possui MENOS de k sucessores.
    """

    if lista[indice] < len(lista[indice: ]):
        xx = True
    else:
        xx = False

    return xx

#sucessores([1,1,2,1,0], 4)


cont = 0
vetor = geometrica(3, .5) + [1]
print(vetor)

while cont < len(vetor):
#while vetor[-1] != 0:
    if vetor[0] == 0:
        vetor = [0]
        break
    elif falta_suc(vetor, cont) == True:
        #print("deu ruim")
        vetor.append(1001)
        vetor[cont + 1: ] = ""
        vetor += geometrica(vetor[cont], .5) + [1]

        print(cont, vetor)
        #cont += 1


    elif falta_suc(vetor, cont) == False:
        cont += 1

    else:
        break
    cont+=1
```

```
    '''elif vetor[-1] != 0:
        # v final
        #print(vetor)
        #break
        continue'''

[2, 1, 0, 1]
0 [2, 1, 0, 1]
1 [2, 1, 0, 1]
2 [2, 1, 0, 1]

def contagem(lista, indice):
    """
    Conta os sucessores de um índice de lista.
    """
    sucessores = len(lista[indice:])

    return sucessores

x = list(gge.rvs(size=10, p=.5))
while lista[-1] == 0 and :


def n2(prob):
    """"""
    raio = list(gge.rvs(size=3, p=prob) - 1)
    contador, raios = 0, raio
    while True:
        if raios[0] == 0:
            print('ja foi')
            raios = [0]
            break
        else:
            try:
                while raios[-1] != 0:
                    if raios[contador] > len(raios[contador: ]):
                        ademais = list(gge.rvs(size=raios[contador] +
3, p=prob))
                        raios[contador + 1:] = ""
                        raios += ademais
                        contador += 1

                    else:
                        contador += 1

            except IndexError:
                raios += list(gge.rvs(size=1, p=prob))
                break

        print(raios)
```

```
    print(raios)

n2(.5)

def maisnovodia2(prob):
    """"""
    raio = list(gge.rvs(size=3, p=prob) - 1)
    contador, raios = 0, raio
    while True:
        if raios[0] == 0:
            print('ja foi')
            raios = [0]
            break
        else:
            try:
                if raios[contador] > len(raios[contador: ]):
                    ademais = list(gge.rvs(size=raios[contador] + 1,
p=prob))
                    raios[contador + 1:] = ""
                    raios += ademais
                    contador += 1

                else:
                    contador += 1

            except IndexError:
                sorteio = gge.rvs(p=prob)
                if sorteio == 0:
                    raios += [0]
                    break
                else:
                    raios += [sorteio]
                    break
                contador += 1

        print(raios)

    #print(raios)

def novodia2(prob):
    """"""
    raio = list(gge.rvs(size=3, p=prob) - 1)
    contador, raios = 0, raio
    while True:
        if raios[0] == 0:
            print('ja foi')
            raios = [0]
            break
        else:
```

```
            try:
                if raios[contador] > len(raios[contador: ]):
                    ademais = list(gge.rvs(size=raios[contador] + 1,
p=prob))
                    raios[contador + 1:] = ""
                    raios += ademais
                    contador += 1

                else:
                    contador += 1

            except IndexError:
                raios += list(gge.rvs(size=1, p=prob))
                break

        print(raios)

    print(raios)

novodia2(.3)
```

```
[1, 3, 3]
[1, 3, 3, 1, 2, 1]
[1, 3, 3, 1, 2, 1]
[1, 3, 3, 1, 2, 1]
[1, 3, 3, 1, 2, 1]
[1, 3, 3, 1, 2, 1]
[1, 3, 3, 1, 2, 1, 9]
```

```
def dia2(prob):
    """"""
    raio = list(gge.rvs(size=3, p=prob) - 1)
    contador, raios = 0, raio
    while True:
        if raios[0] == 0:
            print('ja foi')
            x = 0
            break
        else:
            if raios[contador] > len(raios[contador: ]):
                ademais = list(gge.rvs(size=raios[contador] + 3,
p=prob))
                raios[contador + 1:] = ""
                raios += ademais
                contador += 1

            else: # raios[contador] < len(raios[contador: ]):
                contador += 1

            #elif raios[contador] == 0:
            #    break
```

```
            print(raios)

    print(x)



dia2(.5)
```

```
[4, 2, 2, 4, 2, 4, 1, 4]
[4, 2, 2, 4, 2, 4, 1, 4]
[4, 2, 2, 4, 2, 4, 1, 4]
[4, 2, 2, 4, 2, 4, 1, 4]
[4, 2, 2, 4, 2, 4, 1, 4]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 3]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]
[4, 2, 2, 4, 2, 4, 1, 1, 1, 1, 7, 1, 1, 2, 4, 2, 1, 2, 1, 4, 7, 1, 3,
3, 3, 2, 1]

-----------------------------------------------------------------------
-----
IndexError                                    Traceback (most recent call
last)
<ipython-input-4-64acf8d01edf> in <cell line: 1>()
```

```
----> 1 dia2(.5)

<ipython-input-2-0df919e0fbfb> in dia2(prob)
      9             break
     10         else:
---> 11             if raios[contador] > len(raios[contador: ]):
     12                 ademais = list(gge.rvs(size=raios[contador] +
3, p=prob))
     13                 raios[contador + 1:] = ""

IndexError: list index out of range
```

```
def f(prob):
    """
    """
    raio = gge.rvs(prob) - 1
    raios = [raio]

    if raio == 0:
        x = 0

    else:
        ademais = list(gge.rvs(size=raio, p=prob) - 1)
        raios += ademais

        while True:
            for i in range(len(raios)):
                num = raios[i]
                if num <= len(raios[i:]):
                    pass
                else:
                    raios[i+1:] = ""
                    add = list(gge.rvs(size=num, p=prob) - 1)
                    raios += add
            break



    print(raios)
```

```
z=[1,2,3,4]
z[1:]=""
z
```

```
[1]
```

```
f(.25)
```

```
[5, 1, 6, 1, 5, 3, 3, 0, 11]
```

```
def rg(prob=.15):
    ""
    ""
    raio = stata.geom.rvs(size=100, prob) - 1
    if raio == 0:
        print("já foi")
        x = 0
    else:
        raios = [0] * raio
        for i in range(len(raios)):
            raios = [raio] + raios

    print(raio, raios)


rg()
```

```
11 [11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
teste = stata.geom.rvs(size=100, p=.5) - 1
```

```
array([2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 2, 1, 0, 1, 0, 0,
1,
       6, 0, 6, 0, 0, 0, 3, 0, 0, 0, 0, 4, 1, 2, 0, 0, 5, 0, 3, 0, 0,
0,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
0,
       1, 1, 1, 0, 0, 1, 0, 1, 3, 0, 1, 1, 0, 0, 0, 3, 1, 2, 3, 0, 0,
1,
       0, 0, 1, 0, 0, 0, 1, 2, 0, 1, 0, 2])
```

```
for indice in range(len(teste)):
    num = teste[indice]
    maximo = max(teste[:indice])

    if num == 0:
        indice

def ensaio(prob=.5):
    """
    """
    raio = stata.geom.rvs(prob)
    raios = [raio]

    while raio - 1 > 0:
        maximo = max(raios)
        raio = stata.geom.rvs(prob)
        #raio = stata.geom.rvs(p=prob)
        raios.append(raio)

    print(raios)
```

```
ensaio(prob=.5)
```

```
[2, 3, 2, 2, 2, 1]
```

```
x = [ensaio(prob=.1) for i in range(10**4)]
#np.mean(x)
```

```
x = [ensaio(prob=.2) for i in range(10**4)]
np.mean(x)
```

```
4.9532
```

```
x = [ensaio(prob=.5) for i in range(10**4)]
np.mean(x)
```

```
2.0095
```

```
x = [ensaio(prob=.9) for i in range(10**4)]
np.mean(x)
```

```
1.1141
```

```
np.mean(stata.geom.rvs(size=100, p=.2))
```

```
5.1
```

```
def ensaio2(prob=.5):
    """
    """
    raios = []

    raio = 2
    while raio - 1 > 0:
        raio = stata.geom.rvs(p=prob)
        raios.append(raio)
        #cont += 1

    return len(raios), np.sum(raios)/len(raios)
```

```
xx = [ensaio2(prob=.5)[1] for i in range(10**4)]
print(xx[1:10])
np.mean(xx)
```

```
[1.0, 1.0, 1.0, 1.0, 4.333333333333333, 1.0, 2.75, 1.0,
1.6666666666666667]
```

```
1.61117193001443
```

```
xx = [ensaio2(prob=.9)[1] for i in range(10**4)]
print(xx[1:10])
np.mean(xx)
```

```
[1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

```
1.0587999999999997
```

```
xx = [ensaio2(prob=.2)[1] for i in range(10**4)]
print(xx[1:10])
np.mean(xx)
```

```
[3.6666666666666665, 8.0, 4.5, 1.0, 1.0, 2.5, 4.25, 1.0, 1.0]
```

```
3.9612350475444598
```

```
ss = [1, 4]
max(ss)
```

```
4
```

```
def ensaioc(prob=.5):
    """
    """
    raios = []

    raio = 2
    while raio - 1 > 0:
        raio = stata.cauchy.rvs()
        raios.append(raio)
        #cont += 1

    return len(raios), np.sum(raios)/len(raios)
```

```
xx = [ensaio2(prob=.2)[0] for i in range(10**4)]
xx
```

```
[3,
 7,
 2,
 2,
 7,
 4,
 4,
 4,
 2,
 2,
 1,
 3,
 3,
 1,
 2,
 9,
 2,
 4,
 10,
 2,
```

```
2,
14,
2,
13,
7,
8,
6,
15,
10,
5,
8,
4,
1,
20,
2,
1,
1,
1,
10,
1,
10,
1,
1,
3,
6,
1,
2,
5,
3,
1,
6,
1,
2,
9,
6,
8,
3,
2,
1,
2,
3,
8,
16,
2,
2,
4,
6,
1,
21,
```

```
1,
11,
11,
2,
4,
3,
12,
3,
9,
1,
9,
9,
9,
8,
1,
1,
8,
4,
10,
6,
3,
2,
3,
2,
8,
8,
2,
2,
7,
3,
4,
3,
3,
3,
6,
4,
1,
11,
2,
4,
2,
3,
2,
14,
11,
1,
1,
8,
9,
```

```
11,
1,
8,
6,
1,
2,
2,
5,
1,
3,
3,
4,
4,
10,
1,
3,
3,
6,
10,
15,
2,
1,
8,
7,
17,
1,
12,
1,
3,
2,
8,
1,
7,
1,
8,
1,
10,
3,
5,
6,
3,
3,
5,
1,
5,
2,
3,
7,
1,
```

```
14,
1,
2,
3,
3,
7,
10,
1,
13,
3,
2,
3,
23,
4,
4,
16,
1,
2,
1,
2,
8,
11,
1,
11,
27,
16,
2,
19,
1,
4,
11,
2,
3,
15,
1,
2,
2,
3,
4,
2,
2,
2,
13,
5,
2,
2,
1,
14,
2,
```

```
5,
4,
2,
2,
7,
3,
1,
5,
3,
1,
1,
2,
1,
2,
7,
10,
5,
11,
1,
6,
7,
12,
2,
5,
7,
13,
5,
5,
6,
6,
1,
4,
1,
2,
1,
2,
15,
4,
2,
10,
14,
4,
3,
3,
2,
4,
3,
2,
5,
```

```
2,
1,
1,
19,
4,
5,
13,
1,
5,
2,
6,
5,
1,
2,
1,
5,
9,
1,
9,
21,
4,
2,
1,
4,
21,
2,
3,
15,
1,
13,
9,
16,
14,
8,
2,
13,
2,
2,
1,
4,
2,
2,
24,
1,
3,
6,
4,
4,
1,
```

```
14,
7,
5,
2,
6,
4,
3,
1,
10,
1,
6,
3,
13,
15,
12,
14,
1,
3,
9,
4,
4,
1,
4,
1,
6,
4,
3,
3,
1,
5,
2,
3,
3,
1,
12,
1,
1,
6,
7,
3,
2,
4,
3,
5,
11,
5,
1,
5,
1,
```

```
8,
1,
4,
4,
4,
11,
12,
1,
7,
7,
3,
6,
3,
13,
1,
4,
1,
11,
1,
1,
2,
3,
1,
3,
6,
1,
2,
2,
2,
1,
1,
17,
2,
8,
1,
4,
6,
4,
2,
7,
18,
3,
1,
1,
2,
8,
3,
14,
1,
```

```
2,
4,
1,
2,
4,
7,
5,
3,
10,
4,
1,
9,
15,
1,
3,
2,
1,
4,
2,
1,
3,
4,
1,
6,
1,
1,
1,
1,
8,
1,
9,
3,
1,
6,
5,
3,
11,
6,
7,
6,
1,
2,
2,
6,
6,
2,
2,
7,
10,
```

```
2,
9,
2,
11,
2,
1,
4,
3,
1,
5,
6,
3,
1,
4,
5,
7,
2,
9,
7,
6,
4,
4,
1,
5,
1,
2,
1,
2,
1,
1,
4,
4,
1,
1,
5,
3,
5,
3,
2,
3,
17,
2,
1,
4,
1,
19,
6,
1,
10,
13,
```

```
3,
24,
4,
1,
7,
1,
6,
4,
11,
1,
2,
6,
11,
5,
4,
1,
2,
4,
3,
5,
4,
1,
4,
2,
3,
8,
7,
3,
1,
2,
2,
4,
6,
6,
12,
2,
8,
3,
1,
6,
4,
8,
7,
5,
2,
14,
3,
3,
3,
```

```
2,
11,
5,
5,
1,
5,
18,
2,
3,
15,
7,
6,
6,
1,
13,
14,
4,
1,
1,
9,
7,
2,
7,
1,
7,
11,
5,
1,
5,
19,
4,
6,
5,
1,
9,
6,
2,
6,
10,
3,
3,
7,
3,
7,
3,
1,
7,
2,
5,
```

```
2,
1,
1,
2,
5,
12,
1,
5,
7,
3,
7,
4,
7,
4,
15,
12,
2,
5,
5,
4,
12,
2,
3,
4,
4,
10,
10,
4,
4,
8,
10,
7,
4,
1,
9,
2,
3,
1,
6,
7,
5,
3,
2,
2,
2,
9,
6,
15,
7,
```

```
2,
2,
6,
1,
3,
7,
5,
9,
2,
1,
4,
33,
5,
5,
2,
19,
1,
3,
1,
2,
5,
27,
2,
1,
2,
1,
5,
4,
5,
6,
4,
5,
6,
1,
1,
7,
7,
4,
3,
1,
1,
9,
6,
4,
4,
8,
5,
12,
6,
```

```
3,
2,
1,
1,
2,
9,
3,
2,
2,
2,
9,
6,
5,
1,
11,
3,
7,
7,
3,
5,
2,
4,
4,
2,
2,
3,
23,
8,
9,
1,
2,
36,
3,
1,
5,
2,
5,
9,
2,
11,
7,
3,
7,
1,
1,
17,
5,
3,
4,
```

```
3,
2,
10,
1,
12,
2,
1,
7,
5,
10,
2,
7,
3,
1,
7,
2,
8,
9,
1,
5,
1,
4,
2,
3,
5,
8,
2,
4,
6,
3,
1,
4,
1,
2,
1,
2,
4,
1,
4,
2,
13,
10,
2,
3,
2,
4,
18,
1,
1,
```

```
11,
3,
3,
20,
5,
6,
12,
3,
2,
9,
2,
3,
7,
1,
3,
1,
3,
10,
7,
10,
7,
2,
9,
4,
2,
2,
6,
2,
3,
1,
2,
2,
14,
6,
1,
3,
1,
2,
15,
10,
7,
2,
4,
8,
2,
1,
1,
13,
1,
```

```
1,
6,
2,
9,
3,
7,
7,
4,
7,
11,
1,
20,
21,
1,
4,
1,
2,
1,
3,
4,
6,
2,
2,
2,
2,
3,
1,
3,
4,
12,
4,
16,
2,
4,
9,
7,
7,
10,
2,
4,
2,
1,
3,
4,
5,
8,
2,
5,
4,
```

```
11,
3,
3,
19,
3,
1,
7,
8,
4,
10,
5,
3,
1,
4,
13,
3,
22,
6,
1,
1,
1,
4,
2,
5,
7,
5,
16,
12,
1,
4,
5,
1,
4,
1,
1,
1,
2,
2,
8,
3,
8,
1,
2,
3,
2,
5,
8,
4,
7,
```

```
5,
3,
22,
4,
23,
21,
5,
10,
13,
3,
2,
2,
1,
14,
7,
1,
2,
1,
4,
9,
4,
13,
3,
2,
3,
10,
1,
3,
7,
3,
2,
2,
3,
1,
12,
10,
12,
5,
1,
19,
4,
11,
5,
14,
2,
1,
2,
1,
...]
```

16/07/2024

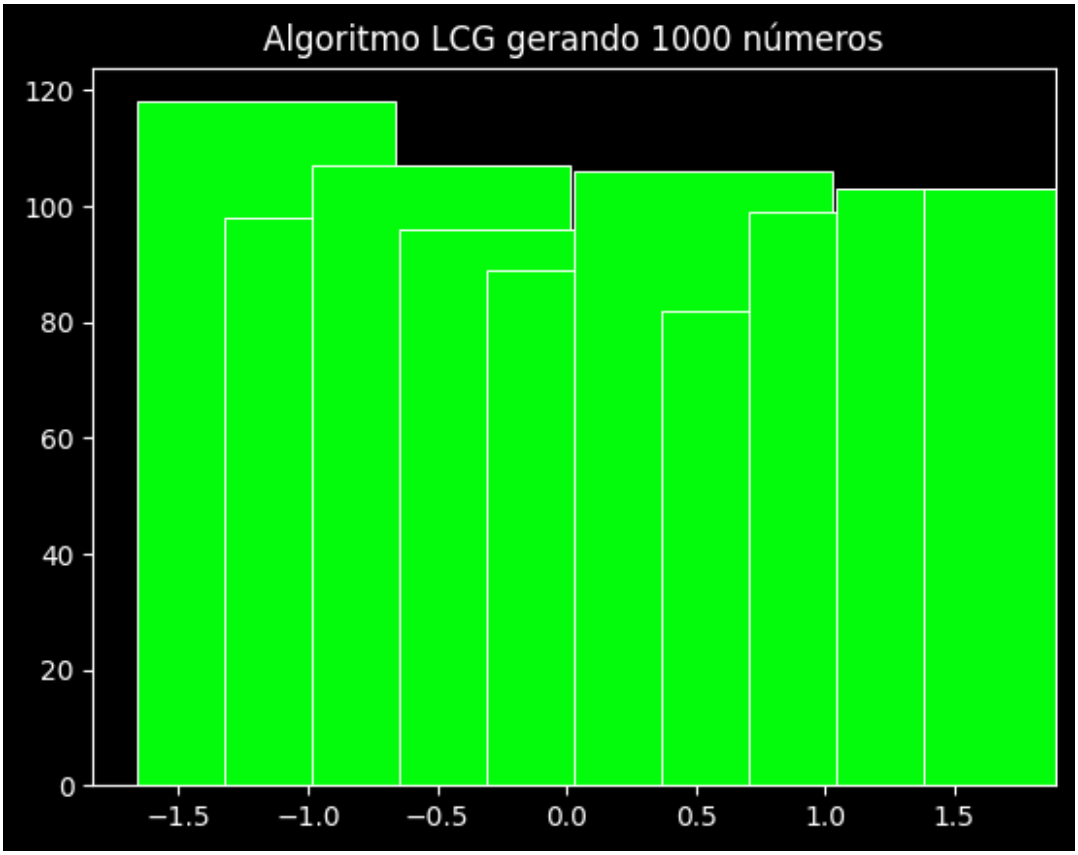
```
import numpy as np
import matplotlib.pyplot as plt
```

LCG (Geradores congruentes lineares)

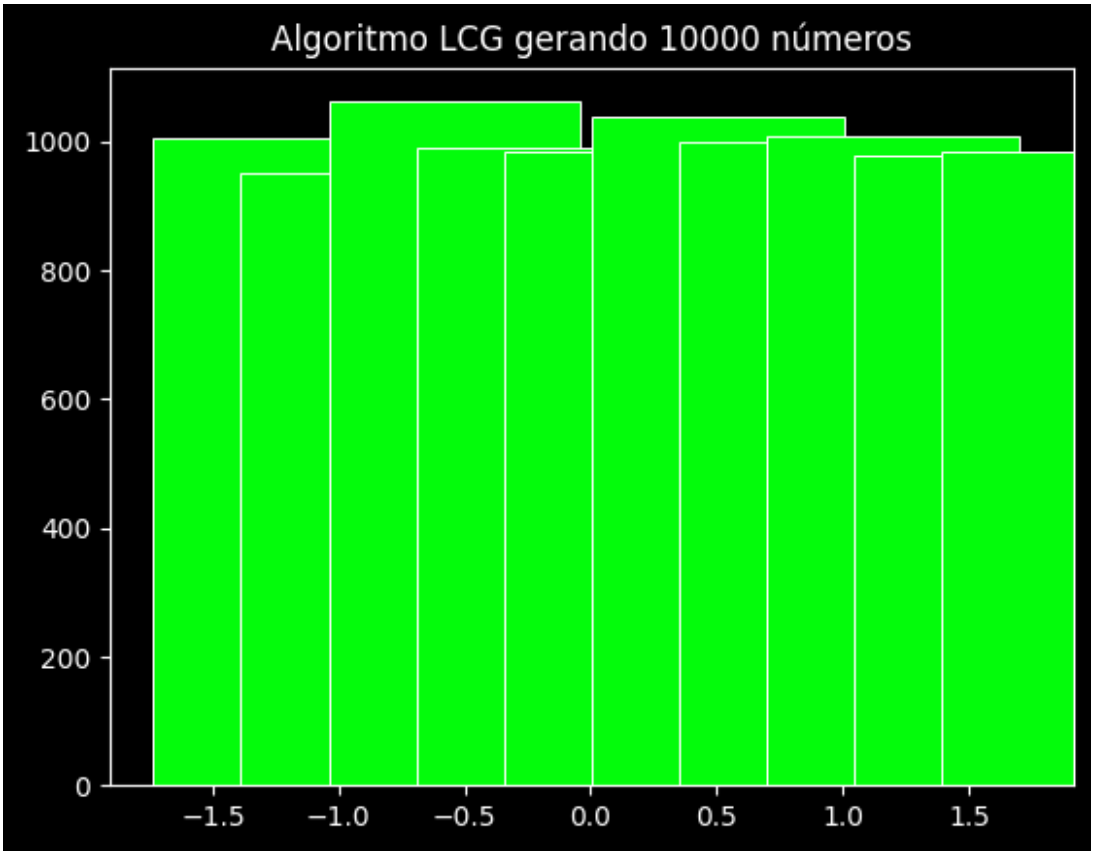
$$x_{i+1} = \left(A \cdot x_i + C \right) mod\ M$$

1. algoritmo para calcular o período
2. gerar os nº a partir da fórmula
3. construir histogramam
4. teste de aderência

```
def lcg(a,c, m, n, plot=True):
    """
    """
    x = [0]
    for i in range(n):
        x.append((a*x[i]+c)%m)

    if plot == True:
        plt.style.use('dark_background')
        fig, ax = plt.subplots()
        ax.hist((x-np.mean(x))/np.std(x), width=1,
                color="#03fc0b", edgecolor="white", linewidth=0.7)
        ax.set_title(f"Algoritmo LCG gerando {n} números")
        plt.show()
    print(f"A={a}, C={c}, M={m}")

    return x

a=lcg(40692,127,10**7,10**3)
```

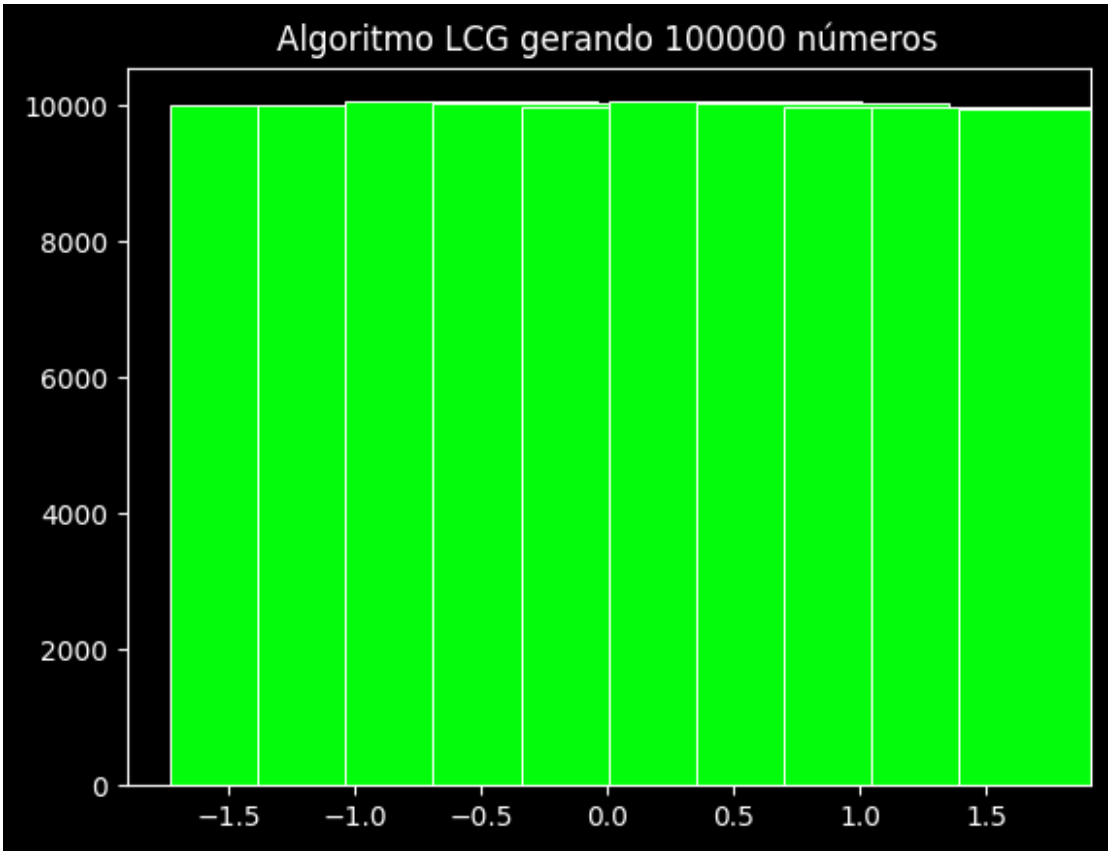



```
A=40692, C=127, M=10000000
b=lcg(40692,127,10**7,10**4)
```




```
A=40692, C=127, M=10000000
```

```
lcg(40692,127,10**7,10**5)
```




```
A=40692, C=127, M=10000000

[0,
 127,
 5168011,
 6703739,
 8547515,
 5480507,
 2790971,
 192059,
 5264955,
 1548987,
 1379131,
 9598779,
 3515195,
 315067,
 706491,
 8531899,
 34235,
 3090747,
 8677051,
 6559419,
 5878075,
 628027,
```

```
5674811,
9409339,
4822715,
5918907,
2163771,
8169659,
9764155,
2995387,
8287931,
2488379,
7118395,
1729467,
5471291,
7773499,
9221435,
8633147,
17851,
6393019,
4729275,
3658427,
8711611,
2874939,
7017915,
2997307,
6416571,
3107259,
583355,
7881787,
5676731,
7537979,
5441595,
9383867,
8316091,
8375099,
9528635,
9215547,
9038651,
786619,
9100475,
6528827,
1028411,
8100539,
7133115,
715707,
3549371,
1004859,
9722555,
208187,
1545531,
```

```
747579,
484795,
7278267,
7240891,
6336699,
2955835,
8837947,
3739451,
5740219,
991675,
3239227,
625211,
1086139,
7168315,
3074107,
1562171,
7862459,
9181755,
3974587,
3894331,
8117179,
4247995,
9412667,
245691,
7658299,
1503035,
1500347,
2120251,
7253819,
2402875,
7789627,
5502011,
7831739,
9123515,
4072507,
8454971,
9680059,
960955,
3180987,
723131,
5646779,
8731195,
9787067,
5330491,
8339899,
7170235,
1202747,
2181051,
1327419,
```

```
5334075,
4180027,
3658811,
4337339,
4998715,
7710907,
2227771,
2457659,
7060155,
1827387,
31931,
9336379,
5934395,
2401467,
495291,
4381499,
1957435,
1945147,
1921851,
3961019,
1785275,
6410427,
3095611,
6602939,
6793915,
7989307,
880571,
2195259,
9479355,
3913787,
9820731,
5185979,
7857595,
1255867,
3740091,
1783099,
7864635,
7727547,
9342651,
1154619,
3756475,
8480827,
1812411,
628539,
6509115,
8907707,
2413371,
4892859,
218555,
```

```
3440187,
8089531,
9195579,
6500795,
350267,
3064891,
6544699,
6891835,
2549947,
2443451,
8908219,
3247675,
4391227,
7809211,
2414139,
6144315,
4466107,
4826171,
6550459,
1277755,
4406587,
2838331,
7365179,
3863995,
3684667,
6469691,
4666299,
1039035,
412347,
9224251,
3221819,
2258875,
8141627,
9086011,
7959739,
7699515,
8664507,
6118971,
3168059,
4656955,
812987,
2067131,
5694779,
1947195,
5259067,
1954491,
2147899,
2306235,
5314747,
```

```
7685051,
95419,
2790075,
3732027,
3642811,
3265339,
3174715,
5502907,
4291771,
745659,
2356155,
6659387,
3775931,
184379,
2750395,
9073467,
7519291,
4989499,
2693435,
1257147,
5825851,
5529019,
6841275,
5162427,
9479611,
4330939,
4569915,
8981307,
7344571,
5283259,
6375355,
5945787,
5964731,
6833979,
8273595,
9127867,
1164091,
9191099,
4200635,
2239547,
1646651,
5522619,
6412475,
6432827,
4596411,
7156539,
3885115,
3099707,
3277371,
```

```
2780859,
8714555,
2672187,
6633531,
1643579,
516795,
9422267,
888891,
752699,
8827835,
2261947,
3147451,
6076219,
3503675,
1543227,
6993211,
7742139,
3120315,
1858107,
90171,
9238459,
1373755,
838587,
3782331,
613179,
1479995,
3956667,
4693691,
5674299,
8575035,
5324347,
8328251,
3189819,
114875,
4493627,
4670011,
2087739,
4275515,
9256507,
5782971,
656059,
6352955,
4444987,
5411131,
9742779,
3163195,
6731067,
578491,
9955899,
```

```
5442235,
5426747,
5189051,
2863419,
8246075,
9284027,
5626811,
6193339,
9350715,
9294907,
8355771,
3033659,
5652155,
7491387,
9519931,
5032379,
7566395,
1745467,
6543291,
9597499,
1429435,
6569147,
1729851,
1097019,
9897275,
9914427,
7863611,
6058939,
345915,
5973307,
5808571,
2371259,
1271355,
3977787,
4108731,
2481979,
6689595,
2999867,
588091,
599099,
8536635,
2751547,
5950651,
3890619,
7068475,
384827,
9380411,
7684539,
9261115,
```

```
3291707,
6141371,
4668859,
5210555,
7904187,
7177531,
8091579,
2532795,
4494267,
712891,
8960699,
8763835,
7973947,
5851451,
7244219,
1759675,
4695227,
8177211,
7070139,
8096315,
5250107,
7354171,
5926459,
9469755,
3270587,
6726331,
7861179,
7095995,
228667,
4917691,
682299,
4111035,
6236347,
9432251,
7157819,
5970875,
6845627,
2254011,
215739,
8851515,
5848507,
7446971,
2144059,
6048955,
4076987,
755131,
7790779,
2379195,
4203067,
```

```
1202491,
1763899,
6578235,
1538747,
4693051,
9631419,
1702075,
836027,
9610811,
3121339,
3526715,
9086907,
4419771,
9321659,
6948155,
4323387,
7263931,
3880379,
382395,
417467,
7567291,
8205499,
8165435,
7881147,
9633851,
665019,
953275,
666427,
8247611,
1786939,
4121915,
8965307,
6272571,
3459259,
4167355,
8009787,
4252731,
2129979,
3105595,
2871867,
2012091,
6007099,
872635,
9263547,
2254651,
6258619,
5724475,
336827,
6164411,
```

```
2212539,
2637115,
9483707,
1005371,
556859,
9706555,
9136187,
9721531,
8539579,
2548795,
5566267,
2536891,
1168699,
6699835,
9685947,
555451,
2412219,
8015675,
3847227,
1361211,
398139,
1072315,
4642107,
6618171,
6614459,
5565755,
1702587,
1670331,
9109179,
711995,
2500667,
7141691,
9690299,
7647035,
3148347,
2536251,
5125819,
9826875,
5197627,
1838011,
2343739,
1427515,
8440507,
1110971,
7632059,
3744955,
9708987,
8099131,
9838779,
9595195,
```

```
7675067,
3826491,
7571899,
5714235,
3650747,
6197051,
399419,
3158075,
8388027,
5594811,
4049339,
5702715,
4878907,
2483771,
9609659,
6244155,
7155387,
7007931,
6728379,
1198395,
5089467,
591291,
813499,
2901435,
5193147,
9537851,
4233019,
9275,
7418427,
631611,
1514939,
5897915,
7957307,
8736571,
8547259,
5063355,
8041787,
6396731,
5777979,
7521595,
8743867,
5436091,
5415099,
1208635,
1775547,
558651,
2626619,
2380475,
6288827,
```

```
4948411,
740539,
4013115,
1675707,
7869371,
444859,
2202555,
6368187,
4265531,
2987579,
564795,
2638267,
6360891,
7376699,
2635835,
7397947,
7259451,
1580219,
2271675,
8999227,
6545211,
7726139,
2048315,
34107,
7882171,
1302459,
9661755,
6134587,
8614331,
4357179,
2327995,
772667,
1365691,
2698299,
9183035,
6060347,
7640251,
7093819,
1682875,
9549627,
3422011,
8471739,
2003515,
7032507,
6774971,
7120059,
9440955,
1340987,
7443131,
```

```
5886779,
4811195,
7147067,
8450491,
7379899,
2850235,
1762747,
9701051,
5167419,
2614075,
1940027,
3578811,
8977339,
5878715,
6670907,
2547771,
3897659,
3540155,
5987387,
8751931,
3576379,
14395,
5761467,
5615291,
7421499,
5637435,
8505147,
1441851,
1801019,
7065275,
170427,
5015611,
5242939,
5673915,
2949307,
3200571,
7635259,
3959355,
4073787,
540731,
3425979,
9937595,
615867,
860091,
8823099,
9544635,
287547,
862651,
2994619,
```

```
7036475,
8240827,
5732411,
3268539,
3389115,
9867707,
6733371,
4332859,
2698555,
9600187,
809531,
1435579,
6580795,
5710267,
2184891,
7584699,
6571835,
1109947,
5963451,
4748219,
4527675,
151227,
3729211,
9054139,
1024315,
1426107,
1146171,
9990459,
1757755,
6566587,
7558331,
3605179,
1943995,
5044667,
7589691,
9706299,
8719035,
4972347,
4744251,
3061819,
1538875,
9901627,
7006011,
8599739,
579515,
1624507,
4438971,
608059,
3136955,
```

```
8972987,
8787131,
5934779,
8027195,
2619067,
5074491,
1187899,
7986235,
5874747,
5205051,
3935419,
70075,
1492027,
3562811,
7905339,
4054715,
4462907,
4611771,
2185659,
8836155,
819387,
2495931,
4424379,
6830395,
2433467,
2639291,
8029499,
6373435,
7817147,
5345851,
3369019,
2121275,
8922427,
1399611,
2970939,
3449915,
3941307,
9664571,
723259,
855355,
6105787,
6684731,
5073979,
353595,
8487867,
8284091,
6231099,
5880635,
4799547,
```

```
3166651,
7362619,
9692475,
6192827,
8516411,
9796539,
765115,
4059707,
7597371,
2220859,
1194555,
8832187,
9353531,
3883579,
596795,
4782267,
8891,
1792699,
8507835,
821947,
6667451,
1916219,
4783675,
7303227,
2913211,
4382139,
8000315,
8818107,
6410171,
2678459,
1853755,
2998587,
8502331,
6853179,
9559995,
5316667,
5813691,
714299,
6255035,
9884347,
3848251,
3029819,
9394875,
6253627,
2590011,
2727739,
7155515,
2216507,
4102971,
```

```
8096059,
4832955,
2604987,
2131131,
9982779,
9243195,
4091067,
3698491,
8995899,
1122235,
5986747,
2709051,
6703419,
5526075,
7044027,
5546811,
833339,
230715,
8254907,
8675771,
4473659,
2132155,
1651387,
8239931,
9272379,
1646395,
5105467,
1663291,
2637499,
5109435,
3129147,
1249851,
8937019,
5177275,
3674427,
9783611,
4698939,
9225915,
933307,
8128571,
7811259,
5751355,
4137787,
4828731,
721979,
8769595,
2359867,
7708091,
7639099,
```

```
216635,
5311547,
7470651,
5730619,
348475,
144827,
3300411,
324539,
6141115,
4251707,
461371,
4108859,
7690555,
4064187,
9897531,
331579,
2612795,
9854267,
9832891,
699,
8443835,
6533947,
9371451,
3084219,
3039675,
455227,
4097211,
3710139,
2976315,
2210107,
3674171,
9366459,
9949755,
5430587,
1446331,
4101179,
5175995,
1588667,
6037691,
5722299,
1791035,
796347,
4952251,
6997819,
5250875,
8605627,
174011,
855739,
1731515,
```

```
8808507,
5766971,
9584059,
4528955,
2236987,
7475131,
8030779,
8459195,
1563067,
4322491,
803899,
2258235,
2098747,
2213051,
3471419,
8982075,
8596027,
9530811,
7761339,
4406715,
8046907,
4739771,
761659,
3428155,
8483387,
5983931,
8120379,
4462395,
3777467,
2687291,
1245499,
1845435,
4441147,
9153851,
8505019,
6233275,
4426427,
167611,
426939,
3001915,
3925307,
8592571,
8899259,
8647355,
8169787,
4972731,
369979,
5185595,
2231867,
```

```
 9132091,
 3047099,
 2552635,
 1823547,
 3774651,
 8098619,
 9004475,
 96827,
 84411,
 4852539,
 9517115,
 443707,
 5325371,
 9996859,
 2186555,
 5296187,
 2441531,
 779579,
 2628795,
 926267,
 1656891,
 2208699,
 6379835,
 8245947,
 4075451,
 8252219,
 9295675,
 9607227,
 7281211,
 7038139,
 5952315,
 1602107,
 2938171,
 54459,
 6045755,
 3862587,
 6390331,
 5349179,
 8791995,
 3860667,
 8261691,
 4730299,
 5327035,
 7708347,
 8056251,
 4965819,
 ...]

def lcg(a,c, m, n):
    """
```

```
    """
    x = [0, ]
    i = 0

    while x[i] not in x[1:i-1]:
        x.append((a*x[i]+c)%m)

    return x
```