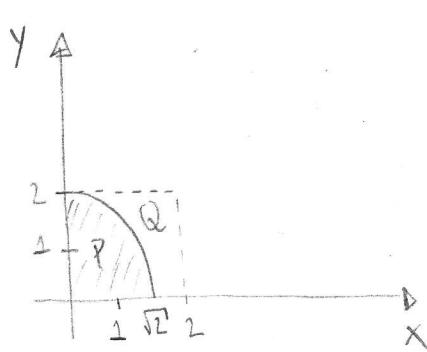


3ª Lista de Exercícios - CPS767

Pergunta 1:

- 1) Seja  $g(x, y)$  uma função indicadora de forma que  $g(x, y) = 1$  se o ponto  $a = (x, y)$  está abaixo da curva que delimita a parábola  $y = -x^2 + 2$  e  $g(x, y) = 0$  caso contrário. Seja  $X$  e  $Y$  duas variáveis aleatórias contínuas com distribuição uniforme  $[0, 2]$ .



$$\text{Tenho que } E[g(x, y)] = \frac{A_p}{A_g} = \frac{A_p}{4}$$

$$A_p = \int_0^{\sqrt{2}} \int_0^{-x^2+2} dy dx = \int_0^{\sqrt{2}} -x^2 + 2 dx = \left[ -\frac{x^3}{3} + 2x \right]_0^{\sqrt{2}}$$

$$A_p = \frac{-2\sqrt{2}}{3} + 2\sqrt{2} = \frac{4\sqrt{2}}{3} \rightarrow E[g(x, y)] = \frac{\sqrt{2}}{3}$$

2)  $\text{Var}[g(x, y)] = E[g^2(x, y)] - [E[g(x, y)]]^2 \rightarrow$  Nesse caso,  $E[g^2(x, y)] = E[g(x, y)]$

$$\text{Var}[g(x, y)] = E[g(x, y)](1 - E[g(x, y)]) \rightarrow \text{Variância de uma v.a. indicadora}$$

$$\text{Var}[g(x, y)] = \frac{\sqrt{2}}{3} \left(1 - \frac{\sqrt{2}}{3}\right) = \frac{\sqrt{2}}{3} \cdot \frac{2}{9} = \frac{2\sqrt{2}}{27} \rightarrow \text{Var}[g(x, y)] \approx 0,25$$

- 3) Seja o estimador  $M_m = \frac{1}{m} \sum_{i=1}^m g(x_i, y_i)$  para estimar a fração de pontos que estão abaixo da parábola. Pela Lei dos Grandes Números,  $M_m$  converge para  $E[g(x, y)]$ , ou seja, converge para  $\frac{\sqrt{2}}{3}$ . Assim,  $\sqrt{2}$  pode ser estimado fazendo  $M_m \cdot 3$ .

O algoritmo para gerar amostras está em anexo!

Para  $m = 10^6$ , o algoritmo estimou  $\sqrt{2} = 1,415508$

- 4) Gráfico também em anexo!

↳ Todos os algoritmos e gráficos estão juntos em outras folhas

## Questão 2:

1)  $f_X(x) = \lambda e^{-\lambda x}$ , com  $\lambda > 0$  e  $x \geq 0 \rightarrow F(x) = \int f_X(x) dx$

$$F(x) = \int_0^x \lambda e^{-\lambda x} dx \rightarrow \text{Substituição } y = -\lambda x \rightarrow F(x) = - \int_0^x -\lambda e^{-\lambda x} dy$$

$$F(x) = - \int_0^{-\lambda x} dy = -e^y = \left[ -e^{-\lambda x} \right]_{x=0}^{x=x} \rightarrow F(x) = 1 - e^{-\lambda x}$$

Gerando uma v.a. uniforme  $U \sim \text{uniforme}(0,1)$ , temos que:  $x = F_x^{-1}(U)$

$$y = 1 - e^{-\lambda x} \rightarrow y - 1 = -e^{-\lambda x} \rightarrow 1 - y = e^{-\lambda x} \rightarrow \ln(1 - y) = -\lambda x \rightarrow x = -\frac{1}{\lambda} \ln(1 - y)$$

Logo, 
$$\boxed{x = -\frac{1}{\lambda} \ln(1-u)}$$
 com  $U \sim \text{unif}(0,1)$

2)  $f_X(x) = \frac{\alpha x_0^\alpha}{x^{\alpha+1}}$ , com  $x_0 > 0$ ,  $\alpha > 0$  e  $x \geq x_0 \rightarrow F(x) = \int f_X(x) dx$

$$F(x) = \int_{x_0}^x \frac{\alpha x_0^\alpha}{x^{\alpha+1}} dx \rightarrow \text{Substituição } y = x^{-\alpha} \rightarrow F(x) = \int_{x_0}^x \alpha x_0^\alpha x^{-\alpha-1} dy$$

$$F(x) = -x_0^\alpha \int_{x_0}^x -\alpha x^{-\alpha-1} dx = -x_0^\alpha \int_{x_0}^x dy \rightarrow F(x) = \left[ -x_0^\alpha x^{-\alpha} \right]_{x=x_0}^{x=x} = 1 - \left( \frac{x_0}{x} \right)^\alpha$$

Gerando uma v.a. uniforme  $U \sim \text{uniforme}(0,1)$ , temos que:  $x = F_x^{-1}(U)$

$$y = 1 - \left( \frac{x_0}{x} \right)^\alpha \rightarrow y = 1 - x_0^\alpha x^{-\alpha} \rightarrow y - 1 = -x_0^\alpha x^{-\alpha} \rightarrow 1 - y = x_0^\alpha x^{-\alpha}$$

$$x^\alpha = \frac{x_0^\alpha}{1-y} \rightarrow x = \sqrt[\alpha]{\frac{x_0^\alpha}{1-y}} = \frac{x_0}{(1-y)^{1/\alpha}} = x_0 (1-y)^{-1/\alpha}$$

Logo, 
$$\boxed{x = x_0 (1-u)^{-1/\alpha}}$$
 com  $U \sim \text{unif}(0,1)$

### Questão 3 :

1) Seja  $g(s)$  uma função indicadora em que  $g(s)=1$  se a permutação  $s$  representa um domínio existente e  $g(s)=0$  caso contrário. Seja  $N$  o número total de permutações (sequências de caracteres) de comprimento  $K$  ou menor  $[a-z]^K$ . Seja  $X$  uma variável aleatória uniforme em  $[1, N]$ . Temos que:

$$E[g(x)] = \sum_{s=1}^N P[X=s] g(s) = \frac{1}{N} \sum_{s=1}^N g(s) \rightarrow P[X=s] = \frac{1}{N}, \text{ para } s \in [1, N]$$

Portanto,  $E[g(x)] = \sum_{i=1}^K \sum_{s=1}^{26^i} \frac{1}{26^i} \cdot g(s)$  porque  $N = \sum_{i=1}^K 26^i$

2)  $\text{Var}(x) = E[x^2] - [E(x)]^2 \rightarrow$  Neste caso,  $E[g^2(x)] = E[g(x)]$

$$\rightarrow \left( \sum_{i=1}^K \sum_{s=1}^{26^i} \frac{1}{26^i} \cdot g(s) \right) - \left( \sum_{i=1}^K \sum_{s=1}^{26^i} \frac{1}{26^i} \cdot g(s) \right)^2 = \text{Var}[g(x)]$$

$$\text{Var}[g(x)] = \sum_{i=1}^K \sum_{s=1}^{26^i} \frac{1}{26^i} g(s) \left[ 1 - \sum_{i=1}^K \sum_{s=1}^{26^i} \frac{1}{26^i} g(s) \right]$$

$$\text{Var}[g(x)] = E[x] \cdot (1 - E(x)) \rightarrow \text{visto que } g(s) \text{ apenas pode assumir os valores } 1 \text{ e } 0!$$

3) Algoritmo em anexo. Total de 68 domínios existentes.  
68 domínios existentes pelo método "todas-possibilidades" do algoritmo

4) Para  $K=4$ ,  $N=475254$ . gráfico em anexo.

## Questão 4 :

Siga  $Z$  uma v.a. com distribuição  $Z \sim N(0, 1)$ ,  $Z = f_Z(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

com  $-\infty < x < \infty$  e  $P[Z \geq z] = P[Z \leq -z]$ .

Ao definirmos uma v.a. contínua uniforme  $U \sim \text{Unif}[0, 1]$ , pode-se aplicar o método da rejeição sabendo que  $P[Z \geq z] = P[Z \leq -z]$ .

$f(x) \leq c g(x)$  para todo  $x \in X \rightarrow$  levando  $g(x) = e^{-x}$ , temos que  $\lambda = 1$  e a amostragem pode ser feita pelo método de transformada inversa, já que  $x = -\ln(1-U)$

$$c \geq \frac{f(x)}{g(x)} \rightarrow c \geq \frac{1}{\sqrt{2\pi}} \cdot \frac{e^{-\frac{x^2}{2}}}{e^{-x}} = \frac{1}{\sqrt{2\pi}} e^{x - \frac{x^2}{2}} \rightarrow \begin{array}{l} \text{Para achar o valor máx} \\ \text{de } c: \text{deriva } c = \text{zero} \end{array}$$

$$\frac{d}{dx} \left( \frac{1}{\sqrt{2\pi}} e^{x - \frac{x^2}{2}} \right) = 0 \rightarrow \left( \frac{1}{\sqrt{2\pi}} e^{x - \frac{x^2}{2}} \right) \cdot (1-x) = 0 \rightarrow x=1 \cdot \text{logo, } c = \sqrt{\frac{1}{2\pi}}$$

$$\text{Achata-se o elemento em que } U \leq \frac{f(x)}{cg(x)} \rightarrow U \leq \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{\frac{\sqrt{1}}{\sqrt{2\pi}} \cdot e^{-x}} = \frac{e^x}{\sqrt{2\pi}}$$

$$\text{Então, } U \leq \frac{e^x}{e^{\frac{x}{2}} \cdot e^{\frac{x^2}{2}}} = \frac{e^x}{e^{\frac{(x+1)^2}{2}}} = e^{\frac{x-(x^2+1)}{2}}$$

3 algoritmos para gerar números aleatórios para  $Z$  é:

1) Amostrar, ou seja, gerar  $U \sim \text{Unif}[0, 1]$ .

2) Aplicar o  $U$  gerado em  $x = -\ln(1-U)$ , se  $U \leq e^{\frac{x-(x^2+1)}{2}}$ .

3) caso não, achar-se o valor de  $X$ . caso contrário, retorna-se ao passo 1

↳ Achar significa assumir  $U$  como um valor possível para  $Z$ !

## Questão 6:

1)  $f(x) = x^\alpha$  com  $\alpha > 0$ ,  $g(\alpha, a, b) = \int_a^b f(x) dx$  com  $0 \leq a \leq b$

$$g(\alpha, a, b) = \int_a^b x^\alpha dx = \left[ \frac{x^{\alpha+1}}{\alpha+1} \right]_a^b$$

$$\boxed{g(\alpha, a, b) = \frac{b^{\alpha+1}}{\alpha+1} - \frac{a^{\alpha+1}}{\alpha+1}}$$

2) Seja  $X$  uma variável aleatória contínua uniforme em  $[0, 1]$ , temos que:

$$E[g(x)] = E[f(x)] = \int_{x=0}^{x=1} f_x(x) \cdot f(x) dx$$

(Como  $f_x(x) = 1$  por ser a densidade da v.a uniforme em  $[0, 1]$ ):

$$E[f(x)] = \int_{x=0}^{x=1} f(x) dx = \left[ \frac{x^{\alpha+1}}{\alpha+1} \right]_0^1 = \frac{1^{\alpha+1}}{\alpha+1} - \frac{0^{\alpha+1}}{\alpha+1}$$

Logo,  $\boxed{E[g(x)] = \frac{1^{\alpha+1}}{\alpha+1}}$

3) Algoritmo em anexo.

Obs: Foram criados 3 algoritmos distintos cuja diferença está somente na plotagem. Não sabia qual era a melhor forma de plotar.

4) Gráfico em anexo.

Obs: Dessa forma foram criados 3 tipos de gráficos também.

## Questão 7:

$S_{k,m}$  é um espaço amostral dado por todos os subconjuntos de tamanho  $k$  dentre  $m$  objetos.

A probabilidade de pegar qualquer um desses subconjuntos é a mesma,  $P = \frac{1}{|S_{k,m}|}$ , ou seja, apresenta uma distribuição uniforme. Os subconjuntos não são dados por:

$[1, \dots, k], [2, \dots, k+1], [3, \dots, k+2], \dots$  Até que todos os subconjuntos

sejam formados, considerando que eles não são dependentes da ordem dos objetos!

Sendo assim,  $P = \frac{1}{|S_{k,m}|} = \frac{1}{C_m^k} = \frac{1}{\text{todas as possíveis combinações de subconjuntos } k}$

Logo, um algoritmo eficiente é: Embaralha, gera um número aleatório de 1 até  $m$ , para gerar amostras →  $\rightarrow [1 \text{ até } m-1]$

retira esse objeto, reorganiza os  $(m-1)$  objetos restantes (com índice sorteado)

gera um novo número aleatório e retira um novo objeto com o índice sorteado. Realiza esse procedimento  $K$  vezes. Depois, após ter sua primeira amostra de tamanho  $K$ , limpa os objetos retirados, embaralha novamente e recomeça até ter o número de amostras (subconjuntos) desejado. Os subconjuntos têm a mesma probabilidade pois:

Dado  $S_m^k = \boxed{1 | 2 | \dots | k | k+1 | k+2 | \dots | m}$

$\underbrace{\hspace{1cm}}_K \quad \underbrace{\hspace{1cm}}_{m-k}$

A primeira parte tem  $K!$  permutações

A segunda parte tem  $(m-K)!$  permutações

$$\text{Assim, } P = K! (m-K)! \prod_{i=0}^{k-1} \frac{1}{(m-i)} \rightarrow P = \frac{K! (m-K)!}{\prod_{i=0}^{k-1} (m-i)} = \frac{k! (m-k)!}{m(m-1)\dots(m-k-1)} = \frac{1}{|S_{k,m}|}$$

Algoritmo: embaralha conjunto para i no intervalo  $[\emptyset, k]$

→ intervalo  $[\emptyset, k]$  ou  $[\emptyset, k-1]$

gera  $m^2$  aleatório no intervalo  $[1, m-i] = a$

retira objeto com índice a

reorganiza objetos restantes

volta para reinsere objetos volta ao primeiro passo

## Questão 8:

- 1) Para estimar  $P[X \geq x]$  com  $x > 0$ , pode-se aplicar um algoritmo de monte carlo simples.
- Seja  $g(x)$  uma função indicadora em que  $g(x) = 1$  se  $x \geq x_c$  e  $g(x) = 0$  caso contrário.
- Seja  $N$  o espaço amostral, a partir de uma v.a.  $X$  com distribuição uniforme  $X \sim \text{Unif}[0, 1]$ .
- É possível escrever um algoritmo para  $E[g(X)]$ . Ele é:

- 1) gerar  $s \sim \text{Unif}(1, N) = 1$
- 2) Aplicar a  $s$  gerada em  $g$ . Ou seja, verificar  $g(s)$ .
- 3) Repetir os passos acima  $m$  vezes.
- 4)  $E[g(s)] = \frac{1}{m} \sum_{i=1}^m g(s_i) = \hat{\mu}_g^m$

Apenas um nome diferente para a amostra gerada para não confundir com os  $X$ 's.

Dessa forma é possível amostrar  $P[X \geq x]$ , no entanto, não é uma forma interessante porque para fazer uma amostragem com alta confiabilidade o  $m$  teria que ter um valor muito grande, já que a probabilidade de  $g(s)=1$  é pequena e diminui

- 2) Variância de estimador via média amostral:  $\text{Var}[\hat{\mu}_g^m] = \frac{\sigma_g^2}{m}$

Assim sabemos  $\sigma_g^2$ , podemos estimá-la a partir de  $\hat{\sigma}_g^2$ . Retirando o viés, temos:

$$\text{Var}[\hat{\mu}_g^m] = \frac{1}{m-1} \sum_{i=1}^{m-1} (g(i) - \hat{\mu}_g^m)^2$$

Sendo  $g(x)$  uma função indicadora  $\rightarrow \text{Var}[g(x)] = E[g(x)].(1 - E[g(x)])$ .

Essa variância apresenta valor pequeno somente se  $E[g(x)]$  for bem alta, o que requer um número muito grande de amostras.

- 3) A ideia de importance sampling é fazer a amostragem a partir de uma distribuição proposta e depois modificar os parâmetros de forma que alcance a distribuição realmente desejada. Seja uma função  $h$  e uma v.a.  $X$  com distribuição dada por  $f$ .

$$\mu_f = E_f[h(X)] = \int_{i=1}^{i=N} h(i)f(i) di \quad \leftarrow f(i) = P[X=i]$$

Seja  $y$  uma v.a. contínua com distribuição  $g$ , tal que  $f(i) > 0 \rightarrow g(i) > 0$ :

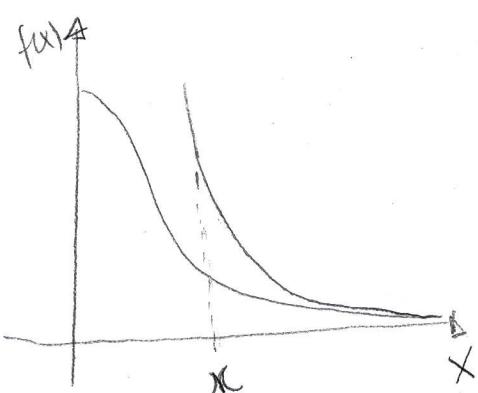
$$\mu_f = \int_{i=1}^{i=N} \frac{h(i)f(i)}{g(i)} g(i) di = E_g \left[ \frac{h(x)f(x)}{g(x)} \right]$$

Se  $g$  for bem escolhida, a Variância do estimador com Importance Sampling pode ser menor que a do estimador original!

Assumindo que  $X$  tem distribuição Normal padrão  $\rightarrow p[X \geq x] = ?$

Assumindo que a função densidade de  $X$  é  $f_X(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}$  (como na questão nº 4).

Fazendo  $g(y) = e^{-(y-x)}$  se  $y \geq x$  e  $g(y) = 0$  caso contrário, temos que  $g(y)$  é uma exponencial deslocada  $x$  para a direita.



$$\frac{f_X(y)}{g(y)} = \frac{1}{\sqrt{2\pi}} e^{\left(\frac{-y^2}{2} + y - x\right)}$$

Decrescente no intervalo  $[x, \infty)$

Isto significa dizer que o seu máximo é em  $y = x$ . Dessa forma, sua variância fica limitada ao segundo momento, como escrito no próximo item, item 4.

4)  $\sigma_{f/g}^2 = \text{Var } g\left[\frac{f(y)}{g(y)}\right] = E_g\left[\left(\frac{f(y)}{g(y)}\right)^2\right] \rightarrow$  Variância depende apenas do segundo momento!

Conforme mencionado no item acima, o máximo ocorre quando  $y = x$  e, conseguintemente, também ocorre a variância máxima! Logo,

$$\sigma_{f/g}^2 = E_g\left[\left(\frac{f(y=x)}{g(y=x)}\right)^2\right] = \frac{1}{\sqrt{2\pi}} e^{\left(\frac{-x^2}{2} + x - x\right)} = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

Se  $x = 5$ , conforme pede o item 5 da questão,

$$\left( \sigma_{f/g}^2 = \frac{1}{\sqrt{2\pi}} e^{-25/2} \right)$$

$\rightarrow$  Esse número é bem pequeno e menor do que o apresentado no item 2 dessa questão!

**Questão 1 – Item 3 – Algoritmo**

# Acessar via GitHub: <https://github.com/cadupsg/mcmc/blob/master/questao1.py>

```
import random
import math as m
import numpy as np
from matplotlib import pyplot as plt

# monte carlo
def monte_carlo(n):

    # numero de amostras dentro da parabola
    amostra = 0

    for i in xrange(0, n):
        # gera uniforme em [0, 2].
        x = random.uniform(0,2)
        y = random.uniform(0,2)
        # verifica se esta dentro da parabola
        if y < (-1*(x**2)) + 2:
            amostra += 1

    return (float(amostra)/n)*3

def mostra_grafico(n):

    # definicao de variaveis
    estim_m = [0 for i in range(n)]
    erro_m = [0 for i in range(n)]

    # calcula o estimador e o erro
    for i in xrange (1, n+1):
        estim_m[i-1] = monte_carlo(i)
        erro_m[i-1] = (abs(estim_m[i-1]-m.sqrt(2)))/m.sqrt(2)
        print estim_m[i-1]

    # parte que cria os graficos
    eixoX = np.linspace(0,n,n)

    fig1, ax1 = plt.subplots()
    ax1.plot(eixoX, erro_m, ls = '-', lw = '1.5', c = 'blue')
    ax1.set_title("Erro Estimador de Nn x n")
    ax1.set_ylabel("Erro")
    ax1.set_xscale("log")
    plt.show()
    plt.close()

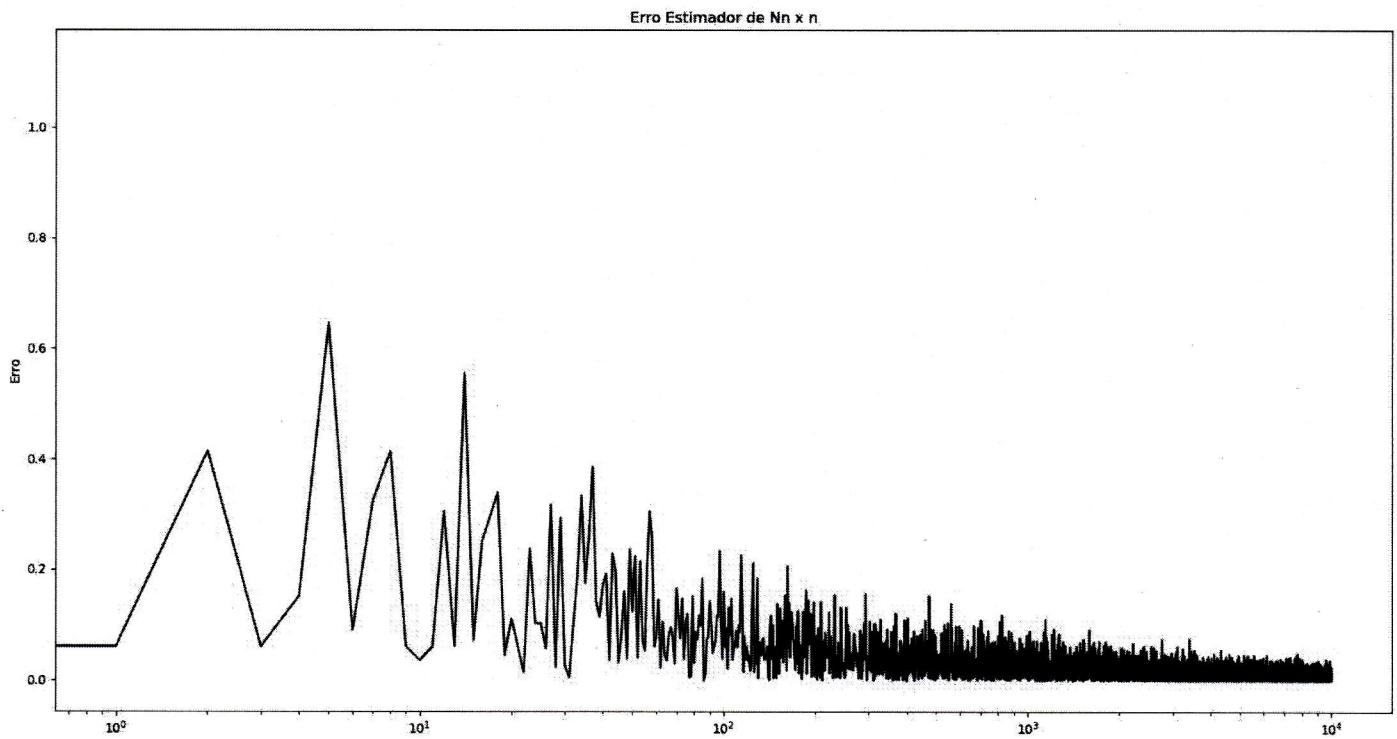
    return

    # calcula a media do estimador
    # Ep = monte_carlo(1000000)
    # print(Ep)

    # cria o grafico
    mostra_grafico(10000)
```

**Questão 1 – Item 4 – Gráfico**

Seja erro =  $(\text{estimador}(n) - \sqrt{2})/\sqrt{2}$  para  $n = [1, 10^4]$  em escala log no eixo x.



### Questão 3 – Item 3 – Algoritmo

```
# Algoritmo completo via GitHub: https://github.com/cadupsg/mcmc/blob/master/questao3.py

# verifica se a URL existe
def checkIfUrlExists(url):

    # testa URLs
    try:
        request = requests.get(url)
        if request.status_code == 200:
            # print('Url ' + url + ' existe')
            return 1
        else:
            # print('Url ' + url + ' nao existe')
            return 0
    except:
        # print('Erro ao pegar a url ' + url)
        return 0

# checa todas as possibilidades existentes
def todas_possibilidades(k):

    # declara as variaveis necessarias
    alphabet = list(string.ascii_lowercase)
    cont = 0
    samples = []

    # cria as permutacoes possiveis
    for i in range(1,k+1):
        permutations = itertools.permutations(alphabet, i) # gera permutacoes de tamanho i
        samples += list(permutations)

    for elem in samples: # para cada permutacao, valida a url
        cont += checkIfUrlExists('http://www.' + ''.join(elem) + '.ufrj.br')
    print cont

    return cont

# metodo para escolher uma permutacao aleatoria
def gera_rand(k):

    # primeiro deve-se gerar um numero aleatorio de 1 ate k
    num = random.randint(1,4)

    # em seguida retorna uma amostra aleatoria de num elementos com reposicao
    return np.random.choice(list(string.ascii_lowercase), num, replace=True)

# metodo de monte carlo para amostragem aleatoria
def monte_carlo(n_amostras, k):

    # calcula a media amostral e a utiliza para estimar o valor de cont
    soma_amostras = 0
    for i in range(n_amostras):
        x = gera_rand(k)
        soma_amostras += checkIfUrlExists('http://www.' + ''.join(x) + '.ufrj.br')
```

```

# imprime quantas amostras sao dominios existentes ate o presente momento
print soma_amostras

return soma_amostras

# gera o grafico do valor do estimador em relacao ao numero de amostras
def mostra_grafico(n, k):

    # definicao de variaveis
    estim_w1 = [0 for i in range(n)]
    estim_w2 = [0 for i in range(n)]

    # calcula o estimador
    for i in xrange(0, n):
        estim_w1[i] = monte_carlo(n, k)

    # parte que cria os graficos
    eixoX = np.linspace(0,n,n)

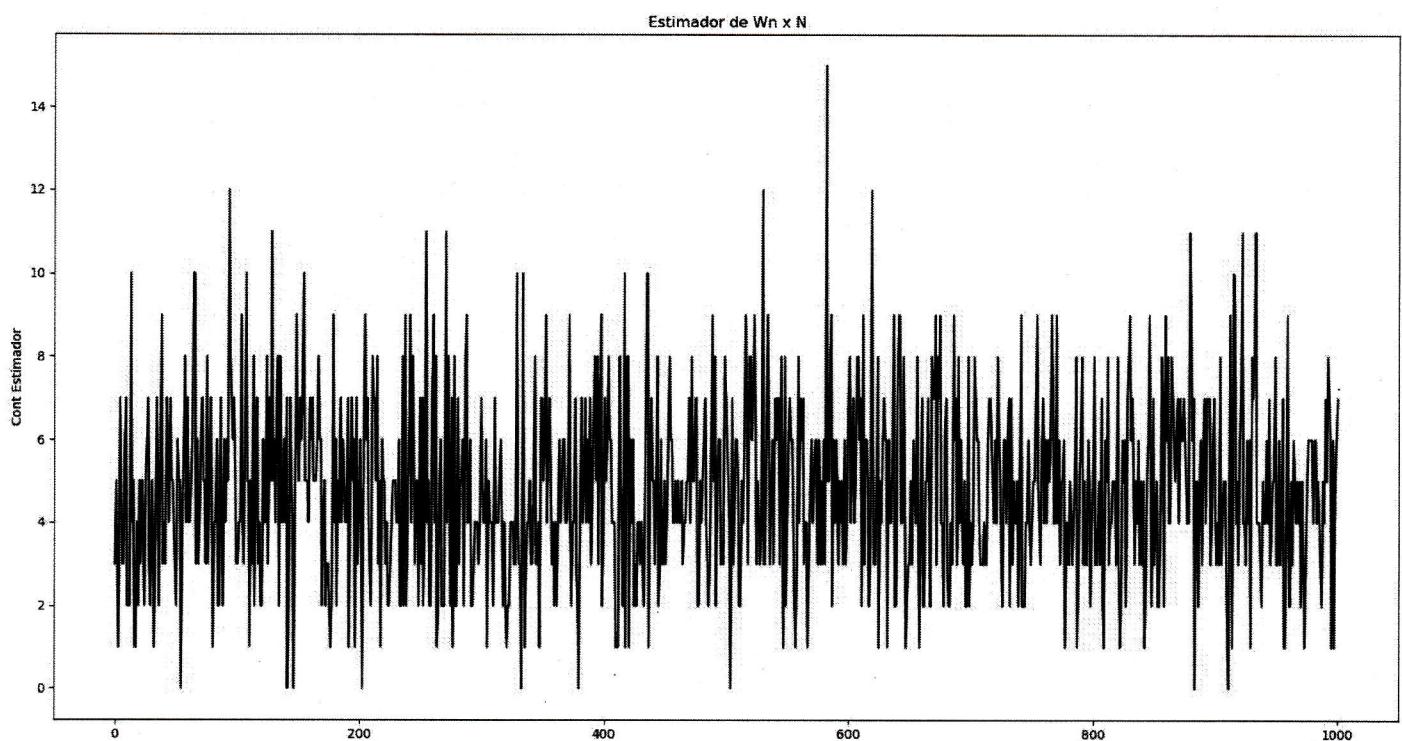
    fig, ax = plt.subplots()
    ax.plot(eixoX, estim_w1, c = 'blue')
    ax.set_title("Estimador de Wn x N")
    ax.set_ylabel("Cont Estimador")

    plt.show()
    plt.close()
    return

```

### Questão 3 – Item 4 – Gráfico

Seja o eixo x o número de amostras (n) e o eixo y o valor do estimador de domínios existentes com relação a n.  
 Seja n = 1000 por motivos computacionais.



### Questão 6 – Item 3 – Algoritmo v1

```
# Algoritmo completo via GitHub: https://github.com/cadupsg/mcmc/blob/master/questao6v1.py

def gera_rand(valor_min, valor_max):
    #gera valor aleatorio a partir de uma distribuicao uniforme entre os valores a e b (limites de integracao)
    limite = valor_max - valor_min
    num = random.uniform(0,1)
    return valor_min + limite*num

def f_de_x(x, alfa):
    # retorna a funcao que sera integrada
    return (x**alfa)

def monte_carlo(n_amostras, alfa, a, b):
    # calcula a media amostral e a utiliza para estimar o valor de g(alfa,a,b)
    soma_amostras = 0
    for i in range(n_amostras):
        x = gera_rand(a, b)
        soma_amostras += f_de_x(x, alfa)

    return (b - a) * float(soma_amostras/n_amostras)

def integral_analitica(alfa, a , b):
    alfa = float(alfa)
    a = float(a)
    b = float(b)
    return (((b***(alfa+1))/(alfa+1))-(a***(alfa+1))/(alfa+1)))

def mostra_grafico(n):
    # definicao de variaveis
    estim_g1 = [[0 for i in range(n)] for ialfa in range(3)]
    erro1 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g2 = [[0 for i in range(n)] for ialfa in range(3)]
    erro2 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g3 = [[0 for i in range(n)] for ialfa in range(3)]
    erro3 = [[0 for i in range(n)] for ialfa in range(3)]

    # calcula os erros para cada alfa e b
    for alfa in xrange(1,4):
        g_analitica1 = integral_analitica(alfa, 0, 1)
        g_analitica2 = integral_analitica(alfa, 0, 2)
        g_analitica3 = integral_analitica(alfa, 0, 4)
        for i in xrange(1, n+1):
            estim_g1[alfa-1][i-1] = monte_carlo(i, alfa, 0, 1)
            erro1[alfa-1][i-1] = (abs(estim_g1[alfa-1][i-1]-g_analitica1))/g_analitica1
            estim_g2[alfa-1][i-1] = monte_carlo(i, alfa, 0, 2)
            erro2[alfa-1][i-1] = (abs(estim_g2[alfa-1][i-1]-g_analitica2))/g_analitica2
            estim_g3[alfa-1][i-1] = monte_carlo(i, alfa, 0, 4)
            erro3[alfa-1][i-1] = (abs(estim_g3[alfa-1][i-1]-g_analitica3))/g_analitica3

    # parte que cria os graficos
    cor = ['red', 'blue', 'magenta', 'k']
    eixoX = np.linspace(0,n,n)
```

```

for i in xrange(0,3):
    # cria figura com 3 graficos
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 20), dpi=80)

    axes[0].plot(eixoX, erro1[i], ls = '-.', lw = '1.5', c = cor[0])
    axes[0].set_ylabel('Erro', color='k')
    axes[0].set_title('b = 1', color='k')
    axes[0].set_xscale("log")

    axes[1].plot(eixoX, erro2[i], ls = '-.', lw = '1.5', c = cor[1])
    axes[1].set_ylabel('Erro', color='k')
    axes[1].set_title('b = 2', color='k')
    axes[1].set_xscale("log")

    axes[2].plot(eixoX, erro3[i], ls = '-.', lw = '1.5', c = cor[2])
    axes[2].set_ylabel('Erro', color='k')
    axes[2].set_title('b = 4', color='k')
    axes[2].set_xscale("log")

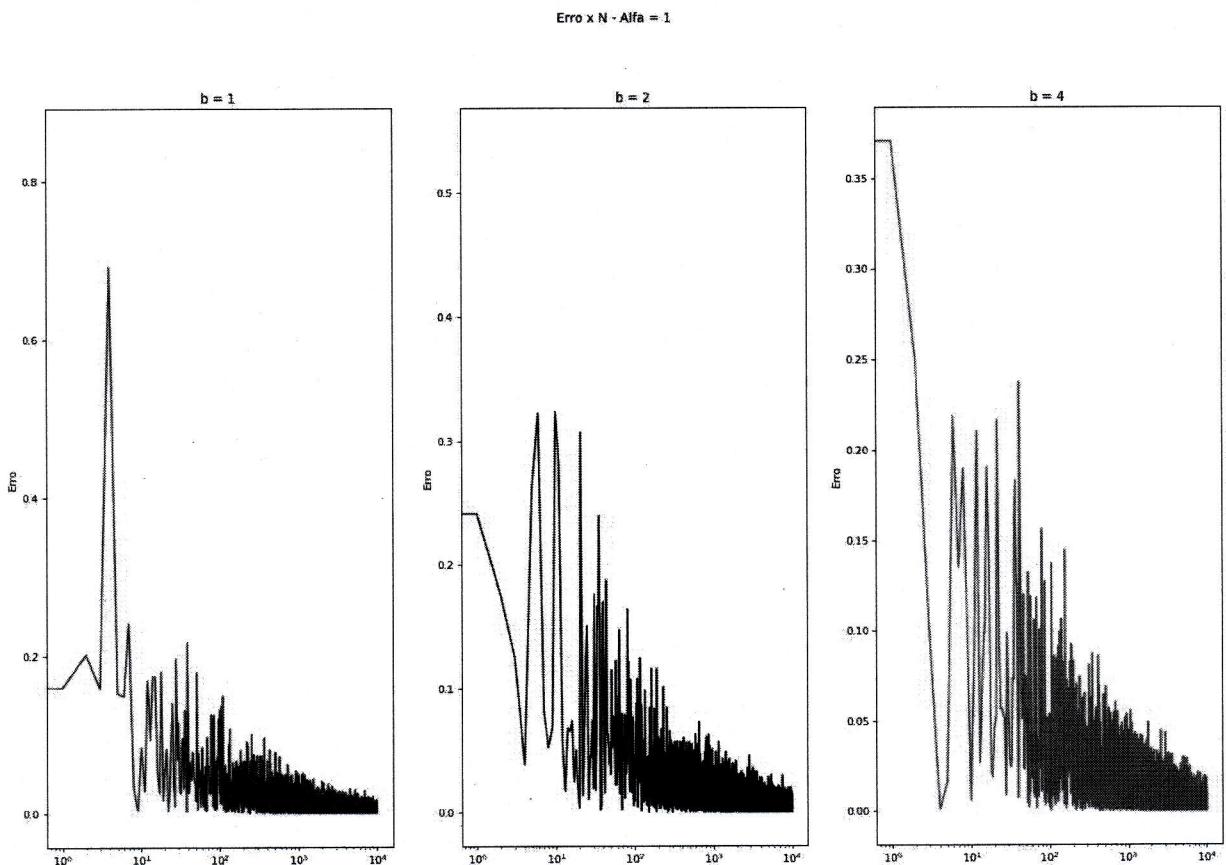
    # Colocando titulo do grafico
    fig.suptitle('Erro x N - Alfa = ' + str(i+1) )

plt.show()
plt.close()
return

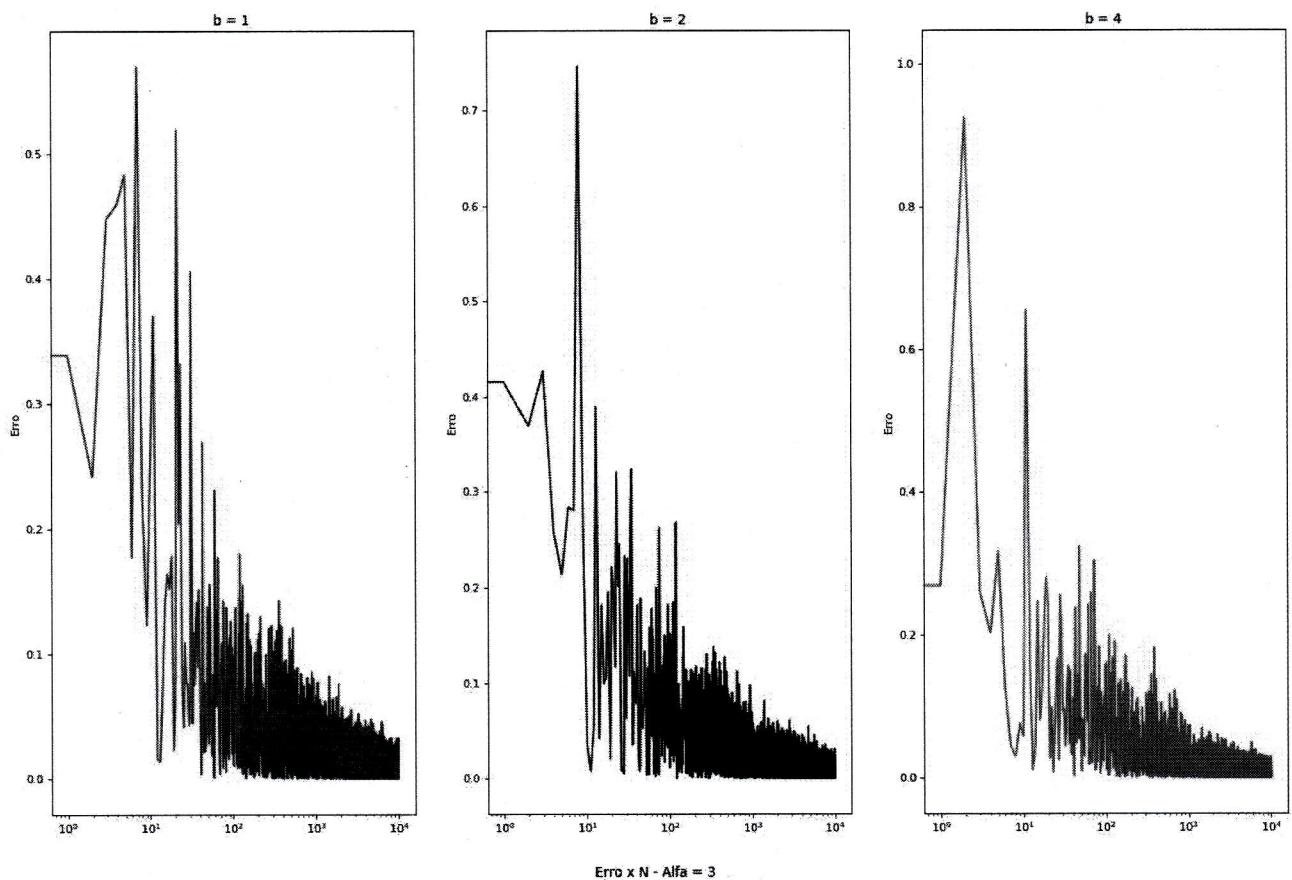
```

#### Questão 6 – Item 4 – Gráfico v1

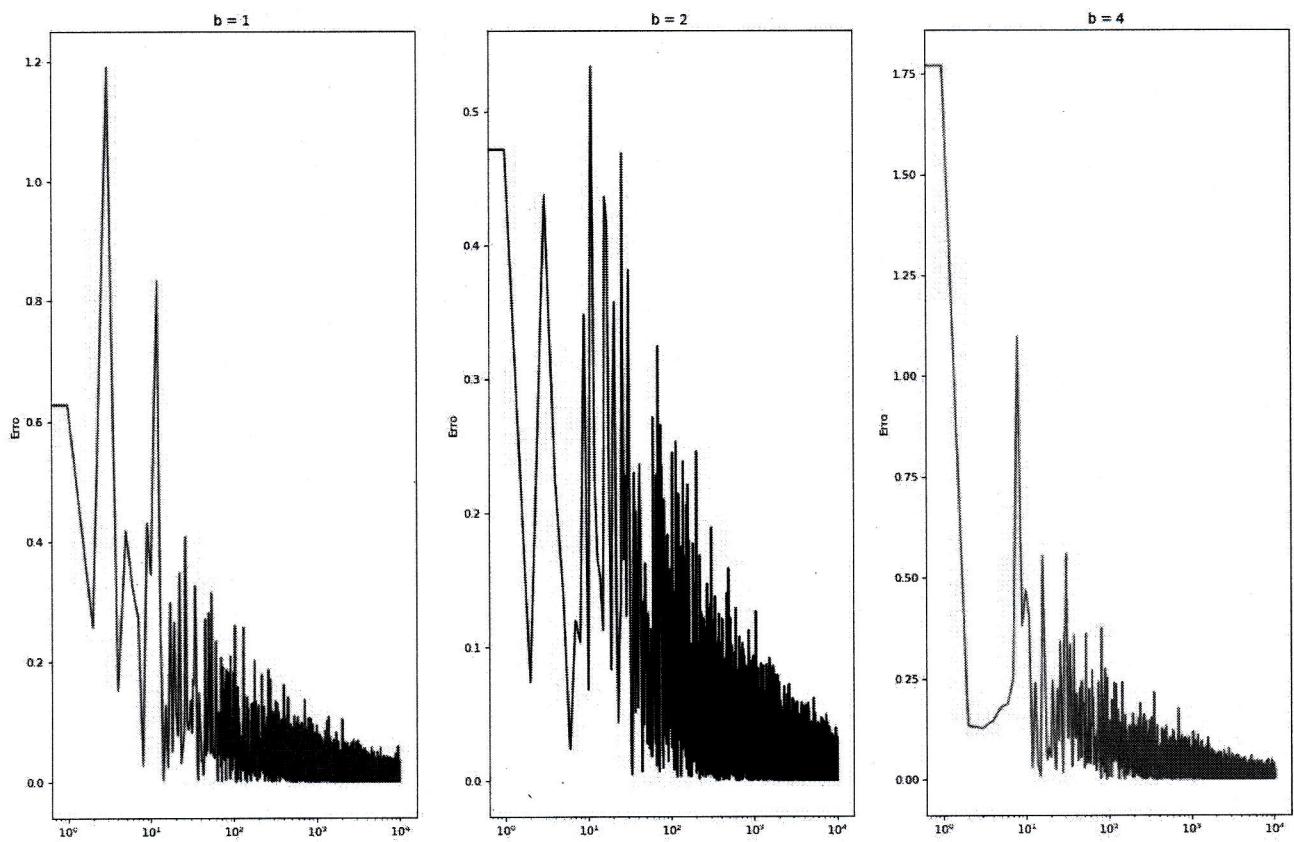
Seja  $\text{erro} = (\text{estimador}(n) - g(\alpha, a, b)) / g(\alpha, a, b)$  para  $n = [1, 10^4]$  em escala log no eixo x.



Erro x N - Alfa = 2



Erro x N - Alfa = 3



### Questão 6 – Item 3 – Algoritmo v2

```
# Algoritmo completo via GitHub: https://github.com/cadupsg/mcmc/blob/master/questao6v2.py

def gera_rand(valor_min, valor_max):
    #gera valor aleatorio a partir de uma distribuicao uniforme entre os valores a e b (limites de integracao)
    limite = valor_max - valor_min
    num = random.uniform(0,1)
    return valor_min + limite*num

def f_de_x(x, alfa):
    # retorna a funcao que sera integrada
    return (x**alfa)

def monte_carlo(n_amostras, alfa, a, b):
    # calcula a media amostral e a utiliza para estimar o valor de g(alfa,a,b)
    soma_amostras = 0
    for i in range(n_amostras):
        x = gera_rand(a, b)
        soma_amostras += f_de_x(x, alfa)

    return (b - a) * float(soma_amostras/n_amostras)

def integral_analitica(alfa, a , b):
    alfa = float(alfa)
    a = float(a)
    b = float(b)
    return (((b***(alfa+1))/(alfa+1))-((a***(alfa+1))/(alfa+1)))

def mostra_grafico(n):
    # definicao de variaveis
    estim_g1 = [[0 for i in range(n)] for ialfa in range(3)]
    erro1 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g2 = [[0 for i in range(n)] for ialfa in range(3)]
    erro2 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g3 = [[0 for i in range(n)] for ialfa in range(3)]
    erro3 = [[0 for i in range(n)] for ialfa in range(3)]

    # calcula os erros para cada alfa e b
    for alfa in xrange(1,4):
        g_analitica1 = integral_analitica(alfa, 0, 1)
        g_analitica2 = integral_analitica(alfa, 0, 2)
        g_analitica3 = integral_analitica(alfa, 0, 4)
        for i in xrange(1, n+1):
            estim_g1[alfa-1][i-1] = monte_carlo(i, alfa, 0, 1)
            erro1[alfa-1][i-1] = (abs(estim_g1[alfa-1][i-1]-g_analitica1))/g_analitica1
            estim_g2[alfa-1][i-1] = monte_carlo(i, alfa, 0, 2)
            erro2[alfa-1][i-1] = (abs(estim_g2[alfa-1][i-1]-g_analitica2))/g_analitica2
            estim_g3[alfa-1][i-1] = monte_carlo(i, alfa, 0, 4)
            erro3[alfa-1][i-1] = (abs(estim_g3[alfa-1][i-1]-g_analitica3))/g_analitica3

    # parte que cria os graficos
    cor = ['red', 'blue', 'magenta', 'k']
    eixoX = np.linspace(0,n,n)
```

```

for i in xrange(0,3):
    # cria figura com 3 graficos
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 20), dpi=80)

    axes[0].plot(eixoX, erro1[i], ls = '-', lw = '1.5', c = cor[0])
    axes[0].set_ylabel('Erro', color='k')
    axes[0].set_title('b = 1', color='k')

    axes[1].plot(eixoX, erro2[i], ls = '-', lw = '1.5', c = cor[1])
    axes[1].set_ylabel('Erro', color='k')
    axes[1].set_title('b = 2', color='k')

    axes[2].plot(eixoX, erro3[i], ls = '-', lw = '1.5', c = cor[2])
    axes[2].set_ylabel('Erro', color='k')
    axes[2].set_title('b = 4', color='k')

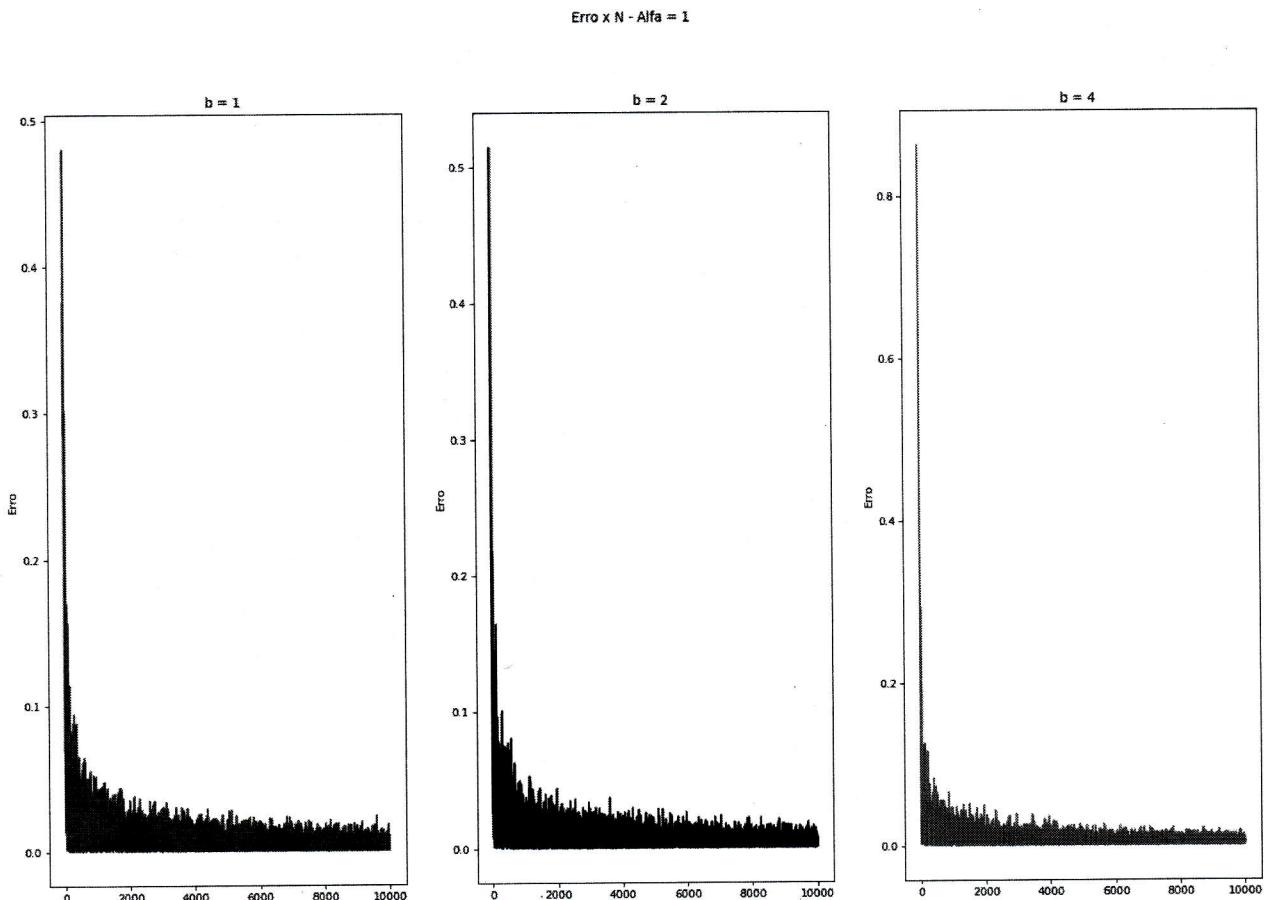
    # Colocando titulo do grafico
    fig.suptitle('Erro x N - Alfa = ' + str(i+1) )

plt.show()
plt.close()
return

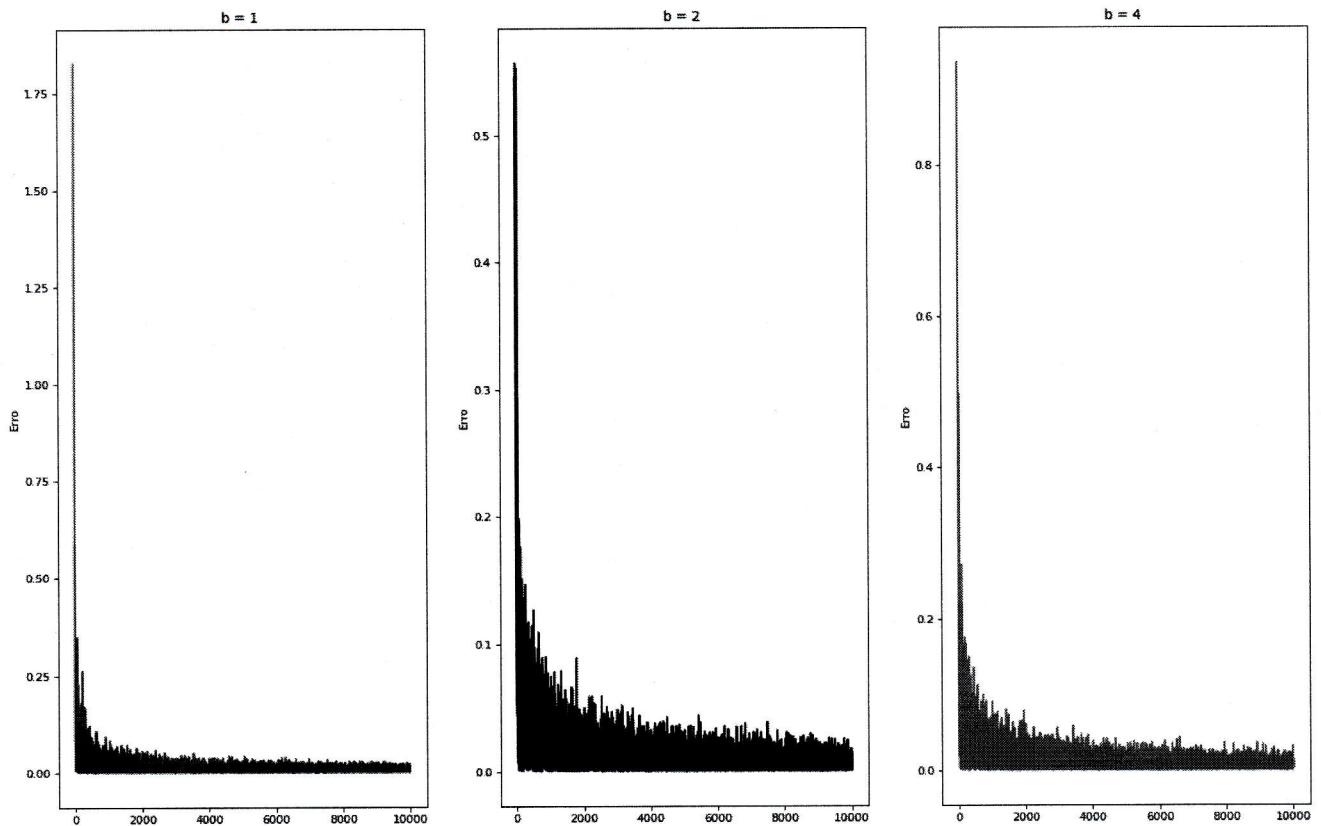
```

### Questão 6 – Item 4 – Gráfico v2

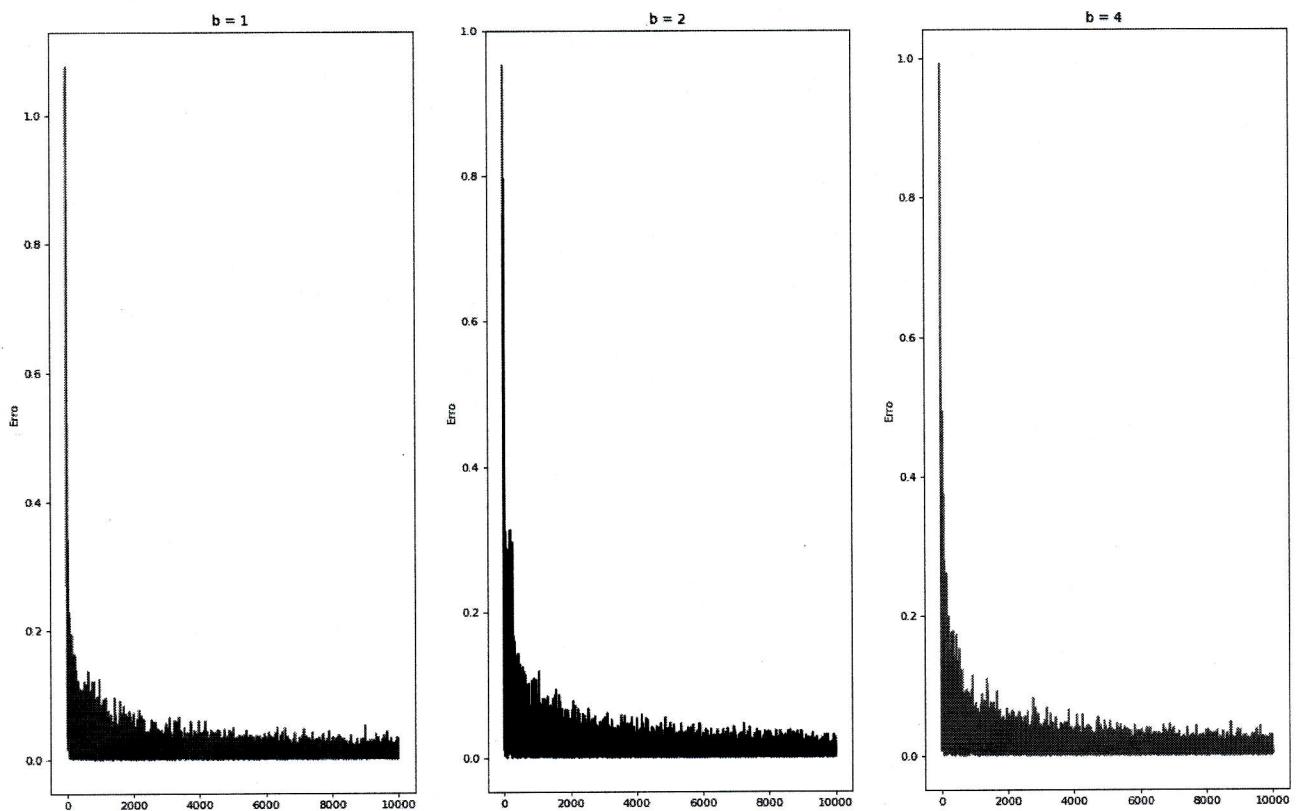
Seja  $\text{erro} = (\text{estimador}(n) - g(\alpha, a, b)) / g(\alpha, a, b)$  para  $n = [1, 10^4]$  sem escala log no eixo x (eixo x = n).



Erro x N - Alfa = 2



Erro x N - Alfa = 3



### Questão 6 – Item 3 – Algoritmo v3

```
# Algoritmo completo via GitHub: https://github.com/cadupsg/mcmc/blob/master/questao6v3.py

def gera_rand(valor_min, valor_max):
    #gera valor aleatorio a partir de uma distribuicao uniforme entre os valores a e b (limites de integracao)
    limite = valor_max - valor_min
    num = random.uniform(0,1)
    return valor_min + limite*num

def f_de_x(x, alfa):
    # retorna a funcao que sera integrada
    return (x**alfa)

def monte_carlo(n_amostras, alfa, a, b):
    # calcula a media amostral e a utiliza para estimar o valor de g(alfa,a,b)
    soma_amostras = 0
    for i in range(n_amostras):
        x = gera_rand(a, b)
        soma_amostras += f_de_x(x, alfa)

    return (b - a) * float(soma_amostras/n_amostras)

def integral_analitica(alfa, a , b):
    alfa = float(alfa)
    a = float(a)
    b = float(b)
    return (((b***(alfa+1))/(alfa+1))-((a***(alfa+1))/(alfa+1)))

def mostra_grafico(n, a, b):
    # definicao de variaveis
    estim_g1 = [[0 for i in range(n)] for ialfa in range(3)]
    erro1 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g2 = [[0 for i in range(n)] for ialfa in range(3)]
    erro2 = [[0 for i in range(n)] for ialfa in range(3)]
    estim_g3 = [[0 for i in range(n)] for ialfa in range(3)]
    erro3 = [[0 for i in range(n)] for ialfa in range(3)]

    # calcula os erros para cada alfa e b
    for alfa in xrange(1,4):
        g_analitica1 = integral_analitica(alfa, 0, 1)
        g_analitica2 = integral_analitica(alfa, 0, 2)
        g_analitica3 = integral_analitica(alfa, 0, 4)
        soma_amostras = 0
        for i in xrange(1, n+1):
            x = gera_rand(a, b)
            soma_amostras += f_de_x(x, alfa)
            erro1[alfa-1][i-1] = (abs((b - a) * float(soma_amostras/n)-g_analitica1))/g_analitica1
            erro2[alfa-1][i-1] = (abs((b - a) * float(soma_amostras/n)-g_analitica2))/g_analitica2
            erro3[alfa-1][i-1] = (abs((b - a) * float(soma_amostras/n)-g_analitica3))/g_analitica3

    # parte que cria os graficos
    cor = ['red', 'blue', 'magenta', 'k']
    eixoX = np.linspace(0,n,n)
```

```

for i in xrange(0,3):
    # cria figura com 3 graficos
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(20, 20), dpi=80)

    axes[0].plot(eixoX, erro1[i], ls = '-.', lw = '1.5', c = cor[0])
    axes[0].set_ylabel('Erro', color='k')
    axes[0].set_title('b = 1', color='k')
    axes[0].set_xscale("log")

    axes[1].plot(eixoX, erro2[i], ls = '-.', lw = '1.5', c = cor[1])
    axes[1].set_ylabel('Erro', color='k')
    axes[1].set_title('b = 2', color='k')
    axes[1].set_xscale("log")

    axes[2].plot(eixoX, erro3[i], ls = '-.', lw = '1.5', c = cor[2])
    axes[2].set_ylabel('Erro', color='k')
    axes[2].set_title('b = 4', color='k')
    axes[2].set_xscale("log")

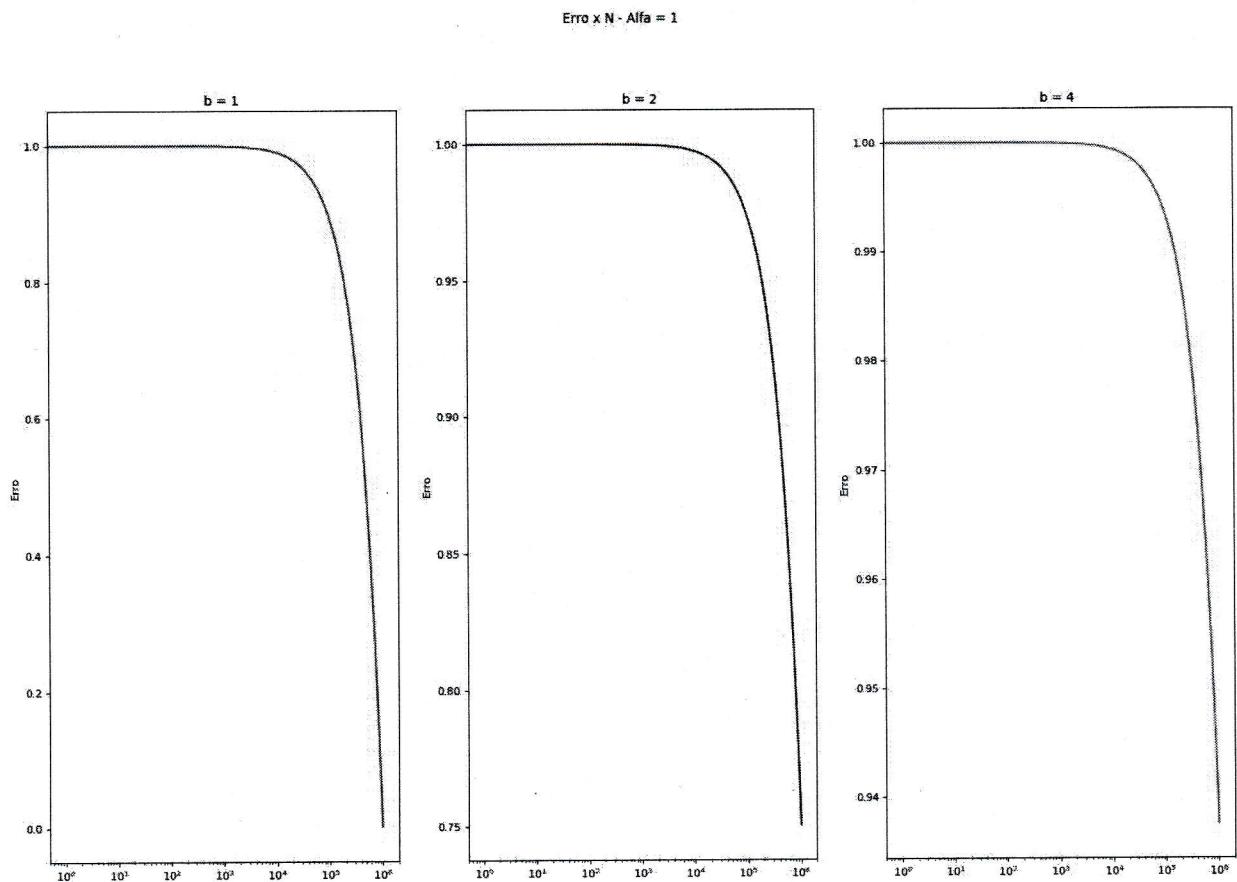
    # Colocando titulo do grafico
    fig.suptitle('Erro x N - Alfa = ' + str(i+1) )

plt.show()
plt.close()
return

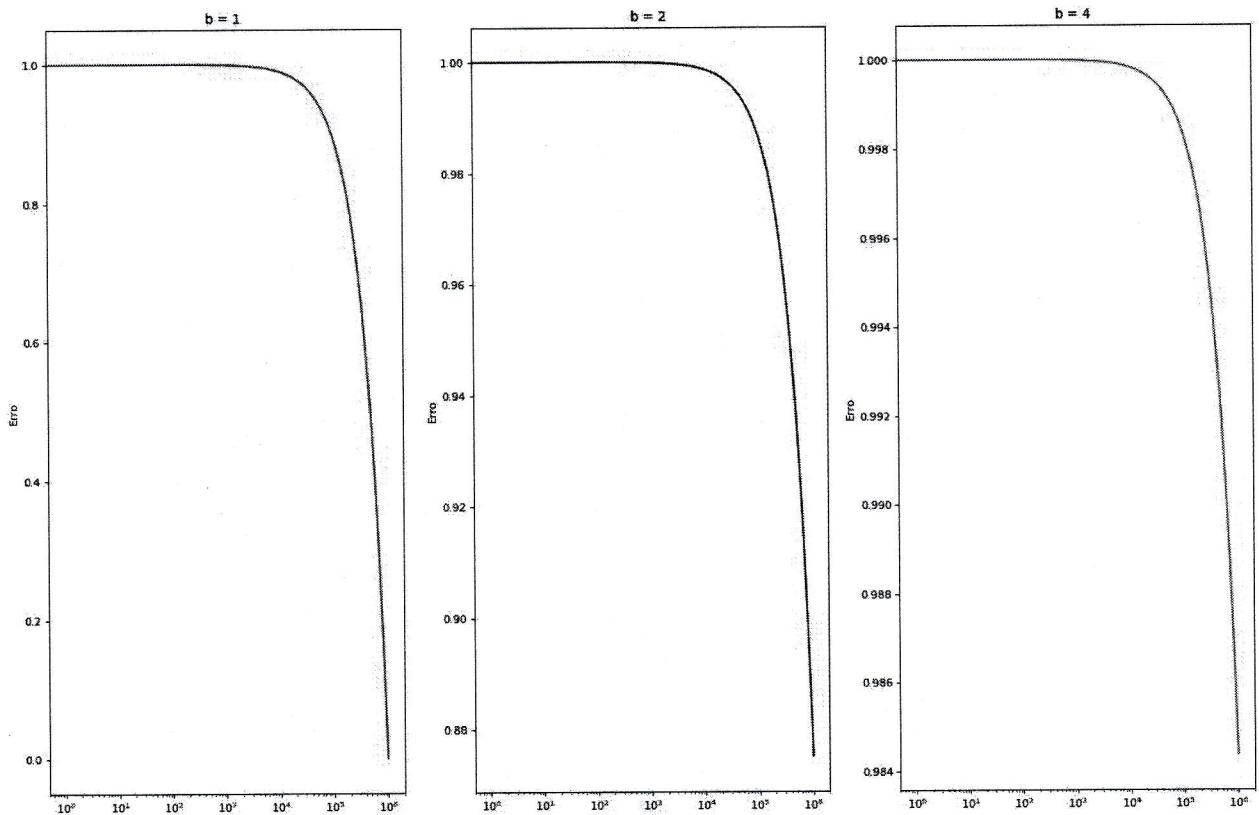
```

### Questão 6 – Item 4 – Gráfico v3

Seja  $\text{erro} = (\text{estimador}(n) - g(\alpha, a, b)) / g(\alpha, a, b)$  para  $n = [1, 10^6]$  em escala log no eixo x.



Erro x N - Alfa = 2



Erro x N - Alfa = 3

